Sifting Truth from Coincidences: A Two-Stage Positive and Unlabeled Learning Model for Coincidental Correctness Detection

Chunyan Liu[1], Huan Xie[1], Yan Lei[1]*, Zhenyu Wu[1], Jinping Wang[1]
[1]School of Big Data and Software Engineering, Chongqing University, Chongqing, China
Email:{chunyanliu, huanxie, yanlei}@cqu.edu.cn, {zhenyu_wu, jpwang}@stu.cqu.edu.cn

***Abstract*— Fault localization (FL) can identify the fault's location by analyzing the execution information from test cases in the program. This execution information serves as the foundation for FL to infer latent causal relationships between fault entities and failed results. However, this execution information contains coincidental correctness (CC), which reduces the accuracy of FL. CC arises when a test case executes faulty program entities but still produces the correct output, leading to misleading FL inferences. In widely used datasets, the presence of CC compromises the reliability of passed test cases (i.e., negative labels). In contrast, failed test cases (i.e., positive labels) remain definitive. In FL scenarios, unlabeled data is typically abundant and primarily consists of passed test cases. Therefore, systematically leveraging positive and unlabeled data for accurate CC detection is essential, which is beneficial to FL. To tackle the problem, we propose a two-stagE positiVe and unlAbeled learning model for coiNcidental correctneSs detection, EVANS. *EVANS* defines failed test cases as positive samples and treats the remaining ones as unlabeled data. It comprises two core modules: (1) A module for selecting high-quality pseudo-negative samples. This module leverages vector distance metrics to identify high-quality pseudo-negative test cases, using inter-class distances computed via a pre-trained model. (2) A weakly supervised contrastive learning module. This module utilizes the labeled samples from Stage (1) to train a contrastive learning model, which then detects CC in unlabeled test cases. Experimental results demonstrate that EVANS significantly outperforms current CC detection methods.**

***Index Terms*—Coincidental correctness detection, positive and unlabeled learning, contrastive learning**

I. Introduction

Fault localization (FL) is an essential debugging technique in the software development process. By prioritizing program entities (e.g., statement, function, or class) with higher fault probabilities, FL reduces developers' debugging effort. A survey by Kochhar et al. [1] found that developers find FL most useful when it ranks faults among the top 5. This reliance on FL's ranking places higher demands on its precision. However, FL's precision depends on the accuracy of its suspiciousness scores, which are computed using execution information of test cases. In real world, these executions information can be noisy, weakening FL's reliability.

A major source of noise is coincidental correctness (CC) [2], a widely observed phenomenon in software testing datasets. CC occurs when a test case executes faulty entities but still produces an expected outcome. Since FL relies on test execution data to infer suspiciousness scores, CC introduces misleading patterns that distort fault ranking. Specifically, CC test cases create an illusion of correctness, causing FL to rank faulty entities lower than they should be. It's similar to a malfunctioning traffic signal that directs developers down the wrong path. To systematically analyze the impact of CC, we categorize test cases into three groups based on their execution traces and outcomes (as illustrated in Fig. 1): **(1) Failed Test Cases:** Execute faulty code entities and result in failures; **(2) CC Test Cases:** Execute faulty code entities but produce expected outcomes; **(3) Reliable Passed Test Cases:** Passed test cases excluding CC (i.e., the actual passed ones). Our

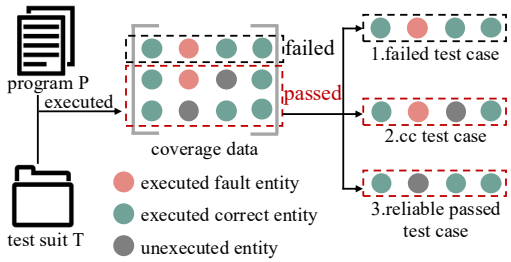

Fig. 1: The classification of test cases.

statistical analysis on the Defects4J dataset [3] confirms CC's prevalence. As shown in Table I, reliable passed (RP) tests significantly outnumber CC test cases, accounting for 68% to 96% of all passed test cases [4]. This distribution reveals a key challenge: FL assumes most passed test cases are reliable, but CC introduces deceptive signals that disrupt fault ranking.

TABLE I: The distribution of failed test cases, CC, and reliable passed test cases in Defects4J

Program	Tests	Failed Test Cases	CC	Reliable Passed Test Cases	$\frac{\|RP\|}{\|CC+RP\|}$
Chart	3,565	92	484	2,989	86.06%
Closure	241,642	4,457	47,441	189,744	80.00%
Lang	4,589	124	737	3,728	83.45%
Math	12,089	176	1,830	10,083	84.63%
Mockito	20,200	120	6,371	13,709	68.26%
Time	50,933	76	1,661	49,196	96.73%
Total	**333,018**	**5,045**	**58,524**	**269,449**	**82.16%**

Most existing FL methods [5]–[12], whether traditional or deep learning-based, rely on correlations between execution

*Yan Lei is corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00127

information (e.g., coverage data) and test case outcomes. CC distorts this fundamental relationship, reducing FL's ability to correctly rank faulty entities. Thus, detecting and filtering CC is essential for improving FL's accuracy. In the work of Xie et al. [13], CC detection methods are categorized into two major groups: heuristic algorithms [14], [15] and machine learning (ML) algorithms [4], [16]–[19]. The ML-based methods are further divided into supervised and unsupervised approaches. However, our analysis reveals key limitations in both categories. Heuristic algorithms [14] and unsupervised ML methods [17] suffer from a *local coverage limitation*: they rely on test case coverage patterns but fail to capture global semantic contexts. Supervised ML algorithms [16], [19] face the *label noise issue*: since "passed test cases" inherently contain CC cases, models trained on these labels become unreliable. Given this, a more robust approach is needed.

To address these limitations, we adopt Positive and Unlabeled (PU) learning, which avoids explicit reliance on potentially incorrect negative labels [20], [21]. PU learning treats all passed test cases as unlabeled data and failed test cases as positive samples. This setting aligns well with the CC detection task, where CC cases are hidden within passed tests. It avoids the interference of label noise on CC detection. Moreover, PU learning leverages global semantic structure across all test cases. It contrasts failed and passed behaviors, enabling more reliable CC identification without handcrafted rules or noisy supervision.

Building on this, we propose ***EVANS***, a two-stage PU learning framework for CC detection. ***EVANS*** operates in two stages: (1) **High-Confidence Pseudo-Negative Sample Selection:** It leverages pre-trained embeddings to measure distances between failed test cases and passed test cases, selecting the most reliable passed test cases as high-confidence pseudo-negative (HN) samples. (2) **Weakly-Supervised Contrastive Learning:** It refines CC detection by using contrastive learning to distinguish CC from reliable passed test cases in unlabeled data.

Experimental results on Defects4J show that ***EVANS*** outperforms eight state-of-the-art CC detection methods in F1-score. When integrated into FL, ***EVANS*** significantly improves fault ranking metrics, including MAR, MFR, and Top-K accuracy.

In summary, the primary contributions of this paper are as follows:

- We propose ***EVANS***, a two-stage CC detection model based on PU learning. It selects HN samples using inter-class distance and then distinguishes CC cases through weakly-supervised contrastive learning.
- This is the first work to integrate pseudo-negative selection and contrastive learning into a unified PU learning framework. The design enables effective semantic separation between CC and reliably passed test cases without relying on explicit negative labels.
- We comprehensively evaluate ***EVANS*** on the Defects4J benchmark and demonstrate its superior performance in both CC detection and FL tasks, outperforming eight state-of-the-art baselines across multiple datasets.
- We release our implementation [22] to support further research in CC detection and FL.

The remainder of this paper is organized as follows. Section II introduces related work. Section III presents the methodology framework. Section IV and Section V describe the experimental setup, including datasets, baselines, and results. Section VI discusses the rationale behind the effectiveness of ***EVANS***. Section VII concludes the paper.

II. RELATED WORK

A. Positive and Unlabeled learning

As surveyed by Bekker and Davis [20], PU learning is a weakly supervised classification approach that trains a classifier to distinguish positive and negative samples using only labeled positive and unlabeled samples [21], [23], [24]. Any labeled sample is considered positive, while unlabeled samples have unknown labels. To ensure reliability, only highly confident positive samples are labeled.

1) PU Learning Paradigms: PU learning can be categorized based on the data sources of the positive set (P) and the unlabeled set (U) [20]:

- **Single Training Set:** Both P and U originate from the same dataset.
- **Case-Control:** P and U come from different datasets.

This study focuses on the single training set scenario, where all samples follow a shared distribution: $x \sim f(x), x \sim \alpha f_+(x) + (1-\alpha) f_-(x), x \sim \alpha c f_l(x) + (1-\alpha c) f_u(x)$, α denotes the proportion of positive samples, and c represents the fraction of positive samples selected for labeling. Thus, the dataset contains αc labeled samples. Since positive samples are sparse in the FL dataset, we set $c = 1$ to maximize their utilization. The terms $f_+(x)$, $f_-(x)$, $f_l(x)$, and $f_u(x)$ represent the probability density functions of positive, negative, labeled, and unlabeled samples, respectively.

2) PU Learning Assumptions: PU learning relies on two key assumptions: label assumptions and data assumptions. For labeling assumption, ***EVANS*** modifies the standard PU setting for CC detection. It labels all failed test cases as positives since they are rare in the FL dataset. For data assumptions, ***EVANS*** follows two fundamental assumptions: separability [20], [25], [26] and smoothness [24], [26]–[28].

Separability Assumption. A classifier exists that can distinguish positive and negative samples. A decision threshold θ is used for classification. Samples with scores above θ are classified as positive, while those below θ are classified as negative.

Smoothness Assumption. Samples with similar representations should have similar predicted probabilities. Formally, if two instances x_1 and x_2 have similar feature representations, their probability of being positive should also be similar:

$$\Pr(y = 1|x_1) \approx \Pr(y = 1|x_2). \quad (1)$$

This assumption allows negative samples to be identified by measuring their distance from positive samples. It also provides the foundation for contrastive learning in ***EVANS***, which enhances sample differentiation.

3) Two-Stage PU Learning Framework: PU learning is typically performed in two stages:

1) Identify high-confidence pseudo-negative (HN) samples from the positive set (P) and the unlabeled set (U).
2) Train a binary classifier using P and HN samples.

EVANS follows this framework. It leverages inter-class distance calculation and contrastive learning to improve HN sample selection and classification.

B. Fault Localization

Faults are the root cause of program errors, often manifesting as incorrect steps or processes [29]. FL identifies faults by analyzing execution information. It has evolved from spectrum-based FL (SFL) to deep learning-based FL (DLFL), with Large Language Models (LLMs) emerging as a powerful alternative.

SFL remains popular due to its efficiency and lightweight. Methods like Dstar [7], Ochiai [5], and Tarantula [30] assign suspiciousness scores based on test coverage, analyzing how program entities appear in passed and failed test cases [6], [7]. However, they struggle with complex fault patterns that require deeper semantic understanding.

DLFL enhances FL accuracy by leveraging neural networks. Early work [31] applied Multi-Layer Perceptron (MLP) to model execution data and test results. Later methods introduced convolutional [8], [32], graph-based [9], [33]–[35], and recurrent [36]–[40] neural networks. Many approaches further integrate auxiliary information, such as program changes [41], control flow [9], and fault context modeling [42], among others [34], [35], [43].

LLMs improve FL by incorporating extensive knowledge and reasoning. AGENTFL [44] mimics developer behavior through a three-stage debugging process, integrating behavioral tracking and document-guided search to overcome LLMs' context length limitations. AutoFL [45] enhances fault repair and interpretability using function call-based navigation. FuseFL [46] combines statistical FL, label vectors, and code descriptions, leveraging stepwise reasoning and diverse evaluation metrics to improve interpretability.

C. Coincidental Correctness

Budd et al. [47] first introduced CC to describe test cases that execute faulty entities but still produce expected outputs. Masri et al. [2] later demonstrated that CC test cases can degrade the accuracy and effectiveness of FL. The key challenge in CC detection is distinguishing CC test cases from reliable passed test cases. To address this, researchers have proposed various techniques.

Heuristic-based approaches assume that test cases covering statements executed only in failed test cases are more likely to be CC test cases. However, these methods may introduce false positives due to their reliance on execution similarity heuristics [14]. Masri et al. [14] proposed cleansing techniques that systematically identify and remove CC tests from passing suites to enhance the safety and precision of FL.

Clustering methods detect CC test cases by measuring test case similarity. Beyond basic clustering techniques, researchers have incorporated execution information such as basic blocks and control flow edges [4], reconstructed coverage matrices [48], and weighted clustering techniques [49] to improve detection accuracy. More recently, Yu et al. [17] proposed a context-based clustering approach that utilizes failure-inducing contexts to enhance CC identification. Bandyopadhyay et al. [15] introduced weighting and iterative prediction strategies that mitigate the negative impact of CC on FL by assigning lower weights to likely CC tests or iteratively removing them.

Classifier-based approaches have also proven effective for CC detection. Dass et al. [19] proposed a Random Forest classifier, which constructs multiple decision trees and applies majority voting for CC identification. Their method leverages execution paths and coverage matrices as features. Later, Dass refined this approach by optimizing feature extraction and data preprocessing, further improving classification accuracy [16]. Beyond traditional Random Forest methods, neural networks have demonstrated strong performance in CC detection, with studies utilizing MLP [50], triplet neural networks [13], and other deep learning models [18].

Moreover, Fuzzy theory has also gained attention in CC detection [51], [52]. These approaches address the uncertainty in test case execution data, refining CC identification precision by incorporating probabilistic functions and fuzzy similarity measures. Building upon this, Liu et al. [53] proposed a weighted fuzzy classification framework using fuzzy weighted K-Nearest Neighbor algorithm and fuzzy manipulation strategies, which significantly improves both the accuracy and efficiency of CC test case identification.

III. METHODOLOGY

A. Overview

The architecture of ***EVANS*** is illustrated in Fig. 2. ***EVANS*** first leverages a MLP to pre-train by self-supervision, allowing it to capture significant features from coverage information. Once pre-training is completed, the hidden and output layers are fine-tuned to adapt the model specifically for CC detection. Based on the refined representations, ***EVANS*** detects CC through two core modules: (1) High-Confidence Pseudo-Negative Sample Selection Module. (2) Weakly-Supervised Contrastive Learning Module. In the following sections, we will provide details on each component of ***EVANS***.

B. Problem Formulation

In CC detection, our goal is to distinguish CC tests (false negative samples) from reliable passed tests (true negative samples) within the set of unlabeled passed tests (unlabeled samples). This is achieved through PU learning, where only failed tests (positive samples) are labeled, while the label of passed tests remains unknown. The objective is to learn representations that enable reliable separation of CC from RP.

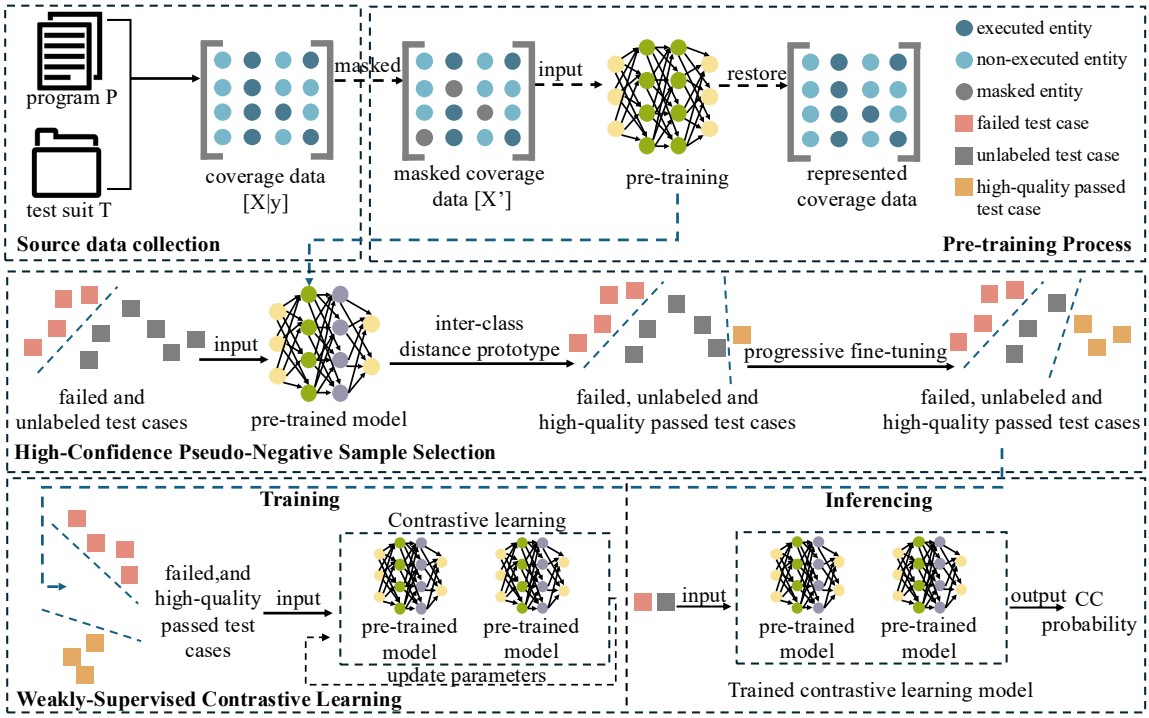

Fig. 2: The architecture of ***EVANS***

Dataset. Let D denote the dataset, containing both labeled and unlabeled samples:

$$D = \{(x_i, y_i, l_i) \mid x_i \in X, y_i \in Y, l_i \in C, i \in \{1, 2, ..., n\}\} \tag{2}$$

Each x_i is the coverage vector of a test case that corresponds to one sample. The label set $Y = \{0, 1\}$ indicates failed ($y_i = 1$) or passed ($y_i = 0$), $C = \{0, 1\}$ indicates whether a sample is labeled:

$$l_i = 1 \Rightarrow y_i = 1, \quad l_i = 0 \Rightarrow y_i \in \{0, 1\}. \tag{3}$$

Labeled samples ($l_i = 1$) correspond to failed tests and are treated as positive (X_p), while unlabeled samples ($l_i = 0$) correspond to passed tests, which could be CC ($y_i = 1$) or RP ($y_i = 0$) and form the set X_u. For clarity, we use *samples* and *coverage vectors* interchangeably. Since the true labels of X_u are unknown, direct supervised classification is infeasible.

Pre-training. To enable ***EVANS*** to effectively represent coverage data of tests, we first pre-train it via self-supervised learning. Specifically, given the original coverage data X, we generate a masked version X' by applying a masking function M:

$$X' = M(X) \tag{4}$$

EVANS is then trained to restore the original data X from X':

$$G : X' \to X \tag{5}$$

The MLP encoder $G(X)$ is trained with a reconstruction loss $\mathcal{L}_{\text{recon}}$. $\mathcal{L}_{\text{recon}}$ is a standard Mean Squared Error (MSE), which encourages learning meaningful features by minimizing the difference between the original input and its reconstruction.

$$\mathcal{L}_{\text{recon}} = \frac{1}{n} \sum_{i=1}^{n} \|G(X_i') - X_i\|^2 \tag{6}$$

This stage encourages ***EVANS*** to capture structural patterns of coverage data.

HN Selection. Within the hidden layers, ***EVANS*** calculates the cosine similarity between positive and unlabeled samples and then selects HN samples based on this similarity. Cosine similarity captures directional closeness in high-dimensional semantic space more robustly than raw coverage distances, ensuring more accurate HN selection. For each unlabeled sample $x_i \in X_u$, we compute the cosine similarity $s(x_i, x_j)$ between x_i and all positive samples $x_j \in X_p$ as follows:

$$s(x_i, x_j) = \frac{f(x_i) \cdot f(x_j)}{\|f(x_i)\| \|f(x_j)\|} \tag{7}$$

where $f(x_i)$ and $f(x_j)$ represent the feature embeddings of the x_i and the x_j, respectively, and $\|f(x_i)\|$ and $\|f(x_j)\|$ are the Euclidean norms of these feature vectors. Next, we select HN samples by identifying the unlabeled samples that are most distant from all positive samples. The HN sample x^- is selected as follows:

$$x^- = \arg \max_{x_i \in X_u} \min_{x_j \in X_p} s(f(x_i), f(x_j)) \tag{8}$$

These samples serve as reliable negative samples in subsequent contrastive learning. To train ***EVANS*** for effective HN selection, we minimize the similarity between unlabeled and positive samples:

$$\mathcal{L}_{\text{pseudo-neg}} = \frac{1}{|X_u|} \sum_{x_i \in X_u} \min_{x_j \in X_p} s(f(x_i), f(x_j)) \tag{9}$$

This self-supervised contrastive loss encourages HN samples to be selected from the most dissimilar unlabeled samples, ensuring that ***EVANS*** can reliably separate CC from RP in subsequent stages.

Contrastive learning. With positive and HN samples, ***EVANS*** learns to separate CC from RP tests by maximizing intra-class similarity and minimizing inter-class similarity. Specifically, for an unlabeled sample representation $f(x)$, the goal is to ensure that its similarity to a positive sample is greater than its similarity to a HN sample:

$$s(f(x), f(x^+)) \gg s(f(x), f(x^-)) \tag{10}$$

where x^+ is a positive sample (failed test case). x^- is a HN sample. $s(f(x), f(x^+))$ measures similarity between representations. The model minimizes the following contrastive loss:

$$\mathcal{L}_{\text{con-loss}} = -\sum_{(x_i,x_j)\in P} \log \frac{\exp(s(f(x_i), f(x_j)))}{\sum_{(x_k,x_m)\in N} \exp(s(f(x_k), f(x_m)))} \tag{11}$$

where P and N denote sets of similar and dissimilar pairs, respectively. This loss is a commonly used pairwise contrastive loss that allows ***EVANS*** to learn class-aware representations and reliably identify unlabeled tests likely to be CC.

C. Pre-Training Module

The pre-training of ***EVANS*** initializes the MLP model to effectively represent test coverage data. Following the "Single Training Set" assumption [26], it relies on the same coverage data as the subsequent PU learning phase. In this section, we provide a step-by-step analysis of the pre-training process.

(1) Masked Coverage Data Generation. This process aims to construct self-supervised training samples by selectively masking parts of the original coverage data in a controlled manner. Let the coverage data be denoted as $X \in R^{n\times d}$, where n represents the number of test cases and d is the coverage dimension, such as the number of program entities. The objective is to generate a masked version X' from X by applying a masking function M, as formulated in Eq. (4).

To achieve this, We set $\alpha = 0.05$, as lower ratios are better suited for capturing simple coverage patterns and ensuring stable reconstruction performance. For each sample $X_i \in X$, $\lfloor \alpha \times d \rfloor$ dimensions are randomly selected to be masked, resulting in the masked data X_i'. The masking function is formally defined in Eq. (12), where masked dimensions are replaced with a fixed value of -1:

$$X_i'[m] = \begin{cases} -1 & \text{if the dimension } m \text{ is masked} \\ X_i[m] & \text{otherwise} \end{cases} \tag{12}$$

Afterward, input-output pairs (X_i', X_i) are generated, where the masked data X_i' serves as the model input, and the original data X_i is the restore target. This design ensures that the model effectively learns the key features of the coverage data while maintaining adaptability to various scenarios.

(2) Restore From Masked Coverage Data. The goal of this process is to enable the model to learn feature representations of the coverage data by restoring original inputs.

As shown in Fig. 3, the MLP adopts an encoder-decoder structure, consisting of an input layer, seven hidden layers, and an output layer. It is designed to learn nonlinear mappings and extract high-level representations from corrupted inputs, effectively capturing meaningful data distributions. To enhance

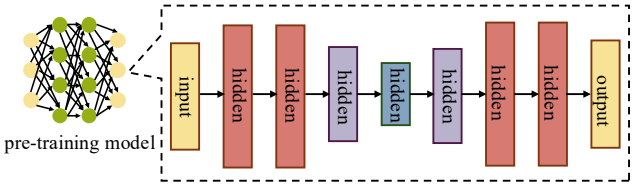

Fig. 3: The structure of trained model

feature extraction and restoration, the first and second hidden layers contain the same number of neurons, which is $\frac{4}{3}$ times the number in the input layer. Afterward, the number of neurons gradually decreases until reaching a minimum in the fourth hidden layer. By progressively reducing and then increasing the neuron count, this symmetric structure enforces a low-dimensional feature representation. This allows the model to extract essential coverage patterns while discarding redundant information.

Specifically, the input layer takes the masked coverage data X' and maps it to a high-dimensional feature space. The subsequent hidden layers extract hierarchical features using the ReLU activation function.

Finally, the output layer restores the original coverage data as in Eq. (5). The model is trained with the MSE loss in Eq. (6), which directly minimizes the reconstruction loss between original and restored coverage data. This objective guides the encoder to learn meaningful representations from masked inputs through backpropagation.

To enable the effective selection of HN samples and CC detection, we fine-tune the pre-trained MLP, and the refined structure is shown in Fig. 4. First, one hidden layer is chosen as the distance calculation layer. The identification of this layer is determined through ablation studies (in section V-A), ensuring that it provides optimal feature representation for distinguishing positive and unlabeled samples. Second, we refine the output layer. The original MLP was designed for restoration tasks, taking masked and restored coverage data as input and output. We fine-tuned it to adapt to the downstream contrastive learning task to produce a probability score in the range [0,1], representing the probability that a test case belongs to CC. These lightweight adjustments adapt the model to downstream tasks without affecting the semantic representations learned during pre-training.

D. High-Confidence Pseudo-Negative Sample Selection Module

In this section, we introduce the High-Confidence Pseudo-Negative Sample Selection Module, which aims to select HN from unlabeled samples. Given the challenge of distinguishing CC from RP, this module leverages distance calculations to select unlabeled samples that are most dissimilar to positive samples. In the following, we will elaborate on the details of this module.

(1) Inter-class Distance Calculation.

In the distance calculation layer, we calculate the cosine similarity $Sim(x_i, x_j)$ between each unlabeled sample x_i and all positive samples x_j, using the feature representations extracted from this layer, as defined in Eq. (7).

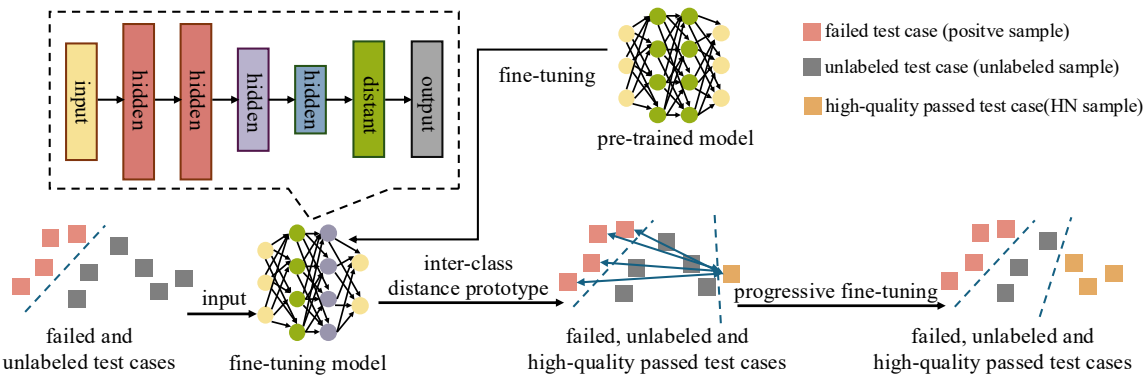

Fig. 4: The label selection process of ***EVANS***

$Sim(x_i, x_j)$ ranges from [-1,1]. A value close to 1 indicates high similarity, while a value close to -1 suggests greater dissimilarity. In this context, lower similarity implies greater inter-class distance, increasing the possibility that x_i is an HN sample. To quantify this dissimilarity, we define $MaxSim$, which represents the maximum distance between an unlabeled sample and any positive sample:

$$\text{MaxSim}(x_i) = \max_{x_j \in x_p} (1 - \text{Sim}(x_i, x_j)) \tag{13}$$

A higher MaxSim value indicates that x_i is more distinct from the all positive samples. Next, we rank all unlabeled samples based on their MaxSim scores and select the Top-K samples with the highest MaxSim values as HN samples. This strategy is effective because CC tests often exhibit similar execution behaviors to failing tests, rather than RP test cases. Consequently, unlabeled samples with higher MaxSim values are more likely to be RP tests rather than CC tests.

(2) Progressive Fine-tuning. To enhance the model's ability to distinguish CC from RP tests, we propose a progressive fine-tuning strategy. This strategy enables the model to iteratively select reliable HN samples from the unlabeled set X_u, gradually expanding the training set for the contrastive learning module while maintaining high sample quality. The overall process consists of four key steps.

a) Step 1: Initial Fine-tuning: The pre-trained model is first fine-tuned using the available labeled samples. Specifically, we use the labeled failed test cases as positive samples (X_p) and the initially selected HN samples as negative samples. The model weights are updated and stored as Ω_0. No additional pseudo-labels are introduced in this step.

b) Step 2: Pseudo-Negative Selection: After the initial fine-tuning, we leverage the trained model to predict the similarity scores between unlabeled and positive samples. The selection of new HN samples follows these steps:

- For each unlabeled sample $x_i \in X_u$, calculate its cosine similarity with all labeled positive samples $x_j \in X_p$.
- Identify the minimum cosine similarity score for each unlabeled sample.
- Rank all unlabeled samples based on their MaxSim scores and select the *Top-K* samples as HN samples.

c) Step 3: Incremental Training: After selecting new HN samples, we expand the training set by incorporating them alongside the original labeled samples. This iterative process ensures that the model learns progressively from more reliable negative samples, improving its ability to differentiate between CC and RP test cases.

d) Step 4: Adaptive Weight Adjustment: Since newly added HN samples may contain noise, we employ a dynamic weight decay strategy to reduce their influence during fine-tuning. Specifically, the weight of each training sample at iteration e is defined as:

$$W_i^e = 1 - \frac{e}{E_m + 1}, \quad e = \{1, 2, ..., E_m\} \tag{14}$$

where e represents the current fine-tuning iteration, and E_m is the maximum number of iterations. This approach ensures that earlier HN samples have a stronger influence on the training process, while later ones contribute less to prevent noise accumulation.

The progressive fine-tuning process terminates when the model performance on the validation set ceases to improve.

E. Weakly-Supervised Contrastive Learning Module

This module is an essential component of ***EVANS***. In this module, ***EVANS*** constructs contrastive learning pairs using HN and positive samples X_p selected in Section III-D. The goal is to bring similar samples closer together while ensuring that different samples remain distinct. To achieve this, ***EVANS*** employs a contrastive loss function based on InfoNCE [54]:

$$\mathcal{L}_{\text{con}} = -\sum_{(x_i, x_j) \in P} \log \frac{\exp(s(f(x_i), f(x_j)))}{\sum_{(x_k, x_m) \in N} \exp(s(f(x_k), f(x_m)))} \tag{15}$$

where: $(x_i, x_j) \in P$ are positive sample pairs, meaning both samples belong to the same category, such as $\{HN, HN\}$ and $\{X_p, X_p\}$. $(x_k, x_m) \in N$ are negative sample pairs, meaning the two samples belong to different categories, such as $\{HN, X_p\}$. $f(x)$ represents the feature embedding of sample x. $s(f(x_i), f(x_j))$ measures the similarity between two sample embeddings. By minimizing L_{con}, the model learns to increase the similarity of samples in the same category. At the same time, it reduces the similarity of samples from different categories. This improves feature discrimination.

After training, the model uses the learned feature representations to estimate whether an unlabeled sample $x_i \in X_u$ belongs to the CC. The inference process consists of the following steps:

1) Pair each unlabeled sample X_u with a positive sample X_p and input them into the trained model.

2) Calculate the probability $p(X_u)$, which represents the probability that X_u is a CC.
   - If $p(X_u)$ is close to 1, the sample is likely a CC.
   - If $p(X_u)$ is close to 0, the sample is likely an RP.

This contrastive learning module helps ***EVANS*** refine its classification of CC and RP. By using HN samples and contrastive loss, the model improves its ability to separate different test case categories. This enhances overall classification accuracy in PU learning.

F. Overall Training Workflow

For each program, all samples are used without splitting into training, validation, or test sets. The model is first pre-trained to learn semantic representations of coverage data, then utilizes all samples in the HN Selection stage to identify HN samples. Finally, contrastive learning is performed using positive and HN samples, enabling the model to predict CC probabilities for the remaining unlabeled samples.

IV. EXPERIMENTS SETTING

In this section, we evaluate ***EVANS*** for CC detection, investigate the effect of key parameters, and examine its impact on FL. To achieve this, we address the following research questions (RQs):

RQ1: How effectively does ***EVANS*** identify CC test cases?

RQ2: How do distance calculation layers, progressive tuning and pre-training impact ***EVANS***'s performance?

RQ3: How does ***EVANS*** improve FL effectiveness?

A. Dataset

To evaluate ***EVANS*** and answer these research questions, we use Defects4J, a widely adopted benchmark dataset. Introduced by Just et al. [3] in 2014, it provides a comprehensive collection of real-world Java programs with known faults. Table II presents details of six representative programs in Defects4J. Versions denote the number of faulty program versions. LoC (k) indicates program size (in thousands of lines of code). Tests indicate the total test cases per program. The description summarizes each program's functionality. These metrics illustrate the dataset's diversity and complexity, supporting a comprehensive evaluation of ***EVANS***.

TABLE II: The details of Defects4J

Program	Versions	LoC(k)	Tests	Description
Chart	26	96	3565	Java chart library
Closure	133	90	4,457	Closure compiler
Lang	65	22	4,589	Apache commons-lang
Math	106	85	12,089	Apache commons-math
Mockito	38	23	20,200	Mocking framework for Java
Time	27	28	50,933	Standard date and time library
Total	**395**	**388**	**333,018**	**-**

B. Baselines

To answer RQ1, we evaluate ***EVANS*** against several CC detection methods, including:

(1) **RF** [16]: Combines ensemble random forests with coverage data and majority voting for CC detection.
(2) **CBCFL** [17]: Integrates failure context (impactful statements) via cluster analysis to identify CC test cases.
(3) **ContraCC** [50]: Uses contrastive learning to maximize intra-class similarity and minimize inter-class similarity.
(4) **FCCI** [55]: Fuzzy expert system with suspiciousness, fault-proneness, and fault-masking features.
(5) **NeuralCCD** [18]: Learns expert features via MLP for CC detection.
(6) **MLCCI** [56]: Trains random forest on 60 expert features derived from 20 spectrum-based formulas.
(7) **TriCoCo** [57]: Uses triplet network to detect CC with high precision.
(8) **CORE** [13]: Integrates multi-features via ensemble learning for CC detection.

To answer RQ3, we follow Xie et al. [57], integrate ***EVANS*** into the FL methods Dstar [7] and MUSE [58], and additionally include the machine learning based FL method CNN-FL [32]. We evaluate its impact through two strategies.

(1) Removes identified CC test cases.
(2) Relabels CC test cases from 0 to 1.

C. Evaluation Metrics

In this section, we present the evaluation metrics used to evaluate the effectiveness of ***EVANS*** [12], [14], [17], [53], [59]–[61].

For RQ2, we use three widely adopted metrics to evaluate ***EVANS***'s performance:

(1) **Recall:** Measures the proportion of correctly identified CC test cases among all actual CC test cases:
$$\text{Recall} = \frac{|T_{cc} \cap T'_{cc}|}{|T_{cc}|} \tag{16}$$
where T_{cc} is the set of actual CC test cases, and T'_{cc} is the set predicted as CC.
(2) **Precision:** Evaluates how many predicted CC test cases are actually correct:
$$\text{Precision} = \frac{|T_{cc} \cap T'_{cc}|}{|T'_{cc}|} \tag{17}$$
(3) **F1-score:** The harmonic mean of recall and precision, balancing both metrics:
$$F1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \tag{18}$$

For RQ3, four widely used evaluation metrics in FL are used to evaluate the effectiveness of FL [8]–[10], [13], [17], [33], [57], [62]–[65].

(1) **Top-K:** Checks if at least one faulty statement ranks within the top K positions. We follow prior studies by setting $K \in \{1, 3, 5\}$.
(2) **Mean Average Rank (MAR):** Computes the average rank of all faulty statements.
(3) **Mean First Rank (MFR):** Records the rank of the first correctly identified faulty statement.
(4) **Wilcoxon Signed-Rank tests (WSR):** Determines whether the differences between two experimental conditions are statistically significant, following prior studies [13] with a significance threshold of p = 0.05.
(5) **Cohen's d:** Quantifies the standardized mean difference between two methods. Following common practice, $d = 0.2$, $d = 0.5$, and $d = 0.8$ indicate small, medium, and large effects, respectively.

TABLE III: The effectiveness of ***EVANS*** in detecting CC test cases compared to the state-of-the-art baselines. The best results are highlighted in bold and background. The suboptimal results are highlighted in background.

Program	Method	Recall	Precision	F1	Program	Method	Recall	Precision	F1	Program	Method	Recall	Precision	F1
Chart	RF	**96%**	13%	24%	Math	RF	**95%**	15%	26%	Lang	RF	**89%**	15%	26%
	CBCFL	58%	18%	28%		CBCFL	56%	23%	33%		CBCFL	61%	29%	40%
	ContraCC	81%	34%	48%		ContraCC	51%	35%	41%		ContraCC	69%	34%	46%
	FCCI	60%	71%	65%		FCCI	48%	69%	57%		FCCI	33%	50%	40%
	NeuralCCD	32%	67%	43%		NeuralCCD	16%	60%	25%		NeuralCCD	19%	93%	32%
	MLCCI	59%	70%	64%		MLCCI	50%	52%	51%		MLCCI	73%	50%	59%
	TriCoCo	59%	**90%**	**71%**		TriCoCo	30%	**93%**	46%		TriCoCo	48%	**96%**	64%
	CORE	38%	44%	41%		CORE	32%	48%	38%		CORE	29%	42%	34%
	EVANS	70%	73%	**71%**		***EVANS***	73%	77%	**75%**		***EVANS***	68%	66%	**67%**
Closure	RF	**100%**	20%	33%	Mockito	RF	**99%**	32%	48%	Time	RF	**99%**	3%	6%
	CBCFL	75%	20%	31%		CBCFL	86%	42%	57%		CBCFL	84%	5%	10%
	ContraCC	59%	49%	54%		ContraCC	47%	57%	52%		ContraCC	50%	45%	47%
	FCCI	23%	60%	33%		FCCI	30%	66%	41%		FCCI	19%	67%	30%
	NeuralCCD	28%	66%	39%		NeuralCCD	13%	47%	20%		NeuralCCD	30%	35%	32%
	MLCCI	36%	67%	47%		MLCCI	55%	56%	55%		MLCCI	47%	50%	48%
	TriCoCo	35%	**76%**	48%		TriCoCo	32%	72%	44%		TriCoCo	63%	55%	58%
	CORE	36%	39%	37%		CORE	35%	41%	38%		CORE	33%	43%	37%
	EVANS	70%	67%	**68%**		***EVANS***	75%	**77%**	**76%**		***EVANS***	41%	**69%**	**52%**

D. Implementation Details

For the pretraining stage, we adopt a masked autoencoder architecture with hidden layer dimensions [1024,512,256,128,16], ReLU activation, and dropout ratios ranging from 0.2 to 0.35. The input features are randomly masked with a ratio of 5%. The model is optimized using Adam with a learning rate of 0.001, batch size 64, mean square error loss, and trained for 30 epochs. For the weakly-supervised stage, we use the pretrained encoder followed by a two-layer classifier [32,2] with ReLU and dropout (0.3). The model is optimized with Adam (lr = 0.001), batch size 32, cross-entropy loss, and trained for 100 epochs. Training/validation splits are 80/20 unless otherwise stated.

V. EXPERIMENTAL RESULTS

*A. RQ1: How effectively does **EVANS** identify CC test cases?*

To evaluate the effectiveness of ***EVANS*** in CC detection, we compare it against eight widely used baseline methods on the Defects4J dataset. These baselines include supervised methods (RF [16], NeuralCCD [18], MLCCI [56], TriCoCo [13], CORE [57]), unsupervised methods (CBCFL [17], FCCI [55]), and self-supervised methods (ContraCC [50]). It is worth noting that CORE relies on a large set of additional features and operates under a fully supervised setting, whereas ***EVANS*** is a weakly supervised method that only leverages coverage data. For a fair evaluation, we re-implemented CORE under the same experimental conditions as ***EVANS***, that is, excluding its extensive additional features except for coverage data. For evaluation, we employ widely used metrics: Recall, Precision, and F1 score.

Table III presents the detection results of ***EVANS*** and the baselines. The best results are highlighted in bold with a dark gray background, while the second-best results are shown with a light gray background. The experimental results demonstrate that ***EVANS*** consistently achieves the highest F1 score across all datasets. In particular, in the Mockito program, ***EVANS*** attains an F1 score of 76%, surpassing the second-best method, CBCFL, by 19%, indicating its superior ability to correctly classify CC test cases. Similarly, in Math and Closure, ***EVANS*** achieves F1 scores of 75% and 68%, respectively, outperforming all baselines. The variation in F1-scores across programs (52% Time vs. 76% Mockito) can be explained by dataset-specific characteristics. Time has fewer fault-revealing tests and highly imbalanced coverage, making HN selection more difficult. These challenges affect all methods, most of which fall below 50%. Even so, ***EVANS*** achieves the best result (52%) on Time, showing robustness. Beyond overall performance, we observe notable trends across different baselines. Although TriCoCo achieves high precision (up to 90%–96%), this is accomplished by explicitly sacrificing recall to reduce false positives, as acknowledged in [13]. This trade-off results in limited CC coverage and lower F1 performance, whereas ***EVANS*** achieves a better balance between precision and recall, enabling more robust detection. CORE, when re-implemented with only coverage features for fair comparison, yields poor F1 scores (34%–41%), indicating that its advantage largely comes from additional features. In contrast, RF and CBCFL exhibit high recall but very low precision (e.g., 15%–20%), leading to some false positives that degrade detection reliability. These noises impact downstream tasks such as FL by introducing misleading patterns that reduce localization accuracy. FCCI and MLCCI, despite their relatively high Precision, struggle to comprehensively capture CC test cases, resulting in Recall values as low as 19%-50%. Traditional supervised methods, such as NeuralCCD, heavily rely on fully labeled datasets. In contrast, ***EVANS*** effectively learns from a limited set of positive labels and a large pool of unlabeled test cases, allowing it to generalize well across diverse datasets.

> ***Answer to RQ1: EVANS*** *consistently outperforms all baselines in CC detection, achieving the highest F1 score across multiple datasets. Its ability to balance Precision and Recall ensures accurate CC identification while minimizing false positives. Moreover,* ***EVANS****'s PU learning framework enables effective generalization in weakly supervised settings, making it a practical and scalable approach for CC detection.*

*B. RQ2: How do distance calculation layers, progressive tuning and pre-training impact **EVANS**'s performance?*

In RQ2, we investigated the impact of the selected distance calculation layer and progressive fine-tuning on ***EVANS***' per-

TABLE IV: Performance Comparison Across Programs

Method	Chart	Closure	Lang	Math	Mockito	Time
EVANS	95.55%	89.68%	91.65%	89.04%	93.27%	99.39%
EVANS_input	86.45%	80.91%	83.09%	79.27%	84.56%	90.45%
EVANS_middle	89.67%	83.49%	83.81%	83.78%	88.71%	91.85%

formance. Specifically, we aimed to determine which hidden layer should be used to calculate the distance between unlabeled and positive samples. Based on previous studies [66], we focused on three key layers: the input (IN) layer, the lowest-dimensional (LD) hidden layer, and the intermediate (IM) hidden layer. The IN layer retains the original coverage information, reflecting detailed testing behavior. The LD layer provides a compact feature representation, reducing redundancy and enhancing the model's ability to distinguish subtle differences between samples. The IM layer balances original input and compressed representations, preserving structural details and key distinguishing features. We evaluated the HN selection performance for each layer, as shown in Table IV. To investigate the impact of different layer selections, we define three configurations: ***EVANS*** utilizes the LD hidden layer for HN selection while ***EVANS*_input** selects the IN layer and ***EVANS*_middle** applies the IM layer. Table IV presents the selection accuracy across six benchmark programs. ***EVANS*** consistently achieves the highest accuracy across all datasets, with an average performance significantly surpassing ***EVANS***_input and ***EVANS***_middle. For instance, in the Chart and Time programs, ***EVANS*** attains 95.55% and 99.39% accuracy, respectively, demonstrating its robust capability in identifying reliable HN samples. In contrast, ***EVANS***_input exhibits the lowest performance and ***EVANS***_middle also underperforms compared to ***EVANS***. These results confirm that selecting the LD layer in ***EVANS*** provides the most effective feature representation for distinguishing CC from RP test cases. This validates the rationale behind our choice, ensuring both optimal accuracy and stability in HN selection.

We also evaluated the impact of progressive fine-tuning and pre-training by comparing ***EVANS*** with its two variants: ***EVANS* (w/o PF)**, which removes progressive fine-tuning, and ***EVANS* (w/o pre-t)**, which excludes the pre-training phase. As shown in Table V, ***EVANS*** consistently outperforms both variants across most datasets. Although ***EVANS* (w/o PF)** achieves slightly higher accuracy in *Chart* and *Math*, it identifies significantly fewer HN samples, confirming the value of progressive fine-tuning. More importantly, ***EVANS* (w/o pre-t)** shows a clear performance drop in all programs, with accuracy decreases ranging from about 2% to 7%. This indicates that pre-training effectively enhances the representation quality of coverage vectors, which in turn improves HN selection and overall CC detection performance. Overall, both components contribute to ***EVANS***'s effectiveness, where progressive fine-tuning stabilizes HN selection and pre-training strengthens the feature space for reliable distance computation.

Answer to RQ2: *The results of RQ2 show that using the LD layer for distance calculation significantly improves the model's ability to distinguish between CC and RP test cases. Additionally, progressive fine-tuning and pre-training play a key role in the selection of HN samples. These findings enhance the HN selection process in* ***EVANS*** *and improve its effectiveness in CC detection.*

TABLE V: Comparison results of ***EVANS (w/o PF)***, **EVANS (w/o pre-t)** and EVANS

Method	**Chart**	**Closure**	**Lang**	**Math**	**Mockito**	**Time**
EVANS	95.55% (658)	89.68% (34032)	91.65% (802)	89.04% (2097)	93.27% (3736)	99.39% (9954)
EVANS (w/o PF)	96.00% (564)	88.15% (30042)	92.55% (615)	90.45% (1751)	91.04% (3048)	99.54% (7153)
EVANS (w/o pre-t)	92.94% (640)	82.47% (31299)	87.96% (770)	86.42% (2034)	85.76% (3435)	97.84% (9799)

The percentage values indicate the accuracy of identifying RP in selected HN, while the numbers in parentheses represent the total number of selected RP.

C. RQ3: How does *EVANS* improve FL effectiveness?

FL techniques rank suspicious program entities based on test case results. However, CC test cases introduce noise, misranking faulty entities, and reducing localization accuracy [14]. To evaluate this impact, we integrate ***EVANS*** into the three widely used FL methods and compare it with eight CC detection baselines. We adopt two strategies to process CC test cases identified by ***EVANS*** and the baselines. The clean strategy removes CC test cases to eliminate misleading execution data. The relabel strategy reassigns their labels from passed to failed (0 to 1) to ensure fault ranking accurately reflects their influence.

Table VI presents the FL effectiveness under both Relabel and Clean strategies. Under the Relabel strategy, ***EVANS*** consistently outperforms all baselines. For example, on Dstar, ***EVANS*** improves Top-3 accuracy by 100% over the original baseline. In contrast, under the Clean strategy, ***EVANS*** achieves competitive results: while it is slightly inferior to TriCoCo on some projects and metrics, it still outperforms all remaining baselines. This is because TriCoCo detects fewer CC tests but favors precision, making it particularly effective when CC tests are removed under the Clean strategy. Nevertheless, ***EVANS*** remains robust, consistently ranking among the top methods under both strategies and delivering clear improvements in FL accuracy. Figure 5 further illustrates the impact of CC handling on MFR and MAR. ***EVANS*** achieves lower values than most baselines across both strategies, confirming its effectiveness in FL. To verify statistical significance, we conduct the WSR [61] and Cohen's d [67] on the MFR metric. As summarized in Table VII, ***EVANS*** achieves statistically significant improvements over all eight baselines, with medium to large effect sizes, further validating its advantage.

Answer to RQ3: ***EVANS*** *significantly enhances FL effectiveness under both strategies. With the* ***clean strategy****,* ***EVANS*** *reduces MFR and MAR while improving Top-3 accuracy by 100% over original Dstar. With the* ***relabel strategy****,* ***EVANS*** *achieves gains of 53.85%, 72%, and 63.16% in Top-1, Top-3, and Top-5 metrics, respectively.*

TABLE VI: Comparison results under Relabel and Clean strategies for Dstar, MUSE, and CNN-FL.

Strategy	Method	Dstar					MUSE					CNN-FL				
		MFR	MAR	Top-1	Top-3	Top-5	MFR	MAR	Top-1	Top-3	Top-5	MFR	MAR	Top-1	Top-3	Top-5
Relabel	Original	360.12	822.91	13	25	38	308.05	771.40	16	26	49	1055.44	2208.65	5	9	15
	RF	1696.48	2081.82	2	6	10	1545.64	1986.99	2	7	10	3250.76	3890.14	1	2	3
	CBCFL	1284.81	1424.71	6	19	25	939.60	1379.46	10	24	31	2400.31	3155.62	3	6	10
	ContraCC	254.62	738.31	23	47	60	201.26	634.95	23	44	63	1180.92	2540.31	7	15	22
	FCCI	334.98	784.45	10	21	34	301.20	744.23	12	21	35	1402.15	2894.65	4	10	16
	NeuralCCD	289.04	735.53	15	29	43	274.00	645.73	19	31	56	1350.88	2680.55	5	12	18
	MLCCI	270.77	639.70	16	31	48	222.91	559.31	21	32	60	1225.73	2410.92	6	14	21
	TriCoCo	220.20	512.61	18	35	52	186.51	470.99	22	35	68	1108.34	2290.88	7	16	24
	CORE	590.04	1000.63	11	22	33	538.90	939.28	13	25	39	1573.24	2733.70	2	2	8
	EVANS	230.40	541.45	25	50	69	183.59	482.51	25	51	69	950.22	1985.73	9	18	28
Clean	Original	360.12	822.91	13	25	38	308.05	771.40	16	26	49	1055.44	2208.65	5	9	15
	RF	499.98	907.57	14	28	38	442.68	843.79	14	31	39	1880.16	3310.44	2	4	7
	CBCFL	718.15	980.43	3	3	4	623.40	882.99	3	4	4	1450.22	2680.87	1	2	3
	ContraCC	342.02	1277.26	12	38	47	283.22	939.54	16	39	54	1255.33	2505.44	6	12	20
	FCCI	430.22	1003.29	13	29	37	410.32	934.33	19	42	57	1388.54	2704.19	5	11	18
	NeuralCCD	300.11	702.32	15	35	48	267.86	614.47	18	38	52	1209.64	2412.85	7	14	22
	MLCCI	255.85	698.30	17	40	55	222.51	611.74	19	45	63	1155.22	2295.61	8	15	24
	TriCoCo	208.64	652.11	22	57	70	186.30	612.46	26	75	87	1120.81	2248.90	10	21	30
	CORE	362.47	865.15	10	33	35	334.05	615.59	11	38	40	1345.66	2589.61	3	7	10
	EVANS	241.97	603.45	20	43	62	200.89	609.41	20	56	66	980.12	2005.41	13	29	33

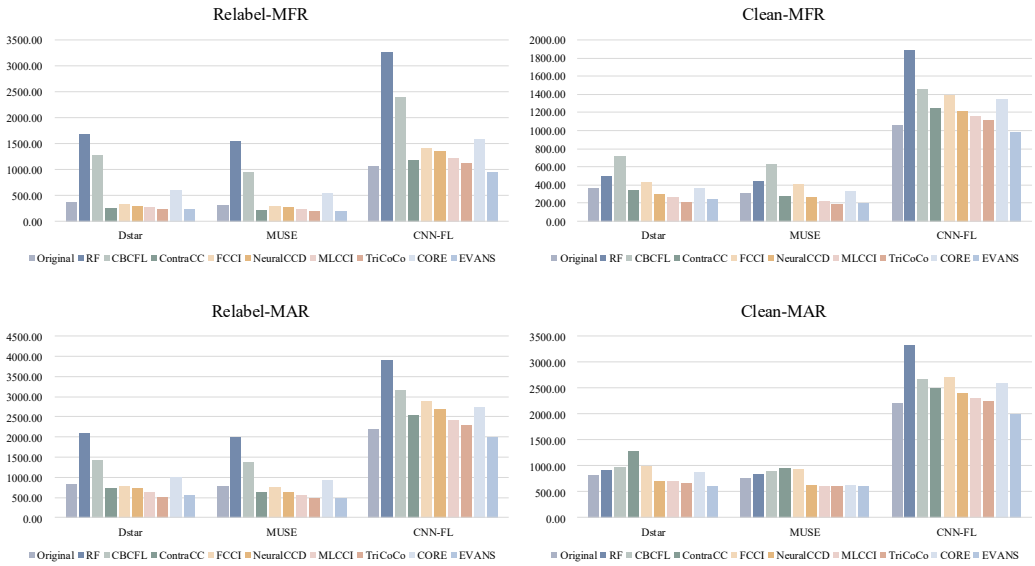

Fig. 5: Comparison of MFR and MAR under clean and relabel strategies.

TABLE VII: The WSR and Cohen's d results comparing EVANS with eight baselines.

Comparison	WSR	Cohen's d	Comparison	WSR	Cohen's d	Comparison	WSR	Cohen's d
EVANS vs. Dstar	**BETTER**	0.90	EVANS vs. MUSE	**BETTER**	0.92	EVANS vs. CNN-FL	**BETTER**	0.87
EVANS vs. RF	**BETTER**	2.7	EVANS vs. RF	**BETTER**	2.13	EVANS vs. RF	**BETTER**	1.23
EVANS vs. CBCFL	**BETTER**	2.33	EVANS vs. CBCFL	**BETTER**	1.63	EVANS vs. CBCFL	**BETTER**	0.93
EVANS vs. ContraCC	**BETTER**	0.64	EVANS vs. ContraCC	**BETTER**	0.62	EVANS vs. ContraCC	**BETTER**	0.82
EVANS vs. FCCI	**BETTER**	0.91	EVANS vs. FCCI	**BETTER**	0.90	EVANS vs. FCCI	**BETTER**	0.92
EVANS vs. NeuralCCD	**BETTER**	0.81	EVANS vs. NeuralCCD	**BETTER**	0.81	EVANS vs. NeuralCCD	**BETTER**	0.88
EVANS vs. MLCCI	**BETTER**	0.72	EVANS vs. MLCCI	**BETTER**	0.70	EVANS vs. MLCCI	**BETTER**	0.85
EVANS vs. TriCoCo	**BETTER**	0.13	EVANS vs. TriCoCo	**BETTER**	0.11	EVANS vs. TriCoCo	**BETTER**	0.76
EVANS vs. CORE	**BETTER**	1.27	EVANS vs. CORE	**BETTER**	1.08	EVANS vs. CORE	**BETTER**	0.81

WSR and Cohen's d results confirm that these improvements are statistically significant, demonstrating that accurate CC identification leads to more effective FL.

VI. Discussion

A. Why does *EVANS* Work?

We evaluate the effectiveness of ***EVANS*** in CC detection by examining its key advantages. ***EVANS*** uses the *High-Confidence Pseudo-Negative Sample Selection Module* to identify HN samples from unlabeled data. As shown in Table IV, it achieves 95.55% HN precision in Chart and 99.39% in Time. Through progressive fine-tuning, ***EVANS*** expands the HN set in Mockito from 3,048 to 3,736 samples, a 16.7% increase, while maintaining 93.27% precision (Table V). ***EVANS*** also surpasses all baselines in overall CC detection. In Table III, it achieves an F1-score of 75% in Math and 68% in Closure, outperforming the second-best methods by 19–28%. In Lang, it achieves balanced performance: 66% Precision, 68% Recall, and a 67% F1 scoremore than twice that of NeuralCCD (32%). These improvements are further enhanced by contrastive learning. The weakly supervised module learns fine-grained representations from 89,132 positive–HN sample pairs. This directly contributes to a 100% gain in Top-3 FL accuracy (Table VI).

B. Impact of Top-K selection.

During weakly supervised learning in ***EVANS***, HN samples are crucial for helping the model learn the relationship between positive and HN samples. Although these samples may contain

some noise, they provide essential contrastive information for training. If the proportion of RP test cases within the HN samples is too low, the model may fail to capture key differences between CC and RP test cases, hindering its ability to effectively distinguish between the two. In HN selection process, we rank all unlabeled samples by their inter-class distance to positive samples and select the top K with the greatest distance as HN samples. To evaluate the impact of different values of K, we conducted an empirical study on four datasets: Lang-26, Lang-2, Math-76, and Math-88. As shown in Fig. 6, setting $K = 0.2$ resulted in peak precision across all datasets. Specifically, Lang-26 and Math-76 achieved 100% precision, while Math-88 achieved 97.1% precision at $K = 0.2$, with minimal fluctuation. This configuration effectively reduced the risk of model confusion due to excessive noise.

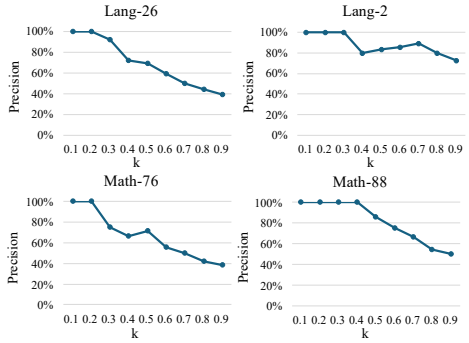

Fig. 6: The line chart of the precision values under different values of k.

We further validated this choice on the Defects4J dataset. As shown in Table VIII, setting $K = 0.2$ maintained high precision across all evaluated programs. Notably, the Time program achieved 99.39% precision, while the Math program, despite being the lowest, still achieved 89.04%. The average precision across all six projects was 93.23%.

TABLE VIII: The proportion of RP in HN

Program	Chart	Closure	Lang	Math	Mockito	Time
Precision	95.55%	89.68%	91.65%	89.04%	93.27%	99.39%

C. Computational Overhead and Scalability

We report the computational overhead of ***EVANS*** and other CC detection methods in Table IX. It shows that ***EVANS*** maintains a moderate and acceptable computational overhead across programs of different sizes. Although it incurs higher costs than lightweight baselines such as RF, CBCFL, and FCCI, its execution time remains comparable to neural-based methods like NeuralCCD and TriCoCo. For instance, on Lang and Math, ***EVANS*** requires 16.54s and 14.61s, which are close to other neural approaches. Even on large-scale programs such as Closure (58.68s), its overhead is slightly lower than CORE (61.43s). These results indicate that ***EVANS*** achieves a good balance between efficiency and effectiveness, with overhead remaining within a reasonable range for practical use.

D. Threats to Validity

This section discusses the potential threats to the validity of ***EVANS*** and the measures taken to mitigate them. We categorize these threats into internal validity, external validity, and conclusion validity.

TABLE IX: Comparison of Computational Overhead Among EVANS and Other CC Detection Methods

Method	Program					
	Chart	Closure	Lang	Math	Mockito	Time
RF	3.57	50.79	1.53	2.63	3.77	29.73
CBCFL	1.43	11.62	0.30	0.82	0.84	3.59
ContraCC	4.01	51.92	1.48	2.44	4.12	29.37
FCCI	1.37	12.14	0.29	0.79	0.87	3.73
NeuralCCD	12.28	45.05	13.69	10.55	18.35	33.49
MLCCI	3.92	53.21	1.46	2.51	4.06	28.94
TriCoCo	12.39	50.15	12.92	10.82	15.99	32.04
CORE	14.33	61.43	15.34	12.67	18.95	35.76
EVANS	16.35	58.68	16.54	14.61	19.51	40.37

Internal Validity. Implementation errors may affect results. We conducted code reviews, unit tests, and open-sourced our implementation for replication. To reduce variability, all experiments were repeated ten times and results are averaged.

External Validity. Our evaluation focuses on Java datasets, which may limit generalizability to other programming languages (e.g., C, Python) and application domains. Language-specific syntax, runtime behavior, or testing frameworks could affect ***EVANS***'s performance. Additionally, ***EVANS*** relies on test coverage data, which may be incomplete or unavailable in practice. Future work can explore applying ***EVANS*** to multilingual datasets and diverse domains, such as embedded systems, web applications, or scientific computing. Incorporating additional dynamic signals, such as execution logs, variable tracking, or runtime traces, may further improve robustness and broaden applicability.

Conclusion Validity. Results are averaged over ten runs to ensure consistency. ***EVANS***'s performance depends on accurate HN selection. Misclassifying CC as RP introduces noise. We mitigate this by sensitivity analysis on similarity thresholds and by optimizing hyperparameters (e.g., margin α) using cross-validation to ensure stable performance. Experiments use single-fault localization and multi-fault scenarios are left for future work.

VII. Conclusion

In this paper, we propose ***EVANS***, an approach for detecting coincidental correctness test cases using positive and unlabeled learning. ***EVANS*** comprises two key modules. The *high-confidence pseudo-negative selection module* selects high-confidence pseudo-negative samples by analyzing inter-class distances. The *weakly supervised contrastive learning module* improves coincidental correctness detection by enhancing the distinction between CC and reliably passed test cases. Experiments on Defects4J confirm that ***EVANS*** performs well compared with other CC detection methods. In future work, we plan to extend ***EVANS*** to a wider range of software projects and programming languages to improve its generalizability and robustness in CC detection.

VIII. Acknowledgements

This research is supported by the National Natural Science Foundation of China (No. 62272072) and the Chongqing Technology Innovation and Application Development Special Project (No. CSTB2023TIAD-STX0029)

REFERENCES

[1] 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.

[2] W. Masri, R. Abou-Assi, M. El-Ghali, and N. Al-Fatairi, "An empirical study of the factors that reduce the effectiveness of coverage-based fault localization," in *Proceedings of the 2nd International Workshop on Defects in Large Software Systems: held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2009)*, 2009, pp. 1–5.

[3] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 international symposium on software testing and analysis*, 2014, pp. 437–440.

[4] W. Masri and R. A. Abou-Assi, "Prevalence of coincidental correctness and mitigation of its impact on fault localization," 2014.

[5] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. IEEE, 2006, pp. 39–46.

[6] 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.

[7] 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.

[8] 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.

[9] Y. Lou, Q. Zhu, J. Dong, X. Li, Z. Sun, D. Hao, L. Zhang, and L. Zhang, "Boosting coverage-based fault localization via graph-based representation learning," in *Proceedings of the 29th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2021, pp. 664–676.

[10] H. Xie, Y. Lei, M. Yan, Y. Yu, X. Xia, and X. Mao, "A universal data augmentation approach for fault localization," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 48–60.

[11] J. Sohn and S. Yoo, "Empirical evaluation of fault localisation using code and change metrics," *IEEE Transactions on Software Engineering*, vol. 47, no. 8, pp. 1605–1625, 2019.

[12] 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.

[13] H. Xie, Y. Lei, M. Yan, S. Li, X. Mao, Y. Yu, and D. Lo, "Towards more precise coincidental correctness detection with deep semantic learning," *IEEE Transactions on Software Engineering*, 2024.

[14] 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.

[15] A. Bandyopadhyay, "Mitigating the effect of coincidental correctness in spectrum based fault localization," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 2012, pp. 479–482.

[16] 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.

[17] J. Yu, Y. Lei, H. Xie, L. Fu, and C. Liu, "Context-based cluster fault localization," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, 2022, pp. 482–493.

[18] Z. Tao, Y. Lei, H. Xie, and J. Hu, "Neuralccd: Integrating multiple features for neural coincidental correctness detection," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2023, pp. 85–96.

[19] S. Dass, X. Xue, and A. S. Namin, "Detection of coincidentally correct test cases through random forests," *arXiv preprint arXiv:2006.08605*, 2020.

[20] J. Bekker and J. Davis, "Learning from positive and unlabeled data: A survey," *Machine Learning*, vol. 109, no. 4, pp. 719–760, 2020.

[21] S. Jain, M. White, and P. Radivojac, "Recovering true classifier performance in positive-unlabeled learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[22] Anonymous, "Our artifact." [Online]. Available: https://anonymous.4open.science/r/EVANS-B651

[23] X.-C. Wen, X. Wang, C. Gao, S. Wang, Y. Liu, and Z. Gu, "When less is enough: Positive and unlabeled learning model for vulnerability detection," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 345–357.

[24] E. Sansone, F. G. De Natale, and Z.-H. Zhou, "Efficient training for positive unlabeled learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 11, pp. 2584–2598, 2018.

[25] K. Jaskie and A. Spanias, "Positive and unlabeled learning algorithms and applications: A survey," in *2019 10th international conference on information, intelligence, systems and applications (iisa)*. IEEE, 2019, pp. 1–8.

[26] J. Bekker, "Learning from positive and unlabeled data," 2018.

[27] Z. Zhu, L. Wang, P. Zhao, C. Du, W. Zhang, H. Dong, B. Qiao, Q. Lin, S. Rajmohan, and D. Zhang, "Robust positive-unlabeled learning via noise negative sample self-correction," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 3663–3673.

[28] H. Mu, R. Sun, G. Yuan, and G. Shi, "Positive unlabeled learning-based anomaly detection in videos," *International Journal of Intelligent Systems*, vol. 36, no. 8, pp. 3767–3788, 2021.

[29] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[30] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[31] W. E. Wong, L. Zhao, Y. Qi, K.-Y. Cai, and J. Dong, "Effective fault localization using bp neural networks." in *SEKE*. Citeseer, 2007, pp. 374–379.

[32] Z. Zhang, Y. Lei, X. Mao, and P. Li, "Cnn-fl: An effective approach for localizing faults using convolutional neural networks," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 445–455.

[33] Z. Zhang, Y. Lei, X. Mao, M. Yan, X. Xia, and D. Lo, "Context-aware neural fault localization," *IEEE Transactions on Software Engineering*, vol. 49, no. 7, pp. 3939–3954, 2023.

[34] Y. Yan, S. Jiang, Y. Zhang, and C. Zhang, "Improving fault localization via weighted execution graph and graph attention network," *Journal of Software: Evolution and Process*, p. e2619, 2024.

[35] X. Gou, A. Zhang, C. Wang, Y. Liu, X. Zhao, and S. Yang, "Software fault localization based on network spectrum and graph neural network," *IEEE Transactions on Reliability*, 2024.

[36] 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.

[37] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and X. Zhang, "A study of effectiveness of deep learning in locating real faults," *Information and Software Technology*, vol. 131, p. 106486, 2021.

[38] X. Wang, H. Yu, X. Meng, H. Cao, H. Zhang, H. Sun, X. Liu, and C. Hu, "Mtl-transfer: Leveraging multi-task learning and transferred knowledge for improving fault localization and program repair," *ACM Transactions on Software Engineering and Methodology*, 2024.

[39] L. Zhang, S. Guo, Y. Guo, H. Li, Y. Chai, R. Chen, X. Li, and H. Jiang, "Context-based transfer learning for structuring fault localization and program repair automation," *ACM Transactions on Software Engineering and Methodology*, 2024.

[40] X. Meng, X. Wang, H. Zhang, H. Sun, and X. Liu, "Improving fault localization and program repair with deep semantic features and transferred knowledge," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1169–1180.

[41] J. Sohn and S. Yoo, "Fluccs: 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.

[42] Y. Lei, T. Wen, H. Xie, L. Fu, C. Liu, L. Xu, and H. Sun, "Mitigating the effect of class imbalance in fault localization using context-aware generative adversarial network," in *2023 IEEE/ACM 31st International*

Conference on Program Comprehension (ICPC). IEEE, 2023, pp. 304–315.

[43] Y. Lei, H. Xie, T. Zhang, M. Yan, Z. Xu, and C. Sun, "Feature-fl: Feature-based fault localization," *IEEE Transactions on Reliability*, vol. 71, no. 1, pp. 264–283, 2022.

[44] Y. Qin, S. Wang, Y. Lou, J. Dong, K. Wang, X. Li, and X. Mao, "Agentfl: Scaling llm-based fault localization to project-level context," *arXiv preprint arXiv:2403.16362*, 2024.

[45] S. Kang, G. An, and S. Yoo, "A quantitative and qualitative evaluation of llm-based explainable fault localization," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1424–1446, 2024.

[46] R. Widyasari, J. W. Ang, T. G. Nguyen, N. Sharma, and D. Lo, "Demystifying faulty code with llm: Step-by-step reasoning for explainable fault localization," *arXiv preprint arXiv:2403.10507*, 2024.

[47] T. A. Budd and D. Angluin, "Two notions of correctness and their relation to testing," *Acta informatica*, vol. 18, no. 1, pp. 31–45, 1982.

[48] L. Weishi and X. Mao, "Alleviating the impact of coincidental correctness on the effectiveness of sfl by clustering test cases," in *2014 Theoretical Aspects of Software Engineering Conference*. IEEE, 2014, pp. 66–69.

[49] X. Yang, M. Liu, M. Cao, L. Zhao, and L. Wang, "Regression identification of coincidental correctness via weighted clustering," in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2. IEEE, 2015, pp. 115–120.

[50] M. Li, Y. Lei, H. Xie, J. Wang, C. Liu, and Z. Deng, "Contrastive coincidental correctness representation learning," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 252–263.

[51] H. Cao, L. Li, Y. Chu, M. Deng, P. Wang, and C. Zhao, "A coincidental correctness test case identification framework with fuzzy c-means clustering," *Multimedia Systems*, vol. 29, no. 3, pp. 1089–1101, 2023.

[52] Z. Li, M. Li, Y. Liu, and J. Geng, "Identify coincidental correct test cases based on fuzzy classification," in *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*. IEEE, 2016, pp. 72–77.

[53] 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.

[54] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[55] 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.

[56] Y. Wu, S. Tian, Z. Yang, Z. Li, Y. Liu, and X. Chen, "Identifying coincidental correct test cases with multiple features extraction for fault localization," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2023, pp. 800–809.

[57] H. Xie, Y. Lei, M. Li, M. Yan, and S. Zhang, "Combining coverage and expert features with semantic representation for coincidental correctness detection," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1770–1782.

[58] S. Moon, Y. Kim, M. Kim, and S. Yoo, "Ask the mutants: Mutating faulty programs for fault localization," in *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE, 2014, pp. 153–162.

[59] F. Feyzi and S. Parsa, "A program slicing-based method for effective detection of coincidentally correct test cases," *Computing*, vol. 100, no. 9, pp. 927–969, 2018.

[60] 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.

[61] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 196–202.

[62] 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.

[63] M. N. Rafi, D. J. Kim, A. R. Chen, T.-H. Chen, and S. Wang, "Towards better graph neural network-based fault localization through enhanced code representation," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 1937–1959, 2024.

[64] Y. Yan, S. Jiang, Y. Zhang, and C. Zhang, "An effective fault localization approach based on pagerank and mutation analysis," *Journal of Systems and Software*, vol. 204, p. 111799, 2023.

[65] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *Proceedings of the 30th ACM SIGSOFT international symposium on software testing and analysis*, 2021, pp. 67–78.

[66] Y. Yin, Y. Feng, S. Weng, Y. Yao, J. Liu, and Z. Zhao, "Datactive: Data fault localization for object detection systems," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 895–907.

[67] K. Kelley and K. J. Preacher, "On effect size." *Psychological methods*, vol. 17, no. 2, p. 137, 2012.