# Systematic Analysis of Learning-Based Software Fault Localization

Ya Zou[1], Hui Li[1,*], Dongcheng Li[2], Man zhao[1], and Zizhao Chen[3]
[1]School of Computer Science, China University of Geosciences, Wuhan, China
[2]Department of Computer Science, California State Polytechnic University - Humboldt, Arcata, USA
[3]Department of Computer Science, University of Texas at Dallas, Richardson, USA
zouya@cug.edu.cn, huili@vip.sina.com, dl313@humboldt.edu, zhaoman@cug.edu.cn, zizhao.chen2@utdallas.edu
*corresponding author

*Abstract*—This paper reviews the evolution of learning-based software fault localization methods, examining their benefits, challenges, and prospective developments in practical applications. We analyze the limitations of traditional methods and provide an in-depth discussion of learning-based fault localization approaches, covering the application of supervised, unsupervised, and reinforcement learning techniques in software fault localization. Furthermore, we explore key issues in current research and future research directions, and analyze threats that could impact the effectiveness of the research. In conclusion, we summarize the current status and future prospects of these methods, along with our perspective on the forthcoming advancements in this field.

*Keywords-software fault localization; learning based; suspicious code; survey*

## 1. Introduction

When encountering software errors, programmers often invest significant time and effort in attempting to reproduce, understand, and fix these errors [1][2]. Fault localization in software systems has always been a key issue in the field of software engineering [3]. Existing fault localization techniques encompass various traditional methods [4][5]. Print Debugging involves inserting print statements to monitor program execution by outputting variable values and states. Although simple, this approach becomes time-intensive in complex, large-scale systems. Breakpoint Debugging allows developers to pause program execution at specific locations, providing detailed inspection. It offers greater control but can be laborious during debugging. Static Analysis inspects source or binary code for potential errors, detecting common issues like unused variables or memory leaks. For example, FindBugs [6] is an open-source Java static analysis tool and has experience in production environments. FindBugs evaluates which types of defects can be effectively detected with relatively simple techniques, and aids developers in understanding how to integrate these tools into software development; Checkstyle [7] is a development tool that helps developers write Java code that adheres to coding standards. It automates the process of checking Java code, thereby freeing humans from this monotonous but important task. This makes it highly suitable for projects that wish to enforce coding standards [8]. However, it may lack accuracy for dynamic problems or highly complex systems.

Traditional methods often rely on rules, manual experience, and static analysis, but perform poorly in dealing with complex, large-scale, and highly dynamic systems. As machine learning technology rapidly advances, learning-based approaches to software fault localization are increasingly coming into focus. Leveraging the strengths of vast data and learning algorithms, these methods achieve more precise and efficient software fault localization.

According to our repository, Figure 1 shows the number of papers related to learning-based software fault localization published in top journals and leading conferences focusing on software engineering from 2010 to November 2023. These journals and conferences include IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, International Conference on Software Engineering, ACM International Symposium on Foundations of Software Engineering, and ACM International Conference on Automated Software Engineering. This trend supports the view that learning-based software fault localization is not only an important research topic but has also been widely discussed and studied in top software engineering journals and conferences over the past decade.

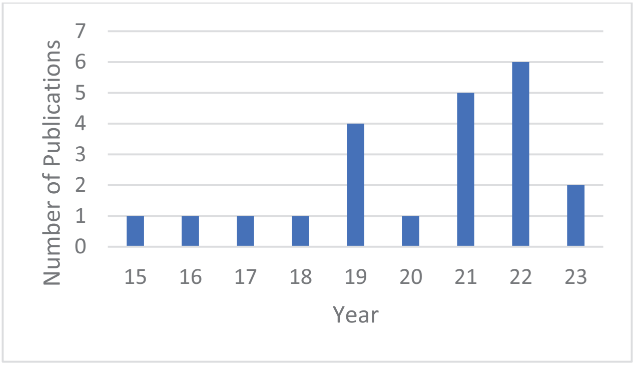

Figure 1. Papers on software fault localization from 2010 to November 2023

This paper first reviews the limitations of traditional fault localization methods, such as the challenges of locating faults in large-scale systems, over-reliance on manual expertise, and inadequate adaptation to system dynamics. Following that, we explore in depth the learning-based fault localization methods, encompassing the application of techniques like supervised learning, unsupervised learning, and reinforcement learning in software fault localization. We evaluate the advantages and limitations of these methods and discuss their application in real-world scenarios.

The remainder of this paper is organized as follows: We first provide an in-depth exploration of learning-based fault localization methods and elucidate their principles and performance in Section 2. In Section 3, critical topics and research domains, including evaluation metrics, fault localization tools, and datasets, are extensively discussed to comprehensively examine their impact on fault localization research. Section 4 focuses on potential challenges that may affect the credibility and validity of the study, evaluating limitations in drawing research conclusions. The final chapter, concluding the entirety of this paper, summarizes the research findings, presents conclusions, and explores future directions for further study.

## 2. Learning-Based Software Fault Localization

Learning-based approaches to software fault localization present an innovative and effective solution to overcome the constraints of traditional methods in managing complexity and diversity. In this chapter, we systematically examine the application of learning algorithms in fault localization, encompassing subfields such as supervised learning, unsupervised learning, semi-supervised learning, reinforcement learning, and transfer learning.

### 2.1. Supervised Learning Methods

In recent years, supervised learning [9] methods have been widely applied in the field of software fault localization, significantly improving the accuracy of fault detection and prediction. The fundamental principle of supervised learning is to learn from a labeled training dataset and generate a model, which is then used to predict unknown data [10]. Within the realm of fault localization tasks, supervised learning has the capacity to uncover latent associations between program characteristics and errors, thus efficiently pinpointing potential error sites.

#### 2.1.1. Machine Learning-Based Methods

These methods utilize machine learning models, such as Support Vector Machines (SVM), clustering algorithms, etc., for software fault localization. They integrate supervised or unsupervised learning techniques, extracting information from program execution or static features, and build models to predict and localize potential fault locations.

Wu et al. [11] proposed a direct fault localization method, combining Gaussian Mixture Models (GMM) and Support Vector Machines (SVM). This method initially preprocesses the training data using a GMM-based clustering algorithm, then improves the learning capability of SVM by replacing its constant penalty factor with two adjustable parameters, revealing the relationship between test case coverage information and execution results. By comparing its efficiency with other techniques on the Siemens Suite, the study shows that this method significantly improves localization accuracy in both single and multiple fault scenarios, without incurring additional testing costs.

Roychowdhury and Khurshid [12] proposed a new method for fault localization using feature selection techniques in machine learning. Every additional failure or successful run can provide a plethora of distinct information, which can help locate errors in the code. Statements with the greatest diversity of feature information can point to the most suspicious lines of code. This method outperforms the most advanced fault localization methods in most subject programs in the Siemens Suite.

Shaikh et al. [13] proposed a software fault localization method that utilizes a subset of dynamic invariants as features, combined with supervised learning techniques. This method focuses on extracting runtime information during program execution and possible static features in the source code to identify fault locations. These features are used to train supervised learning models to capture the association between erroneous and normal states. After training, the model can be employed to predict fault locations in unfamiliar data. By integrating subsets of dynamic invariants with other features, this method has made significant progress in enhancing the accuracy of fault localization.

Liang et al. [14] proposed a fault localization system named CAST, which utilizes deep learning technology and custom program abstract syntax trees (ASTs) to automatically and efficiently locate potential software bug source files. The system achieves this by extracting lexical semantics from bug reports (such as words) and source files (e.g., method names), as well as program semantics (such as abstract syntax trees, ASTs). CAST also utilizes custom ASTs to enhance the Tree-based Convolutional Neural Network (TBCNN) model, which can distinguish between user-defined methods and system-provided methods, and reflect their contributions to causing defects. Additionally, the custom AST groups syntactic entities with similar semantics and removes those with little or redundant semantics to enhance learning performance.

#### 2.1.2.Deep Learning-Based Fault Localization Methods

These methods utilize deep learning models, such as Convolutional Neural Networks (CNN) and enhanced neural networks, for software fault localization. They focus on using neural network architectures to process software code, features, and contextual information, aiming to improve the accuracy and efficiency of fault localization.

Xiao et al. [15] proposed a deep learning-based model named DeepLoc, consisting of an enhanced Convolutional Neural Network (CNN) that takes into account bug fix time and frequency, along with word embedding and feature detection technologies. DeepLoc uses word embeddings to represent words in bug reports and source files, preserving their semantic information, and employs various CNNs to detect their features. Research results show that DeepLoc, compared to traditional CNNs, improved the MAP by 10.87% to 13.4%. In terms of Accuracy@k, MAP, and MRR, DeepLoc outperforms the current four most advanced methods (DeepLocator, HyLoc, LR+WE, and BugLocator) in a shorter computation time.

Li et al. [16] proposed a deep learning method named DeepFL, which uses multiple dimensions of fault diagnostic information to predict potential fault locations. Research results indicate that DeepFL significantly outperforms the state-of-the-art trap/FLUCCS methods in fault localization (for instance, locating over 50 faults in the Top-1 category). Furthermore, DeepFL exhibits high efficiency in making cross-project predictions.

Li et al. [17] proposed a deep learning-based fault localization method named FixLocator. FixLocator can identify erroneous statements within one or several methods; its method-level fault localization model zeroes in on fixed methods, while its statement-level model attends to jointly fixed statements. The correct execution of this learning pattern can promote each other, using cross-stitch units for soft sharing of model parameters, allowing the effects of MethFL and StmtFL to propagate mutually. Furthermore, the study explores a new feature for fault localization – co-change statements, and employs a graph-based convolutional network to integrate different types of program dependencies. Experiments prove that, compared to the state-of-the-art statement-level FL baseline, FixLocator achieved an improvement of 26.5% to 155.6% in localizing CC fixed statements.

### 2.1.3.Learning-to-Rank Based Fault Localization Methods

These methods rely on learning-to-rank techniques, using ranking algorithms or models to arrange code elements (such as methods, statements, etc.) in order of likelihood, to determine the probable location of faults. They combine learning and ranking phases, sorting and locating faults through trained models or algorithms.

B. Le et al. [18] proposed a new fault localization method, named Savant, that adopts a learning-to-rank strategy. It utilizes potential invariant differences and suspicion scores as features, ranking methods based on these features to assess their likelihood of being the root cause of faults. Savant comprises four key steps: method clustering and test case selection, invariant mining, feature extraction, and method ranking. After these four steps, Savant produces a brief ranked list, highlighting methods that are likely to harbor bugs. Evaluation results indicate that, in the top 1, 3, and 5 positions of the generated ranking list, Savant correctly identified 63.03, 101.72, and 122 erroneous methods, respectively. Compared to several state-of-the-art spectrum-based fault localization baselines, Savant achieved an increase of 57.73%, 56.69%, and 43.13% in the number of successfully localized faults at the top 1, 3, and 5 positions, respectively.

Kim et al. [19] proposed a new fault localization technique, PRINCE, which is a learning-to-rank based method. PRINCE utilizes Genetic Programming (GP) to combine multiple sets of localization input features previously studied independently. It encompasses dynamic features, including Spectrum-Based Fault Localization (SBFL) and Mutation-Based Fault Localization (MBFL) techniques. Additionally, it employs static features, such as dependency information and the structural complexity of program entities. GP combines all these pieces of information to train a ranking model for fault localization. Empirical evaluation results on 65 real faults from CoREBench, 84 artificial faults from SIR, and 310 real faults from Defects4J indicate that PRINCE significantly outperforms the most advanced SBFL, MBFL, and learn-to-rank techniques. On average, PRINCE only needs to inspect 2.4% of the executed statements to locate faults, showing a 4.2-fold and 3.0-fold increase in precision compared to SBFL and MBFL, respectively.

Sohn and Yoo [20] expanded SBFL by incorporating code and change metrics previously studied in the context of defect prediction, such as size, age, and code changes. This work utilizes the suspicion values from existing SBFL formulas and these source code metrics as features, applying two learning-to-rank techniques: Genetic Programming (GP) and Linear Rank Support Vector Machines (SVM). They used 210 real-world faults from the Defects4J repository, conducted 10-fold cross-validation, and evaluated method-level fault localization. GP with additional source code metrics ranked the faulty methods at the top in 106 out of 210 faults, and achieved similar results in 173 faults. This work significantly improved the state-of-the-art SBFL formulas, with the best formula ranking 49 and 127 faults at the top, respectively.

Xuan and Monperrus [21] introduced MULTRIC, a learning-based fault localization method that integrates multiple ranking metrics for efficient fault localization. The method unfolds in two key stages: learning and ranking. By learning from both faulty and non-faulty code, MULTRIC constructs a ranking model. When a new fault arises, it uses the learned model to calculate the final ranking. After empirical comparison with four extensively studied metrics and three recently proposed metrics, results indicate that MULTRIC is more effective in fault localization than advanced measuring methods like Tarantula, Ochiai, and Ample.

### 2.1.4.Coverage-Based Fault Localization Methods

These methods use program coverage information, typically capturing this information through program representations or graphs, and apply various techniques and models to analyze and infer fault locations.

Lou et al. [22] introduced Grace, a novel coverage-based fault localization technique. This technique makes full use of detailed coverage information and graph-based representational learning. Firstly, it introduces a new graph-based representation method, encapsulating exhaustive coverage information and the fine-grained structure of the code into a graph. Subsequently, Grace utilizes a gated graph neural network to learn valuable features from the graph-based coverage representation and ranks program entities in a list format. Evaluations on the widely-used benchmark Defects4J (V1.2.0) show that Grace's performance significantly surpasses that of the most advanced coverage-based fault localization methods. The study also reveals that Grace learns fundamental features from coverage information, which complements the information used by current learning-based fault localization methods.

### 2.1.5.Causal Inference-Based Fault Localization Methods

These methods integrate causal inference with machine learning, employing statistical causal reasoning and machine learning models for inferring and locating software faults. They consider causal relationships and reason about code variables and conditions to predict the probable location of faults.

Kucuk et al. [23] proposed UniVal, a new software fault localization technique. This method utilizes causal inference techniques and machine learning, integrating information about predicate outcomes and variable values. UniVal uses a random forest learner as a supervised learning model to localize faults in various elements of a program, enabling a more precise assessment of the true fault-inducing effects of program statements. This is a novel fault localization technique based on statistical causal inference methods, combining value-based and predicate-based fault localization approaches by converting predicates into assignment statements. UniVal utilizes a machine learning model to minimize bias in estimates and assess the average causal effect of program variables, without needing to actually change the program or its execution. Currently, UniVal can handle numerical, Boolean, categorical, and string values. Experimental results show that UniVal performs better than several competing techniques.

Figure 2 shows the distribution of papers across supervised learning methods in our repository. Machine learning-based papers are the most predominant, accounting for 31% of all papers, followed by deep learning-based at 23%, and then causal inference-based at 8%. Coverge-based are the least, comprising only 7% of the total.

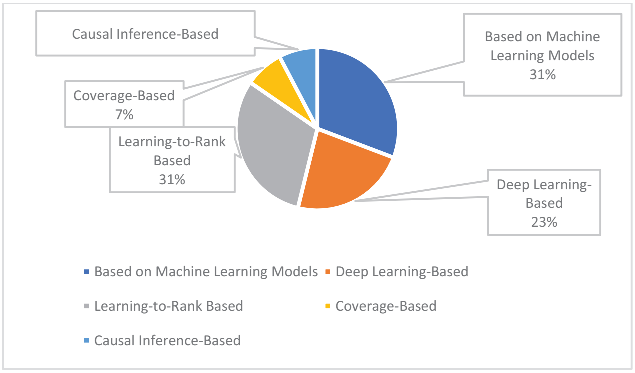

Figure 2. Distribution of papers across supervised learning methods

In summary, historical research has offered a diverse array of supervised learning approaches for software fault localization, spanning from traditional SVMs to advanced deep learning models, and even innovative methods incorporating causal inference. These techniques have significantly contributed to enhancing localization precision and adapting to a variety of projects and datasets. Future studies can delve deeper into the generalizability, practicality, and real-world project applications of these methods, aiming to identify the most appropriate technique for specific contexts.

## 2.2. Unsupervised Learning Methods

Unsupervised learning does not rely on labeled data, but instead seeks hidden patterns or intrinsic structures, such as anomaly detection and clustering. Clustering is one of the most commonly used unsupervised learning methods, dividing similar objects into groups or clusters to uncover the data's intrinsic structure. In fault localization tasks, clustering methods can identify anomalous code lines or modules and differentiate them from normally functioning parts, as in the k-means clustering and similarity-based method proposed by An [24]. At the start of testing, K-Means clustering is performed on the test case set, and the filtered test cases can cover more execution information. Subsequently, for test cases with failed execution results, test cases with similar execution information are filtered to better highlight the error information in the failed test cases. Experiments on the Defects4J dataset show that this method can be combined with other techniques to improve its efficiency and is well compatible with traditional software fault localization algorithms, achieving an average improvement rate of 13.37% in 8 scenarios.

Zhang et al. [25] proposed a set of anomaly detection techniques based on prefix trees, with the prefix tree model serving as a compact, lossless data representation of execution traces. Furthermore, the prefix tree distance metric provides an effective heuristic for guiding the search for closely related execution traces. In density-based algorithms, using prefix tree distance confines the k-nearest neighbors search to a small subset of nodes, significantly reducing computation time without sacrificing accuracy. Experimental studies show that in the automatic identification of software faults, methods guided by prefix trees and prefix tree distance significantly speed up the process.

## 2.3. Semi-Supervised Learning Methods

Although supervised learning is effective in fault localization, the problem of requiring a large amount of labeled data still exists. Semi-supervised learning fully utilizes both labeled and unlabeled data, employing self-training on unlabeled data or generative adversarial networks with minimal labeled data for precise training.

Wei et al. [26] points out that although supervised learning is effective in fault localization, the issue of needing a large amount of labeled data still exists. Semi-supervised learning fully exploits both labeled and unlabeled data, using self-training on unlabeled data or generative adversarial networks with a small amount of labeled data for precise training. Furthermore, the algorithm was validated on the Siemens

Suite dataset. By comparison with traditional supervised learning algorithms, the effectiveness of semi-supervised learning algorithms in software fault localization was demonstrated.

Zhu et al. [27] proposed the first semi-supervised bug localization model, BL-GAN, and introduced a promising generative adversarial network into BL-GAN, which generates file paths by searching the project directory tree instead of comparing all code file contents, thereby constructing synthetic error correction records that closely mimic reality, For handling error reports, BL-GAN utilizes an attention-based Transformer architecture to capture semantic and sequential information, and to capture proprietary structural information in code files, BL-GAN uses a novel multi-layer graph convolutional network to process source code in a graphical view. Extensive experiments on large-scale real-world datasets demonstrate that the BL-GAN model significantly outperforms the most advanced techniques in all evaluation metrics.

Yan et al. [28] proposed an entropy-based framework filter to filter unlabeled test cases, wherein statement-based entropy and testsuite-based entropy were constructed to measure the localization uncertainty of a given test set. Compared to a threshold value, unlabeled test cases with lower statement-based entropy or testsuite-based entropy are selected. Based on this, an integrated feature strategy based on statement entropy and testsuite entropy was presented to calculate the suspiciousness of statements. The efficiency of the filter was evaluated through six open-source programs and three spectrum-based fault localization methods. Results indicated that the fault localization efficiency using the effilter strategy, based on statement entropy and testsuite entropy, improved by 18.8% and 16.5%, respectively, compared to not using the effilter strategy. The filter based on statement entropy and testsuite entropy can improve fault localization in scenarios lacking test oracles, and has a certain enhancing effect on fault localization in practical applications.

## 2.4. Reinforcement Learning Methods

Reinforcement learning, as a method that learns through interaction with the environment, has shown significant potential in recent applications of software fault localization. The objective of reinforcement learning methods is to learn a policy that guides actions to maximize long-term rewards.

Chakraborty et al. [29] proposed a bug localization method based on reinforcement learning named RLOCATOR, which is an RL-based software fault localization approach. The innovative aspect of RLocator lies in its use of RL for error localization, including the formulation of the error localization process into an MDP, and its comparison with two state-of-the-art bug localization tools (FLIM and BugLocator), and evaluations indicate that RLocator significantly outperforms these two methods.

Moran et al. [30] proposed a method for automatically locating faults in reinforcement learning programs. The method, termed SBFL4RL, scrutinizes multiple executions to identify internal states that are often detrimental to program performance. Locating these states can aid testers in understanding known faults and even in detecting unknown faults. SBFL4RL underwent validation in two case studies, successfully pinpointing the injected faults. Preliminary results indicate that faults in reinforcement learning programs can be automatically located, and there is room for further research.

Li et al. [31] developed DEEPRL4FL, a deep learning-based fault localization approach, that frames fault localization as an image pattern recognition challenge, and pinpoints errors at both statement and method levels. DEEP RL4FL accomplishes this using cutting-edge code coverage representation learning and data dependency reinforcement learning for program statements. This method integrates two forms of reinforcement learning based on dynamic info in code coverage matrices with code representation learning focused on static info from commonly suspect source code. The approach draws inspiration from crime scene investigations, where detectives scrutinize crime scenes (failed test cases and statements) and relevant individuals (dependent statements), along with usual suspects with a history of similar offenses (analogous error codes in the training data). Regarding code coverage info, DEEP RL4FL initially sorts test cases and flags statements indicating errors, anticipating the model to discern patterns that distinguish between erroneous and correct statements/methods. In terms of inter-statement dependencies, it considers not only the statements themselves but also their data dependencies on other statements and data flows during execution. Finally, the code coverage matrix, the data dependencies between statements, and the vector representation of source code are combined and used as input for a classifier built with convolutional neural networks to detect faulty statements/methods.

## 2.5. Transfer Learning Methods

Transfer learning can utilize previously acquired knowledge and experience to solve new problems, enabling models to effectively address issues like low utilization of training data and high data annotation costs. By applying acquired knowledge to new, unlabeled software fault localization problems, transfer learning methods can significantly enhance the efficiency and accuracy of fault localization.

Meng et al. [32] introduced TRANSFER, a novel approach that leverages deep semantic features and harnesses knowledge from open-source data to enhance fault localization and program repair. The first step involves creating two extensive open-source bug datasets and developing 11 BiLSTM-based binary classifiers and a BiLSTM-based multi-classifier to discern deep semantic features of statements for fault localization and program repair. Following that, a blend of semantic, spectrum-based, and mutation-based features was employed, utilizing an MLP-based model for fault localization. Finally, semantic-based features were used to rank repair templates in the context of program repair.

Alhuman [33] proposed a model to generate counterfactual explanations for decisions made by fault localization models. The construction of a nonlinear neural network model enables the approximation and propagation representation of input information through neural systems. This indicates a high proficiency in transfer learning, even with minimal training data. The proposed XFL ensures transparent decision-making without compromising the model's performance. The proposed XFL ranks software program statements based on potential vulnerability scores approximated from training data. The model's performance is further assessed using various metrics, such as the number of evaluated statements, confidence in fault localization, and Top-N evaluation strategies.

Huo et al. [34] introduced a deep transfer learning approach for bug localization across different projects. TRANP-CNN, the proposed method, harnesses transferable semantic features from the source project and leverages the target project's labeled data for efficient cross-project error localization. This performance notably surpasses the cutting-edge deep learning-based bug localization solutions and several other advanced alternatives, taking into account a variety of standard evaluation metrics.

Zhu et al. [35] introduced an adversarial approach to transfer learning for bug localization, centered on the cross-project transfer of shared characteristics (namely, public information). COOBA employs a shared encoder to glean indicative public information from bug reports across projects, and utilizes distinct feature extractors for each project to extract private information from code files. COOBA integrates adversarial learning to guarantee efficient extraction of commonly shared public information across various projects. Comprehensive testing on four large-scale real-world datasets shows that the proposed COOBA method significantly surpasses current technological benchmarks.

Figure 3 shows the distribution of papers across all categories in our repository. Supervised learning-based papers are the most predominant, accounting for 52% of all papers, followed by transfer learning-based at 16%, and then both reinforcement learning and semi-supervised learning at 12%. Papers categorized under unsupervised learning are the least, comprising only 8% of the total.

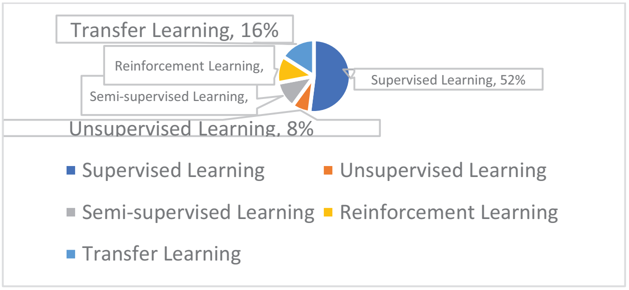

Figure 3. Distribution of papers across all categories

As a software fault localization technique, learning-based methods, with their immense potential and wide applicability, have gradually become a focus of research. However, these methods have their own strengths and limitations, and are suited to different problems and scenarios. In the future, with the emergence of new technologies such as deep learning and transfer learning, we anticipate more innovations and breakthroughs in learning-based fault localization methods.

## 3. Important Issues and Research Areas

This section focuses on the core issues and future research directions in the field of software fault localization. We thoroughly examine the selection and precision of assessment metrics, evaluate the merits and limitations of existing software fault localization tools and methods, and suggest approaches for their integration to boost efficacy. Moreover, we critically review current datasets, highlighting their limitations and stressing the need to create datasets that are more representative and practical. In conclusion, we spotlight nascent research domains within software fault localization, probing solutions for large-scale, intricate systems, the synergistic application of machine learning in fault localization, and strategies for precisely identifying faults in specialized domains. This section aims to provide an exhaustive review of these issues and offer guidance for upcoming research avenues, committed to propelling cutting-edge investigations and innovative advancements in software fault localization.

### 3.1. Evaluation Metrics

Measuring the effectiveness of fault localization methods using relevant formulas or scores is crucial as it provides objective quantitative metrics to assess the performance and accuracy of the localization algorithms. Such metrics disclose the efficacy of fault localization methods in detecting and pinpointing program errors, offering developers dependable feedback. The indispensability of this quantitative approach stems from its capacity to assist development teams in pinpointing the most efficient fault localization methods, guiding the judicious distribution and enhancement of resources, thus expediting the fault repair process. Comparing the scores or metrics of various methods enables the identification of which techniques are more reliable or apt for addressing certain types of problems in specific scenarios. Consequently, these metrics serve not just as an objective assessment of the localization algorithms' effectiveness but also furnish vital directions and strategies for teams to enhance and refine localization methods. Table 1 presents an overview of Metrics and Papers utilizing these metrics. It includes details such as the names of the metrics, brief descriptions of each metric, and information on which papers have utilized these metrics.

The following are some commonly used evaluation metrics:

Top N: Top N Rank refers to the number of errors associated with the files ranked in the top N in the returned results. Based on previous research [36][37][38], three values for N are considered: 1, 5, 10. For an error report, if at least one file that should be fixed is included in the top N query results, it indicates that the error has been located. The higher the value of this metric, the better the performance of error localization [39].

Table 1. Metrics

| Name | Formula | Paper Using The Metrics |
|---|---|---|
| MRR | $\mathrm{MRR} = \frac{1}{\lvert Q\rvert}\sum_{i=1}^{\lvert Q\rvert}\frac{1}{\mathrm{rank}_i}$ | [37,40,41,42] |
| MAP | $\mathrm{AvgP}_i = \sum_{i=1}^{M}\frac{P(j)\times \mathrm{pos}(j)}{\text{number of positive instances}}$ | [20,41,42] |
| Precision rate | $p = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$ | [7,25,43,44] |
| Recall rate | $R = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$ | [7,25,43,44] |
| F-measure | $F = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$ | [7,25,43] |
| MFR | $\mathrm{MFR} = \frac{1}{N}\sum_{i=1}^{N}\max(r_i)$ | [16,31,32,45,46,47,48] |
| MAR | $\mathrm{MAR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{m_i}\sum_{j=1}^{m_i} r_{ij}$ | [16,31,32,45,46,47,48] |

- MRR: MRR (Mean Reciprocal Rank) is a metric used to assess the process of generating potential response lists for queries. It measures by calculating the reciprocal of the rank of the first correct answer in the results of a query. The mean reciprocal rank then represents the average of the reciprocal ranks across a set of query results.

$$\mathrm{MRR} = \frac{1}{\lvert Q\rvert}\sum_{i=1}^{\lvert Q\rvert}\frac{1}{\mathrm{rank}_i} \tag{1}$$

  The higher the MRR value, the better the performance of the fault localization.

- MAP: MAP (Mean Average Precision) is a single metric for the quality of information retrieval [49], particularly suitable for cases where one query may involve multiple relevant documents. For a single query, Average Precision (AvgP) is the average of precision values obtained for that query. The calculation of precision values is as follows:

$$\mathrm{AvgP}_i = \sum_{i=1}^{M}\frac{P(j)\times \mathrm{pos}(j)}{\text{number of positive instances}} \tag{2}$$

  In this formula, j represents the ranking, M is the number of retrieved instances, and pos(j) indicates whether the instance at rank j is relevant. P(j) is the precision at a given cut-off rank j, defined as follows:

$$P(j) = \frac{\text{number of positive instances in top j positions}}{j} \tag{3}$$

  The Mean Average Precision for a set of queries is the average of the average precision values of all queries. In fault localization, a bug may be associated with multiple files.

- Precision rate: Precision rate is calculated as follows:

$$p = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{4}$$

  True positive refers to the number of identified code segments that are defective and are indeed faulty, while false positive denotes the number of code segments incorrectly identified as defective when they are actually free of faults. Precision rate is utilized in software fault localization to measure the accuracy and reliability of identifying faulty code segments.

- Recall rate: Recall rate measures the coverage of a localization method in identifying actual faults. This metric represents the probability of successfully identifying real faults, indicating the proportion of correctly identified faults among all actual faults. The formula for Recall rate is:

$$R = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{5}$$

  The Recall rate assesses the capability of fault localization methods in discovering actual faults. In the context of software fault localization, a higher Recall rate signifies the method's effectiveness in capturing and identifying genuine faults while minimizing undetected faults. This metric is crucial for evaluating the comprehensiveness and efficacy of fault localization techniques.

- F-measure: F-measure is a comprehensive evaluation metric that combines information from Precision and Recall. It quantifies the balance between Precision and Recall, offering an assessment of the overall performance of fault localization methods. The formula for F-measure is:

$$F = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

  In software fault localization, the F-measure serves as a holistic evaluation metric to assess the balance between accuracy and coverage of a localization method. A higher F-measure value indicates a well-balanced approach between precision and recall, crucial in the selection and evaluation of fault localization techniques. It stands as a significant indicator for evaluating the overall performance of methods, guiding improvements and optimizations in fault localization techniques.

- MFR: The formula for MFR is as follows:

$$\mathrm{MFR} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} \max(\mathrm{r_i}) \quad (7)$$

  N is the total number of faults.$r_i$ is the rank of the highest faulty statement in the i-th fault. This formula represents MFR as a metric for a project, which calculates the mean of the highest faulty statement's rank for each fault.

- MAR: The formula for MAR is as follows:

$$\mathrm{MAR} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}} \frac{1}{\mathrm{m_i}}\sum_{\mathrm{j}=1}^{\mathrm{m_i}} \mathrm{r_{ij}} \quad (8)$$

  N is the total number of faults. $m_i$is the number of faulty statements in the ii-th fault.$r_{ij}$ is the rank of the j-th faulty statement in the i-th fault. This formula represents MAR as a metric for a project, which calculates the average rank of all faulty statements for each fault, and then computes the mean of these average ranks for all faults.

These evaluation metrics play distinct roles in the realm of software fault localization. Top N focuses on the quantity of error files ranked highly, while MRR delves into the ranking of correct answers within query results. MAP suits scenarios involving multiple relevant documents in information retrieval. Precision and recall strike a balance between performance and accuracy, while the F-measure amalgamates both. MFR and MAR concentrate more on the efficiency and degree of improvement in inspecting code statements. When determining performance, choosing the appropriate metric relies on specific objectives and research contexts to ensure that the evaluation aligns with the desired outcomes. Each metric emphasizes different facets of fault localization methods, hence their selection should be context-dependent to comprehensively assess the performance and potential enhancements of fault localization methods.

## 3.2. Software Fault Localization Tools

One challenge faced by many empirical studies in software fault localization is the need for appropriate tools to support automatic or semi-automatic data collection and suspicion calculation. Table 2 provides a list of commonly used tools, including their names, brief descriptions and which papers have used these tools. All these seven tools can be obtained by contacting their authors.

Table 2. Summary of tools used in the fault localization studies

| Name | Brief Description | Paper Using The Tool |
|---|---|---|
| NP-CNN | Convolutional Neural Networks blend natural language and source code data to pinpoint potential error sources in software bug reports automatically. | [37] |
| DNNLOC | Employing Deep Neural Networks and Information Retrieval methods to facilitate the automatic identification of potential error sources in software bug reports. | [42] |
| DeepLocator | Leveraging deep learning models with semantic insights to boost the precision of automatically linking error source code with software bug reports, expediting the bug resolution process. | [44] |
| CAST | Utilizing deep learning and bespoke Abstract Syntax Trees to amalgamate semantic details from bug reports and source code, pinpointing potential error sources automatically. | [50] |
| HyLoc | Autoencoders and vector space models are applied to discern potential bug files linked to specific bug reports. | [40] |
| DeepLOC | Word embeddings represent semantically informative words in error reports and source files, with various CNNs employed to explore their features | [15] |
| BugFix | Tools for program debugging powered by machine learning techniques. | [51] |

- NP-CNN: A novel Convolutional Neural Network, NP-CNN, utilizes lexical and program structure information to learn unified features from the natural language and source code of programming languages, thereby automatically locating potential bug source code based on bug reports.
- DNNLOC: DNNLOC is a novel approach that combines Deep Neural Networks (DNN) with rVSM (an Information Retrieval (IR) technique). rVSM gathers text similarity features between bug reports and source files. DNN is used to learn connections between terms in error reports and potentially varying code tokens and terms in source files.
- DeepLocator: DeepLocator consists of an enhanced CNN (Convolutional Neural Network) proposed in the study that considers bug-fixing experience, a new rTF-IDuF method, and pre-trained word2vec technology. It enhances bug location performance by fully utilizing semantic information.
- CAST: CAST utilizes deep learning and custom program Abstract Syntax Trees (AST) for the automatic and effective localization of potential bug source files. Specifically, CAST extracts lexical semantics from bug reports (e.g., words) and source files (e.g., method names) as well as program semantics (e.g., AST) from source files. Furthermore, CAST uses a customized AST to enhance the Tree-based Convolutional Neural Network (TBCNN) model, which differentiates between user-defined methods and system-provided methods to reflect their contributions to defects. Additionally, the

custom AST groups syntactic entities with similar semantics and trims those with less or redundant semantics to improve learning performance.

- HyLoc: HyLoc is a novel approach that integrates Deep Neural Networks (DNN) with Information Retrieval (IR) technique rVSM. rVSM gathers textual similarity features between bug reports and source files. DNN is employed to learn associations between terms in error reports with different code tokens and terminologies in source files and documents, based on their frequency of occurrence in the reports and error files. HyLoc, combining functionalities built on DNN, rVSM, and project error repair history, achieves higher accuracy than state-of-the-art IR and machine learning techniques.
- DeepLOC: DeepLoc is built around an advanced Convolutional Neural Network that accounts for the timeliness and frequency of bug fixes, complemented by word embedding and feature detection techniques. It uses word embeddings for capturing the semantics of words in bug reports and source files and leverages different CNNs for feature extraction.
- BugFix: BugFix integrates machine learning concepts, enabling it to automatically learn from new debugging scenarios and bug fixes over time. This allows for more effective prediction of the most relevant bug fix suggestions for newly encountered debugging scenarios. The tool takes into account the static structure of statements, dynamic values used in successful and failed runs in those statements, and interesting value mapping pairs related to the statements.

### 3.3. Dataset

- Defects4J: Defects4J [52] is a widely used dataset for software defect research, comprising a series of real Java projects, each with multiple versions, some of which contain known defects. The uniqueness of this dataset lies in providing verifiable and reproducible real-world program defects, enabling researchers to assess the performance and accuracy of fault localization methods. Yiling Lou conducted an evaluation of Grace using both Defects4J (V1.2.0) and Defects4J (V2.0.0).
- Eclipse Platform UI: A graphical user interface provided by the Eclipse development platform, designed for software development purposes.
- AspectJ: A specialized extension for the Java programming language, introducing aspect-oriented programming capabilities.
- JDT: An integrated collection of Java development tools specifically designed for the Eclipse platform.
- Tomcat: A widely-used open-source Servlet container that facilitates the deployment and running of Java Servlet and JSP technologies.
- Birt: An open-source solution for generating and publishing reports, offering capabilities for building and customizing reports via the Eclipse platform.
- SWT: A comprehensive open-source widget toolkit designed for Java applications.

Table 3 offers an inventory of Datasets and Papers that utilize these datasets. It comprises the names of the datasets, brief descriptions of each dataset, and information on which papers have utilized these datasets.

Table 3. Dataset

| Dataset | Brief Description | Paper using the dataset |
|---|---|---|
| Defects4J | a software testing dataset designed for software defect research, comprising multiple projects with known bugs and corresponding test cases. | [17,20,22,23,32,53,54,55,56,57,58] |
| Eclipse UI | a user interface of a development platform for Eclipse | [40,42,44] |
| JDT | a suite of Java development tools for Eclipse | [40,42,44,59] |
| SWT | an open source widget toolkit for Java. | [40,42,44] |
| Tomcat | An open-source Servlet container used for running Java Servlets and JSP technologies | [40,42,44,60,61,62] |
| AspectJ | an aspect-oriented programming extension for Java programming language | [40,42,44,63,64,65,66,67] |
| Birt | An open-source reporting tool for creating and deploying reports, supporting construction and customization through the Eclipse IDE. | [51,68] |

## 4. Threats to Validity

During our comprehensive review study, we identified potential factors that could undermine the reliability and applicability of our research findings. Initially, the process of selecting literature and gathering data could exhibit some selection bias, potentially impacting the breadth and integrity of our analysis. We endeavored to include a diverse array of sources and studies comprehensively, but there remains a possibility that certain critical research works were inadvertently omitted.

Moreover, the reliability and accuracy of certain publications could fluctuate, necessitating careful consideration of how these variations might influence our findings. Additionally, the methodologies and techniques employed in our study might introduce specific constraints or limitations. Certain approaches might be limited in scope, failing to encapsulate every facet of the fault localization field comprehensively. Furthermore, the currency and representativeness of the data might impact the extent to which our results can be generalized, particularly in rapidly advancing technological areas. Additionally, the personal perspectives and biases of our research team could have influenced the selection and interpretation of the literature. Despite our efforts to remain objective and impartial, it's important to acknowledge that personal biases may still subtly influence the research process. In an attempt to minimize these potential threats, we adopted a multi-faceted approach, drawing from various sources and perspectives, and aimed for a comprehensive coverage of diverse research findings. Concurrently, we made concerted efforts to interpret and cite literature objectively and impartially while verifying the accuracy of our data. Nevertheless, it is still necessary to be mindful of the potential impact these threats could have on the interpretation and generalizability of the research results.

## 5. Conclusion

This article reviews both traditional and learning-based emerging methods in the field of software fault localization. Through in-depth discussions of supervised, unsupervised, semi-supervised, reinforcement, and transfer learning methods, we demonstrate the advantages, limitations, and applicable scenarios of each method in software fault localization. After detailed discussions of key issues, research domains, evaluation metrics, and datasets, we examined the threats that could impact the reliability and effectiveness of the research. Overall, despite certain limitations in the research, it offers new insights and directions for future research in software fault localization, and encourages further exploration and innovation to advance the field.

## References

[1] Jones, J.A., Harrold, M.J. and Stasko, J., 2002, May. Visualization of test information to assist fault localization. In Proceedings of the 24th international conference on Software engineering (pp. 467-477).

[2] Wong, W.E. and Debroy, V., 2010. Software Fault Localization. Encyclopedia of Software Engineering, 1, pp.1147-1156.

[3] Li, Y., Liu, P., Wong, W.E., Chau, N. and Hsu, C.W., 2023. Alternative Ranking Distance Metrics for Fault-Focused Clustering in Parallel Fault Localization. International Journal of Performability Engineering, 19(10), p.633-643.

[4] Wong, W.E. and Tse, T.H., editors. Handbook of software fault localization: foundations and advances. John Wiley & Sons; 2023.

[5] Wong, W.E., Gao, R., Li, Y., Abreu, R. and Wotawa, F., 2016. A survey on software fault localization. IEEE Transactions on Software Engineering, 42(8), pp.707-740.

[6] Ayewah, N., Pugh, W., Hovemeyer, D., Morgenthaler, J.D. and Penix, J., 2008. Using static analysis to find bugs. IEEE software, 25(5), pp.22-29.

[7] Zampetti, F., Mudbhari, S., Arnaoudova, V., Di Penta, M., Panichella, S. and Antoniol, G., 2022. Using code reviews to automatically configure static analysis tools. Empirical Software Engineering, 27(1), p.28.

[8] Novak, J. and Krajnc, A., 2010, May. Taxonomy of static code analysis tools. In The 33rd international convention MIPRO (pp. 418-422). IEEE.

[9] Arya, A. and Malik, S.K., 2023. Software Fault Prediction using K-Mean-Based Machine Learning Approach. International Journal of Performability Engineering, 19(2), p.133-143.

[10] Cunningham, P., Cord, M. and Delany, S.J., 2008. Supervised learning. In Machine learning techniques for multimedia: case studies on organization and retrieval (pp. 21-49). Berlin, Heidelberg: Springer Berlin Heidelberg.

[11] Wu, X., Zheng, W., Chen, J., Bai, H., Hu, D. and Mu, D., 2018, December. A GMM and SVM Combined Approach for Automatically Software Fault Localization. In 2018 IEEE International Conference on Progress in Informatics and Computing (PIC) (pp. 357-363). IEEE.

[12] Roychowdhury, S. and Khurshid, S., 2011, November. Software fault localization using feature selection. In Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering (pp. 11-18).

[13] Shaikh, A., Rizwan, S., Alghamdi, A., Islam, N., Elmagzoub, M.A. and Syed, D., 2022. A Learning-Based Fault Localization Approach Using Subset of Likely and Dynamic Invariants. Intelligent automation & soft computing, 31(3).

[14] Liang, H., Sun, L., Wang, M. and Yang, Y., 2019. Deep learning with customized abstract syntax tree for bug localization. IEEE Access, 7, pp.116309-116320.

[15] Xiao, Y., Keung, J., Bennin, K.E. and Mi, Q., 2019. Improving bug localization with word embedding and enhanced convolutional neural networks. Information and Software Technology, 105, pp.17-29.

[16] Li, X., Li, W., Zhang, Y. and Zhang, L., 2019, July. Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization. In Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis (pp. 169-180).

[17] Li, Y., Wang, S. and Nguyen, T.N., 2022, November. Fault localization to detect co-change fixing locations. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 659-671).

[18] B. Le, T.D., Lo, D., Le Goues, C. and Grunske, L., 2016, July. A learning-to-rank based fault localization approach using likely invariants. In Proceedings of the 25th international symposium on software testing and analysis (pp. 177-188).

[19] Kim, Y., Mun, S., Yoo, S. and Kim, M., 2019. Precise learn-to-rank fault localization using dynamic and static features of target programs. ACM Transactions on Software Engineering and Methodology (TOSEM), 28(4), pp.1-34.

[20] Sohn, J. and Yoo, S., 2017, July. Fluccs: Using code and change metrics to improve fault localization. In Proceedings of the 26th ACM SIGSOFT International

Symposium on Software Testing and Analysis (pp. 273-283).

[21] Xuan, J. and Monperrus, M., 2014, September. Learning to combine multiple ranking metrics for fault localization. In 2014 IEEE International Conference on Software Maintenance and Evolution (pp. 191-200). IEEE.

[22] Lou, Y., Zhu, Q., Dong, J., Li, X., Sun, Z., Hao, D., Zhang, L. and Zhang, L., 2021, August. 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 (pp. 664-676).

[23] Küçük, Y., Henderson, T.A. and Podgurski, A., 2021, May. Improving fault localization by integrating value and predicate based causal inference techniques. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 649-660). IEEE.

[24] An, D., Wang, S., Zhu, L., Yang, X. and Yan, X., 2022, December. Prefilter: A Fault Localization Method using Unlabelled Test Cases based on K-Means Clustering and Similarity. In 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C) (pp. 263-269). IEEE.

[25] Zhang, W., Bastani, F., Yen, I.L., Hulin, K., Bastani, F. and Khan, L., 2012, October. Real-time anomaly detection in streams of execution traces. In 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering (pp. 32-39). IEEE.

[26] Wei,Z., Xue,X., Xin,T., 2015. Research on Software Fault Localization Based on Semi-Supervised Learning Methods. Journal of Northwestern Polytechnical University, 33(2),pp.332-336.

[27] Zhu, Z., Tong, H., Wang, Y. and Li, Y., 2022. BL-GAN: Semi-supervised bug localization via generative adversarial network. IEEE Transactions on Knowledge and Data Engineering.

[28] Yan, X., Liu, B., Wang, S., An, D., Zhu, F., and Yang, Y., 2021. Efilter: An effective fault localization based on information entropy with unlabelled test cases. Information and Software Technology, 134, p.106543.

[29] Chakraborty, P., Alfadel, M. and Nagappan, M., 2023. RLocator: Reinforcement Learning for Bug Localization.

[30] Morán, J., Bertolino, A., De La Riva, C. and Tuya, J., 2023, July. Fault Localization for Reinforcement Learning. In 2023 IEEE International Conference On Artificial Intelligence Testing (AITest) (pp. 49-50). IEEE..

[31] Li, Y., Wang, S. and Nguyen, T., 2021, May. Fault localization with code coverage representation learning. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE) (pp. 661-673). IEEE.

[32] Meng, X., Wang, X., Zhang, H., Sun, H. and Liu, X., 2022, May. Improving fault localization and program repair with deep semantic features and transferred knowledge. In Proceedings of the 44th International Conference on Software Engineering (pp. 1169-1180).

[33] Alhumam, A., 2022. Explainable Software Fault Localization Model: From Blackbox to Whitebox. Computers, Materials & Continua, 73(1).

[34] Huo, X., Thung, F., Li, M., Lo, D. and Shi, S.T., 2019. Deep transfer bug localization. IEEE Transactions on software engineering, 47(7), pp.1368-1380.

[35] Zhu, Z., Li, Y., Tong, H. and Wang, Y., 2020, July. Cooba: Cross-project bug localization via adversarial transfer learning. In IJCAI.

[36] Schütze, H., Manning, C.D. and Raghavan, P., 2008. Introduction to information retrieval (Vol. 39, pp. 234-265). Cambridge: Cambridge University Press.

[37] Huo, X., Li, M. and Zhou, Z.H., 2016, July. Learning unified features from natural and programming languages for locating buggy source code. In IJCAI (Vol. 16, No. 2016, pp. 1606-1612).

[38] Abreu, R., Zoeteweij, P. and Van Gemund, A.J., 2007, September. On the accuracy of spectrum-based fault localization. In Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007) (pp. 89-98). IEEE.

[39] Saha, R.K., Lawall, J., Khurshid, S. and Perry, D.E., 2014, September. On the effectiveness of information retrieval based bug localization for c programs. In 2014 IEEE international conference on software maintenance and evolution (pp. 161-170). IEEE.

[40] Lam, A.N., Nguyen, A.T., Nguyen, H.A. and Nguyen, T.N., 2015, November. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 476-481). IEEE.

[41] Wang, B., Xu, L., Yan, M., Liu, C. and Liu, L., 2020. Multi-dimension convolutional neural network for bug localization. IEEE Transactions on Services Computing, 15(3), pp.1649-1663.

[42] Lam, A.N., Nguyen, A.T., Nguyen, H.A. and Nguyen, T.N., 2017, May. Bug localization with combination of deep learning and information retrieval. In 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC) (pp. 218-229). IEEE.

[43] Fang, F., Wu, J., Li, Y., Ye, X., Aljedaani, W. and Mkaouer, M.W., 2021. On the classification of bug reports to improve bug localization. Soft Computing, 25, pp.7307-7323.

[44] Lee, N.K., Azizan, F.L., Wong, Y.S. and Omar, N., 2018. DeepFinder: An integration of feature-based and deep learning approach for DNA motif discovery. Biotechnology & Biotechnological Equipment, 32(3), pp.759-768.

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

[46] Jones, J.A. and Harrold, M.J., 2005, November. Empirical evaluation of the tarantula automatic fault-localization technique. In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (pp. 273-282).

[47] Wong, W.E., Debroy, V., Li, Y. and Gao, R., 2012, June. Software fault localization using dstar (d*). In 2012 IEEE Sixth International Conference on Software Security and Reliability (pp. 21-30). IEEE.

[48] Papadakis, M. and Le Traon, Y., 2015. Metallaxis‐FL: mutation‐based fault localization. Software Testing, Verification and Reliability, 25(5-7), pp.605-628.

[49] Voorhees, E.M., 1999, November. The trec-8 question answering track report. In Trec (Vol. 99, pp. 77-82).

[50] Cord, M. and Cunningham, P. eds., 2008. Machine learning techniques for multimedia: case studies on

organization and retrieval. Springer Science & Business Media.

[51] Jeffrey, D., Feng, M., Gupta, N. and Gupta, R., 2009, May. BugFix: A learning-based tool to assist developers in fixing bugs. In 2009 IEEE 17th International Conference on Program Comprehension (pp. 70-79). IEEE.

[52] Just, R., Jalali, D. and Ernst, M.D., 2014, July. 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 (pp. 437-440).

[53] Vancsics, B., Horváth, F., Szatmári, A. and Beszédes, A., 2021, March. Call frequency-based fault localization. In 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 365-376). IEEE.

[54] Vancsics, B., Horváth, F., Szatmári, A. and Beszédes, A., 2022. Fault localization using function call frequencies. Journal of Systems and Software, 193, p.111429.

[55] Silva, A., Martinez, M., Danglot, B., Ginelli, D. and Monperrus, M., 2021. Flacoco: Fault localization for java based on industry-grade coverage. arxiv preprint arxiv:2111.12513.

[56] Zhang, Z., Lei, Y., Mao, X., Yan, M., Xia, X. and Lo, D., 2023. Context-aware neural fault localization. IEEE Transactions on Software Engineering.

[57] Yang, A.Z., Le Goues, C., Martins, R. and Hellendoorn, V., 2024, February. Large language models for test-free fault localization. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (pp. 1-12).

[58] Li, Z., Shi, B., Wang, H., Liu, Y. and Chen, X., 2021, August. Hmbfl: Higher-order mutation-based fault localization. In 2021 8th International Conference on Dependable Systems and Their Applications (DSA) (pp. 66-77). IEEE.

[59] Amario de Souza, H., de Souza Lauretto, M., Kon, F. and Lordello Chaim, M., 2022. Understanding the use of spectrum‐based fault localization. Journal of Software: Evolution and Process, p.e2622.

[60] Li, C., Liu, L. and Li, X., 2012, January. Software networks of java class and application in fault localization. In 2012 Second International Conference on Intelligent System Design and Engineering Application (pp. 1117-1120). IEEE.

[61] Sinha, S., Shah, H., Görg, C., Jiang, S., Kim, M. and Harrold, M.J., 2009, July. Fault localization and repair for Java runtime exceptions. In Proceedings of the eighteenth international symposium on Software testing and analysis (pp. 153-164).

[62] de Souza, H.A., Chaim, M.L. and Kon, F., 2016. Spectrum-based software fault localization: A survey of techniques, advances, and challenges. arxiv preprint arxiv:1607.04347.

[63] Keller, F., Grunske, L., Heiden, S., Filieri, A., van Hoorn, A. and Lo, D., 2017, July. A critical evaluation of spectrum-based fault localization techniques on a large-scale software system. In 2017 IEEE International Conference on Software Quality, Reliability and Security (QRS) (pp. 114-125). IEEE.

[64] Heiden, S., Grunske, L., Kehrer, T., Keller, F., Van Hoorn, A., Filieri, A. and Lo, D., 2019. An evaluation of pure spectrum‐based fault localization techniques for large‐scale software systems. Software: Practice and Experience, 49(8), pp.1197-1224.

[65] Zhang, S. and Zhao, J., 2007. Locating faults in AspectJ programs. Technical Report SJTU-CSE-TR-07-03, Center for Software Engineering, SJTU.

[66] Gabor, U.T., 2021. Software fault injection and localization in embedded systems..

[67] Bartocci, E., Ferrère, T., Manjunath, N. and Ničković, D., 2018, April. Localizing faults in Simulink/Stateflow models with STL. In Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week) (pp. 197-206).

[68] Sinha, V.S., Mani, S. and Mukherjee, D., 2012, October. Is text search an effective approach for fault localization: a practitioners perspective. In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity (pp. 159-158).