*Grenze International Journal of Engineering and Technology, June Issue*

# Analysis of Software Bug Localization Models from A Statistical Perspective using Machine Learning

Darshana Gajbhiye[1], Vishal Yadav[2], Shushant Walunj[3], Samarth Bhagade[4] and Rohit Lamkhade[5]
[1-5]Department of Information Technology VPPCOE&VA, University of Mumbai
Email: darshana.tambe@pvppcoe.ac.in, vu4f2021058@pvppcoe.ac.in, vu4f2021014@pvppcoe.ac.in, vu4f2021046@pvppcoe.ac.in, vu4f2021023@pvppcoe.ac.in

***Abstract*— In the ever-evolving digital world, software dependability is a must. With a commitment to software quality assurance excellence, the Bug Localization Project hopes to strengthen and certify the digital ecosystem. The Bug Localization Project is a key component in the continuous attempt to achieve higher software quality in the dynamic field of software development. The urgent need for effective bug localization and identification is what this project is meant to solve. Our goal is to accurately identify, classify, and record shortcomings in software using state-of-the-art techniques and tools. This will enable much quicker problem solving, lower development costs, and ultimately higher user happiness. The primary objective of this program is to expedite the process of localizing bugs. Our goal is to reduce software bug-related disruptions and downtime by improving the accuracy and efficiency of issue detection and reporting. Our group works closely alongside one another all the time to eliminate software flaws. We propose a seamless experience for software users through careful bug localization. Specifically, to increase the accuracy of the insect localization job, we suggested a machine learning model that incorporates methods such as CNN, TF-IDF, Bayesian belief network, deep learning, information retrieval techniques, and Abstract Syntax Trees (ASTs) that can be trained on datasets.**

***Index Terms*— Bug Localization, Software testing, Accuracy, CNN, Deep Learning, Bayesian Belief, Comparative analysis, Machine Learning**

## I. INTRODUCTION

In the ever-changing software development business, detecting and fixing bugs is essential for reliable systems. Complex codebases have long made bug localization difficult for software programmers. Modern automated bug localization systems use cutting-edge methods and technologies to speed up debugging for researchers and industry professionals. Manual bug localization is time-consuming and error-prone; people must find the cause. Automatic bug localization solutions use machine learning, data mining, and other methods to analyze source code, prior problem reports, and other software artifacts. These technologies precisely identify program issues to reduce debugging and human interaction. This document discusses bug localization system developments, methods, and challenges. We examine how automation can handle software projects' increased complexity and the necessity for precise, effective, and scalable bug localization. We intend to evaluate state-of-the-art methodologies and their practical implementations to help researchers, software developers, and industry professionals choose innovative bug localization systems. Automatic bug localization solutions improve code quality, speed up software

*Grenze ID: 01.GIJET.10.2.506_3*

development, and make software more stable and durable, according to this investigation. We also address the drawbacks, moral issues, and future prospects of this discipline, laying the groundwork for understanding how automated methods may completely revolutionize how we fix & localize software bugs.

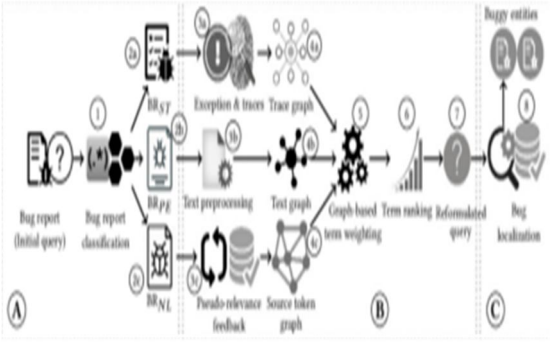


Fig. 1. A general bug localization system [1]

## II. Literature Review

A machine learning technique called the NetML algorithm uses data from program spectra and bug reports to build a model that can forecast the possibility that a program element has a bug. It learns the model parameters using a convex optimization technique, and then it ranks the program pieces according to how likely it is that each one has a bug.[1] Seven software systems' worth of program spectra and bug reports made up the data used in this investigation. While the program spectra record the frequency at which each program element is executed throughout the execution of test cases, bug reports detail the symptoms of the bugs. The NetML approach encourages the model parameters of related bug reports and program elements to be near together by using network Lasso regularization. It uses a feature-by-feature adaptive learning process that is based on Newton's approach to update the parameters. A number of metrics, such as Mean Average Precision (MAP), Top-1, Top-5, and Top-10 accuracy, are used to assess the performance of the NetML approach. It is discovered that the NetML approach performs better than a number of cutting-edge bug localization methods.

*Precision*

According to the project's accuracy results, when a developer examines the top 1, 5, and 10 methods and Mean Average Precision (MAP), NetML outperforms the best-performing baseline by 31.82%, 22.35%, 19.72%, and 19.24%, respectively, in terms of the number of bugs that are successfully localized.

The authors of this research [2] introduced a novel strategy to insect localization by combining an information retrieval (IR) technique called rVSM with a deep neural network (DNN). They made use of a benchmark dataset made up of 50,000 source files and 2,500 bug reports from 5 open-source projects. Lexical, syntactic, and semantic aspects were the three categories of features that the authors collected from the bug reports and source files. They then used an autoencoder developed using DNN to project these feature vectors onto a continuous space with reduced dimensions. The feature extraction and classification components make up the two primary parts of the authors' DNN LOC model. While the classification component utilizes rVSM to determine how similar the features in the bug reports and source files are to one another, the feature extraction component uses DNN to extract features from the source files and problem reports. To increase the model's accuracy, the writers additionally included a history of problem fixes for the project. Three measures were used by the authors to assess their DNN LOC model on the benchmark dataset: mean average precision (MAP), top-ranked accuracy, and top-k accuracy. With the exception of the smallest project, AspectJ, which was split into three folds, the model was trained on one-fold and tested on the subsequent folds, totaling ten folds. The DNN LOC model outperformed the most recent IR and machine learning methods in terms of accuracy, according to the authors' comparison. All things considered, the DNN LOC model developed by the authors offers a viable path for further bug localization research. Lexical mismatch is a problem that is addressed by combining DNN and rVSM features with the project's issue-fixing history to increase bug localization accuracy. For use in next studies, the authors have made their benchmark dataset, bug localization tool, and evaluation scripts freely available. Future research in bug localization appears to have a promising trajectory thanks to the authors' approach of merging features from DNN and rVSM with the project's history of bug fixes.

*Precision*

For the Tomcat and AspectJ projects, the authors presented a top-k accuracy comparison between their DNN LOC model and the LR, BL, and NB models. Depending on the value of k, the DNN LOC model for the Tomcat project scored top-k accuracy ranging from 53.9% to 85.9%, while the LR, BL, and NB models achieved top-k accuracy ranging from 5.2% to 80.1%. The LR, BL, and NB models achieved top-k accuracy ranging from 20.2% to 68.2% for the AspectJ project, but the DNN LOC model earned top-k accuracy ranging from 47.8% to 86.2%.

One important duty in software development is bug localization, which entails figuring out which source code files are in charge of a particular issue report. The authors of this study suggest a novel approach to supervised topic modeling-based bug localization. The foundation of the suggested approach is the fundamental finding that source code files and bug reports have a same vocabulary and that it is possible to deduce themes in source code files from those in bug reports.[3] The two primary parts of the suggested approach are a similarity measure and a supervised topic model. Using Gibbs sampling to estimate the model parameters, the supervised topic model is trained using a dataset of bug reports and the source code files that go along with them. Based on the subjects deduced from the model, the similarity measure is then utilized to calculate the similarity between a specific bug report and each source code file. The suggested approach is assessed by the authors using three open-source projects, and it is contrasted with various other information retrieval techniques currently in use for bug localization. The findings demonstrate that, in terms of both efficacy and efficiency, the suggested strategy performs better than the current ones. It is also demonstrated that the suggested approach is scalable and capable of managing big datasets containing thousands of source code files and bug reports. All things considered, the suggested technique offers a viable approach to bug localization and has the potential to drastically cut down on the time and effort needed by software developers to find and address defects.

*Precision*

The trials demonstrate that the suggested approach, with an average Hit@1 score of 0.54, an average Hit@5 score of 0.83, and an average MRR score of 0.63 across the three datasets, achieves high accuracy in locating the pertinent source code files for a particular bug report.

CAST processes bug reports and source files into two sections for word embedding, data preprocessing, and feature extraction before combining them into a feature combination. Lastly, CAST ranks the provided source files after feeding fusion features into the fully-connected network. Pairs of source code files, bug reports, and their labels are sent into the CAST network during the training phase, where iterative training is used to optimize the loss function. During the testing stage, a fresh bug report and its candidate source files are provided to the trained network, which then produces the source files' scores. Every score denotes the degree of correspondence between a source code file and the specified bug.[4] Multiple metrics, including Accuracy@1,5,10,15, MAP, and MRR, are presented in the paper along with a performance comparison of CAST against four cutting-edge bug localization algorithms on four popular Java projects. Based on the testing results, CAST performs better on most of the projects than the four competitors in terms of accuracy, MAP, and MRR. Nevertheless, the study does not offer a single, percentage-based accuracy statistic for the CAST system. The study also reviews relevant work in the topic of bug localization, encompassing deep learning, machine learning, and traditional information retrieval techniques. The authors demonstrate how CAST performs better than several of these methods in terms of efficiency and accuracy. The paper concludes with a novel approach to bug localization that makes use of customized abstract syntax trees and deep learning. According to the trial results, CAST performs more accurately and efficiently than cutting-edge methods. CAST may make it easier for developers to find potentially problematic source files and swiftly fix them, which would enhance software quality and save on expenses and development time.

*Precision*

Rather, it displays other metrics including MRR, MAP, Accuracy@1,5,10, and 15.

One paper based on CNN was studied for the survey which suggests a unique deep transfer learning method for cross-project bug localisation. To discover defects in target projects, TRANP-CNN, a suggested approach, extracts transferable semantic properties from source projects. The authors point out that when trained on data from other projects, current supervised bug localization methods cannot achieve good results. In order to overcome this constraint, they suggest a deep transfer learning model that produces project-specific predictions and learns transferable latent features that are shared by the source and target projects. This model enables supervised knowledge transfer from the source project to the target project. The authors test the suggested method on 12 tasks with 10 open-source projects, and they demonstrate that, when all evaluation metrics are taken into account,

TRANP-CNN performs better than earlier cutting-edge bug localization techniques on all 12 tasks. Additionally, they emphasize how well the project-specific prediction layer and the transferable feature extraction layer—two essential TRANP-CNN components—work. [5] As training data, the suggested method needs a compilation of bug reports along with the pertinent source code files. A good model that can associate fresh bug reports with the appropriate source code files is then trained using this data. A supervised approach's requirement for an adequate supply of excellent training data is one of its drawbacks. Its efficacy may be negatively impacted by incomplete or poor-quality data. This issue is especially significant when applying a bug localization strategy to new projects with little experience addressing bugs. The essential component of the suggested method put out by the authors is a convolutional neural network (CNN). To represent the linguistic data in bug reports and source code files, they employ pre-trained word embeddings. In the course of training, they additionally optimize using the Adadelta method. In order to enable supervised knowledge transfer from the source project to the target project, the suggested method, TRANP-CNN, is a deep transfer learning model that learns transferable latent features shared by both source and target projects. It then delivers project-specific prediction. It makes use of the Adadelta method for optimization during training and a CNN with pre-trained word embeddings. This approach has demonstrated promising results in cross-project bug localization; however, it requires a collection of bug reports and their corresponding source code files as training data.

For the bug localization challenge, one research suggests a supervised topic modeling technique called L2SS. [6] The suggested approach treats the source filenames as the supervision information and makes use of previously released problem fixes. When developers only have access to the source filenames, L2SS can be used in certain situations. Additionally, the authors examine the value of several information types found in the bug report and update L2SS to include these metadata (referred to as L2SS+). For instance, the L2SS+CM technique is used when component metadata is included. The suggested approach for bug localization based on bug reports is based on static analysis. Four open-source projects were used for the trials, and data was gathered by looking through the git logs for relevant altered files that contained a bug report ID. The efficacy and efficiency of the suggested solutions are demonstrated by the evaluations conducted on multiple real datasets. L2SS+CM, for instance, can outperform its best current competitors by up to 16.9%. The suggested approach differs from the current approaches by computing similarity and adhering to the IR framework. An exception can be found in Kim et al. In order to forecast the possible problematic files for a bug report, they suggested approaching the issue as a classification problem (many classes) and applying Naive Bayes. Using a bag-of-words, they were able to extract features from the bug description and information. The authors of this paper use supervised modeling to localize bugs. In contrast to Kim et al., they suggest a unique approach for every form of metadata and continue to employ topic modeling based on the bag-of-words. The supervised approach makes the supposition that the developers adhere to naming standards and that the names have significance. The similarity calculation between the source file and the problem report may be worthless in the event that developers disregard naming rules. Ye et al. and Almhana et al. suggested introducing API documentation of classes and methods in order to achieve this. In subsequent work, the authors might think about include API documentation in their methodology. To enhance the precision of bug localization, the supervised topic modeling technique L2SS+X, as suggested, integrates information into the process. Four open-source projects were used for the trials, and data was gathered by looking through the git logs for relevant altered files that contained a bug report ID. The efficacy and efficiency of the suggested solutions are demonstrated by the evaluations conducted on multiple real datasets. The suggested approach differs from the current techniques that adhere to the IR.

Accuracy: The efficacy and efficiency of the suggested methods are demonstrated by the evaluation of the proposed method L2SS+X on a number of real datasets. L2SS+CM, for instance, can outperform its best current competitors by up to 16.9%.

In a particular version of a software project, the authors suggest a framework for calculating a prior probability distribution for the defect proneness of the files. Using both sorts of histories as saved in the versioning tools, they offer two models to estimate the priors: one from the modification histories and the other from the defect histories. To calculate the posterior probability that a file is the source of a defect, the authors integrate the priors into an information retrieval (IR) framework. With a hitrate of 63.5% and a P@10 of 0.1065, the authors demonstrate a notable improvement over the BugScout tool in their evaluation of their approach on the AspectJ project. Additionally, the authors find that when temporal decay is included in the estimate of priors, their method enhances the mean average precision for problem localization by as much as 80%.[7] The authors propose various avenues for further development of their work in the future, such as verifying their bug localization method with alternative software repositories, investigating alternative retrieval algorithms, and creating a program that applies their method. The authors also note that the BugScout tool uses Latent Dirichlet Allocation (LDA), upon which their

method is based. In summary, this work offers a viable method for enhancing the precision of IR-based problem localisation through the utilization of software project version histories. The authors show that mean average precision for bug localization is significantly improved when temporal decay is included in the prior estimation process. Other retrieval strategies might be investigated in future research, and the method could be tested with different software repositories.

## III. Proposed Methodology

Bug localization requires several critical components to increase accuracy and efficiency. From Fig. 2 Initial data collection and preprocessing include bug reports, source code repositories, and crash reports. Text data is preprocessed by removing stop words, stemming, and tokenizing, while bug reports and version histories provide structured data. To capture semantic meaning in bug reports and source code, CNNs or RNNs are trained on preprocessed text data to learn feature representations. Using labeled data, supervised learning algorithms like SVM or decision trees must categorize bug reports and associate them with relevant code files. Combining information retrieval, deep learning, and supervised learning improves bug localization. Crash reports are connected to detect common crash-related bugs, and algorithms prioritize bug reports and important code parts. An overview of the techniques proposed are as follows:

*Deep Learning:* A subset of machine learning techniques called "deep learning" is based on representation learning in artificial neural networks. The usage of numerous layers in the network is indicated by the adjective "deep" in deep learning. The employed techniques can be unsupervised, semi-supervised, or supervised. [8]

*Supervised Machine Learning:* A machine learning paradigm called supervised learning (SL) trains a model using input objects (such a vector of predictor variables) and a desired output value, which is also referred to as a human-labeled supervisory signal. After processing the training set of data, a function mapping fresh data to anticipated output values is created. [9] The algorithm will be able to determine output values for unseen occurrences appropriately in an ideal case. This means that the learning system must make "reasonable" generalizations from the training data to unknown scenarios (see inductive bias). The so-called generalization error is used to gauge an algorithm's statistical performance.

*Information Retrieval*: In computers and information science, information retrieval (IR) is the act of selecting resources from a collection of information system resources that are pertinent to a particular information demand. Full-text or another content-based indexing method may be the basis for a search. The science of searching for information within a document, for documents themselves, for the metadata that describes data, and for text, image, or sound databases is known as information retrieval [10].

One way to lessen what has been referred to as information overload is to use automated information retrieval systems. An information retrieval system (IR) is a software that stores and organizes data in addition to granting users access to books, journals, and other types of material. IR applications that are most noticeable are web search engines.

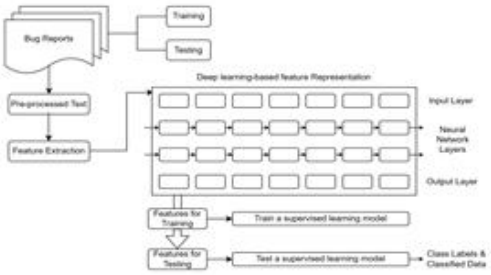


Fig. 2. Proposed Architecture

## IV. Conclusion

In conclusion, a bug localization project that seamlessly integrates deep learning, supervised learning, information retrieval, and crash report correlation could solve modern software development's complex bug identification and resolution problems. This experiment showed the ability to improve problem localization precision and recall, lowering the time and effort needed to find and resolve software defects. Deep learning extracts complex patterns and relationships from code and bug reports, while supervised learning guides localization. Information retrieval

methods leverage software repositories' rich history data, making bug localization more context-aware. Crash report correlation gives us a new perspective to prioritize and validate problem reports. This study shows how data-driven and trans disciplinary bug localization methods improve software maintenance and user experiences. The success of this project highlights the need for continued study and development in this field to improve bug localization for software developers and end-users. These integrated strategies help improve software maintenance agility and reliability as software systems become more sophisticated, ensuring that software issues are quickly recognized and fixed.

## FUTURE WORK

The suggested bug localization project, which blends deep learning, supervised learning, information retrieval, and crash report correlation, has significant promise. First, the project can improve deep learning models by studying new neural network designs and using cutting-edge natural language processing to analyze code and bug reports. Transfer learning and multi-modal input sources can help the system adapt to different software settings. Additionally, fine-tuning supervised learning algorithms and extending the dataset helps improve insect categorization and model generalization. The project can employ advanced indexing, semantic search, and recommendation algorithms to improve bug localization. Finally, crash report correlations enable predictive bug localization and proactive issue solutions. By improving and expanding these features, the project can help bug localization, lowering software defects and improving software quality for developers and consumers and try to locate bugs with 99% accuracy.

## REFERENCES


[1] Hoang, Thong, et al. "Network-clustered multi-modal bug localization." IEEE Transactions on Software Engineering 45.10 (2018): 1002-1023.
[2] Lam, An Ngoc, et al. "Bug localization with combination of deep learning and information retrieval." 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). IEEE, 2017.
[3] Wang, Yaojing, et al. "Bug localization via supervised topic modeling." 2018 IEEE international conference on data mining (ICDM). IEEE, 2018.
[4] Liang, Hongliang, et al. "Deep learning with customized abstract syntax tree for bug localization." IEEE Access 7 (2019): 116309-116320.
[5] Huo, Xuan, et al. "Deep transfer bug localization." IEEE Transactions on software engineering 47.7 (2019): 1368-1380.
[6] Zhang, Xiaofei, et al. "Exploring metadata in bug reports for bug localization." 2017 24th Asia-Pacific Software Engineering Conference (APSEC). IEEE, 2017.
[7] Sisman, Bunyamin, and Avinash C. Kak. "Incorporating version histories in information retrieval based bug localization." 2012 9th IEEE working conference on mining software repositories (MSR). IEEE, 2012.
[8] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436-444.
[9] Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). Foundations of machine learning. MIT press.
[10] Luk, R. W. (2022). Why is Information Retrieval a Scientific Discipline?. Foundations of Science, 1-27.