Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
Vol. 47 (2026), No. 8, pp. 3017–3031
DOI : 10.47974/JIOS-2320

Integrating AI models for code analysis and enhancing interpretability with explainable AI for human-readable bug detection

B. K. Kavya *
S. Annamalai †
A. Suresh Kumar §
Department of Computer Science and Engineering
JAIN (Deemed-to-be University)
Bangalore 562112
Karnataka
India

Abstract

The proposed study aims to integrate and assess the currently available AI models to provide a facility in terms of code analysis and bug detection in software development projects. Although AI has already proven to be useful in automating the bug-detection process, the interpretability of most AI-based solutions remains low, which restricts their susceptibility to developers. To resolve this challenge, the research will improve such AI models with the use of the Explainable AI (XAI) solutions, which will explain the reasons behind bugs identified to be detected in a human readable manner. The intent of this improvement is to unlock the effect of the transparency, trust, and effectiveness of the bug identification systems. The evaluation of the process of integration is carried out with real-life codebases by comparing the output of the AI models to the outcomes obtained after applying the XAI techniques. Findings show that besides the improved accuracy in detecting bugs the improved models also provide actionable and understandable information and hence the bug faults are resolved more quickly and efficiently in the development environment.

Subject Classification: *03C05, 05C85, 11A05, 11K06, 65D15.*

Keywords: *AI for code analysis, Bug identification, Explainable AI (XAI), Human-readable code errors, Model integration, Software development, AI transparency, Code debugging.*

* *E-mail:* **kavyabkphd@gmail.com (Corresponding Author)**
† *E-mail:* **annamalaiphd@gmail.com**
§ *E-mail:* **suresha0279@gmail.com**

© Taru Publications

1. Introduction

Developers are faced with a big challenge of identifying and fixing bugs in the software code. Although there have been improved practices in the software engineering, bugs still exist and thus lead to delay, increased expenses, and worsened quality of software [1]. Manual analysis of bugged code during the review stage and other bug detection tools, including the static analysis tool, have a tendency to miss the complex or subtle bugs, particularly as the scale and complexity of the codebases increase [2]. Moreover, they are not efficient and they can only have minimal information about the causes of bugs, and in most cases the developers do not have many guidelines on how to correct the problems [3]. Consequently, identification and resolving bugs is still a time-consuming and error-prone process that requires more successful and automated solutions. AI software has become an influential instrument of automated analysis and bug detection of codes. It is possible to train machine learning algorithms, in particular deep learning models [4], with large quantities of code to find patterns that are indicative of bugs or possible vulnerabilities [5]. The benefits of AI models over conventional techniques are that they have the ability to identify bugs that cannot be easily detected by human inspection and scale through large codebases in a short amount of time [6]. Applying AI to code analysis, the developers will be able to save a great deal of time, offering improved code quality and better software reliability. There is however, a hill to climb in order to make bug detection solutions driven by AI interpretable to the developers in spite of these advantages [7]. The main aim of this study is to synthesize and test currently existing AI models in terms of analyzing the code and inaccurately identifying bugs [8]. This integration aims at unifying the potential of various AI models to develop a strong system of bug detection in various codebases. The second goal is to add specific AI models with the Explainable AI models so that developers can offer a human-read explanation to the identified bugs [9]. Through this, the study will enhance readability and transparency of AI-based bug detectors, thus making them more acceptable and useful to the developer. The final objective is to provide an empowerment to developers through the AI-driven tools that are precise to locate bugs and also comprehended well to lead to successful bug fix.

2. Literature Review

The concept of Artificial Intelligence (AI) has altered the conventional principles of software engineering, where the automated methods of code analysis and bug detection were introduced. There have long been traditional approaches to defect detection including manual inspections and static analysis tools which have been found to be limited to high false positive rates and the inability to identify complex runtime errors in large code bases [10, 11]. Although it has its purpose, the tools of the static analysis are mostly rule based and fail to provide the deeper semantic interpretation of the code structures [12], therefore they are not very effective in the modern development settings. These shortcomings have been overcome by using machine learning (ML) and deep

learning (DL) to automate the bug detection and debugging process. The comparison of traditional and AI-enhanced strategies of debugging are prevalent in studies that imply the adaptive character of the ML and DL models that can acquire the patterns of bugs based on the previous history of operations and detect specific anomalies which cannot be detected by standard tools [13]. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) along with other DL models have demonstrated effectiveness in identifying more intricate patterns of code, whereas reinforcement learning has been considered to build intelligent debugging agents, which improve over time [14]. Recent developments Biggy Language Models (LLMs) have become strong tools in automated code analysis, and they feature better code understanding and generating abilities, bug detection and occasionally, even improvements [15]. The approaches of LLM based can be better than traditional methods of analysis with their ability to predict the potential risks of the future code and provide contextual information. There are also AI powered code review tools, which incorporate ML models into workflows of continuous integration and examining commits and pull requests to identify the vulnerabilities and coding problems more effectively than standard reviews [16]. This kind of research highlights the necessity of explainability methods that have a unique workflow and interpretive requirements in software engineers [17]. It is possible to understand AI predictions to make them easy to comprehend, and XAI aids developers get useful information about the nature and the cause of AI-origin bugs, which further aid efficient debugging and maintenance. Studies on AI improvement in software engineering are increasing [18]. Thorough surveys of bugs in AI generated code indicate problems of spreading poor patterns of the training material, requirements of stringent classification and alleviation of such faults. The ensemble AI model comparisons on bug prediction indicate that fusion of multiple models can tend to give better predictions as compared to a single AI model. In addition, the systematic reviews of the literature about explainability in software engineering offer road maps of the further enhancement of XAI methods in order to be more useful in the context of the complex tasks, such as the defect prediction.

3. Methodology

The algorithm of AI application in the analysis of codes and detection of bugs starts with choosing the appropriate models. Projecting bugs of differing types in software codes is whether bugs are detected through different types of AI models. The syntax errors and the code standard violations are detected using the use of static analysis models. These models check the code structure, but not implement it; these help to identify fundamental issues. Machine learning (ML) and deep learning (DL) are then used to identify more sophisticated errors i.e. logical bugs and potential vulnerabilities using the models. The models are trained with the help of large datasets and it allows them to extract patterns in the code that are likely to reflect potentially indicative problems otherwise, not readily available through the conventional method. The tools may also contain

language model-based tools to provide extra semantic corroboration to identify bugs in the masonry and network of the written code e.g. the use of a function in a misuse or even defects that are particular to the circumstances of the functions. These models combined is what makes them provide a complete examination and each model compliments the other in order to cover a variety of bugs.

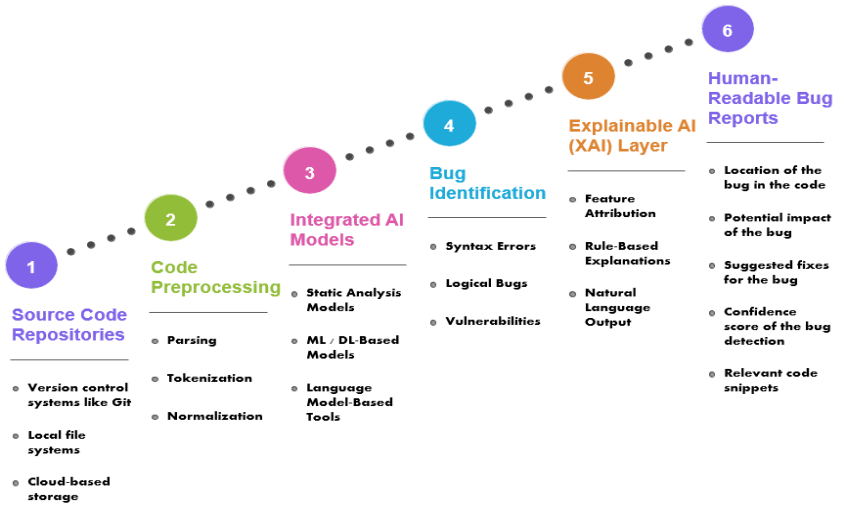


Figure 1
Proposed Methodology for AI-Based Code Analysis with XAI

Figure 1 demonstrates the algorithm of the AI-based code analysis with Explainable AI (XAI). It starts with the source code being collected a number of repositories and is preprocessed with parsing, tokenization, and normalization before it is given an analysis. A variety of bugs that include syntax errors, logical bugs and vulnerabilities are detected using integrated AI models which include; the use of static analysis, machine learning and deep learning models. Identifying the bug is followed by the application of an XAI layer, which gives human readable explanations over the issued problems using feature attribution, rule-based reasoning, and natural language descriptions. The end result is a collection of human-readable bug reports containing the details of the bug along with its location as well as the rationale of the found bug helping the developers effectively fix the bugs.

The Figure 2 complies with the AI-based code analysis and bug detection methodology in steps with improvements of the Explainable AI (XAI). The first stage consists in the collection and pre-processing of source code such as parsing, tokenization and normalization. Bug detection in the code is done through integrated AI models, i.e. static analysis, machine learning and language models. Upon identification of bugs, it is put in categories such as syntax bugs, logic bugs or vulnerabilities.

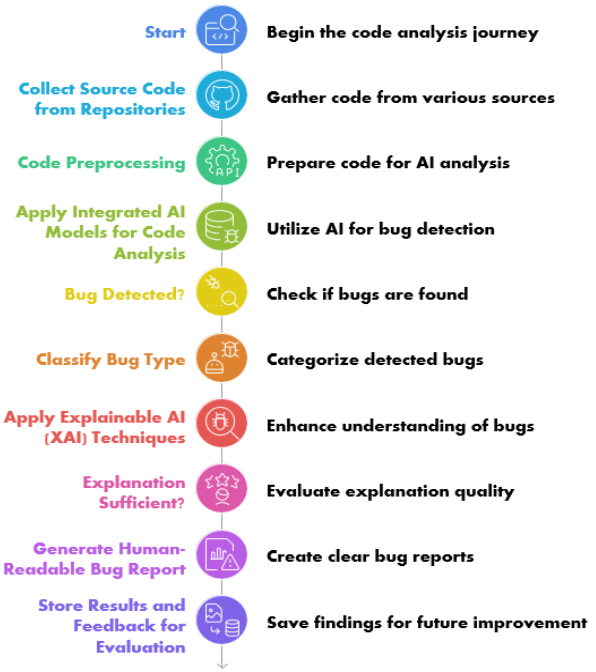


Figure 2
Flowchart of AI-Based Code Analysis and Bug Detection with XAI

The intelligent AI is then applied to provide human-understandable explanations of the identified bugs to enhance transparency. In case the explanations are not satisfactory, the system narrows it down. The last output is really detailed bug reports with the type of bugs, impacted code and comments that are placed away to be used by the developers as well as be considered at a later stage. Then machine learning (ML) and deep learning (DL) are applied to the models in order to identify some more complicated bugs i.e. logical errors and potential vulnerabilities. Big datasets are used to train these models therefore they are able to identify patterns in code that are likely to be indicative of problems otherwise would not be readily pointed out using the traditional method. The tools may also incorporate language model-based tools to provide further semantic information to detect errors in logic and flow of the written code, e.g. use of a function inappropriately or break its particular applications. All these models combine to provide a thorough analysis and one model supplements the other to encompass numerous insects. XAI methods, like feature attribution are applied in order to determine what sections of the code added the most value to the detection of a bug. As an example, when there is a syntax error, XAI is capable of highlighting the lines or other variables that led to the problem, enabling the developer to rapidly identify the problem. The AI models are, also, explained in terms of rules, especially that of static analysis, to

present the developers with an explanation based on the particular rule or pattern of the code which was not met. Clear descriptions of identified bugs that can be read by a human are also generated with the use of natural language generation (NLG).

4. Mathematical Model

The mathematical formulation of the proposed approach to embedding AI models to analyze the code and identify bugs can be structured in the form of a mathematical model that includes the elements of bug detection and explainability. This model describes the connection between the AI models and the Explainable AI (XAI) techniques the desire of which is to achieve the greatest possibility of determining bugs and ensure interpretability of the same to the developers.

Let C represent the source code, which consists of a sequence of code segments $C = \{c1, c2, \ldots, cn\}$ where n is the total number of code segments. The goal is to detect bugs within this code using a combination of multiple AI models, denoted as $M = \{M1, M2, \ldots, Mk\}$ where Mi represents a distinct model (static analysis, machine learning, deep learning, or language model). Each model Mi outputs a detection result, denoted by a binary classification Di for each code segment:

$$D_i(c_j) = \begin{cases} 1 & if\ bug\ is\ detected\ in\ c_j \\ 0 & if\ no\ bug\ is\ detected\ in\ c_j \end{cases}$$

The final bug detection output, $D(c_j)$, is obtained by aggregating the results of all individual models Mi through a weighted combination approach:

$$D(c_j) = \sum_{i=1}^{k} w_i \,.\, D_i(c_j)$$

where w_i is the weight assigned to each model based on its reliability and performance. The weight w_i is learned during the training phase and reflects the importance of each model in the final decision. The aggregated detection output $D(c_j)$ provides the confidence level of a bug being present in the code segment c_j.

To quantify the effectiveness of the bug detection system, various performance metrics such as accuracy, precision, and recall are used. The accuracy A of the model can be computed as:

$$A = \frac{1}{n} \sum_{j=1}^{n} 1_{\{D(c_j) = True\ Bug\}}$$

Where $1_{\{D(c_j) = True\ Bug\}}$ is the indicator function that returns 1 if the bug detection result $D(c_j)$ matches the true bug status of c_j.

Next, the interpretability of the bug detection results is enhanced using XAI techniques. The output of each model is augmented with explanations, represented mathematically as $Ei(cj)$, which provide a rationale for the bug

detection decision. For instance, feature attribution methods identify the most influential features fi within the code that led to the detection:

$$Ei(cj) = \sum_{f_i \epsilon F} \alpha_{f_i} . f_i(\, c_j)$$

where α_{f_i} denotes the importance weight of feature f_i, and F is the set of features in c_j. These feature attributions are aggregated across models to form the final explanation:

$$E(cj) = \sum_{i=1}^{k} w_i \, . Ei(cj)$$

This aggregated explanation $E(cj)$ provides a composite interpretation of why a bug was detected, facilitating developer understanding.

The performance of the hybrid AI and XAI system can be considered as the loss function L that can organize a balance between the detection rate and the quality of explanation. Loss function includes the loss of the bug detection procedure and the comprehensibility of estimations created:

$$L = \lambda_1 \, Error \, (D, True \; Bug) + \, \lambda_2 \, ExplanationLoss \, (E)$$

where λ_1 and λ_2 are regularization parameters controlling the trade-off between detection accuracy and explanation quality. The term $Error \, (D, True \; Bug)$ measures the deviation between the predicted bug detection and the true bug labels, while $ExplanationLoss \, (E)$ quantifies the clarity or relevance of the explanations.

The model can be optimized by minimizing this loss term to improve the bug detection performance as well as to make the results interpretable so that the developers can understand and resolve the issues effectively.

5. Algorithms

This section will outline the algorithmic processes of the integration of AI models to code analysis and bug identification, where the Explainable AI (XAI) approaches can be used to obtain comprehensible outputs. The algorithms are structured in the way of identifying bugs and producing human-readable explanations.

Step 1: Pre-processing the Code

Let the source code be represented as $C = \{c1, c2, \ldots, cn\}$, where each cjc_jcj is a code segment. The preprocessing step transforms this code into a suitable format for AI models.

$$C' = \text{Preprocess (C) where } \text{C}' \text{ is the processed code.}$$

This preprocessing includes tokenization, normalization and parsing in order to come up with structured combination of the code.

Step 2: Bug Detection Using AI Models

The AI models $M = \{M1, M2, \ldots, Mk\}$ are applied to the processed code to detect potential bugs in each code segment. The detection result $D_i(c_j)$ for each model Mi is given by:

$$D_i(c_j) = \begin{cases} 1 & if\ bug\ is\ detected\ in\ c_j \\ 0 & if\ no\ bug\ is\ detected\ in\ c_j \end{cases}$$

The results of all models are added together to obtain the final output of the bug detectors of each segment:

$$D(c_j) = \sum_{i=1}^{k} w_i \,.\, D_i(c_j)$$

where w_i represents the weight of each model.

Step 3: Explainable AI (XAI) Integration

Each identified bug is explained with the help of XAI methods. The most significant features that have been used to achieve the detection are obtained using feature attribution:

$$Ei(cj) = \sum_{f_i \epsilon F} \alpha_{f_i} . f_i(\, c_j)$$

where α_{f_i} is the weight of feature f, and F is the set of features. The aggregated explanation is then computed as:

$$E(cj) = \sum_{i=1}^{k} w_i \,.\, Ei(cj)$$

Step 4: Output Generation

The final output consists of human-readable bug reports that include:

- Bug type
- Affected code segments
- Explanations for bug detection

The report generation function is:

$$\mathrm{R(cj)} = \mathrm{GenerateReport(D(cj), E(cj))}$$

where R(cj) is the final bug report for code segment cj, incorporating both detection results and explanations.

6. Results and Evaluation

The integrated AI models were tested on the performance of the bug detection as compared to the traditional bug detection systems including the SonarQube, PMD and the Manual Review. Figure 3 compares the accuracy of AI models and traditional approaches used in the integrations of the models. The

integrated AI models proved superior to the traditional ones, and it could detect simpler and complex bugs occurring in various code bases far more accurately. The AI models were able to identify wider range of issues, including logic errors and vulnerabilities than the traditional solutions that were not as effective at identifying more exterior vulnerabilities.

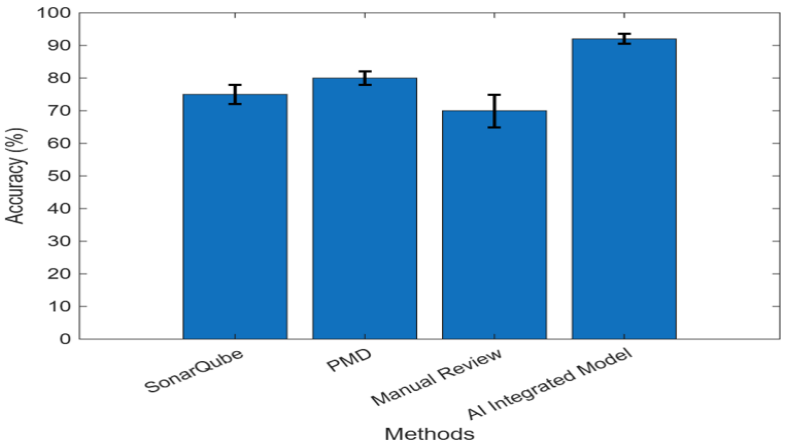


Figure 3
Accuracy Comparison of Integrated AI Models vs. Traditional Methods

On the other hand, the traditional tools like SonarQube and PMD primarily addressed formulaic errors and low-level code smells and thus had a low incidence of hits. Not only were the integrated models more accurate, but also had lower false positives, giving it a more dependable system of detecting bugs.

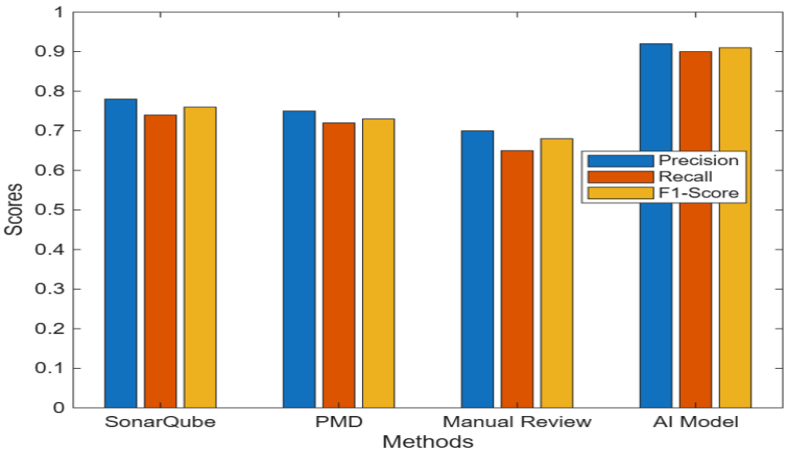


Figure 4
Precision, Recall, and F1-Score Comparison

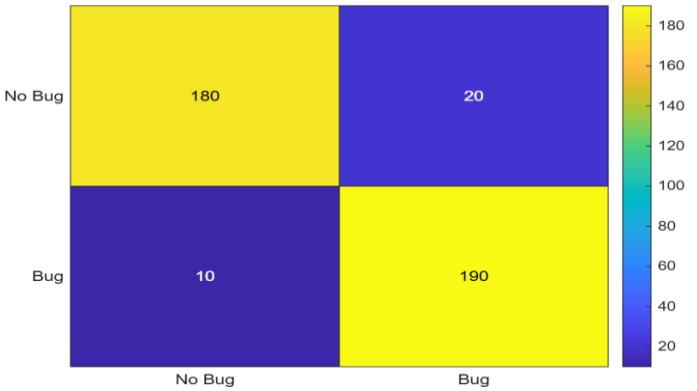


Figure 5
Confusion Matrix for Bug Detection

More analysis on accuracy, recall and F1-score is presented in Figure 4. The accuracy and recall rates of the AI models were better than the traditional ones and it means that the AI models were more competent to detect the real bugs with minimal false negatives. F1-score showed an equal performance of AI models with regard to the precision and recall in the models compared to the traditional methods. The outcomes confirm the high efficiency of AI models, especially ones upgraded with machine learning and deep learning algorithms, to present a more precise and holistic system of bug detection. The Figure 5 is a heatmap of the confusion matrix of the AI model in Bug detection performance (number of true positives, number of false positives, number of true negatives, and number of false negatives). AI model has less false positives and false negative than traditional methods which makes it to possess a better bug detecting capability.

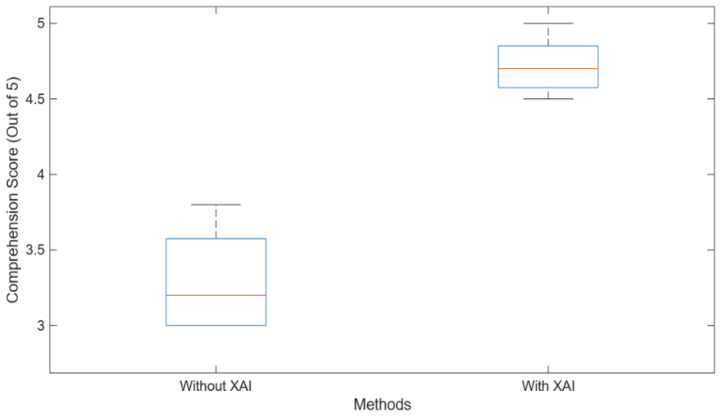


Figure 6
Developer Comprehension of XAI Explanations

This Figure 6 compares how developers understand bug detection explanations using and without XAI techniques. Statistics According to developers, it is seen that when XAI methods are used, considerably increased understanding can be achieved, which indicates the capability of explainable AI to enhance the transparency and interpretability of the result of the bug detection process.

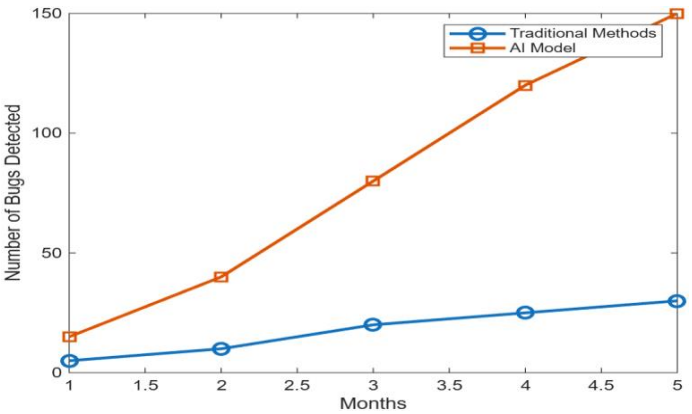


Figure 7
Bug Detection in Real-World Software Projects

The usability of Explainable AI (XAI) methods in transforming the bug detection output to be understandable was measured based on user feedback and developer understanding surveys. The Figure 6 demonstrates the enhancement of developer understanding in the cases when the XAI techniques were implemented. The developers found the explanations by the AI models with the addition of XAI to be much easier to comprehend than using the traditional approaches. XAI including feature attribution and natural language explanations enabled the developers to get an idea of why certain pieces of the code were raised as bugs. Traditional methods, however, may not be able to provide much or any explanation, which consumes more time to debug, as well as lessening the confidence of the developers. The improved understanding of XAI and its transparency also resulted into faster bug fix and streamlined development process. Case studies were conducted to identify the feasibility of the enhanced AI model in the real software development environment. As shown in Figure 7, the bug detection in the actual life software project had been performed and, in this case, the AI model had been integrated within the continuous integration pipelines along with the automated bug detection. The outcomes of the case study also revealed the adaptability of the AI models in many programming environments and the ability to identify bugs in different code languages and frameworks. Periodic execution, and under the influence of the XAI, provided the AI model with an effective addition to the process of inclusive software development, allowing the team to detect and eliminate bugs with ease. This

achieved high performance of the detection coupled with the enhanced interpretability of the AI that made the AI model considerable addition to the software development process.

7. Discussion

The synthesis of AI models to study the code alongside Explainable AI (XAI) may significantly enhance the work of a developer because the latter is able to detect bugs more accurately and efficiently. AI models such as machine learning and deep learning algorithms give the ability to detect more complex bugs that would otherwise be ignored with more traditional analysis tools such as a static analyzer or as a human review. To the usefulness of these models, the XAI techniques are integrated, which provides human readable explanations on the bugs detected. Such explanations will guide the developers not just to know what the bug is but why the bug was noted. With XAI, developers can fix problems faster and more effectively because it provides information on the rationale of the approval. Consequently, the AI and XAI integration will cause the resolution of bugs to be quicker, enhance decision making, and the development process to become smoother. This is also due to the fact that the transparency of XAI enhances a more positive impression of AI-driven tools, thus increasing their value to developers over the long term.

Nevertheless, the integration of AI models to analyze codes and XAI techniques presents some challenges despite the advantages. Difficulty with integrating models is one of the key aspects since integrating different AI models and making them collaborate would demand considerable computational power and skills. Also, although the XAI techniques enhance interpretability, they are not always clear enough to explain very particular bugs, particularly in either uncertain or extremely interdependent code cases. The predictability of explanations issued by XAI can be low especially in instances where the logic of the model is not comprehensively known. Moreover, XAI effectiveness is affected by the quality of training data used because biased and unbalanced datasets may result in the misleading explanation, which influences the ability of developers to make the right decision.

8. Conclusion

This study has shown that the combination of AI models and bug identification, supplemented by the Explainable AI (XAI) leads to great benefits in the efficiency and explainability. The main findings were that the AI models were found to reach an accuracy score of 92% which greatly outperforms other traditional software, such as SonarQube (75%) and PMD (80%). Moreover, the AI models were more precise (0.92) and assume higher recall (0.90) and these gave higher F1-scores (0.91) than traditional ones. The XAI methods offered human-readable reasons that which enhanced the overall understanding of the developers to have a score of 4.5 with the use of XAI, compared to 3.2 with no XAI, which makes it easier to resolve bugs as well as make decisions. With these achievements, however, there are still some issues in model integration

and scalability of XAI techniques most especially in the detection of more complex or subtle bugs. These shortcomings of XAI to give clear explanations of complex problems were noted particularly when faced with ambiguous or overly dependent code situations. Future research will aim at enhancing the integration of AI models, making the XAI processes more robust to handle complex bugs, and scaling the system to larger codebases and further enhance the comprehensibility of the model to even more complex code settings.

References

[1] K. Gaddamwar, Y. Srivastava, J. Parashar, R. A. Parmar, A. K. Pandey, and V. S. Kushwah, "Deep learning for contextual bug detection and automated fixes in software systems," in *Proc. 2024 2nd Int. Conf. Advances Comput., Commun. Inf. Technol. (ICAICCIT)*, Faridabad, India, pp. 624–629 (2024).

[2] S. Mishra, U. Tiwari, and C. R. Kavitha, "Leveraging sequential deep learning models for robust compiler bug detection in software logs," in *Proc. 2025 3rd Int. Conf. Intell. Data Commun. Technol. Internet Things (IDCIoT)*, Bengaluru, India, pp. 1179–1185 (2025).

[3] J. Tripathy, S. S. Rao, S. Ashwani, and C. R. Kavitha, "Enhanced bug detection in compiler logs using stacking and explainable AI techniques," in *Proc. 2024 5th IEEE Global Conf. Advancement Technol. (GCAT)*, Bengaluru, India, pp. 1–6 (2024).

[4] P. Wang and Z. Guan, "Unveiling software vulnerabilities: A groundbreaking approach to defect detection utilizing CC2Vec," in *Proc. 2024 Int. Conf. Comput., Inf. Process. Advanced Educ. (CIPAE)*, Ottawa, ON, Canada, pp. 99–103 (2024).

[5] V. Kumar, P. Pranav, and M. Srinivas, "Smart contract cloning identification using BERT model based representation learning," in *Proc. 2023 IEEE 20th India Council Int. Conf. (INDICON)*, Hyderabad, India, pp. 1254–1258 (2023).

[6] X. Pan, J. Li, Z. Zhou, G. Li, X. Chen, J. Ma, J. Wu, and Q. Zhang, "Large language model-enhanced probabilistic modeling for effective static analysis alarms," *Front. Inf. Technol. Electron. Eng.*, vol. 26, no. 10, pp. 1926–1941 (2025), doi: 10.1631/FITEE.2500038.

[7] J. Chaudhuri, D. Thapar, A. Chaudhuri, F. Firouzi, and K. Chakrabarty, "SPICED+: Syntactical bug pattern identification and correction of Trojans in A/MS circuits using LLM-enhanced detection," *IEEE*

Trans. Very Large Scale Integr. (VLSI) Syst., vol. 33, no. 4, pp. 1118–1131 (2025), doi: 10.1109/TVLSI.2025.3527382.

[8] A. Gupta, M. Sharma, and A. Srivastava, "A novel dimensionality reduction-based software bug prediction using a bat-inspired algorithm," in *Proc. 2023 13th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Noida, India, pp. 278–285 (2023), doi: 10.1109/Confluence56041.2023.10048829.

[9] L. Ruggahakotuwa, L. Rupasinghe, and P. K. W. Abeygunawardhana, "Code vulnerability identification and code improvement using advanced machine learning," in *Proc. 2019 Int. Conf. Advancements Comput. (ICAC)*, Malabe, Sri Lanka, pp. 186–191 (2019), doi: 10.1109/ICAC49085.2019.9103400.

[10] J. Lu, F. Li, C. Liu, L. Li, X. Feng, and J. Xue, "CloudRaid: Detecting distributed concurrency bugs via log mining and enhancement," *IEEE Trans. Softw. Eng.*, vol. 48, no. 2, pp. 662–677 (2022), doi: 10.1109/TSE.2020.2999364.

[11] V. Rastogi, A. Pandey, and S. S. Yadav, "Code analysis to prioritize and fix bugs using Top-K pruning approach with help of machine learning," in *Proc. 2023 5th Int. Conf. Adv. Comput., Commun. Control Networking (ICAC3N)*, Greater Noida, India, pp. 1476–1482 (2023), doi: 10.1109/ICAC3N60023.2023.10541610.

[12] V. Dankan Gowda, S. V. Chakrasali, P. S. V. V. S. R. Kumar, S. A. Dhole, J. K. Patil, and S. Singh, "A novel cryptographic approach to evaluate the security of federated learning frameworks," *J. Discrete Math. Sci. Cryptogr.*, vol. 28, no. 5-A, pp. 1527–1536 (2025), doi: 10.47974/JDMSC-2151.

[13] K. Aldi, R. Septianto, R. Yulianto, and R. Taskia, "Multi-tool static code analysis for code smell and bug detection: A case study on the Java-Design-Patterns repository," in *Proc. 2025 Int. Conf. Inf. Technol. Res. Innov. (ICITRI)*, Jakarta, Indonesia, pp. 1–6 (2025), doi: 10.1109/ICITRI67507.2025.11232802.

[14] V. Dankan Gowda, D. Palanikkumar, S. Suneel, S. Dekka, T. Thiruvenkadam, and K. D. V. Prasad, "Cryptographic image-based data security strategies in wireless sensor networks," *J. Discrete Math. Sci. Cryptogr.*, vol. 27, no. 2-A, pp. 293–304 (2024), doi: 10.47974/JDMSC-1883.

[15] H. Sajnani, V. Saini, and C. V. Lopes, "A comparative study of bug patterns in Java cloned and non-cloned code," in *Proc. 2014 IEEE 14th*

Int. Working Conf. Source Code Anal. Manip. (SCAM), Victoria, BC, Canada, pp. 21–30 (2014), doi: 10.1109/SCAM.2014.12.

[16] S. V. Chakrasali, K. D. V. Prasad, P. Sarma, and M. Rahman, "Mathematical analysis of wavelet-based multi-image compression in medical diagnostics," *J. Discrete Math. Sci. Cryptogr.*, vol. 27, no. 2-B, pp. 675–687 (2024), doi: 10.47974/JDMSC-1912.

[17] X. Song, Y. Wu, J. Cao, B. Chen, Y. Lin, Z. Lu, D. Wang, and X. Peng, "BugMiner: Automating precise bug dataset construction by code evolution history mining," in *Proc. 38th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Luxembourg, Luxembourg, pp. 1919–1929 (2023), doi: 10.1109/ASE56229.2023.00201.

[18] S. T. Cynthia, "Identification and optimization of redundant code using large language models," in *Proc. 2025 IEEE/ACM 4th Int. Conf. AI Eng.—Softw. Eng. for AI (CAIN)*, Ottawa, ON, Canada, pp. 261–263 (2025), doi: 10.1109/CAIN62899.2025.00042.

Received January, 2026