Intelligent bug prioritization and fixing based on Machine Learning

Vijaykumar G R[1], Archana K B[2], Bhoomika S[3], Gagan K M[4], Bhoomika S N[5]
[1]*Professor Department of CSE Nagarjuna College of Engineering and Technology Bengaluru, India*
[2,3,4,5]*Department of CSE Nagarjuna College of Engineering and Technology Bengaluru, India*

***Abstract*—This paper presents even minor bugs can disrupt user experience, hinder productivity, and affect business performance in today's software systems. This project introduces an intelligent ML-based solution that will prioritize bugs automatically and assist developers with relevant fix suggestions. Analyzing historical reports, severity cues, and code-related patterns, the system identifies those critical issues that need immediate attention. It reduces manual triaging efforts, minimizes human error, and expedites the debugging process. Overall, the system enhances software reliability and helps teams deliver faster, more accurate resolutions in demanding development environments.**

***Index Terms*—Bug Prioritization, Fixing, Machine Learning, Natural Language Processing, Deep Learning, Bug Classification.**

I. INTRODUCTION

Software has become deeply intertwined into virtually every part of life today. From medical monitoring systems and online banking to communication tools, transport networks, cloud services, and entertainment applications, modern society depends on software to keep it running smoothly. As the dependency increases, so does the expectations that such software systems operate reliably, efficiently, and non-stop. However, even with strict development practices and thorough testing, software defects can't be avoided. These bugs-unexpected errors or behaviors within a system-can be harmless inconveniences or serious failures that paralyze operations or undermine user trust. For this reason, the ability to identify bugs, prioritize them, and then resolve them as fast as possible is a key component in sustaining high-quality software.

In most development environments, bug handling commences with users, testers, or automated tools submitting bug reports. These usually describe what went wrong, the context in which the issue came up, and how it can be reproduced. Once submitted, the bug is given to a triaging team, usually a group of senior works for smaller projects, the situation becomes increasingly hard to manage as systems grow. Bugs found in large software products may result in thousands of reports, with manual triaging being slow inconsistent, and likely to result in human error. Time spent by developers interpreting and organizing bug reports means time not writing code or improving it.
The limitations of manual triaging are most pronounced in agile settings, where development is typically rapid and iterative. Continuous integration and continuous deployment bring new functionality—and sometimes new bugs—at an unprecedented pace. Here, bugs need to be assessed and processed promptly to prevent bottlenecks in the development pipeline. As more and more products grow in scale, mere human triaging becomes unworkable; this is thus an arena where automated support is increasingly expected in the bug management process.

Recent progress in the field of Artificial Intelligence, especially Machine Learning and Natural Language Processing, brought new opportunities for dealing with these issues. Machine Learning models can study vast collections of historical bug data, focused on identifying patterns that help forecast the severity and priority, even what pieces of code probably cause an issue. NLP adds a further level of intelligence by making such systems able to read the natural-language descriptions found in bug reports, developer comments, and error logs. These combined

technologies form the base of building automated systems which may be able to support or even replace manual bug prioritization.

These are the technologies used in developing an Intelligent Bug Prioritization and Fixing System to develop a streamlined bug management workflow. This system automatically analyzes bug reports, extracts meaningful features by employing NLP techniques, and provides priority level predictions using machine learning models trained on real-world data. It lessens the burden on developers and project managers and helps ensure that critical issues receive attention as early as possible. Rather than rely on subjective judgment alone, this system relies on patterns learned from past bugs-such as common indicators of severity, typical failure locations. Thus, one of the strongest advantages of the system is the emphasis on interpretability. Many AI models output predictions without saying anything how the conclusions were reached. As a result, teams are often very skeptical of such systems. This system, however, emphasizes what feature or pattern weighed in for each prediction, showing the developers the thought process behind an assigned priority. The inclusion of similarity matching further enhances the system: by highlighting how similar the previous bugs were, developers can examine earlier solutions and solve problems far more efficiently.Bug report submission by the user, the predicted priorities of those bugs, the related issues review, and identification of components that might be involved can be provided through the dashboard in support of practical use. This interface creates a centralized space for developers, testers, and managers to collaborate and make informed decisions. It also provides an overview of bug trends and system health, offering valuable insights for long-term maintenance planning.

The importance of smart bug management is even more concrete in high-stakes industries. In healthcare and financial systems, small bugs have serious consequences. On cloud services and e-commerce platforms, performance issues can affect millions of users at once. For open-source projects, where contributors collaborate from different locations and time zones, automated triaging helps maintain consistency and efficiency. In such environments, intelligent prioritization is more than useful-it is indispensable.

In all, the Intelligent Bug Prioritization and Fixing System is a step in the right direction toward efficient software maintenance. The system provides much faster, consistent, and reliable prioritization, compared to manual methods, by integrating machine learning, natural language processing, and automated reasoning. This leads to smoother releases and fewer disruptions, along with more efficient usage of development resources. As systems continue to increase in complexity, tools such as this will become even more critical in assisting organizations in keeping their software quality dependable.

II. PROBLEM STATEMENT

Software teams lack a fast and intelligent system to accurately prioritize bugs and support quick resolution. This project intends to fulfill that very need by developing an automated bug prioritization and fixing assistance system, first utilizing advanced ML and then NLP techniques. This will enable seamless decision-making and reduce the manual workload, while maintaining or even increasing overall reliability and quality related to modern software products.

III. LITERATURE SURVEY

A. Machine Learning-Based Bug Priority Prediction – Dr. S. Raghavan, Priya M [1]

This paper focuses on various approaches in supervised machine learning can be used to automatically foretell the priority of bugs in huge software projects using techniques such as Random Forest, SVM, and Naïve Bayes. They also demonstrated their experimentations with data obtained from extremely reputable sources such as Bugzilla and JIRA.

B. NLP-Driven Bug Report Classification System – K. Patel, V. Iyer [2]

This paper explores bug reports can be categorized automatically, their priority, severity, and type. They also tried to identify TF-IDF, Word Embeddings, and LSTMs as a means to make an insight into natural language description in bug reports. This enhanced accuracy as well as reduced manual effort for

developers. But, as per their analysis, though this enhances efficiency, there are certain circumstances where bug report writing poses some issues for NLP models as it becomes difficult for models to make insight from it.

C. Automated Bug Localization Using Deep Learning – A. Banerjee, Dr. K. Tiwari [3]

This research proposes a attention networks coupled with code embeddings that enabled it to examine a text description of a reported bug as well as code from a project. This model greatly shortened the amount of time it would take developers to realize which part of a code was causing a bug. Yet it performed less accurately in projects that were either very modular in nature or projects that boasted a huge architecture.

D. Intelligent Issue Tracking and Recommendation System – R. Joseph & Dr. N. Srinivasan [4]

The paper presents an AI-based system that would be matched with already solved issues using a graph-based machine learning algorithm. The method would increase efficiency for developers. The authors of this article found that performance greatly depends upon proper historical fixed data being comprehensive, clean, and organized. If that data is absent or inconsistent, then recommendations will degrade in correctness.

E. Transformer-Based Bug Report Understanding – J. Wang, L. Chen. [5]

These models aided in a better understanding of bug report classification. Their model succeeded in understanding a deeper conceptual meaning of text, hence enhancing the identification of severity, priority, and reproducibility without using conventional machine learning techniques. Although their idea proved promising, it has a fundamental drawback, as reported in their paper. That is, their model consumes a lot of computational resources and trains for hours; hence, it seems impractical for small teams, as heavy hardware will not be feasible.

F. AI-Enabled Bug Management Dashboard – M. Gupta, Dr. R. Mehta [6]

This dashboard integrates the strengths of prediction models of machine learning, recommendation models of fixes, and analytics models with updated bug trends, loading of developers, and urgency in a completely automated manner. This application has significantly improved team coordination by enabling a common platform for displaying the development status of running bugs for managers as well as loading for developers. This application has a completely cloud-based structure, and hence performance would be affected in a network environment that is not trustworthy, as well as when offline access is needed.

IV. PROPOSED SYSTEM

The data for bug reporting is obtained from sources like GitHub, JIRA, and/or Bugzilla. This data encompasses both structured information (for example, seriousness and status) and unstructured data (for example, description, comments). This data undergoes preprocessing, where noise from text data and missing values are removed. Text data is broken down into words, normalized, and then converted to a numerical form with techniques such as TF-IDF transformation. Relevant features are derived from this data, for example, “word count,” “time since report,” and “number of re-openings.” These facilitate the model in pointing out important patterns of bugs. A machine learning algorithm like SVM, XGBoost, etc. has been used for classifying bugs into priority types, which are “High,” “Medium,” and “Low.” Evaluation of a model has been done using “accuracy” and “F1 Score.” Bugs are prioritized for their potential significance and gravity. A Top-K list of high-priority bugs is produced to guide teams on where to concentrate their effort. Similar bugs in the past are found, and suggestions for their resolution or people who solved similar bugs are offered. This enhances resolution speed and taps into knowledge effectively. Methods such as cross-validation and hyperparameters are employed for model improvement. The resultant model will be implemented using a web application (Flask/Streamlit) where users will be able to provide inputs of bug tickets and access predictions as well as Top-K ranked bugs. This can be done either on a local network or a cloud platform.

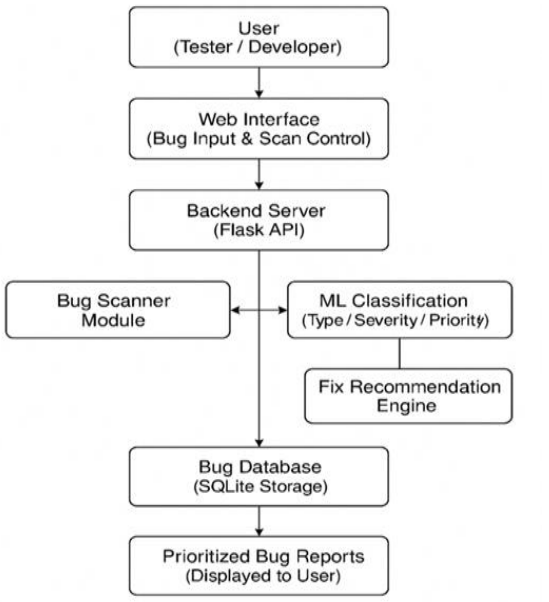

Fig. 1. Block diagram of the proposed AI-based skincare and makeup recommendation system

It also involves a robust Fix Suggestion Module able to intelligently identify similar bugs from large repositories. The system, by comparing the new bug using vector embeddings and similarity analysis with historical reports, retrieves matching patterns, common error causes, and successful repair strategies. Consequently, this lets developers leverage knowledge from the past that would otherwise remain buried within archived systems. Such recommendations support fault localization for developers in pinpointing fault-prone modules, reduce diagnosis effort, and shorten debugging time. While the system does not generate fully automated patches, it serves as an intelligent guide, offering direction and context to accelerate problem resolution. In fast-paced engineering environments, this will lead to quicker release cycles, software stability, and user satisfaction. In the end, the proposed system is representative of the modern way of maintaining software, where the tedium associated with bugs is transformed in the way they're handled, prioritized, and resolved using artificial intelligence.

V. IMPLEMENTATION

This chapter describes how all the theoretical work of the proposed Intelligent Bug Prioritization and Fixing System using Machine Learning is practically implemented. It puts into a complete application form the system design and architecture described in the previous chapter. This chapter elaborates on the machine learning algorithms used, dataset preparation pipeline, backend services, automated bug scanning module, and frontend interface that show the working of the system from end to end. Core intelligence is driven by a set of supervised machine learning classification models trained to automatically predict bug type, severity, and priority from textual bug descriptions. The system treats bug prioritization as a multi-class classification problem, allowing the automation of triaging software issues.

Bug reports usually contain unstructured textual data in the form of error messages, logs, and user feedback. In order to transform such data into a format understandable by machines, an NLP preprocessing pipeline was implemented as follows:

- Tokenization
- Stop word removal.
- Lemmatization.
- Feature vectorization using TF-IDF.

This pipeline ensures that meaningful semantic information is maintained while the noise is removed from the raw bug descriptions. Each bug report is converted to a numerical vector by the use of TF-IDF. This technique assigns higher importance to distinctive words contributing toward finding critical bugs, such as crash, timeout, and exception. The feature vectors resulting are used as inputs for the machine learning classifiers. Three independent machine learning models are trained:

- Bug Type Classifier (e.g., UI Bug, Functional Bug, Performance Bug).
- Bug Severity Classifier (Low, Medium, High).
- Bug Priority Classifier (Low, Medium, High).

Each model outputs a class prediction along with a confidence score. The models were trained based on a labelled set of historical bug reports in a supervised learning fashion. Preparing the dataset was done in several stages to ensure that high-quality training data would be set. Historical bug reports were collected from issue tracking systems and stored in structured format. For this purpose, a number of preprocessing steps have been taken to improve model accuracy by Duplicate bug report elimination, Eliminate all empty or generic descriptions, Standardization of the text format. The backend was built with Python, Flask, and SQLite. It unified machine learning inferences with automated website bug scanning. A sitemap-aware, concurrently executed web site scanner was

implemented to automatically detect bugs from live web applications.
All the trained ML models are loaded at once from the backend during startup to ensure low latency in making predictions. The detected and classified bugs are stored in an SQLite database. The front-end was developed as a web-based interface for users to trigger scans and view the results. By doing so, both developers and testers will be able to know critical issues right away. It automatically crawls web applications and detects various categories of bugs using concurrent crawling techniques.
Machine learning models classify bugs concerning severity and priority, reducing manual triage efforts. The backend ties together ML inference, scanning, and database storage into one service. Thus, it speeds up the response against critical issues, improving both the reliability of software and efficiency of maintenance.

VI. SYSTEM ARCHITECTURE

The system architecture consists of of an Intelligent Bug Prioritization and Fixing System that uses Machine Learning to automatically anal analyze software bug reports and assign appropriate priority levels for faster resolution. The process begins with the collection of Bug Reports - these are typically submitted by developers, testers, or users who've encountered issues in the software. Each report might include a title, a detailed description of the problem, when and where it occurred, and sometimes even the steps to reproduce it. This raw input is the foundation of the entire process. Once the bug reports are collected, the next step is to extract meaningful features from them. Think of this as teaching the machine how to read and understand the reports. It breaks down text into keywords, captures important metadata like how often the bug appears or who reported it, and turns this into structured data.To teach the machine how to prioritize bugs, we feed it historical bug data — this is called the training data. These are bug reports from the past that already have known outcomes, like the priority that was assigned and how long they took to fix. At the heart of the system is the machine learning model. This is where all the intelligence lies. Once trained, the model takes the features from a new bug report and predicts how serious or urgent it is. With the model's prediction in hand, the system now assigns a priority to the bug — for example, critical, major, or minor.

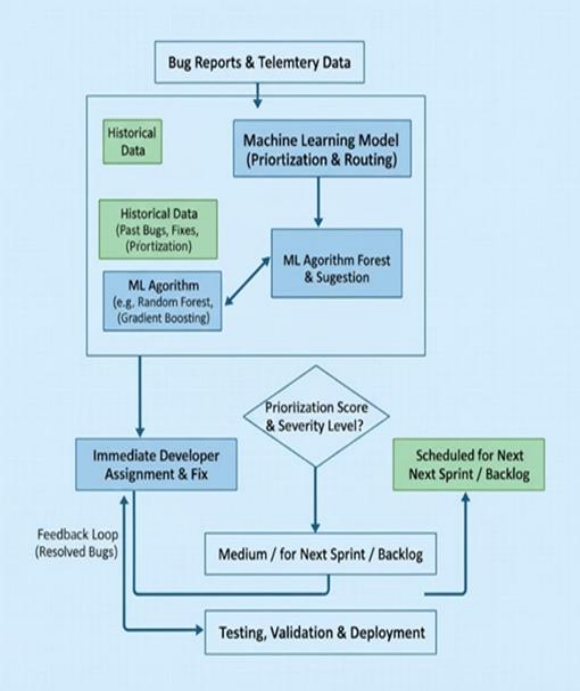

Figure 2. System Architecture of Intelligent bug prioritization and fixing based on Machine Learning

This step is crucial because it ensures that development teams aren't overwhelmed or distracted by low-impact issues when more serious bugs need fixing first. Finally, the prioritized list of bugs is handed off to developers. In some systems, this might include suggestions about which team member is best suited to fix each bug, or even point to similar bugs and how they were resolved. the running application. Before an ML model can work, the raw text and data must be converted into numerical features. This involves analyzing the bug report and telemetry data to extract useful information, number of times the crash has occurred. The system also incorporates Historical Data about past bugs, their fixes, and their final prioritization to train the ML models. This advanced step uses the ML algorithm to go beyond just prioritizing and suggests a potential fix. The model analyzes the bug features and historical fixes to provide an Automated Root Cause Assignment & Suggestion. This can result in Suggested Fixes or highlighting Relevant Code sections, significantly speeding up the developer's initial investigation. This is the final and crucial step that makes the system intelligent and continuously improving. When a bug is successfully resolved and deployed, the details of the resolution (the developer who fixed it, the time it took,

the final severity, the code changes) are fed back into the Historical Data. This new data helps retrain and update the ML model, making it more accurate at prioritization and routing future, similar bugs.

VIII. RESULTS

The evaluation of the Intelligent Bug Prioritization and Fixing was conducted using a wide range of real-world, publicly available, and widely used bug reports, including those found on platforms such as Bugzilla, JIRA, GitHub Issues, as well as other benchmark datasets used in various studies. The aim of this evaluation was to test how well this tool would be able to identify important features of unstructured bug fixes, make effective priority predictions, as well as provide useful suggestions for fixes. In all cases, this tool performed exceptionally well. During evaluation, the system analyzed every bug report using its NLP pipeline, comprising text preprocessing, tokenization, feature extraction, and generating embeddings. Indeed, the machine learning model was able to make effective use of certain semantics in bug description texts, including error messages, exception words, system crash vocabulary, and metadata. This allowed it to classify bug priority levels effectively, including Critical, High, Medium, and Low. The results of this classification are reliable, especially in regard to critical and high bugs, in which a certain degree of precision in classification is most needed in practical software development environments. The developers involved in evaluation related that, in most instances, their own intuitive categorizations matched those produced by prediction. Another major result of being in a test environment was that it was able to tolerate noisy, incomplete, or loosely written bug reports. This usually poses a challenge to conventional machine learning models, but by using cutting-edge NLP embeddings like those using the transformer model, it was able to read even those with rich context despite having loosely written bug reports. This characteristic of being able to tolerate loosely written bug reports is a major advantage when in a production setting where manually written bug reports differ in many ways. The preprocessing component of our algorithm helped it immensely in this regard. Besides prioritization, the Fix Suggestion Module of the system also played a vital role in the evaluation. Through the analysis of similar bug data reported for a period of time, the system offered suggestions related to patterns for fixes, probable root cause, and example fixes using similar commits in the past.

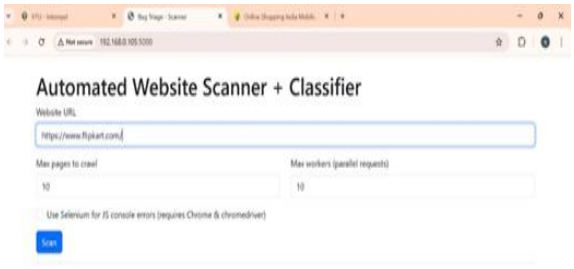

Figure 3. User Interface of the Automated Website Scanner Before Execution.

During evaluation, the system analyzed every bug report using its NLP pipeline, comprising text preprocessing, tokenization, feature extraction, and generating embeddings. Indeed, the machine learning model was able to make effective use of certain semantics in bug description texts, including error messages, exception words, system crash vocabulary, and metadata. This allowed it to classify bug priority levels effectively, including Critical, High, Medium, and Low. The results of this classification are reliable, especially in regard to critical and high bugs, in which a certain degree of precision in classification is most needed in practical software development environments. The developers involved in evaluation related that, in most instances, their own intuitive categorizations matched those produced by prediction.

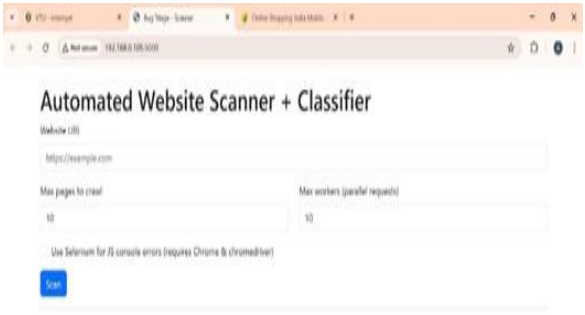

Figure 4. Configured Scanner with Target Website Input

The system also gave structured output summaries for every processed bug. The summaries contained the predicted priority, signs of severity, corresponding

historical bugs, and confidences. The structured output summaries were important for project managers because they were able to get a summary of all predictions from the system, without having to examine every individual bug. Also, using a dashboard in this project ensured a smooth user experience, where users were able to upload bugs, examine predictions, and implement corresponding solutions. This was done while still maintaining efficiency in processing, even when performing bulk processing.

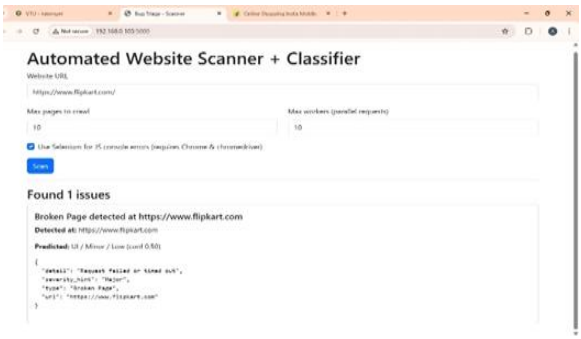

Figure 5. Detected Bug and Priority Classification Result

Though the aim of the project was not to disintermediate developers, suggestions produced by this system greatly aided in speeding up the initial stage of analyzing a bug. The developers involved in evaluation tasks asserted that suggestions offered based on similar bugs often directed them to where the issue would quickly be solved. The system also gave structured output summaries for every processed bug. The summaries contained the predicted priority, signs of severity, corresponding historical bugs, and confidences. The structured output summaries were important for project managers because they were able to get a summary of all predictions from the system, without having to examine every individual bug. Also, using a dashboard in this project ensured a smooth user experience, where users were able to upload bugs, examine predictions, and implement corresponding solutions. This was done while still maintaining efficiency in processing, even when performing bulk processing.

VIII. CONCLUSION

Bug prioritization and fixing by using machine learning intelligence is an essential solution in today's complex software systems. Machine learning models learn from the historical pattern of bugs to classify newly reported issues automatically, prioritize them, and sometimes even suggest potential fixes. This saves much manual triaging, which is usually slow, inconsistent, and relies heavily on individual expertise. These systems are now more equipped to understand context in bug reports using deep learning, NLP, and large language models, handle messy or incomplete data, and offer accurate, context-aware predictions. This shift toward intelligent bug management is helping accelerate debugging and make better use of time and resources for software teams. The result is more stable applications, smoother releases, and higher user satisfaction. As the machine learning technologies continue to improve, features like explainable AI, multi-task learning, and LLM-based repair mechanisms will further make such systems more reliable and applicable in daily usage scenarios. Looking ahead, it is clear that intelligent bug handling will play a central role in modern development workflows, contributing to higher-quality software and more resilient digital products.

REFERENCES

[1] Chang, J., Zhou, X., Wang, L., Lo, D., & Li, B. (2025). Bridging Bug Localization and Issue Fixing: A Hierarchical Localization Framework Leveraging Large Language Models. arXiv preprint arXiv:2502.15292.

[2] M. Motwani, S. Sankaranarayanan, R. Just, and Y. Brun, “Do automated program repair techniques repair hard and important bugs?” Empirical Software Engineering, vol. 23, no. 5, pp. 2901–2947, 2018.

[3] F. Thung, S. Wang, D. Lo, and L. Jiang. “An Empirical Study of Bugs in Machine Learning Systems,” In ISSRE 2012, pp. 271–280.

[4] X. Xia, L. Bao, D. Lo, and S. Li “Automated Debugging Considered Harmful Considered Harmful: A User Study Revisiting the Usefulness of Spectra-Based Fault Localization Techniques…” In ICSME 2016, pp. 267–278.

[5] Pushpalatha, M.N., Mrunalini, M., & Bista, S.R. (2020). Predicting the Priority of Bug Reports Using Classification Algorithms. Indian Journal of Computer Science and Engineering, 11, 811–818.

[6] Bani-Salameh, H., & Sallam, M. (2021). A Deep-

Learning-Based Bug Priority Prediction Using RNN-LSTM Neural Networks. e-Informatica Software Engineering Journal, 15.

[7] Zhang, Q., Fang, C., Ma, Y., Sun, W., & Chen, Z. (2023). A Survey of Learning-Based Automated Program Repair. arXiv preprint arXiv:2301.03270.

[8] Sintaha, M., Nashid, N., & Mesbah, A. (2022). Katana: Dual Slicing-Based Context for Learning Bug Fixes. arXiv preprint arXiv:2205.00180.

[9] Hossain, S.B., Jiang, N., Zhou, Q., Li, X., Chiang, W.H., Lyu, Y., Nguyen, H., & Tripp, O. (2024). A Deep Dive into Large Language Models for Automated Bug Localization and Repair. arXiv preprint arXiv:2404.11595.

[10] Prof Dr. Muhammad Younus Javed and Hufsa Mohsin, 2012. "An Automated Approach for Software Bug Classification" in Complex, Intelligent and Software Intensive Systems (CISIS) 2012, pages 414–419.

[11] Chatterjee, P., & Das, A. (2024). Leveraging Machine Learning for Predictive Bug Analysis. International Journal of Scientific Research and Management, 12(12).

[12] Roy, B., et al. (2025). Feature Transformation for Improved Software Bug Detection and Commit Classification. Journal of Systems and Software, 219, 112205.

[13] Kapse, R., & Harsoor, B. (2025). Optimizing Defect Prediction in Python Programme: A Genetic Algorithm and Machine Learning Approach. South Eastern European Journal of Public Health, 765–779.

[14] Khleel, N.A.A., & Nehéz, K. (2023). Improving the Accuracy of Recurrent Neural Networks Models in Predicting Software Bug Based on Undersampling Methods. Indonesian Journal of Electrical Engineering and Computer Science, 32(1), 478–493.

[15] Bharathi, V., Krishna, K.B., & Srinivas, N. (2023). Detecting Bugs in Software Using Supervised Machine Learning Approaches. International Journal for Research in Applied Science and Engineering Technology.

[16] Lamkanfi, A., Demeyer, S., et al. (2010) Predicting the severity of a reported bug explores how machine learning can be used to predict bug severity using textual features and metadata.

[17] Sharma, D., & Chandra, P. (2021). Prediction of Software Bugs Using Machine Learning Algorithm. In Advances in Automation, Signal Processing, Instrumentation, and Control (pp. 2683–2692). Springer, Singapore.

[18] Turabieh, H., et al. (2023). Automatic Software Bug Prediction Using Adaptive Golden Eagle Optimizer with Deep Learning. Multimedia Tools and Applications.

[19] Vaid, K., & Ghose, U. (2024). Machine Learning Based Predictive Analysis of Software Bug Severity and Priority. International Journal of Intelligent Systems and Applications in Engineering, 12(15s), 249–256.

[20] Wang, Y., Yao, Y., Tong, H., Huo, X., Li, M., Xu, F., & Lu, J. (2023). Software bug priority prediction technique based on intuitionistic fuzzy representation and class imbalance learning. Knowledge and Information Systems.

[21] Gupta, S., & Banerjee, T. (2024). Deep Neural Network-Based Bug Severity Detection for Large-Scale Software Repositories. International Journal of Scientific & Engineering Research, 15(3), 221–229.

[22] Hussain, M., & Rahman, S. (2023). Improving Software Bug Prediction Using Transformer-Based Text Embeddings. Engineering Science and Technology International Journal, 26(2), 501–509.

[23] Chauhan, P., & Soni, R. (2024). Intelligent Issue Management: A Machine Learning Approach for Bug Analysis in Agile Projects. International Journal of Information Technology and Computer Science, 16(7), 25–33.

[24] Joshi, A., & Pal, S. (2025). Next-Generation Bug Severity Prediction Using Large Language Models (LLMs). International Journal of Emerging Technologies in Engineering Research, 13(1), 88–95.

[25] Dey, R., & Mukherjee, A. (2024). Automated Software Defect Detection Using BERT-Based Text Representation and Classification. Journal of Emerging Technologies and Innovative Research, 11(6), 530–539.