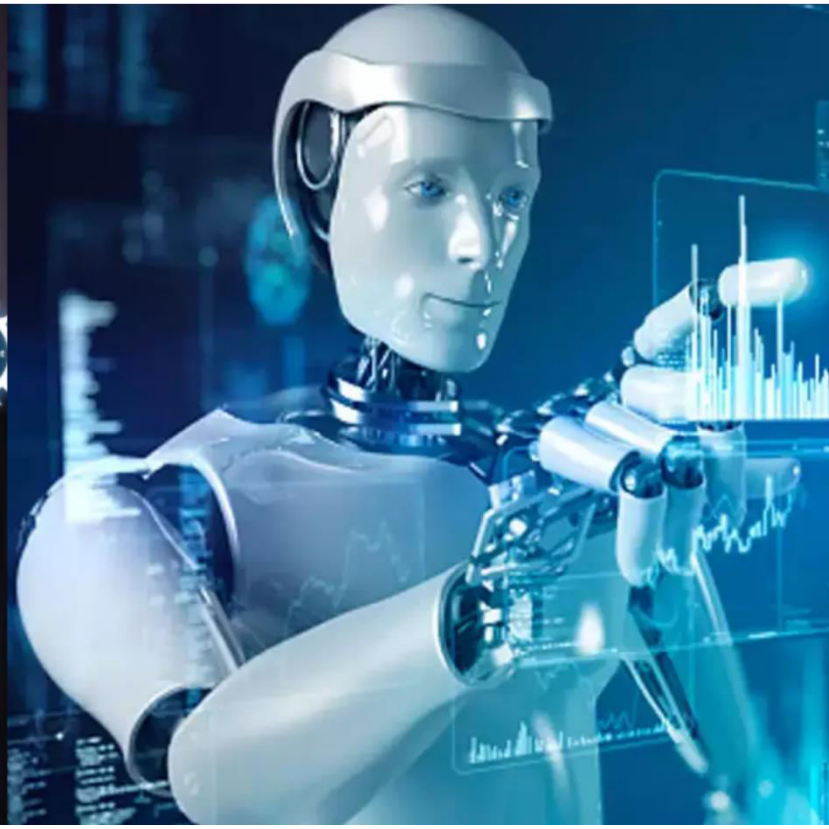
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 15, Issue 3, March 2026

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503348|

AI Assisted Secure Code Review and Vulnerability Prediction Platform

S.Subashini[1], T. B. Karthikeyan[2], M.G.I.Jithamanyu Babu[3], S.Vishnu Priyan[4]

Associate Professor, Dept. of CSE, K.L.N. College of Engineering, Sivagangai, Tamil Nadu, India[1]

UG Scholar, Dept. of CSE, K.L.N. College of Engineering, Sivagangai, Tamil Nadu, India [2,3,4]

ABSTRACT: Secure software development has become a critical requirement in modern computing environments due to the increasing number of cyber threats and software vulnerabilities. Traditional manual code review techniques are time-consuming, error-prone, and often fail to detect complex security issues in large codebases. This paper proposes an AI Assisted Secure Code Review and Vulnerability Prediction Platform that automatically analyzes source code to identify potential security flaws and coding vulnerabilities. The system utilizes a Gated Graph Neural Network (GGNN) to model source code as graph-structured data and capture semantic relationships between program components. The proposed platform integrates graph-based code representation, GGNN-based vulnerability prediction, and automated report generation to assist developers in improving code quality and security. Experimental evaluation demonstrates that the proposed system improves vulnerability detection accuracy, reduces manual review effort, and enhances overall software security. The platform provides a scalable and intelligent solution for modern software development environments.

KEYWORDS: Artificial Intelligence, Secure Code Review, Vulnerability Prediction, Gated Graph Neural Network (GGNN), Graph Neural Networks, Static Code Analysis, Software Security, Deep Learning

I. INTRODUCTION

Software security has become a critical concern in modern software development due to the increasing number of cyber threats and vulnerabilities present in applications. Vulnerabilities in source code can lead to severe consequences such as data breaches, unauthorized access, and system failures. Traditional approaches such as manual code review and static analysis tools are widely used to detect vulnerabilities; however, they often suffer from limitations including high false positive rates and significant manual effort. Reports from the OWASP Top 10 highlight the most common and critical security risks in web applications, emphasizing the need for more effective and automated vulnerability detection techniques [7].

Earlier research focused on software metrics and static analysis to identify vulnerabilities. For example, Shin et al. evaluated software metrics as indicators of vulnerabilities [6], while Johnson et al. discussed the challenges developers face in using static analysis tools effectively [5]. Although these approaches provide useful insights, they lack the ability to capture deep semantic relationships within source code.With the advancement of machine learning and deep learning, researchers have proposed automated approaches for vulnerability detection. Li et al. introduced VulDeePecker, a deep learning-based system that learns vulnerability patterns from code snippets [1]. Similarly, Zhou et al. proposed Devign, which utilizes graph neural networks to learn program semantics for vulnerability identification [2]. Other studies, such as Chakraborty et al., highlight that while deep learning techniques improve detection accuracy, there is still a need for models that can effectively capture complex code dependencies and reduce false positives [4]. In addition, Russell et al. explored automated vulnerability detection using deep learning techniques, demonstrating the effectiveness of neural models in identifying security flaws [3].

Recent advancements in graph-based learning have shown promising results in analyzing structured data such as source code. According to Allamanis et al., machine learning models benefit significantly from representing source code as graphs, as it preserves both syntactic and semantic information [8]. Graph-based representations such as Abstract Syntax Trees (AST), Control Flow Graphs (CFG), and Data Flow Graphs (DFG) enable a deeper understanding of program behavior.To address the limitations of existing approaches, this paper proposes an AI Assisted Secure Code Review and Vulnerability Prediction Platform based on the Gated Graph Neural Network (GGNN). The GGNN model

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503348|

is capable of processing graph-structured representations of source code and capturing both local and global dependencies through iterative message passing. This enables the system to detect complex vulnerabilities more accurately compared to traditional machine learning and static analysis methods.The proposed system integrates graph construction, GGNN-based analysis, and automated report generation to assist developers in identifying and fixing security issues efficiently. By leveraging deep learning and graph-based representations, the system aims to improve vulnerability detection accuracy, reduce false positives, and provide a scalable solution for secure software development.

II. METHODOLOGY

The proposed AI Assisted Secure Code Review and Vulnerability Prediction Platform follows a structured methodology based on graph-based deep learning to analyze source code and detect potential security vulnerabilities. The methodology begins with the collection of source code datasets containing both secure and vulnerable code samples. These datasets are obtained from open-source repositories, vulnerability databases, and previously analyzed software projects. The collected data provides a strong foundation for training the GGNN model to recognize patterns associated with secure and insecure coding practices.The first stage of the methodology involves code preprocessing and graph construction. The input source code is analyzed using lexical and syntax analysis techniques to understand its structure. Instead of extracting flat features, the system converts the code into graph representations such as Abstract Syntax Trees (AST), Control Flow Graphs (CFG), and Data Flow Graphs (DFG). In these graphs, nodes represent program elements such as statements, variables, and functions, while edges represent relationships such as control dependencies and data flows. This graph-based representation enables the system to preserve the structural and semantic information of the program.Once the graph is constructed, each node is transformed into a vector representation through an embedding process. These node embeddings serve as the initial input to the **Gated Graph Neural Network (GGNN)** model. The GGNN processes the graph through multiple iterations of message passing, where each node exchanges information with its neighboring nodes. This process allows the model to propagate contextual information across the graph and learn meaningful representations of the code structure.

During each iteration, the hidden state of each node is updated using a gated mechanism, which helps retain important information while filtering out irrelevant details. This iterative propagation enables the model to capture both local relationships and long-range dependencies within the code. After completing the propagation steps, the final node representations are aggregated to form a graph-level representation.The graph-level representation is then passed to a classification layer, which predicts whether the given code contains vulnerabilities. The model is trained using labeled datasets, where each code sample is annotated as secure or vulnerable. During deployment, the trained GGNN model analyzes new code inputs and predicts potential vulnerabilities with high accuracy.Finally, the system generates a detailed vulnerability report highlighting affected code segments, severity levels, and recommended fixes. This methodology enables automated, intelligent, and scalable vulnerability detection by leveraging the power of graph-based deep learning, significantly improving the efficiency and effectiveness of secure code review.

III. PROCESS FLOW

The proposed platform follows a systematic workflow to ensure efficient vulnerability detection and secure code analysis using graph-based deep learning techniques. The process flow consists of several stages that transform raw source code into a detailed vulnerability report. The first stage is code submission, where developers upload their source code through the system interface. The platform supports multiple programming languages and allows users to submit individual files or complete project repositories.Once the code is submitted, the system performs code preprocessing. During this stage, lexical analysis is carried out to break the source code into tokens such as keywords, identifiers, operators, and constants. Syntax analysis is then performed to understand the structural relationships between these tokens. This step helps in identifying important programming constructs and prepares the code for graph transformation.The next stage involves graph construction and representation. The processed code is converted into graph-based structures such as Abstract Syntax Trees (AST), Control Flow Graphs (CFG), and Data Flow Graphs (DFG). In these graphs, nodes represent program elements like statements, variables, and functions, while edges represent relationships such as control flow and data dependencies. This representation preserves the semantic and structural information of the code, which is essential for accurate vulnerability detection.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503348|

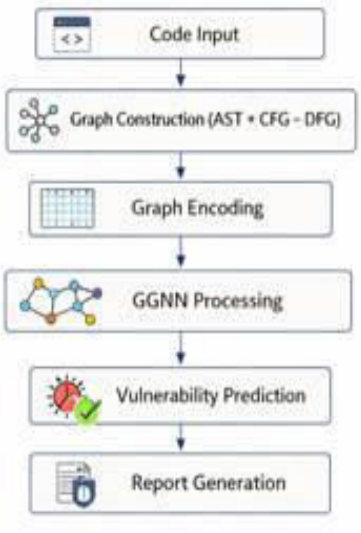


Figure 1: Process Flow

After graph construction, the system enters the GGNN processing stage. The graph is provided as input to the **Gated Graph Neural Network (GGNN)**, where each node is initialized with a feature vector. The GGNN performs iterative message passing between nodes, allowing the model to propagate contextual information across the graph. Through multiple iterations, the network learns complex relationships and dependencies within the code structure.Once the GGNN processing is completed, the system performs vulnerability prediction. The learned graph representations are passed through a classification layer, which determines whether the code contains vulnerabilities. The prediction is based on patterns learned during the training phase and reflects the likelihood of security issues in the code.Finally, the system generates a security analysis report. The report includes detailed information about detected vulnerabilities, their severity levels, possible causes, and recommended solutions. Developers can review this report to understand and fix security issues in their code. The corrected code can be resubmitted to the system for further analysis, enabling continuous improvement in software security.

IV. ALGORITHMS USED

The proposed AI Assisted Secure Code Review and Vulnerability Prediction Platform primarily utilizes the Gated Graph Neural Network (GGNN) as the core algorithm for vulnerability detection. Unlike traditional machine learning algorithms that rely on handcrafted features, GGNN operates on graph-structured representations of source code, enabling the system to capture complex semantic and structural relationships between different components of a program.In the proposed system, the source code is first converted into graph representations such as Abstract Syntax Trees (AST), Control Flow Graphs (CFG), and Data Flow Graphs (DFG). Each node in the graph represents a program element such as a statement, variable, or function, while edges represent relationships such as control flow and data dependencies. This graph-based representation allows the GGNN model to analyze how different parts of the code interact, which is critical for identifying vulnerabilities.The GGNN processes the graph using an iterative message-passing mechanism. Initially, each node is assigned a feature vector that encodes its properties. During each iteration, nodes receive messages from their neighboring nodes, and these messages are aggregated to update the node's hidden state. The update process is controlled using a gated mechanism similar to a Gated Recurrent Unit (GRU), which helps retain relevant information while filtering out noise.

The hidden state update in GGNN can be mathematically expressed as:

$$h_v^{(t+1)} = GRU\left(h_v^{(t)}, \sum_{u \in N(v)} W \cdot h_u^{(t)}\right)$$

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503348|

where hv(t)h_v^{(t)}hv(t) represents the hidden state of node vvv at iteration ttt, and N(v)N(v)N(v) represents the neighboring nodes. This iterative process continues for a fixed number of steps, enabling the model to capture both local and global dependencies within the code.After the propagation phase, the final node representations are aggregated to form a graph-level embedding, which is then passed to a classification layer for vulnerability prediction. The GGNN model is trained using labeled datasets containing secure and vulnerable code samples, allowing it to learn patterns associated with different types of vulnerabilities.The use of GGNN provides several advantages, including improved semantic understanding of code, better handling of complex program structures, and reduced dependence on manual feature engineering. This makes the proposed system more accurate and effective compared to traditional machine learning-based approaches for vulnerability detection.

V. SYSTEM ARCHITECTURE

The architecture of the proposed AI Assisted Secure Code Review and Vulnerability Prediction Platform is designed to provide a scalable and efficient framework for automated code analysis using graph-based deep learning techniques. The system follows a modular architecture where each component performs a specific function within the vulnerability detection pipeline.The first component is the User Interface Module, which allows developers to interact with the platform. Through this interface, users can upload source code files, initiate the analysis process, and view the generated vulnerability reports. The interface also provides visualization of detected vulnerabilities and suggestions for fixing security issues.The second component is the Code Preprocessing and Graph Construction Module. This module is responsible for analyzing the submitted source code and converting it into graph representations such as Abstract Syntax Trees (AST), Control Flow Graphs (CFG), and Data Flow Graphs (DFG). It performs lexical analysis, syntax parsing, and graph generation to capture the structural and semantic relationships within the code.

The constructed graphs are then passed to the core component of the system, the GGNN-Based Graph Processing Engine. This module utilizes the Gated Graph Neural Network (GGNN) to process the graph data. The GGNN performs iterative message passing between nodes, updating their hidden states and learning meaningful representations of the code structure. This enables the system to detect complex vulnerabilities based on program dependencies.Once the graph processing is completed, the output is forwarded to the Vulnerability Detection Module. This module applies a classification mechanism to determine whether the code contains vulnerabilities and categorizes them based on severity levels such as low, medium, or high risk. It also identifies specific code segments associated with the detected vulnerabilities.The final component is the Report Generation Module, which generates a comprehensive security report for the developer. The report includes detailed descriptions of detected vulnerabilities, affected code sections, severity levels, and recommended solutions. This helps developers understand and fix security issues efficiently.

The modular architecture ensures flexibility, scalability, and seamless integration with modern development environments, making the system suitable for real-world software development workflows

VI. PERFORMANCE ENHANCEMENT

The proposed AI-assisted code review platform significantly enhances software development efficiency and security by incorporating graph-based deep learning techniques. Traditional manual code review methods require developers to inspect large volumes of code, which is time-consuming and prone to human error. The integration of the Gated Graph Neural Network (GGNN) enables automated and intelligent analysis of source code by capturing complex structural and semantic relationships that are often missed by conventional approaches.One of the key improvements provided by the system is increased vulnerability detection accuracy. Unlike traditional machine learning models that rely on flat feature representations, the GGNN processes graph-structured code, allowing it to understand dependencies between program components such as variables, functions, and control flows. This deep structural understanding enables the model to identify complex vulnerabilities more effectively, including logical errors, improper input validation, and insecure data handling practices.

Another significant improvement is the reduction of false positives. Many traditional static analysis tools generate numerous warnings, including irrelevant ones, which can overwhelm developers. The GGNN-based approach leverages contextual information from graph structures to distinguish between genuine vulnerabilities and safe coding patterns.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503348|

This results in more precise predictions and allows developers to focus on critical security issues.The platform also reduces code review time by automating the entire vulnerability detection process. Developers can upload their code and receive detailed analysis reports within a short period. This eliminates the need for extensive manual inspection and accelerates the software development lifecycle. Additionally, the system supports continuous security improvement by allowing developers to iteratively analyze and refine their code.Overall, the integration of GGNN enhances the system's capability to perform deep semantic analysis, improves detection accuracy, reduces false positives, and increases productivity, making it a highly efficient solution for secure software development.

VII. RESULT AND DISCUSSION

The performance of the proposed AI Assisted Secure Code Review and Vulnerability Prediction Platform was evaluated using a diverse set of source code samples containing both secure and vulnerable patterns. The evaluation aimed to measure the effectiveness of the system in detecting vulnerabilities and assisting developers in improving code security.

Figure 2: Home Interface

The experimental results demonstrate that the proposed system successfully identifies common security vulnerabilities such as improper input validation, insecure authentication mechanisms, and unsafe data handling operations. The use of the **Gated Graph Neural Network (GGNN)** enables the system to analyze complex program structures and capture dependencies that are not easily detected by traditional methods. As a result, the system achieves higher detection accuracy compared to conventional static analysis and machine learning approaches.

Figure 3: Code Input Interface

Another important observation is the reduction in false positive rates. By leveraging graph-based representations, the GGNN model effectively distinguishes between actual vulnerabilities and benign coding patterns. This improves the reliability of the system and ensures that developers receive meaningful and actionable insights.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503348|

Figure 4: AI Security Analysis Report

The system also significantly reduces the time required for code review. Developers can submit their code and obtain detailed vulnerability reports within a short duration. These reports provide clear explanations of identified issues along with recommended fixes, enabling faster resolution of security problems.Overall, the results confirm that integrating GGNN into the code review process enhances vulnerability detection performance, improves accuracy, and provides a scalable solution for modern software development environments.

VIII. CONCLUSION

In this research, an AI Assisted Secure Code Review and Vulnerability Prediction Platform has been proposed to improve the efficiency and accuracy of identifying software vulnerabilities during the development process. Traditional code review techniques rely heavily on manual inspection, which is time-consuming and often insufficient for detecting complex vulnerabilities in modern software systems.The proposed system integrates the **Gated Graph Neural Network (GGNN)** to perform graph-based analysis of source code. By representing programs as graphs and applying deep learning techniques, the system captures both structural and semantic relationships within the code. This enables accurate detection of vulnerabilities that are difficult to identify using traditional approaches.

The platform performs key operations such as code preprocessing, graph construction, GGNN-based analysis, vulnerability prediction, and automated report generation. Experimental results demonstrate that the system improves detection accuracy, reduces false positives, and minimizes manual effort in the code review process.Overall, the proposed approach provides an effective and scalable solution for secure software development. The use of GGNN enhances the system's ability to understand complex program logic and contributes to building more reliable and secure applications.

IX. FUTURE ENHANCEMENT

Although the proposed system provides an effective solution for automated vulnerability detection, several enhancements can be made to further improve its performance and capabilities. One potential improvement is the integration of advanced graph-based models such as Graph Transformers, which can capture even more complex relationships in code structures and improve prediction accuracy.

The system can also be extended to support a wider range of programming languages and frameworks, making it more versatile for real-world applications. Additionally, integration with Continuous Integration and Continuous Deployment (CI/CD) pipelines can enable automated security analysis during the development lifecycle.Future work may also include real-time vulnerability detection within Integrated Development Environments (IDEs), allowing developers to receive instant feedback while writing code. Furthermore, combining static graph-based analysis with dynamic analysis techniques can enhance the system's ability to detect runtime vulnerabilities.

With these enhancements, the platform can evolve into a comprehensive intelligent security system that provides robust, real-time, and highly accurate vulnerability detection for modern software development environments.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503348|

REFERENCES

[1] Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2018). VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. *Network and Distributed System Security Symposium (NDSS).*

[2] Zhou, Y., Liu, S., Siow, J., Du, X., & Liu, Y. (2019). Devign: Effective Vulnerability Identification by Learning Program Semantics. *Advances in Neural Information Processing Systems (NeurIPS).*

[3] Russell, R., Kim, L., Hamilton, L., et al. (2018). Automated Vulnerability Detection in Source Code Using Deep Learning. *IEEE International Conference on Machine Learning and Applications.*

[4]Chakraborty, S., Krishna, R., Ding, Y., & Ray, B. (2020). Deep Learning Based Vulnerability Detection: Are We There Yet? *IEEE Transactions on Software Engineering.*

[5] Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why Developers Don't Use Static Analysis Tools to Find Bugs. *International Conference on Software Engineering.*

[6] Shin, Y., Meneely, A., Williams, L., & Osborne, J. (2011). Evaluating Software Metrics as Indicators of Vulnerabilities. *IEEE Transactions on Software Engineering.*

[7] OWASP Foundation (2021). OWASP Top 10: Web Application Security Risks. OWASP Documentation.

[8]Allamanis, M., Barr, E., Devanbu, P., & Sutton, C. (2018). Machine Learning for Source Code: A Survey. *ACM Computing Surveys.*

[9] Dam, H. K., Tran, T., Pham, T., et al. (2021). Deep Learning Approach for Automated Software Vulnerability Detection. *Empirical Software Engineering.*

[10] McGraw, G. (2006). Software Security: Building Security In. Addison-Wesley Professional.

[11] Howard, M., & Lipner, S. (2006) The Security Development Lifecycle. Microsoft Press.

[12] Sutton, M., Greene, A., & Amini, P. (2007). Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley Professional.

[13] Kim, D., Nam, J., Song, J., & Kim, S. (2013). Automatic Patch Generation Learned from Human-Written Patches. *International Conference on Software Engineering.*

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details