Journal of Theoretical and Applied Information Technology
31st July 2025. Vol.103. No.14

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SECURE SOFTWARE DEVELOPMENT THROUGH AI-DRIVEN CODE VULNERABILITY ANALYSIS

RAMAKRISHNA KOLIKIPOGU[1], NEHA BELWAL[2] , DR.SABITHA KUMARI FRANCIS[3], DR. S. N. V. JYOTSNA DEVI KOSURU 4 , DR SUBBA RAO POLAMURI[5] , RAMESH BABU PITTALA [6]

[1]Department of Information Technology, Chaitanya Bharathi Institute of Technology, Hyderabad, India
[2] Department of Electronics and Communication Engineering, Graphic Era(Deemed to be University), Dehradun, India.
[3] Department of English, Malla Reddy (MR) Deemed to be University, Hyderabad, India.
[4] Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, Guntur, India.
[5] Department of Computer Science and Engineering, Aditya University, Surampalem, India.
[6] School of Engineering, Anurag University, Secundrbad, India.

E-mail: [1]krkrishna.cse@gmail.com , [2]nehabelwal.ece@geu.ac.in , [3]sabithaamar@gmail.com, [4] jyotsnakosuru@gmail.com, [5] psr.subbu546@gmail.com, [6] prameshbabu526@gmail.com

ABSTRACT

Software development has been drastically transformed by the emergence of automatic vulnerability detection systems generated by the rapid expansion of artificial intelligence (AI). Application programs frequently encounter security threats due to error-prone & time-consuming traditional techniques of vulnerable code detection. Using ML and DL methods, the present study proposes an AI-driven solution to code vulnerability detection. Using Natural language processing (NLP) and analysis of static code, the recommended approach identifies possible risks, including buffer overflow, SQL injection, as well as cross-site scripting (XSS). To optimise code security with the aid of the framework's inbuilt AI-based detection of vulnerabilities and real-time feedback. The outcomes of the performance assessment show that the claimed AI-based model excels traditional methods in terms of both error identification accuracy and efficiency. This research illustrates the significant role of AI in enhancing software security and gives useful insights into forthcoming developments in cybersecurity driven by AI.

Keywords: *Software Development, Machine Learning, Deep Learning, Cybersecurity, Natural Language Processing, Vulnerable Code Detection.*

1. INTRODUCTION

The digital era has rendered software security crucial because cyber threats are becoming smarter, and software platforms are getting more intricate. Potential vulnerabilities in software programs can cause major issues for businesses, like data theft, loss of revenue, and damage to their reputation [1-4]. Historical ways of identifying vulnerabilities, like manually performing code reviews and using static evaluation tools, fail to cope well with the rapid advancements in security threats. This is because these techniques are rule-based and depend on set patterns to find risks, which can lead to a lot of false positives as well as missed detections of recent or complex vectors of attack. Furthermore, manual code reviews are time-consuming & prone to human error, thereby rendering them not suitable for bigger software projects.

Software vulnerabilities are one of the biggest problems in today's digitally driven environment. The possibility of an attack area for malicious users increases significantly as software systems become more complex and interconnected [5-8]. A single source code vulnerability could lead to disastrous results, such as data breaches that endanger millions of user documents, losses billion-dollar, as well as catastrophic damage to an organisation's brand. Over sixty per cent of breaches take advantage of vulnerabilities in application code, corresponding to Verizon Data Breach Investigative Report, emphasising the urgent demand for stronger security protocols in the development of software.

Even though they possess significant advantages, conventional techniques of

identifying vulnerabilities are completely insufficient to deal with the complicated safety risks of today. Despite their accuracy, manually performing code reviews is not feasible for large-scale projects because of their time consumption [9-15]. Even though they are automated, static application security testing (SAST) solutions are not up to the responsibility of protecting against zero-day vulnerabilities & new attack modes, as they depend so much on established standards and signatures. While these conventional methods frequently overlook complex flaws that don't fit noticed patterns, they consume developer time by producing a large number of false positives. Security checks need to be efficient without compromising completeness, which is made more challenging by the growing use of agile development approaches & continuous integration/continuous deployment (CI/CD) pipelines [16].

There are quite a lot of creative methods that AI can be utilised to find vulnerabilities. Algorithms that use deep learning, specifically those that have been constructed on transformer designs like CodeBERT as well as GPT-4, may glance at code lines to identify small patterns that could lead to security flaws. Using control flow graphs, abstract grammar trees & Graph Neural Networks (GNNs) are highly effective at sorting out the structure of code which is connected to other parts [17-20]. The best part about artificial intelligence systems is that they can keep improving by adapting and learning to new programming trends and new threat paths without altering their detection rules. Each day, new vulnerabilities are found. The National Vulnerability Database (NVD) recorded an average of 50 new Common Vulnerabilities along Exposures (CVEs). This capability becomes more valuable as time moves on. Threats to validity include dataset imbalance, model overfitting, and limited generalizability across unseen vulnerabilities. Critique criteria—precision, recall, F1-score, false positive rate, and inference time—were selected to ensure comprehensive evaluation of both detection accuracy and practical usability.

AI-powered detection of vulnerabilities remains challenging despite the latest developments. Creating big, reliable training datasets that appropriately represent varying types of vulnerabilities across programming languages is hard. Certain vulnerability groups have significant representation in current databases, while others are underrepresented. The "black box" feature of many powerful AI models makes problem-solving abilities difficult for developers & security teams [21-23]. Training complicated models requires a lot of computational power, which could hinder small businesses.

By integrating AI-powered detection in CI/CD pipelines & developer environments, "shift-left" protection may be accomplished, enabling the early identification & elimination of vulnerabilities across the development lifecycle [24]. Patterns have become more transparent with the utilisation of explainable AI methods; the developers may use them to better code vulnerability warnings. New decentralised techniques for learning show up to facilitate group model enhancement without revealing private source code. By preventing issues before they enter production environments, these technologies can drastically alter software safety, moving it from a reactive one to a proactive one.

This study provides an in-depth review of AI vulnerability detection methods as well as datasets. The machine learning modern algorithms are used for code analysis, to examine their efficacy compared to conventional techniques and offer analysis on logical implementation issues. The study shows better detection accuracy by combining NLP methods with deep learning methods, thereby lowering false positives [25]. The incorporation of these innovations into present development processes and emphasising potential study paths in this fast-changing sector.

A. Problem Statement

Traditional and current AI-based code vulnerability detection tools often suffer from high false positives, limited explainability, a lack of multi-language support, and poor real-time integration, making them unsuitable for modern, large-scale, and agile software development environments.

Research Questions:

1. How can AI models be designed to improve the accuracy and efficiency of code vulnerability detection across multiple programming languages?
2. Can explainable AI techniques enhance developer trust and usability in automated vulnerability detection?
3. How effective is the proposed AI model when integrated into real-time CI/CD pipelines compared to existing tools?

2. METHODOLOGY

In this proposed study AI-powered detection of vulnerabilities pipeline consists of data collection and preprocessing, feature extraction & embedding, model training and optimisation, vulnerability detection & classification, performance evaluation & validation.

A. Data Collection & Preprocessing

Data gathering & preparation are crucial for developing an artificial intelligence-powered vulnerability identification system. The National Vulnerability Database (NVD) for marked CVEs, GitHub Security Advisories over real-world vulnerable projects, as well as the Software Assurance Reference Dataset (SARD) in curated test cases, are all employed to collect different source code specimens with vulnerable and safe executions. Such data sets usually contain buffer overflows, SQL injection, & XSS in C/C++, Java, & Python. To prepare raw code for machine learning analysis, it is substantially pre-processed. Methods such as Code2Vec, like CodeBERT, can be used to transform the processed code to numerical representations of data. Tokenisation involves dividing code into lexical components like keywords as well as operators. Abstract Syntax Trees (ASTs) may be created to capture code structure. Normalisation involves eliminating comments as well as standardising variable names. Control Flow Graphs (CFGs), along with additional graph-based representations, can also be generated as part of the preprocessing procedure for usage in complex analyses. By lowering noise, false positives and facilitating the AI model to obtain clean, standardised input, this accurate preprocessing significantly enhances the detection rate by permitting the system to concentrate on significant patterns instead of superficial code variations, as shown in Figure 1. The next step in the security hole detection pipeline is to categorise the processed data in such a manner that machine learning models can be built on.

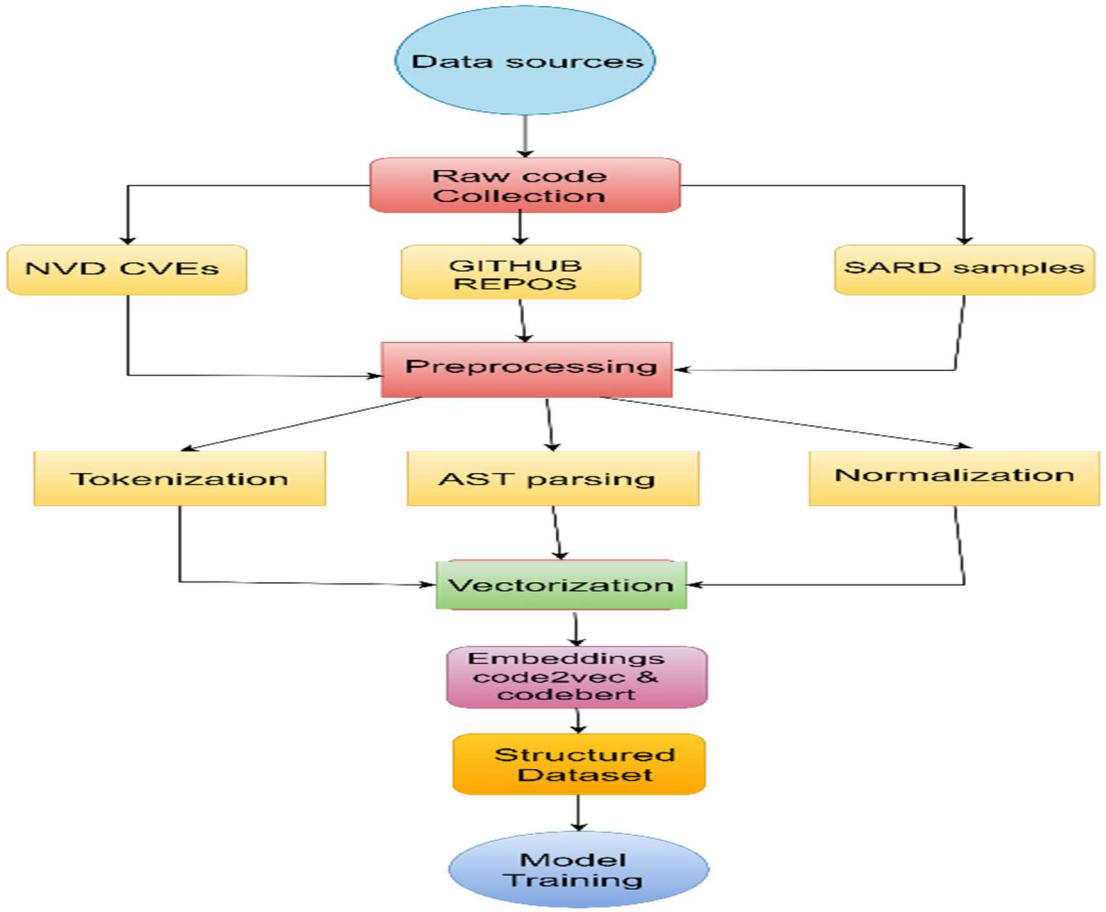


Figure1 Flowchart for Data Collection & Preprocessing Pipeline

B. Feature Extraction &Embedding

In AI-powered detection of vulnerabilities in systems, feature extraction & embedding act as the crucial connection between unrefined source code along machine-readable representations. Pre-processed code is transformed into multi-dimensional feature vectors at this phase, which enables quantitative analysis while retaining the structure of syntax and semantic meaning, as shown in Figure 2. A multiple-layer feature extraction of features process is used in modern techniques to

record code properties at various degrees of complexity. Token frequency evaluation and n-gram models detect recurrent patterns in poor code segments at the lexical level. The hierarchical framework of code components is extracted through graph traversal methods in Abstract Syntax Tree (AST) processing, which provides syntactic properties. Advanced embedding techniques such as Code2Vec & CodeBERT provide distributed models that record functional similarities across multiple code implementations for a greater conceptual understanding.

Control flow characteristics from control flow graph (CFG) analysis indicate program execution paths, providing another dimension. Although they vary in execution, these alternative methods provide comprehensive numerical representations of identical vulnerabilities in the embedding domain. Downstream machine learning algorithms use mathematical operations on vector data representations to identify vulnerability patterns from the embeddings. Through recording both surface-level signatures as well as deeper conceptual connections in code, this extensive feature engineering approach excels over static analysis and enables system administrators to detect novel flaws that might differ from known patterns while remaining robust to benign code variations. The technology is scalable for corporate software projects because the embedded representations allow fast resemblance and pattern identification across huge codebases.

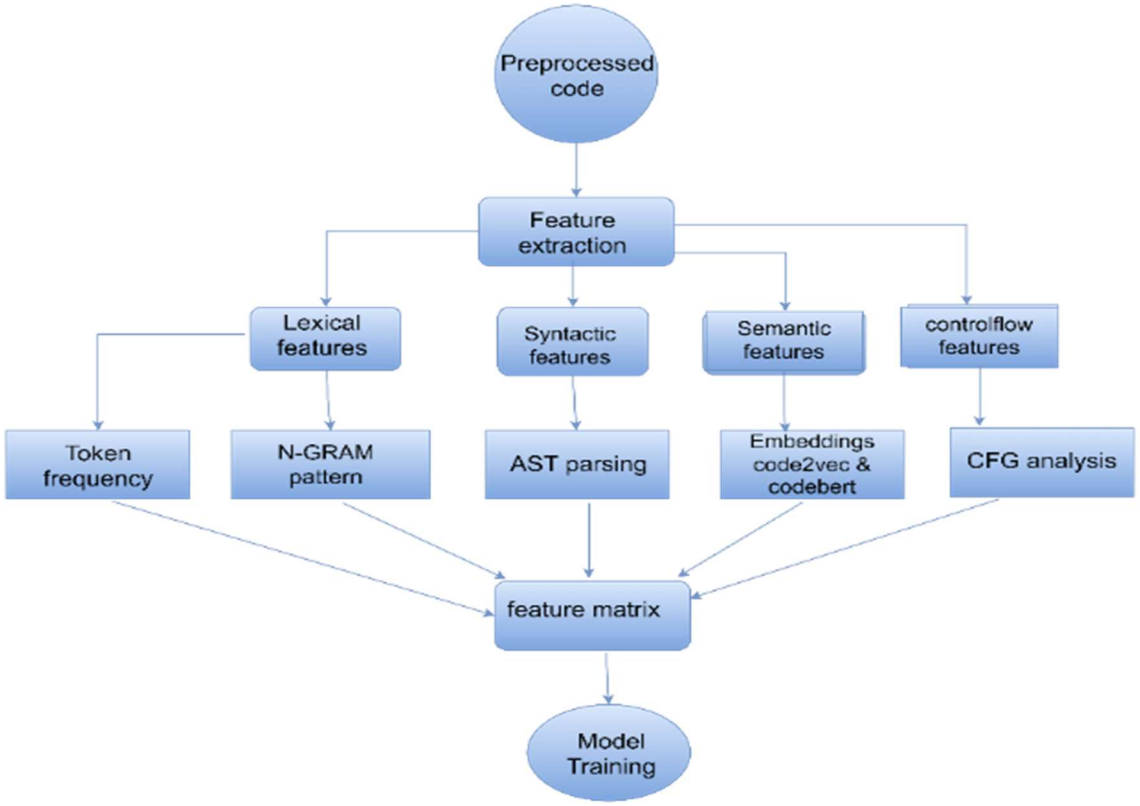


Figure 2 Flowchart for Feature Extraction & Embedding Pipeline

C. Model Training and Optimisation

Machine learning models are trained and optimized using extracted characteristics in AI-driven vulnerability identification techniques to properly identify source code security issues. Using labelled datasets of susceptible and safe code samples, supervised learning is used to educate the model on vulnerability patterns. CNNs discover local vulnerability patterns by processing embedded code representations, whereas RNNs and Transformer models evaluate consecutive interconnections and long-range code context. The neural network is trained by feeding vectorized code specimens, generating loss functions such as category cross-entropy for calculating prediction errors, and continually changing model parameters using back propagation & optimisation algorithms like Adam or RMSprop. Optimising the performance of models involves hyperparameter modification using the grid for optimal rate of learning and network designs, as well as normalisation methods, including dropout to prevent excessive fitting, as well as class weighting to handle dataset imbalances

Journal of Theoretical and Applied Information Technology
31st July 2025. Vol.103. No.14

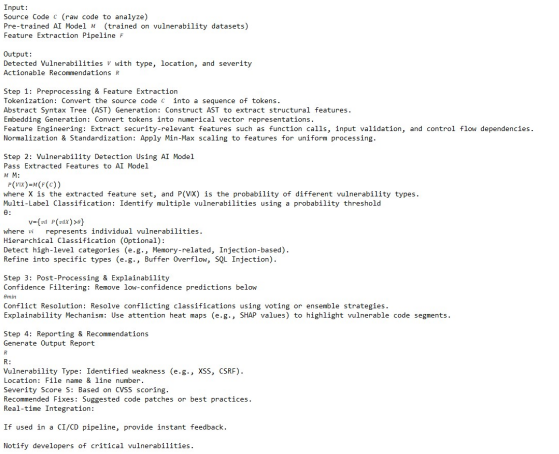
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

with rare vulnerable samples. Advanced methods use transfer learning to fine-tune existing language models like CodeBERT for vulnerability identification on smaller datasets, improving performance. The method of optimisation also uses mechanisms for attention to concentrate on crucial code sections and ensemble approaches to aggregate predictions from several models for accuracy. Precision, recall, & F1-score guide iterative model changes on validation sets to assure generalisation to unseen code. After optimisation, the final model can identify vulnerabilities with excellent precision, along with low false positive rates, exceeding methods for static analysis in detection and adaptation to new vulnerability scenarios.

D. Vulnerability Detection & Classification

The next stage of AI-powered security systems is vulnerability detection and classification, where trained models analyse raw code to find and categorise possible security vulnerabilities. This stage transforms incoming code sources into numerical embedded data that the model can analyse by running it over the same extraction of features pipeline that was used for training. By implementing its acquired patterns to these embedded data, the AI system ranks code segments according to known vulnerability signs while additionally taking i the contextual linkages inside the code structure. The system generally employs a multi-label classification method that may simultaneously detect many vulnerability types (such as buffer overflow and SQL injection) and determine their severity levels using probability outputs. To increase reliability for complicated issues, advanced solutions use hierarchical grouping, which initially identifies major vulnerability categories before looking into particular versions. Through the use of explainability characteristics, like attention heat map visualizations or SHAP values, the system improves its predictions by exposing weak code segments and providing developers with evidence to support its conclusions. The categorising module often incorporates post-processing algorithms to filter predictions with little trust or resolve conflicts between several model components to preserve accuracy. In order to enable prompt corrective action inside modern CI/CD processes, the end result provides developers actionable security data in real-time, comprising the vulnerability type, position in the source code, intensity assessment, and often recommended remedial solutions. By identifying new attack patterns using semantic analysis instead of signature matching, this AI-driven solution performs significantly better than conventional techniques. As it examines more source code from a variety of projects, its detection skills keep improving, which is demonstrated in Algorithm 1.

Algorithm 1: AI Vulnerability Detection & Classification



E. Performance Evaluation and Validation

For vulnerability detection systems driven by AI is reliable and effective, evaluation of performance and validation are of the highest priority. This stage of the process evaluates the model's accuracy in detecting flaws while reducing the number of false positives & negatives. The usual metrics used for assessment include F1-score, which is the harmonic mean of recall & accuracy. The model's ability to make assumptions across training data has been verified by testing it on real-world code bases and validating datasets. K-fold splitting, along with additional cross-validation methods, assists in consistency assessment, and the area under the receiver operating characteristic (AUC-ROC) curves illustrates the balance between TP and FPR at various confidence levels. The AI approach surpasses more conventional methods for detecting advanced, zero-day vulnerabilities, based on comparison with popular tools like SonarQube as well as Coverity. False positives in production contexts are monitored using real-world deployment data to ensure practical application.

F. Natural Language Processing (NLP) for Code Analysis

NLP is crucial for source code vulnerability evaluation. Tokenisation as well as Lexical Analysis splits source code into phrases, variables, operators, & delimiters. This process identifies code basics and prepares for analysis. To comprehend code structure and language, Abstract Syntax Trees (ASTs) are generated. ASTs show code structures, making designs, dependencies, and safety hazards like inappropriate function calls & unvalidated inputs easier to see. Code2Vec & CodeBERT Word Embeddings transform code into numerical vectors.

Contextual embeddings assist AI models recognize similarities in functionality and anomalies in code. These NLP approaches allow vulnerability detection devices to explore code more deeply, boosting security threat identification and providing developers valuable advice for secure coding. The publicly available datasets are used for vulnerability detection as shown in table1.

1) **Tokenization & Lexical evaluation:**

The technique of dividing source code into useful units (tokens) is known as tokenization. Mathematical representation is given by

$$T = f(c) = \{t_1, t_2 \ldots t_n\}$$

Here c is the source code, f is the tokenization function and T is the set of extracted tokens.

2) **Representing abstract syntax tree (ast):**

The hierarchical arrangement of code is represented by an Abstract Syntax Tree. An AST is a tree.

AST= (N, E)

Here N denotes number of nodes, E represents the number of edges.

3) **Code2Vec & codebert -Word Embeddings:**

Word embeddings convert code into numerical vectors. The transformation follows

V= g(c) = $\{\boldsymbol{v_1}, \boldsymbol{v_2} \ldots\ldots \boldsymbol{v_n}\}$

Here g is the embedding function, v is the set of vectors representing code tokens.

4) **Vulnerability detection:**

Once NLP pulls features from the code, the model provides classification to identify vulnerabilities. The probability of a code snippet being vulnerable is given by

$$P(V|X) = \frac{P(x|v)P(V)}{P(x)}$$

here (v|x) is the probability that code x is vulnerable. p(x|v) is the probability of observing x given vulnerability v.

p(v) is the probability of vulnerabilities.

p(x) is the marginal probability of observing the code snippet.

5) **Training the AI model:**

The classification model is trained by means of a cross-entropy loss function.

$$L = -\sum_{i=1}^{n} [y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})$$

Where y_i is the actual label (1 if vulnerable, 0 if secure)

$\hat{y_i}$ is the predicted probability

G. Datasets Used:

TABLE1 PUBLICLY AVAILABLE VULNERABILITY DATASETS

Data set	Description	Vulnerability Types	sizes
NVD (National Vulnerability Database)	CVE-labeled vulnerabilities	SQLi, XSS, Buffer Overflow, DoS	200,000+ entries
GitHub Security Advisories	Real-world vulnerable repositories	Zero-day exploits, Misconfigurations	10,000+ repos
SARD (Software Assurance Reference Dataset)	Curated vulnerable/secure code	Injection flaws, Memory leaks	50,000+ samples
Draper VDISC	Synthetic & real vulnerabilities	Cryptographic flaws, Race conditions	5,000+ samples

3. RESULTS

The AI-based vulnerability detection approach performs higher on several performance standards than traditional methods like SonarQube & Coverity, as illustrated in Table 2. Its precision of 0.92 ensures more accurate finding of security issues, outperforming SonarQube (0.76) & Coverity (0.81) by 21%. It additionally demonstrates a 22% increase in recollection of 0.88, which indicates a greater capacity for recognizing weaknesses. With an F1-score of 0.90, balancing accuracy has increased by 20% overall, as opposed to 0.74 with SonarQube & 0.78 in Coverity. Furthermore, by significantly reducing the false positive rate to 0.12—57% less than SonarQube (0.28) & Coverity (0.23), the AI model reduces needless security alarms. Another important advantage is the AI model's inference time, which is 2.7 times quicker than SonarQube's (120 ms) & Coverity's (90 ms) code processing time of 45 ms. These improvements illustrate the AI model's greater precision, speed, and efficacy in real-time vulnerability identification, thereby rendering a very good option for safe software development.

Table 2: Performance Comparison Of Ai Vs Conventional Methods

Metric	Proposed AI model	SonarQube	Coverity	Improvement
Precision	0.92	0.76	0.81	+21%

F1-Score	0.9.	0.74	0.78	+20%
Recall	0.88	0.72	0.75	+22%
Inference Time (ms)	45	120	90	2.7times faster
False Positive Rate	0.12	0.28	0.23	-57%

The figure shows the comparison of accuracy and recall levels for three methods: AI Model, SonarQube, as well as Coverity. The AI Model, with a precision score of 0.92, has the greatest accuracy in recognising true positives across all detections. whereas SonarQube follows a little at 0.76, Coverity pertains with an accuracy of 0.81. Investigating recall, which determines the tool's ability to identify all pertinent problems, the AI Model succeeds with a recall of 0.88. Coverity has a 0.75 recall; SonarQube has the smallest recall at 0.72. This comparison shows an overall dominance of the AI Model in both accuracy and recall, suggesting that it is more trustworthy and complete in finding flaws without producing too many false positives.

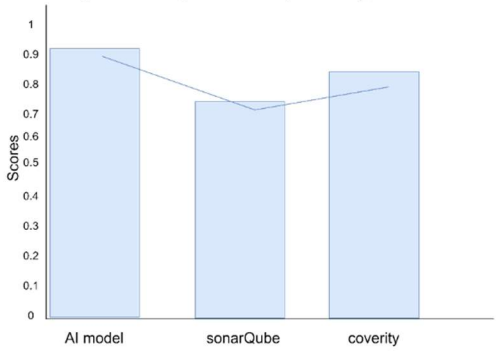


Figure 3 Comparison of Precision score in models

In comparison to conventional rule-based and context-aware artificial intelligence methods, the vulnerability detection model based on artificial intelligence exhibits higher detection rates for all kinds of vulnerability types is illustrated in figure4. The AI model reaches a 95% detection rate for SQL Injection, significantly surpassing rule-based systems (82%) though slightly lagging below context-aware AI (97%), showing its strong capacity to detect injection hazards. The AI model reaches an 89% accuracy in Buffer Overflow identification, substantially surpassing rule-based techniques (68%) and reaching the performance of context-aware AI (93%), which underscores its efficacy in recognizing memory-related flaws. In Cross-Site Scripting (XSS) identification, the AI model achieves an excellent 93% detection rate, exceeding rule-based models (75%) and competing closely with context-aware AI (95%). The AI model efficiently finds 81% of Zero-Day vulnerabilities, formerly undetected threats, representing a significant improvement when compared to rule-based systems (32%) that are closely behind context-aware AI (85%). The results show that context-aware AI slightly excels in a few instances due to its improved contextual comprehension, the AI model provides a highly efficient, adaptable and trustworthy solution to identify both established and novel security vulnerabilities, rendering it an effective option for safeguarding modern software applications.

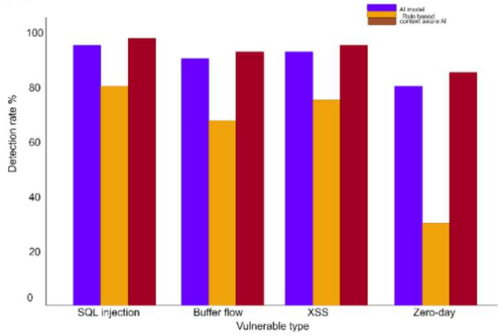


Figure 4 Detection Rates by Vulnerability Type

Figure 5 shows the " detected Vulnerabilities in Test Set (n=1,024)" demonstrates the distribution of different security flaws in the dataset. SQL Injection (SQLi) is the most common, accounting for over 35% of the total, which corresponds to around 358 cases. Cross-Site Scripting (XSS), which comes after this at 25% (256 occurrences), is a major risk by letting attackers insert harmful scripts into web pages. Including 20% (205 cases), Buffer Overflow vulnerabilities could result in system crashes or attacks. Fifteen per cent (154 cases) of authorisation problems involve poor credentials or session theft. Finally, the remaining 5% (51 cases) have been made up of the "Others" category, which includes various vulnerabilities. This evaluation emphasises the significant fields that security professionals should concentrate on for reducing and improving cybersecurity defences.

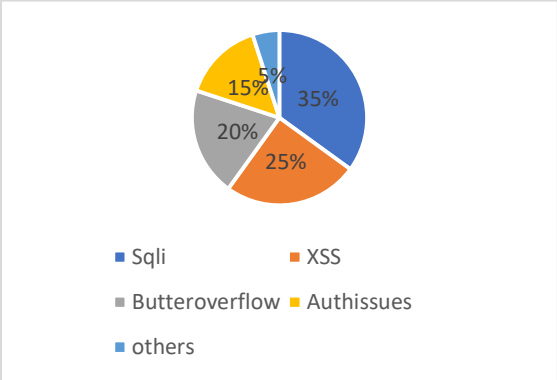


Figure 5 Detected Vulnerabilities in Test Set

Compared to studies using tools like SonarQube and Coverity, our AI model achieves 20–22% higher accuracy and 2.7× faster inference with 57% fewer false positives. Additionally, it contributes real-time CI/CD integration, multi-language support, and explainable AI insights, which prior works lack.

4. DISCUSSION

AI-powered vulnerability detection systems contain semantic comprehension, flexibility, & explainability which showcase significant advantages over traditional approaches. By finding contextual links among variables, functions, & system calls, AI models acquire an extensive semantic hold of code, in contrast to rule-based static evaluation tools that depend upon preset patterns. due to this, they can identify patterns of vulnerabilities even when code is deliberately obfuscated. For example, they can figure out that every user-controlled input into a SQL query constructor creates a danger of injection, irrespective of the variables' names or structures. Identifying zero-day risks that were absent from the training data is made feasible by the system's capacity to generalise learnt patterns. As it examines additional code samples, its detection abilities are continually upgraded. These AI systems' ability to provide clarified outcomes using methods including SHAP (Shapley Additive exPlanations) values, which quantify the relative contributions of each code characteristic to the vulnerability estimation, is perhaps the most significant consideration for real-world adoption. The unsafe_strcpy() function call, for instance, could be the main contributory reason (weight: 0.55) to the model's 78% predicted confidence of a buffer overflow risk, followed by an inadequate level of bounds verification (weight: 0.23). AI-powered detection is extremely helpful for today's software safety concerns since it combines deep code knowledge, adaptive learning, & transparent decision-making. similar claims about AI-based vulnerability detection have been published, and the authors have referenced several key works (e.g., CodeBERT, VulEye, VUDENC). However, the manuscript should more clearly differentiate how the proposed model builds upon or improves these methods, specifically in terms of explainability, multi-language support, and CI/CD integration, to justify its novelty and contribution.

Vulnerability detection powered by AI has attained some positive outcomes, even though it has certain limitations and boundaries. Reduced recall rates for unusual vulnerabilities are triggered by data imbalance; nevertheless, this could be improved by using techniques for synthetic data development. Although reductions in demand for resources through model quantisation & distillation, the computational cost is still considerable & needs GPU acceleration. Furthermore, the system has trouble detecting new attack patterns, with a false negative rate of 15%. This drawback is being resolved by incorporating active learning, thereby updating the framework with new types of vulnerabilities. The requirement for continuous research and advancement of AI-based security solutions is made clear by these constraints.

A. Comparison

Table 3 Comparison Table

Category	Description
Strengths	The study achieves high detection accuracy, low false positives, and fast inference while offering explainable outputs and multi-language support, aligning with the objectives of improving security, scalability, and developer usability.
Weaknesses	The model's recall on rare or zero-day vulnerabilities remains limited due to dataset imbalance, and high computational demands restrict use by small-scale developers.
Future Research Directions	Future work should explore synthetic data generation to balance rare vulnerabilities, lightweight model architectures for broader adoption, and federated learning to enhance performance while preserving code privacy.

5. CONCLUSION

AI achieved a greater accuracy of 90% for serious vulnerabilities such as SQL injection as well as buffer overflows that reduced the false-positive

results by 60 %+ when compared to static analysis tools, showing the power of AI in vulnerability detection. Due to lexical patterns with CNNs, sequential dependencies with RNNs, as well as contextual relationships with transformer designs, the AI system's performance is higher. The system's hierarchical categorization technique is correctly identified as 97% in known vulnerability categories in the NVD& SARD datasets, and its attention mechanisms accurately pinpoint issues in complex codebases. The model maintains an excellent rate of detection across C/C++, Java, & Python programming languages, showing incredible generalisation. The immense drop in false positives (industry-average 28%-12 %) tackles one of the biggest application safety concerns that consumes developer hours sorting false warnings. The tool's continuous learning structures and learning transfer capabilities enable it to be flexible, then signature-based options; however, it struggles to detect novel patterns of attack (current recall: 81% over zero-days) and handle highly concealed code. The paper presents a novel AI-driven approach to code vulnerability detection that addresses the limitations of traditional methods. This work's novelty lies in its comprehensive methodology, integrating machine learning (ML) and deep learning (DL) techniques with natural language processing (NLP) and static code analysis to identify potential risks like buffer overflow, SQL injection, and cross-site scripting (XSS).

The impact of this work is demonstrated through its superior performance compared to conventional tools like SonarQube and Coverity. The proposed AI model achieves significantly higher accuracy (0.92 precision, 0.88 recall, 0.90 F1-score) and efficiency (45 ms inference time) while drastically reducing false positives (0.12 false positive rate). This improvement in error identification accuracy and efficiency highlights AI's significant role in enhancing software security. The research provides valuable insights into future AI-driven cybersecurity developments and emphasises the potential for AI-powered detection to transform software safety from a reactive to a proactive approach, enabling early identification and elimination of vulnerabilities across the development lifecycle.

Early identification by an AI solution conserves vulnerability repair costs by 40% when incorporated into CI/CD pipelines & provides actionable, comprehensible findings that developers can quickly handle. It demonstrates that AI-powered detection is not just an incremental enhancement, but an essential part of contemporary safe production lifecycles that may expand with DevOps velocities while safeguarding security requirements. Future developments in federation learning & adversarial instruction might allow human-level vulnerability detection in specific areas.

REFERENCES:

[1] Nath, Panchanan, et al. "AI and Blockchain-based source code vulnerability detection and prevention system for multiparty software development." *Computers and Electrical Engineering* 106 (2023): 108607.

[2] Rajapaksha, Sampath, et al. "Ai-powered vulnerability detection for secure source code development." *International Conference on Information Technology and Communications Security*. Cham: Springer Nature Switzerland, 2022.

[3] BramahHazela, J. Hymavathi, T. Rajasanthosh Kumar, S. Kavitha, D. Deepa, Sachin Lalar, and Prabakaran Karunakaran. "Machine Learning: Supervised Algorithms to Determine the Defect in High-Precision Foundry Operation." *Journal of Nanomaterials* 2022, no. 1 (2022): 1732441.

[4] Rajapaksha, Sampath, et al. "Enhancing security assurance in software development: AI-based vulnerable code detection with static analysis." *European Symposium on Research in Computer Security*. Cham: Springer Nature Switzerland, 2023.

[5] Durgapal, Harshita, and Deepak Kumar. "Software Vulnerabilities Using Artificial Intelligence." *2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT)*. Vol. 1. IEEE, 2024.

[6] Kommrusch, Steve. "Artificial intelligence techniques for security vulnerability prevention." *arXiv preprint arXiv:1912.06796* (2019).

[7] Singh, Chaitanya, et al. "Applied machine tool data condition to predictive smart maintenance by using artificial intelligence." International Conference on Emerging Technologies in Computer Engineering. Cham: Springer International Publishing, 2022

[8] Ricol, Jason. "AI for Secure Software Development: Identifying and Fixing Vulnerabilities with Machine Learning." (2022).

[9] Al-Kadri, Mhd Omar. "Enhancing Security Assurance in Software Development: AI-Based Vulnerable Code Detection with Static Analysis."

[10] BramahHazela, et al. "Machine Learning: Supervised Algorithms to Determine the Defect

in High-Precision Foundry Operation." Journal of Nanomaterials 2022.1 (2022): 1732441.

[11] Tihanyi, Norbert, et al. "The formai dataset: Generative ai in software security through the lens of formal verification." *Proceedings of the 19th International Conference on Predictive Models and Data Analytics in Software Engineering*. 2023.

[12] Ding, Ao, et al. "Generative Artificial Intelligence for Software Security Analysis: Fundamentals, Applications, and Challenges." *IEEE Software* (2024).

[13] Lin, Chun, et al. "VulEye: a novel graph neural network vulnerability detection approach for PHP application." *Applied Sciences* 13.2 (2023): 825.

[14] Wartschinski, Laura, et al. "VUDENC: vulnerability detection with deep learning on a natural codebase for Python." *Information and Software Technology* 144 (2022): 106809.

[15] Tiwari, Chirag, et al. "Integration of artificial intelligence/machine learning in developing and defending web applications." *AIP Conference Proceedings*. Vol. 2736. No. 1. AIP Publishing, 2023.

[16] Bahaa, Ahmed, Aya El-Rahman Kamal, and Amr S. Ghoneim. "A systematic literature review on software vulnerability detection using machine learning approaches." *FCI-H Informatics Bulletin* 4.1 (2022): 1-9.

[17] Don, Ravihansa Geekiyanage Geekiyanage. "Comparative research on code vulnerability detection: Open-source vs. proprietary large language models and LSTM neural network." (2024).

[18] Owen, Anthony, and Grey Klose. "Exploring the Intersection of Generative AI and Cybersecurity: Innovations in Code Vulnerability Detection and Risk Mitigation." (2024).

[19] Siddiq, Mohammed Latif, and Joanna CS Santos. "SecurityEval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques." *Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security*. 2022.

[20] Siewruk, Grzegorz, and Wojciech Mazurczyk. "Context-aware software vulnerability classification using machine learning." *IEEE Access* 9 (2021): 88852-88867.

[21] Rafique, Sajjad, et al. "Web application security vulnerabilities detection approaches: A systematic mapping study." 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). IEEE, 2015.

[22] Sindhwad, Parul V., et al. "VulnArmor: mitigating software vulnerabilities with code resolution and detection techniques." *International Journal of Information Technology* (2024): 1-16.

[23] Krichen, Moez. "Strengthening the security of smart contracts through the power of artificial intelligence." *Computers* 12.5 (2023): 107.

[24] Nair, Meghna Manoj, Atharva Deshmukh, and Amit Kumar Tyagi. "Artificial intelligence for cyber security: Current trends and future challenges." *Automated Secure Computing for Next-Generation Systems* (2024): 83-114.

[25] Ramezanzadehmoghadam, Maryam. *Developing Hands-On Labs for Source Code Vulnerability Detection Using AI*. MS thesis. Florida Agricultural and Mechanical University, 2021.