Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 10, 2025

CODE-SEMANTIC BASED INTELLIGENT VULNERABILITIES DETECTOR

Shumaila Hussian[*1], Muhammad Imran Ghafoor[2], Paras Perviaz[3], Shariqa Fakhar[4]

[*1]*Computer Science Department, Sardar Bhadur Khan Women's University, Quetta, Pakistan*
[2]*Department of Information Security, PUCIT Punjab University, Lahore, Pakistan*
[3]*Department of Computer Science, Balochistan University of Information Technology, Engineering and Management Sciences, Quetta, Pakistan*
[4]*Computer Science Department, Sardar Bhadur Khan Women's University, Quetta, Pakistan*

DOI:https://doi.org/10.5281/zenodo.17275526

Keywords
Vulnerability Detection, Deep Learning, Feature Extraction, BiLSTM, Software Security

Article History
Received: 11 July 2025
Accepted: 21 September 2025
Published: 06 October 2025

Corresponding Author: *
Shumaila Hussain

Abstract
The presence of vulnerabilities in a software system can significantly compromise its security, underscoring the importance of automatic software vulnerability detection. Current detection techniques, however, grapple with issues such as dependency problems, lexical bias, and low detection resolution. Though machine learning has been widely studied for predictive applications, its use in vulnerability detection has been hampered by lack of semantic information in deep learning techniques, which results in diminished accuracy. The study proposes an innovative automatic vulnerability detection system that synthesizes feature selection and deep neural networks (BiLSTM) to mitigate the challenges. This system selects features based on novel code metric that incorporates control flow and suspected functions, with feature selection implemented using a decision tree. The model is trained using standardized dataset (SARD) and is capable of detecting improper input validation, SQL injection attacks, cross-site scripting attacks, missing authorizations, and buffer overflow attacks. We compare our system's performance in terms of precision, recall, accuracy, and area under the receiver operating characteristics curve (AUC-ROC) with various techniques and commercial systems. The empirical evaluation demonstrates that our proposed system achieves a 4% improvement over commercial systems and a 6% improvement over other machine learning-based systems.

INTRODUCTION

During the COVID-19 and post pandemic the technological adoption has drastically boosted causing unprecedented increase in software vulnerabilities. There were 20,158 reported security vulnerabilities in 2021 [1]. Managing software security threats is a challenging task due to rapid technological inventions and unexploitable security loop holes. This exponential growth in security threats is causing impactful economical loss globally [2–7]. It has been observed from literature that the existing vulnerabilities detection systems are incapable of effectively analyzing security vulnerabilities, therefore there is an immense need of generalized, resource-adoptive, scalable, granular and effective vulnerability detector.

Vulnerabilities typically stem from programming oversights, and are either syntactic or semantic flaws that creates potential loop holes for the attackers. The automatic vulnerability detectors commonly focus on static or dynamic analyzers. It has been observed through literature that the existing static code analyzers utilizes high resources and the dynamic vulnerability detector have extraordinary execution time and complexity which degrades the efficiency [8].

The described techniques are language dependent, rule-based, and knowledge dependent, and thus are most likely to have errors, expert dependent, biased, and unsustainable that can cause inefficacy.

Deep learning based vulnerability analyzers are surpassing in syntax analysis and code pattern recognition [9–11]. However, they are lacking semantic information of source code and thus reducing efficiency [12–14]. Moreover, their range is narrow, targeting specified and small source code, or rely on APIs. Furthermore, existing techniques brawl to highlight the data transfer among function within source codes as the code semantic are ignored [15]. Thus they appear to be less adoptable and inappropriate for diverse source codes [16–17].

The semantic understanding of the source code is now accessible and applicable because of the availability of open source software repositories. The literature highlights that currently available vulnerability detection techniques ignore semantic understanding of code. The research study therefore focused on both syntactic analysis and code semantic of the source code to elevate the performance. The feature selection is performed using semantic understanding of code using deep learning based decision tree. The study focused a novel code matrix composed of 'suspected functions' and 'control flows' used as features in the proposed model. The Software Assurance Reference Dataset (SARD) dataset is used to validate the model it contains Java code.

The proposed vulnerability detection system analyses improper input validation, SQL injection attacks, cross-site scripting, missing authorization, and buffer overflow attacks.

This research contributes as follow

1. Develops a novel approach integrating semantic information for feature selection to incorporate complex patterns and elevate the effectiveness.
2. Uses BiLSTM (bidirectional long-short-term memory) to fetch the semantic-based information.
3. Employs benchmark datasets, Software Assurance Reference Dataset (SARD) dataset and Juliet java 1.3 to train the model.
4. Provides a comprehensive comparison of the proposed system using CNN (convolutional neural networks) and SVM (support vector machines) and commercially available tools such as SonarQube and Agnitio, signifying performance evaluation metrics.
5. Highlighting the effectiveness of code semantics in detecting software vulnerabilities thus donating to a wider understanding of vulnerability detection and providing a more effective solution.

2. Related Work

Traditional vulnerability detection involves manual source code auditing using security experts [18]. Automatic vulnerability detection using deep learning techniques is prevailing it involves static, dynamic, and hybrid analysis. The literature highlights that these techniques are ill-equipped to detect the frequently developing latest vulnerabilities [19–25]. Furthermore, their effectiveness and reliability is shaking.

The ML based Vulnerability analyzers involve regression trees abstract syntax tree (AST), and adoption of self-attentive deep neural network techniques to detect security flaws [26-29]. The hybrid deep learning-based technique, intermediate representation learning is among the implementing strategies as well [30-31]. Similarly, BGRU encapsulated with program slicing was introduced as well incorporating code metrics for features selection are developed to detect the vulnerable code [32-33]. Other ML based algorithm CNN along with feature engineering is used to analyze vulnerabilities [34-38]. The feature selection using ML techniques is frequently adopted [39-42]. Moreover, GGNN, word2vec, and LSTM are used for SQL injection detection [43-46]. The CNN, RNN, LSTM and BiLSTM is used with taint analysis too and GNN is deployed for vulnerabilities analysis as well [47-54].

Another researches focused Naïve Bayes, k-nearest neighbor, neural networks, RF, SVM, decision trees [55-58]. Yet other studies focused on BERT, multinomial Naïve Bayes and SVM [59-71]. The research highlights that none of the studies focused the semantic information of the source code resulting in unreliable performance of the evaluators, in contrast, the proposed study focused on semantic understanding of code while digging the syntactic structure of the code. Moreover, the proposed study focuses multiple vulnerabilities.

Deep learning is becoming vital for cybersecurity issues, as demonstrated by recent research findings though studies like [72-73]. The studies [74–79] explored, password security, intrusion detection, anomaly detection focusing AI and GANs being effective techniques for defending digital systems against dynamic attacks.

3. Methodology

The proposed system incorporates deep learning (DL) techniques and feature selection for a nuanced vulnerability assessment with semantic information about code.

3.1 Framework

The software vulnerability detector analyses source code using DL, focusing on inter-procedural statement-level granularity and code semantics using feature selection. Proposed system is designed to fill the gap of ignoring semantic information of source code to effectively contribute in software security domain. We propose a novel mechanism that integrates DL with semantic-based feature selection, using a decision tree. The feature selection is carried out using novel code metrics introduced in our study. The model training is performed using a benchmark dataset SARD. The pre-processing is performed to achieve a balanced dataset and duplicate code removal to improve the performance of feature selection and classification. The pre-processing is performed using the decision tree. Moreover, the decision tree is used for feature selection as well. The novel semantic-based code metrics used for feature selection are 1. suspected functions: analyzing sensitive functions against each chosen vulnerability, and 2. control flow: evaluating control flow to identify dependencies. The features are set as input to BiLSTM classifiers for vulnerability classification. The selected vulnerabilities include improper input validation, SQL injection attacks, cross-site scripting, missing authorization, and buffer overflow attacks. See Figure 2. for the proposed system's framework.

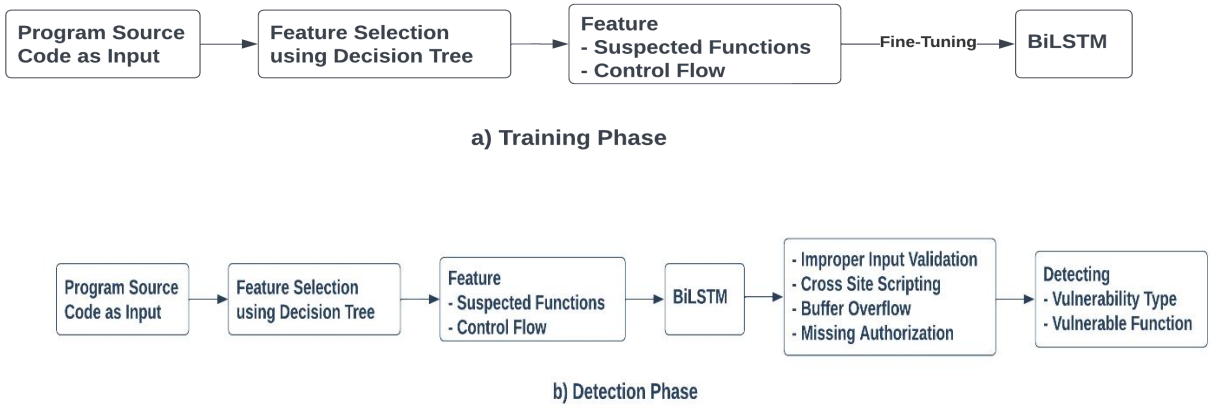

Figure 2. The framework of the proposed vulnerability detection system

3.1 Dataset/Data Acquisition

The Software Assurance Reference Dataset (SARD) is used with total 42212 files and 150 vulnerable classes listed by CWE. The 46,447 Java programs with selected vulnerabilities are used. The system is valided using Juliet java 1.3 dataset which holds 112 CWEs with 28,881 test cases.

3.2 Training Phase

The SARD dataset is used to train the system, training it to select features and classify vulnerabilities using BiLSTM. The dataset is pre-processed to make it more effective, the dataset is balanced and duplicate code is removed using the decision tree. Feature selection is performed using a decision tree, the model is trained to select novel features suspected functions, and control flow by inspecting novel code metrics.

3.2.1 Pre-Processing

1. Dataset balancing

The vulnerability detection datasets contain a huge balance difference for vulnerable codes and non-vulnerable codes, dataset balancing can improve the performance. The balanced dataset can improve the performance of classifier and thus reduces the FP rate. The dataset can be balanced by interchanging the ratio of data points of minority and majority classes. We have used a decision tree using Python Weight = 'balanced' to balance the dataset.

2. Duplicate code removal

The system used decision tree to removes duplicate code or code clones implementing Panda's framework in Python.

3.2.2 Feature Selection

Feature selection imposes most useful features from the dataset. It has been observed that feature selection can significantly reduce computational time, complexity and improves performance. The feature selection generally consists of creating a subset, analyzing the subset, managing the subset criteria, and result validation. We have focused on the fine granularity and semantic information of the code while performing feature selection. The code is divided into smaller chunks considering the semantics of the code which is further labeled to boost the performance. Among many ML based techniques decision tree is used to implement feature selection. The decision tree allows to define the logical set of rules to resolve the task, considers both continuous and categorical variables, and uses less computational power.

Decision tree is a classifier consisting of several nodes demonstrating features, branches pointing the decision rules, and leaf nodes demonstrating output. The classification in the decision tree is performed by passing data from top to bottom. The learning parameters set in this study are quality criterion, maximum depth, minimum objects in a tree leaf, stopping criteria, and separation strategy. We have set the Gini criterion for quality. The mathematical formula is given below

$$H(X) = \sum_{k=1}^{k} P_k\ (1 - p_k) = 1 - \sum_{k=1}^{k} p_k^2 \quad \text{.....(1)}$$

$$p_k = \frac{1}{|X|}\sum_{i \in X}[y_i = k] \quad \text{.....(2)}$$

Where the minimum value of the Gini index is 0 while p is the probability of occurrence of class k. We have not set any limit for the max. depth. The minimum objects for the leaf are set to 2 while the Gini test is used as a separation strategy. The stopping criteria are set concerning the max. depth. The decision nodes are generated using information obtained from the value of each function. A branch is considered using value of each corresponding attribute at each decision node.

Decision tree is constructed using Python is developed that investigates the suspected functions from the vulnerable source code by analyzing variables (local, and global variables), code volume, APIs, pointer usage, array values, data descriptors, and function calls used in the code. For better performance of the algorithm, we have labeled functions into numbers to make it more convenient for the computer to understand. We have used the package of sklearn called LabelEncoder() concerning the vulnerability types we have selected in our study. Once the suspected functions are analyzed each suspected function is inspected to analyze their dependencies by considering the variables (common variables used in function) used in the function, relation of the intersection of operation, which is named as control flow. The suspected functions and control flow are considered selected features and are served as input to BiLSTM.

3.5.1 CodeBERT

The research highlights that pre-trained models are proven to be effective in software security. The CodeBERT is a pre-trained model. The CodeBERT uses natural languages and programming language. It is effective in analyzing the semantic information of code. Moreover, the multi head attention mechanism

is effective in analyzing different key variables of dataflow.

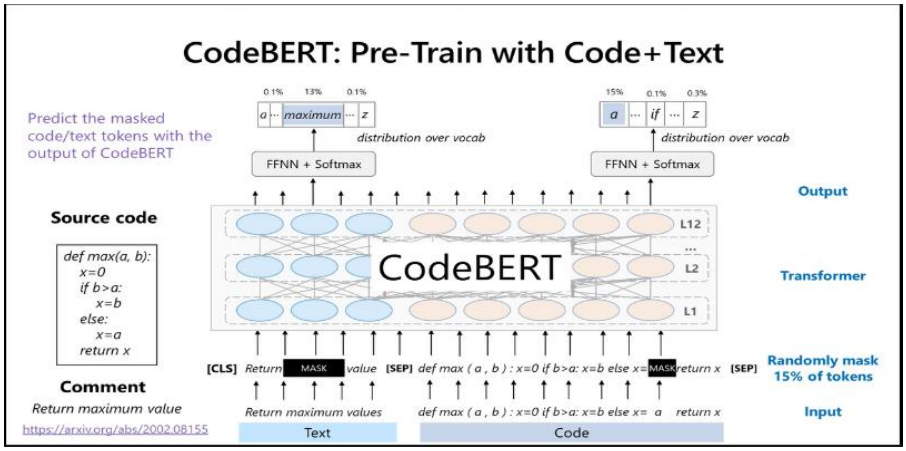

Figure 4. Structure of CodeBERT model

The CodeBERT in the proposed system is deployed to takes code as input. The greedy longest match tokenizes the code. These tokens serves as features for the proposed system. The CodeBERT is fine-tuned using 32 batch size, learning rate of 10^{-3}, 50 epoch size and early stopping to avoid overfitting.

3.2.4 BiLSTM

In feed-forward networks, the connection of the node does not form a cycle/loop while in RNN once the output of a certain process is obtained it feeds it back to the input stage forming a loop. The basic feed-forward is not capable to manage sequential data, handles current input, and unable to remember it. The RNN can deal with sequential data and has memory to keep track of input. In the traditional RNN consists of an input layer, hidden layer, and output layer the hidden layer holds activation function, weight, and biases. The hidden layer can be said a memory of previous input. The mathematical representation of RNN is given below

$$h^t = g_h(Bx^t + Ch^{(t-1)} + b_h) \quad \text{.....(3)}$$

$$y^t = g_y(Ah^t + b_y) \quad \text{.....(4)}$$

where h^t a is a hidden output, g_h is a squashing function, b is bias, y^t is final output and W is a weighted matrix.

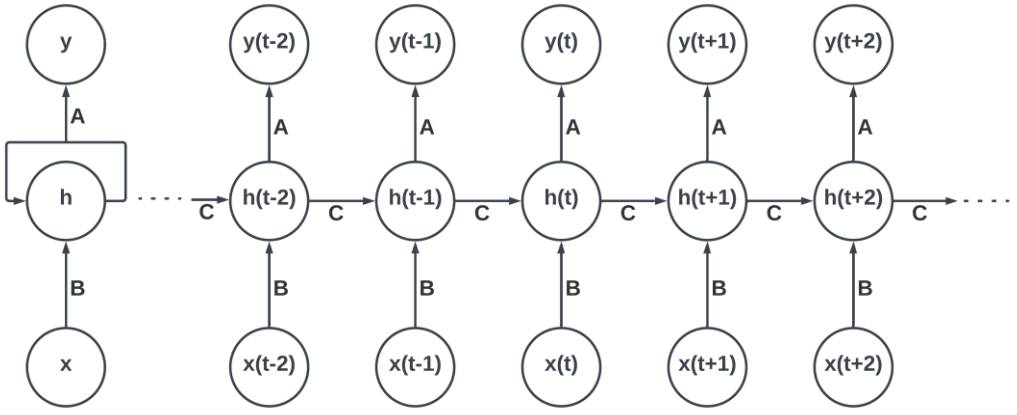

Figure 2. Structure of RNN

The simple RNN model can suffer from vanishing and exploding gradients similarly it does have memory but can't memorize the long time steps thus fails to handle long-term dependencies. The LSTM removed these issues by adding extra memory lines and control gates which allows it to select the most valuable past information to be stored in memory and thus it can identify the long-term dependencies. The memory structure of LSTM is composed of three gates 1. Input gate 2. Forget gate and 3. Output gate. The input gates provide information about cell state. The mathematical representation of the input gate is below in equation 5.

$$I_t = \sigma\,(W_i\,X_i + U_i\,h_{t-1} + b_i\,) \quad (5)$$

Where I_t represents the input gate, σ is a sigmoid function that indicates the dot product, b_i is biased, h_t is a hidden layer function, W_i and U_i are weighted matrices, h_{t-1} the input from the previous time step X_i is a input.

The forget gate is responsible to manage get and forgets state. The mathematical representation f the forget gate is given below in equation (6)

$$f_t = \sigma\,(W_f\,X_f + U_f\,h_{t-1} + b_f\,) \quad (6)$$

Where f_t represents the input gate, σ is a sigmoid function that indicates the dot product, b_f is biased, W_f and U_f are weighted matrices, h_{t-1} the input from the previous time step X_f is a input. The mathematical representation of the output gate is

$$O_t = \sigma\,(W_o\,X_o + U_o\,h_{t-1} + b_o\,) \quad (7)$$

Where O_t represents the output gate, σ is a sigmoid function that indicates the dot product, b_o is biased, W_o and U_o are weighted matrices, h_{t-1} the input from the previous time step X_o is an input. The C_t describes the candidate memory cell at a certain time span while the $\tan h$ is a tangent hyperbolic function.

$$C_t = \tan h\,(W_c\,X_c + U_c\,h_{c-1} + b_c\,) \quad (8)$$

The weight matrices (Wf, Wi, Wo, Wc, Uf, Ui, Uo, Uc) and biases (bf, bi, bo, bc) are not time-dependent. The RC_t denotes the recent state value in memory. The values of forget gate f_t and input gate I_t lies in 0 and 1. The dot product of the input gate and candidate memory cell $I_t \odot C_t$ describes that new information is saved in the recent memory cell taken from the candidate memory cell. Similarly the $f_t \odot RC_{t-1}$ describes that the information saved is important.

$$RC_t = I_t \odot C_t + f_t \odot RC_{t-1} \quad (9)$$

The LSTM can store contextual information and overcome the issue of gradient descent. The BiLSTM model is considered to provide the highest accuracy as it is capable of capturing past and future data information simultaneously. The BiLSTM is an extension of LSTM. The LSTM considers the backward propagation and thus captures only the previous information of the data while BiLSTM trains two LSTM on the input sequences both for forward and backward input by adding a reverse layer of LSTM. The BiLSTM replicates the first input sequence and reverses the other input sequence to simultaneously obtain the context information. The BiLSTM is more context-aware which improves the learning and provides better results. Moreover; the additional training layers in BiLSTM make it more appropriate for prediction. The BiLSTM can be mathematically represented as

$$H_f = f(\,w_{f1}x_t + w_{f2}h_{t-1}\,) \quad (10)$$

$$H_b = f(\,w_{b1}x_t + w_{b2}h_{t+1}\,) \quad (11)$$

Where H_f denotes the outcome with forward LSTM while the H_b denotes the outcome in backward direction.

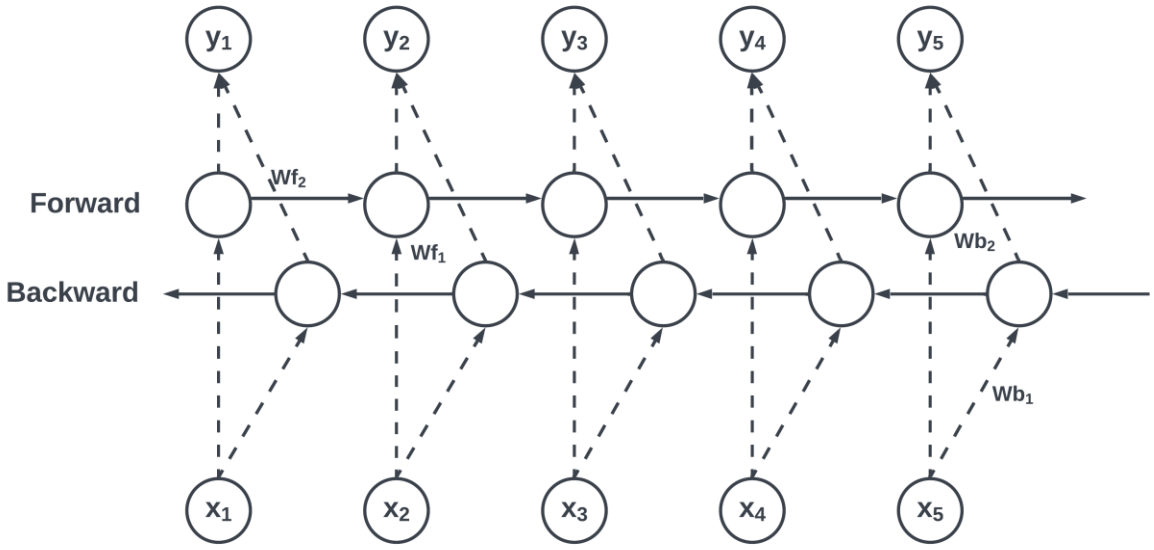

Figure 3. Structure of BiLSTM

We have selected BiLSTM for the proposed model to detect the vulnerability. Vulnerability detection is a complex task we therefore can not label a piece of code as vulnerable without analyzing it in context to it's surrounding functions and its dependencies. The BiLSTM model is capable of considering the locality of code and its context as it holds the memory and trains two LSTM on the input sequence. The fixed parameters used are batch size set to 65, the Adam optimizer, the sigmoid activation function, and the input length set as maximum length. Each gate in BiLSTM holds its number of neurons. It can be observed that setting up more neurons can help the model to deeply learn complex problems like vulnerability detection. To maintain the training time, we have set the number of neurons on the hidden layer using the rule of thumb as 2/3 of size of input layer. The model is run with 50 epochs.

The extracted features, suspected functions, and control flow are used as input to BiLSTM. The BiLSTM classifies the features into vulnerability types. The selected features are classified into improper input validation, cross-site scripting, buffer overflow, and SQL injection vulnerabilities. Moreover; the BiLSTM is responsible to identify the vulnerable function and type of vulnerability.

3.3 Detection Phase

The BiLSTM classifies the selected types of vulnerabilities.

4. Experiments and Results

The experimental setup includes Windows-based system with Intel® Core™ i7-10700H processor. The Python and Tensorflow is used as programming language.

The performance metrics involves AUC of RoC, precision, recall and accuracy. Accuracy is formulated as under

The proposed system successfully detects the selected vulnerabilities from the source code.

Moreover, the proposed system is compared with the existing machine learning techniques to mitigate security vulnerabilities.

Table 2. Comparative analysis with machine learning techniques

	Accuracy (%)	Precision (%)	Recall (%)
CNN	93%	91%	94%
SVM	90%	92%	90%
GNN	94 %	92 %	94%
LSTM	94 %	93 %	96 %
Proposed Model	**96%**	**94%**	**97%**

Table 2 presents a comparison with existing deep learning based techniques such as CNN, LSTM, GNN, and SVM. The proposed model outperformed in terms of accuracy, precision, and recall.
In order to validate the performance, the system is trained using another dataset Juliet java 1.3 as well. The SARD and Juliet java 1.3 are benchmark datasets made public by NIST.

Table 3. Performance of the Proposed Sytem using benchmark Datasets

Dataset	Precision	Recall
SARD	94 %	97 %
Juliet java 1.3	96 %	95%

Table 3. depicts that the proposed system performed well with the other dataset as well which proves the validity of the proposed system.

Table 4. Comparative analysis with commercial vulnerabilities detector

	Acc(%)	P(%)	R(%)
SonarQube	90%	55%	40%
Agnitio	92%	45%	40%
Proposed Model	**96 %**	**62%**	**45%**

In Table 4, our proposed model is compared with the commercial vulnerability detection tools SonarQube and Agnitio. Despite their commercial popularity, our system outperformed them in terms of accuracy and recall rates.

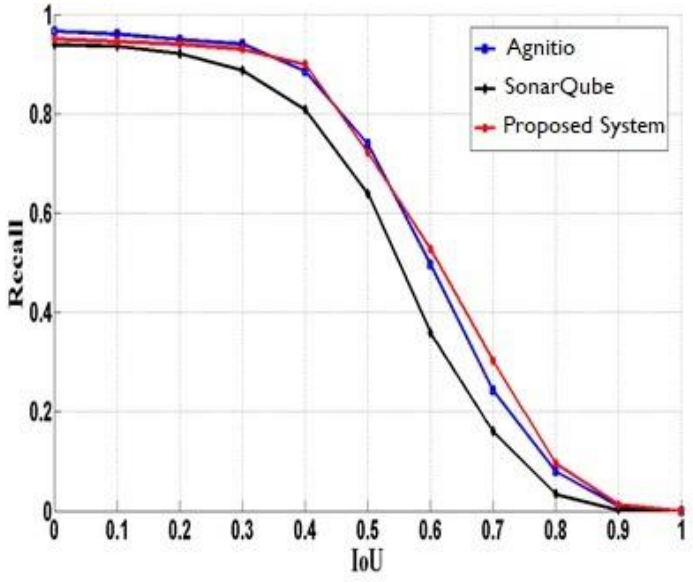

Figure 7. The recall of the proposed system in comparison with commercial systems

Figures 7 display the recall metrics for commercial tools as compared with our proposed system. The results demonstrate the effectiveness of proposed system achieved by integrating feature selection and considering code semantics, with deep learning. The results demonstrate that system successfully reduced the false-positive rate while ensuring a minimum number of missing values.

5. Conclusion

The proposed system effectively managed the existing limitations of vulnerability detectors and encapsulates syntactic and semantic information of the source code. The system uses Bidirectional Long Short-Term Memory (BiLSTM) model along with feature selection performed with the help of CodeBERT a pre-trained model and successfully detect source code for potential security vulnerabilities.

To elevate the performance of the DL based algorithms the dataset is pre-processed for duplicate code removal and data balancing. The feature selection is heighted as critical component to outline the semantic similarity of the source code. The decision tree is deployed for feature selection using novel code matrices based on suspected functions and control flow. The features are the input to the BiLSTM used as classifier.

In order to validate the model, it is compared with convolutional neural networks (CNN), graph neural networks (GNN), long short-term memory (LSTM), and support vector machines (SVM) and with commercially available systems such as SonarQube and Agnitio. The results demonstrate that the system outperformed signifying a promising advancement in the field of automatic vulnerability detection.

REFERENCES

CVSS Severity Distribution Over Time. Available online: https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time (accessed on 30 April 2025).

Tassey and Gregory, "The Economic Impacts of Inadequate Infrastructure for Software Testing," RTI Health, Social, and Economics Research, Gaithersburg, 2002.

M. Zhivich and R. K. Cunningham, "The Real Cost of Software errors," IEEE Security & Privacy, vol. 7, no. 2, pp. 87-90, April 2009.

J. Strasburg and J. Bunge, "Loss Swamps Trading Firm," The Wall Street Journal, pp. 1-5, 2012.

L. Geppert, "Lost Radio Contact Leaves Pilots on Their Own," IEEE Spectrum, vol. 41, no. 11, pp. 16-17, 2004.

J. Berr, "wannacry-ransomware-attacks-wannacry-virus-losses," 16 May 2017. [Online]. Available: https://www.cbsnews.com/news/wannacry-ransomware-attacks-wannacry-virus-losses/. [Accessed 6 April 2025].

Y. Chen, J. Chen, Y. Gao, D. Chen and Y. Tang, "Research on Software Failure Analysis and Quality Management Model," in IEEE International Conference on Software Quality,Reliability and Security Companion, Lisbon,Portugal, 2018.

T. Marjanov, I. Pashchenko and F. Massacci, "Machine Learning for Source Code Vulnerability Detection: What Works and What Isn't There Yet," IEEE Security and Privacy, vol. 20, pp. 60-76, 2022.

A. N. Lam, A. T. Nguyen, H. A. Nguyen and T. N. Nguyen, "Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports (N)," in IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, 2015.

Y. Pu, K. Narasimhan, A. Solar-Lezama and R. Barzilay, "A neural program corrector for MOOCs," in Companion proceddings of ACM SIGPLAN International Conference on System, Programming, Languages and Applications Software for Humanity, New York, 2016.

M. White, C. Vendome, M. Linares-Vásquez and . D. Poshyvanyk, "Towards Deepl Learning Software Repositories," in Mining Software Repositories, 2015.

R. Scandariato, J. Walden, A. Hovsepyan and W. Joosen, "Predicting Vulnerable Software Components via Text Mining," IEEE Transactions on Software Engineering, vol. 40, no. 10, pp. 993-1006, 2014.

P. Morrison, K. Herzig, B. Murphy and L. Williams, "Challenges with applying vulnerability prediction models," in HotSoS '15: Proceedings of the 2015 Symposium and Bootcamp on the Science of Security, New York, 2015.

H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy and A. Ghose, "Automatic feature learning for vulnerability Predictions".

M.-J. Choi, S. Jeong, H. Oh and J. Choo, "End-to-end prediction of buffer overruns from raw source code via neural memory networks," in IJCAI'17: Proceedings of the 26th International Joint Conference on Artificial Intelligence, Malbourne, 2017.

Y. Pang, X. Xue and A. N. , "Predicting Vulnerable Software Components through N-Gram Analysis and Statistical Feature Selection," in IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015.

A. Hovsepyan, R. Scandariato, W. Joosen and J. Walden, "Software vulnerability prediction using text analysis techniques," in MetriSec '12: Proceedings of the 4th international workshop on Security measurements and metrics, 2012.

V. Piantadosi, S. Scalabrino and R. Oli, "Fixing of Security Vulnerabilities in Open Source Projects: A Case Study of Apache HTTP Server and Apache Tomcat," in International Conference on Software Testing, Verification, and Validation, ICST, 2019.

Hanif, H., Md Nasir, M. N., Ab Razak, M., Firdaus, A., & Nor Badrul, A. (2021). The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches Journal of Network and Computer Applications.

R. L. Russel, L. Kim, L. H. Hamilton, T. L. J. A. Harer, O. Ozdemir, P. M. Ellingwood and M. W. McConley, "automated vulnerabilities detection in source code using deep representation learning," in IEEE International Conference on machine learning and application, Florida, 2018.

J. Kronjee, A. Hommersom and H. Vranken, "Discovering software vulnerabilities using data-flow analysis and machine learning," in Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, 2018.

S. Kim, S. Woo, H. Lee and H. Oh, "VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery," in IEEE Symposium on Security and Privacy, 2017.

B. Shuai, H. Li, L. Zhang, Q. Zhang and C. Tang, "Software Vulnerability Detection Based on Code Coverage and Test Cost," in International Conference on Computational Intelligence and Security, 2015.

Z. Yu, C. Theisen, L. Williams and T. Menzies, "Improving Vulnerability Inspection Efficiency Using Active Learning," IEEE Transactions on Software Engineering, vol. 47, no. 11, pp. 2401 - 2420, 2021.

Liu, S., Lin G., Han, Q.L., Wen, S., Zhang, J. & Xiang, Y. (2019). DeepBalance: Deep-Learning and Fuzzy Oversampling for Vulnerabilities Detection. IEEE Transactions on Fuzzy Systems, 1329-1343.

J. Ren, Z. Zheng, Q. Liu, Z. Wei and H. Yan, "A Buffer Overflow Prediction Approach Based on Software Metrics and Machine Learning," Security and Communication Networks, 2019.

Liang, H., Sun, L., Wang, M., & Yang, Y. (2019). Deep Learning With Customized Abstract Syntax Tree for Bug Localization. IEEE Access, 116309 - 116320.

Parteza, G., Amburgey, T., Deng, L., Dehlinger, J., & Chakraborty, S. (2021). Automatic

Identification of Vulnerable Code: Investigations with an AST-Based Neural Network. 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC) (pp. 1475-1482). IEEE.

P. R. Vishnu, P. Vinod, & Yerima, S. (2022). A Deep Learning Approach for Classifying Vulnerability Descriptions Using Self Attention Based Neural Network. Journal of Network and Systems Management.

R. li, C. Feng, X. Zhang and C. , "A Lightweight Assisted Vulnerability Discovery Method Using Deep Neural Networks," IEEE Access, pp. 80079-80092, 2019.

X. Li, L. Wang, Y. Xin, Y. Yixian and Y. Chen, "Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning," Applied Sciences, 2020.

Tian, J., Xing, W., & Li, Z. (2020). BVDetector: A program slice-based binary code vulnerability intelligent detection system. Information and Software Technology.

Zagane, M., Abdi, M. K., & Alenezi, M. (2020). Deep Learning for Software Vulnerabilities Detection Using Code Metrics. IEEE Access, 74562 - 74570.

X. Li, L. Wang, Y. Xin, Y. Yixian and Y. Chen, "Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning," Applied Sciences, 2020.

M. Zagane, M. K. Abdi and M. Alenezi, "Deep Learning for Software Vulnerabilities Detection Using Code Metrics," IEEE Access, vol. 8, pp. 74562-74570, 2020.

Y. Zhou, S. Liu, J. Siow, X. Du and Y. Liu, "Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks," in NeurIPS, 2019.

N. Guo, X. Li, H. Yin and Y. Gao, "VulHunter: An Automated Vulnerability Detection System Based on Deep Learning and Bytecode," in International Conference on Information and Communications Security, 2019.

Xuan, C. D., Mai, D. H., Thanh, M. C., & Cong, B. V. (2023). A novel approach for software vulnerability detection based on intelligent cognitive computing. Journal of supercomputing.

R. L. Russell, L. Kim, L. H. Hamilton, T. Lazovich, J. A. Harer, O. Ozdemir, P. M. Ellingwood and M. W. McConley, "Automated Vulnerability Detection in Source Code Using Deep Representation Learning," in 17th IEEE International Conference on Machine Learning and Applications (IEEE ICMLA 2018), Orlando, Florida, USA, 2018.

L. Hu, J. Chang, Z. Chen and B. Hou, "Web application vulnerability detection method based on machine learning," Journal of Physics, 2021.

H. Alves, B. Fonseca and N. Antunes, "Experimenting Machine Learning Techniques to Predict Vulnerabilities," in Latin-American Symposium on Dependable Computing, LADC, 2022.

Hin, D., Kan, A., Chen, H., & Babar, M. (2022). LineVD: statement-level vulnerability detection using graph neural networks. MSR '22: Proceedings of the 19th International Conference on Mining software repositories. ACM

Fan, Y., Wan, C., Fu, C., Han, L., & Xu, H. (2023). VDoTR: Vulnerability detection based on tensor representation of comprehensive code graphs. Computer & Security.

Saccente, N., Dehlinger, J., Deng, L., Chakraborty, S., & Xiong, Y. (2019). Project Achilles: A Prototype Tool for Static Method-Level Vulnerability Detection of Java Source Code Using a Recurrent Neural Network. IEEE/ACM International Conference on Automated Software Engineering - Workshops (ASE Workshops). IEEE.

Y. Pang, X. Xue and H. Wang, "Predicting Vulnerable Software Components through Deep Neural Network," in ICDLT '17: Proceedings of the 2017 International

Conference on Deep Learning Technologies, 2017.

L. Wartschinski, Y. Noller, T. Vogel, T. Kehrer and L. Grunske, "VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python," Information and Software Technology, vol. 144, 2022.

W. Niu, X. Zhang, X. Du, L. Zhao, R. Cao and M. Guizani, "A deep learning based static taint analysis approach for IoT software vulnerability location," Measurement, vol. 152, 2020.

Niu, W., Zhang, X., Du, X., Zhao, L., Cao, R., & Guizani, M. (2020). A deep learning based static taint analysis approach for IoT software vulnerability location. Measurement, 152.

F. Wu, J. Wang, J. Liu and W. Wang, "Vulnerability detection with deep learning," in International Conference on Computer and Communications (ICCC), 2017.

F. Wu, J. Wang, J. Liu and W. Wang, "Vulnerability detection with deep learning," in International Conference on Computer and Communications (ICCC), 2017.

A. V. Phan, M. L. Nguyen and L. T. Bui, "Convolutional Neural Networks over Control Flow Graphs for Software Defect Prediction," in International Conference on Tools for Artificial Intelligence (ICTAI), 2017.

H. Hanif, M. H. N. Ms Nasir, M. F. Ab Razak, A. Firdaus and N. Badrul Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," Journal of Network and Computer Applications, vol. 179, 2021.

Luo, Y., Xu, W., & Xu, D. (2022). Compact Abstract Graphs for Detecting Code Vulnerability with GNN Models. ACSAC '22: Proceedings of the 38th Annual Computer Security Applications Conference. Texas: ACM

Nguyen, V.-A., Nguyen, D., Nguyen, V., Le, T., Tran, Q. H., & Phung, D. (2022). ReGVD: revisiting graph neural networks for vulnerability detection. ICSE '22: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings. New York: ACM.

Al-Boghdady, A., El-Ramly, M., & Wassif, K. (2022). iDetect for vulnerability detection in internet of things operating systems using machine learning. Scientific Report.

H. Perl, S. Dechand, M. Smith, D. Arp, F. Yamaguchi, K. Rieck, S. Fahl and Y. Acar, "VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits," in VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits, 2015.

G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist and L. Mounier, "Toward Large-Scale Vulnerability Discovery using Machine Learning," CODASPY '16: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, pp. 85-96, 2016.

B. Chernis and R. Verma, "Machine Learning Methods for Software Vulnerability Detection," in IWSPA '18: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics, 2018.

D. Iorga, D. Corlatescu, O. Grigorescu, C. Sandescu, M. Dascalu and R. Rughiniş, "Early Detection of Vulnerabilities from News Websites using Machine Learning Models," in Roedunet International Conference (RoEduNet), 2020.

M. A. Hameed, F. Yang, S. U. Bazai, M. I. Ghafoor, A. Alshehri, I. Khan, S. Ullah, and M. Baryalai, “Convolutional Autoencoder-Based Deep Learning Approach for Aerosol Emission Detection Using LiDAR Dataset,” Journal of Sensors, vol. 2022, Article ID 3690312, Jun. 2022. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2022/3690312

S. Hussain, S. U. Bazaib, S. Qadir, S. Marjan, M. I. Ghafoor, and P. Pervaiz, “Title of the article,” VAWKUM Transactions on Computer Science, vol. 13, no. 1, pp. xx–xx, 2025. Available:

https://www.vfast.org/journals/index.php/VTCS/article/view/2081

M. Akram et al., "EEMLCR: Energy-Efficient Machine Learning-Based Clustering and Routing for Wireless Sensor Networks," in IEEE Access, vol. 13, pp. 70849-70871, 2025.

S. Hussain, S. U. Bazaib, S. Qadir, S. Marjan, M. I. Ghafoor, and P. Pervaiz, "Title of the Article," Computers, Materials & Continua, vol. 79, no. 1, pp. x–y, 2024.

S. Ullah, S. U. Bazai, M. Imran, Q. M. Ilyas, A. Mehmood, M. A. Saleem, M. A. Rafique, A. Haider, I. Khan, S. Iqbal, Y. Gulzar, and K. Hameed, "Title of the article," Computers, Materials & Continua, vol. 79, no. 1, pp. xx–xx, Apr. 1, 2024.

U. A. Bhatti, S. U. Bazai, S. Hussain, S. Fakhar, S. Marjan, and P. L. Yee, "Title of the article," Computers, Materials & Continua, vol. 77, no. 1, pp. xx–xx, Nov. 1, 2023.

M. Ahmad, S. Ullah Bazai, S. Hussain, A. I. Ashirova, Y. J. Erkaboy Ugli and U. Aslam Bhatti, "Predicting Household Electricity Consumption Using Machine Learning and Big Data Analytics," 2025 IEEE 2nd International Conference on Deep Learning and Computer Vision (DLCV), Jinan, China, 2025, pp. 1-7, doi: 10.1109/DLCV65218.2025.11088841.

H. Shumaila, et al. "Vulnerability detection in Java source code using a quantum convolutional neural network with self-attentive pooling, deep sequence, and graph-based hybrid feature extraction." Scientific Reports 14.1 (2024): 7406.

M. A. Shoaib, R. Ali, S. U. Bazai, and T. Mir, "Deep Learning Techniques for Image Segmentation and Data Annotation," in Modern Intelligent Techniques for Image Processing, IGI Global Scientific Publishing, 2025, pp. 63–94.

U. Rehman, S. Ullah, S. U. Bazai, F. Ullah, M. I. Ghafoor, and B. Kasi, "Efficient Smart Aeroponic Cultivation and Monitoring System Through Internet of Things," in Proc. 2024 Int. Conf. on IT and Industrial Technologies (ICIT), Dec. 10, 2024, pp. 1–5. IEEE.

S. Zulfiqar, S. U. Bazai, M. I. Ghafoor, M. Soomro, L. Hussain, and A. Haider, "Automatic Question Generation for Cognitive Domain of Outcome-Based Education System," in Proc. 2024 Int. Conf. on IT and Industrial Technologies (ICIT), Dec. 10, 2024, pp. 1–6. IEEE.

B. Sibghat Ullah, et al. "Water Quality Prediction Using Random Forest Classifier: An Analysis of Chemical Attributes and Their Feature Importance." 2024 5th International Conference on Innovative Computing (ICIC). IEEE, 2024. R. Noor, A. Wahid, S. U. Bazai, A. Khan, M. Fang, S. MS, U. A. Bhatti, and Y. Y. Ghadi, "DLGAN: Undersampled MRI reconstruction using deep learning based generative adversarial network," Biomedical Signal Processing and Control, vol. 93, p. 106218, Jul. 2024, Elsevier.

Haider, L. Hussain, A. W. Tareen, S. U. Bazai, S. Aslam, M. Neo, and A. Amphawan, "A comparative study of machine learning techniques for accurate disease prediction using symptom-based diagnosis," AIP Conf. Proc., vol. 3153, no. 1, p. 020005, Jun. 27, 2024, AIP Publishing LLC.

S. Tareen, S. U. Bazai, S. Ullah, R. Ullah, S. Marjan and M. I. Ghafoor, "Phishing and Intrusion Attacks: An Overview of Classification Mechanisms," 2022 3rd International Informatics and Software Engineering Conference (IISEC), Ankara, Turkey, 2022, pp. 1-5 IEEE

Z. Zaland, S. U. Bazai, S. Marjan and M. Ashraf, "Three-Tier Password Security Algorithm for Online Databases," 2021 2nd International Informatics and Software Engineering Conference (IISEC), Ankara, Turkey, 2021, pp. 1-6IEEE

I. Tabassum, S. U. Bazai, Z. Zaland, S. Marjan, M. Z. Khan and M. I. Ghafoor, "Cyber Security's Silver Bullet - A Systematic Literature Review of AI-Powered Security," 2022 3rd International Informatics and Software Engineering Conference (IISEC), Ankara, Turkey, 2022, pp. 1-7 IEEE

S. Noor, S. U. Bazai, M. I. Ghafoor, S. Marjan, S. Akram and F. Ali, "Generative Adversarial Networks for Anomaly Detection: A Systematic Literature Review," 2023 4th International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), Sukkur, Pakistan, 2023, pp. 1-6 IEEE

H. Huimin, et al. "Hybrid Climate Forecasting: Variational Mode Decomposition and Convolutional Neural Network with Long-Term Short Memory." Polish Journal of Environmental Studies 33.2 (2024).

B. Uzair Aslam, et al., eds. Deep learning for multimedia processing applications: Volume two: Signal processing and pattern recognition. CRC Press, 2024.

B. Laila, et al. "A review of big data trends and challenges in healthcare." International Journal of Technology 14.6 (2023): 1320-1333.