Enhancing Vulnerability Detection Precision through Ensemble Learning with Large Language Models

Hussein Al-Ofeishat
Department of Computer Science, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan | Faculty of Engineering, Al-Balqa Applied University, Al-Salt, Jordan
h.ofeishat@ammanu.edu.jo

Azhar Hussain
Department of Computer Science, The Islamia University of Bahawalpur, Pakistan
azharhussain.engr@gmail.com

Muhammad Rehan Faheem
Fakulti Kecerdasan Buatan dan Keselamatan Siber, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia
rehan@utem.edu.my (corresponding author)

Syed Asim Ali Shah
Fakulti Pengurusan Teknologi Dan Teknousahawanan, Kampus Teknologi, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia
syed.asim@utem.edu.my

Hannan Adeel
Fakulti Kecerdasan Buatan dan Keselamatan Siber, Universiti Teknikal Malaysia Melaka, Melaka, Malaysia
hannan@utem.edu.my

Muzammil Hussain
Department of Software Engineering, Faculty of Information Technology, Al-Ahliyya Amman University, Amman, Jordan
m.hussain@ammanu.edu.jo

Received: 27 February 2026 | Revised: 19 May 2026 | Accepted: 27 May 2026

 DOI: https://doi.org/10.48084/etasr.18412

ABSTRACT

This study investigates the use of ensemble learning with Large Language Models (LLMs) to improve the accuracy of software vulnerability prediction, following a structured experimental approach to assess whether combining multiple models can enhance performance. Three baseline models, CodeBERT, GraphCodeBERT, and CodeT5, were trained and assessed on the Devign dataset, which provides a large collection of labeled source code snippets. Their outputs were then integrated using three ensemble techniques: Majority Voting, Weighted Voting, and Stacking. Precision, recall, and F1-score metrics were used to gauge performance. Ensemble approaches outperformed all standalone models. In particular, Majority Voting increased precision from 0.601 (CodeBERT) to 0.690, representing a 14.81% improvement. Keeping in view the detection accuracy, this study focused on reducing the false positives. The results show that the ensemble techniques are a practical approach to boost the precision of LLMs in the detection of vulnerabilities. Ensemble learning can address the challenges faced by standalone models

by reducing false positives and improving the overall trade-off between accuracy and reliability. The study suggests that ensemble methods offer great potential in the advancement of software security analysis.

Keywords-ensemble learning; large language models; vulnerability detection; precision; majority voting; weighted voting; stacking

I. INTRODUCTION

As the use of LLMs in the development of security tools continues to increase, malicious individuals are also exploiting the same technologies to craft convincing phishing scripts that can evade AI-based detection and prevention systems [1]. For several decades, static approaches were used in research. In contrast, dynamic techniques complement static approaches, detecting vulnerabilities that do not appear in testing and cannot be detected with static analysis [2]. The scenario of "False Alarm vs The Blind Spot" has been appropriately documented, and researchers are in the process of identifying solutions that can better generalize on code patterns that traditional methods are unable to see [3]. Static approaches often generate a large number of false positives, leading to what is commonly described as "alert fatigue" among developers. Dynamic approaches, on the other hand, execute code under controlled environments to observe runtime behavior. While dynamic approach methods capture vulnerabilities missed by static approaches, they are computationally expensive and less scalable to large codebases [4]. The study in [5] contributed an approach capable of automatically learning malicious pattern features for raw source code without the necessity to handcraft such features. This opened new avenues for transformer and graph-based models to enhance representational depth in vulnerability detection systems. Conventional approaches have failed to adapt to the current software sophistication [5]. Recent surveys have observed that traditional tools are fragile in new attacks [6]. This disconnect has prompted researchers to explore ML-based methods that go beyond handwritten rules.

With the rapid advancement of Large Language Models (LLMs), such as OpenAI's GPT series, and transformer-based models such as Google's BERT, researchers have explored their application beyond traditional natural language processing tasks. These models have shown strong potential in software security, including the detection and analysis of vulnerabilities in software systems [7]. Devign [8] used graph neural networks to extract rich program semantics that effectively discriminate vulnerable from non-vulnerable code [8]. Since then, this benchmark has become a de facto standard for the assessment of contemporary transformer and ensemble-based models in vulnerability detection research.

Recent studies have extended previous ones by using LLMs for vulnerability detection and continuous monitoring. LLMs can capture intricate semantic relationships in source code, but when used for diverse or unknown projects, they still have significant precision issues [9]. LLMs have shown outstanding capabilities to learn the context in the identification of patterns within textual data, suggesting that they can be employed in the automation of vulnerability detection systems. Some studies have shown that LLMs trained on large datasets can identify vulnerabilities like buffer overflow, SQL injection, and cross-Site Scripting (XSS) [5]. Many LLMs have been proven to be the most suitable options for software security tasks, handling large amounts of data with their plug-and-play adaptability for many coding languages and structures. LLMs can find vulnerabilities that traditional methods cannot, focusing on associations and code patterns that do not immediately jump out as issues to a human analyst [10]. VulBERTa [11] is a lightweight transformer model pre-trained on source code, which achieved state-of-the-art performance comparable to larger LLMs while having lower computational cost. This contribution emphasized that task-specific models may work well, but assembling them in the form of ensembles might improve consistency and accuracy. In [12], a transformer-based model was able to predict vulnerabilities at the line level (instead of the function). The results of this study increased interpretability and precision while exposing the instability of model predictions between complex codebases, which serves to justify the integration of ensembles. In [13], the concept was promoted by presenting a heterogeneous GNN-based training structure that is efficient for modeling multiple code dependencies simultaneously. While this approach achieved improved recall and deeper semantic understanding, it also showed the constraints of scalability and generalization, which can be addressed by using ensemble methods.

Ensemble learning strategies have been used to enhance the detection of network intrusions and vulnerabilities in a wide range of ML, such as DL models with various optimization algorithms [14] and LLMs. Studies like [15] have demonstrated that combining multiple DL models can detect considerably higher rates of network anomalies and intrusions with drastically fewer false positives. In the same way, ensemble learning methods have been shown to increase the performance of LLMs when used for vulnerability detection tasks, where combining the different LLM architectures or model configurations helps to explore diverse patterns in the data and minimize errors. In previous works [11-13], individual LLMs were used for vulnerability detection systems. This research provides an empirical comparison of three transformer-based LLMs, namely CodeBERT, GraphCodeBERT, and CodeT5, and their ensemble. This research provides a focus on precision tuning and model stability with respect to vulnerability detection.

II. METHODOLOGY

This study systematically investigated LLMs to detect software vulnerabilities and build ensembles to increase performance. The methodology is designed to answer two specific questions: (1) what are the limitations and challenges when using LLMs in vulnerability detection tasks, and (2) how do ensemble techniques contribute to model precision while maintaining recall. Figure 1 represents the schematic diagram of an ensemble approach.

A. Dataset

Experiments were performed on the Devign dataset [8], a well-known benchmark set up to detect vulnerabilities in C and C++ source code. Each sample of the dataset is a code snippet at the function level, with its associated binary label (1), indicating whether the corresponding vulnerability exists in the source code, or (0) not. The dataset has a wide variety of vulnerability types to effectively evaluate the model's generalizability. The dataset was acquired from a public repository [16], enabling re-analysis. Table I shows the number of samples in each subset.

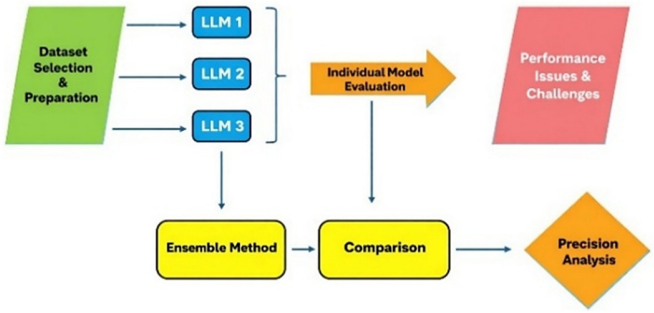

Fig. 1. Schematic diagram of an ensemble approach.

TABLE I. DATASET: NUMBER OF SAMPLES PER SUBSET

Dataset split	Number of samples
Train	21,854
Validation	2,732
Test	2,732
Total	27,300

Tokenization was performed using the pre-trained tokenizer corresponding to each selected model, with a maximum sequence length of 512 tokens to balance computational efficiency and code context preservation. Special tokens were handled according to the model's architecture to ensure compatibility. Since this research uses transformer-based LLMs for vulnerability detection, these models cannot read raw source code directly but require token inputs and also have fixed input size limits (usually 512 tokens). These special tokens ensure that the code is converted into a format the model understands, the input fits the model's maximum capacity, and training remains computationally feasible on Google Colab GPUs.

B. Model Selection

The choice of models plays a critical role in determining the effectiveness of a vulnerability detection framework. Three representative models, i.e., CodeBERT, GraphCodeBERT, and CodeT5, were chosen in this research.

C. Baseline Model Implementation

The baseline models were implemented in the Hugging Face Transformers framework. All experiments were conducted on Google Colab using a Tesla T4 GPU. Each model was trained for three epochs on the training set with a learning rate of 2e-5, and a batch size of 16 for training and 32 for evaluation. The AdamW optimizer was used with a weight decay of 0.01 to prevent overfitting. Experiments were conducted using consistent training settings; however, a fixed random seed was not explicitly specified.

1) CodeBERT

CodeBERT is pre-trained from raw code snippets to meaningful predictions. The sequence starts with the input of a source code fragment to be analyzed. As ML models cannot process raw text directly, the snippet is tokenized first. At this point, the code is broken into smaller chunks (referred to as tokens), and each of them is associated with a number. After tokenization, the data is passed into the Transformer Encoder (central component of the CodeBERT model). This encoder applies multiple layers of self-attention and deep neural computations to capture both the structural and contextual characteristics of the code. By doing so, it is able to learn hidden relationships among different tokens, producing a rich and context-aware representation of the original snippet.

2) GraphCodeBERT

The GraphCodeBERT model is trained by encoding both the structural and semantic knowledge of code. First, a piece of code is provided as input, and a data flow graph is obtained, unlike the other models that tokenize the code. These enriched embeddings are fed into the classification layer. This study demonstrates the potential of treating source code as graph-based learning between sequence-only models and software engineering and security tasks by GraphCodeBERT.

3) Code T5

CodeT5 is a transformer-based model that is used to understand and generate source code. It starts with code snippets in raw text input. The code snippet is fed to the tokenizer to be converted into smaller expressive tokens. Tokenization helps the model in processing the code efficiently. The data then enters the transformer encoder, which captures the syntactic and semantic information within the code sequence via attention mechanisms. In this way, the model learns the relationships between tokens and constructs representations based on the context that reproduces the sense and meaning of the complete snippet. Finally, the learned and encoded representations are fed into the classification head. This head is task-specific, such as bug identification, vulnerability predictions, or classification. The output provides a final decision based on the predictions in the previous steps.

D. Ensemble Learning Implementation

Three ensemble learning methods were used to integrate predictions from the baseline models.

1) Majority Voting

Each model contributes equally, and the final output is decided by the most frequent prediction among the models.

2) Weighted Voting

Weighted voting assigns more importance to the models with higher F1-scores, adjusting the final output toward more reliable and accurate classifiers. The weights for each model were calculated using the validation F1-score from the baseline models with the following formula:

$$Total_{F1} = F1_{CodeBERT} + F1_{GraphCodeBERT} + F1_{CodeT5}$$

$$w_{CodeBERT} = \frac{F1_{CodeBERT}}{Total_{F1}}$$

$$w_{GraphCodeBERT} = \frac{F1_{GraphCodeBERT}}{Total_{F1}}$$

$$w_{Codet5} = \frac{F1_{CodeT5}}{Total_{F1}}$$

This allows the ensemble method to give more weight to models that demonstrate higher accuracy, thereby improving the overall performance.

3) Stacking Classifier

This technique utilizes the predictions of chosen baseline models as input parameters for a meta-classifier (Logistic Regression), allowing it to learn optimal combination patterns. Validation predictions used for training the meta-learner and test sets were kept strictly unseen. The stacking classifier was trained using validation predictions, investigating the results on the test set to ensure unbiased performance predictions.

E. Evaluation Strategy

Precision, recall, and F1-score were selected as evaluation metrics, where precision was considered to be the most important in line with the research aim of minimizing false positives. Reducing false positives in vulnerability detection systems was the main objective, and these matrices are considered appropriate for assessing the efficiency of the proposed ensemble techniques. The performance of ensemble learning methods was directly compared with that of the best individual models. When required, significance tests were used to determine the validity of the observed improvements.

III. RESULTS

A. Performance of Baseline Models

Table II summarizes the results of the three baseline models. Each model has a different trade-off between precision and recall due to architectural reasons.

TABLE II. PERFORMANCE OF BASELINE MODELS USING THE DEVIGN TEST DATASET

Model	Precision	Recall	F1-score
CodeBERT	0.601	0.404	0.483
GraphCodeBERT	0.587	0.459	0.515
CodeT5	0.595	0.568	0.581

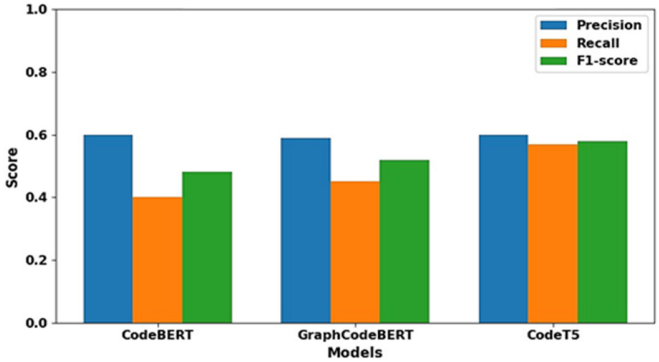

Fig. 2. Performance comparison of baseline models.

Figure 2 shows a comparison of the baseline models. The results provide a clear indication of the challenges involved in applying LLMs for vulnerability detection. Although these models can capture some of the vulnerability patterns, their performance is still not suitable for practical usage. For example, CodeBERT achieved the best precision of 0.601, demonstrating its effectiveness in finding vulnerable code snippets successfully. However, this benefit is gained with a very low recall (0.404), indicating that many vulnerabilities were missed in the detection process. This imbalance illustrates the first important issue, i.e., the current pre-trained LLMs have difficulty in striking a good precision–recall tradeoff, which is crucial to security-critical applications where both false negatives and false positives can cause catastrophic damage.

GraphCodeBERT, with a recall of 0.459 and a slightly reduced precision of 0.587, performed more balanced but still suffered from moderate overall accuracy. CodeT5 offered the most balanced outcome, with recall reaching 0.568 and precision at 0.595, resulting in the best F1-score (0.581) among the three models. No baseline model achieved the best results for all metrics at the same time. Individual models either fail to generalize the output results beyond the training data or overfit to specific patterns due to the imbalance between recall and precision. Therefore, the use of the individual models is insufficient for identifying vulnerabilities. Additionally, these models are unable to handle real-world complex code vulnerabilities due to the low F1-score. Model bias, generalization issues, and the precision–recall tradeoff discourage the use of LLMs in industry for information security. Therefore, there is a dire need to evaluate and explore the design and implementation approaches, such as ensemble learning methods, to mitigate these issues by combining the strengths of multiple standalone models. These conclusions straightforwardly address the research question 1.

B. Performance of Ensemble Methods

The ensemble methods were applied to the predictions made by the baseline models, and Table III shows their results.

TABLE III. PERFORMANCE METRICS FOR ENSEMBLE METHODS

Method	Precision	Recall	F1-score	Precision improvement (%)
Majority Voting	0.690	0.590	0.636	14.81 %
Weighted Voting	0.671	0.605	0.636	11.65 %
Stacking	0.669	0.615	0.641	11.31 %

The results of the ensemble approaches demonstrate that combining the strengths of different models leads to measurable improvements over individual baselines. Majority Voting achieved the most significant improvement in precision, reaching 0.690, which is substantially higher than all standalone models. This result demonstrates the effectiveness of simple voting mechanisms in reducing false positives by leveraging consensus among multiple learners. Although recall remains moderate at 0.590, the balance achieved between the two measures results in a marked increase in the overall F1-score compared to a single-model performance, as shown in Figure 3. Weighted Voting also proves the benefit of

combining multiple models, as it provides more weight to the stronger classifiers. This method avoids the trade-off observed in the baseline models and additionally contributes to enhancing precision and recall at the same time. The resulting F1-score of 0.641 is the overall best trade-off using the stacking classifier. By enabling a meta-classifier to learn when to trust each base model, stacking achieves higher recall without sacrificing precision and thus can offer improved robustness in vulnerability detection. Taken together, these findings provide strong evidence in support of research question 2. The experiments clearly show that ensemble learning is effective in addressing the limitations of individual LLMs, particularly the imbalance between precision and recall observed in the baselines. The ensemble methods not only boosted the precision, which was the actual focus of this study, but also contributed to a better trade-off between the performance evaluation metrics by combining the strengths of multiple models, CodeBERT token-level understanding, GraphCodeBERT's structural sensitivity, and CodeT5's semantic coverage, which is necessary in the detection of software vulnerabilities.

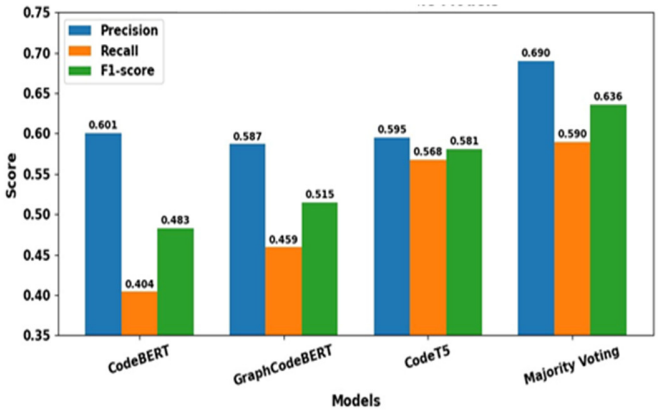

Fig. 3. Performance comparison of Majority Voting vs baseline models.

Weighted voting achieved a better balance, with an F1-score of 0.636, a precision of 0.671, and a recall of 0.605. The weighting system ensured that stronger models, such as CodeT5, had more of an impact, which made the performance more stable. Stacking took this trend even further, achieving the best overall balance across the three metrics, as shown in Figure 4. With precision at 0.670 and recall at 0.615, stacking delivered the highest F1-score (0.641), highlighting its capacity to dynamically adjust to the strengths and weaknesses of individual models through the use of a meta-classifier. The comparative results in Figure 5 demonstrate that ensemble learning can effectively alleviate the aforementioned limitations in baseline methods. While individual models were challenged by the precision–recall trade-off, ensemble learning techniques achieved higher precision and better recall. The significant improvements in F1-scores using all three ensemble methods again confirm that concatenating various LLM architectures is a more robust method for identifying vulnerabilities. This comparison not only confirms the rationale behind using ensemble-based strategies but also directly highlights the main research hypothesis: combining different models can ultimately outperform any of them in terms of a more reliable and robust detection system.

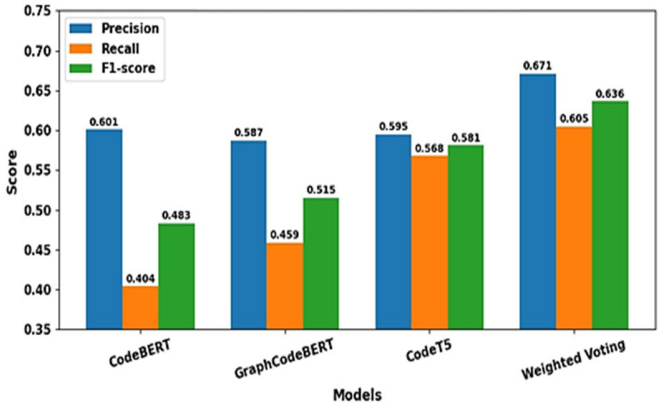

Fig. 4. Performance comparison of Weighted Voting vs baseline models.

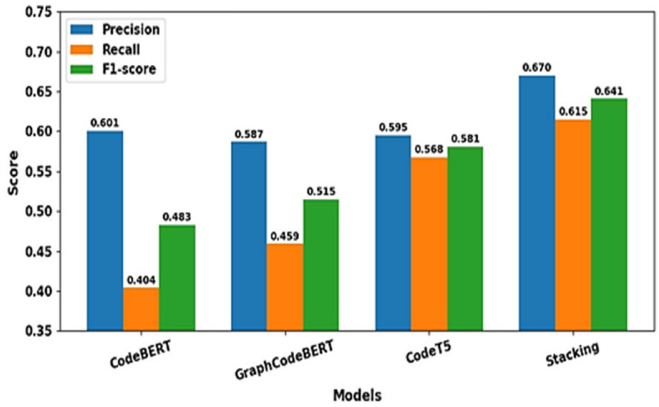

Fig. 5. Performance comparison of Stacking ensemble vs baseline models.

IV. CONCLUSIONS

This study aimed to investigate whether ensemble learning techniques could improve the precision of LLMs in the task of software vulnerability detection. The results confirm that ensemble methods not only improve precision but also contribute to better overall balance across recall and F1-score. To address false positives, Majority Voting proved effective, while Weighted Voting and Stacking achieved improvements in recall and F1, making them better suited for applications where capturing vulnerabilities is critical. This study also confirms that single models like CodeBERT, GraphCodeBERT, or CodeT5 have limitations when they are used standalone for vulnerability detection. In addition, the study also reveals that ensemble methods can be applied systematically to address these deficiencies, thereby providing a practical way for future development of security solutions with such LLMs.

Although this study demonstrates the potential of ensemble learning in this domain, several limitations should be acknowledged. First, the experiments were conducted using only the Devign dataset, which may not fully represent the diversity of vulnerabilities found in real-world software systems. Second, the evaluation was limited to three baseline

models and a small number of ensemble strategies. As a result, the findings may not generalize to newer LLM-based architectures or to broader datasets involving different programming languages, vulnerability types, and code structures. Another limitation is that the study evaluates the ensemble as a combined prediction framework without conducting an ablation analysis to measure the individual contribution of each base model. Future research should examine the effect of removing, replacing, or pairing different models within the ensemble to determine how each component influences overall performance. Such analysis could also help assess the robustness of ensemble-based vulnerability detection against adversarial attacks. Future work should also explore hybrid approaches that integrate static and dynamic analysis, along with the use of larger and more diverse datasets.

DECLARATION ON COMPETING INTERESTS

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

ACKNOWLEDMENT

No institutional, governmental, or commercial funding was received for this work.

DATA AVAILABILITY

The dataset used in this study is publicly available and can be accessed through the Devign dataset repository on Hugging Face [16].

REFERENCES

[1] W. Li, S. Manickam, Y. Chong, and S. Karuppayah, "Talking Like a Phisher: LLM-Based Attacks on Voice Phishing Classifiers." arXiv, July 22, 2025, https://doi.org/10.48550/arXiv.2507.16291.

[2] S. Illindala and S. Sabie, "Assessment of the Practicality of LLMs in the Field of Cybersecurity and Detection of Malicious Code," *American Journal of Student Research*, pp. 177–186, 2025, https://doi.org/10.70251/HYJR2348.34177186.

[3] K. Gladkikh and A. A. Zakharov, "Approach to Forming Vulnerability Datasets for Fine-Tuning AI Agents," in *2025 International Russian Smart Industry Conference (SmartIndustryCon)*, Mar. 2025, pp. 771–776, https://doi.org/10.1109/SmartIndustryCon65166.2025.10986048.

[4] J. Saxe and K. Berlin, "eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys." arXiv, Feb. 27, 2017, https://doi.org/10.48550/arXiv.1702.08568.

[5] Z. Li *et al.*, "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection," in *Proceedings 2018 Network and Distributed System Security Symposium*, 2018, https://doi.org/10.14722/ndss.2018.23158.

[6] J. Gui *et al.*, "Deep Anomaly Detection of Temporal Heterogeneous Data in AIOps: A Survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 26, no. 9, pp. 1551–1576, Sept. 2025, https://doi.org/10.1631/FITEE.2400467.

[7] A. Shestov *et al.*, "Finetuning Large Language Models for Vulnerability Detection," *IEEE Access*, vol. 13, pp. 38889–38900, 2025, https://doi.org/10.1109/ACCESS.2025.3546700.

[8] 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 *Advances in Neural Information Processing Systems*, 2019.

[9] V. Akuthota, R. Kasula, S. T. Sumona, M. Mohiuddin, M. T. Reza, and M. M. Rahman, "Vulnerability Detection and Monitoring Using LLM," in *2023 IEEE 9th International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, Nov. 2023, pp. 309–314, https://doi.org/10.1109/WIECON-ECE60392.2023.10456393.

[10] V. Nguyen, S. Nepal, X. Yuan, T. Wu, and C. Rudolph, "SAFE: A Novel Approach For Software Vulnerability Detection from Enhancing The Capability of Large Language Models," in *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, May 2025, pp. 392–406, https://doi.org/10.1145/3708821.3736208.

[11] H. Hanif and S. Maffeis, "VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection," in *2022 International Joint Conference on Neural Networks (IJCNN)*, July 2022, pp. 1–8, https://doi.org/10.1109/IJCNN55064.2022.9892280.

[12] M. Fu and C. Tantithamthavorn, "LineVul: a transformer-based line-level vulnerability prediction," in *Proceedings of the 19th International Conference on Mining Software Repositories*, July 2022, pp. 608–620, https://doi.org/10.1145/3524842.3528452.

[13] Y. Luo, W. Xu, and D. Xu, "Detecting code vulnerabilities with heterogeneous GNN training," *International Journal of Information Security*, vol. 24, no. 5, Sept. 2025, Art. no. 213, https://doi.org/10.1007/s10207-025-01132-x.

[14] M. M. Abualhaj, S. N. Al-Khatib, M. A. Zyoud, I. Qaddara, and M. Anbar, "Enhancing Intrusion Detection System Performance Using a Hybrid of Harris Hawks and Whale Optimization Algorithms," *Engineering, Technology & Applied Science Research*, vol. 15, no. 4, pp. 24354–24361, Aug. 2025, https://doi.org/10.48084/etasr.10919.

[15] S. Elsayed, K. Mohamed, and M. A. Madkour, "A Comparative Study of Using Deep Learning Algorithms in Network Intrusion Detection," *IEEE Access*, vol. 12, pp. 58851–58870, 2024, https://doi.org/10.1109/ACCESS.2024.3389096.

[16] "DetectVul/devign." Hugging Face, [Online]. Available: https://huggingface.co/datasets/DetectVul/devign.