MATHEMATICAL MODELING AND COMPUTING, Vol. 12, No. 4, pp. 1077–1086 (2025)

Advanced Approaches for Vulnerability Detection in Solidity-Based Smart Contracts: A Comparative Review

El Inani N.[1,*], Benabbou F.[1], Sabiri K.[2], Zaouch A.[1]

[1] *Artificial Intelligence and Systems Laboratory, University Hassan II of Casablanca, Faculty of Sciences Ben M'Sik, Morocco*
[2] *Collaborative Laboratory Mountains of Research (MORE), Portugal*
[*] *Corresponding author: najwa.elinani-etu@etu.univh2c.ma*

(Received 28 March 2025; Revised 4 October 2025; Accepted 8 October 2025)

With the advancement of blockchain technology, Solidity-based smart contracts have become essential for automating and securing digital transactions across various sectors, from finance to supply chain management. These contracts enable decentralized exchanges without intermediaries, enhancing transparency. However, their immutable nature poses security challenges: any flaw in the code becomes permanent, exposing contracts to attacks and leading to financial and reputational losses. This paper provides a comparative analysis of recent machine learning (ML) and deep learning (DL) techniques developed for detecting vulnerabilities in Solidity based smart contracts. By evaluating various approaches, we assess their effectiveness in identifying common threats such as reentrancy attacks and integer overflows. Finally, we highlight the importance of scalable, AI driven security solutions to address the growing complexity of vulnerabilities.

Keywords: *blockchain; vulnerability; smart contract; machine learning; deep learning.*
2010 MSC: 91B40, 68M14, 93A14, 97P10, 97R40 **DOI:** 10.23939/mmc2025.04.1077

1. Introduction

Blockchain technology has rapidly evolved, revolutionizing industries by providing secure, decentralized solutions for various applications. Originally utilized primarily within the realm of cryptocurrencies, blockchain technology has since branched out significantly, particularly with the advent of smart contracts on second generation platforms like Ethereum. Smart contracts are autonomous programs that execute predefined actions when certain conditions are met. Their use has expanded beyond financial transactions to applications in sectors such as healthcare, supply chain, and governance [1, 2]. Given their ability to manage and transfer substantial value without intermediaries, the security of these contracts has become a critical concern. Vulnerabilities in smart contracts can lead to significant financial losses, undermining trust in blockchain systems and potentially harming the users and organizations that rely on them [3].

These vulnerabilities can generally be categorized into three main areas. First, there are vulnerabilities within the blockchain dafase itself, where issues such as the '51% attack' [4] can compromise data integrity by allowing attackers to control the majority of mining power, thus manipulating transaction records. Second, there are risks associated with the execution environment, such as Ethereum's EVM (Ethereum Virtual Machine), where attackers may exploit transaction ordering or gas usage to gain an unfair advantage. Lastly, vulnerabilities also exist within the source code, particularly in the Solidity programming language used for writing smart contracts [5] . Despite their potential, Ethereum blockchain-based smart contracts are particularly susceptible to various vulnerabilities, primarily due to the complexity of coding and the immutable nature of the blockchain [6]. Common vulnerabilities include reentrancy attacks, integer overflows, and others that can have severe repercussions, including unauthorized fund transfers and smart contract failure. These flaws occur at three levels, starting from the code development stage where incorrect coding practices lead to security risks to how these contracts interact with the Ethereum Virtual Machine (EVM) and, subsequently, the blockchain database.

Traditional methods of vulnerability detection, such as manual code audits and static analysis tools, often fall short due to the evolving nature of smart contracts and their susceptibility to increasingly sophisticated attacks [7, 8]. Approaches based on Machine Learning (ML) and Deep Learning (DL) models have emerged as promising tools for the detection of vulnerabilities in Solidity code. By leveraging large datasets and learning patterns within smart contract code, ML and DL approaches provide enhanced capabilities for identifying weaknesses that traditional methods may overlook. These models can efficiently adapt to new types of attacks by learning from past vulnerabilities.

This review paper explores the application of ML and DL models in detecting vulnerabilities within Solidity-based smart contracts. It aims to analyze the effectiveness of these advanced models compared to traditional methods and evaluate how they can improve security measures. The paper is structured as follows: Section 2 reviews related works, tracing the evolution and recent progress in using ML and DL to analyze smart contracts. Section 3 provides a comparative analysis of existing ML and DL models, assessing their effectiveness and limitations. Section 4 concludes with key findings, discusses the implications for future security enhancements in smart contracts, and offers recommendations for ongoing research.

2. Literature review

This literature review interests in recent methods based on ML an DL. The detection of vulnerabilities in smart contracts leverages a combination of ma-chine learning (ML) and deep learning (DL) approaches. These techniques are used to learn and recognize patterns within the contract code, with the goal of identifying potential security issues more effectively than conventional approaches. However, many models still incorporate traditional methods.

Feng et al. [9] combined a static technic and Machine learning model to develop an interpretable model for detecting vulnerabilities in large scale smart contracts. Specifically, the model leverages static analysis of opcode sequences, followed by feature engineering, and building a detection model using XGBoost. The dataset con-sists of 40 000 Ethereum smart contracts, and the model achieved notable perfor-mance with accuracies between 93% and 100% in detecting six common vulnerabilities. Similarly, Jie et al. [10], used static analysis at the function level through Slither, which extracts features from the compiled code of smart contracts. The framework organized tasks into intramodal and intermodal settings, using Bi-LSTM, textCNN models, and Random Forest (RF) for feature selection and fusion. By integrating the modalities, the framework achieves an accuracy of 99.71% and an F1-score of 99.50%. Xie et al. [11] introduced the Blockgram model to analyze and extract features by segmenting opcode sequences into blocks. The features are used with machine learning models such as XGBoost, RF, and K-Nearest Neighbors (KNN) to classify vulnerabilities. Evaluated on the NKSC and ContractWard datasets, the RF model achieved an accuracy of 94.36%, a recall of 98.92%, and an F1-score of 96.80%, with a significant reduction in latency compared to N-gram features. In a related research, Ren et al. [12] proposed using program slicing technique to isolates code portions relevant to potential vulnerabilities, and the Bi-LSTM-Att model analyzes these slices, using bidirectional processing and an attention mechanism to effectively extract sequential and contextual information's.

This approach consistently outperformed the other models across all vulnerability categories. For integer overflow vulnerability, Bi-LSTM-Att achieved a recall of 80.85%, a precision of 87.04%, and an F1-score of 83.83%, which was significantly higher than the baseline LSTM and GRU models. For re-entrancy vulnerabilities, Bi-LSTM-Att attained the highest recall (86.53%), precision (87.45%), and F1-score (86.99%). Similarly, for dangerous delegatecall, Bi-LSTM-Att showed improved recall, precision, and F1-score, reaching values of 79.46%, 83.05%, and 81.22%, respectively. The best achieved performance is for timestamp dependence vulnerabilities, where Bi-LSTM-Att achieved a recall of 89.48%, precision of 94.40%, and an F1-score of 91.87%, Zhang et al. [13] proposed the Serial-Parallel Convolutional Bidirectional Gated Recurrent Network Model incorporating Ensemble Classifier (SPCBIG-EC), which, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) based models, to enhance accuracy and robustness. The methodology begins with feature extraction,

where smart contract code is converted into vector representations before building the model using. Multiple datasets containing different types of vulnerabilities. This approach demonstrated interesting results, achieving an F1-score of 96.74% for reentrancy vulnerabilities, 91.62% for timestamp dependency, and 95.00% for infinite loop vulnerabilities, with overall accuracy and F1-scores above 85% for hybrid datasets. Another work of Zhang et al. [14] introduced a SVScanner method, a novel deep learning based approach. Global semantic features are extracted using a Bi-LSTM model from the code tokens, while deep structural semantics are captured through an attention mechanism applied to the Abstract Syntax Tree (AST). These combined features are then fed into a TextCNN to classify contracts as either vulnerable or non-vulnerable. This approach achieved an accuracy of 94.77%, a recall of 90.72%, and an F1-score of 94.64%, demonstrating its effec-tiveness in smart contract vulnerability detection.

Additionally Zhou et al. [15] pre-sented Multiple Objective Detection Neural Network (MODNN), that involves using pre-trained models BERT to extract explicit features from sequences of operations and constructing co-occurrence matrices for implicit feature learning. The MODNN was trained on a dataset of over 18 000 smart contracts, and its performance was evaluated using metrics like the F1-score. It demonstrated a high accuracy rate, reaching 96.15% for detecting underflow vulnerabilities, and a recall rate of 99.16%. Another methodology proposed by Osei et al. [16] introduced Wide and Deep Neural Network (WIDENNET), an approach based on deep learning model including linear wide model to capture simple patterns with a deep neural network to learn complex, hierarchical features, optimized together using a joint loss function. The model uses Stochastic Gradient Descent (SGD) for backpropagation and softmax for final classification, to address existing vulnerability's. In experimental evaluations, WIDENNET archieved an accuracy of 80.06%. Similarly Sun et al. [17] introduces Active and Semi-Supervised Bert (ASSBert), a novel framework that combines active and semi-supervised learning techniques to address the limitations in smart contract vulnerability detection. The ASSBert model was validated using a benchmark dataset of 20 829 smart contracts, containing six types of vulnerabilities, and demonstrated superior results compared to traditional baseline methods with a F1-score of 89%. Cheng et al. [18] propose a new framework called Enhanced Graph Feature Learning (EGFL). That represents smart contract bytecode as a Control Flow Graph (CFG) and constructs a linear node feature matrix to provide global context. To manage long-range dependencies, it uses feature-aware and relationship-aware modules, while a graph neural network extracts local features. These global and local features are then combined through a self-attention mechanism, and classification is performed using a Multi Layer Perceptron (MLP). The approach was compared to against 14 state-of-the-art methods using a benchmark dataset, achieving an accuracy, recall, precision, and F1-score (ranging from 83.67% to 90.47% varing across different types of vulnerabilities). Cai et al. [19] proposed a new dataset statement-level annotations derived from security best practices in smart contract code. The proposed approach Syntax-Sensitive Graph Neural Network (SS-GNN) combines tree based learning via Child-Sum Tree-LSTM, Graph Convolutional Networks (GCN), and attention mechanisms to capture comprehensive code relationships. A Multi Layer Perceptron (MLP) classifier is used for final prediction. The SS-GNN outperformed other tools such as Slither, Smartcheck, and TMP [20], with an F1-score of 90.57%, as well as interesting accuracy, recall, and precision, demonstrating its superiority in vulnerability detection at a detailed statement-level. While traditional methods are still utilized, advanced models leveraging a combination of static analysis, feature extraction, and neural networks have demonstrated superior performance, making them crucial for addressing the growing complexity and security challenges in blockchain technologies. In following section, a deep comparison is conducted.

3. Comparison and analysis

The literature offers a comprehensive overview of smart contract vulnerabilities and detection models. However, a comparative approach is necessary to pinpoint strengths and weaknesses across different

vulnerability levels. In this section, we will delve deeper into these levels, with a particular focus on vulnerabilities at the Solidity level.

Furthermore, while ML and DL models have seen application across various domains, their comparative effectiveness in detecting vulnerabilities remains underexplored, a gap this section seeks to address.

3.1. Vulnerability levels

In Ethereum blockchain technology, vulnerabilities can exist at various levels, each presenting unique challenges to the security of decentralized applications. In this section, Table 1 presents multiple types of vulnerabilities within the Ethereum platform at each level.

At the Blockchain level, issues like Block Timestamp Dependency, can be exploited by manipulating the block time to alter the contract's behavior, while Unsecured Balance may allow attackers to drain funds from inadequately protected accounts. Moving to the Ethereum Virtual Machine (EVM) level, vulnerabilities include the Call Stack Depth Limitation, which can prevent critical functions from executing correctly if the stack becomes too deep, and Unchecked and Failed Send, where insufficient checks on fund transfers could lead to losses. Issues like Short Address Attack exploit incomplete transaction data, and using tx.origin for authentication creates risks of impersonation attacks [21].

Solidity-level vulnerabilities are highly prevalent due to the direct exposure of smart contract code to attackers. These vulnerabilities are divided into two categories: heavy and light harm. Heavy harm vulnerabilities include issues like Integer Over/Under Flow, which alters arithmetic operation results, Re-Entrancy Problems, where attackers repeatedly call functions to drain funds, and Greedy Contracts, where assets become permanently locked. Detection of these vulnerabilities involves advanced models such as SPCBIG-EC, SVScanner, and WIDENNET, which leverage deep learning techniques like CNN, RNN, and Bi-LSTM to analyze patterns in op-code or feature data. For instance, Ren et al.'s Bi-LSTM-Att model captures both contextual and sequential information, improving the detection of re-entrancy and arithmetic problems by analyzing contract semantics in detail. On the other hand, light harm vulnerabilities, such as Denial of Service, Gas Overspent, Transaction Ordering, and other risks like Prodigal Contracts and Suicidal Contracts, represent less severe but still impactful threats. These vulnerabilities are efficiently detected using models like Feng et al.'s XGBoost-based approach, which uses static analysis and feature engineering to reduce false positives and improve detection efficiency. Jie et al. also utilize Slither for function-level static analysis, followed by textCNN and bi-LSTM for deeper feature examination, achieving high accuracy in identifying common vulnerabilities. Overall, Solidity-level vulnerabilities are critical due to their direct impact on smart contract behavior and fund security. By combining ma-chine learning techniques like XGBoost and Random Forest with deep learning models such as CNN and Bi-LSTM, detection models aim to improve accuracy, interpretability, and speed, helping developers secure smart contracts more effectively. Table 1 presents the types of vulnerabilities across each category. In the following section, we focus specifically on the Solidity level to explore advanced tools for vulnerability detection.

3.2. Approaches performance comparison

This section presents the various models and their performance to detect the issues in solidity level. Table 2 provides a comprehensive synthesis of multiple methods used for detecting vulnerabilities in smart contracts based on: name of the approach, ML/DL used, Feature extraction technique based on, Dataset and the achieved performance using metrics such as: accuracy (A), F1-score (F1), Precision (P) or Recall (R).

3.2.1. Datasets

Among the various datasets and models discussed, the CESC + UCESC dataset, the largest with 203 716 contracts, is likely to contribute to the most powerful performance. Models trained on such large datasets, including advanced deep learning models like SPCBIG-EC and Bi-LSTM-Att, consistently achieve superior results. For example, Bi-LSTM-Att outperformed other models with F1-scores

Table 1. Table of vulnerability levels.

Levels	Types	Sub-type	Description
Solidity	Heavy Harm Vulnerabilities	Integer Over/Under Flow	Errors from exceeding numeric limits in arithmetic operations.
		Re-Entrancy Problem	External contract calls can alter the original contract's state unexpectedly, leading to exploits.
		Greedy Contract	Reliance on external libraries can render a contract inoperable if those libraries are terminated by attackers.
	Light Harm Vulnerabilities	Denial of Service	Reliance on external function results for state transfer.
		Gas Overspent	Unnecessary code increases gas costs, wasting resources.
		Transaction Ordering	Execution order is unpredictable due to mining order, causing potential exploits.
		Prodigal Contract	Incorrectly refunds or transfers funds to unintended parties.
		Destroyable/Suicidal Contract	Contracts can be destructed, sometimes by unauthorized users, causing loss of functionality.

Table 2. Summary of models, approaches, and performances in vulnerability detection.

Refю	Approach	Model	Feature Extraction	Dataset	Performance
[9]	Static analysis with feature selection and XGBoost	XGBoost	Slither	Ethereum platform	F: 93%–100%
[11]	Feature reduction with Block-gram and ML models	XGBoost, RF, KNN	N-gram	NKSC	A: 94.36%, R: 98.92%, F: 96.80%
[12]	Bi-LSTM-Att	Bi-LSTM	Bi-LSTM-Att	Collected from GitHub	R: 80.85%–89.48%, F: 81.22%–91.87%
[14]	SPCBIG-EC	Hybrid CNN-RNN	RNN	CESC + UCESC	F: 91.62%–96.74%
[13]	SVScanner	Bi-LSTM	RNN+LSTM	Ethereum platform	A: 94.77%, F: 94.64%
[15]	MODNN	BERT deep neural network	Bert	Ethereum platform	A: 96.15%, F: 94.8%
[16]	WIDENNET	BERT	Bert	Ethereum platform	A: 80.06%
[17]	ASSBert	BERT	Bert	SC-Base + SC-Val	F: 89%
[18]	EGFL	GNN	GNN	XBlock platform	F: 83.67%–90.47%
[19]	SS-GNN	GNN	GNN	VNT Chain smart contract	F: 90.57%

above 90% across multiple vulnerability categories. Additionally, the Ethereum platform dataset, with 40 000 contracts, has also demonstrated high performance in models like XGBoost and SVScanner, achieving accuracies between 93% and 100%. These models show strong performance in detecting common vulnerabilities like reentrancy and timestamp dependence. Although NKSC is a smaller dataset with 33 885 contracts, the Block-gram model trained on it has performed exceptionally well. It achieved 94.36% accuracy and substantial reductions in latency, showing that specialized models can still deliver high performance even with smaller datasets. This figure provides an overview of the dataset's size and distribution, showcasing how many smart contracts were examined for vulnerability detection. By depicting the volume of contracts processed, Figure 1 highlights the scale of the analysis, which supports the reliability and generalizability of the findings related to model performance in detecting vulnerabilities across different types of smart contracts.

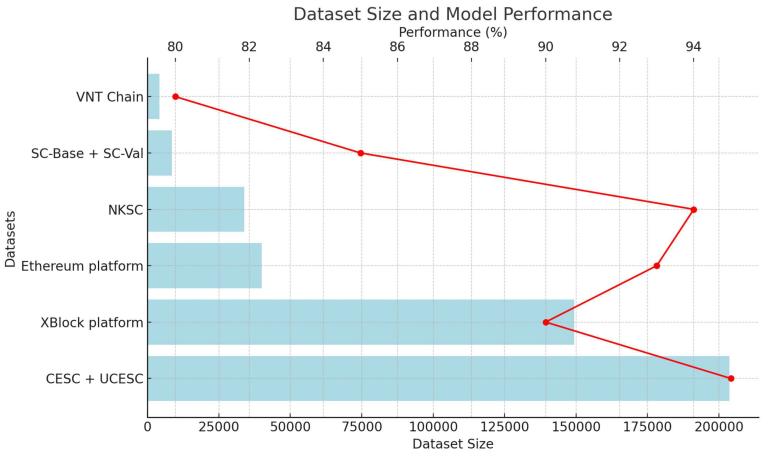

Fig. 1. Dataset size and model performance.

3.2.2. Preprocessing techniques

The detection process includes multiple stages to prepare the data after collection and prior to modeling. The process begins with data cleaning to eliminate unnecessary elements, followed by tokenization, and concludes with the extraction of essential features. Figure 2 illustrates the Preprocessing Tasks.

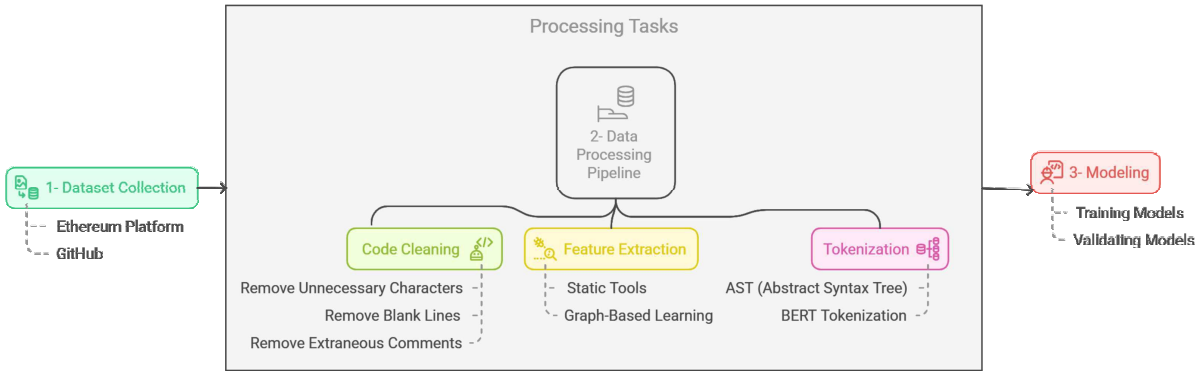

Fig. 2. Chart of preprocessing pipeline.

— **Cleansing**

Effective vulnerability detection in smart contracts relies heavily on robust data cleansing and preprocessing strategies to ensure high-quality input data for ML/DL models [22]. Most state-of-the-art approaches utilize static analysis tools to remove irrelevant features, filter noisy data, and prepare structured datasets. For instance, Slither, widely employed for static analysis, provides feature extraction by analyzing opcode sequences and detecting potential vulnerabilities at the code level. Similarly, N-gram techniques, as seen in Xie et al.'s Blockgram model, further enhance cleansing by segmenting opcode sequences into meaningful blocks for feature reduction. Moreover, program slicing techniques, as utilized in the Bi-LSTM-Att model, isolate only the relevant portions of code, thus reducing dataset complexity and improving training efficiency.

These methods underscore the importance of tailored cleansing techniques for specific vulnerability detection tasks. Among the various tools and techniques, Slither emerges as a standout choice

due to its consistent integration with ML/DL models like XGBoost and its ability to preprocess Ethereum smart contracts effectively. This tool has shown to support high-performing models, achieving accuracies ranging from 93% to 100% across multiple vulnerability categories, thereby making it a reliable and efficient solution for the dataset cleansing stage.

— **Tokenization**

Tokenization is a critical step in preparing smart contract code for machine learning and deep learning models [23]. It involves breaking down code into smaller, interpretable units, such as tokens or embeddings, that can be processed by the models. Different tokenization strategies are utilized depending on the model architecture and feature extraction requirements. For instance, N-gram techniques, used in Xie et al.'s Blockgram model, tokenize opcode sequences into fixed-length segments, allowing machine learning models like XGBoost, RF, and KNN to capture local patterns effectively. Similarly, transformer-based models such as BERT rely on advanced tokenization methods like WordPiece embeddings, which capture both subword-level semantics and contextual relationships within the code.

Graph-based models, such as Syntax-Sensitive Graph Neural Networks (SS-GNN) and Enhanced Graph Feature Learning (EGFL), tokenize the smart contract bytecode by representing it as Control Flow Graphs (CFGs) or Abstract Syntax Trees (ASTs). These structural tokenization approaches enable the models to extract hierarchical and contextual information, leading to enhanced performance in detecting vulnerabilities like reentrancy and integer overflows.

Among the various tokenization strategies, WordPiece tokenization used in BERT-based models has demonstrated superior performance in several studies. For example, MODNN and ASSBert, which leverage BERT embeddings, achieved exceptional F1-scores of 96.15% and 89%, respectively, highlighting the effectiveness of transformer-based tokenization in capturing complex patterns in smart contract data.

— **Feature extraction methods in smart contract vulnerability detection**

Feature extraction serves as the foundation for building high-performing models, enabling the identification of key patterns and relationships within smart contract code [24]. Different models leverage unique strategies to extract features that are most relevant to detecting vulnerabilities. Traditional approaches, such as static analysis (e.g., Slither), are often employed to extract opcode sequences, which are then transformed into meaningful representations using techniques like N-gram segmentation. For instance, the Blockgram model utilizes segmented opcode sequences to capture sequential relationships, enabling machine learning models such as XGBoost, RF, and KNN to deliver high accuracy, as demonstrated by the RF model's F1-score of 96.80%.

Advanced methods, particularly in deep learning frameworks, focus on leveraging both contextual and semantic information. The Bi-LSTM-Att model, for example, employs an attention mechanism to extract sequential and contextual patterns, achieving remarkable precision and recall rates, such as 89.48% recall and 94.40% precision for timestamp dependency vulnerabilities. Similarly, graph-based models, including EGFL and SS-GNN, utilize Control Flow Graphs (CFGs) and Abstract Syntax Trees (ASTs) to extract both local and global features, ensuring comprehensive vulnerability detection. The EGFL framework's ability to achieve accuracies ranging from 83.67% to 90.47% across various vulnerability types underscores the effectiveness of graph-based feature extraction.

Moreover, transformer-based models like BERT have revolutionized feature extraction by providing pre-trained embeddings that encapsulate both structural and contextual information. For example, the MODNN model utilizes BERT embeddings combined with co-occurrence matrices, achieving an impressive F1-score of 96.15% for underflow vulnerability detection.

Among these methods, the use of graph-based feature extraction (e.g., CFGs, ASTs) stands out for its ability to capture intricate relationships within the code. However, the superior performance of BERT-based embeddings, as seen in MODNN and ASSBert, highlights the transformative potential of pre-trained models in capturing rich, multi-dimensional features for vulnerability detection.

3.2.3. ML and DL used, the most used, and the best performing in each category

The machine learning models utilize a hybrid approach, integrating static opcode sequence analysis with XGBoost and hyperparameter optimization, achieving accuracy rates ranging from 93% to 100%. Additionally, the Block-gram model focuses on efficiency and feature reduction, delivering strong results with an accuracy of 94.36% and a recall of 98.92%. In contrast, deep learning models exhibit superior performance in vulnerability detection tasks. One model introduces Bi-LSTM-Att, which uses program slicing and achieves recall rates exceeding 80%, outperforming LSTM and GRU baselines. Another proposes SPCBIG-EC, a hybrid CNN-RNN model, achieving impressive F1-scores (96.74%, 91.62%, 95.00%) across different vulnerabilities. Similarly, SVScanner combines Bi-LSTM and attention mechanisms, delivering an accuracy of 94.77% and an F1-score of 94.64%. Another model presents MODNN, using BERT for feature extraction, achieving an F1-score of 94.8% and an accuracy of 96.15%. Other models like WIDENNET achieve 80.06% accuracy, while ASSBert integrates active and semi-supervised learning for enhanced accuracy. The EGFL model, which incorporates Graph Neural Networks (GNNs) with self-attention, demonstrates substantial improvements, with F1-score gains of up to 60.28%. Lastly, SS-GNN, a graph based learning approach, outperforms existing tools like Slither and Smartcheck with an F1-score of 90.57%. Figure 3 provides a comparison of various models used for identifying vulnerabilities within smart contracts using the F1-score metrics. The F1-score is chosen for problems like vulnerability detection in smart contracts because it effectively balances the model's ability to identify vulnerabilities accurately (high recall) while ensuring those detections are reliable (high precision).

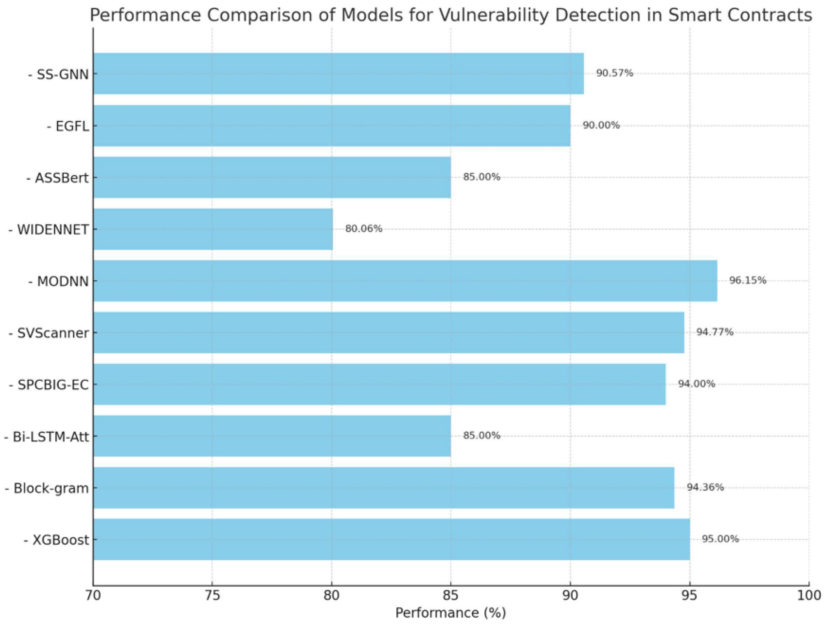

Fig. 3. General vulnerability metrics for each ML/DL model.

Comparing these models, deep learning techniques outperform traditional machine learning approaches, particularly in handling complex vulnerability patterns. Among the deep learning models, SPCBIG-EC and MODNN stand out due to their robust performance across various vulnerabilities, with SPCBIG-EC demonstrating balanced performance in precision, recall, and F1-scores. EGFL also shows significant improvements, particularly in managing long-range dependencies. SPCBIG-EC emerges as the most effective approach, providing a comprehensive solution with high generalizability, while MODNN is also highly effective in detecting both known and novel vulnerabilities. In conclusion, deep learning, particularly hybrid models like SPCBIG-EC, represents the most promising direction for smart contract vulnerability detection, given the increasing complexity of blockchain technologies.

4. Conclusion

The rapid advancement of blockchain technology, particularly with the integration of smart contracts, has brought significant opportunities but also substantial risks due to vulnerabilities inherent in the code. This paper has reviewed a variety of machine learning and deep learning models that have been developed to detect vulnerabilities in smart contracts, highlighting the advantages of hybrid approaches that combine static and ML, DL approaches. Our analysis shows that models such as

bi-LSTM, XGBoost, and graph neural networks (GNNs) significantly outperform traditional methods, offering higher accuracy and better generalization in vulnerability detection. While the current machine learning models provide promising results, challenges remain, especially in ensuring scalability and adapting to the fast-evolving landscape of blockchain technologies. Future research should focus on enhancing the robustness of these models and expanding their ability to detect new and more sophisticated vulnerabilities. By leveraging AI advancements, blockchain security can be greatly improved, ensuring that smart contracts become even more reliable and secure in critical applications such as decentralized finance (DeFi) and beyond.

[1] Sarmah S. S. Understanding Blockchain Technology. Computer Science and Engineering (2018).

[2] Tripathi G., Ahad M. A., Casalino G. A comprehensive review of blockchain technology: Underlying principles and historical background with future challenges. Decision Analytics Journal. **9**, 100344 (2023).

[3] Kushwaha S. S., Joshi S., Singh D., Kaur M., Lee H.-N. Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract. IEEE Access. **10**, 6605–6621 (2022).

[4] Aponte-Novoa F. A., Orozco A. L. S., Villanueva-Polanco R., Wightman P. The 51% Attack on Blockchains: A Mining Behavior Study. IEEE Access. **9**, 1405491–40564 (2021).

[5] Jamwal S., Cano J., Lee G. M., Tran N. H., Truong N. A survey on Ethereum pseudonymity: Techniques, challenges, and future directions. Journal of Network and Computer Applications. **232**, 104019 (2024).

[6] Hu B., Zhang Z., Liu J. et al. A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems. Patterns. **2** (2), 100179 (2021).

[7] Taş R. Smart Contract Security Vulnerabilities. Erzincan University Journal of Science and Technology. **16** (1), 196–211 (2023).

[8] Tang X., Zhou K., Cheng J., Li H., Yuan Y. The Vulnerabilities in Smart Contracts: A Survey. Advances in Artificial Intelligence and Security. 177–190 (2021).

[9] Feng X., Liu H., Wang L., Zhu H., Sheng V. S. An interpretable model for large-scale smart contract vulnerability detection. Blockchain: Research and Applications. **5** (3), 100209 (2024).

[10] Jie W., Chen Q., Wang J., Koe A. S. V. et al. A novel extended multimodal AI framework towards vulnerability detection in smart contracts. Information Sciences. **636**, 118907 (2023).

[11] Xie X., Wang H., Jian Z., Fang Y., Wang Z., Li T. Block-gram: Mining knowledgeable features for efficiently smart contract vulnerability detection. Digital Communications and Networks. **11** (1), 1–12 (2023).

[12] Ren X., Wu Y., Li J., Hao D., Alam M. Smart contract vulnerability detection based on a semantic code structure and a self-designed neural network. Computers and Electrical Engineering. **109** (B), 108766 (2023).

[13] Zhang H., Zhang W., Feng Y., Liu Y. SVScanner: Detecting smart contract vulnerabilities via deep semantic extraction. Journal of Information Security and Applications. **75**, 103484 (2023).

[14] Zhang L., Li Y., Jin T., Wang W. et al. SPCBIG-EC: A Robust Serial Hybrid Model for Smart Contract Vulnerability Detection. Sensors. **22** (12), 4621 (2022).

[15] Zhou K., Huang J., Han H., Gong B. et al. Smart contracts vulnerability detection model based on adversarial multi-task learning. Journal of Information Security and Applications. **77**, 103555 (2023).

[16] Osei S. B., Ma Z., Huang R. Smart contract vulnerability detection using wide and deep neural network. Science of Computer Programming. **238**, 103172 (2024).

[17] Sun X., Tu L., Zhang J., Cai J., Li B., Wang Y. ASSBert: Active and semi-supervised bert for smart contract vulnerability detection. Journal of Information Security and Applications. **73**, 103423 (2023).

[18] Cheng J., Chen Y., Cao Y., Wang H. A vulnerability detection framework with enhanced graph feature learning. Journal of Systems and Software. **216**, 112118 (2024).

[19] Cai J., Li B., Zhang T., Zhang J., Sun X. Fine-grained smart contract vulnerability detection by heterogeneous code feature learning and automated dataset construction. Journal of Systems and Software. **209**, 111919 (2024).

[20] Wu G., Wang H., Lai X., Wang M., He D., Chan S. A comprehensive survey of smart contract security: State of the art and research directions. Journal of Network and Computer Applications. **226**, 103882 (2024).

[21] Vidal F. R., Ivaki N., Laranjeiro N. Vulnerability detection techniques for smart contracts: A systematic literature review. Journal of Systems and Software. **217**, 111919 (2024).

[22] Hu K., Zhu J., Ding Y., Bai X., Huang J. Smart contract engineering. Electronics. **9** (12), 2042 (2020).

[23] Aziz R. M., Mahto R., Goel K., Das A., Kumar P., Saxena A. Modified Genetic Algorithm with Deep Learning for Fraud Transactions of Ethereum Smart Contract. Applied Sciences. **13** (2), 697 (2023).

[24] Xu G., Liu L., Dong J. Vulnerability Detection of Ethereum Smart Contract Based on SolBERT-BiGRU-Attention Hybrid Neural Model. CMES – Computer Modeling in Engineering and Sciences. **137** (1), 903–922 (2023).

Передові підходи до виявлення вразливостей у смарт-контрактах на основі Solidity: порівняльний огляд

Ель Інані Н.[1], Бенаббу Ф.[1], Сабірі К.[2], Заух А.[1]

[1] *Лабораторія штучного інтелекту та систем, Університет Хасана II Касабланки, Факультет природничих наук Бен Мсік, Марокко*
[2] *Спільна лабораторія досліджень у горах (MORE), Португалія*

З розвитком технології блокчейн, смарт-контракти на основі Solidity стали важливими для автоматизації та захисту цифрових транзакцій у різних секторах, від фінансів до управління ланцюгами поставок. Ці контракти дозволяють здійснювати децентралізовані обміни без посередників, підвищуючи прозорість. Однак їхня незмінна природа створює проблеми безпеки: будь-який недолік у коді стає постійним, наражаючи контракти на ризик атак та призводячи до фінансових та репутаційних втрат. У цій статті наведено порівняльний аналіз нещодавніх методів машинного навчання (ML) та глибокого навчання (DL), розроблених для виявлення вразливостей у смарт-контрактах на основі Solidity. Аналізуючи різні підходи, оцінено їхню ефективність у виявленні поширених загроз, таких як атаки повторного входу та переповнення цілих чисел. Накінець, підкреслено важливість масштабованих рішень безпеки на основі штучного інтелекту для вирішення зростаючої складності вразливостей.

Ключові слова: *блокчейн; вразливість; смарт-контракт; машинне навчання; глибоке навчання.*