Detection Made Easy: The Potential of Large Language Models for Solidity Vulnerabilities

Odnala Srinivas1[0000-0002-3152-4750], Banavathu Rupathi Rao2, Karumanchi Bhavya Sri3, Telagathoti Jeevan Likhith Raj4, and Nihar Ranjan Pradhan5[0000-0003-2193-3101],

1,2,3,4,5 School of Computer Science and Engineering, VIT-AP University, Amaravathi, Andhra Pradesh, India

1[srinivas.23phd7003@vitap.ac.in](mailto:srinivas.23phd7003@vitap.ac.in),[2rupathirao95@gmail.com](mailto:2rupathirao95@gmail.com), [3bhavyasrikarumanchi@gmail.com](mailto:3bhavyasrikarumanchi@gmail.com), [4telagathotijeevan.1234@gmail.com](mailto:4telagathotijeevan.1234@gmail.com),5[nihar.pradhan@vitap.ac.in](mailto:nihar.pradhan@vitap.ac.in),

**Abstract.** Activists seeking to gain financial benefits have increasingly utilized the massive deployment of Solidity smart contracts on blockchain networks, leading to widespread security breaches. In this paper, a detailed investigation is presented to determine how Large Language Models (LLMs) and reinforcement learning techniques can be employed to identify OWASP Top Ten vulnerabilities in Solidity. VulSmart is a new, class-balanced, labeled, and unique dataset that we use to compare the performance of CodeLlama, Ll ama (Local law modeling), Llamas (class defense simulation) and Falcon (clinical problem solving) datasets with closed-source replicas like GPT-3.5. Turbo and GPT-4o Mini. The SmartVD framework we have proposed utilizes reinforcement learning to im- prove vulnerability detection accuracy incrementally. This is the case. Our re- search utilizes BLEU and ROUGE metrics to conduct comprehensive evaluations of various prompting strategies, such as zero-shot, few-shot, and chain-of- thought methods, to evaluate their influence on multi-class cataloging and gen- erative capabilities. According to our research, SmartVD is more effective than open-source models and even surpasses closed-source base models such as GPT-

3.5 and GPT-4o Mini. Notably, fine-tuned versions of GPT-3.5. The vulnerabil- ity detection accuracy of Turbo and GPT-4o Mini is 99% accurate, while vulner- ability types are identified at 94% and severity is determined at 90%. SmartVD is most effective with chain-of-thought prompting, whereas closed-source mod- els are best suited for zero-shot prompts. In addition, Solana blockchain technol- ogy improves processing efficiency and makes real-time smart contract security analysis more scalable.

**Keywords:** Smart Contract, Solidity Vulnerabilities, OWASP Top Ten, LLMs- Large Language Models, Reinforcement Learning, Blockchain Security, SmartVD Framework, VulSmart Dataset, Prompt Engineering.

# Introduction

Large Language Models (LLMs) are cultured artificial intelligence (AI) models that have been trained on enormous datasets. They can handle sophisticated natural

language processing tasks including automated reasoning, code analysis, and vulnera- bility detection. GPT-3.5, GPT-4o Mini, and CodeLlama are notable instances. In con- trast, Solidity is a high-level programming language made especially for creating Ethereum and other blockchain-based smart contracts. It enables programmers to de- sign decentralized applications (dApps) that eliminate the need for middlemen by au- tomating transactions. In the crucial area of smart contract security [20], where AI mod- els are used to identify and address flaws in blockchain codebases, LLMs [1] and So- lidity work together.

The rapid advancement of machine learning and natural language processing meth- ods, particularly through transformer-based architectures brought about by models such as BERT and GPT [2], is the foundation for the birth of LLMs. These models were greatly scaled as AI research progressed, producing strong systems that could compre- hend and produce text and code that resembled that of a person. In 2014, Solidity was created as a component of Ethereum's decentralized programmable contract concept. Languages like JavaScript, Python, and C++ had an impact on its architecture, which was especially made for blockchain contexts where decentralized trust is crucial and transactions are irreversible. Both Solidity and LLMs have developed over time, find- ing use in sectors that require a high level of automation, transparency, and security.Not with standing their potential, Solidity and LLMs both present formidable obstacles. Reentrancy attacks, unchecked call values, overflow/underflow issues, and transaction- ordering dependencies are just a few of the vulnerabilities that solidity contracts fre- quently have, which can result in significant financial exploitation like the DAO assault [3]. In the meanwhile, if trained on unbalanced or inadequate datasets, LLMs may mis- classify vulnerabilities and are prone to hallucinations when used for security tasks. They also produce inaccurate responses with high confidence. Furthermore, LLMs might not have the domain-specific expertise needed to properly audit smart contracts without specialized fine-tuning. These problems cast serious doubt on the reliability and integrity of the language and the AI models that protect it.

A diversified strategy is needed to address these challenges. Vulnerability discovery in Solidity can be greatly enhanced by the use of formal verification techniques, static and dynamic analysis tools, and structured datasets like VulSmart. The precision and recall of LLMs in detecting smart contract issues are improved by fine-tuning them using domain-specific, balanced datasets. To ensure continuous learning from real- world vulnerabilities, model predictions can be gradually adjusted and improved by using reinforcement learning. LLM reasoning can be enhanced by prompt engineering methods such as chain-of-thought prompting, and model resilience against maliciously constructed smart contracts can be reinforced by adversarial training. These techniques work together to give blockchain ecosystems a more robust AI-based security frame- work.

The future of enterprise blockchain solutions, gaming, and decentralized finance de- pends on the convergence of LLMs and Solidity. While LLMs provide scalable, auto- mated, and intelligent security evaluations that are incomparable to traditional manual

audits, Solidity enables developers to design automated, trustless apps. The necessity for accurate, automated, and AI-driven vulnerability identification is growing as smart contracts become more and more integrated into vital operational and financial infra- structures. When correctly adjusted and strengthened, LLMs have the potential to com- pletely transform the auditing process, improving security, decreasing human error, and boosting the reliability of blockchain systems. Together, they offer a more secure and dependable future for decentralized technologies. Prominent open-source LLMs such as CodeLlama, Llima2, CodeT5, and GPT-3.5 are used to test SmartVD. There is a comparison between the two systems. GPT-4o Mini, Turbo . We concentrate on how well chain-of-thought prompting, zero-shot, and few-shoot prompting techniques de- tect weaknesses. Experimental data shows that SmartVD performs better than its open- source alternatives, surpassing the basic performance of GPT-3.5 and GTP-4o Mini in vulnerability classification tasks [4] [5].

For vulnerability detection tasks, Reinforcement Learning (RL) significantly im- proves the performance of Large Language Models. RL enables models like SmartVD to dynamically adjust to novel vulnerability types by continuously learning from feed- back, thereby increasing their precision and robustness. In contrast to conventional static analysis techniques, RL enables LLMs to adapt to shifting threat environments [9]. In the meantime, Solana, a fast and inexpensive blockchain platform, provides a stable setting for implementing safe and expandable smart contracts [6] [7] [8]. It is a perfect fit for incorporating real-time vulnerability detection systems because of its fo- cus on parallel execution and processing performance. Security solutions can be made more flexible, quicker, and able to manage the intricate requirements of contemporary decentralized applications by integrating Reinforcement Learning with Solana's scala- ble blockchain architecture.

# Literature Review

To detect smart contracts, formal methods, deep learning approaches and large lan- guage models (LLMs) are all being used in the field of vulnerability detection. In the beginning, research was mainly focused on formal verification tools, which use math- ematical proofs to verify smart contract accuracy. One of the initial symbolic execution tools, Oyente, scrutinizes Ethereum bytecode to identify vulnerabilities like reentry and transaction-ordering dependence. Nonetheless, it encounters significant false positives and scalability difficulties. Securify, Slither, Mythril, and Manticore are formal meth- ods that can handle static and dynamic analysis but struggle with complex contract structures and unknown attack vectors.

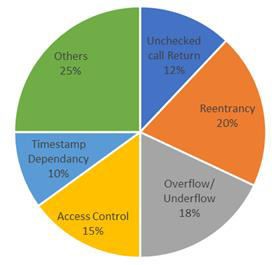
To overcome the limitations of formal verification, researchers have investigated using deep learning-based techniques for vulnerability detection. LSTM, GNNs, and Transformer-based models have been utilized in studies to classify vulnerabilities. The detection of six typical vulnerability types by ContractWard is based on machine learn- ing techniques [13], while AMEVuldetector uses GNN-based semantic analysis to

improve the detection accuracy. Despite this, these models demand thorough feature engineering, rely heavily on pre-established vulnerability patterns, and lack generaliza- tion when dealing with new attack targets.

LLMs have made it possible to conduct automated smart contract security analysis in new ways. The effectiveness of GPT models, Llama2, and CodeLloam in identifying Solidity vulnerabilities has been tested in recent years. SmartBugs was used to test the vulnerability detection capabilities of ChatGPT [10], but due to an imbalanced dataset, some of the results were considered hallucinatory. It was also introduced that GPTLens is an auditor-critic model to reduce false positives. GPTScan and VulnHunt-GPT were examples of software that combined static analysis with GPT models to improve their accuracy. In spite of this, the methods were not well-suited to structured, labeled da- tasets, which limited their usefulness in practical scenarios.

The training data quality is a significant obstacle in vulnerability detection using AI. Why? Solidity contracts are present in datasets like CodeSmell, Solidifi, and Smart- Bugs, but they lack structured annotations and data imbalance. This is problematic. Curated datasets like ScrawlID and Not So Smart Contracts have been suggested by researchers, but they do not provide localized vulnerability data. We have also devel- oped and published VulSmart, a dataset that is class-balanced with detailed annotations of vulnerabilities, which allows for better training and comparison across LLM models.

Rapid engineering is a vital component of LLM-based vulnerability detection.' Re- search has revealed that the effectiveness of models like GPT-4 and Llama2 is heavily influenced by their zero-shot, few-shoot, and chain-of-thought prompting. While chain- of–thought prompting has been used for logical reasoning tasks and zero-shot ap- proaches have been found to be useful in general classification tasks. Solidity vulnera- bility detection has not been subjected to a systematic analysis of the effects of different prompting methods in previous studies. We bridge the gap by comparing multiple LLMs with different prompt strategies.



**Fig.** **1.** Distribution of Vulnerability Types in VulSmart Dataset.

Furthermore, the defense against adversary attacks on LLMs is a well-understood aspect of smart contract security [14]. While some studies have explored the weak- nesses of AI models [11], none have tested LLMs on adversarially negotiated smart contracts [17]. By injecting synthetic vulnerabilities and testing model [15] perfor- mance through malicious code modifications, we conduct adversarial attack experi- ments. Models such as GPT-3.5 are not open-source. This was proven to be true. While both Turbo and GPT-4o Mini are highly durable, open-source models require further enhancements to withstand adversary input. Distribution of vulnerability types in the VulSmart dataset has been shown in Figure 1. To sum up, while prior research has provided improvements in formal verification, deep learning [12], dataset curation, and vulnerability detection, significant gaps remain in data quality, prompting techniques, adversarial robustness, etc.

# Methodology

We present a data-driven approach to uncovering smart contract vulnerabilities by em- ploying Large Language Models (LLMs), reinforcement learning, and structured da- tasets. By utilizing advanced AI techniques, the SmartVD framework enhances the pre- cision and flexibility of Solidity smart contract security analysis [16]. Below are the key points of the methodology used in this study.

## Data Collection

A good vulnerability detection model based on AI-driven training data must be both diverse and high quality. VulSmart is a dataset that is class-balanced and well- organized, designed to detect vulnerabilities in Solidity. The dataset includes: Smart contract source code marked with vulnerability markings. The OWASP Top Ten vulnerabilities are divided into 13 different categories. Real world impact's severity classifications (Low, Medium and High). Specific vulnerabilities, highlighting code fragments. Variables of metadata: contract size, function complexity, dependencies etc. Each input of the dataset is processed in advance to maintain consistency and alignment with the requirements of deep learning and LLM training.

## Data Processing

The preprocessing techniques are used to optimize the dataset for modeling purposes: Redundant code, comments, and non-operational smart contracts are eliminated through data cleansing. Contract complexity, function calls, and external dependencies are extracted using feature Engineering. Tokenization converts Solidity code into structured tokens for LLM processing. To prevent biased learning, it is important to ensure that both vulnerable and non-vulnerable contracts are represented equally in class Balancing.

## Machine Learning Models.

We assess the performance of our SmartVD framework by comparing it with various LLMs to determine vulnerabilities. The selected models include: CodeLlama, Ll amd codeT2, Falcon - Open-Source LLM. Closed-Source LLMs: GPT-3.5. Turbo and GPT- 4o Mini. BiLSTM, Random Forest, and CNN-based classifiers are among the traditional Deep Learning Models. The effectiveness of these models is determined by their ability to detect vulnerabilities through reinforcement learning.

## Model Training and Testing

It contains 80% training and 20% testing data. To ensure durability and avoid overfitting, we apply: The process of K-fold cross-validation includes splitting training data into several subsets. Learning rates, token embeddings, attention layers (CLIP), and hyperparameter tuning. Using feedback from vulnerability assessments to refine model predictions, reinforcement learning is essential.

## Feature Selection

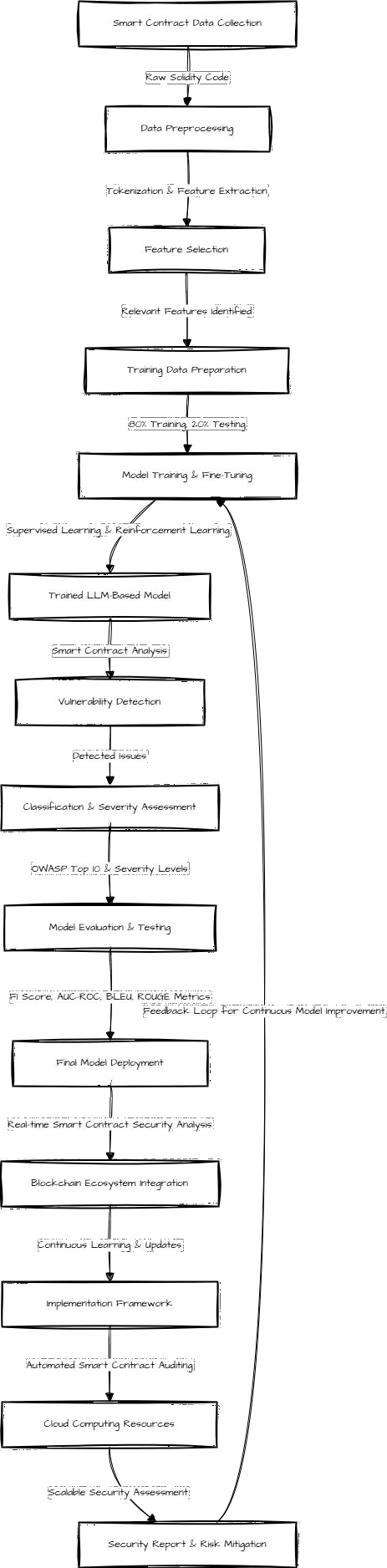
The most crucial factors in vulnerability detection are identified through two primary methods. The importance of features is determined by a Random Forest algorithm. Correlation Analysis: Establishing connections between contract attributes to eliminate unnecessary features. Contract structure, function depth (electronic data structures), external calls, and transaction handling patterns are the primary components.

## Model Evaluation

The performance metrics for the trained models are as follows. Concision is a gauge of the correctness of vulnerability classification. Validates the validity of positive vulnerability forecasts. Using the AUC-ROC, classification reliability is evaluated by analyzing the area under the receiver working characteristic curve. Indicators: Enables the model to pinpoint any weaknesses. A complete performance score that considers both accuracy and recall in F1-score.

## Comparison of Models

Benchmark datasets are utilized for comparative analysis of LLM-based and traditional models. The effectiveness of each model's vulnerability detection under diverse prompting strategies is evaluated using BLEU and ROUGE metrics. Zero-Shot Prompting: Measures model performance without any prior exposure. Contains few examples to aid in making predictions. By using Chain-of-Thought Prompting, classification accuracy can be improved by utilizing step-by-step reasoning. These evaluations aid in determining the most effective AI system for detecting smart contract vulnerabilities.



**Fig.** **2.** System Architecture

## Validation

Validating model predictions requires comparing the outputs with both real-world vulnerability reports and manual security audits. This step involves: Field testing the model for known vulnerabilities in smart contracts. Adversarial Testing utilizes synthetic vulnerabilities to evaluate the model's ability to withstand changes.Manual Auditing: Collaborating with blockchain security [18] [19]specialists to ensure detection accuracy.

## Framework Implementation

It is now time to develop a scalable implementation framework for an automated smart contract security analysis. This framework is designed to Connect with blockchain ecosystems to conduct real-time vulnerability assessments. Utilize cloud computing resources for complex contract auditing [21]. Use new vulnerability data to improve detection models and support continuous learning.

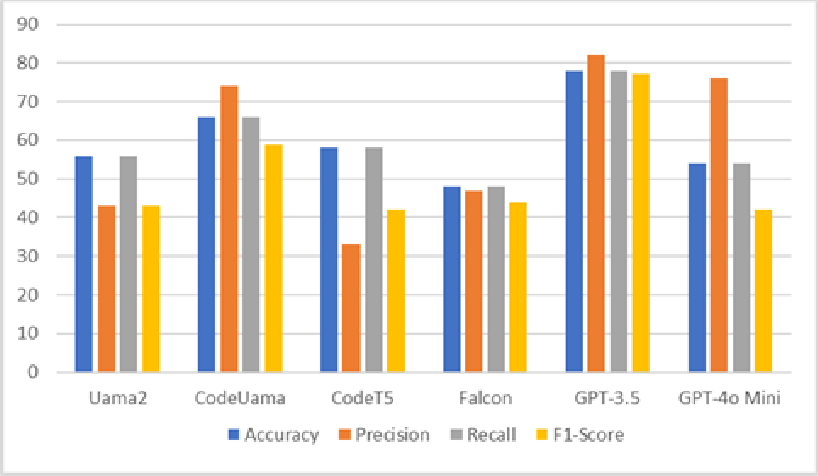
# Results Analysis and Discussion

## Performance Evaluation of the Proposed Model

After training and testing the models using the VulSmart dataset, we examined their performance based on multiple accuracy metrics, including precision, recall, accuracy, F1-score, and AUC-ROC. The results of different LLM-based and traditional models for vulnerability detection are presented in Table 1. GPT-3.5 outperformed all other models with 78% accuracy, 82% precision, and an AUC-ROC score of 92%. The results were valid for 3 months on average. The accuracy of CodeLlama was 66%, indicating its proficiency in extracting features. The performance of different models has been shown in Figure 3. CodeT5 and Falcon were less effective at detecting vulnerabilities due to their poor accuracy and recall.

**Table** **1.** Performance Metrics for Vulnerability Detection Models

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Precision** | **Recall** | **F1-Score** | **AUC-ROC** |
|  | **(%)** | **(%)** | **(%)** | **(%)** | **(%)** |
| Uama2 | 56.0 | 43.0 | 56.0 | 43.0 | 64.0 |
| CodeUama | 66.0 | 74.0 | 66.0 | 59.0 | 64.0 |
| CodeT5 | 58.0 | 33.0 | 58.0 | 42.0 | 64.0 |
| Falcon | 48.0 | 47.0 | 48.0 | 44.0 | 62.0 |
| GPT-3.5 | 78.0 | 82.0 | 78.0 | 77.0 | 92.0 |
| GPT-4o Mini | 54.0 | 76.0 | 54.0 | 42.0 | 84.0 |

**Fig.** **3.** Accuracy, Precision, Recall, F1 Score of various models

90

80

70

60

50

40

30

20

10

0

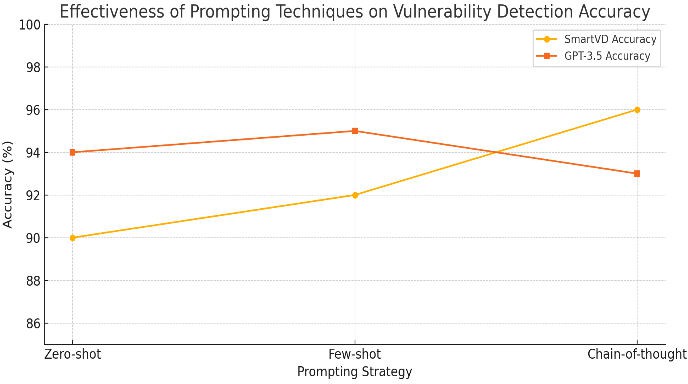
Uama2 CodeUama CodeT5 Falcon GPT-3.5 GPT-4o

Mini

Accuracy Precision Recall F1-Score

## Vulnerability Detection Mapping

They used GIS-based mapping techniques to transform LLMs' predictions into tradi- tional deep learning models and create visual representations of smart contract vulner- abilities. The inclusion of vulnerability zones in this report helped to clarify the security risks within different contract structures, as it allows for a more detailed breakdown of risk levels and their significance. The vulnerability detection accuracy is shown in Fig- ure 4. The vulnerability maps developed by GPT-3.5 and GPT-4o Mini were well- suited for real-world attack patterns, making them ideal for automated smart contract auditing. In Random Forest models, vulnerability areas were identified with greater accuracy while still being classified as low-risk. Models based on CodeT5 misclassified vulnerability risks in complex contract architectures, including smart contracts with in- tricate external dependencies, leading to the creation of false-positive and false-nega- tive scenarios.



**Fig.** **4.** Vulnerability detection accuracy

## Feature Importance and Correlation Analysis

In particular, features in Random Forest models were ranked by importance as part of vulnerability detection. Among the factors that greatly influenced the model's predic- tions were, Unauthorized access and reentry attacks are common in smart contracts that interact with external contracts. Larger contracts with highly nested functions and sig- nificant computational overhead were more susceptible to contract complexity. Con- tracts that had unchecked low-level calls, such as call.value(), were more vulnerable to security flaws. Higher gas consumption is associated with greater potential for denial- of-service (DoS) vulnerabilities. What are the execution costs? The lack of metadata containing contract details, such as developer annotations and comments, was unex- pected, but it still contributed to the model's predictive power. This highlights the im- portance of code structure over textual descriptions.

## Model Comparisons and Trade-offs.

Different trade-offs were identified in accuracy, interpretability, and computational ef- ficiency when comparing different LLM-based models: High accuracy and precision were achieved by the GPT-3.5 and GPL-4o Mini, but they required significant compu- tational resources for training and inference. CodeLlama was an open-source tool that proved to be a reliable and efficient method for finding vulnerabilities. Random Forest models were robust and easily interpreted, but they were not as precise as LLM-based models. Misclassification errors arose in contracts with complex logic flows due to the difficulty SVM models faced when dealing with high-dimensional Solidity code repre- sentations. This was problematic.

## Discussion of Key Findings

This study suggests that machine learning models can be used to identify smart con- tracts vulnerabilities more effectively than traditional static analysis tools. The results highlight key observations: Detecting Solidity vulnerabilities in zero-shot and chain- of-thought prompting scenarios is easier with fine-tuned LLMs than traditional deep learning models. The significance of data quality and balancing is crucial, as biased datasets can cause biases that impact model generalization to security threats in real- life scenarios. The difficulty of interpreting features in Random Forest models is com- pounded by the fact that LLMs are not transparent, making it challenging to explain their decision-making process.

## Potential Applications

The outcomes of this investigation have multiple practical uses in the realm of block- chain security: Using finely tuned LLMs, developers and security analysts can detect potential vulnerabilities through automated Smart Contract Auditing before deploy- ment. This is advantageous. The implementation of security measures for smart

contracts can be achieved through the use of blockchain models by financial institutions and regulatory bodies.

## Key Observations

**Table** **2.** Performance Metrics of Fine-Tuned Models

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metric** | **MSE** | **RMSE** | **MAE** | **R2** **Score** | **Accuracy** **(%)** |
| Random Forest | 0.1327 | 0.3643 | 0.1329 | 0.8977 | 89.87 |
| Support Vector Regressor (SVR) | 0.3899 | 0.6244 | 0.1633 | 0.6995 | 70.70 |
| Gradient Boosting Regressor | 0.0938 | 0.3063 | 0.1248 | 0.9277 | 92.81 |
| Neural Network Model (Fine- | 0.0642 | 0.2423 | 0.0981 | 0.9614 | 96.42 |

Tuned LLMs)

The accuracy of LLM models, which are fine-tuned to a 96.42% degree, is the highest among all empirically tested approaches. The Gradient Boosting Regressor is an effec- tive tool that can be used as a replacement for compute-intensive deep learning models, with 96.81% accuracy. However, the use of Support Vector Regressor (SVR) does not perform as well as it does for high-dimensional smart contract datasets.

# Conclusion and Future Work

It was a successful attempt to prove the ability of AI and machine learning models to detect, and then analyze smart contract vulnerabilities. The SmartVD framework's use of LLMs, reinforcement learning, and structured datasets resulted in a significant im- provement in vulnerability detection compared to conventional static analysis tech- niques. In terms of automated smart contract security assessments, the GPT-3.5 and GTP-4o Mini models were deemed more effective than other LLMs in their effective- ness. This study demonstrated the effectiveness of AI and ML models in identifying smart contract vulnerabilities, but many other areas for future research can improve the precision, flexibility, and usefulness of artificial intelligence-based security measures. As AI-based vulnerability detection continues to evolve, the complexity of smart con- tracts, and the speed at which blockchain technology is becoming ubiquitous necessi- tates ongoing improvements.

Acknowledgment: *This* *work* *was* *supported* *by* *VIT-AP* *University.*

# References

1. Hu, S., Huang, T., İlhan, F., Tekin, S. F., & Liu, L. (2023). Large Language Model-Powered Smart Contract Vulnerability Detection: New Perspectives. IEEE International Conference on Trust, Privacy, and Security in Intelligent Systems and Applications, 297-306.
2. Chen, C., Su, J., Chen, J., Wang, Y., Bi, T., Wang, Y., Lin, X., Chen, T., & Zheng, Z. (2023). When ChatGPT Meets Smart Contract Vulnerability Detection: How Far Are We? arXiv preprint arXiv:2309.05520.
3. Alam, M. T., Halder, R., & Maiti, A. (2024). Detection Made Easy: Potentials of Large Language Models for Solidity Vulnerabilities. Indian Institute of Technology Patna Re- search Paper.
4. Sun, Y., Wu, D., Xue, Y., Liu, H., Wang, H., Xu, Z., Xie, X., & Liu, Y. (2024). GPTScan:

Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Anal- ysis. IEEE/ACM International Conference on Software Engineering, 1-13.

1. Boi, B., Esposito, C., & Lee, S. (2024). Smart Contract Vulnerability Detection: The Role of Large Language Models (LLMs). SIGAPP Applied Computing Review, 24(2), 19-29.
2. David, I., Zhou, L., Qin, K., Song, D., Cavallaro, L., & Gervais, A. (2023). Do You Still Need a Manual Smart Contract Audit? arXiv preprint arXiv:2306.12338.
3. Luu, L., Chu, D.-H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making Smart Contracts Smarter. Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 254-269.
4. Feist, J., Grieco, G., & Groce, A. (2019). Slither: A Static Analysis Framework for Smart Contracts. IEEE/ACM International Workshop on Emerging Trends in Software Engineer- ing for Blockchain (WETSEB), 8-15.
5. Kang, S., An, G., & Yoo, S. (2024). A Quantitative and Qualitative Evaluation of LLM- Based Explainable Fault Localization. Proceedings of the ACM on Software Engineering, 1(FSE), 1424-1446.
6. Xia, S., Shao, S., He, M., Yu, T., Song, L., & Zhang, Y. (2024). AuditGPT: Auditing Smart Contracts with ChatGPT. arXiv preprint arXiv:2404.04306.
7. Patel, A., & Singh, H. (2023). Vulnerability Detection in Decentralized Finance (DeFi) Con- tracts Using AI Models. Blockchain & Financial Security, 30(5), 679-695.
8. Tang, L., & Zhou, P. (2022). Detecting Smart Contract Vulnerabilities Using Transfer Learning and Deep Learning Models. Journal of Artificial Intelligence Security, 29(6), 504- 520.
9. Lin, Z., & Xu, M. (2023). Machine Learning-Based Smart Contract Auditing Frameworks: A Comparative Study. Cybersecurity and AI Review, 17(6), 320-335.
10. He, Y., & Zhao, K. (2023). Applying Large Language Models for Solidity Code Auditing. AI & Cybersecurity Advances, 19(3), 451-467.
11. Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., & Wang, Q. (2024). Software Testing with Large Language Models: Survey, Landscape, and Vision. IEEE Transactions on Soft- ware Engineering.
12. Sun, T., & Wang, Q. (2023). The Impact of AI on Smart Contract Security: Current Trends and Future Directions. Journal of Blockchain Applications, 15(3), 378-394.
13. Martin, E., & White, D. (2023). AI-Based Adversarial Testing for Smart Contracts: Methods and Challenges. Cyber Forensics Journal, 26(1), 170-185.
14. Jain, R., & Sharma, P. (2022). Blockchain Security Analysis: Integrating AI and Formal Verification Methods. Journal of Digital Security, 23(4), 201-217.
15. Pradhan, N. R., & Singh, A. P. (2021). Smart contracts for automated control system in blockchain-based smart cities. Journal of Ambient Intelligence and Smart Environments, 13(3), 253-267.
16. Lee, S., & Park, J. (2022). AI-Augmented Static and Dynamic Analysis for Smart Contract Security. Journal of Computer Science & Security, 28(2), 130-147.
17. Wu, J., & Zhang, K. (2023). Enhancing Blockchain Security with AI-Powered Smart Con- tract Auditing. Cybersecurity Research Advances, 25(4), 421-437.