DOI: https://doi.org/10.61762/pjstvol9iss1art001

PLATFORM - A Journal of Science & Technology

Received: 28 July 2025, Revised: 2 January 2026, Accepted: 28 April 2026, Published: 30 June 2026, Publisher: UTP Press, Creative Commons: CC BY 4.0

GRAPH NEURAL NETWORKS-BASED DETECTION OF ACCESS CONTROL AND EXTERNAL CALL VULNERABILITIES IN SMART CONTRACTS

Nur Sarah Mohamad Suhaimi, Norshakirah Aziz

Faculty of Science, Management and Computing, Universiti Teknologi PETRONAS, Perak, Malaysia

*E-mail: nur_19000422@utp.edu.my

ABSTRACT

The growing adoption of smart contracts, the self-executing digital agreements stored on blockchain networks, has resulted from the increasing adoption of blockchain technology. While these contracts offer automation and transparency, they also introduce significant security risks, with vulnerabilities often resulting in substantial financial losses. Existing tools, such as Slither, are widely used for vulnerability detection. However, their reliance on expert-defined rules and static analysis patterns limits their effectiveness, resulting in yielding false positives and failing to identify complex or evolving attack patterns. This research proposes an automated vulnerability detection model tailored for Solidity smart contracts, leveraging Graph Neural Networks (GNNs) with a specific focus on the Graph Attention Network (GAT). The model constructs a unified graph structure by integrating Abstract Syntax Tree and Control Flow Graph, preserving semantic edge types and incorporating a custom edge-aware attention mechanism to capture rich structural and relational information. The model is trained on a labelled dataset of Solidity contracts to detect two critical vulnerabilities: access control and unchecked external calls. Experimental evaluation demonstrates that the proposed GATv2EdgeAwareNet achieves an accuracy of 78.98%, along with strong performance in precision (79.14%), F1 score (78.99%) and recall (78.98%). Compared to Slither, it delivers higher accuracy, precision and F1 score, with only a marginal trade-off in recall. Slither, despite achieving the highest recall (92.4%), exhibits inconsistent precision and elevated false positive rates. These findings show the potential of graph-based deep learning approaches as scalable, adaptive and accurate solutions for detecting vulnerabilities in smart contracts amidst evolving blockchain security threats.

Keywords: *Blockchain, solidity, security risks, graph attention network, unified graph object, edge-aware*

INTRODUCTION

Blockchain technology has rapidly gained traction across industries due to its decentralisation, transparency and security features. A key innovation within this domain is the smart contract, a self-executing program that enforces predefined terms. Despite their benefits, smart contracts are vulnerable to security flaws that may lead to severe financial and reputational consequences. Notably, the decentralised autonomous organisation (DAO) attack exploited a reentrancy issue to extract a large amount of Ether, emphasising the importance of pre-deployment contract analysis [1]-[2]. Other recurring vulnerabilities include access control flaws, unchecked external calls and integer overflows. These issues underscore the urgent need for robust, adaptive vulnerability detection mechanisms.

Existing detection methods rely heavily on expert-defined rules and static analysis tools such as Slither and Mythril [2]-[3]. While effective to a degree, these tools are limited by their static nature and susceptibility to evasion by adversaries aware of their detection patterns. Moreover, the evolving landscape of smart contract threats renders static techniques insufficient. An adaptive approach capable of learning complex and emerging patterns, such as a Graph Neural Network (GNN)-based approach, is required to improve detection accuracy.

This study seeks to navigate the constraints of traditional methods by pursuing three core objectives: to analyse the limitations of popular static tools, such as Slither,

in identifying smart contract vulnerabilities; to design and implement a Graph Attention Network (GAT) for vulnerability detection in Solidity smart contracts; and to evaluate the proposed model's performance in terms of accuracy, precision, and recall. Given the broad spectrum of smart contract vulnerabilities, this study focuses on two critical yet underexplored types, namely access control and unchecked external calls. While reentrancy has been widely studied and ranked highest in 2023, its importance has since declined, now ranking fifth in the OWASP Top 10 for 2025, making these two vulnerability types more pertinent for further exploration. Accordingly, this research investigates the feasibility of using a GNN-based approach to detect these specific vulnerabilities, with an emphasis on both effectiveness and practical applicability.

LITERATURE REVIEW

Blockchain and Smart Contracts

Blockchain, introduced by Satoshi Nakamoto in 2008 through Bitcoin, enables decentralised consensus across a distributed network without a central authority [1]-[5]. It operates as an append-only, tamper-resistant ledger, ensuring transparency and immutability [1]. Due to these properties, blockchain has been widely adopted across sectors to improve security, privacy, and data integrity [6]. One of its most significant applications is the smart contract, a self-executing code that automates the enforcement of agreements under predefined conditions [3],[6]. Smart contracts facilitate secure transactions among untrusted parties, reduce legal overhead, and improve scalability [3],[5]-[6]. However, their immutability, whereby deployed code cannot be modified, presents a significant security challenge. Bugs cannot be patched post-deployment, making secure development and pre-deployment analysis crucial [6]. Smart contracts are frequent targets for attackers due to their direct handling of financial assets, immutability, and lack of standardised evaluation metrics [3]. These challenges emphasise the need for effective vulnerability detection methods during development.

Smart Contract Vulnerabilities and Security Analysis

Smart contracts are prone to coding flaws, often with severe financial consequences [3],[5]. The DAO attack, in which a reentrancy vulnerability enabled the unauthorised withdrawal of approximately 3.6 million Ether, marked a critical turning point in blockchain security history [1]-[2],[7]. Vulnerabilities typically fall into four categories: security, functional, operational, and developmental, with access control flaws and unchecked external calls among the most exploited [3]. Access control vulnerabilities arise when unauthorised entities can invoke privileged functions. For instance, the Parity Wallet bug allowed attackers to reset ownership and destroy the contract, freezing over $280 million in Ether [8]. Unchecked external calls occur when return values from functions like call() or send() are not validated. Such unchecked behaviour may result in silent failures and unexpected program states [9].

To address these risks, several security analysis tools have emerged. Static analysis tools like Slither, Mythril, and Oyente analyse contract code without execution to detect known vulnerability patterns. Slither, in particular, translates Solidity into SlithIR—an intermediate representation that preserves high-level semantics for more effective detection and code review [5],[10]-[11].

More advanced techniques include formal verification, which mathematically proves contract correctness; symbolic execution, which explores all logical execution paths; and fuzz testing, which injects random inputs to uncover crash-prone behaviours [7]. Tools like Slither also incorporate intermediate representation (IR) analysis to enhance semantic extraction. Taint analysis tracks how untrusted input flows through a program, identifying insecure data propagation [7]. Additionally, deep learning methods are gaining traction by training on labelled contract data to automatically identify complex vulnerability patterns.

Despite these tools, significant limitations persist. Static analysis can be exhaustive and prone to false results, especially in large or complex codebases. Dynamic methods, though effective at detecting runtime behaviour, may overlook rare or sequence-specific vulnerabilities [7]. Rule-based systems struggle with unseen attack vectors and require manual pattern engineering [2]. Combining tools may increase detection, but it also adds computational overhead.

Table 1 Limitation of existing smart contract vulnerability detection approaches

Tool / Approach	Strengths	Limitations
Slither	Fast and uses SlithIR (high-level intermediate representation)	Relies on predefined rules; limited generalisation to novel vulnerabilities; prone to false positives
Mythril	Symbolic execution enables exploration of potential unsafe execution paths	High computational cost; limited scalability
Oyente	Early vulnerability detection via symbolic execution	Struggles with complex contract logic; incomplete path exploration
Fuzz testing	Effective at runtime bug discovery	Misses rare or sequence-dependent vulnerabilities
Formal verification	Provides mathematically provable correctness guarantees	High complexity; time-consuming; requires specialized expertise
Taint analysis	Track tainted data propagation paths efficiently	Prone to false positives or negatives; sensitive to complex sanitization logic

Through the analysis of popular static tools and approaches summarised in Table 1, this study identified several recurring limitations, including reliance on predefined detection rules, limited generalisation to novel vulnerabilities, vulnerability to false positives and false negatives and scalability challenges in complex contracts. Due to these constraints, researchers have turned to GNNs, which analyse smart contract structures as graphs. GNNs offer adaptive learning of structural and semantic patterns, making them promising for accurate and scalable vulnerability detection beyond static rule sets [7].

GNNs for Vulnerability Analysis

GNNs are a class of deep learning architectures tailored to handle graph-structured inputs by representing information as nodes and edges. They operate via iterative message-passing mechanisms, making them highly effective for tasks such as link prediction and node classification [12]. GNNs are typically categorised into spectral methods, which adapt convolutional operations to graph signals, and spatial methods, which aggregate neighbouring node features using message passing [2].

Recent advances have extended GNN applications to code analysis, where graphs such as Abstract Syntax Trees (ASTs) and Control Flow Graphs (CFGs) are used to model a program's behaviour [13]. In smart contracts, GNNs enable vulnerability detection by encoding both syntactic and semantic information within code graphs [11].

Among GNN variants, Graph Convolutional Networks (GCNs) were among the earliest models that aggregated neighbourhood information with uniform weights [13]. However, this limitation led to the development of GATs, which incorporate attention layers that weigh neighbouring node contributions differently during feature learning [13]. GATv2 further improved this by introducing dynamic, content-aware attention coefficients for better expressiveness in complex domains such as smart contract analysis [14]. These models are particularly effective for detecting vulnerabilities that depend on subtle control-flow or state-dependent logic, such as access control violations and unchecked external calls.

Related Works

Several works have explored the application of GNNs to identify security flaws in smart contracts. Zhuang et al. [2] introduced a framework combining syntactic and semantic graphs using a Degree-Free GCN with Temporal Message Propagation, achieving high accuracy in detecting issues such as reentrancy attacks, timestamp manipulation, and infinite execution loops. Ma et al. [13] introduced a Hierarchical GAT (HGAT) that compresses AST and CFG structures and applies hierarchical attention, outperforming traditional tools and deep learning baselines across multiple vulnerability types.

Zhen et al. [11] developed a Dual-Attention GNN (DA-GNN) to model control-flow semantics using opcode classification and dual attention layers. Their model showed strong performance in detecting integer

overflows, self-destruct risks, and transaction order dependencies. Similarly, Chen and Chen [15] proposed Edge-Featured GAT (EGAT), integrating edge attributes directly into the attention mechanism, which improved tasks with high-dimensional or continuous edge features. However, EGAT's general-purpose design is less effective in smart contract analysis, where edge types are discrete and semantically specific.

In response, this study proposes an edge-aware GATv2 architecture tailored for smart contracts, incorporating learnable edge-type embeddings to represent semantic relationships and control flow. This design maintains architectural simplicity while improving relevance and interpretability.

Beyond the blockchain domain, attention-based GNNs have proven effective across a range of structured and temporal tasks. Wang et al. [12] proposed the Trend GAT Network (TGAN) for traffic forecasting by modelling spatial heterogeneity and temporal trends, while Zhang et al. [16] introduced a Spatial-Temporal GAT (ST-GAT) that integrates GAT with LSTM to capture dynamic dependencies in time-series data. These works demonstrate the flexibility and scalability of attention-based GNNs across domains.

Despite their success, most GNN-based models for smart contract analysis primarily focus on legacy vulnerabilities, such as reentrancy and integer overflow. However, the OWASP Top 10 (2025) highlights emerging and increasingly prevalent risks, including access control issues, unchecked external calls, oracle manipulation, and input validation flaws. Existing approaches rarely address these newer vulnerability classes, revealing a significant research gap in developing more comprehensive and adaptive smart contract security analysis models.

To address this, the study introduces an edge-aware GATv2 model designed to detect access-control and unchecked external-call vulnerabilities. Unlike prior approaches relying on handcrafted features, this model leverages GAT's ability to learn structural and semantic patterns directly from graph-encoded smart contracts. By integrating edge-type embeddings into the attention mechanism, the model aims to deliver a scalable, accurate, and adaptable solution for modern smart contract security analysis.

METHODOLOGY

This study applies the CRISP-DM methodology, an industry-recognised standard for organising data mining projects, to structure the design of an automated vulnerability detection model for Solidity smart contracts. The CRISP-DM process, as illustrated in Figure 1, consists of six stages. These steps are adapted to the specific requirements of smart contract analysis, where raw source code must first be transformed into a graph-structured representation before training a GNN model.

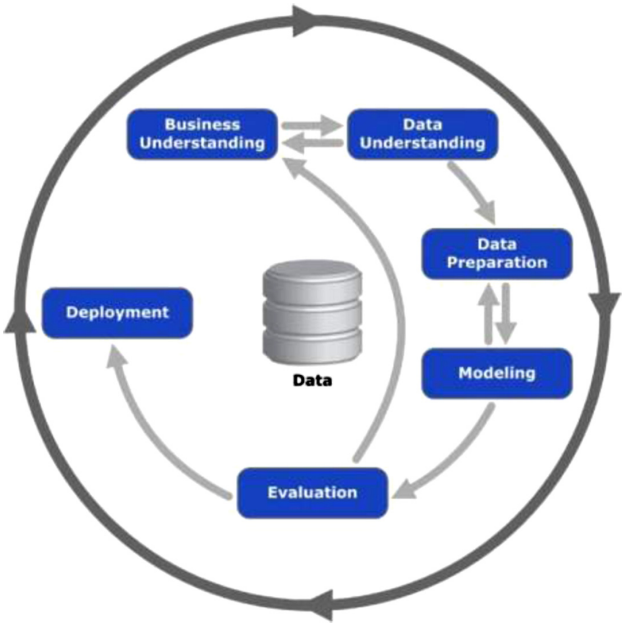

Figure 1 CRISP-DM process diagram

Business Understanding

The objective of this study is to enhance smart contract security by improving the detection of two critical vulnerability types: access control violations and unchecked external calls. Traditional static analysis tools such as Slither rely on rule-based heuristics that often fail to generalise to evolving contract structures or novel attack patterns. This work aims to address those limitations by leveraging deep learning to automatically learn meaningful patterns from graph-structured code representations.

Data Understanding and Collection

The dataset comprises manually labelled Solidity smart contracts categorised as either vulnerable to access-

control issues or to unchecked external calls. Contracts were sourced from:

1. SmartBugs Curated [17], which includes 143 annotated contracts across several vulnerability types,
2. SWC Registry, a publicly maintained reference of common smart contract weaknesses, and
3. ScrawlD [18], a large-scale dataset of 6,780 Ethereum contracts labelled using majority voting from five static analysers.

An initial set of 665 contracts was collected: 332 labelled as access-control vulnerabilities and 333 as unchecked external-call vulnerabilities. Contracts that were syntactically invalid or lacked analysable structure were excluded. After validation, 310 access-control and 277 unchecked external-call contracts remained for training and evaluation.

Data Preparation

Each contract was transformed into a structured graph representation called the Unified Graph Object (UGO), which integrates both syntactic and semantic properties of the code.

AST and CFG Extraction

Slither was used to extract two intermediate representations:

1. The AST capturing code structure (e.g., functions, parameters, variable use).
2. The CFG models execution paths and runtime behaviour.

UGO Construction

The AST and CFG were merged into a single directed multigraph. Node types included:

1. Function nodes (e.g., transfer)
2. Parameter nodes (e.g., amount)
3. Variable nodes (e.g., state variables)
4. CFG nodes (e.g., control expressions)

Edges between nodes captured semantic relationships:

1. "defines": links functions to parameters
2. "uses" / "modifies": indicate variable access or assignment
3. "executes": links functions to CFG entry points
4. "cfg_edge": represents control flow transitions

Only contracts with successfully parsed AST and CFG were retained.

Feature Encoding

Each graph was converted into a format compatible with PyTorch Geometric. Feature encoding included:

1. Node types: one-hot encoded (function, param, variable, cfg_node)
2. Edge types: integer-encoded and passed through a learnable embedding layer
3. Optional node features: content length (e.g., expression length)
4. Global features: number of nodes, edges, average degree, and node type counts

The final graph objects were saved with structured fields (e.g., x, edge_index, edge_attr, global_features, y) for use during model training. The UGO workflow is depicted in Figure 2.

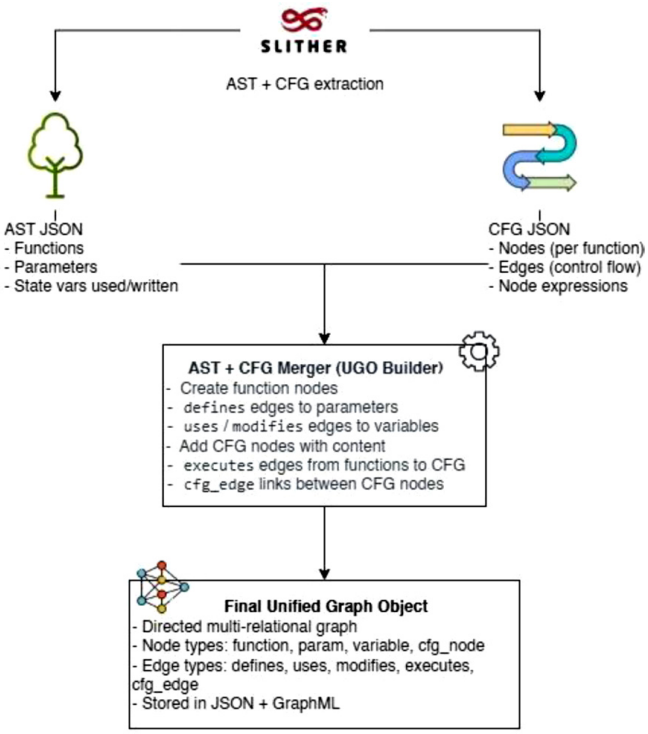

Figure 2 Unified graph construction workflow

Modelling

The proposed architecture, named GATv2EdgeAwareNet, extends the GATv2 model by incorporating edge-type semantics into the attention mechanism. The model flow is illustrated in Figure 3.

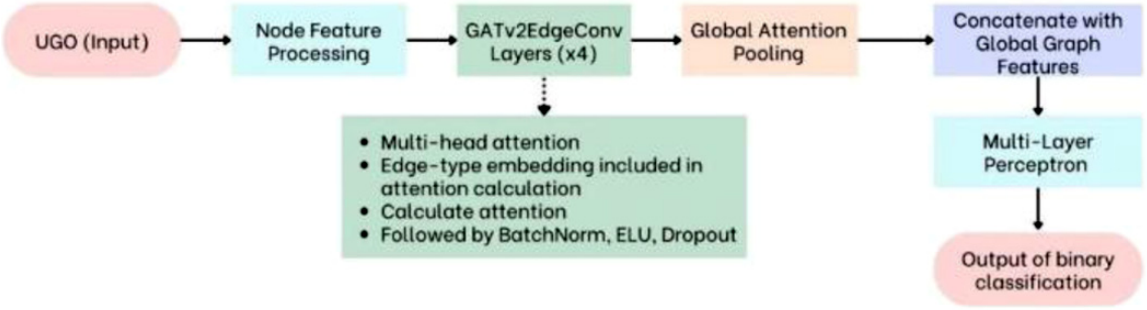

Figure 3 GNN model flow

GATv2EdgeConv Layers

In standard GATv2, attention coefficients for edges (i, j) are computed as:

$$a_{ij} = LeakyReLU(a^T [Wh_i \,||\, Wh_j])$$

where:

h_i and h_j = node features,

W = learnable linear transformation matrix,

a = trainable attention vector, and

$||$ = concatenation.

This model is extended with edge-aware attention, adding a learnable edge-type embedding e_{ij}, resulting in:

$$a_{ij} = LeakyReLU(a^T [Wh_i \,||\, Wh_j \,||\, e_{ij}])$$

This formulation allows the model to differentiate not only node importance but also the nature of their relationships, which is crucial for semantic-rich graphs like smart contracts. The model includes four stacked GATv2EdgeConv layers with Layer 1 (8 attention heads), Layer 2 (4 heads), Layer 3 (2 heads) and Layer 4 (1 head). Each layer is followed by batch normalisation, ELU activation, and dropout for regularisation.

Graph-Level Embedding and Classification

A graph-level embedding is derived by aggregating node features through a Global Attention Pooling mechanism. The layer is concatenated with global structural features and passed through a Multi-Layer Perceptron (MLP) for final binary classification (access control vs. unchecked external call vulnerability).

Evaluation

The effectiveness of the proposed model was assessed using commonly used classification criteria, including accuracy, precision, recall, F1 score, and balanced accuracy. These evaluation indicators offer a well-rounded understanding of how the model performs, particularly in the presence of class imbalance. A detailed analysis is presented in Section 4.

Deployment

Although deployment is not the focus of this study, the trained model can be serialised for use in practical auditing tools. It supports batch inference and could be integrated into development environments or security platforms as a plugin or service for automated smart contract vulnerability scanning.

RESULTS AND DISCUSSION

Experimental Configuration

All experiments were executed on a performance-oriented workstation featuring an NVIDIA GeForce RTX 3050 GPU (6GB VRAM), a 13th Gen Intel Core i7-13650HX CPU (2.60 GHz), and 64-bit Windows 11 OS. The model was implemented using the PyTorch Geometric framework due to its efficiency and flexibility in building message-passing GNN architectures.

Training was carried out over 50 epochs, utilising the AdamW optimisation algorithm with a learning rate of 0.001 and a weight decay factor of 5e-4. A batch size of 32 was used. To address class imbalance, a weighted cross-entropy loss function was employed, with weights dynamically computed from class frequency distributions to avoid bias toward the majority class. To obtain a robust estimate of model performance, a stratified 5-fold cross-validation strategy was employed alongside a fixed test set. 20% of the dataset was held out as a test set to simulate evaluation on unseen real-world contracts. The remaining 80% was used for 5-fold cross-validation using StratifiedKFold to ensure consistent label distribution across folds. The overall data split strategy is illustrated in Figure 4.

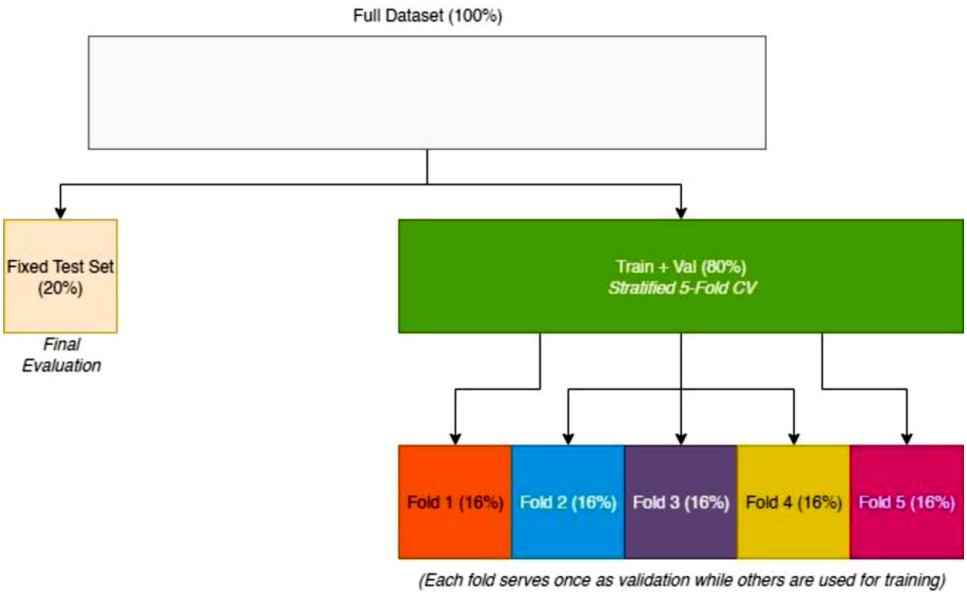

Figure 4 Data split strategy using 20% fixed test set and 5-fold stratified cross-validation

Assessment Criteria

To assess the effectiveness of the proposed model, several classification measures were applied: accuracy, precision, recall, F1 score, and balanced accuracy. These indicators provide a well-rounded view of the model's ability to correctly identify contracts that are vulnerable to unchecked external calls (positive class) and access control violations (negative class). Balanced accuracy was particularly important given the class imbalance, ensuring a fair performance evaluation across both classes.

For each fold, the model was re-initialised and trained from scratch. The best epoch was selected based on validation balanced accuracy, and each trained model was evaluated on both its validation split and the fixed test set. Performance metrics were reported as the mean and standard deviation across the five folds. To avoid information leakage, class weights were recalculated using only the training portion within each fold.

Results and Analysis

The performance of the proposed GATv2EdgeAwareNet was evaluated against Slither (Table 2), a widely used rule-based static analysis tool. Both approaches were tested on the same dataset, targeting two vulnerability types: unchecked external calls and access control violations.

Slither's high recall (92.4%) is attributed to its tendency to overpredict the "unchecked" class, resulting in a significant number of false positives—216 misclassifications for contracts labelled as "access." This leads to lower precision (54.2%) and an overall F1 score of 68.4%. Its balanced accuracy of 61.4% further highlights inconsistent performance across classes. In contrast, GATv2EdgeAwareNet achieved consistently high scores across all metrics. Its accuracy (78.98%), precision (79.14%), recall (78.98%), and F1 score (78.99%) were closely aligned, indicating balanced classification performance. The balanced accuracy of 79.05% confirms the model's robustness across both vulnerability classes. These results, averaged over five stratified folds with standard deviation under 1.5%, show low variance and high stability, demonstrating strong reliability for real-world auditing scenarios where both false positives and false negatives carry significant risk.

Table 2 Performance comparison of GATv2EdgeAwareNet and Slither

Metric	GATv2EdgeAwareNet	Slither
Accuracy	78.98%	59.6%
Precision	79.14%	54.2%
Recall	78.98%	92.4%
F1 Score	78.99%	68.4%
Balanced Accuracy	79.05%	61.4%

Analysis of Model Development

Training behaviour was stable, with rapid convergence observed in early epochs. As shown in Figure 5, both training and validation losses dropped sharply and stabilised around epoch 15-20. The narrow and consistent gap between the two curves indicates a stable generalisation rather than overfitting. The shaded regions represent the standard deviation across folds; they narrow after the early epochs and remain relatively tight through training, indicating consistent performance across different data splits. This pattern indicates that the model learns effectively and is reliable across various data subsets.

Figure 6 depicts the average validation accuracy and loss across all folds during training. Validation accuracy rises quickly in the early epochs and plateaus by roughly epochs 15-20, fluctuating around ~0.76 – 0.80 thereafter, while the dashed validation-loss curve declines steadily with small bumps. The shaded region around the accuracy curve represents the standard deviation across folds, which remains moderate-to-wide across epochs, indicating non-trivial variability between folds. These results reinforce early convergence and a stable average trend.

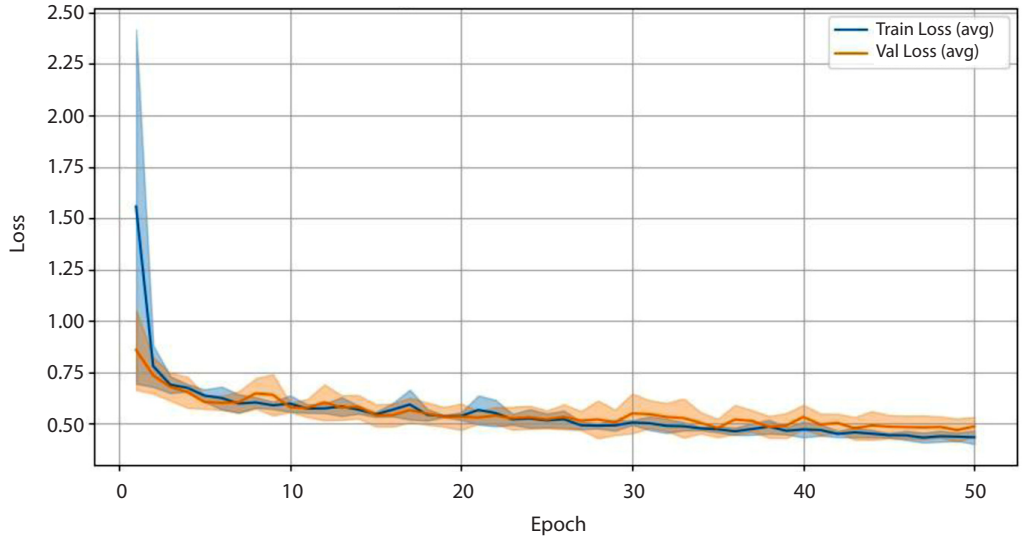

Figure 5 Visualisation of average model loss curves for training and validation over 50 epochs

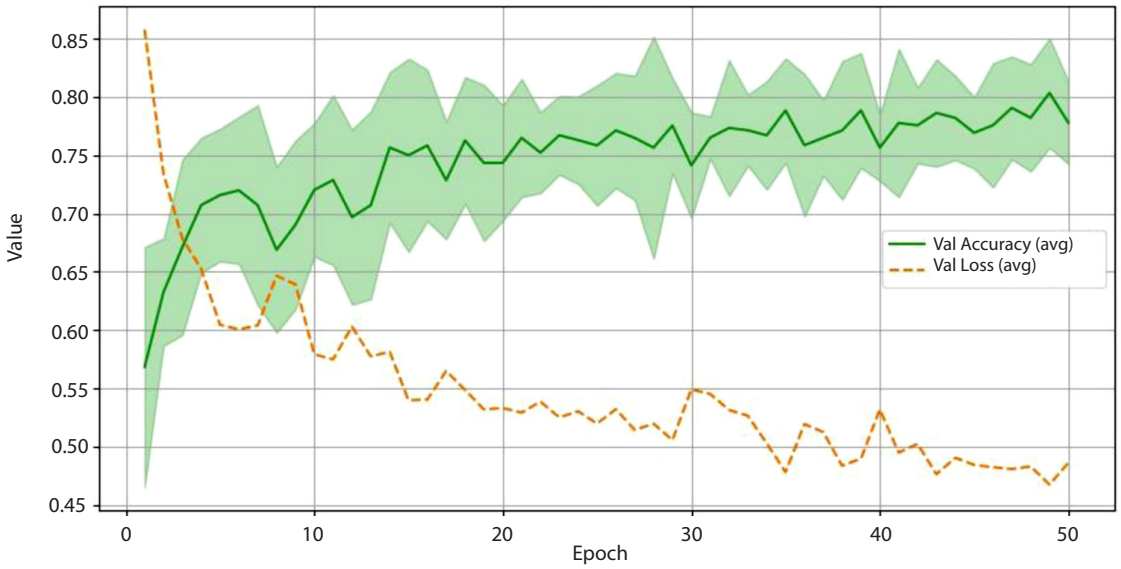

Figure 6 Average validation loss and accuracy trends

In summary, GATv2EdgeAwareNet outperforms a traditional static tool in both accuracy and consistency, offering a more scalable and intelligent solution for smart contract vulnerability detection.

CONCLUSION

This study addressed the critical challenge of smart contract vulnerability detection by proposing GATv2EdgeAwareNet, a Graph Neural Networks (GNN) model that leverages both syntactic and semantic information through a unified graph representation, incorporating a novel edge-aware attention mechanism to enable more accurate detection of access control violations and unchecked external calls. Experimental results demonstrated that GATv2EdgeAwareNet achieved strong performance across all key evaluation criteria, including accuracy (78.98%), precision (79.14%), recall (78.98%), F1 score (78.99%), and balanced accuracy (79.04%), outperforming traditional rule-based tools such as Slither in accuracy, precision, F1 score, and balanced accuracy, while maintaining competitive recall. The model generalised well to unseen contracts, required no manual rule engineering, and proved practical for scalable, automated vulnerability detection. Looking ahead, future work may explore extending the binary classification framework to multi-label settings to detect a broader range of vulnerabilities, while integrating larger and more diverse datasets or experimenting with transformer-based and ensemble models could further enhance robustness and adaptability. Improving interpretability through attention visualisation or feature attribution will also be valuable for real-world adoption. Ultimately, this study highlights the promise of graph-based deep learning as a foundation for next-generation smart contract auditing tools, offering both technical robustness and practical applicability, while underscoring the importance of aligning detection models with emerging threats and real-time blockchain developments to maintain relevance in this rapidly evolving domain.

ACKNOWLEDGEMENT

The author expresses sincere gratitude to Dr. Norshakirah Binti Ab. Aziz for her invaluable guidance, continuous encouragement and insightful feedback throughout this study. Special thanks are also extended to Universiti Teknologi PETRONAS and the internal examiners for their constructive critiques and recommendations during the proposal defense and viva, which significantly improved the quality of this manuscript. The author also sincerely appreciates the reviewers for their valuable comments, which helped improve the clarity and overall quality of the manuscript.

CONFLICT OF INTEREST

The author declares that there is no conflict of interest regarding the publication of this article.

AUTHORS' CONTRIBUTION

NSMS and NA conceptualised the study. NSMS developed the methodology, designed the smart contract evaluation framework, collected and analysed the data, drafted the original manuscript, and revised the text based on the reviewers' comments. NA supervised the project, provided guidance on applying graph neural networks to smart contract vulnerabilities, and critically reviewed the manuscript for intellectual content. Both authors read and approved the final version for publication.

REFERENCES

[1] J. Kolb, M. AbdelBaky, R. H. Katz, and D. E. Culler, "Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial", *ACM Comput. Surv*, vol. 53, no. 1, pp. 1-39, 2020, doi: 10.1145/3366370.

[2] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network," *Proc. 29th Int. Joint Conf. Artif. Intell. (IJCAI-20)*, pp. 3283–3290, 2020, doi: 10.24963/ijcai.2020/454.

[3] K. L. Narayana and K. Sathiyamurthy, "Automation and smart materials in detecting smart contracts vulnerabilities in Blockchain using deep learning," *Mater. Today: Proc.*, vol. 81, pp. 653–659, 2023, doi: 10.1016/j.matpr.2021.04.125.

[4] S. Gupta, S. Sinha, and B. Bhushan, "Emergence of blockchain technology: Fundamentals, working and its various implementations," in *Proc. Int. Conf. Innovative Comput. Commun. (ICICC 2020)*, 2020, doi: 10.2139/ssrn.3569577.

[5] A. Vacca, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges," *J. Syst. Softw.*, vol. 174, p. 110891, 2021, doi: 10.1016/j.jss.2020.110891.

[6] B. Cao, Z. Wang, L. Zhang, D. Feng, M. Peng, L. Zhang, and Z. Han, "Blockchain systems, technologies, and applications: A methodology perspective," *IEEE Commun. Surv. Tutor.*, vol. 25, no. 1, pp. 353–385, 2023, doi: 10.1109/COMST.2022.3204702.

[7] D. He, R. Wu, X. Li, S. Chan, and M. Guizani, "Detection of vulnerabilities of blockchain smart contracts," *IEEE Internet Things J.*, vol. 10, no. 14, pp. 12178–12185, 2023, doi: 10.1109/JIOT.2023.3241544.

[8] A. Ghaleb, J. Rubin, and K. Pattabiraman, "AChecker: Statically detecting smart contract access control vulnerabilities," in *Proc. 2023 IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, pp. 945–956, 2023, doi: 10.1109/ICSE48619.2023.00087.

[9] H. Rameder, M. di Angelo, and G. Salzer, "Review of automated vulnerability analysis of smart contracts on Ethereum," *Front. Blockchain*, vol. 5, p. 814977, 2022, doi: 10.3389/fbloc.2022.814977.

[10] J. Feist, G. Greico, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. 2019 IEEE/ACM 2nd Int. Workshop Emerging Trends Softw. Eng. Blockchain (WETSEB)*, 2019, pp. 8–15, doi: 10.1109/WETSEB.2019.00008.

[11] Z. Zhen, X. Zhao, J. Zhang, Y. Wang, and H. Chen, "DA-GNN: A smart contract vulnerability detection method based on dual attention graph neural network," *Comput. Netw.*, vol. 242, p. 110238, 2024, doi: 10.1016/j.comnet.2024.110238.

[12] C. Wang, R. Tian, J. Hu, and Z. Ma, "A trend graph attention network for traffic prediction," *Inf. Sci.*, vol. 623, pp. 275–292, 2023, doi: 10.1016/j.ins.2022.12.048.

[13] C. Ma, S. Liu, and G. Xu, "HGAT: Smart contract vulnerability detection method based on hierarchical graph attention network," *J. Cloud Comput.: Adv. Syst. Appl.*, vol. 12, p. 93, 2023, doi: 10.1186/s13677-023-00459-x.

[14] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?," *arXiv preprint arXiv:2105.14491*, 2022, doi: 10.48550/arXiv.2105.14491.

[15] J. Chen and H. Chen, "Edge featured graph attention network," in *Artificial Neural Networks and Machine Learning – ICANN 2021*, vol. 12891, I. Farkaš, P. Masulli, and S. Wermter, Eds. Cham: Springer, 2021, pp. 253–264, doi: 10.1007/978-3-030-86362-3_21.

[16] C. Zhang, J. J. Q. Yu, and Y. Liu, "Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting," *IEEE Access*, vol. 7, pp. 166246–166256, 2019, doi: 10.1109/ACCESS.2019.2953888.

[17] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "SmartBugs: A framework to analyze Solidity smart contracts," in *Proc. 35th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, 2020, pp. 1349–1352, doi: 10.48550/arXiv.2007.04771.

[18] S. Y. Chavhan, S. Kumar, and A. Karkare, "ScrawlD: A dataset of real-world Ethereum smart contracts labelled with vulnerabilities," *arXiv preprint arXiv:2202.11409*, 2022, doi: 10.48550/arXiv.2202.11409.