SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

Metaheuristic-Optimized Vulnerability-Aware Graph Attention Fusion Framework for Smart Contract Vulnerability Detection

Mrs. Saltanat Mohammed Abdullah Shaikh[1*], Dr. Sangeeta Vhatkar[2]

[1*]PhD Scholar – Information Technology, Thakur College of Engineering & Technology. E-mail: 1032221794@tcetmumbai.in
[2]Professor – Information Technology, Thakur College of Engineering & Technology. E-mail: sangeeta.vhatkar@thakureducation.org

ABSTRACT
Ethereum smart contracts enable decentralized applications, yet they are highly prone to vulnerabilities, which results in severe financial losses.
Objective: This research develops an intelligent and automated detection framework called Osprey-Hen Optimized Gated Graph Sequence with Vulnerability Aware Graph Attention (Oshen-VAGA-GGS) Network for detecting vulnerabilities in Ethereum smart contracts.
Method: The acquired data are preprocessed via normalizing Solidity, stripping comments, flattening inheritance. Moreover, a heterogeneous multigraph is constructed by combining Abstract Syntax Tree (AST), basic-block Control Flow Graph (CFG), def-use Data Flow Graph (DFG), and inter-contract call graph using Slither, and labelled edges as control/data/call. Then significant features are extracted via Word2Vec-embed opcodes and identifiers, numeric literal bins, positional encodings and one-hot node-types and balance the data by SMOTE during training only. These features are passed to the proposed Gated Graph Sequence Neural Network (GGNN) and Set2Set readout with attention pooling is added to give contract-level and function-level logits. To enhance the efficacy of the proposed Oshen-VAGA-GGS, an optimization is developed using wrap AdamW with OsHen metaheuristic global osprey exploration, local hen exploitation.
Novelty: The VAGA enhances heterogeneous graph representations by explicitly modeling vulnerability semantics. The framework represents smart contracts using structural and semantic program information and employs gated graph propagation to learn long-range dependencies among contract components.
Findings: Comprehensive experiments are conducted against GNN, Bi-LSTM, MLP+LSTM, and BERT-ATT-BiLSTM baselines. With 80% training data, the corresponding results increase to 98.57%, 98.56%, 99.79%, 98.57%, and 98.56%, respectively. In the ablation analysis, the complete GGNN + OsHen-VAGA configuration achieves 98.80% accuracy and 98.79% F1-score, compared with 94.99% for the baseline GGNN.

Keywords: Blockchain Security, Ethereum Smart Contracts, Vulnerability Detection, Graph Attention Networks, Metaheuristic Optimization, Smart Contract Security Analysis, Neural Models for Code Analysis, Fusion of Heterogeneous Graph Representations

INTRODUCTION

The aggressive growth of the blockchain-based applications has placed Ethereum as one of the most popular platforms when it comes to decentralized finance (DeFi), tokenization, and distributed applications. The technology of smart contracts is the heart of the Ethereum ecosystem and it is self-enforcing programs that are deployed to the blockchain to manage assets and implement the terms of digital agreements. Smart contracts have a reputation for being highly vulnerable to attacks given their immutability and financial value, and can potentially transform the field of trustless computation and digital governance. Typical vulnerabilities have resulted in costly damage to the blockchain sector on numerous occasions . Discovering and countering these vulnerabilities is thus an important research field because even small bugs in the programming can lead to stolen or frozen millions of dollars [1].

Over the last few years, several researchers explored numerous methods of how to detect vulnerability in Ethereum smart contracts. Symbolic execution: Static analysis tools including Oyente, Mythril and Slither have been customized extensively to identify common errors by executing them symbolically and applying rules to verify [3]. Geometric analysis. Deep learning techniques have been used to analyze contract code, bytecode, and structural representations . In addition, given the hybrid increases around ASTs, CFGs, and DFGs, there has been such improvement in vulnerability localisation. These developments have raised the prospect of smart contract analysis using intelligent graph-based models in an automated process[4].

Although great strides have been made, there are a few disadvantages in the existing vulnerability detection model . Static analyzers tend to have high levels of false positives and a lack of scalability, whereas deeper learning models can be imbalanced, uninterpretable, and lack generalizability in novel contract patterns [5]. Moreover, the existing models do not capture the multirelational structural dependency of heterogeneous graphs, and hence they cannot capture the underlying semantic interactions between functions and external functions. The nature of such gaps requires the building of more representative, understandable, and

optimization-friendly architectures of graph learning that can be relied upon to identify Ethereum smart contract vulnerabilities . This research has made the following contribution.

- The study introduced GGNN with multi-head graph attention, which can infer both structural, semantic, and inter-contract relationships in Ethereum smart contracts to predict both the function-level and contract-level vulnerabilities.
- The combination of heterogeneous graph structures with rich node features such as Word2Vec opcodes and identifiers, numeric literal bins, positional encodings, one-hot node types and balanced data via SMOTE, allowing the accurate modeling of security-relevant patterns.
- OsHen metaheuristic with wrap AdamW is developed to explore the global space and exploit the local space, automatically optimizing hyperparameters and training with BCE-with-logits and focal loss to increase the detection rate.
- Our novelty is the Vulnerability-Aware Graph Attention (VAGA) mechanism is proposed to integrate security semantics into graph attention, enabling the model to assign higher importance to vulnerability-critical nodes and heterogeneous graph relationships rather like in GAT than relying solely on structural similarity. VAGA designed to improve representation learning, parameter optimization, and multi-class smart-contract vulnerability detection while maintaining scalability.

2. Literature Review

In 2024, Guo et al. [6] developed a Multi-Scale Encoder Vulnerability Detection (MEVD) framework, which employed a Surface Feature Encoder, Base Transformer Encoder, and Detail CNN Encoder in combination with a Deep Residual Shrinkage Network to learn semantic, global, and local code features. The complexity of the model and the difficulty in handling very large contracts were weaknesses.

In 2025, Pang et al. [7] suggested ConSense, a GNN-based Contract Sensing Framework to detect vulnerabilities in smart contracts through the generation of contract graphs preserving their structural and semantic context, combined with Explore Former to locally and globally combine context at multiple scales. Limited application areas included possible loss of information during such graph creation and difficulties with complicated relationships of contracts.

In 2024, Kim et al. [8] presented was Transformer-based Large Language Model (LLMs) based Smart Contract Vulnerability Detection. The process employed over-sampling for mitigating the imbalance in a curated selection of Solidity datasets to fine-tune eight pre-trained LLMs. The dependence on the quality of datasets, computation cost, and challenges facing the generalization to unknown vulnerabilities were its weaknesses.

In 2024, Colin et al. [9] used an MLP and a hybrid model of MLP and LSTM, respectively, to predict the smart contract vulnerabilities. This was done through the extraction of features, Opcodes, CFG, and AST, and dataset balancing, introducing bugs into the dataset, and standardization of SWC based on elements. Limitations encompassed variation in Solidity version as well as limited detection space in the earlier ML methods.

In 2024, Ehsan et al.[10] used machine learning approaches are XGB, RF, LightGBM, and Extra Trees in the identification of attacks in fog computing smart contracts. It involved TF-IDF, Bag of words and N-gram feature extraction and testing the model using accuracy, precision, recall, F1, and the time taken. Weaknesses were moderate sensitivity of detection and performance-computational complexity trade-offs. In 2024, El Haddouti et al. [11] utilized the ML algorithm to conduct the security audit of smart contracts in an automatic manner. The phenomena entailed vulnerability detection, classification, and testing performance to minimize manual labor and execution time. The limitations were the cost and complexity of the available tools, as well as due to the dependence on empirical benchmarking, validation.

In 2025,A. M. Alghamdi et al[12]. has combine the Hierarchical Graph Attention network wuth multi-agent reinforcement learning to model contract interaction and state transition but it focuses on reinforcement learning and interaction policies rather than vulnerability semantics. In 2025, M. Li et al.[13] adds global virtual nodes fuses with graph attention with runtime behavior but it only uses global behaviour information instead if vulnerability aware semantic attention.

In 2024, Z. Zhen et al.[14] introduces dual attention to improve feature extraction for smartcontract graph however dual attention improves features learning is not designed to prioritize security critical operationss explicitly. In 2024, X. Wang et al[15] Combines feature graphs with multiple attention mechanisms for feature aggregation however it do not incorporate explicit vulnerability semantics. In 2024, X. Liu et al.[16] has introduce Adaptive feature perception attention which select dynamically information code snippet before vulnerability detection and it only operates at code snippet level instead of heterogeneous graph reasoning. In 2024, X. Zhang et al.[17] used Combines heterogeneous semantic graphs with pretrained code representations but Does not explicitly model vulnerability semantics or adaptive security risk.

2.1. Problem Statement

The security threats on Ethereum smart contracts are increasingly becoming diverse and pose a given financial and operational risk. In spite of many detection methods based on ML, GNNs, transformers, and various hybrids, there are still questions regarding coverage, accuracy, scalability, generalization, and efficient detection of novel or exotic vulnerabilities. Table 1 Positioning the Proposed Vulnerability-Aware Graph

Attention (VAGA) among State-of-the-Art Graph Attention Mechanisms like SCVHunter [4], DA-GNN [13],GaGAT [14], GAT [17], Edge-Aware GAT [18].

Table 1: Positioning the Proposed Vulnerability-Aware Graph Attention (VAGA) among State-of-the-Art Graph Attention Mechanisms

Method	Node Attention	Edge Attention	Global Context	Vulnerability Semantics	Risk-Aware Learning	Heterogeneous Graph Support	Explainability	Smart Contract Specific
GAT [17]	✓	✗	✗	✗	✗	✗	Moderate	✗
Edge-Aware GAT [18]	✓	✓	✗	✗	✗	Partial	✓	✗
DA-GNN [13]	✓	Partial	✗	✗	✗	✓	Moderate	✓
GaGAT [14]	✓	✓	✓	✗	✗	✓	Moderate	✓
SCVHunter [4]	✓	✓	✗	✗	✗	✓	Moderate	✓
Proposed VAGA	✓	✓	✓	✓	✓	✓	High	✓

3. Proposed Methodology

The Oshen-VAGA-GGS framework starts by loading Solidity smart contracts that are verified by Etherscan, and contains examples of arithmetic bugs, timestamp/oracle abuse, and other vulnerabilities of interest. Initially, 10,000+ data is sourced from the malicious-smart-contract-dataset [19], ReentrancyStudy-Data [20], and SCRUBD [21], smartbugs-wild[22] which undergo preprocessing. The raw contracts are preprocessed by removing comments and normalizing the code, then flattening inheritance hierarchy. An AST, basic-block CFG, def-use DFG, and inter-contract call graph are then combined into a heterogeneous multigraph, with edges labeled as control, data or call. Based on this graph node-level representations are derived, such as Word2Vec embeddings of opcodes and identifiers, numeric literal bins, positional encodings, one-hot node-type encodings and data balancing is performed. The GGNN is fed with these features and propagates using K steps with edge-type-specific transformations and GRU gating, augmented by multi-head graph attention to learn the importance of neighbours. Set2Set readout and attention pooling are used to aggregate the resulting node embeddings to obtain contract-level and function-level logits. Lastly, wrap AdamW with the OsHen metaheuristic is used to optimize the framework. Figure 1 presents the Schematic overview of the proposed Oshen-VAGA-GGS framework to identify vulnerabilities of Ethereum smart contracts. Compared with existing GAT mechanisms, this Novel VAGA introduces three components.

- Vulnerability semantic embeddings for security-sensitive operations (e.g., call, delegatecall, tx.origin, block.timestamp).
- Node risk estimation, which learns how security-critical each node is before message aggregation.
- Heterogeneous edge-type embeddings, allowing the model to differentiate AST, CFG, DFG, and call graph relationships during attention computation.

3.1 Data Acquisition

In the study, real time information on the Ethereum blockchain is obtained 10,000+ data is sourced, where validated Solidity smart contracts, their source code, ABI, and deployment information are constantly accessed. Security tools, including Slither, Mythril, and Oyente, are used to analyze the collected contracts and identify multi classvulnerabilities and classify them into eight categories, which include: Secure, Dangerous Delegatecall (DE), Ether Frozen (EF), Integer Overflow (OF), Block Number Dependency , Re-entrancy (RE), Ether Strict Equality (SE), and Timestamp Dependency (TP).

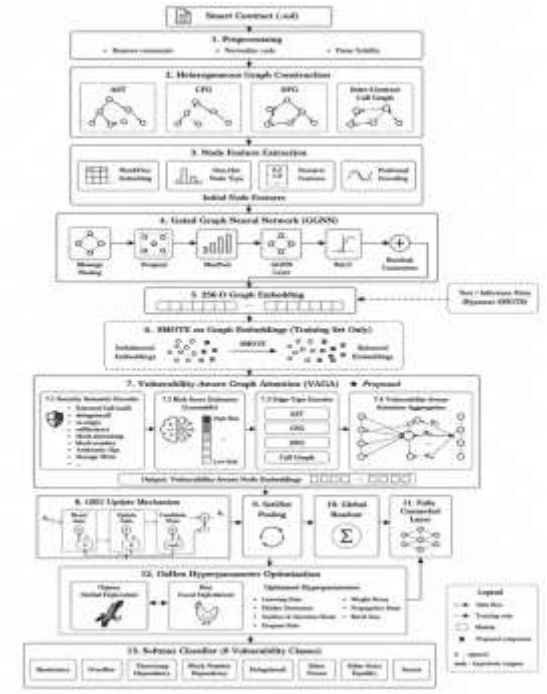

Figure 1: Architecture of Oshen-VAGA model

3.2 Preprocessing

To feed the smart contract data to the proposed OsHen-VAGA-GGS framework, a systematic preprocessing pipeline is applied to maintain consistency and semantic integrity and to make the data ready to be used in graph construction.

3.2.1 Solidity Code Normalization

Solidity smart contracts tend to have syntactic differences in terms of formatting, naming, and literal representation, which add noise to the feature extraction process. To reduce this, the source code is normalised by canonical transformations that do not affect semantic correctness [23]. Assume that the solidity contract be represented as a token sequence is given as $S = \{t_1, t_2, \dots, t_n\}, t_i \in \nu$, where S is the original token sequence. t_i represents the i-th token, and ν denotes the vocabulary set. Here, the normalization function $\mathcal{N}(\cdot)$ is defined that maps the original token sequence to a normalized sequence S' is defined as $S' = \mathcal{N}(S) = \{\tau_1, \tau_2, \dots, \tau_n\}$, where, reserved keywords and operators remain unchanged, and identifiers are normalized using hash-based encoding and is determined using Eq. (1). Here, $h(\cdot)$ is the consistent hash function and I is the set of user-defined identifiers.

$$\tau_i = \begin{cases} h(t_i), & if\ t_i \in \text{I(identifiers)} \\ t_i, & otherwise \end{cases} \quad (1)$$

3.2.2 Comment Removal and Inheritance Flattening

The comment removal operator $\mathrm{C}(\cdot)$ removes non-executable comment tokens from the normalized solidity code S' to reduce noise and the quality of graph construction is improved [24]. This ensures that only semantically relevant code tokens are retained for further analysis. Given the normalized sequence S', the comment-stripping operator $\mathrm{C}(\cdot)$ is given by Eq. (2), where, C_{set} represents the set of all tokens corresponding to single line and multi-line comments. This results in a cleaner source representation is defined as $|S^{"}| \leq |S'|$, $S^{"} \subseteq S'$.

$$S^{"} = \mathrm{C}(S') = \{\tau_i \in S' | \tau_i \notin \mathrm{C}_{set}\} \quad (2)$$

Inheritance flattening combines all functions and state variables of base contracts into a single contract to create a self-contained structure. This guarantees flawless analysis of control and data flows without dependence on external contract hierarchies. Let the contract C_k inherit from a set of base contracts $\mathfrak{B}_k = \{B_1, B_2, \dots, B_m\}$. The linearization function $\mathcal{F}(\cdot)$ is defined that performs C3 linearization is given by Eq. (3), where, $\hat{C}_k$ is the flattened contract, all functions and state variables from $B_i \in \mathfrak{B}_k$ are merged into $\hat{C}_k$, and function resolution respects solidity's Method Resolution Order (MRO). This ensures a single-level contract structure which is suitable for AST and CFG construction.

$$\hat{C}_k = \mathcal{F}(C_k, \mathfrak{B}_k) \quad (3)$$

3.3 Graph Construction

The relationships among Ethereum smart contracts are complex syntactic, semantic and inter-contract. To represent these aspects of vulnerability detection in full, we build a heterogeneous multigraph $G = (V, E, \tau_V, \tau_E)$ that integrates multiple program representations, such as ASTs, CFGs, DFGs, and inter-contract call graphs. Here, V represents the set of nodes representing program entities including operations, variables, literals, functions, and contrast identifiers, $E \subseteq V \times V$ indicates the set of edges representing the relation between nodes, τ_V and τ_E are the node types and edge types. The multigraph is formally determined as $G = \bigcup_{i=1}^{4} G_i,\ G_i = (V_i, E_i)$, where each G_i captures a different type of program representation. The preprocessed data is the input to the graph construction stage.

3.3.1 Abstract Syntax Tree

The AST [4][16] represents the hierarchical nature of Solidity code, which allows the GGNN to learn structural patterns of potentially vulnerable constructs, such as unsafe arithmetic operations or re-entrancy-prone functions. The formulation of AST is determined using Eq. (4), where, G_{AST} is the AST represented as a graph, V_{AST} signifies a set of nodes in the AST, where each node $v \in V_{AST}$ corresponds to a syntactic element of the solidity code, v_p represents the parent node in the AST, v_c is the child node directly connected to v_p representing a syntactic element nested under the parent.

$$G_{AST} = (V_{AST}, E_{AST}),\ E_{AST} = \{(v_p, v_c) | v_c \ is \ child \ of \ v_p\} \quad (4)$$

3.3.2 Control Flow Graph

The temporal relationships between instructions are captured by CFG, and this allows the identification of vulnerabilities associated with conditional execution, loops, or early termination that are typical of re-entrancy and access control bugs [4][16]. Edges represent possible flow of control between block which is determined using Eq. (5), where, G_{CFG} is the CFG representing the execution paths of the smart contract, V_{CFG} represents the set of nodes in the CFG, where each node $v \in V_{CFG}$ signifies a basic block, E_{CFG} represented as the set of directed edges in the CFG representing possible control flow between basic blocks, v_i indicates the source node in the CFG, and v_j is the target \node that execute immediately after v_i, according to the program's control flow.

$$G_{CFG} = (V_{CFG}, E_{CFG}),\ E_{CFG} = \{(v_i, v_j) | v_j \ execute \ after \ v_i\} \quad (5)$$

3.3.3 Data Flow Graph

The DFG [4][26] tracks data propagation, which is essential to detecting vulnerabilities like arithmetic bugs or unchecked external inputs, and variable-level attention, allowing the GGNN to pay attention to variables that affect security-critical operations.The construction of DFG is defined by Eq. (6), where, G_{DFG} is the DFG representing variable definitions and uses in the contract, V_{DFG} denotes the set of node $v \in V_{DFG}$ corresponds to a program element that defines or uses a variable, E_{DFG} is the set of directed edges representing def-use relationships, v_d represented as the node where a variable is defined (assigned a value), v_u is the node where the same variable is subsequently used.

$$G_{DFG} = (V_{DFG}, E_{DFG}),\ E_{DFG} = \{(v_d, v_u) | variable \ defined \ at \ v_d \ and \ used \ at \ v_u\} \quad (6)$$

3.3.4 Inter-Contract Call Graph

Inter-Contract Call Graph [4][27] is used to capture cross-contract interactions, which are essential to identify reentrancy vulnerability, and allows modeling of function-level dependencies across multiple contracts.The call graph models these inter-contract and intra-contract function invocations which are defined by Eq. (7), where, G_{Call} is the inter-contract Call Graph representing function invocations, V_{Call} denotes the set of nodes in the call graph, where each node $f \in V_{Call}$ corresponds to a function in a smart contract, E_{Call} indicates the set of directed edges denoting function calls, f_i is the caller function node, and f_j denoted as the callee function that is invoked by f_i.

$$G_{Call} = (V_{Call}, E_{Call}),\ E_{Call} = \{(f_i, f_j) | f_i \ calls \ f_j\} \quad (7)$$

3.3.5 Edge Typing and Heterogeneity

The assignment of edge types reflects various semantics of relationships between nodes, such as control, data, call, syntax, and allows the model to differentiate between various interactions[28]. To represent multiple semantic relations, each edge $e \in E$ is assigned a type $\tau_e \in \{control, data, call, syntax\}$ which is determined using Eq. (8). Finally, the node types $\tau_v \in \tau_V$ are assigned based on roles such as literals, operations, function definitions, state variables.

$$\tau_e = \begin{cases} control, & e \in E_{CFG} \\ data, & e \in E_{DFG} \\ call, & e \in E_{Call} \\ syntax, & e \in E_{AST} \end{cases} \quad (8)$$

3.3.6 Graph Integration and Alignment

The four individual graphs are merged into a single heterogeneous multigraph is defined by Eq. (9).

$$G = (V, E) = \bigcup_{i=1}^{4} (V_i, E_i) \quad (9)$$

The integrated graph maintains syntactic, semantic, control and inter-contract interactions and is a holistic illustration of each smart contract and is ready to be processed by a GNN.

3.4 Feature Extraction

After constructing the heterogeneous multigraph $G = (V, E, \tau_V, \tau_E)$, the feature extraction incorporates semantic embeddings, syntactic labels, numeric representations, and positional encodings to produce an initial node feature matrix which is used as input to the proposed Oshen-VAGA-GGS model.

3.4.1 Word2Vec Embeddings

Word2Vec [29][30] trains semantic representations of EVM opcodes and Solidity identifier allows the model to learn contextual relationships. To perform opcode embedding, first assume that the compiled EVM bytecode for a contract be represented as an ordered sequence of opcodes is represented as $O = \{o_1, o_2, \dots, o_n\}$, $o_i \in \nu_o$, where, O is the opcode sequence, ν_o is the vocabulary of unique opcodes. Here the Skip-Gram approach is employed for training the Word2Vec model to learn an embedding function is given as $\phi_O: \nu_O \to \mathbb{R}^{d_O}$, where, ϕ_O is the embedding function learned using Word2Vec for EVM opcodes, which maps each opcode to a dense vector representation, ν_O represents the vocabulary of opcodes, and $\mathbb{R}^{dO}$ is the embedding space of dimesion d_O, where each opcode is represented as a continuous-valued vector. For each opcode o_i, the embedding is defined by Eq. (10).

$$x_{o_i} = \phi_O(o_i) \in \mathbb{R}^{dO} \quad (10)$$

To perform the identifier embedding, assume that the set of identifiers extracted from solidity source code is represented as $I = \{id_1, id_2, \dots, id_m\}$, $id_j \in \nu_I$, where, ν_I indicates the vocabulary of unique identifiers. The Word2Vec model is used for training the identifier embedding process is determined as $\phi_I: \nu_I \to \mathbb{R}^{d_I}$. For each identifier id_j, the embedding is defined by Eq. (11). This captures semantic similarities between identifiers, such as variable names related to balances, ownership, and access control.

$$x_{id_j} = \phi_I(id_j) \in \mathbb{R}^{d_I} \quad (11)$$

Smart contracts often contain numeric constants that are critical for security. Numeric Literal Bins are grouped into discrete bins to capture ranges of values effectively, which helps to detect vulnerabilities. Numeric constants significantly influence contract security. Here, binning is employed instead of raw scalar encoding to represent numeric literals to enhance robustness against outliers. Let the set of numeric literals is represented as $L = \{l_1, l_2, \dots, l_p\}$. The binning function $\beta(\cdot)$ is defined that maps each literal l_i into discrete bin index b_k is given as $b_k = \beta(l_i), b_k \in \{1,2, \dots, B\}$. Each bin index is then represented using one-hot encoding is determined using Eq. (12). This strategy captures numeric ranges relevant to security-critical operations.

$$xl_i = \delta(\beta(l_i)) \in \{0,1\}^B \quad (12)$$

To capture the sequential structure of smart contracts, sinusoidal positional encodings are added to the node features. The positional encodings encode the relative position of nodes to retain the sequential and structural ordering information in the graph to enable better pattern recognition. It is incorporated into the feature vectors to preserve the relative ordering of tokens and instructions. Here, the sinusoidal positional embeddings are employed for a node v_i at position p_i in its sequence is defined by Eq. (13), where, p_i is the position index of the node, d represents the embedding dimension, and k denotes the dimension index.

$$PE(p_i, 2k) = sin\left(\frac{p_i}{10000^{2k/d}}\right), \; PE(p_i, 2k+1) = cos\left(\frac{p_i}{10000^{2k/d}}\right) \quad (13)$$

3.4.2 One-Hot Encoding

One-Hot Encoding [29][30] represents the different node types uniquely, enabling the model to distinguish between various program elements during graph-based learning. The nodes in the heterogeneous multigraph belong to different types such as statements, expressions, variables, functions, literals. Here, one-hot encoding

is applied to represent the node categories. Let the set of all node types is signified as $\tau_V = \{t_1, t_2, \dots, t_K\}$. For each node v_i, its type $t(v_i)$ is encoded as a one-hot vector is determined using Eq. (14), where, $\delta(\cdot)$ denotes the one-hot function and K is the total number of node types. This representation enables the GGNN to differentiate among the various elements of the program and perform type-specific transformations.

$$x_{t_i} = \delta(t(v_i)) \in \{0,1\}^K \quad (14)$$

3.4.3 Final Node Feature Representation

For each node $v_i \in V$, all extracted features are concatenated into a unified feature vector is defined by Eq. (15), where, || represents the concatenation. The resulting node feature matrix is determined using Eq. (16).

$$x_i = [x_{o_i} \| x_{id_i} \| x_{t_i} \| x_{l_i} \| PE(p_i)] \quad (15)$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{|V|} \end{bmatrix} \in \mathbb{R}^{|V| \times d} \quad (16)$$

3.5 Data Balancing

Training Oshen-VAGA-GGS model with such imbalanced data may result in biased predictions, which are in favour of the majority class. To minimize this, Synthetic Minority Over-sampling Technique (SMOTE) [31][32] is used to create synthetic samples of the minority group during training of the dataset.

Let the feature matrix after the extraction is denoted as $X = [x_1, x_2, \dots, x_N] \in \mathbb{R}^{N \times d}, y = [y_1, y_2, \dots, y_N] \in \{0,1\}^N$, where, N represents the total number of samples, d represents the feature dimension, $y_i = 1$ indicates a vulnerable contract, and $y_i = 0$ represents the safe contract. To generate synthetic samples, SMOTE is employed for each minority sample x_i, find its k-nearest neighbors in feature space (k is a hyperparameter) and select the neighbor x_{nn} randomly. Furthermore, the synthetic samples are generated using Eq. (17) and the minority class label $y_{new} = 1$ is assigned to the synthetic sample. Then, repeat the process until the minority class reaches the desired balance relative to the majority class.

$$x_{new} = x_i + \lambda \cdot (x_{nn} - x_i), \ \ \lambda \sim Uniform(0,1) \quad (17)$$

3.6 Vulnerability Aware Graph Attention Detection

The proposed Oshen-VAGA-GGS framework utilizes a GGNN augmented with vulnerability aware graph attention to effectively model structural, semantic and inter-contract relationships within Ethereum smart contracts. The architecture involves three main parts: graph propagation through GGNN, proposed Osprey optimization with VAGA attention and readout pooling to make contract-level and function-level predictions by giving input as balanced data.

The GGNN is a message-passing neural network that aims to transmit information across nodes in a graph, while respecting edge types [18]. Assume that the heterogeneous multigraph be $G = (V, E, \tau_V, \tau_E)$ with node features $X \in \mathbb{R}^{|V| \times d}$. The GGNN performs K propagation steps, updating node states iteratively, which is given by Eq. (18), where, $h_v^{(k)}$ denotes hidden state of node v at step k, $\mathcal{N}(v)$ represents the set of neighbors of v, τ_e represents edge type between v and neighbor u, $f_{\tau_e}(\cdot)$ denotes the edge-type-specific linear transformation, and GRU is the GRU controlling information flow across steps [17][29]. This mechanism allows the GGNN to aggregate multi-hop information while differentiating the influence of control, data, call, and syntax edges. For each edge type $\tau_e \in \tau_E$, a dedicated weight matrix $W_{\tau_e} \in \mathbb{R}^{d \times d}$ transforms messages is represented as $m_{u \to v}^{(\tau_e)} = W_{\tau_e} h_u^{(k-1)}$. This allows the model to treat various semantic relationships differently, paying extra attention to data-flow edges when detecting arithmetic bugs.

$$h_v^{(k)} = GRU\left(h_v^{(k-1)}, \sum_{u \in \mathcal{N}(v)} f_{\tau_e}\left(h_u^{(k-1)}\right)\right), \ k = 1, \dots, K \quad (18)$$

The GRU gating mechanism combines the previous node state with incoming messages is determined using Eq. (19), where, $m_v^{(k)} = \sum_{u \in \mathcal{N}(v)} m_{u \to v}^{(\tau_e)}$. This avoids excessive smoothing when communicating messages and enables nodes to maintain significant local information when combining global context.

$$h_v^{(k)} = GRU(h_v^{(k-1)}, m_v^{(k)}) \quad (19)$$

Vulnerability Semantic Encoder introduces explicit security information into the attention mechanism.Suppose Si=[s1,s2,...,sm] extracts security semantics which include security feature information such delegatecall, call.value, tx.origin, block.timestamp, arithmetic, storage write etc. For VAGA, classify graph nodes into a soft

three-level semantic preference i.e P(vj)∈{critical,contextual,non-critical} shown in Table 2. Then the equation for semantic encoder will be zi=WsSi+bs where Ws learn security semantics.
Where zi=[eiword||eitype||eivuln||eiflow||eipos]
Secondly the Risk Score Estimation is evaluated by predicting the rist score estimation of each nodes ri = σ(Wrzi+br) Where 0 ≤ ri ≤ 1. Interpretation of model give more attention to high risk node and less attention to safe node for example delegatecall,arithmetic,assignment node can have risk 0.95, 0.88, 0.15 respectively. Edge-Type importance ωe than carried out using softmax to the aggregated graph AST,CFG,DFG,CALL. Thus risk estimation of node j multi-dimensional rather than keyword based i.e Rj=σ(wsSj+wcCj+wdDj+wtTj+wxXj+wpPj)

Algorithm 1: Algorithm of Vulnerability-Aware Graph Attention

Input: Node features H, graph edges E, edge types T, and vulnerability-semantic features S
Output: Vulnerability-aware node representations H′
VAGA(H, E, T, S)

Encode vulnerability semantics:
zi=WsSi+bs
Estimate vulnerability risk:
ri = σ(Wrzi+br)
For each edge (i,j) ∈ E:
Eij = WeTij
eij = LeakyReLU(aT[Whi ‖Whj]+λri+γωe)
Normalize attention:
αij = softmaxj(eij)
Aggregate neighbors:
hi' = ReLU(Σj∈N(i) αij Whj)

Return H'

Where Sj, Cj, Dj, Tj, Xj, Pj, w*, σ are security-sensitive operation score, control-flow risk; data-flow risk; external-call/transaction interaction risk; vulnerability-type semantic similarity; propagation/dependency risk; learnable or empirically initialized weights; sigmoid function respectivitly. Unlike traditional GAT, proposed model will capture vulnerability aware attention incorporating both risk and edge importances. Where ri is vulnerability risk score, ωe is edge importance λ,γ are learnable scaling coefficients.

eij = LeakyReLU(aT[Whi ‖Whj]+λri+γωe)

Table 2 :Security-Sensitivity Classification and Risk Prior of Smart Contract Operations

Category	Security-Sensitive Operations	Risk Prior (Sj)
Critical	delegatecall, external call, block.timestamp, tx.origin, unchecked arithmetic, state-changing Ether transfer	0.90–1.00
Contextual	require(...), if(...), state-variable read/write, function modifier	0.50–0.65
Non-critical	Variable declaration, ordinary assignment, local initialization, simple arithmetic, return, event/logging	0.05–0.15

The GGNN is augmented with multi-head graph attention to capture neighbor importance dynamically which is defined by Eq. (20), where, H is the total number of attention heads, W_h signifies learnable linear transformation for head h, a_h denotes the attention vector for head h, and || is the concatenation. The weighted sum over neighbors is employed to ypdate node states are given by Eq. (21). This enables the model to prioritize on the most pertinent neighbors and enhances identification of minor vulnerabilities that rely on selective control and data flows.

$$\alpha_{vu}^{(h)} = \frac{\exp(LeakyReLU(a_h^T[W_h h_v \| W_h h_u] + \lambda ri + \gamma \omega e))}{\sum_{k \in \mathcal{N}(v)} \exp(LeakyReLU(a_h^T[W_h h_v \| W_h h_k] + \lambda ri + \gamma \omega e)} \quad (20)$$

$$h_v^{(k)} = ||_{h=1}^{H} \sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(h)} W_h h_u \quad (21)$$

After K propagation steps, a readout function converts node embeddings into graph-level representations for contract and function-level predictions. Here, Set2Set pooling models complex dependencies between nodes and its formulation is given by Eq. (22-23), where, q_t is the query vector at iteration t, q_{t-1} represents the previous state from iteration $t-1$, r_{t-1} denotes the readout vector from iteration $t-1$, r_t indicates the readout vector at iteration t, V denotes set of all nodes in the heterogeneous multigraph and and h_v denotes the hidden embedding vector of node v after GGNN propagation and graph attention layer, $\alpha_v^{(t)}$ represented as the attention weight assigned to node v at iteration t is defined by Eq. (24), here, softmax attention emphasizes important nodes, enhancing the detection of security-critical patterns.

$$q_t = LSTM(q_{t-1}, r_{t-1}) \quad (22)$$
$$r_t = \sum_{v \in V} \alpha_v^{(t)} h_v \quad (23)$$
$$\alpha_v^{(t)} = softmax(h_v^T q_t) \quad (24)$$

An additional attention mechanism computes node importance scores for graph-level embeddings are determined using Eq. (25-26), here, $w \in \mathbb{R}^d$ represents the learnable vector, and $z \in \mathbb{R}^d$ indicates the final graph-level embedding. Furthermore, the graph embedding z is passed via fully connected layer to produce logits and its mathematical formulation is defined by Eq. (27), where, $\hat{y}_{contract} \in \mathbb{R}$ is the contract-level vulnerability prediction, and $\hat{y}_{function} \in \mathbb{R}$ denotes the function-level predictions for all functions F in the contract.

$$z = \sum_{v \in V} \beta_v h_v \quad (25)$$
$$\beta_v = softmax(w^T h_v) \quad (26)$$
$$\hat{y}_{contract}, \hat{y}_{function} = MLP(z) \quad (27)$$

3.7 Optimization Strategy

The optimization method uses a hybrid method that integrates AdamW optimizer and Osprey-Hen metaheuristic. Global exploration is steered by the Osprey phase that utilizes Levy flight sampling to efficiently explore the solution space. Local exploitation is implemented in the Hen phase by using brood perturbation and selective updates. Critical parameters including learning rate, attention heads, K, dropout, and class weights are optimized to improve optimization. It describes the population updatesPopulation size: 100, Number of trials : 100, Validation set : 10% of testing data, Random seed : 42, stopping rule: Maximum epoch, objective function : Accuracy, objective function evaluation and the iterative process to get the best solution.

3.7.1 Global Osprey Exploration

The OOA [24] is a population-based optimization algorithm that searches to obtain optimal solutions by iterative search using multiple agents. The ospreys are candidate's solutions, modeled as vectors in the searching space using Eq. (28). The OOA population is the aggregate of the set of ospreys, which are randomly initialized at the beginning by Eq. (29). Here, X denotes the population matrix of ospreys' locations, X_i denotes the $i-th$ osprey, $x_{i,j}$ indicated as the j-th dimension, N represents the total quantity of ospreys, m indicates the problem variables, $r_{i,j}$ denoted as the random value in the interval [0,1], ub_j and lb_j signifies the upper bound and lower bound.

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_i \\ \vdots \\ X_N \end{bmatrix}_{N \times m} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,j} & \cdots & x_{1,m} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ x_{i,1} & \cdots & x_{i,j} & \cdots & x_{i,m} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,j} & \cdots & x_{N,m} \end{bmatrix}_{N \times m} \quad (28)$$

$$x_{i,j} = lb_j + r_{i,j} \cdot \left(ub_j - lb_j\right), i = 1,2,\ldots,N, j = 1,2,\ldots,m \quad (29)$$

The values of the objective functions that are obtained are collectively expressed as a vector, as in Eq. (30), where, F denotes the value of the objective function vector, and F_i signifies the attained objective function value.

$$F = \begin{bmatrix} F_1 \\ \vdots \\ F_i \\ \vdots \\ F_N \end{bmatrix}_{N \times 1} = \begin{bmatrix} F(X_1) \\ \vdots \\ F(X_i) \\ \vdots \\ F(X_N) \end{bmatrix}_{N \times 1} \quad (30)$$

The collection of these target fishes per osprey is mathematically determined by means of Eq. (31), where, FP_i signifies the fish positions set, and X_{best} represents the best candidate solution.

$$FP_i = \{X_k | k \in \{1,2,\ldots,N\} \wedge F_k < F_i\} \cup \{X_{best}\} \quad (31)$$

The osprey selects the position of one of the fishes and flies towards it in simulating an attack randomly. A new osprey position is then obtained by using Eq. (32-33). When this new position yields a better value of the objective function then the previous position is replaced according to Eq. (34). Here, $X_i^{P_1}$ represents the position of i-th osprey depends upon the first phase, $x_{i,j}^{P_1}$ signifies the j-th dimension, $F_i^{P_1}$ denotes the objective function value, SF_i signifies the selected fish for i-th osprey, $SF_{i,j}$ represent its j-th dimension, $I_{i,j}$ is the random values from the set {1,2}, and $r_{i,j}$ denotes the random values in the interval [0,1].

$$x_{i,j}^{P_1} = x_{i,j} + r_{i,j} \cdot (SF_{i,j} - I_{i,j} \cdot x_{ij}) \quad (32)$$

$$x_{i,j}^{P_1} = \begin{cases} x_{i,j}^{P_1}, \ lb_j \le x_{i,j}^{P_1} \le ub_j; \\ lb_j, \ x_{i,j}^{P_1} < lb_j \ ; \\ ub_j, \ x_{i,j}^{P_1} > ub_j; \end{cases} \quad (33)$$

$$X_i = \begin{cases} X_i^{P_1}, F_i^{P_1} < F_i \\ X_i, \ else \end{cases} \quad (34)$$

In OOA, each of the population members is allocated a new random location as a location that is suitable to eat fish using Eq. (35-36). When this new position is better than the objective function of the osprey, then it supersedes the previous position of the osprey as given in Eq. (37). Here, X_i^2 represents the position of i-th osprey based on the second phase, $x_{i,j}^2$ signifies the j-th dimension, $F_i^{P_2}$ denotes the value of the objective function, $r_{i,j}$ denotes the random values in the interval [0,1], T represents the iterations and t signifies the iteration counter of the model.

$$x_{i,j}^{P_2} = x_{i,j} + \frac{lb_j + r \cdot (ub_j - lb_j)}{t}, i = 1,2, \dots, N, j = 1,2, \dots, m, t = 1,2, \dots, T, \quad (35)$$

$$x_{i,j}^{P_2} = \begin{cases} x_{i,j}^{P_2}, \ lb_j \le x_{i,j}^{P_2} \le ub_j; \\ lb_j, \ x_{i,j}^{P_2} < lb_j \ ; \\ ub_j, \ x_{i,j}^{P_2} > ub_j; \end{cases} \quad (36)$$

$$X_i = \begin{cases} X_i^{P_2}, F_i^{P_2} < F_i \\ X_i, \ else \end{cases} \quad (37)$$

3.7.2 Local Hen Exploitation

The CSO algorithm [25] is based on the social and hierarchical foraging behavior of chicken flock. Chickens are divided into rooster, hens and chicks, with different foraging abilities in a hierarchical system.

The solution space in CSO algorithm consists of n vectors in j-dimensional space, with n being the population size. They are divided into roosters, hens and chicks depend upon fitness values. The best performers with the worst fitness are roosters (RN), the worst performers with the best fitness are chicks (CN) and the rest are hen group (HN) using Eq. (38-40). This is a hierarchical classification that influences the search dynamics of the algorithm depending on the inherent social structure of chickens.

$$R_i = \{R_1, R_2, \dots, R_{RN}\} \quad (38)$$

$$C_i = \{C_1, C_2, \dots, C_{CN}\} \quad (39)$$

$$H_i = \{H_1, H_2, \dots, H_{HN}\} \quad (40)$$

In the CSO algorithm, every hen is matched with a dominant rooster and every chick with a mother hen. Their location is updated to new positions with the help of unique mathematical Eq. (38-42) that mimic realistic foraging behaviours with the help of these hierarchical relationships. The estimation for location Succession (LS) of rooster groups. In CSO algorithm, the new location $R_{i,j}^t$ of the i-th rooster at the j-th dimension is defined by a Gaussian-distributed random value with mean 0 and variance 2 to add variability by Eq. (41). Fitness function f directs movement towards improved solutions by Eq. (42). A small constant S guarantees the numerical stability by avoiding division by zero.

$$R_{i,j}^{t+1} = R_{i,j}^t[1 + randn(0, \delta^2) \quad (41) \qquad \delta^2 = \begin{cases} 1, & f_i \le f_s \\ e^{\frac{f_s - f_i}{|f_i| + \varepsilon}}, & f_i > f_s \\ s \in [1, n], & s \ne i \end{cases} \quad (42)$$

The calculation for LS of the hen groups. Here, the position of the hen $H_{i,j}^t$ is updated with random influence of

its leader rooster and a randomly chosen individual using Eq. (43). The leader rooster effect (R_{Hi}^t) and the random individual effect (RH^t) are multiplied by the factors of k_1 and k_2, respectively using Eq. (44) (45). The fitness values f_{Hi}, f_{rHi} and f_{RH} informs the extent to which each influence is involved in the update. This process balances learning and dominance of individuals and diversity within the search process.

$$H_{i,j}^{t+1} = H_{i,j}^t + k_1 * rand\left(R_{Hi}^t - M_{i,j}^t\right) + k_2 * rand * (RH^t - H_{i,j}^t) \quad (43)$$

$$k_1 = e^{\frac{f_{Hi} - f_{rHi}}{|f_{Hi}| + \varepsilon}} \quad (44)$$

$$k_2 = e^{f_{RH} - f_{Hi}} \quad (45)$$

The calculation for the LS of the chick groups. In Eq. (46), the i-th chick location $C_{i,j}^t$ is updated according to the mother hen's location Hi_j^t. The movement is scaled by some random factor $F \in (0,2)$ which represents the tendency of the chick to follow its hen but with some randomness.

$$C_{i,j}^{t+1} = C_{i,j}^t + F * (Hi_j^t - C_{i,j}^t) \quad (46)$$

The objective function $F(\vec{X})$ is given in an m-dimensional space that contains a set of n vectors. In this case, n is a number of solutions, and m denotes the vector dimensionality $\vec{X}$. The CSO algorithm initializes a population of size N individuals, which are the position vectors $\vec{X}$ of the objective function. The individuals are categorized as CN chicks, HN hens, RN roosters and MN mother hens. The important parameters like position dimensions (j), iteration limit ($Max\ G$) and initial iteration ($t = 0$) are fixed. The algorithm see whether $t < Max\ G$; in case it is true, it move to the next step, otherwise it halt and print the optimal solution found. Then, the fitness value f_i is calculated of each individual in the population of chicken. Furthermore, whether Eq. (47) is true if true, go for establishing the corresponding connection and define the individuals with fitness value. If not, go to the position coordination updation and the optimal solution is recorded once a better function value appears.

$$t\ mod\ G == 0 \quad (47)$$

The chicken swarm is ranked according to fitness: roosters have the least, hens medium, and chicks the highest degree of fitness. All the groups start to forage, and each of them updates their position coordinates depending on their roles. The objective function value F is computed and in case of a better solution F_{best} is found, then it is noted. The iteration counter t is increased after every update and process returns to determine if $t < Max\ G$. The objective function $F(X_i)$is assessed every update, the best solution F_{best} is stored, and the process repeats till the stopping criterion is achieved. For i-th individual (osprey and chicken) at dimension j is defined by Eq. (48). The training loss combines BCE with logits and focal loss to handle imbalance is determined using Eq. (49).

$$X_{i,j}^{t+1} = \begin{cases} x_{i,j}^{(OOA)} = x_{i,j} + r_{i,j} \cdot \left(SF_{i,j} - I_{i,j} \cdot x_{ij}\right) + \frac{lb_j + r \cdot (ub_j - lb_j)}{t} & if\ OOA\ (osprey) \\ R_{i,j}^{t+1} = R_{i,j}^t[1 + randn(0, \delta^2), & if\ rooster\ (CSO) \\ H_{i,j}^{t+1} = H_{i,j}^t + k_1 * rand\left(R_{Hi}^t - M_{i,j}^t\right) + k_2 * rand * \left(RH^t - H_{i,j}^t\right), & if\ hen\ (CSO) \\ C_{i,j}^{t+1} = C_{i,j}^t + F * \left(Hi_j^t - C_{i,j}^t\right), & if\ chick\ (CSO) \end{cases} \quad (48)$$

$$\mathcal{L} = BCE(\hat{y}, y) + \gamma \cdot Focal(\hat{y}, y) \quad (49)$$

4. Result and Discussion

The Oshen-VAGA-GGS framework is implemented in Python tool based on PyTorch, PyTorch Geometric, and Slither to extract graphs, Word2Vec and SMOTE to process features. Ubuntu 22.04 was tested on an Intel Xeon CPU, NVIDIA RTX 3090, 128 GB RAM, and SSD storage using CUDA 12.0 and cuDNN 8.x. AdamW was optimized using the OsHen metaheuristic, BCE-with-logits and focal loss, on repository-level data splits of 70-30%,Fold Cross-Validation and 80-20%.

4.1 Performance Metrics

The balanced dataset post-SMOTE application during training, in which synthetic samples have been created to represent minority classes in order to attain a more homogeneous class distribution. This trade-off measure increases the learning capacity of the model on rare vulnerabilities and the performance of its detection across all classes.

Table 3 presents the impact of hyperparameter optimization with the OsHen metaheuristic and VAGA attention on Oshen-VAGA-GGS model, learning rate, attention heads, K steps, dropout and class weight multiplier. The findings show that the best settings, including increased heads and K with moderate dropout, have the highest

accuracy of 0.9867, Node feature dimension is 128, GGNN hidden dimension: 256, GGNN layers: 3, Propagation steps: 8, Attention heads: 4, Head dimension: 64, Dropout: 0.4, Set2Set steps: 4, Batch size: 32 , Learning rate: 0.001, Weight decay: 1×10^{-5}, Optimizer: AdamW (hyperparameters tuned by OsHen), Loss: Focal Loss ($\gamma = 2.0$) which proves the necessity of fine-tuning to obtain a strong vulnerability detection. SMOTE has applied to the 256-D dimensional graph embedding vectors, not to the raw graphs.

Table 3: Impact of hyperparameter tuning using the OsHen metaheuristic on the performance of the Oshen-VAGA-GGS model.

lr	heads	K	dropout	class_weight_mul	final_acc
0.000185	6	4	0.32	1.25	0.9731
0.000245	8	5	0.35	1.3	0.9814
0.00021	7	4	0.33	1.22	0.9785
0.000195	6	5	0.31	1.28	0.9842
0.00023	8	5	0.36	1.35	0.9867

4.2 Comparative Analysis

As Figure 2 demonstrates, the proposed Oshen-VAGA-GGS model is always superior to the already existing models (GNN, Bi-LSTM, MLP+LSTM, BERT-ATT-BiLSTM) when using 70% with accuracy 97.13, 85.6777,82.9023,86.0755 respectively and and 80% of training data with accuracy 98.5696, 84.6452,85.1757,84.4937 respectively.

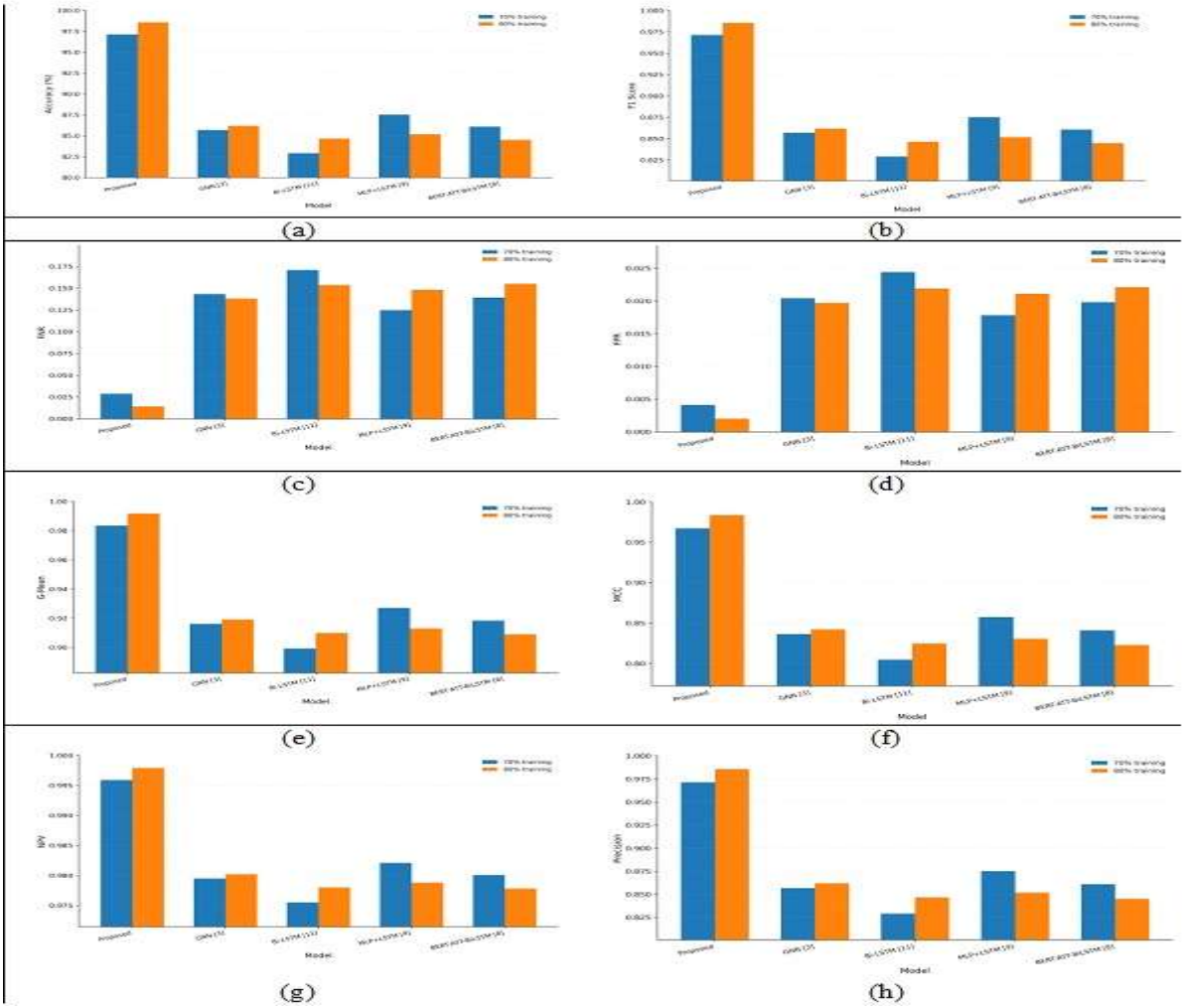

Figure 2: Analysis of performance Metrics (a) Accuracy, (b) Sensitivity, (c) Specificity, (d) Precision, (e) F1-score, (f) FNR, (g) FPR, (h) NPV, (i) MCC, (j) G-Mean

Table shows Folds Cross perfoemance if the novel model with the highest accuracy in fold 5 is 97.19 and Mean± SD 97.13 ± 0.14.

Table 2: Fold Cross-Validation Performance

Fold	Accuracy (%)	Sensitivity	Specificity	Precision	F1 Score	MCC	G-Mean
Fold 1	96.92	0.9690	0.9954	0.9698	0.9694	0.9645	0.9821
Fold 2	97.08	0.9709	0.9958	0.9712	0.9710	0.9668	0.9833
Fold 3	97.31	0.9732	0.9961	0.9735	0.9733	0.9698	0.9846
Fold 4	97.15	0.9716	0.9959	0.9718	0.9717	0.9676	0.9837
Fold 5	97.19	0.9720	0.9960	0.9721	0.9720	0.9681	0.9840
Mean± SD	97.13 ± 0.14	0.9713 ± 0.0016	0.9958 ± 0.0003	0.9717 ± 0.0015	0.9715 ± 0.0015	0.9674 ± 0.0020	0.9835 ± 0.0010

A SHapley Additive exPlanations [33] summary of a prediction made by a model showing the contribution of individual features. Positive contributions (in red) boost the prediction output, whereas negative contributions (in blue) reduce the same. The most positive effect is on feature f 100, and the prediction is severely diminished by f_102 and f_109. There are 122 other features that are also important contributors to the final output.

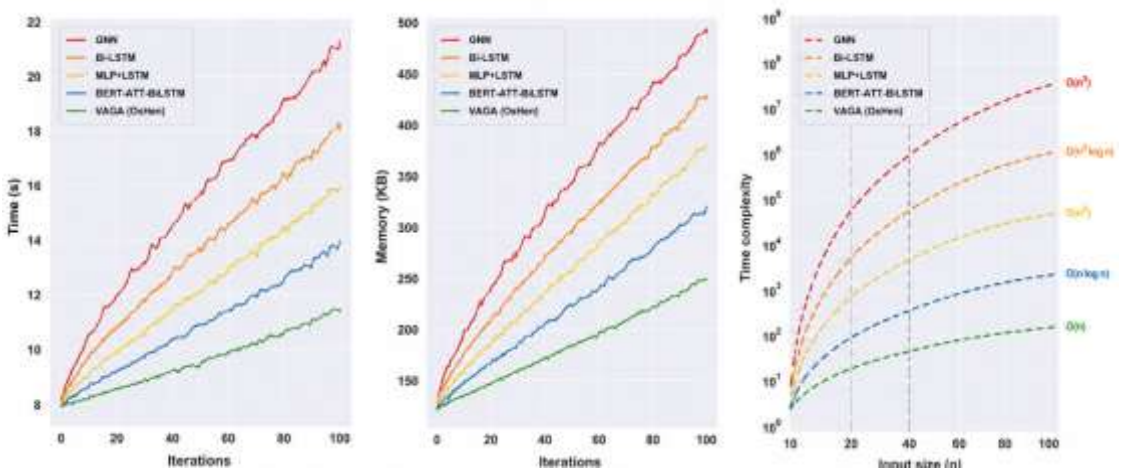

Figure 3: Computational, Memory, and Space Complexity Analysis of the Proposed VAGA Framework.

Figure 5 showing the memory and space analysis of the proposed framework which The ablation experiment in Table 5 determines the Ablation study showing the effects of optimizationand attention stage improves the performance of the smart contract vulnerability detection. GGNN + Osprey, GGNN + Hen, GGNN + OsHen-VAGA has accuracy gain +2.66 pp,+3.06 pp,+3.81 pp respectively.

Table 5: Ablation study showing the effects of optimization stage improves the performance

Configuration	Accuracy Gain vs. GGNN	Precision Gain	Recall Gain	Specificity Gain	F1 Gain
GGNN + Osprey	+2.66 pp	+2.61 pp	+2.70 pp	+2.60 pp	+2.65 pp
GGNN + Hen	+3.06 pp	+3.01 pp	+3.10 pp	+3.00 pp	+3.06 pp
GGNN + OsHen-VAGA	+3.81 pp	+3.71 pp	+3.90 pp	+3.70 pp	+3.80 pp

5. Conclusion

The suggested OsHen-GGS framework is a useful tool to find vulnerabilities in Ethereum smart contracts through combining heterogeneous graph representations (AST, CFG, DFG, and inter-contract call graphs) and a GNN with multi-head graph attention and Set2Set readout. The Oshen-VAGA metaheuristic of hyperparameter optimization and vulnerability aware attention using AdamW and BCE-with-logits and focal loss address class imbalance and guarantee robust learning, respectively. VAGA Uses node features and security semantics, Risk-aware attention weighting, Explicitly models vulnerability indicators (e.g., delegatecall, tx.origin, block.timestamp, arithmetic operations) , Attention influenced by both node risk and edge type unlike standard GAT. Experimental outcomes indicate high accuracy (up to 98.57%), sensitivity, specificity, precision, F1-score, MCC, and G-Mean, as well as very low FNR and FPR, and are superior to baseline models, including GNN, Bi-LSTM, MLP+LSTM, and BERT-ATT-BiLSTM. Although the approach performs well, it

is constrained by reliance on static analysis tools and may be scaled poorly with very large contracts. Future research can be aimed at incorporating the element of dynamic analysis, enhancing the capability of real-time detection, and extending the framework to cross-chain smart contract vulnerability detection to enhance security in decentralized applications further.

Declarations:

Competing Interests: The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper. There are no conflicts of interest associated with this publication, and no external funding sources influenced the study design, data analysis, or manuscript preparation.
Funding Declaration: The authors received no specific funding for this work.
Clinical Trial Registration: This study is not a clinical trial. Therefore, a Clinical Trial Number is not applicable
Consent to Publish: Not applicable.
Consent to Participate: Not applicable.
Ethics Declaration: Not applicable.
Data Availability Statement: The datasets analysed during the current study are available in the:

- Forta Network, Malicious Smart Contract Dataset. Hugging Face Datasets. [Online]. Available: https://huggingface.co/datasets/forta/malicious-smart-contract-dataset. Accessed: Feb. 2026.
- InPlusLab, ReentrancyStudy-Data. GitHub Repository. [Online]. Available: https://github.com/InPlusLab/ReentrancyStudy-Data. Accessed: Feb. 2026.
- S. Chakraborty et al., SCRUBD Dataset. GitHub Repository. [Online]. Available: https://github.com/sujeetc/SCRUBD. Accessed: Feb. 2026.

Author Contributions
Author 1: Mrs. Saltanat Mohammed Abdullah Shaikh conceptualized the research idea, performed the methodology design, implemented the proposed model, conducted experiments, analysed the results, and drafted the manuscript.
Author 2: Dr. Sangeeta Vhatkar supervised the research work, provided critical guidance on methodology and technical aspects, reviewed and edited the manuscript, and contributed to the refinement of the overall study.
Both authors read and approved the final manuscript.

References

[1] C. Xu, H. Xu, L. Zhu, X. Shen, and K. Sharif, "Enhanced Smart Contract Vulnerability Detection via Graph Neural Networks: Achieving High Accuracy and Efficiency," IEEE Transactions on Software Engineering, vol. 51, no. 6, pp. 1854–1865, Jun. 2025, doi: 10.1109/TSE.2025.3570421.
[2] 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," Computer Networks, vol. 242, Art. no. 110238, Apr. 2024, doi: 10.1016/j.comnet.2024.110238.
[3] Y. Wang, X. Zhao, L. He, Z. Zhen, and H. Chen, "ContractGNN: Ethereum Smart Contract Vulnerability Detection Based on Vulnerability Sub-Graphs and Graph Neural Networks," IEEE Transactions on Network Science and Engineering, vol. 11, no. 6, pp. 6382–6395, 2024, doi: 10.1109/TNSE.2024.3470788.
[4] F. Luo et al., "SCVHunter: Smart Contract Vulnerability Detection Based on Heterogeneous Graph Attention Network," in Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), 2024, doi: 10.1145/3597503.3639213.
[5] C. Sendner, R. Zhang, A. Hefter, A. Dmitrienko, and F. Koushanfar, "G-Scan: Graph Neural Networks for Line-Level Vulnerability Identification in Smart Contracts," arXiv preprint arXiv:2307.08549, 2023.
[6] J. Guo, L. Lu, and J. Li, "Smart Contract Vulnerability Detection Based on Multi-Scale Encoders," Electronics, 2024.vol. 13, no. 3, p. 489, Jan. 2024, doi: 10.3390/electronics13030489.
[7] Y. Pang et al., "Graph-Based Contract Sensing Framework for Smart Contract Vulnerability Detection," IEEE Trans. Big Data, vol. 11, no. 6, pp. 3356–3368, Nov./Dec. 2025, doi: 10.1109/TBDATA.2025.3594303.
[8] J. Kim, S. Lee, and H. Kim, "Robust Vulnerability Detection in Solidity-Based Ethereum Smart Contracts Using Fine-Tuned Transformer Encoder Models," IEEE Access, vol. 12, pp. 154700–154717, 2024.
[9] L. S. H. Colin et al., "An Integrated Smart Contract Vulnerability Detection Tool Using Multi-Layer Perceptron on Real-Time Solidity Smart Contracts," IEEE Access, vol. 12, pp. 23549–23567, 2024.
[10] T. Ehsan et al., "Securing Smart Contracts in Fog Computing: Machine Learning-Based Attack Detection for Registration and Resource Access Granting," IEEE Access, vol. 12, pp. 42802–42815, 2024.
[11] S. El Haddouti et al., "Smart Contracts Auditing and Multi-Classification Using Machine Learning Algorithms: An Efficient Vulnerability Detection in Ethereum Blockchain," Computing, vol. 106, no. 9, pp. 2971–3003, Sep. 2024, doi: 10.1007/s00607-024-01314-w.
[12] A. M. Alghamdi et al., "A Graph Attention Network-Based Multi-Agent Reinforcement Learning Framework for Robust Detection of Smart Contract Vulnerabilities," Scientific Reports, vol. 15, 2025.

[13] M. Li, Y. Zhu, Y. Liu, Z. Li, and J. Huang, "Graph Attention Network Vulnerability Detection Model with Global Feature Augmentation for Smart Contracts," Journal of Cloud Computing, vol. 14, no. 1, 2025.
[14] X. Wang et al., "Smart Contract Vulnerability Detection Method Based on Feature Graph and Multiple Attention Mechanisms," Computers, Materials & Continua, vol. 81, no. 3, 2024.
[15] X. Liu, H. Zhang, and Y. Li, "Vulnerability-Hunter: An Adaptive Feature Perception Attention Network for Smart Contract Vulnerabilities," arXiv preprint arXiv:2407.05318, 2024.
[16] X. Zhang, Y. Liu, and Z. Wang, "A Smart Contract Vulnerability Detection Method Based on Heterogeneous Contract Semantic Graphs and Pre-Training Techniques," Electronics, vol. 13, no. 18, 2024.
[17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in International Conference on Learning Representations (ICLR), 2018.
[18] R. Haque, A. Ali, S. McClean, and N. Khan, "Explainable Vulnerability Detection in C/C++ Using Edge-Aware Graph Attention Networks," arXiv preprint arXiv:2507.16540, Jul. 2025, doi: 10.48550/arXiv.2507.16540.
[19] Forta Network, Malicious Smart Contract Dataset. Hugging Face Datasets. [Online]. Available: https://huggingface.co/datasets/forta/malicious-smart-contract-dataset. Accessed: Feb. 2026.
[20] InPlusLab, ReentrancyStudy-Data. GitHub Repository. [Online]. Available: https://github.com/InPlusLab/ReentrancyStudy-Data. Accessed: Feb. 2026.
[21] S. Chakraborty et al., SCRUBD Dataset. GitHub Repository. [Online]. Available: https://github.com/sujeetc/SCRUBD. Accessed: Feb. 2026.
[22] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "SmartBugs: A Framework to Analyze Solidity Smart Contracts," in Proceedings of the IEEE/ACM 35th International Conference on Automated Software Engineering Workshops (ASEW), 2020, pp. 69–79.
[23] A. Vaswani et al., "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), 2017.
[24] M. Dehghani and P. Trojovský, "Osprey Optimization Algorithm: A New Bio-Inspired Metaheuristic Algorithm for Solving Engineering Optimization Problems," Frontiers in Mechanical Engineering, vol. 8, Art. no. 1126450, 2023.
[25] B. Chen et al., "A Comprehensive Survey on the Chicken Swarm Optimization Algorithm and Its Applications: State-of-the-Art and Research Challenges," Artificial Intelligence Review, vol. 57, no. 7, 2024.
[26] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017.
[27] S. M. A. Shaikh and S. Vhatkar, "Raptor Vision-Invasive Hunt Optimization Enabled Light Graph Attention Coupled Gated Graph Sequence Neural Network for Vulnerability Detection," Journal of Computer Virology and Hacking Techniques, vol. 21, no. 1, 2025.
[28] S. Shaikh and S. Vhatkar, "Mathematical Analysis of Existing Techniques for Ethereum Smart Contract Vulnerability Detection," Communications on Applied Nonlinear Analysis, vol. 31, 2024.
[29] K. Cho et al., "Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.
[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," in Proceedings of the International Conference on Learning Representations (ICLR) Workshop, 2013.
[31] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002.
[32] P. Soltanzadeh and M. Hashemzadeh, "RCSMOTE: Range-Controlled Synthetic Minority Over-Sampling Technique for Handling the Class Imbalance Problem," Information Sciences, vol. 542, pp. 92–111, 2021.
[33]S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, 2017, pp. 4768–4777.