MRV-GCN: Multi-Relational Semantic Graph Learning with Explainable Subgraph Fusion for Smart Contract Vulnerability Detection

Yanqin Luo, Gehao Lu*
School of Information Science and Engineering, Yunnan University, Kunming 650500, China
Email: yanqinluo@stu.ynu.edu.cn, glu@ynu.edu.cn
*Corresponding author

***Abstract*—Smart contracts are immutable once deployed, making vulnerabilities prone to severe and irreversible losses. Existing deep-learning–based detectors often operate as black boxes and fail to capture vulnerability-relevant semantics underlying real attack mechanisms. We propose MRV-GCN (Multi-Relational Vulnerability-aware Graph Convolutional Network), a graph-based framework that integrates explicit expert knowledge into deep representation learning for smart contracts. MRV-GCN constructs a multi-relational semantic graph (MRSG) that explicitly models AST structures together with control- and data-flow dependencies, and introduces hierarchical vulnerability semantic features (HVSF) to inject expert-defined cues—such as external calls, state updates, unsafe arithmetic, and delegatecall chains—directly into node representations. An interpretable hybrid readout (R-Hybrid) module further fuses semantic subgraph evidence with non-semantic context to produce robust predictions. Experiments on real-world datasets demonstrate that MRV-GCN achieves strong performance across multiple vulnerability types, while the learned fusion coefficient α provides a lightweight semantics-aware signal for interpretation by quantifying the influence of expert-defined semantic evidence in each prediction.**

***Index Terms*—Smart Contract Vulnerability Detection, Graph Neural Networks, Multi-Relational Semantic Graphs, Explainable AI**

I. Introduction

Blockchain systems enable transparent and tamper-resistant distributed execution through consensus protocols [1] and have been widely adopted in finance, healthcare, IoT, governance, etc. Smart contracts, as self-executing programs, support automated transactions, digital notarization [2], decentralized computation [3], and supply chain traceability [4]. However, their immutability and increasing complexity also introduce significant security risks [5]. High-profile incidents, such as the DAO reentrancy attack causing losses of over $60M [6], the Parity wallet failures [7], and the BEC integer overflow vulnerability [8], highlight the urgent need for accurate smart contract vulnerability detection.

Existing approaches can be broadly categorized into traditional and deep learning–based methods. Traditional techniques include symbolic execution, formal verification, intermediate representation (IR) analysis, and fuzzing. Representative tools such as Oyente [9], Securify [10], Slither [11], SmartCheck [12], and sFuzz [13] exhibit strong interpretability and effectiveness for known vulnerability patterns. However, these methods rely heavily on manually crafted expert rules, limiting their scalability and generalization to complex or evolving vulnerabilities, and often suffering from high false positive and false negative rates.

Deep learning–based approaches leverage automatic feature learning and can be broadly divided into sequence-based and graph-based methods. Sequence-based models, including Vanilla-RNN [14], LSTM [15], and GRU [16], treat smart contract code as token sequences. Although effective at capturing local lexical patterns, such models inevitably lose critical structural information, such as control flow, data dependencies, and cross-function interactions.

Graph-based approaches address this limitation by constructing structured contract representations from source code and learning over them with graph neural networks. Representative methods include DR-GCN and TMP [17], which build normalized contract graphs and apply graph convolution and temporal message propagation to model vulnerability-relevant structural dependencies; AME [18], which enhances graph-based detection by integrating expert-defined vulnerability patterns to improve interpretability; and Peculiar [19], which focuses on crucial data-flow graphs and employs pre-training techniques to strengthen representation generalization. Despite their advantages, existing graph-based models typically employ simplified relational structures and incorporate expert knowledge only at the classification stage, which can dilute vulnerability-critical cues during message propagation and limit the model's ability to provide semantics-aware signals for interpretation.

Recently, large language model (LLM)–based approaches have also been explored for smart contract vulnerability detection, offering an alternative paradigm based on large-scale pre-training. Complementary to these efforts, this work focuses on structure-aware graph learning and enhances it by explicitly embedding expert knowledge into representation learning, enabling effective and interpretable vulnerability detection without relying on large pre-trained models.

Motivated by these considerations, we propose MRV-GCN (Multi-Relational Vulnerability-aware Graph Convolutional Network), which constructs a multi-relational semantic graph (MRSG) integrating abstract syntax with control-flow and data-flow relations. Hierarchical Vulnerability Semantic Fea-

tures (HVSF) are embedded into node representations to encode expert knowledge as semantic priors. A relational GCN (R-GCN) propagates information over heterogeneous relations, and an interpretable hybrid readout mechanism (R-Hybrid) fuses HVSF-enhanced semantic subgraphs with non-semantic background context to produce function-level predictions along with an explicit semantics-aware fusion weight.

We evaluate MRV-GCN on four common smart contract vulnerabilities—reentrancy, timestamp dependency, delegatecall misuse, and integer overflow/underflow—achieving F1-scores of 95.61%, 98.12%, 97.09%, and 97.82%, respectively. Beyond strong detection performance, MRV-GCN provides lightweight interpretability signals: vulnerability-aware semantic subgraphs explicitly participate in the prediction process, and the contribution of expert-defined semantics is quantified through a learnable fusion mechanism.

The main contributions of this work are as follows:

1) We develop MRSG, a multi-relational graph representation that explicitly models abstract syntax, control-flow, and data-flow dependencies for smart contract analysis.
2) We introduce HVSF to encode expert knowledge as explicit semantic priors, enhancing vulnerability awareness and interpretability.
3) We design an interpretable vulnerability-aware GNN framework, MRV-GCN, which integrates heterogeneous relational learning with R-Hybrid for accurate and interpretation-aware vulnerability detection.

II. Method

The overall framework of MRV-GCN is illustrated in Fig. 1 and consists of three stages. First, the smart contract source code is parsed into an MRSG, integrating abstract syntax structures with control-flow and data-flow dependencies. Second, HVSF are extracted from expert-defined patterns and embedded into node features to explicitly emphasize vulnerability-relevant program semantics. Finally, MRV-GCN processes the constructed graph. A relational GCN encoder learns node representations over heterogeneous relations, and R-Hybrid fuses semantic and non-semantic representations to produce the vulnerability prediction $\hat{y}$ and the fusion weight α.

A. MRSG Construction

The abstract syntax tree (AST) is a commonly used representation of smart contract code, as it preserves syntactic and partial semantic information. However, raw ASTs often contain redundant nodes and fail to explicitly capture execution order, control-flow, and data-flow dependencies, which are critical for vulnerability detection. To address these limitations, we construct an MRSG by reorganizing AST information into a vulnerability-aware graph representation, inspired by prior multi-relational graph modeling approaches for smart contracts [20].

We first parse each smart contract function using python-solidity-parser [21], obtaining an initial AST that encodes the function's syntactic structure. To reduce noise and retain only information pertinent to vulnerability analysis, we prune redundant attributes, descriptive nodes, and auxiliary flags.

Despite pruning, the AST still lacks explicit representations of control-flow and data-flow relationships. To remedy this, we augment the AST with heterogeneous relational edges that encode multiple types of dependencies between nodes. Unlike previous methods that represent all dependencies using a single edge type [22], our MRSG explicitly differentiates multiple relational categories. The complete set of edge types is summarized in Table I, where Seq., Op, L/R, Cond, T/F, CF, VF, and FB denote Sequential, Operator, Left/Right, Condition, TrueBody/FalseBody, computeFrom, valueFrom, and fallback relations, respectively. Each relation is constructed by deterministic rules over AST nodes and their structural or semantic dependencies.

TABLE I
Relation types and construction rules in MRSG.

Rel.	Src $\rightarrow$ Tgt	Rule
Seq.	$\text{stmt}_i \rightarrow \text{stmt}_{i+1}$	Execution order
Type	param $\rightarrow$ var	Param/return linkage
Op	expr $\rightarrow$ op	Operator attachment
L/R	expr $\rightarrow$ sub	Expression decomposition
Cond	ctrl $\rightarrow$ cond	Control condition link
T/F	branch $\rightarrow$ body	Branch-body link
CF	RHS $\rightarrow$ LHS	Assignment dependency
VF	def $\rightarrow$ use	Def-use link
FB	call $\rightarrow$ entry	Fallback call link

For data flow modeling, we introduce `computeFrom` edges (right-hand side (RHS) $\rightarrow$ left-hand side (LHS)) and `valueFrom` edges (definition $\rightarrow$ use). A fallback edge connects external call nodes to the contract entry. Both `computeFrom` and `valueFrom` edges are constructed by tracking variable definitions and usages during AST traversal.

By integrating syntactic structure, control flow, data flow, and execution semantics into a unified multi-relational graph, MRSG provides a compact and expressive representation for vulnerability detection with semantics-aware signals.

B. HVSF Embedding

Following prior work on pattern-guided smart contract security analysis, we design HVSF to encode expert knowledge into node-level representations over the MRSG. Each node feature $\mathbf{X}_v$ concatenates a one-hot node-type indicator, a normalized token scalar, and a compact set of binary semantic flags $\mathbf{S}_v$ that capture vulnerability-specific patterns. These flags are organized hierarchically into core nodes, which directly trigger or realize an exploit; important semantic nodes, which mediate or contextualize the vulnerability; and context nodes, which provide auxiliary structural cues.

For each vulnerability type, semantic patterns are aligned with its root cause. Reentrancy arises when an external call occurs before a critical state update; HVSF marks value-carrying external calls and persistent state-modification operations as core nodes. Timestamp dependence stems from miner-controllable block timestamps influencing execution paths;

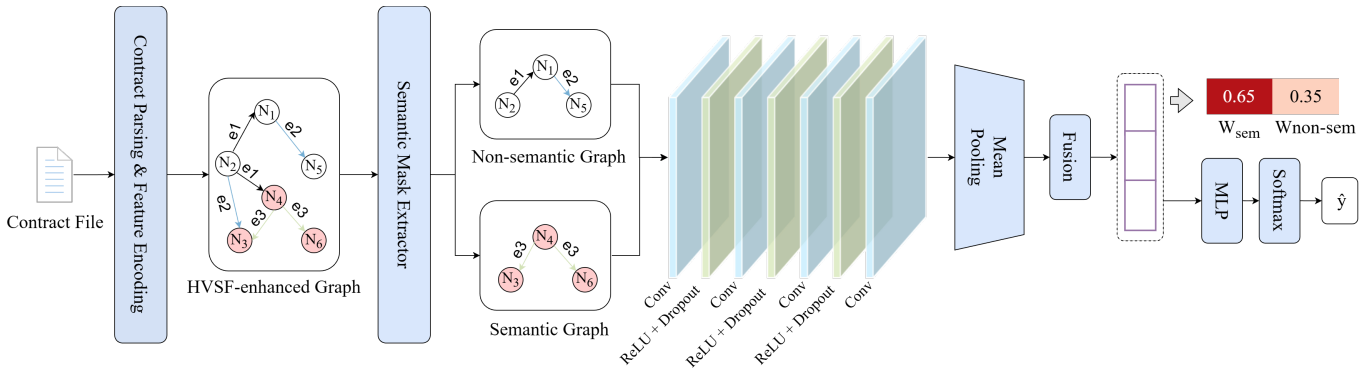


Fig. 1. The framework of our proposed method. The semantic mask is computed from HVSF but applied after R-GCN at the pooling/fusion stage.

HVSF emphasizes direct and indirect timestamp dependencies within conditional gates. Delegatecall vulnerabilities result from executing untrusted logic in the caller's storage context, captured by invocation chains and storage writes. Integer overflow/underflow occurs due to unbounded arithmetic on vulnerable types, with HVSF encoding arithmetic operations, type propagation, and mitigation signals (e.g., SafeMath usage and checked or unchecked arithmetic).

HVSF translates these patterns into compact node-level feature vectors that guide the GNN to focus on causal vulnerability paths. To ensure robustness under identifier normalization or obfuscation, the mechanism leverages AST structural patterns, MemberAccess chains, FunctionCall ancestry, and edge-type–aware positional indices. Formally, the node feature vector is defined as:

$$\mathbf{X}_v = \mathbf{X}_v^{\text{AST}} \oplus \mathbf{X}_v^{\text{Token}} \oplus \mathbf{S}_v, \qquad \mathbf{X}v \in \mathbb{R}^{|\mathcal{T}|+1+d\text{sem}} \tag{1}$$

where $\mathbf{X}_v^{\text{AST}}$ is the one-hot node-type indicator, $\mathbf{X}_v^{\text{Token}}$ is the normalized token scalar, and $\mathbf{S}v$ is the HVSF vector. Here, $\mathcal{T}$ denotes the set of AST node types, and $d\text{sem}$ is the HVSF dimensionality, providing a lightweight yet semantically rich representation for vulnerability-focused GNN learning.

C. MRV-GCN Framework

MRV-GCN integrates HVSF-enhanced node features with relational graph learning. Node embeddings are updated through L layers of R-GCN:

$$\mathbf{h}_i^{(\ell+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in\mathcal{N}_r(i)}\frac{1}{c_{i,r}}\mathbf{W}_r^{(\ell)}\mathbf{h}_j^{(\ell)} + \mathbf{W}_0^{(\ell)}\mathbf{h}_i^{(\ell)}\right) \tag{2}$$

where $\mathcal{N}_r(i)$ is the set of neighbors of node i under relation r, $c_{i,r}$ is a normalization constant, and $\sigma(\cdot)$ denotes ReLU.

Using HVSF, we define a binary semantic mask:

$$m_v = \mathbb{I}\Big(\sum \mathbf{S}_v > 0\Big) \tag{3}$$

where $m_v = 1$ indicates that node v participates in at least one expert-defined vulnerability pattern.

Node embeddings are aggregated into semantic and non-semantic views:

$$g_{\text{sem}} = \frac{1}{\sum m_v + \epsilon}\sum_{v\in V} m_v \cdot \mathbf{h}_v^{(L)} \tag{4}$$

$$g_{\text{non-sem}} = \frac{1}{\sum(1-m_v) + \epsilon}\sum_{v\in V}(1-m_v) \cdot \mathbf{h}_v^{(L)} \tag{5}$$

where g_{sem} captures embeddings of semantic nodes, and $g_{\text{non-sem}}$ aggregates the remaining background nodes.

These views are fused via the learnable scalar in R-Hybrid:

$$g_{\text{final}} = \alpha \cdot g_{\text{sem}} + (1-\alpha) \cdot g_{\text{non-sem}} \tag{6}$$

where α quantifies the contribution of semantic nodes. The final embedding is fed into a two-layer MLP classifier and optimized with cross-entropy loss, producing interpretable vulnerability predictions.

III. Experiments

In this section, we conduct an extensive evaluation of our proposed framework. We aim to address the following research questions:

- **RQ1:** How effectively does the proposed method detect the four types of vulnerabilities? How does it compare with existing techniques in terms of accuracy, precision, recall, and F1-score?
- **RQ2:** Is the method able to provide interpretability in vulnerability detection? Can it yield new insights?
- **RQ3:** How do the various modules contribute to the overall detection performance?

A. Experimental Setup

Datasets. (1) Qian et al. [22] released a benchmark dataset covering four representative smart contract vulnerability types: reentrancy, timestamp dependence, integer overflow/underflow, and delegatecall misuse. The dataset was built by collecting contracts from three major sources—the Ethereum mainnet (accounting for over 96% of samples), GitHub repositories, and security-analysis blog posts. (2) Yu et al. [23] further expanded this benchmark by incorporating

TABLE II
PERFORMANCE COMPARISON. A TOTAL OF TWELVE METHODS ARE INVESTIGATED. "–" DENOTES NOT APPLICABLE.

Methods	Reentrancy				Timestamp				Delegatecall				IntegerOverflow/Underflow			
	Acc(%)	Pre(%)	Rec(%)	F1(%)	Acc(%)	Pre(%)	Rec(%)	F1(%)	Acc(%)	Pre(%)	Rec(%)	F1(%)	Acc(%)	Pre(%)	Rec(%)	F1(%)
Smartcheck	35.04	29.25	86.88	44.93	51.54	50.00	45.12	48.52	42.27	29.28	77.00	42.42	34.16	22.88	63.32	33.61
Oyente	60.92	37.12	53.07	45.72	53.42	52.28	60.68	59.43	–	–	–	–	70.63	59.67	57.68	58.29
Securify	70.91	68.58	74.50	70.10	–	–	–	–	–	–	–	–	–	–	–	–
Slither	74.33	74.58	73.90	73.39	66.29	67.14	66.28	66.03	62.75	74.11	64.95	69.51	–	–	–	–
sFuzz	48.45	10.86	15.50	12.88	16.71	31.64	26.33	29.87	67.00	38.96	65.27	47.37	59.71	53.67	46.46	49.34
VanillaRNN	45.45	58.71	58.33	45.44	48.57	24.29	50.00	32.69	51.50	48.60	50.57	49.55	67.27	33.64	50.00	40.22
LSTM	54.55	60.50	62.50	54.00	51.43	25.71	50.00	33.96	55.00	50.29	56.14	53.82	69.09	67.95	54.20	49.97
GRU	70.91	36.11	48.75	41.49	58.57	66.76	57.52	51.33	59.50	61.58	57.37	59.94	72.73	68.89	68.32	68.57
DR-GCN	78.18	57.14	80.00	66.67	69.57	81.82	51.43	63.16	65.76	66.01	66.74	66.37	76.90	77.13	75.58	76.32
TMP	76.45	66.67	75.30	72.67	76.81	74.36	82.86	78.38	57.14	72.81	67.45	70.22	82.31	86.34	81.35	83.77
AME	86.23	81.08	86.96	83.92	89.14	82.58	89.96	85.02	–	–	–	–	–	–	–	–
Peculiar	83.40	84.89	74.39	78.12	70.18	67.29	71.19	69.29	70.17	70.98	60.67	65.79	87.70	88.62	87.03	87.79
Ours	97.66	97.56	93.85	95.61	97.74	97.96	98.29	98.12	98.36	96.39	98.00	97.09	98.71	97.51	98.66	97.82

additional samples across all four vulnerability categories, with samples dated between 2020 and 2024.We merged the two datasets, removed duplicates, and discarded samples that failed parsing [20], resulting in a high-quality function-level dataset of 3,130 samples, including 1,970 non-vulnerable and 1,160 vulnerable instances.

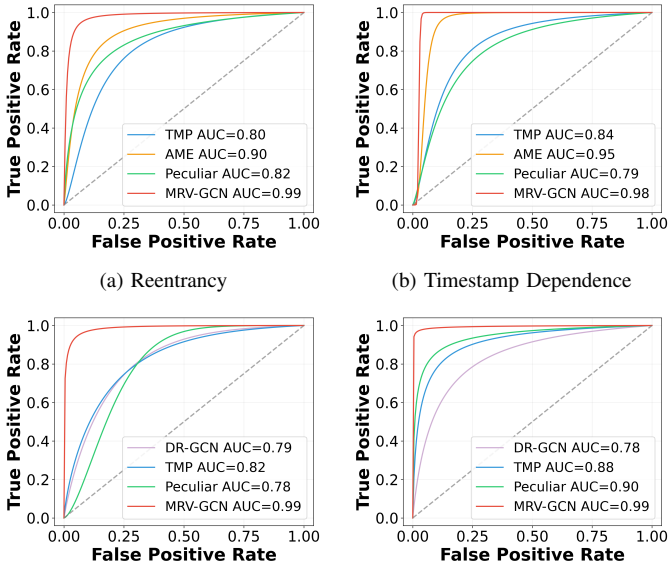
Implementation details. Experiments are conducted on Ubuntu 22.04 with an Intel Xeon Silver 4310 CPU, 64 GB RAM, and an NVIDIA A10 GPU (24 GB VRAM). Models are implemented in PyTorch Geometric with CUDA 11.1. Following prior work, we adopt an 80%/20% train–test split at the function level. As functions from the same contract may appear in both splits, potential data leakage cannot be fully ruled out under this setting. We therefore interpret the results with caution and leave contract-level or temporal splits for future work. Each experiment is repeated 30 times with different seeds, and average results are reported. The R-GCN encoder uses four layers with hidden size 96 and 15 relation bases.

B. Performance Comparison (RQ1)

In this section, we compare the proposed approach with representative state-of-the-art tools and neural network–based methods. LLM-based approaches are not included in the primary comparison, as they typically rely on substantially different training resources, input granularity, and evaluation protocols, which makes controlled and fair comparison difficult in our setting. Instead, our focus is on methods operating on comparable graph-based representations. LLM-based techniques are orthogonal and potentially complementary, and a systematic comparison is left for future investigation. Accuracy, precision, recall, and F1-score are reported for evaluation.

1) Comparison with Conventional Detection Tools: We first compare our method MRV-GCN with existing smart contract vulnerability detection tools including Smartcheck [12], Oyente [9], Securify [10], Slither [11] and sFuzz [13]. Quantitative results are summarized in Table II.

We first evaluate MRV-GCN on reentrancy and timestamp-dependence vulnerabilities, where conventional tools still struggle to achieve satisfactory accuracy. For reentrancy detection, state-of-the-art static analyzers such as Securify and Slither only reach 70.91% and 74.33% accuracy, respectively, whereas MRV-GCN achieves 97.66%, yielding a 23.33% absolute improvement over the strongest baseline. A similar trend is observed for timestamp dependence: Slither attains only 66.29% accuracy, largely due to its rule-based detection strategy that ignores whether `block.timestamp` affects critical operations. In contrast, MRV-GCN consistently delivers the best performance across all four metrics, improving accuracy by 31.45% over the best traditional tool.



(a) Reentrancy (b) Timestamp Dependence

(c) Delegatecall (d) Integer Overflow/Underflow

Fig. 2. ROC curves comparing MRV-GCN with the top three baselines across four vulnerability detection tasks.

We further evaluate MRV-GCN on delegatecall and integer overflow/underflow vulnerabilities. MRV-GCN achieves 98.36% accuracy on delegatecall detection, surpassing the best traditional baseline sFuzz (67.00%) by 31.36%. For integer overflow/underflow, MRV-GCN reaches 98.71% accuracy, significantly outperforming Oyente (70.63%). Overall, MRV-GCN consistently outperforms existing tools by a large margin across all evaluated vulnerability categories.

TABLE III
COMPARISON OF PERFORMANCE BETWEEN OUR METHOD AND ITS VARIANTS

Methods	Reentrancy				Timestamp				Delegatecall				IntegerOverflow/Underflow			
	Acc(%)	Pre(%)	Rec(%)	F1(%)	Acc(%)	Pre(%)	Rec(%)	F1(%)	Acc(%)	Pre(%)	Rec(%)	F1(%)	Acc(%)	Pre(%)	Rec(%)	F1(%)
MRV-GCN (g_sem)	82.55	64.84	82.31	72.16	71.69	85.46	63.76	72.98	93.42	95.23	80.00	86.73	87.97	71.67	91.14	79.99
MRV-GCN (g_non-sem)	92.34	83.23	90.77	86.63	86.67	85.82	87.29	86.40	95.89	95.87	87.00	92.01	93.16	95.43	81.78	88.03
MRV-GCN (w/o HVSF)	96.38	94.61	92.31	93.27	96.82	96.61	96.12	95.35	97.26	94.99	95.00	94.94	96.95	95.59	96.05	96.01
MRV-GCN (GCN-only)	95.32	95.93	86.92	91.02	94.36	93.22	97.95	95.46	95.07	91.93	91.00	91.14	94.96	85.70	97.71	91.16
MRV-GCN (dynamic α)	97.02	94.86	94.62	94.62	96.90	97.41	96.58	97.02	97.26	92.72	98.00	95.19	98.05	96.00	97.43	96.43
MRV-GCN (static α)	97.66	97.56	93.85	95.61	97.74	97.96	98.29	98.12	98.36	96.39	98.00	97.09	98.71	97.51	98.66	97.82

The superior performance of MRV-GCN can be attributed to its ability to overcome the inherent limitations of traditional vulnerability detection tools, which rely on a small set of fixed expert rules (e.g., detecting reentrancy solely via `call.value`) and fail to capture rich semantic dependencies in contract code. By explicitly modeling key variables and complex control-flow and data-flow semantics through the constructed MRSG, MRV-GCN achieves more precise and robust vulnerability detection.

2) Comparison with Deep Learning Methods: We further compare our method with available deep learning based methods, namely Vanilla-RNN [14], LSTM [15], GRU [16], DR-GCN [17], TMP [17], AME [18] and Peculiar [19]. For a feasible comparison, Vanilla-RNN, LSTM, and GRU take function-level code sequence embeddings as input, whereas DR-GCN, TMP, AME, Peculiar, and our MRV-GCN operate on graph-based feature representations. Since AME's expert-pattern extraction was not developed for delegatecall or integer overflow/underflow vulnerabilities, we evaluate it only on reentrancy and timestamp dependency vulnerability detection. We present the results in Table II.

Across all four vulnerability types, GRU represents the best-performing sequence-based model. Our MRV-GCN consistently improves detection accuracy over GRU, with increases of 26.75%, 39.17%, 38.86%, and 25.98% for reentrancy, timestamp-dependence, delegatecall, and integer overflow/underflow, respectively. When compared with GNN-based baselines, MRV-GCN surpasses the strongest model—AME for reentrancy and timestamp-dependence, and Peculiar for delegatecall and integer overflow/underflow—by 11.43%, 8.6%, 28.19%, and 11.01% in accuracy on the corresponding tasks. These improvements result from our MRSG construction and the incorporation of HVSF, which capture finer-grained semantics than homogeneous graph representations. Notably, all five GNN-based models consistently outperform the three sequence-based models across all tasks. This observation suggests that treating smart contracts purely as sequential text is suboptimal, as it discards essential structural properties—such as data-flow and control-flow dependencies—leading to the loss of vulnerability-critical semantics. In contrast, modeling contracts as structured graphs and applying GNNs demonstrates greater effectiveness for vulnerability detection.

To further evaluate the models, we plot Receiver Operating Characteristic (ROC) curves, which show the true positive rate (TPR) against the false positive rate (FPR) under varying decision thresholds. ROC is robust to class imbalance, making it a reliable metric even when vulnerable and non-vulnerable samples are unevenly distributed. We also compute the Area Under the Curve (AUC) to quantify each classifier's ability to discriminate between vulnerable and non-vulnerable samples.

The ROC curves and AUC values for all models on the four vulnerability localization tasks—reentrancy, timestamp dependence, delegatecall, and integer overflow/underflow—are shown in Fig. 2. MRV-GCN achieves AUC scores of 0.99, 0.98, 0.99, and 0.99 on these tasks, respectively, outperforming all compared approaches.

C. Interpretability Evaluation (RQ2)

R-Hybrid provides a lightweight semantics-aware signal by quantifying the model's reliance on expert-defined semantic evidence via a learnable fusion weight α (denoted as W_{sem}), which balances semantic and non-semantic representations.

We analyze the average fusion weights across detected functions for different vulnerability types (Fig. 3). The semantic weight remains consistently high (0.65–0.73), with Integer Overflow/Underflow reaching 0.73, indicating substantial reliance on localized semantic evidence.

The fusion weight reflects the relative contribution of each view in the final prediction rather than their standalone performance, which is evaluated in RQ3. These results indicate that MRV-GCN leverages semantic subgraphs as key evidence, focusing on causality-relevant regions identified by HVSF while suppressing irrelevant context. This provides an interpretable alternative to black-box pooling or post-hoc explanation methods.

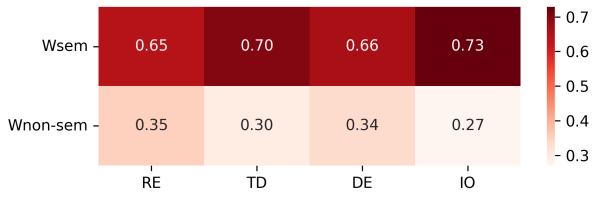


Fig. 3. The interpretability of our method.

Case study. In a typical reentrancy example, nodes corresponding to the external call and subsequent state update are identified as semantic nodes. A high α indicates that the prediction is primarily driven by vulnerability-relevant evidence, consistent with the underlying cause of reentrancy.

D. Ablation study (RQ3)

To analyze the contribution of each component in MRV-GCN, we compare the full model with five controlled variants. Table III reports results on reentrancy, timestamp dependence, delegatecall, and integer overflow/underflow. The full model consistently achieves the best performance.

Each variant modifies a single component: (i) semantic-only ($\mathbf{g}_{\text{sem}}$); (ii) non-semantic-only ($\mathbf{g}_{\text{non-sem}}$); (iii) w/o HVSF; (iv) replacing R-GCN with GCN; (v) dynamic gating instead of static fusion. All other settings remain unchanged.

For the R-Hybrid readout, single-view variants exhibit clear performance degradation, indicating that neither semantic nor contextual information alone is sufficient. The relatively stronger performance of the non-semantic variant suggests that structural context provides substantial predictive signals, while the semantic subgraph contributes complementary, vulnerability-focused information.

Static fusion consistently outperforms dynamic gating (e.g., +1.90% on Delegatecall and +1.39% on Integer Overflow/Underflow in F1), indicating that a globally shared α yields more stable representations. In contrast, sample-adaptive gating introduces additional flexibility, which may be less effective under limited data.

Removing HVSF results in consistent performance degradation (average F1 decrease of 2.27 points, including 2.77% on Timestamp Dependence). This moderate drop indicates that MRSG captures strong structural dependencies, while HVSF provides additional vulnerability-oriented guidance and enhances robustness.

Replacing R-GCN with GCN leads to substantial degradation across all tasks (average F1 decrease of 4.98 points, up to 5.95% on Delegatecall), highlighting the importance of modeling heterogeneous relations, including AST, control-flow, and data-flow edges.

Overall, the results demonstrate that HVSF, multi-relational encoding, and the R-Hybrid readout each contribute to performance gains, and their combination enables effective modeling of vulnerability-relevant semantics and structural context.

IV. Conclusion

We present MRV-GCN, an explainable framework for smart contract vulnerability detection that integrates GNNs with expert knowledge. By constructing a Multi-Relational Semantic Graph and embedding Hierarchical Vulnerability Semantic Features, MRV-GCN incorporates security-aware priors into representation learning. The Hybrid Readout mechanism captures vulnerability-relevant evidence from vulnerability-relevant subgraphs and quantifies expert contributions via a learnable fusion weight. Experimental results demonstrate that MRV-GCN consistently achieves strong performance across multiple vulnerability types, outperforming a range of representative baselines.

Limitations include the restricted coverage of vulnerability types and the lack of evaluation on contract-level and temporal generalization. Future work will extend the framework to broader vulnerability categories and investigate integration with large-scale pre-trained code models.

References

[1] X. Fu, H. Wang, and P. Shi, "A survey of blockchain consensus algorithms: Mechanism, design and applications," *Sci. China Inf. Sci.*, vol. 64, no. 2, p. 121101, 2021.

[2] R. Arenas and P. Fernandez, "CredenceLedger: A permissioned blockchain for verifiable academic credentials," in *Proc. IEEE Int. Conf. Eng., Technol. Innov. (ICE/ITMC)*, Stuttgart, Germany, 2018, pp. 1–6.

[3] J. Teutsch and C. Reitwießner, "A scalable verification solution for blockchains," in *Aspects of Computation and Automata Theory with Applications*, Singapore: World Scientific, 2024, pp. 377–424.

[4] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, "Blockchain technology and its relationships to sustainable supply chain management," *Int. J. Prod. Res.*, vol. 57, no. 7, pp. 2117–2135, 2019.

[5] W. Zou, D. Lo, P. S. Kochhar, X. B. D. Le, X. Xia, Y. Feng, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Trans. Softw. Eng.*, vol. 47, no. 10, pp. 2084–2106, 2021.

[6] M. I. Mehar *et al.*, "Understanding a revolutionary and flawed grand experiment in blockchain: The DAO attack," *J. Cases Inf. Technol.*, vol. 21, no. 1, pp. 19–32, 2019.

[7] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proc. 34th Annu. Comput. Security Appl. Conf. (ACSAC)*, San Juan, PR, USA, 2018, pp. 653–663.

[8] NIST, "CVE-2018-10299: Integer overflow in the batchTransfer function of the BEC contract," Nat. Vulnerability Database, 2018. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-10299

[9] L. Luu, D. H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Vienna, Austria, 2016, pp. 254–269.

[10] P. Tsankov *et al.*, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Toronto, ON, Canada, 2018, pp. 67–82.

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

[12] S. Tikhomirov *et al.*, "SmartCheck: Static analysis of Ethereum smart contracts," in *Proc. Int. Workshop Emerging Trends Softw. Eng. Blockchain*, 2018, pp. 9–16.

[13] T. D. Nguyen *et al.*, "sFuzz: An efficient adaptive fuzzer for Solidity smart contracts," in *Proc. ACM/IEEE Int. Conf. Softw. Eng. (ICSE)*, 2020, pp. 778–788.

[14] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Proc. Int. Conf. Neural Netw.*, 1996, pp. 347–352.

[15] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. Interspeech*, 2014, pp. 338–342.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," arXiv:1412.3555, 2014.

[17] Y. Zhuang *et al.*, "Smart contract vulnerability detection using graph neural networks," in *Proc. IJCAI*, 2021, pp. 3283–3290.

[18] Z. Liu *et al.*, "Smart contract vulnerability detection: From pure neural networks to interpretable graph feature and expert pattern fusion," in *Proc. IJCAI*, 2021, pp. 3971–3978.

[19] H. Wu *et al.*, "Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques," in *Proc. IEEE ISSRE*, 2021, pp. 378–389.

[20] H. Liu, Y. Fan, L. Feng, and Z. Wei, "Vulnerable smart contract function locating based on multi-relational nested graph convolutional network," *J. Syst. Softw.*, vol. 204, p. 111775, 2023.

[21] ConsenSys, "python-solidity-parser," 2021. [Online]. Available: https://github.com/ConsenSys/python-solidity-parser

[22] S. Cao *et al.*, "BGNN4VD: Constructing bidirectional graph neural networks for vulnerability detection," *Inf. Softw. Technol.*, vol. 136, p. 106576, 2021.

[23] P. Qian *et al.*, "Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode," in *Proc. ACM Web Conf.*, 2023, pp. 2220–2229.

[24] L. Yu *et al.*, "Smart-LLaMA-DPO: Reinforced large language model for explainable smart contract vulnerability detection," *Proc. ACM Softw. Eng.*, vol. 2, no. ISSTA, pp. 182–205, 2025.