Neurocomputing 695 (2026) 134050

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

RGCNet: Riemannian graph convolutional networks for end-to-end smart contract vulnerability detection

Yaoxin Chen [a,c,1], Haiming Zhu [b,c,1], Haibo Li [b,c], Yaming Yang [b,c], Qicong Wang [b,c,*], Maozhen Li [d,e,**]

[a] Institute of Artificial Intelligence, Xiamen University, Xiamen 361000, China
[b] Department of Computer Science and Technology, School of Informatics, Xiamen University, Xiamen 361000, China
[c] Shenzhen Research Institute, Xiamen University, Shenzhen 518000, China
[d] School of Computer Science, Shandong Xiehe University, Jinan 250109, China
[e] Department of Engineering, Brunel University of London, Uxbridge UB8 3PH, UK

HIGHLIGHTS

- We propose a novel Riemannian Graph Convolutional Network (RGCNet).
- Based on RGCNet, we design an effective end-to-end vulnerability detection network.
- Superior performance on three smart contract vulnerabilities.

ARTICLE INFO

Communicated by N. Zeng

Keywords:
Smart contract
Vulnerability detection
Blockchain
Riemannian graph convolutional networks

ABSTRACT

Frequent security issues with smart contract vulnerabilities have become a pressing challenge in the industry. Conventional program analysis methods lack flexibility and extensibility, leading to high false positive rates. Deep learning approaches are emerging as a new trend to address this issue. Compared to other neural networks, graph convolutional networks can better capture the structural and logical information of smart contracts. However, existing methods do not fully consider the scale-free characteristics of smart contracts and fail to leverage their complex hierarchical structures and semantic information. Therefore, we develop an end-to-end vulnerability detection framework using Riemannian Graph Convolutional Networks (RGCNet). We first construct smart contract graphs that are rich in semantic and structural information. Next, we learn features of the smart contract graph in the Riemannian manifold, thereby better reflecting its actual topology. Simultaneously, the word embedding network extracts semantic features, forming an end-to-end network where modules promote one another. Extensive experiments are conducted on three vulnerabilities using real-world smart contracts. The results show that the proposed approach exhibits superior performance over state-of-the-art methodologies in terms of accuracy, precision, and recall.

1. Introduction

In recent years, the rapid advancement of artificial intelligence [1–7] and deep learning technologies [8–16] has revolutionized various fields. In particular, the cybersecurity domain has garnered immense attention as these advancements redefine modern threat detection and defense strategies. This technological evolution has not only transformed traditional software development but also catalyzed the emergence of decentralized paradigms, most notably blockchain technology. Introduced by Nakamoto [17], blockchain serves as a decentralized, tamper-resistant, and traceable distributed ledger. Operating at the core of this ecosystem are smart contracts, first proposed by Szabo [18], which are self-executing programs designed to automate the deployment and enforcement of contractual agreements. Based on predefined rules, they execute automatically without relying on trusted

* Corresponding author at: Department of Computer Science and Technology, School of Informatics, Xiamen University, Xiamen 361000, China.
** Corresponding author at: School of Computer Science, Shandong Xiehe University, Jinan 250109, China.
Email addresses: qcwang@xmu.edu.cn (Q. Wang), maozhen.li@brunel.ac.uk (M. Li).

[1] Co-first authors.

https://doi.org/10.1016/j.neucom.2026.134050
Received 27 March 2026; Received in revised form 20 April 2026; Accepted 20 May 2026
Available online 21 May 2026

intermediaries. Today, millions of smart contracts run on platforms like Ethereum [19], supporting a broad range of intelligent and decentralized applications [20,21].

However, smart contract security and system robustness have attracted significant attention. According to SlowMist [22], security vulnerabilities have resulted in losses exceeding $32 billion. High-profile incidents like the DAO [23] and Poly Network [24] exploits not only caused substantial financial losses but also undermined public trust in blockchain systems.

Vulnerabilities in smart contracts typically arise from implementation or design flaws. The transparency of public blockchains allows anyone to examine contract code, increasing the likelihood of attacks. Moreover, since smart contracts often manage substantial financial assets and are difficult to modify, early vulnerability detection is particularly critical.

Conventional program analysis techniques, including symbolic execution, formal verification, and taint analysis, form the foundation of many vulnerability detection tools [25–28]. Despite their effectiveness, these approaches rely heavily on expert-crafted rules, which limits their adaptability, scalability, and detection accuracy.

Recently, deep learning has shown promising results in vulnerability detection [29] by enabling automatic feature learning. However, sequence-based [30] and image-based models [31,32] often fail to capture the rich structural information in contract code. Since smart contract logic naturally forms graphs, graph-based approaches [33–35] are better suited to exploit structural and semantic relationships, thereby improving detection accuracy.

Nevertheless, GNN-based methods [36] still face several challenges. Smart contract graphs are often scale-free and hierarchical, embedding them in Euclidean space can introduce significant geometric distortions [37,38]. Furthermore, many existing studies fail to fully exploit joint semantic and structural representations, and label noise further degrades model generalization.

To address these challenges, we propose an end-to-end RGCNet for smart contract vulnerability detection. By learning representations in negatively curved manifolds [39], RGCNet effectively captures the hierarchical and scale-free graph structures inherent in smart contracts. The framework integrates Word2Vec [40] to enable a deep fusion of semantic and structural information. Furthermore, we introduce an annealing mixup contrastive learning strategy [41] to generate informative positive and hard negative samples, significantly improving model robustness against label noise.

In summary, our contributions are as follows:

- We propose a novel graph convolutional network to explore deep hierarchical and scale-free smart contract graphs in the Riemannian manifold.
- We design an end-to-end vulnerability detection network integrating word embeddings with RGCNet, where the front-end Abstract Syntax Tree (AST) graph and back-end RGCNet mutually enhance, yielding more discriminative graph embeddings.
- We conduct extensive experiments on reentrancy, arithmetic, and timestamp dependencies, outperforming state-of-the-art methods with accuracies of 93.13%, 92.41%, and 92.88% respectively.

2. Related work

2.1. Conventional smart contract vulnerability detection

Traditional vulnerability detection has long been dominated by program-analysis techniques. For instance, symbolic execution [25,42] explores feasible execution paths via constraint solvers, while formal verification [26,43] employs mathematical proofs to verify security properties. Fuzzing-based methods [44] monitor runtime behaviors under diverse inputs, and static analysis tools like Slither [28] leverage taint analysis to pinpoint vulnerable data-flow patterns.

Despite their technical rigor, the effectiveness of these conventional approaches heavily depends on manually defined vulnerability patterns and expert-crafted heuristics. Such reliance not only increases analysis costs but also introduces human-induced errors and limits scalability as smart contracts grow in volume and complexity.

2.2. Deep learning-based smart contract vulnerability detection

Deep learning approaches for smart contract vulnerability detection are typically categorized into text-based, image-based, and graph-based methods.

Text-based methods treat code as token sequences, applying NLP models to learn semantic patterns. For example, SaferSC [30] models opcode sequences with LSTMs [45], while SmartEmbed [46] matches AST-derived sequences against vulnerable contracts. Although subsequent studies improve sequence modeling and optimization, structural dependencies remain insufficiently captured.

Image-based methods convert contracts into matrix representations for CNNs [47]. Slice Matrices [31] organize opcode features, whereas CodeNet [32] transforms bytecode into RGB images using dilated convolutions. To address missing semantics and noise in bytecode, recent approaches like MTVHunter [48] employ multi-teacher knowledge translation to distill source-code semantics into bytecode embeddings. Despite strong empirical performance, these methods often miss fine-grained structural cues inherent in the original code.

Graph-based methods address these limitations using GNNs on code graphs. TMP [33] exploits control-flow graphs, CGE [34] integrates expert rules, and Peculiar [35] extracts data-flow diagrams from ASTs. Others build semantic graphs via Word2Vec and residual GCNs [49], or adopt dual-attention architectures [50]. By capturing both structural and semantic information, these models achieve higher detection accuracy [51,52]. More recently, Clear [53] further improves detection performance by capturing fine-grained correlations between contracts via contrastive learning. However, the hierarchical and scale-free nature of smart contract graphs renders conventional Euclidean representations fundamentally limited.

2.3. Large language models for vulnerability detection

The rapid advancement of Large Language Models (LLMs) has introduced a new paradigm for smart contract analysis. Recent works like EVuLLM [54] and SAEL [55] leverage the extensive pre-training of LLMs to capture complex security patterns. Furthermore, Smart-LLaMA-DPO [56] integrates Direct Preference Optimization (DPO) to enhance the explainability and accuracy of vulnerability detection. Despite their potential, LLMs often require substantial computational resources and may still struggle with precise structural reasoning compared to specialized graph-based models.

2.4. Non-Euclidean graph representation

Standard Euclidean embeddings often fail to accurately represent graphs characterized by intricate, multi-level hierarchies. This geometric limitation has spurred the development of non-Euclidean representation techniques. For instance, Poincaré embeddings [57] leverage hyperbolic geometry to preserve latent tree-like structures, while hyperbolic GNNs [58] and convolutional networks [59] generalize neighborhood aggregation to curved manifolds. By exploiting the exponential expansion of volume in negatively curved spaces, these approaches provide a more natural fit for the high-order relational patterns and structural hierarchies often found in complex networks.

3. Methodology

The main symbols used in this paper and their definitions are shown in Table 1.

Table 1
Symbol Definitions.

Symbols	Definitions	Symbols	Definitions
$\mathscr{M}$	Riemannian manifold	$P_{x\to y}(\cdot)$	Parallel transport from point x to y
$\mathscr{E}$	Euclidean space	Beta$(\cdot,\cdot)$	Beta distribution
γ	A curve in the Riemannian manifold	t	Ground-truth label of the smart contract
p, q	Two points in the Riemannian manifold	$\hat{t}$	Predicted label of the smart contract
v	A vector in the Euclidean space	ϑ	Mixing coefficient for mixup
$dis(p,q)$	Geodesic length between points p and q	ϕ	Label smoothing function
$\mathscr{T}_p\mathscr{M}$	Tangent space at p in the Riemannian manifold	ϵ	Small constant for label smoothing
c	Curvature of the Riemannian manifold	$g(\cdot)$	Progressive annealing function
$\langle\cdot,\cdot\rangle_L$	Minkowski inner product	κ	Scaling rate in the annealing function
X	Feature matrix of the smart contract graph	$epoch$	Epoch index during training
$X^{\mathscr{M}}$	Feature matrix in the Riemannian manifold	μ	Adaptive weight for negative samples
$X^{\mathscr{E}}$	Feature matrix in the Euclidean space	$\mathscr{X}$	Mixup sample representation
N	Number of nodes in the smart contract graph	Q	Index of the query sample
C	Dimensionality of node features	$\mathscr{X}_Q$	Query sample representation
o	Origin of the manifold	$\Gamma(Q)$	Positive sample set for the query
G	Attention weight matrix	δ	Distance threshold for positive samples
$Nbr(\cdot)$	Neighborhood node set	$\mathscr{X}_Q^-$	Composite negative sample for the query
ρ	Attention coefficient	M	Mini-batch size
dtc_L^2	Squared Lorentz distance	T	Total number of words in the contract
D	Degree matrix of the attention weights	L_w	Word embedding loss
W	Weight matrix for feature transformation	L_{cf}	Classification loss
$U(\cdot,\cdot)$	Uniform distribution	L_{cl}	Contrastive learning loss
$\odot$	Riemannian matrix multiplication	ω	Loss weighting hyperparameters
H	Hidden features in Riemannian graph convolution	$\mathscr{W}$	Network parameters
σ	Non-linear activation function (ReLU)	ψ_m^Q	Weighted cosine similarity for negative samples

3.1. Smart contract graph construction

We utilize Python-Solidity-Parser [60] to construct an Abstract Syntax Tree (AST). By applying the lexical and syntactic rules of Solidity, this tool transforms source code into a structured hierarchy of nodes, capturing the fundamental logic of the contract.

3.1.1. AST simplification

Although the AST captures the detailed structural semantics of smart contracts, including variable declarations, data types, and control flows, not all of this information is relevant to vulnerability detection. Attributes such as "initialValue", "isDeclaredConst", and "visibility" rarely contribute to security flaws. Instead, they introduce noise that can hinder the neural network's learning process. To address this, we introduce Algorithm 1 to filter out vulnerability-irrelevant nodes, yielding a simplified AST (S-AST). This simplification ensures that the model focuses exclusively on security-critical structural features.

By applying Algorithm 1, the S-AST accurately isolates the information necessary for security verification. An example of an S-AST generated by Algorithm 1 is shown in Fig. 1.

3.1.2. Smart contract graph construction

While S-AST captures comprehensive syntactic and semantic details, such as code structure and function calls, it lacks explicit data-flow dependencies and sequential execution order. To address this, we augment S-AST with control-flow and data-flow edges to generate the final Smart Contract Graph (SCG), as detailed in Algorithm 2.

Given an S-AST, we first perform a depth-first traversal to extract nodes and edges, initializing the foundation of the smart contract graph.

Algorithm 1 AST simplification.

Input: *AST*
Output: *S-AST*
Create an empty object *S-AST*
for each element *e* in *AST* **do**
  if *e* is an iterable variable **then**
    Recursively process it
  else if *e*'s key is "name" or "type" **then**
    Add *e* to *S-AST*
  end if
end for
return *S-AST*

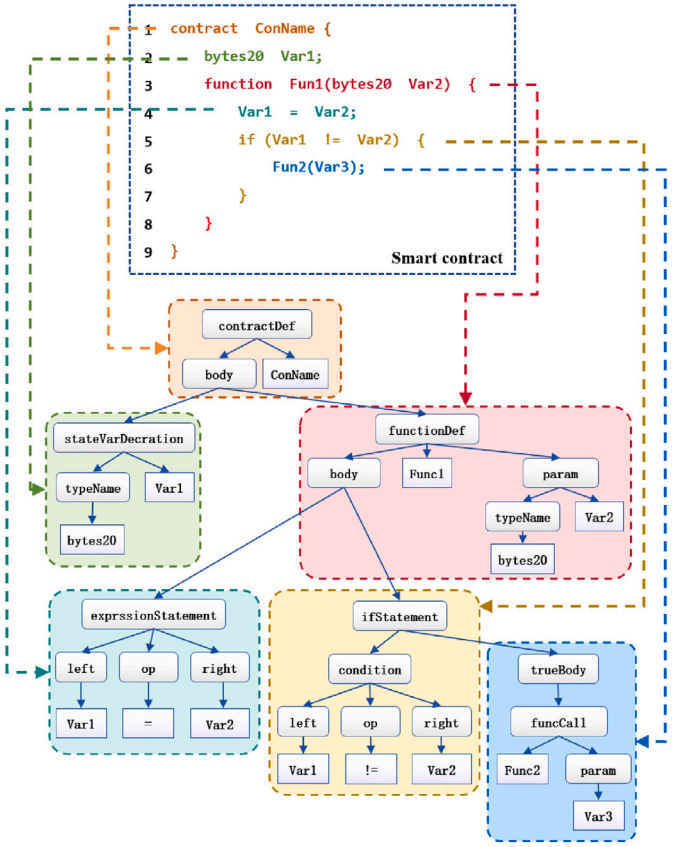


Fig. 1. Visualization of S-AST constructed from smart contract source code. Each node in the tree corresponds to a portion of the source code.

Algorithm 2 Smart contract graph generation.

Input: Simplified Abstract Syntax Tree S-AST
Output: Smart Contract Graph SCG $= (V, E)$

Create an empty set V to store nodes.
Create an empty set E to store edges.
Function *traverse*(*node*):
  Add node to V.
  if node is a variable **then**
    for all v in V **do**
      if `v` is a variable and `v.name == node.name` **then**
        Add ⟨`v`, `node`⟩ to E.
      end if
    end for
  end if
  for all child in node **do**
    if node is an ordered execution block **then**
      Record the index of child.
    end if
    Add ⟨`node.key`, *index* + *child*⟩ to E.
    traverse(child).
  end for
traverse(S-AST)
return SCG $= (V, E)$

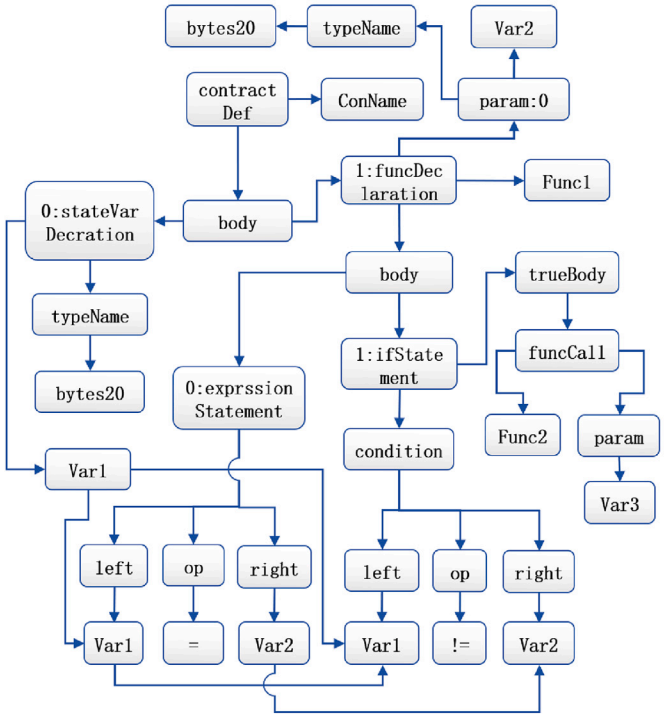


Fig. 2. An example of a smart contract graph generated by Algorithm 2, which captures rich semantics and deep hierarchical structural features.

Before appending child nodes, we record their positional indices to model the sequential execution flow. Additionally, we introduce data-flow edges linking variable definitions to their subsequent usages. As illustrated in Fig. 2, the resulting smart contract graph provides a robust, fine-grained structural representation for downstream feature extraction.

3.2. Riemannian manifold-based graph convolutional network

To effectively model hierarchical, tree-like smart contract graphs, we introduce an RGCNet. By leveraging the properties of negatively curved spaces, where volume grows exponentially with radius, RGCNet accommodates the complex topology of these graphs to enable more precise feature representation learning. The overall architecture is illustrated in Fig. 3.

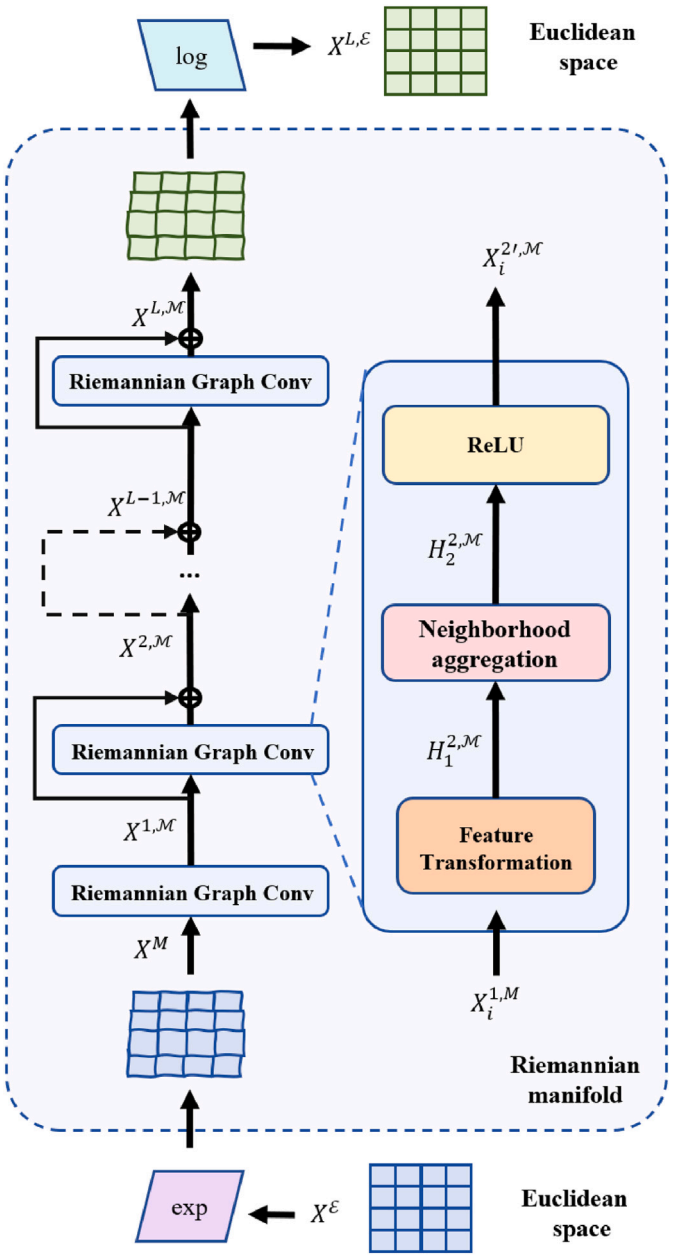


Fig. 3. The graph feature matrix $X^{\mathcal{E}}$ undergoes exp transformation and Riemannian graph convolution for feature extraction and aggregation in the Riemannian manifold, and is then mapped back to Euclidean space through log transformation.

3.2.1. Riemannian manifold feature initialization

In a Riemannian manifold $\mathcal{M}$, a geodesic represents the shortest path between two points $p, q \in \mathcal{M}$, minimizing the curve length defined by the Riemannian metric tensor g:

$$dis(p,q) = \inf_{\gamma}\left(\int_0^1 \sqrt{g_{\gamma(t)}(\gamma'(t),\gamma'(t))}\,dt\right) \tag{1}$$

where γ' is the tangent vector along the curve γ, with $\gamma(0) = p$ and $\gamma(1) = q$. The metric $g_p(x, y)$ is computed via the Minkowski inner

product $\langle x, y\rangle_L$ of tangent vectors x, y in the tangent space $\mathcal{T}_p\mathcal{M}$:

$$g_p(x, y) = \langle x, y\rangle_L = -x_0 y_0 + \sum_{i=1}^{d} x_i y_i \tag{2}$$

where d is the spatial dimension. The geodesic γ_v starting at p with the tangent vector $v \in \mathcal{T}_p\mathcal{M}$ corresponds to a vector in Euclidean space.

This geometric property allows us to map a Euclidean vector $v \in \mathcal{E}$ (acting as the tangent space $\mathcal{T}_p\mathcal{M}$) to a point $q \in \mathcal{M}$ using the exponential map $\exp_p^c : \mathcal{T}_p\mathcal{M} \to \mathcal{M}$. Conversely, the logarithmic map $\log_p^c : \mathcal{M} \to \mathcal{T}_p\mathcal{M}$ projects it back. The specific formulas are:

$$\exp_p^c(v) = \left(\frac{v \tanh(\sqrt{-c\langle v, v\rangle_L})}{\sqrt{-c\langle v, v\rangle_L}} + p \right) \cosh(\sqrt{-c\langle v, v\rangle_L}) \tag{3}$$

$$\log_p^c(q) = \left(\frac{(q - c\langle p, q\rangle_L p) \cdot \operatorname{arcosh}(c\langle p, q\rangle_L)}{\sqrt{-c\langle q - c\langle p, q\rangle_L p, q - c\langle p, q\rangle_L p\rangle_L}} \right) \tag{4}$$

where c denotes the manifold's curvature, and tanh, cosh, and arcosh represent the hyperbolic tangent, cosine, and inverse hyperbolic cosine functions, respectively.

We initially extract node features using Word2Vec, yielding a feature matrix $X \in \mathbb{R}^{N\times C}$, where N is the number of nodes and C is the feature dimensionality. Subsequently, we map X onto $\mathcal{M}$ through the origin o using the exponential map:

$$X^{\mathcal{M}} = \begin{bmatrix} \exp_o^c(X_1) \\ \exp_o^c(X_2) \\ \vdots \\ \exp_o^c(X_N) \end{bmatrix} \tag{5}$$

3.2.2. Attention connection weight matrix

To ensure geometric consistency during aggregation, we compute the attention weights between each node and its neighbors using the centroid of the squared Lorentz distance. For node i and its neighbor j, the attention weight G_{ij}^l at the l-th layer is computed via a softmax function:

$$G_{ij}^l = \frac{e^{\rho_{ij}}}{\sum_{n\in Nbr(i)} e^{\rho_{in}}} \tag{6}$$

where $Nbr(i) = \{j \mid A_{ij} \neq 0, j \neq i\}$ denotes the neighborhood of node i. The attention coefficient ρ_{ij}, representing the importance of node j to i, is derived from the squared Lorentz distance:

$$\rho_{ij} = -dtc_L^2(M_{\text{att}} X_i, M_{\text{att}} X_j) \tag{7}$$

with $M_{\text{att}} \in \mathbb{R}^{C\times C}$ acting as the learnable transformation matrix for self-attention. The squared Lorentz distance $dtc_L^2(\cdot,\cdot)$ is defined as:

$$dtc_L^2(x, y) = \frac{2}{c} - 2\langle x, y\rangle_L \tag{8}$$

Subsequently, we symmetrically normalize the attention weight matrix using its diagonal degree matrix D:

$$\widetilde{G} = D^{-1/2} G D^{-1/2} \tag{9}$$

where $D \in \mathbb{R}^{N\times N}$ is computed as:

$$D_{ii} = \sum_{j=1}^{N} G_{ij}^l \tag{10}$$

As detailed in Algorithm 3, the resulting normalized matrix $\widetilde{G}$ explicitly models the attention-based topological connections while preserving the underlying Riemannian geometry.

Algorithm 3 Attention connection weights construction.

Input: Node feature matrix X, Adjacency matrix A
Output: Normalized attention connection weight matrix $\widetilde{G}$

```
Initialize an N × N zero matrix G
Initialize a list Nbr of size N to store neighboring nodes
for each node i ∈ {1, ..., N} do
    Compute the neighboring nodes Nbr[i] = {j | A[i][j] ≠ 0 and j ≠ i}
    for each j in Nbr[i] do
        ρ[j] = dtc_L^2(M_att · X[i], M_att · X[j])
    end for
    for each j in Nbr[i] do
        G[i][j] = softmax(ρ[j])
    end for
end for
Calculate the degree matrix: D_ii = Σ_j G[i][j]
Normalize G: G̃ = D^{-1/2} · G · D^{-1/2}
return G̃
```

3.2.3. Feature transformation and neighborhood aggregation

To enhance model robustness, we apply Dropout regularization to the Euclidean weight matrix $W^{(l,\mathcal{E})}$. Given a dropout rate p_d, the regularized weight $\widetilde{W}_{ij}^{(l,\mathcal{E})}$ is:

$$\widetilde{W}_{ij}^{(l,\mathcal{E})} = \begin{cases} 0, & \text{if } p_{ij} < p_d \\ W_{ij}^{(l,\mathcal{E})}, & \text{otherwise} \end{cases} \tag{11}$$

where $p_{ij} \sim U(0, 1)$ is a uniformly distributed random variable.

Using the regularized weight matrix, the Riemannian feature transformation is computed as:

$$H_1^{(l,\mathcal{M})} = X^{(l-1,\mathcal{M})} \odot_{c_l} \widetilde{W}_{ij}^{(l-1,\mathcal{E})} \tag{12}$$

where $\odot_{c_l}$ denotes matrix multiplication within the Riemannian manifold of curvature c_l. Specifically, for matrices A and B, this operation is defined as:

$$A \odot_{c_l} B = \begin{bmatrix} \exp_o^{(c_l)}(\log(A_1)B) \\ \exp_o^{(c_l)}(\log(A_2)B) \\ \vdots \\ \exp_o^{(c_l)}(\log(A_N)B) \end{bmatrix} \tag{13}$$

Subsequently, we perform neighborhood aggregation using the normalized attention matrix $\widetilde{G}$:

$$H_2^{(l,\mathcal{M})} = \widetilde{G}^{l-1} \odot_{c_l} H_1^{(l,\mathcal{M})} \tag{14}$$

Finally, a non-linear activation function is applied via tangent space projection to yield the updated feature representations for the next layer:

$$X^{(l',\mathcal{M})} = \begin{bmatrix} \exp_o^{(c_l)}(\sigma(\log_o^{(c_l)}(H_{2,1}^{(l,\mathcal{M})}))) \\ \exp_o^{(c_l)}(\sigma(\log_o^{(c_l)}(H_{2,2}^{(l,\mathcal{M})}))) \\ \vdots \\ \exp_o^{(c_l)}(\sigma(\log_o^{(c_l)}(H_{2,N}^{(l,\mathcal{M})}))) \end{bmatrix} \tag{15}$$

where σ represents the ReLU activation function.

3.2.4. Riemannian residual connections

To mitigate training instability, we employ Riemannian manifold addition to establish residual connections between consecutive layers, combining the previous output $X^{(l-1,\mathcal{M})}$ with the current raw output

Algorithm 4 Riemannian manifold graph convolution.

Input: Graph node feature matrix $X^{l-1,\mathscr{M}}$
Output: Feature matrix processed by RGCNet $X^{l,\mathscr{M}}$
1: $H_1^{l,\mathscr{M}}$ = Riemannian multiplication $(X^{l-1,\mathscr{M}}, \widetilde{W}^{(l-1)})$
2: $H_2^{l,\mathscr{M}}$ = Riemannian multiplication $(\widetilde{G}^{(l-1)}, H_1^{l,\mathscr{M}})$
3: $X^{l',\mathscr{M}}$ = Riemannian nonlinear activation $(H_2^{l,\mathscr{M}})$
4: $X^{l,\mathscr{M}}$ = Riemannian addition $(X^{l',\mathscr{M}}, X^{l-1,\mathscr{M}})$
5: **return** $X^{l,\mathscr{M}}$

$X^{(l',\mathscr{M})}$:

$$X^{(l,\mathscr{M})} = \begin{bmatrix} \exp_{X_1^{(l',\mathscr{M})}}^{(c_l)}(P_{o\to X_1^{(l',\mathscr{M})}}^{(c_l)}(X_1^{(l-1,\mathscr{M})})) \\ \exp_{X_2^{(l',\mathscr{M})}}^{(c_l)}(P_{o\to X_2^{(l',\mathscr{M})}}^{(c_l)}(X_2^{(l-1,\mathscr{M})})) \\ \vdots \\ \exp_{X_N^{(l',\mathscr{M})}}^{(c_l)}(P_{o\to X_N^{(l',\mathscr{M})}}^{(c_l)}(X_N^{(l-1,\mathscr{M})})) \end{bmatrix} \tag{16}$$

where $P_{x\to y}(v)$ denotes the parallel transport from point x to y in the Riemannian manifold, computed as:

$$P_{x\to y}(v) = v - \frac{\langle \log_x(y), v\rangle_L}{dtc_L^2(x,y)}(\log_x(y) + \log_y(x)) \tag{17}$$

After passing through all L graph convolution layers, the aggregated manifold features $X^{(L,\mathscr{M})}$ are mapped back to Euclidean space via the logarithmic map for downstream tasks:

$$X^{(L,\mathscr{E})} = \begin{bmatrix} \log_o^{(c_L)}(X_1^{(L,\mathscr{M})}) \\ \log_o^{(c_L)}(X_2^{(L,\mathscr{M})}) \\ \vdots \\ \log_o^{(c_L)}(X_N^{(L,\mathscr{M})}) \end{bmatrix} \tag{18}$$

The entire graph convolution process within the Riemannian manifold, which effectively captures the hierarchical properties of the smart contract graph, is summarized in Algorithm 4.

3.3. Annealing mixup graph contrastive learning

To improve the robustness of the model against noisy labels, we suggest an innovative graph contrastive learning method that combines annealing and mixup techniques. Contrastive learning focuses on distinguishing between positive and negative sample pairs. Annealing refers to the process of gradually increasing the difficulty of samples during training, enabling the model to better adapt to complex data distributions. The overall process is shown in Fig. 4.

3.3.1. Generating annealing mixup positive sample set

First, two graphs G_i and G_j are encoded by the RGCNet into embedding matrices $\mathscr{X}_i$ and $\mathscr{X}_j$. To generate mixup samples, a mixing coefficient ϑ is drawn from a Beta distribution:

$$\mathrm{Beta}(\alpha,\beta) = \int_0^1 \tau^{\alpha-1}(1-\tau)^{\beta-1}d\tau \tag{19}$$

where $\alpha,\beta > 0$. The mixup sample $\mathscr{X}_Q$ and its corresponding soft label t_Q are computed as:

$$(\mathscr{X}_Q, t_Q) = (\vartheta\mathscr{X}_i + (1-\vartheta)\mathscr{X}_j, \vartheta\cdot\phi(t_i) + (1-\vartheta)\cdot\phi(t_j)) \tag{20}$$

where $\phi(\cdot)$ is a label smoothing function designed to reduce over-reliance on hard labels. Given a small constant ϵ, $\phi(t)$ is defined as:

$$\phi(t) = (1-\epsilon\cdot g(epoch))\cdot t + \frac{\epsilon\cdot g(epoch)}{2} \tag{21}$$

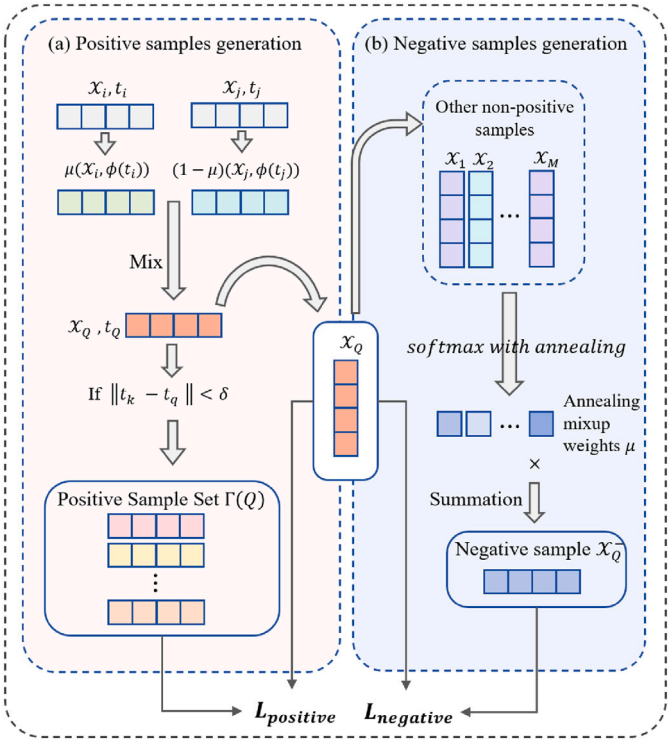


Fig. 4. Annealing mixup graph contrastive learning which adopts two sample generation strategies: (a) generate annealing positive samples by combining graph feature representations through convex hull combinations, and (b) Create annealing hard negative sample pairs by integrating multiple non-positive samples within a batch.

with $g(epoch)$ being a progressive annealing function:

$$g(epoch) = \tanh(\kappa\cdot epoch) \tag{22}$$

Here, κ controls the scaling rate. This progressive annealing prevents excessive early-stage label softening, stabilizing the initial training phase.

Given a batch of mixup representations $\{\mathscr{X}_1,\ldots,\mathscr{X}_M\}$ (where M is the batch size), samples with semantically similar labels are treated as positive pairs. Formally, for a query $\mathscr{X}_Q$, the index set of its positive samples is:

$$\Gamma(Q) = \{k\in\{1,\ldots,M\} \mid \|t_k - t_Q\| < \delta\} \tag{23}$$

where $\|\cdot\|$ denotes the L2 norm, and δ is a small distance threshold.

3.3.2. Generating annealing hard negative samples

To construct hard negative samples, we aggregate all non-positive samples within a mini-batch using adaptive weights. Formally, for a query $\mathscr{X}_Q$, the composite negative sample is defined as:

$$\mathscr{X}_Q^- = \sum_{m=1,m\notin(\Gamma(Q)\cup\{Q\})}^{M} \mu_m^Q \mathscr{X}_m \tag{24}$$

where M is the mini-batch size, and $\sum \mu_m^Q = 1$. To emphasize challenging negatives, the weight μ_m^Q scales exponentially with the cosine similarity between the negative sample and the query. We first define the weighted cosine similarity as:

$$\psi_m^Q = \frac{\mathscr{X}_Q\cdot\mathscr{X}_m}{\|\mathscr{X}_Q\|\|\mathscr{X}_m\|}\cdot g(epoch) \tag{25}$$

Then, the adaptive weight is computed as:

$$\mu_m^Q = \frac{e^{\psi_m^Q}}{\sum_{n \notin (\Gamma(Q) \cup \{Q\})} e^{\psi_n^Q}} \tag{26}$$

The annealing function $g(epoch)$ gradually increases the influence of harder negative samples as training progresses.

3.3.3. Graph contrastive learning

With the positive sample set $\Gamma(Q)$ and the hard negative sample $\mathcal{X}_Q^-$ defined, we formulate the contrastive loss to minimize the distance between the query and its positive samples (L_{positive}) while maximizing its distance from the negative sample (L_{negative}):

$$L_{\text{positive}} = \sum_{Q=1}^{M} \left(\frac{1}{|\Gamma(Q)|} \sum_{n \in \Gamma(Q)} \|\text{MLP}(\mathcal{X}_Q) - \text{MLP}(\mathcal{X}_n)\| \right) \tag{27}$$

$$L_{\text{negative}} = \sum_{Q=1}^{M} \|\text{MLP}(\mathcal{X}_Q) - \text{MLP}(\mathcal{X}_Q^-)\| \tag{28}$$

$$L_{\text{cl}} = L_{\text{positive}} - L_{\text{negative}} \tag{29}$$

Here, MLP$(\cdot)$ projects the graph representations into a new embedding space. Ultimately, L_{cl} enhances the graph encoder's discriminative power by pulling similar mixup samples closer and pushing the aggregated negative samples farther away.

3.4. End-to-end learning framework for smart contract vulnerability detection

We adopt an end-to-end learning strategy to jointly optimize all modules, enabling mutual enhancement (Fig. 5). Specifically, Word2Vec extracts initial semantic information, RGCNet captures structural graph characteristics, and the annealing mixup contrastive learning provides a noise-resistant regularization signal. This unified architecture deepens the fusion of semantic and structural features, effectively uncovering key characteristics indicative of vulnerabilities.

3.4.1. Word embedding feature module

We use the Word2Vec Skip-gram model to generate initial node embeddings by predicting context words. The objective is to maximize the log-likelihood of the conditional probability:

$$L_w = -\sum_{ti=1}^{T} \sum_{-wd \le j \le wd, j \neq 0} \log P(w_{ti+j} \mid w_{ti}) \tag{30}$$

where T is the total number of words, wd is the context window size, w_{ti} is the center word, and w_{ti+j} is the context word.

3.4.2. Riemannian graph convolutional feature extraction module

After RGCNet extracts the graph-level representation X, it is passed through a Multi-Layer Perceptron (MLP) classifier f to yield the final prediction:

$$\hat{t} = f(X) \tag{31}$$

We employ the standard cross-entropy loss L_{cf} for the binary classification task:

$$L_{cf} = -\frac{1}{Num} \sum_{i=1}^{Num} (t_i \log(\hat{t}_i) + (1 - t_i) \log(1 - \hat{t}_i)) \tag{32}$$

where Num is the total number of samples, and t_i is the ground-truth label of sample i.

3.4.3. Combined loss function and optimization

The overall objective function integrates the word embedding loss L_w, the classification loss L_{cf}, and the contrastive loss L_{cl}:

$$L = \omega_1 L_w + \omega_2 L_{cf} + \omega_3 L_{cl} \tag{33}$$

where ω_1, ω_2, and ω_3 are weighting hyperparameters. The network parameters $\mathscr{W}$, comprising both Word2Vec and RGCNet components, are updated simultaneously via stochastic gradient descent to minimize L. This joint optimization ensures that word embeddings are refined by downstream structural feedback, while RGCNet benefits from more accurate semantic initialization. Concurrently, the contrastive loss imposes a robust penalty against label noise, driving the entire system toward optimal generalization.

4. Experiments

4.1. Datasets and experiment settings

The dataset comprises real-world Ethereum smart contracts sourced from SmartBugs [61] and ESC [33], containing 47,581 and 40,932 files, respectively. To establish reliable ground truth, we employ a hybrid labeling strategy that combines traditional program analysis with manual inspection. Specifically, detection results from existing analysis tools serve as preliminary labels, while any discrepancies among tool outputs are resolved through manual review. This hybrid approach ensures higher reliability and scalability than purely manual annotation. For each vulnerability category, secure and vulnerable contracts are labeled as 0 and 1, respectively. To prevent class imbalance, we construct a balanced dataset by pairing the vulnerable samples with an equivalent number of secure samples. Table 2 summarizes the final sample distribution. The dataset is partitioned into training, validation, and test sets at an 8:1:1 ratio.

We evaluated each type of vulnerability dataset, treating it as a binary classification problem, i.e., determining whether there are vulnerabilities in the samples. To comprehensively evaluate the performance of different methods, we employed a confusion matrix including Accuracy, Precision, Recall, and F1-score to compare the model's predictions with actual labels and then calculate key evaluation metrics.

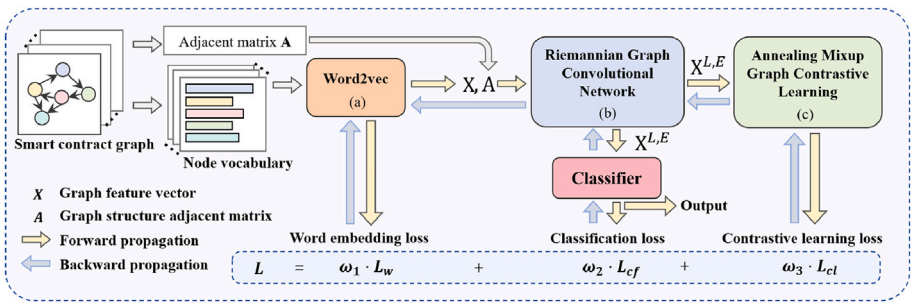


Fig. 5. An end-to-end learning framework for smart contract vulnerability analysis with 3 modules: (a) Word2Vec for semantic feature learning, (b) RGCNet for structural feature learning, and (c) annealing mixup contrastive learning for anti-noise.

Table 2
Sample distributions.

Vulnerability Types	Secure Samples	Vulnerable Samples	Total Samples
Reentrancy	10,000	9617	19,617
Arithmetic	17,000	16,252	33,252
Timestamp Dependence	10,000	9738	19,738
All Types	37,000	35,607	72,607

The specific hardware environment for this experiment included an Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz, an NVIDIA GeForce RTX 3090 GPU, 64GB of memory, and 12TB of disk space. The software environment employed Ubuntu 20.04.1 as the operating system, Python 3.8.18, PyTorch 2.1.1 as the deep learning framework, and CUDA 12.2.

4.2. Comparison with state-of-the-art methods

4.2.1. Baselines

We compared our framework against three categories of widely adopted baselines: conventional program analysis tools, dedicated deep learning models, and Large Language Model (LLM)-based methods. The conventional baselines were Oyente [25], Mythril [27], and Slither [28]. These classic tools rely on established techniques such as symbolic execution and taint analysis. The dedicated deep learning baselines were text-based sequence models (Transformer [62]), state-of-the-art graph neural networks (TMP [33], Peculiar [35], and CGE [34]), and cutting-edge multi-teacher and contrastive methods (Clear [53], MTVHunter [48]). The LLM-based methods were the latest models leveraging large-scale pre-training and preference optimization (SAEL [55] and Smart-LLaMA-DPO [56]). These baselines collectively represent a comprehensive spectrum of structural, semantic, knowledge-translation, and preference-based code representations.

All methods were evaluated on the same dataset. Among the conventional tools, Oyente and Mythril can identify all three targeted vulnerability types. Under the current evaluation setup, Slither does not provide support for detecting arithmetic vulnerabilities. Consequently, these tools were only evaluated on their supported vulnerability categories. Among the deep learning baselines, CGE failed to identify arithmetic vulnerabilities, primarily because such flaws typically stem from low-level programming errors rather than explicit structural or behavioral patterns that expert rules can capture. The experimental results across supported vulnerability types are illustrated in Fig. 6 and detailed in Table 3.

4.2.2. Results

Deep learning-based methods significantly outperformed conventional program analysis tools across all metrics. Conventional tools rely on rigid expert rules that struggle with diverse real-world vulnerability patterns, leading to high false-negative rates and low recall scores (Table 3). Conversely, deep learning approaches automatically extract latent semantic and structural features, achieving superior detection performance and generalization.

Among deep learning baselines, our framework achieves state-of-the-art results (Fig. 6). It substantially outperforms sequence-based models like Transformer, which treat contracts merely as text and fail to capture critical structural dependencies.

Compared to existing graph-based methods, our approach shows distinct advantages. Although TMP pioneered GNNs for this task, its reliance on predefined key functions limits the captured structural information. Relative to Peculiar, our method improves detection accuracy by 4.33%, 7.17%, and 8.07% across the three vulnerabilities. We attribute this to Peculiar's reliance on Euclidean space, which struggles to represent hierarchical features, and its disjointed training pipeline. Moreover,

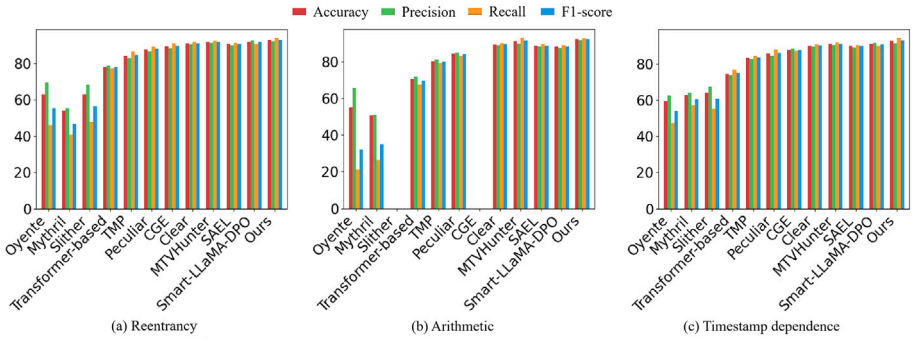


Fig. 6. Experimental results of various tools on three kinds of vulnerabilities including (a) reentrancy, (b) arithmetic, and (c) timestamp dependence.

Table 3
Comparison study experimental results.

Methods	Reentrancy				Arithmetic				Timestamp Dependence			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
Oyente [25]	62.93	69.56	45.98	55.37	55.10	65.76	21.27	32.15	59.58	62.56	47.41	53.94
Mythril [27]	53.91	55.32	40.68	46.88	50.63	51.07	26.44	34.84	62.77	64.25	57.31	60.58
Slither [28]	62.82	68.17	48.09	56.39	–	–	–	–	64.31	67.37	55.29	60.74
Transformer-based	78.18	78.68	77.31	77.99	70.41	71.69	67.45	69.50	74.58	73.65	76.55	75.07
TMP [33]	84.21	82.64	86.61	84.58	80.27	80.97	79.15	80.05	83.33	82.64	84.38	83.50
Peculiar [35]	87.72	86.67	89.14	87.89	84.25	85.03	83.13	84.07	85.81	84.34	87.96	86.11
CGE [34]	89.50	88.23	91.15	89.67	–	–	–	–	87.84	88.34	87.18	87.75
Clear [53]	91.05	90.42	91.78	91.09	89.43	88.92	90.15	89.53	90.12	89.56	90.74	90.15
MTVHunter [48]	91.82	91.24	92.51	91.87	91.28	89.81	**93.12**	91.44	91.08	90.53	91.95	91.23
SAEL [55]	90.65	89.92	91.43	90.66	88.76	88.15	89.54	88.84	89.87	89.24	90.58	89.90
Smart-LLaMA-DPO [56]	91.83	**92.64**	90.87	91.75	88.15	87.52	88.94	88.23	90.99	**91.72**	90.12	90.91
Ours	**93.13**	92.18	**94.09**	**93.13**	**92.41**	**91.77**	92.99	**92.38**	**92.88**	91.47	**94.41**	**92.91**

Acc = accuracy, Prec = precision, Rec = recall, F1 = F1-score (all in %)

Peculiar is tailored primarily for reentrancy, resulting in sub-optimal stability on other vulnerability types.

Finally, our framework outperforms CGE by 2.56% and 2.91% on reentrancy and timestamp dependence, respectively. CGE's reliance on expert patterns introduces scalability bottlenecks similar to conventional methods. Our framework also demonstrates highly competitive performance compared to the latest 2024 and 2025 state-of-the-art methods, including Clear, MTVHunter, SAEL, and Smart-LLaMA-DPO. It is worth noting that while Smart-LLaMA-DPO achieves slightly higher precision on reentrancy (92.64%) and timestamp dependence (91.72%) due to its rigorous preference optimization on semantic outputs, and MTVHunter achieves a marginally higher recall (93.12%) on arithmetic vulnerabilities via multi-teacher knowledge distillation, our approach leverages Riemannian geometry to more naturally and accurately represent the intrinsic hierarchical structure of smart contracts. This specialized end-to-end graph architecture explicitly models structural dependencies, consistently yielding the highest overall detection accuracy and F1-scores across all evaluated vulnerabilities.

4.3. Ablation study

In this section, we conducted ablation experiments to evaluate the effectiveness of each proposed module. The smart contract vulnerability detection framework proposed in this study consists of three core components: an RGCNet, an end-to-end learning framework, and an annealing mixup graph contrastive learning method. To explore the impact of each component on the overall system performance, we evaluated the contribution of each by disabling or replacing these three core parts.

4.3.1. The effects of the RGCNet

To evaluate the RGCNet component, we substituted it with a standard GCN, an LSTM, and a Transformer, while keeping the rest of the framework intact. As shown in Table 4, RGCNet yields the best performance among all variants. Notably, compared to the standard GCN, RGCNet improves detection accuracy by 2.67%, 3.63%, and 3.70% across the three vulnerability types. This significant gain confirms that the scale-free and deep hierarchical structures of smart contract graphs are more naturally embedded and represented in Riemannian manifolds.

4.3.2. The effects of end-to-end learning

To evaluate the end-to-end learning strategy, we decoupled the word embedding generation from the GNN training. Specifically, Word2Vec independently generates a frozen node feature matrix, which is then fed into the RGCNet. As shown in Table 5 ("Disable End-to-End Learning"), this disjointed two-stage approach leads to a noticeable performance drop. Conversely, our unified training increases detection accuracy by 2.27%, 2.85%, and 2.26% across the three vulnerability types. Joint optimization enables mutual feedback between Word2Vec and RGCNet, producing more discriminative semantic and structural representations to uncover complex vulnerability patterns.

4.3.3. The effects of annealing mixup graph contrastive learning

We evaluated the impact of graph contrastive learning by directly feeding the RGCNet outputs into an MLP classifier, bypassing the contrastive loss. As shown in Table 5 ("Disable Contrastive Learning"), omitting this module significantly degrades performance, reducing detection accuracy by 2.14%, 2.45%, and 2.12% across the three vulnerabilities. Due to the fact that existing smart contract datasets rely on manual inspection and traditional tools, their labels inevitably contain noise and omissions. By increasing sample diversity and complexity during training, the annealing mixup contrastive learning strategy effectively mitigates this label noise, thereby strengthening the model's generalization ability and robustness.

4.3.4. The effects of the combined methods

Finally, to comprehensively evaluate the collective contribution of our proposed components, we tested a variant that removes all three core modules. Specifically, this baseline degrades to a standard pipeline: independent Word2Vec embeddings followed by a regular Graph Convolutional Network (GCN) and an MLP classifier. As shown in Table 5 ("Disable All Methods"), this variant exhibits the worst performance, with accuracy dropping by 8.38%, 10.39%, and 9.35% across the three vulnerability types. These substantial declines confirm that RGCNet, the end-to-end learning framework, and the annealing mixup contrastive learning method are all indispensable.

In summary, each proposed module not only provides significant independent performance gains but also exhibits strong complementary and synergistic effects, establishing a highly accurate and robust vulnerability detection framework.

5. Conclusion

In this paper, we have presented a novel smart contract vulnerability detection framework integrating a RGCNet, an end-to-end learning paradigm, and an annealing mixup graph contrastive learning strategy. Extensive experiments demonstrated that our approach significantly outperforms the state-of-the-art baselines in detecting reentrancy, arithmetic, and timestamp dependence vulnerabilities. The RGCNet naturally

Table 4
Experimental results for Different Network Replacements.

Networks	Reentrancy				Arithmetic				Timestamp dependency			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
LSTM	79.02	78.37	79.52	78.94	75.48	76.92	72.03	74.40	76.64	76.98	75.27	76.12
Transformer	85.31	85.18	85.10	85.14	80.73	81.05	79.65	80.35	81.94	80.86	83.17	82.00
Regular GCN	90.46	90.82	89.78	90.30	88.78	87.80	89.33	88.56	89.18	90.20	87.64	88.90
RGCNet	**93.13**	**92.18**	**94.09**	**93.13**	**92.41**	**91.77**	**92.99**	**92.38**	**92.88**	**91.47**	**94.41**	**92.91**

Acc = accuracy, Prec = precision, Rec = recall, F1 = F1-score (all in %)

Table 5
Experimental results of disabling different modules.

Methods	Reentrancy				Arithmetic				Timestamp dependence			
	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1	Acc	Prec	Rec	F1
Disable End-to-End Learning	90.86	91.02	90.44	90.72	89.56	90.42	88.24	89.32	90.62	89.37	91.47	90.41
Disable Contrastive Learning	90.99	91.87	89.72	90.78	89.96	90.05	89.60	89.82	90.76	90.88	90.39	90.63
Disable All Methods	84.75	86.12	82.45	84.25	82.02	83.12	79.86	81.46	83.53	84.98	81.04	82.95
Complete Method	**93.13**	**92.18**	**94.09**	**93.13**	**92.41**	**91.77**	**92.99**	**92.38**	**92.88**	**91.47**	**94.41**	**92.91**

Acc = accuracy, Prec = precision, Rec = recall, F1 = F1-score (all in %)

models the scale-free and hierarchical structures of smart contract graphs, while the end-to-end framework and contrastive learning strategy jointly enhance the discriminative power and robustness against noisy labels. Ablation studies further confirmed the indispensability of each component. By improving detection accuracy and robustness, our method bolsters the security and trustworthiness of blockchain ecosystems. One specific future work will be to adapt our Riemannian graph learning framework to cross-chain vulnerability detection scenarios, enabling robust security analysis across diverse blockchain platforms with varying smart contract languages. Another promising direction is to integrate LLMs for enhanced semantic understanding, which could further improve zero-shot vulnerability detection capabilities.

CRediT authorship contribution statement

Yaoxin Chen: Writing – original draft, Validation, Software, Methodology, Formal analysis, Conceptualization. **Haiming Zhu:** Resources, Methodology, Data curation. **Haibo Li:** Writing – original draft, Validation, Methodology, Investigation, Conceptualization. **Yaming Yang:** Writing – review & editing, Writing – original draft, Validation, Methodology, Data curation, Conceptualization. **Qicong Wang:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Maozhen Li:** Writing – review & editing, Supervision, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant No. 62571464, the Shenzhen Science and Technology Projects under Grant No. JCYJ20200109143035495, and the Natural Science Foundation of Fujian Province under Grant No. 2023J01003.

Data availability

Data will be made available on request.

References

[1] T. Lv, X. Gao, J. Lin, X. Gong, F. Wang, J. Wu, H. Zhang, N. Zeng, Prior knowledge-guided adaptive multi-scale deep learning for surface-enhanced raman spectroscopy-based liver disease classification, Eng. Appl. Artif. Intell. 167 (2026) 113941, https://doi.org/10.1016/j.engappai.2026.113941, https://www.sciencedirect.com/science/article/pii/S0952197626002228.
[2] A. Dong, A. Starr, Y. Zhao, Neural network-based parametric system identification: a review, Int. J. Syst. Sci. 54 (13) (2023) 2676–2688.
[3] Y. Zhan, R. Yang, J. You, M. Huang, W. Liu, X. Liu, A systematic literature review on incomplete multimodal learning: techniques and challenges, Syst. Sci. Control Eng. 13 (1) (2025) 2467083.
[4] X. Hao, Y. Xia, H. Yang, Z. Zuo, Typical motion-based modelling and tracking for vehicle targets in linear road segment, Int. J. Syst. Sci. 55 (5) (2024) 833–843.
[5] R. Deepika, T.S.P. Kumar, An interpretable dual-level lightweight framework for malaria diagnosis for enhanced remote diagnosis in resource-constrained settings, Syst. Sci. Control Eng. 13 (1) (2025) 2579988.
[6] M. Zhong, X. Zhu, T. Xue, L. Zhang, An overview of recent advances in model-based event-triggered fault detection and estimation, Int. J. Syst. Sci. 54 (4) (2023) 929–943.
[7] P. Wu, W. Wen, H. Li, Z. Li, N. Zeng, Ai-driven automation of aviation equipment inspection: Insights from a complex adaptive systems perspective, Innovation 7 (1) (2026) 101084, https://doi.org/10.1016/j.xinn.2025.101084, https://www.sciencedirect.com/science/article/pii/S2666675825002875.
[8] W. Ehab, L. Huang, Y. Li, Unet and variants for medical image segmentation, Int. J. Netw. Dyn. Intell. 3 (2) (2024) 100009.
[9] Y. Zhou, Y. Guo, C. Liu, H. Peng, H. Rao, Synchronization for markovian master-slave neural networks: an event-triggered impulsive approach, Int. J. Syst. Sci. 54 (12) (2023) 2551–2565.
[10] D. Teng, P.-S. Wu, R.-L. Wang, C. Lu, N.-Y. Zeng, Vectorial importance-weighted neural network framework for aviation structural systems multi-failures related reliability estimation, Aerosp. Sci. Technol. 177 (2026) 112219, https://doi.org/10.1016/j.ast.2026.112219, https://www.sciencedirect.com/science/article/pii/S1270963826005997.
[11] Q. Ma, D. Wu, X. Luo, A review of deep learning-based power load forecasting methods, Int. J. Netw. Dyn. Intell. 4 (4) (2025) 100027.
[12] X. Yan, C. Wang, Y. Jin, Federated bimodal graph neural networks for text-image retrieval, Int. J. Netw. Dyn. Intell. 4 (2) (2025) 100009.
[13] B. Song, S. Zhao, L. Dang, H. Wang, L. Xu, A survey on learning from data with label noise via deep neural networks, Syst. Sci. Control Eng. 13 (1) (2025) 2488120.
[14] Y. Zhang, X. Zhang, D. Miao, H. Yu, Real-time semantic segmentation of road scenes via hybrid dilated grouping network, Int. J. Netw. Dyn. Intell. 4 (1) (2025) 100006.
[15] Q. Lan, A. Kaul, S. Jones, Prompt injection detection in llm integrated applications, Int. J. Netw. Dyn. Intell. 4 (2) (2025) 100013.
[16] L. Hu, Z. Wang, P. Wu, K. Yu, N. Zeng, Dartsnext: Bridging the search-evaluation gap in differentiable nas with router-based selection and sequential architecture, IEEE Trans. Emerg. Top. Comput. Intell. (2026) 1–13, https://doi.org/10.1109/TETCI.2026.3670670
[17] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Tech. rep., Bitcoin.org, 2008.
[18] N. Szabo, Smart contracts: building blocks for digital markets, EXTROPY, The Journal of Transhumanist Thought 18 (2) (1996) 28.
[19] G. Wood, Ethereum: A secure decentralised generalised transaction ledger, Tech. rep., Ethereum Foundation, 2014.
[20] D. Tapscott, A. Tapscott, Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world, Penguin, 2016.
[21] V. Buterin, A next-generation smart contract and decentralized application platform, Tech. rep., Ethereum Foundation, 2014.
[22] SlowMist, Slowmist hacked, 2024, https://hacked.slowmist.io (last Accessed 25 March 2025).
[23] Etherscan, The DAO smart contract, 2016, https://etherscan.io/address/0xbb9bc244d798123fde783fcc1c72d3bb8c189413 (last Accessed 25 March 2025).
[24] ODAILY, Poly network 10 million dollar loss attack event analysis, 2023, https://www.odaily.news/post/5188161 (last Accessed 25 March 2025).
[25] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 2016, pp. 254–269.
[26] I. Grishchenko, M. Maffei, C. Schneidewind, A semantic framework for the security analysis of Ethereum smart contracts, in: Principles of Security and Trust: 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, (14 April 2018), Proceedings 7, Springer, Thessaloniki, Greece, 2018, pp. 243–269.
[27] ConsenSys, Mythril, Security analysis tool for evm bytecode, 2024, https://github.com/ConsenSys/mythril (last Accessed 25 March 2026).
[28] J. Feist, G. Grieco, A. Groce, Slither: A static analysis framework for smart contracts, in: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, IEEE, Montreal, Canada, 2019, pp. 8–15.
[29] O. Sürücü, U. Yeprem, C. Wilkinson, W. Hilal, S.A. Gadsden, J. Yawney, N. Alsadi, A. Giuliano, A survey on Ethereum smart contract vulnerability detection using machine learning, Disruptive Technologies in Information Sciences VI 12117 (2022) 110–121.
[30] W.J.-W. Tann, X.J. Han, S.S. Gupta, Y.-S. Ong, Towards safer smart contracts: A sequence learning approach to detecting security threats, arXiv preprint, 2018 arXiv:1811.06632.
[31] C. Xing, Z. Chen, L. Chen, X. Guo, Z. Zheng, J. Li, A new scheme of vulnerability analysis in smart contract with machine learning, Wirel. Netw. 30 (2024) 6325–6334, https://doi.org/10.1007/s11276-020-02379-z
[32] S.-J. Hwang, S.-H. Choi, J. Shin, Y.-H. Choi, Codenet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection, IEEE Access 10 (2022) 32595–32607.
[33] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, Q. He, Smart contract vulnerability detection using graph neural networks, in: Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence, Virtual Event, 2021, pp. 3283–3290.
[34] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, X. Wang, Combining graph neural networks with expert knowledge for smart contract vulnerability detection, IEEE Trans. Knowl. Data Eng. 35 (2) (2023) 1296–1310.
[35] H. Wu, Z. Zhang, S. Wang, Y. Lei, B. Lin, Y. Qin, H. Zhang, X. Mao, Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques, in: 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), IEEE, Virtual Event, 2021, pp. 378–389.
[36] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2008) 61–80.
[37] W. Chen, W. Fang, G. Hu, M.W. Mahoney, On the hyperbolicity of small-world and treelike random graphs, Internet Mathematics 9 (4) (2013) 434–491.
[38] E. Ravasz, A.-L. Barabási, Hierarchical organization in complex networks, Phys. Rev. E 67 (2) (2003) 026112.
[39] M. Yang, M. Zhou, Z. Li, J. Liu, L. Pan, H. Xiong, I. King, Hyperbolic graph neural networks: A review of methods and applications, arXiv preprint, 2022 arXiv:2202.13852.
[40] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint, 2013 arXiv:1301.3781.
[41] H. Zhang, M. Cisse, Y.N. Dauphin, D. Lopez-Paz, arXiv preprint, 2017 arXiv:1710.09412.

[42] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, M. Vechev, Securify: Practical security analysis of smart contracts, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, Canada, 2018, pp. 67–82.
[43] S. Kalra, S. Goel, M. Dhawan, S. Sharma, Zeus: Analyzing safety of smart contracts, in: Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 2018, https://doi.org/10.14722/ndss.2018.23082
[44] B. Jiang, Y. Liu, W.K. Chan, Contractfuzzer: Fuzzing smart contracts for vulnerability detection, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Albuquerque, NM, USA, 2018, pp. 259–269.
[45] Y. Yu, X. Si, C. Hu, J. Zhang, A review of recurrent neural networks: LSTM cells and network architectures, Neural Comput. 31 (7) (2019) 1235–1270.
[46] Z. Gao, L. Jiang, X. Xia, D. Lo, J. Grundy, Checking smart contracts with structural code embedding, IEEE Trans. Softw. Eng. 47 (12) (2020) 2874–2891.
[47] T. Kattenborn, J. Leitloff, F. Schiefer, S. Hinz, Review on convolutional neural networks (CNN) in vegetation remote sensing, ISPRS J. Photogramm. Remote Sens. 173 (2021) 24–49.
[48] G. Sun, Y. Zhuang, S. Zhang, X. Feng, Z. Liu, L. Zhang, Mtvhunter: Smart contracts vulnerability detection based on multi-teacher knowledge translation, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 39, Philadelphia, PA, USA, 2025, pp. 15169–15176.
[49] D. Chen, L. Feng, Y. Fan, S. Shang, Z. Wei, Smart contract vulnerability detection based on semantic graph and residual graph convolutional networks with edge attention, J. Syst. Softw. 202 (2023) 111705.
[50] Z. Zhen, X. Zhao, J. Zhang, Y. Wang, H. Chen, DA-GNN: A smart contract vulnerability detection method based on dual attention graph neural network, Comput. Netw. 242 (2024) 110238.
[51] J. Cai, B. Li, J. Zhang, X. Sun, B. Chen, Combine sliced joint graph with graph neural networks for smart contract vulnerability detection, J. Syst. Softw. 195 (2023) 111550.
[52] H. Xiong, Y. Zhong, C. Wu, W. Yi, Y. Zhao, A multi-code representation fusion smart contract vulnerability line detection method based on graph neural network, in: 2023 11th International Conference on Information Systems and Computing Technology (ISCTech), IEEE, Washington, DC, USA, 2023, pp. 28–33.
[53] Y. Chen, Z. Sun, Z. Gong, D. Hao, Improving smart contract security with contrastive learning-based vulnerability detection, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ICSE '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 1–11, https://doi.org/10.1145/3597503.3639173
[54] E. Mandana, G. Vlahavas, A. Vakali, Evullm: Ethereum smart contract vulnerability detection using large language models, Electronics 14 (16) (2025) 3226, https://doi.org/10.3390/electronics14163226, https://www.mdpi.com/2079-9292/14/16/3226.
[55] L. Yu, S. Cheng, Z. Huang, J. Zhang, C. Shen, J. Lu, L. Yang, F. Zhang, J. Ma, SAEL: Leveraging Large Language Models with Adaptive Mixture-of-Experts for Smart Contract Vulnerability Detection, in: 2025 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE Computer Society, Los Alamitos, CA, USA, 2025, pp. 61–72, https://doi.org/10.1109/ICSME64153.2025.00016, https://doi.ieeecomputersociety.org/10.1109/ICSME64153.2025.00016.
[56] L. Yu, Z. Huang, H. Yuan, S. Cheng, L. Yang, F. Zhang, C. Shen, J. Ma, J. Zhang, J. Lu, C. Zuo, Smart-llama-dpo: Reinforced large language model for explainable smart contract vulnerability detection, in: Proc. ACM Softw. Eng. 2 (ISSTA), 2025, pp. 182–205, https://doi.org/10.1145/3728878
[57] M. Nickel, D. Kiela, Poincaré embeddings for learning hierarchical representations, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc, Red Hook, NY, USA, 2017, pp. 6341–6350.
[58] Q. Liu, M. Nickel, D. Kiela, Hyperbolic graph neural networks, in: Advances in Neural Information Processing Systems, 2019, pp. 8228–8239.
[59] G. Bachmann, G. Bécigneul, O. Ganea, Constant curvature graph convolutional networks, in: International Conference on Machine Learning, PMLR, Virtual Event, 2020, pp. 486–496.
[60] ConsenSys, Python-solidity-parser, 2024, https://github.com/ConsenSys/python-solidity-parser (last Accessed 25 March 2025).
[61] T. Durieux, J.F. Ferreira, R. Abreu, P. Cruz, Empirical review of automated analysis tools on 47,587 Ethereum smart contracts, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Japan, 2020, pp. 530–541.
[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, Curran Associates Inc, Red Hook, NY, USA, 2017, pp. 6000–6010.

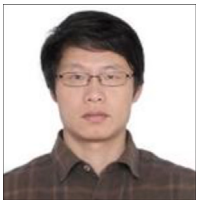
Author biography

Yaoxin Chen received the BE degree in Computer Science and Technology from Xiamen University, Xiamen, China in 2024. He is currently pursuing the graduate degree with the Institute of Artificial Intelligence, Xiamen University, Xiamen, China. His research interests include blockchain, deep learning, and UAVs.

Haiming Zhu received the BE degree in Internet of Things Engineering from Jinan University, Zhuhai, China in 2021. He is currently pursuing the graduate degree with the Department of Computer Science and Technology, Xiamen University, Xiamen, China. His research interests include blockchain security, graph neural networks, and deep learning.

Haibo Li is currently pursuing the graduate degree with the Department of Computer Science and Technology, Xiamen University, Fujian, China. His research interests include computer vision, machine learning and rough sets.

Yaming Yang received a BSc in Marine Science from Sun Yat-Sen University, Zhuhai, China in 2023. He is currently pursuing the graduate degree with the Department of Computer Science and Technology, Xiamen University, Xiamen, China. His research interests include blockchain, SLAM and deep learning.



Qicong Wang received the Ph.D. degree in information and communication engineering from Zhejiang University, Hangzhou, China. He is currently an Associate Professor at the Department of Computer Science and Technology, Xiamen University, Xiamen, China. His research interests include blockchain, computer vision, machine learning and big data analytics.

Maozhen Li is a Professor in the Department of Engineering, Brunel University London, UK. He received the PhD from the Institute of Software, Chinese Academy of Sciences in 1997. His main research interests include high-performance computing, big data analytics, and intelligent systems with applications to smart grids, smart manufacturing and smart cities. He has over 240 research publications in these areas including 4 books. He has served over 30 IEEE conferences and is on the editorial board of a number of journals. He is a Fellow of the British Computer Society and the IET.