2025 International Conference on Intelligent Computing
July 26-29, Ningbo, China
https://www.ic-icc.cn/2025/index.php

Smart Contract Vulnerabilities Detection with Adaptive Loss Weight and Entropy Weight

Jingyuan Hu[1] and Peng Su[2(✉)] and Xuanxia Yao[3]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. hujingyuan@iie.ac.cn
[2] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China. D202310412@xs.ustb.edu.cn
[3] School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China. Kathy.yao@163.com

Abstract. Smart contract security constitutes the foundational cornerstone for ensuring the trusted operational integrity of blockchain ecosystems. In recent years, multi-task learning (MTL) architectures have been widely adopted in smart contract vulnerability detection, owing to their context-aware optimization and superior generalization capabilities compared to single-task learning (STL) frameworks. However, MTL-based approaches for smart contract vulnerability detection face two persistent challenges: (1) the negative transfer phenomenon, the mitigation of negative transfer via adaptive loss weighting remains underexplored in existing research. (2) performance degradation caused by the homogeneous contribution assumption where undifferentiated contract representations impair expert layer learning efficacy. To overcome these limitations, we propose a novel detection framework incorporating adaptive loss weight and entropy-based feature enhancement. Our dual-weighting mechanism introduces: (1) dynamic loss coefficients that automatically balance task-specific optimization objectives based on evolving learning complexity and task significance, and (2) entropy-aware attention weights that prioritize high-information contract features during expert network training. Comprehensive evaluations on real-world smart contract datasets demonstrate the framework's superior detection performance compared to three adaptive weighting baselines. Experimental results reveal significant improvements in F1-score across multiple vulnerability types, validating the effectiveness of our approach in mitigating negative transfer while maintaining robust concurrent detection capabilities.

Keywords: Smart Contract, Vulnerability Detection, Adaptive Loss Weight, Entropy Weight.

1 Introduction

As a mature paradigm in blockchain-based decentralized architectures, smart contracts have become a focal point for academic research and industrial adoption, catalyzing methodological advancements in automated trust management [1,2,3,4,5,6]. However, this immutability of the code and the high value involved make smart contracts a prime

target for cyberattacks [7]. When a vulnerability in smart contract code is identified, it will bring huge economic losses to the blockchain platform. “DAO” attack caused $60 million in economic losses to the blockchain platform in 2016 [8]. These ongoing security incidents highlight the urgent need in blockchain technology to enhance contract vulnerability assessment frameworks.

To date, numerous automated analysis frameworks have emerged to identify security weaknesses in contracts. Fuzz testing [9], dynamic analysis [10,11] and symbolic execution [12,13,14,15] are commonly used detection methods, which identify smart contract vulnerabilities through predefined rules. However, the multitude of vulnerability types may lead to predefined patterns that do not cover all potential vulnerability types. Consequently, these methods can produce false positives or false negatives, affecting their effectiveness in accurately detecting vulnerabilities [16]. Therefore, researchers have recognized the need to explore alternative approaches to reduce costs and enhance the vulnerability detection performance. Recently, deep learning-based approaches for static code analysis have shown promise.

Deep learning employs neural networks to automatically learn vulnerability pattern features from data [17,18,19,20]. Among them, compared with the single-task learning model [19,21,22,23,24,25], models based on multi-task learning are capable of concurrently detecting multiple types of vulnerabilities and possess excellent scalability [20,26,27,28]. However, existing models still face some challenges. (1) Treating the contribution of each smart contract to the expert layer equally causes the learning performance of the expert layer to decrease. (2) Negative transfer phenomenon degrades the model's generalization and overall performance. The adaptive loss weight method can mitigate the negative transfer. Existing work has shown that adaptive loss weight methods are effective in the text image field [29,30,31,32,33,34,35], but the differences between text image data and smart contract data and the characteristics of smart contract data make it difficult for the adaptive loss weight algorithm to be universal between the two. The impact of adaptive loss weights on vulnerability detection remains underexplored in existing research. Existing works support this conclusion [20,26,27,28].

To overcome these problems, we propose a model based on adaptive loss weight and entropy weight. First, information entropy is used to quantify the inherent knowledge of each smart contract, enabling discriminative weighting of their contributions to the expert layer. Then, we develop an adaptive loss weight algorithm to balance the diverse tasks in multitask learning, thereby effectively mitigating negative transfer.

This study advances the field through three principal innovations:

- We propose a novel entropy-weighted contribution allocation framework for hierarchical expert layers in smart contracts, enhancing vulnerability detection capabilities through context-aware prioritization of expert contributions.
- We propose an adaptive loss weighting algorithm informed by the intrinsic characteristics of smart contract data, aiming to mitigate negative transfer in multi-task learning while enhancing vulnerability detection performance.
- The experimental study reveals that the effectiveness of adaptive loss weight algorithms is not universal across different domains. The impact of adaptive loss weights

on vulnerability detection remains underexplored in existing research. Our work bridges this critical gap in the existing literature.

Following this introduction, the discourse proceeds along three axes. Section 2 elaborates on the proposed methodology. Section 3 systematizes the empirical validation through quantitative benchmarking. Section 4 concludes the paper by summarizing the key findings and contributions.

2 Proposed Approach

The proposed framework (Fig. 1) exhibits a tri-stage processing pipeline: **Data Processing**, **Model Building**, and **Model Training**. Next, we'll take a closer look at each module.

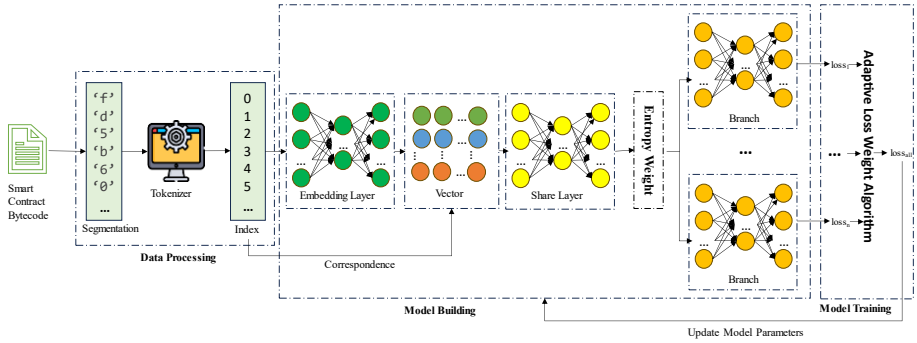

Fig. 1. Overall model flow chart.

2.1 Data Processing

Only *1%* of the existing smart contract source code is public [20], while the smart contract bytecode on the blockchain platform is public. Therefore, our proposed model takes the bytecode as input. Let the set of smart contract bytecodes be denoted as $\mathrm{SCB}_{set} = \{\mathrm{SCB}_1, \mathrm{SCB}_2, \ldots, \mathrm{SCB}_n\}$, with preprocessing conducted according to Equation (1).

$$Index_V = \mathrm{Set}(\mathrm{Tokenizer}(\mathrm{Split}(SCB_i))),\ i \epsilon n \tag{1}$$

Herein, $Index_V \epsilon \mathbb{R}^{(n,k)}$. n denotes the number of smart contract bytecodes. k denotes the number of tokens generated from the segmented instances of the smart contract bytecode. $Split(\cdot)$ denotes the process of partitioning a smart contract bytecode instance into a set of byte tokens. $Tokenizer(\cdot)$ denotes the transformation of a set of byte tokens into an index vector. $Set(\cdot)$ denotes the set of index vectors corresponding to all smart contract bytecodes.

2.2 Model Building

The general basic model architecture, depicted in Fig. 2, comprises four components: **Input Layer**, **Shared Layer**, **Entropy Weight Layer**, and **Branch Layer**.

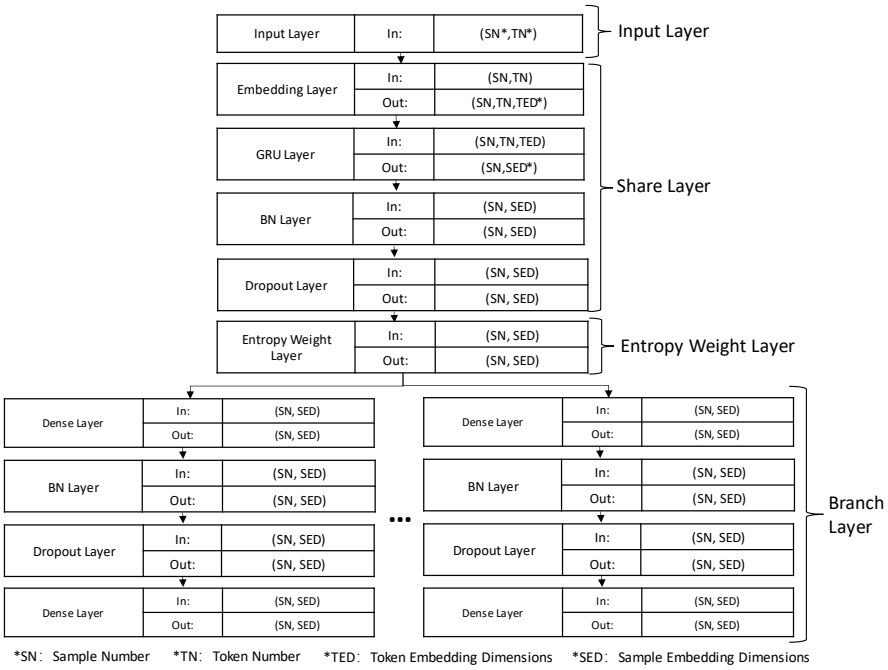

Fig. 2. Model basic structure.

Input Layer. The input layer, the first component of the model, receives external input $Index_V$.

Share Layer. The share layer, the second component of the model, learns common semantic and syntactic features in smart contracts. These common characteristics are shared by the vulnerability branch layer and are common between different vulnerability branches. The share layer consists of four key layers.

- *Embedding Layer*. The embedding layer maps each byte in the smart contract byte sequence to the vector space, as referenced in Equation (2). The embedding layer provides dual advantages for bytecode analysis: 1) Manifold-aware dimensionality reduction that projects high-dimensional bytecode sequences into compact latent representations while preserving semantic topology through neighborhood conservation; 2) Geometry-preserving encoding where syntactically similar byte fragments maintain isomorphic relationships in the vector space.

$$SCB_V = Embedding(Index_V) \tag{2}$$

Herein, $SCB_V \in \mathbb{R}^{(n,k,d)}$. d denotes the vector dimension corresponding to each byte token.

- *GRU Layer.* The GRU architecture employs gate mechanisms to autonomously extract vulnerability patterns from byte sequences, as indicated in Equation (3). We selected the Gated Recurrent Unit (GRU) [36] for our model due to its efficiency in handling sequential bytecode inputs and its smaller parameter count compared to the Long Short-Term Memory (LSTM) [37] network, which is crucial for managing limited types of new vulnerabilities and minimizing parameter load in the shared layer.

$$SCB_{V1} = GRU(SCB_V) \tag{3}$$

Herein, $SCB_{V1} \in \mathbb{R}^{(n,d1)}$. $d1$ denotes the vector dimension produced by the GRU layer from n vectors each of dimension d.

- *BN Layer.* A Batch Normalization (BN) layer is appended after the GRU layer to accelerate the training and convergence of the neural network and mitigate the problem of vanishing gradient, as mentioned in Equation (4).

$$SCB_{V2} = Batch\ Normalization(SCB_{V1}) \tag{4}$$

Herein, $SCB_{V2} \in \mathbb{R}^{(n.d1)}$. The Batch Normalization layer does not alter the vector dimensionality.

- *Dropout Layer.* A Dropout layer is added after the BN layer to prevent the neural network from overfitting.

$$SCB_{V3} = Dropout(SCB_{V2}) \tag{5}$$

Herein, $SCB_{V3} \in \mathbb{R}^{(n,d1)}$. The dropout layer does not alter the vector dimensionality.

Entropy Weight Layer. The entropy weight layer, the third component of the model, is designed to enhance the vulnerability detection capabilities of expert layer (branch layer). The amount of information contained in smart contracts with different vulnerability types is different, and the contribution of each smart contract to the task is also different. Information entropy can represent the amount of information. Information entropy is used as the entropy weight of smart contracts to measure the amount of information contained in smart contracts. The calculation of information entropy is shown in Equation (6).

$$\mathrm{H(X)}=-\sum_{x\epsilon X} P(x) log P(x) \tag{6}$$

Herein, X represents the smart contract bytecode. $H(X)$ represents the information entropy corresponding to the smart contract bytecode. x represents the byte in the bytecode. $P(x)$ represents the frequency of occurrence of a byte in the bytecode.

To stabilize the training of the model, we average the information entropy according to Equation (7). Herein, $length(X)$ represents the length of the smart contract bytecode.

$$\overline{H}(X) = \frac{H(X)}{length(X)} \tag{7}$$

The entropy weight set (SCB_H) corresponding to the smart contract bytecode set (SCB_{set}) is obtained by Equation (8).

$$SCB_H = Set\left(\bar{H}(SCB_i)\right), i\epsilon n \tag{8}$$

The weighted vector is computed using Equation (9).

$$SCB_{VH} = SCB_{V3} \times SCB_H \tag{9}$$

In the model instantiation stage, we use the **Lambda** architecture to combine the Entropy Weight Layer and the Share Layer.

Branch Layer. The branch layer, the fourth component of the model, is used to detect concurrency vulnerabilities. Each vulnerability branch is built from the same neural network and is used to detect a vulnerability type. To obtain high-performance vulnerability detection branches with a limited dataset, the model parameters should be minimized. The probability of vulnerability risk for each branch is calculated according to Equation (10).

$$\text{VD} = \text{Sigmoid}(\text{Dense}(\text{Dropout}(\text{BN}(\text{Dense}(SCB_{VH}))))) \tag{10}$$

Herein, $VD\epsilon(0,1)$. VD denotes the probability that a smart contract is susceptible to vulnerabilities.

We focus on two types of vulnerability and build two branches of vulnerability. The proposed model can detect two types of vulnerabilities concurrently. This model has good scalability. When a new type of vulnerability is discovered, we just expand the new vulnerability branch after the shared layer to fine-tune the model.

2.3 Model Training

We design an adaptive loss weight algorithm based on Task Difficulty Rate (TDR) to mitigate negative transfer during model training. The TDR algorithm is presented in **Algorithm 1**, with its process described as follows.

Algorithm 1 TDR Algorithm

1. for each epoch e do
2. for each batch data i do
3. Get each task loss L_t
4. Store first batch loss $L_{(t,0)}$
5. for each task t do
6. $w_t = (\frac{L_t}{L_{(t,0)}})^{\alpha} / \sum_{t=1}^{T} (\frac{L_t}{L_{(t,0)}})^{\alpha}$
7. end for
8. $Loss = \sum \omega_t \times L_t$
9. Use Loss to update model parameters.
10. end for
11. end for

1. Store the loss value of each task under the initial batch data.

2. Calculate the loss weight of each task under the current batch of data. The calculation of loss weight refers to Equation (11) and Equation (12). L_t indicates the loss value of the task t. $L_{(0,t)}$ indicates the initial loss value of task t. T indicates the number of tasks. α indicates a constant value. w_t indicates the loss weight of the task t.

$$\gamma_t = \frac{L_t}{L_{(0,t)}} \tag{11}$$

$$\omega_t = \frac{(\gamma_t)^\alpha}{\sum_{t=1}^{T} \gamma_t{}^\alpha} \tag{12}$$

3. The model's overall loss is calculated based on the task's loss weight and loss value, which is used to update the model parameters.

Time complexity analysis. Assume that E is the number of $epoch$, BS is the number of $Batch\ Size$, and T is the number of $task$. The time complexity of the TDR algorithm is $O(E \times BS \times T)$, which is consistent with that of existing algorithms such as EMA, REMA, and JAME.

2.4 Theoretical Derivation of the Collaborative Relationship between Entropy Weight and Adaptive Loss Weight

Assume that $L_i = g(H(X), \theta_i)$, where i denotes the $i-th$ task. θ_i denotes the model parameters of the $i-th$ task. L_i denotes the loss of the $i-th$ task. As $H(X)$ increases, L_i also increases. Different tasks exhibit varying degrees of sensitivity to $H(X)$. When $\frac{L_i}{H(x)} > \frac{L_j}{H(x)}$, it suggests that task i is more sensitive to information entropy and should be assigned a higher weight. From Equation (11) and Equation (12), it can be observed that as L increases, γ also increases, leading to a larger allocated weight. This is consistent with the preceding analysis.

3 Experiment

We conduct comprehensive evaluations to address the following Research Questions (RQs):

- **RQ1:** Whether the method improved the performance of concurrent detection of multiple vulnerabilities.
- **RQ2:** Did Algorithm 1 outperform existing adaptive loss weight methods? And were existing adaptive loss weight methods suitable for smart contract vulnerability detection?
- **RQ3:** How do the parameters in Algorithm 1 affect the experimental results?
- ***RQ4:*** Is the entropy weight method effective?
- ***RQ5:*** Does the adaptive loss weight algorithm improve model performance at the expense of time efficiency?

3.1 Datasets

- **Datasets 1:** Due to computational constraints, we selected 22033 contracts for training and 2448 for testing from the existing dataset [27]. The details of Dataset 1 are shown in Fig. 3. Dataset 1 comprises two types of vulnerability, namely Redundant Fallback Function Vulnerability (RFFV) and Integer Overflow or Underflow Vulnerability (IOUV). Dataset 1 comprises smart contract instances deployed in real-world production environments.
- **Dataset 2 and Dataset 3:** Dataset 2 and Dataset 3 [38] are used to assess the effectiveness of the entropy weight. Detailed information on Dataset 2 and Dataset 3 is shown in Fig. 3. Dataset 2 contains one type of vulnerability, Reentrancy Vulnerability (REV), while Dataset 3 contains another type, Unchecked return value Vulnerability (UCV). Dataset 2 and Dataset 3 comprise production-deployed smart contract instances collected from real-world operational environments, respectively.

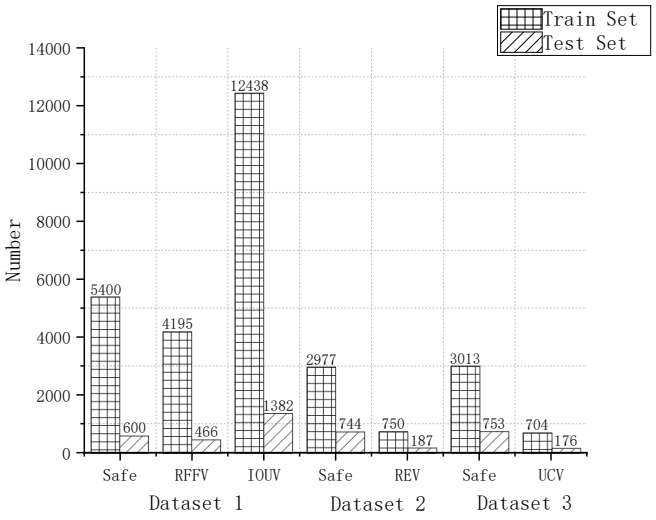

Fig. 3. Detailed information of Dataset.

3.2 Experimental Configuration

Table 1. Summary of model parameters.

Parameter name	Parameter value	Parameter name	Parameter value
Hidden Units	Embedding: 20, GRU: 128, Dropout: 0.3, Dense: [128,1]	α(TDR)	$0.1 - 0.9$
BatchSize	32	Learning Rate	10^{-3}
Optimizer	Adam	MAX Seq.Length	7500
β(EMA, REMA)	$0.1 - 0.9$	Loss Function	BCE

Hardware Configuration. All experiments were performed on server computing nodes with the CentOS operating system. The computing nodes were equipped with Intel Xeon E5-2620 v4 processors. The processor model was 2.40 GHz, 6 cores. The memory was 64GiB.

Software Configuration. The code was written in Python 3.7.3, and the third-party libraries used in the process included Tensorflow 2.11.0, Pandas 0.24.2, Numpy 1.21.6 and Keras 2.11.0.

Model Parameters. Model parameters were shown in Table 1.

3.3 Experiments

(1) Answer to RQ1: As shown in Fig. 4 and Fig. 5, we designed four sets of experiments to confirm the contribution of each submodule to the overall model.

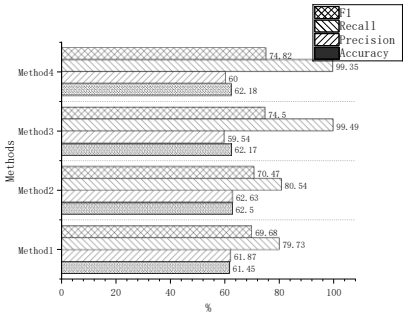

Fig. 4. Under vulnerability branch 1, ablation study results.

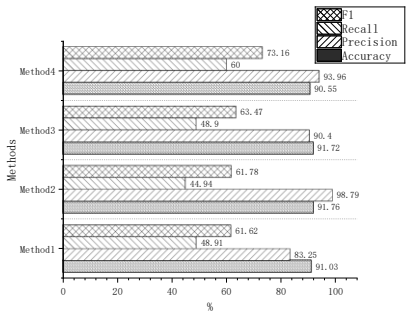

Fig. 5. Under vulnerability branch 2, ablation study results.

- *Method1:* The model removes the entropy weight layer and does not employ the Algorithm 1.
- *Method2:* The model includes an entropy weight layer, but the Algorithm 1 is not applied.
- *Method3:* The model removes the entropy weight layer, yet the Algorithm 1 is applied.
- *Method4:* The model incorporates an entropy weight layer and applies the Algorithm 1.

Vulnerability Branch 1. The experimental results demonstrate significant performance enhancements through the integration of entropy weighting and TDR Algorithm. Compared to the basic model, the entropy-weighted layer achieved respective improvements of 1.05%, 0.76%, 0.81%, and 0.79% in accuracy, precision, recall, and F1 score. Algorithm 1 exhibited more pronounced gains, particularly in recall (3.98% ↑) and the F1 score (2.02% ↑). The combined implementation yielded a notable 19.62% recall improvement and 5.14% F1-score elevation, albeit with a marginal 1.87% precision reduction. Our hybrid approach achieved optimal F1 score performance, indicating

effective model optimization through balanced multimetric enhancement. These systematic improvements substantiate the methodology's efficacy in addressing class imbalance and feature representation challenges.

Vulnerability Branch 2. Compared to the basic model, the addition of the entropy weight layer resulted in a slight decrease in recall, while significantly increasing precision (accuracy +0.73%, precision +15.54%, recall -3.97%, F1 score +0.16%). Incorporating Algorithm 1 alone slightly decreased recall but improved accuracy, precision, and F1 score (accuracy +0.2%, precision +1.98%, recall -0.01%, F1 score +0.53%). When both the entropy weight layer and Algorithm 1 were combined, the model achieved notable improvements in precision, recall, and F1 score, despite a slight decrease in accuracy (accuracy -0.48%, precision +10.71%, recall +11.09%, F1 score +11.54%). Our hybrid approach yielded the highest F1 score.

Scalability. We construct a new branch to systematically assess the scalability of the model. The experimental results indicated that Vulnerability Branch 3 had an accuracy of 62.34%, a precision of 61.32%, a recall of 89.95%, and an F1 score of 72.75%. This shows that the model proposed in this paper has excellent scalability.

(2) Answer to RQ2: To validate the applicability of existing adaptive loss weight algorithms for smart contract vulnerability detection, we experimented with three popular algorithms, namely EMA, REMA, and JAME. It could be seen from Table 2 that training models based on JAME algorithm caused the loss to disappear. To avoid the influence of entropy weights on adaptive loss weights, the following experimental models do not include the entropy weight layer. As summarized in Table 2, focusing on the comprehensive metric, when $\beta = 0.7$ and $\beta = 0.3$, the EMA algorithm and the REMA algorithm achieved the optimal performance of the overall model respectively.

Vulnerability Branch 1. As summarized in Table 2, Algorithm 1 demonstrates systematic improvements over baseline methods in key evaluation metrics. Compared to the EMA algorithm's peak performance (53.85% accuracy, 55.59% precision, 84.15% recall, 66.95% F1), Algorithm 1 achieves significant enhancements of 8.32% in accuracy, 3.95% in precision, 15.34% in recall, and 7.55% in F1-score. When evaluated against the REMA algorithm's optimal results (55.54% accuracy, 55.54% precision, 100% recall, 71.42% F1), Algorithm 1 exhibits marginal yet consistent gains of 6.63% accuracy, 4% precision and 3.08% in the F1 score, despite a negligible 0.51% recall reduction. This comparative analysis underscores Algorithm 1's robust capability to balance performance improvements across complementary metrics.

Vulnerability Branch 2. As summarized in Table 2, Algorithm 1 demonstrates significant performance improvements over baseline methods. Compared to the EMA algorithm's optimal metrics (73.71% accuracy, 31.29% precision, 65.84% recall, 42.42% F1), Algorithm 1 achieves enhancements of 18.01% in accuracy (+59.11% precision and +21.05% F1) despite a 16.94% recall reduction. When benchmarked against the REMA algorithm's peak performance (74.35% accuracy, 31.72% precision, 64.48% recall, 42.52% F1), Algorithm 1 maintains superior gains with 20.3% higher accuracy, 58.68% precision improvement, and 20.95% F1 increase, albeit with a 15.58% recall trade-off. This comparative analysis highlights Algorithm 1's enhanced classification efficacy through precision-F1 optimization, particularly notable given the inherent recall-precision dichotomy in imbalanced learning scenarios.

Table 2. Experimental results of existing methods under different parameter values.

Method	Parameter	Vulnerability Branch 1				Vulnerability Branch 2			
		Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
TDR	0.1	62.17%	59.54%	99.49%	74.50%	91.72%	90.40%	48.90%	63.47%
EMA	0.1	53.81%	55.52%	84.73%	67.08%	76.48%	21.85%	23.22%	22.51%
	0.2	54.05%	55.69%	84.58%	67.16%	76.28%	22.00%	24.04%	22.97%
	0.3	53.93%	55.60%	84.73%	67.14%	76.48%	22.27%	24.04%	23.12%
	0.4	46.70%	56.96%	16.57%	25.67%	75.36%	20.80%	24.04%	22.30%
	0.5	54.34%	55.80%	85.52%	67.54%	75.64%	21.29%	24.31%	22.70%
	0.6	55.62%	55.62%	99.49%	71.35%	44.93%	5.65%	17.48%	8.54%
	0.7 ✓	53.85%	55.59%	84.15%	66.95%	73.71%	31.29%	65.84%	42.42%
	0.8	53.65%	55.81%	79.52%	65.59%	74.75%	20.49%	24.86%	22.46%
	0.9	55.10%	55.37%	98.69%	70.94%	76.76%	22.68%	24.04%	23.34%
REMA	0.1	55.82%	56.01%	95.29%	70.55%	82.79%	41.48%	41.25%	41.36%
	0.2	53.97%	55.65%	84.42%	67.08%	77.49%	23.05%	22.67%	22.86%
	0.3 ✓	55.54%	55.54%	100.00%	71.42%	74.35%	31.72%	64.48%	42.52%
	0.4	55.66%	55.61%	99.92%	71.46%	14.79%	9.28%	54.64%	15.87%
	0.5	55.42%	57.47%	75.90%	64.41%	77.93%	23.78%	22.67%	23.21%
	0.6	53.53%	55.41%	83.64%	66.66%	75.68%	21.34%	24.31%	22.73%
	0.7	55.30%	56.53%	84.51%	67.74%	55.18%	22.32%	82.51%	35.13%
	0.8	61.29%	59.01%	99.27%	74.02%	78.61%	25.00%	22.67%	23.78%
	0.9	40.63%	44.69%	28.94%	35.13%	13.78%	10.02%	60.92%	17.21%
JAME	-	-	-	-	-	-	-	-	-

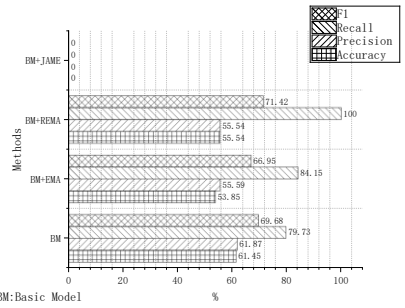

Fig. 6. Under vulnerability branch 1, experimental study of Cross-Domain adaptive loss weight algorithms in smart contract vulnerability detection.

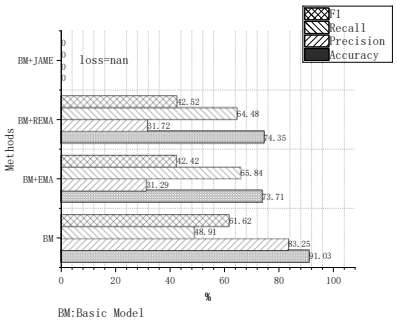

Fig. 7. Under vulnerability branch 2, experimental study of Cross-Domain adaptive loss weight algorithms in smart contract vulnerability detection.

As shown in Fig. 6 and Fig. 7, the performance of both vulnerability detection branches deteriorated after the introduction of the EMA algorithm on the basic model. Upon integrating the REMA algorithm into the basic model, while the comprehensive F1 score of the vulnerability detection branch 1 increased by 1.74%, the comprehensive F1 score of the vulnerability detection branch 2 decreased by 19.1%. After incorporating the JAME algorithm into the baseline model, there was a loss of loss during the model training process. Therefore, adaptive loss weight algorithms from other domains are not readily applicable to the field of vulnerability detection.

(3) Answer to RQ3: Our ablation study of hyperparameter α in the TDR Algorithm reveals task-specific metric tradeoffs across vulnerability detection branches (Table 3). For branch 1, $\alpha = 0.7$ maximizes accuracy (0.633), while $\alpha = 0.1$ achieves Pareto-optimal F1 scores through balanced precision-recall synergy (F1: 0.745). Branch 2 demonstrates asymmetric optimization dynamics, with $\alpha = 0.1$ simultaneously maximizing accuracy (0.914) and F1 (0.867) despite suboptimal precision. Thus, the TDR algorithm achieves superior model performance when $\alpha = 0.1$.

Table 3. Experimental results of TDR algorithm under different parameter values.

Parameter	Vulnerability Branch 1				Vulnerability Branch 2			
	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
0.1	62.17%	59.54%	99.49%	74.50%	91.72%	90.40%	48.90%	63.47%
0.2	61.21%	58.93%	99.49%	74.02%	82.39%	44.04%	72.67%	54.84%
0.3	62.21%	59.60%	99.27%	74.48%	85.28%	52.60%	70.60%	60.28%
0.4	62.62%	60.20%	96.45%	74.13%	90.43%	92.10%	38.25%	54.05%
0.5	61.21%	58.95%	99.34%	73.99%	82.27%	53.78%	72.13%	61.62%
0.6	62.41%	60.00%	97.03%	74.14%	90.19%	88.60%	38.25%	53.43%
0.7	63.30%	62.71%	83.71%	71.70%	91.23%	85.23%	48.90%	62.15%
0.8	62.58%	63.28%	77.71%	69.76%	91.07%	89.56%	44.53%	59.48%
0.9	61.01%	65.30%	63.60%	64.42%	91.03%	89.07%	44.53%	59.38%

(4) Answer to RQ4: Higher information entropy in smart contracts enriches their informational content, enhancing model learning of vulnerability patterns and improving detection performance. Ablation experiments (Fig. 4 and Fig. 5) confirm the efficacy of entropy weighting in boosting vulnerability detection capabilities. We further examine the entropy weighting mechanism through additional analytical dimensions to substantiate its effectiveness.

The contract abstract syntax tree generates multiple token sequences using different traversal methods. If the model achieves the best vulnerability detection performance on a set of token sequences with high information entropy, it indirectly substantiates the effectiveness of information entropy as an entropy weight. We conducted experimental validation on a smart contract vulnerability detection model [38] using Dataset 2 and Dataset 3, respectively.

Empirical data (Table 4) demonstrate a positive correlation between token sequence entropy levels and vulnerability detection efficacy, confirming information entropy's validity as a quantifiable metric for evaluating syntactic patterns' discriminative contributions to detection models.

Additionally, we validate the ability of entropy weights to enhance the performance of vulnerability detection models on another state-of-the-art model (VulnSense [22]) and dataset [39]. As indicated in Table 5, the method by Duy et al. demonstrated significant improvements in model performance on four evaluation metrics after incorporating entropy weights.

Table 4. Association analysis between data information entropy and model vulnerability detection performance in Dataset 2 and Dataset 3.

Dataset	Methods	Information Entropy	Accuracy	Precision	Recall	F1
Dataset2	Pysolc+DFS	5.7850✓	90.67%	81.17%	67.20%	73.53%✓
	Pysolc+BFS	5.2943	90.55%	90.16%	59.14%	71.43%
	Pysolc+SBT	4.2055	89.37%	77.36%	66.13%	71.30%
Dataset3	Pysolc+DFS	5.7925✓	90.11%	73.74%	78.49%	76.04%✓
	Pysolc+BFS	5.2145	90.43%	81.29%	67.74%	73.90%
	Pysolc+SBT	4.2050	85.91%	62.01%	76.34%	68.43%

Table 5. Experimental comparison of Duy et al.'s method with and without entropy weights.

Method	Accuracy	Precision	Recall	F1
VulnSense	57.29%	78.24%	64.03%	70.43%
VulnSense+ Entropy Weight	73.71%	78.53%	92.09%	84.77%✓

Answer to RQ5: This study quantifies the computational efficiency of our adaptive loss weight mechanism through rigorous time complexity analysis. The proposed algorithm introduces only minimal overhead from its weight derivation operations (Equation (11) and Equation (12)), requiring merely ***0.02*** seconds cumulative execution time across *689* training iterations on a $N = 22033$ smart contract dataset with batch size **32**. This represents $< 0.005\%$ of total training duration, demonstrating that our

performance gains in vulnerability detection are achieved without compromising computational tractability. The subsecond latency confirms the algorithm's suitability for large-scale smart contract analysis while maintaining a strict linear-time complexity $O(T)$ relative to task count T.

4 Conclusion

This study addresses limitations in MTL for smart contract vulnerability detection by proposing an entropy-weighted adaptive loss framework to mitigate negative transfer and challenge the homogeneous contribution assumption. We identify critical deficiencies in existing MTL approaches: 1) equal weighting of contract contributions degrades expert layer effectiveness, and 2) conventional adaptive loss weight algorithms (EMA/REMA/JAME) inadequately handle smart contracts' unique data characteristics. Our solution introduces a Task Difficulty Rate (TDR) mechanism that synergizes entropy-based feature importance with dynamic loss adjustment. Experimental validation demonstrates substantial performance gains, with TDR achieving F1-score improvements of 7.87% and 30.74% for distinct vulnerability types compared to baseline models. The results establish a new paradigm for contract-aware MTL optimization, suggesting future extensions through multimodal data fusion with adaptive weighting architectures.

Acknowledgments. This project is supported by the National Key Research and Development Project (Grant No:2022YFB2703100), and the National Natural Science Foundation of China (U22B2032).

Disclosure of Interests. The authors declare no potential conflict of interests.

References

1. Wan, Z., Guan, Z., Cheng, X.: PRIDE: A Private and Decentralized Usage-Based Insurance Using Blockchain. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), pp. 1349-1354. IEEE, New York (2018)
2. Zhang, Y.C., Zhang, J., Gao, W.Z., et al.: Distributed electrical energy systems: Needs, concepts, approaches and vision. Acta Automatica Sinica 43 (2017)
3. Christidis, K., Devetsikiotis, M.: Blockchains and Smart Contracts for the Internet of Things. IEEE Access 4, 2292-2303 (2016)
4. Abdullah, A., Md, Z., Anirban, B., et al.: Privacy-friendly platform for healthcare data in cloud based on blockchain environment. Future generation computer systems 95, 511–521 (2019)
5. Shahriar Rahman, M., Al Omar, A., Bhuiyan, M., et al.: Accountable Cross-Border Data Sharing Using Blockchain Under Relaxed Trust Assumption. In: IEEE Transactions on Engineering Management 67, 1476-1486 (2020)

6. Zheng, Z.B., Xie, S.A., Dai, H.N., et al.: An overview on smart contracts: Challenges, advances and platforms. IEEE Transactions on Engineering Management 67, 1476–1486 (2020)
7. Bartoletti, M., Pompianu, L.: An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns. In: Financial Cryptography and Data Security, pp. 494–509. Springer, Heidelberg (2017)
8. Badruddoja, S., Dantu, R., He, Y., et al.: Making Smart Contracts Smarter. In: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1-3. IEEE, New York (2021)
9. Fu, Y., Ren, M., M, F.C., et al.: Evmfuzzer: detect evm vulnerabilities via fuzz testing. In: Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, pp. 1110–1114. ACM, New York (2019)
10. Xu, J.Z., Dang, F., Ding, X., et al.: A survey on vulnerability detection tools of smart contract bytecode. In 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), pp. 94–98. IEEE, New York (2020)
11. Krupp, J., Rossow, C.: teether: Gnawing at ethereum to automatically exploit smart contracts. In: 27th USENIX Security Symposium (USENIX Security 18), pp. 1317–1333. ACM, New York (2018)
12. Nikoli´c, I., Kolluri, A., Sergey, I., et al.: Finding the greedy, prodigal, and suicidal contracts at scale. In: Proceedings of the 34th annual computer security applications conference, pp. 653–663. ACM, New York (2018)
13. Sukrit, K., Seep, G., Mohan, D., et al.: Zeus: analyzing safety of smart contracts. In: Network and Distributed System Security Symposium, pp. 1–12. San Diego (2018)
14. Mossberg, M., Manzano, F., Hennenfent, E., et al.: Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1186–1189. IEEE, New York (2019)
15. Fu, M.L., Wu, L.F., Hong, Z., et al.: A critical-path-coverage-based vulnerability detection method for smart contracts. IEEE Access 7, 147327–147344 (2019)
16. Qian, P., Liu, Z.G., He, Q.M., et al.: Towards automated reentrancy detection for smart contracts based on sequential models. IEEE Access 8, 19685–19695 (2020)
17. Gao, Z.P., Jayasundara, V., Jiang, L.X., et al.: Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding. In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 394–397. IEEE, New York (2019)
18. Zhao, H., Su, P., Wei, Y.H., et al.: Gan-enabled code embedding for reentrant vulnerabilities detection. In: Knowledge Science, Engineering and Management, pp. 585–597. Springer, Heidelberg (2021)
19. Cai, J., Li, B., Zhang, J.L., et al.: extended abstract of combine sliced joint graph with graph neural networks for smart contract vulnerability detection. In: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 851–852. IEEE, New York (2023)
20. Christoph, S., Huili, C., Hossein, F., et al.: Smarter contracts: Detecting vulnerabilities in smart contracts with deep transfer learning. In: Network and Distributed System Security Symposium, San Diego (2023)
21. Lohith, J., Anusree, K., Guru, N., et al.: TP-Detect: trigram-pixel based vulnerability detection for Ethereum smart contracts. Multimed Tools Appl 82, 36379–36393 (2023)

22. Duy, P.T., Khoa, N.H., Quyen, N.H., et al.: Vulnsense: efficient vulnerability detection in ethereum smart contracts by multimodal learning with graph neural network and language model. International Journal of Information Security 24 (2025)
23. Yuan, D.W., Wang, X.H., Li, Y., et al.: Optimizing smart contract vulnerability detection via multi-modality code and entropy embedding. Journal of Systems and Software 202(111699) (2023)
24. Sun, X.B., Tu, L.Q., Zhang, J.L., et al.: Assbert: Active and semi-supervised bert for smart contract vulnerability detection. Journal of Information Security and Applications 73(103423) (2023)
25. Li, M.L., Ren, X.X., Fu, H., et al.: Convmhsa-scvd: Enhancing smart contract vulnerability detection through a knowledge-driven and data-driven framework. In: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), pp. 578–589. IEEE, New York (2023)
26. Lutz, O., Chen, H.L., Fereidooni, H., et al.: Escort: ethereum smart contracts vulnerability detection using deep neural network and transfer learning. arXiv preprint arXiv:2103.12607 (2021)
27. Huang, J., Zhou, K., Xiong, A., et al.: Smart contract vulnerability detection model based on multi-task learning. Sensors 22(1829) (2022)
28. Zhou, K., Huang, J., Han, H., et al.: Smart contracts vulnerability detection model based on adversarial multi-task learning. Journal of Information Security and Applications 77(103555) (2023)
29. Liu, S.C., Liang, Y.Y., Gitter, A.: Loss-balanced task weighting to reduce negative transfer in multi-task learning. In: Proceedings of the AAAI conference on artificial intelligence, pp. 9977–9978. ACM, New York (2019)
30. Liu, S.K., Johns, E., Andrew J, D.: End-to-end multi-task learning with attention. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1871–1880. IEEE, New York (2019)
31. Fatemeh Salehi, R., Michael, G.: Multi-task network embedding with adaptive loss weighting. In: 2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 1–5. IEEE, New York (2020)
32. Anish, L., Essam, S., Saimourya, S., et al.: Mitigating negative transfer in multi-task learning with exponential moving average loss weighting strategies (student abstract). In: Proceedings of the AAAI Conference on Artificial Intelligence 37(13), 16246–16247 (2023)
33. Lin, B.J., Ye, F.Y., Zhang, Y., et al.: Reasonable effectiveness of random weighting: A litmus test for multi-task learning. arXiv preprint arXiv:2111.10603 (2021)
34. Chen, Z., Badrinarayanan, V., Lee, C., et al.: Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In: International conference on machine learning, pp. 794–803. PMLR (2018)
35. Ting, G., Tyler, L., Cory, S., et al.: A comparison of loss weighting strategies for multi task learning in deep neural networks. IEEE Access 7(141), 627–141632 (2019)
36. Chung, J.Y., Gulcehre, C., Cho, K.H., et al.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
37. Memory, L.: Long short-term memory. Neural computation 9, 1735–1780 (2010)
38. Chen, Y.Z., Sun, Z.Y., Gong, Z.H., et al.: Improving smart contract security with contrastive learning-based vulnerability detection. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1-11. ACM, New York (2024)
39. Gao, Z.P., Jiang, L.X., Xia, X., et al.: Checking smart contracts with structural code embedding. IEEE Transactions on Software Engineering 47(12), 2874–2891 (2021)