计算机科学
COMPUTER SCIENCE

# 基于T5模型的智能合约漏洞修复研究

焦健, 陈瑞翔, 贺强, 渠开洋, 张子怡

引用本文

焦健, 陈瑞翔, 贺强, 渠开洋, 张子怡. 基于T5模型的智能合约漏洞修复研究[J]. 计算机科学, 2025, 52(4): 362-368.

JIAO Jian, CHEN Ruixiang, HE Qiang, QU Kaiyang, ZHANG Ziyi. Study on Smart Contract Vulnerability Repair Based on T5 Model [J]. Computer Science, 2025, 52(4): 362-368.

---

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240800039

# 基于 T5 模型的智能合约漏洞修复研究

焦　健[1,2]　陈瑞翔[1]　贺　强[3]　渠开洋[3]　张子怡[1]
1 北京信息科技大学计算机学院　北京 102206
2 北京信息科技大学未来区块链与隐私计算高精尖创新中心　北京 102206
3 中国信息安全测评中心　北京 100193

**摘　要**　针对以太坊智能合约漏洞修复问题，目前的研究主要集中在人工定义模板的方法上。此方法需要开发者具备丰富的专业知识，面对复杂漏洞时修复效果较差。在 Solidity 智能合约源代码层面，围绕智能合约的漏洞修复技术开展研究。引入机器学习的漏洞修复方式，设计并实现一个 T5 模型智能合约漏洞修复系统，解决人工依赖的问题。利用数据爬虫技术和数据增强技术，构建相应 T5 模型训练数据集。利用机器学习技术，训练智能合约漏洞修复 T5 模型。通过网络爬虫构建了一个测试数据集，对所提系统进行多角度的性能评估。在合约修复准确率、gas 消耗和引入代码量等方面，与 TIPS，SGUARD 和 Elysium 等合约漏洞修复工具进行对比。实验结果表明，所提系统修复效果良好，整体性能优于其他漏洞修复工具。

**关键词**：智能合约；区块链；T5 模型；机器学习；漏洞修复
**中图分类号**　TP309

## Study on Smart Contract Vulnerability Repair Based on T5 Model

JIAO Jian[1,2]，CHEN Ruixiang[1]，HE Qiang[3]，QU Kaiyang[3] and ZHANG Ziyi[1]
1 School of Computer Science，Beijing University of Information Technology，Beijing 102206，China
2 Beijing University of Information Technology Future Blockchain and Privacy Computing High Precision and Advanced Innovation Center，Beijing 102206，China
3 China Information Security Assessment Center，Beijing 100193，China

**Abstract**　The current research on addressing vulnerabilities in Ethereum smart contracts primarily focuses on manually defined templates. This method requires developers to have extensive expertise，and its effectiveness is poor when dealing with complex vulnerabilities. This paper explores vulnerability repair techniques for smart contracts at the source code level in Solidity. By introducing a machine learning approach to vulnerability repair，we designe and implement a T5 model-based smart contract vulnerability repair system to tackle the problem of depending on manual intervention. Using data crawling and data augmentation techniques，we compile a training dataset specifically for the T5 model. The T5 model for repairing smart contract vulnerabilities is trained using machine learning techniques. A test dataset is constructed through web crawling to evaluate the system's performance from various perspectives. The system's accuracy in contract repair，gas consumption，and introduced code volume is compared with other contract vulnerability repair tools such as TIPS，SGUARD，and Elysium. Experimental results show that our system achieves good repair outcomes and overall performance superior to other vulnerability repair tools.

**Keywords**　Smart contracts，Blockchain，T5 model，Machine learning，Vulnerability repair

## 1　引言

近年来，智能合约引发的区块链安全问题引起了社会关注。2016 年，DAO 项目的智能合约存在重入漏洞，导致黑客窃取了价值 5000 万美元以太币，最终以太坊社区进行了硬分叉[1]。2017 年，Parity 多签名钱包漏洞被利用，导致价值 3000 万美元的以太币被盗[2]。2018 年，BeautyChain 的智能合约出现逻辑漏洞，造成超过 2000 万美元的损失[3]。

以上事件凸显了区块链安全的重要性。研究智能合约的安全问题具有重要意义，尤其是智能合约的安全审计开发。漏洞检测已取得一定成果[4-5]，但面对智能合约的安全问题，及时地自动修复代码是解决合约漏洞问题的有效途径[6]。

到稿日期：2024-08-06　返修日期：2024-09-26
基金项目：北京未来区块链与隐私计算高级创新中心(GJJ-23)；促进高校分类发展-大学生创新创业训练计划项目——计算机学院(5112410852)
This work was supported by the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing(GJJ-23) and Computer School of the College Student Innovation and Entrepreneurship Training Program，which Promotes the Development of Classified Universities (5112410852).
通信作者：焦健(jiaojian@bistu.edu.cn)

## 2　相关工作

本文在 Solidity 智能合约源代码层面，围绕智能合约的漏洞修复技术开展研究。本章对智能合约漏洞及其研究现状展开论述。

### 2.1　智能合约漏洞基础研究

智能合约在区块链技术中扮演着重要角色，是一种自动执行的合约代码，旨在管理、验证或执行协议。智能合约漏洞指在智能合约代码中存在的安全缺陷或错误，可能导致不良行为或资产损失[6]。智能合约的复杂性和不断变化的安全威胁使其容易受到各种漏洞的影响，这些漏洞可能是由设计不当、实现错误、逻辑缺陷或安全漏洞等原因造成的[7]。

智能合约存在许多常见的漏洞类型。重入漏洞是智能合约中常见的安全漏洞之一[8]，通常发生在合约中未正确处理资产状态更新顺序的情况下。当一个合约在处理交易时，可能会触发另一个合约的函数调用，如果这个调用又导致原合约的状态变化，而原合约未正确处理状态的更新顺序，就可能导致资产被重复提取或其他不当行为的发生。表 1 列出了一个简化的重入漏洞修复示例。修复合约通过将资金操作放在函数末尾的方式修复合约漏洞，确保在处理资金的操作之前，已经完成状态的更新。

除了重入漏洞外，智能合约还存在许多其他类型的漏洞[9-10]。本文综合当前研究人员对不同漏洞类型的关注度及其在历史上造成的损失，重点关注重入漏洞、整数溢出和随机性不足等 8 种合约漏洞，针对其进行漏洞修复研究。

表 1　重入漏洞修复样例

Table 1　Reentrancy vulnerability repair example

| 重入漏洞合约 |
| --- |
| 1. function withdraw(uint _amount) public {<br>2. require(balances[msg. sender] >= _amount,Insufficient balance");<br>3. //漏洞：资金提取后再转账，可能导致重入攻击<br>4. (bool success,)=msg. sender. call. value(_amount)("");<br>5. require(success,Transfer failed");<br>6. balances[msg. sender] -= _amount;}} |
| 修复合约 |
| 1. function withdraw(uint _amount) public {<br>2. require(balances[msg. sender] >= _amount,Insufficient balance");<br>3. //修复：先进行状态变更，再进行资金提取<br>4. balances[msg. sender] -= _amount;<br>5. (bool success,)=msg. sender. call. value(_amount)("");<br>6. require(success,Transfer failed");}} |

### 2.2　智能合约漏洞修复研究现状

智能合约漏洞检测已取得一定成果，但漏洞修复的研究仍处于起步阶段，面临诸多问题，如表 2 所列。首先，目前以模板为主要修复方式，该方式依赖于模版的人工定义，受限于开发者经验，对复杂代码逻辑的修复效果不佳。其次，机器学习在智能合约漏洞修复中的应用尚未普及，缺乏高质量的数据集，限制了其发展。因此，研究更全面的修复方法和建立相关数据集显得尤为重要，如利用机器学习模型进行合约漏洞修复。

表 2　智能合约漏洞修复总结

Table 2　Summary of smart contract vulnerability repairs

| 修复层次 | 修复工具 | 修复漏洞类型 | 修复方式 |
| --- | --- | --- | --- |
| 字节码 | EVMPatch[11] | 整数溢出；未声明函数可见性 | 模板 |
| | SMARTSHIELD[12] | 重入漏洞；整数溢出；未检查调用返回值 | 语义分析 |
| | Aroc[13] | 重入漏洞；整数溢出；未检查调用返回值 | 静态分析 |
| | Elysium[14] | 重入漏洞；整数溢出；未检查调用返回值；未声明函数可见性 | 模板、语义分析 |
| 源代码 | SCRepair[15] | 重入漏洞；整数溢出；未检查调用返回值 | 搜索 |
| | SGUARD[16] | 重入漏洞；整数溢出；利用“tx. origin”授权 | 约束求解符号执行 |
| | DeFinery[17] | 重入漏洞；整数溢出；未检查的调用范围值 | 搜索、语义分析 |
| | TIPS[18] | 重入漏洞；整数溢出；未检查的调用范围值；利用“tx. origin”授权；非预期的以太币 | 模板 |

针对 Solidity 智能合约的漏洞修复研究有字节码和源代码。字节码层面的漏洞修复方式存在不足之处，如缺乏高级语义信息、难以进行静态分析和可读性差等。相比之下，在源码层面修复合约，直观性和可读性更强，有较大的发展空间。因此，本文聚焦于在智能合约源代码层面展开技术研究。

## 3　智能合约 T5 模型漏洞修复技术研究

智能合约漏洞修复是一个复杂的问题，需要考虑代码的语义、安全性和效率。由于数据集的限制，传统的数据驱动方法面临数据稀缺和不完整的挑战。为解决这一问题，本文利用 T5 模型在大规模文本数据上进行预训练，学习丰富的语言表示和语义理解能力。这样，即使在数据稀缺的情况下，也可以通过微调为智能合约漏洞修复提供强大的基础。

T5 模型基于 Transformer 架构，采用文本到文本的框架，能够处理序列数据并在多种 NLP 任务中表现出色。通过预训练和微调，T5 模型能够学习到语法和语义信息，并生成符合要求的修复代码。因此，利用 T5 模型在大规模文本数据上预训练后进行微调，可以有效地应对智能合约漏洞修复中的数据稀缺问题，提供更准确和高效的解决方案。

### 3.1　T5 模型数据集构建

在智能合约漏洞修复领域，目前还没有用于智能合约漏洞修复的数据集。为构建数据集，采用数据爬虫技术，从互联网上收集 Solidity 智能合约代码。使用数据增强技术，对采集到的数据进行处理，以增加数据的多样性和覆盖范围。

数据爬虫技术通过编程从网页、API 和开源平台等网络资源中提取、整理和存储数据，广泛应用于各种数据驱动任务。Solidity 智能合约爬取流程如图 1 所示。Etherscan 网站[1)]提供了大量的区块链数据，如交易记录、区块信息和合约代码等。通过此合约链接规律，构建访问链接集合，利用爬虫引擎抓取页面数据，并处理反爬虫机制以确保数据获取的有效性。数据爬取后，对其进行数据清洗，去除重复值、处理缺失值和清除特殊字符等，最终存储处理后的数据，便于后续分析和应用。

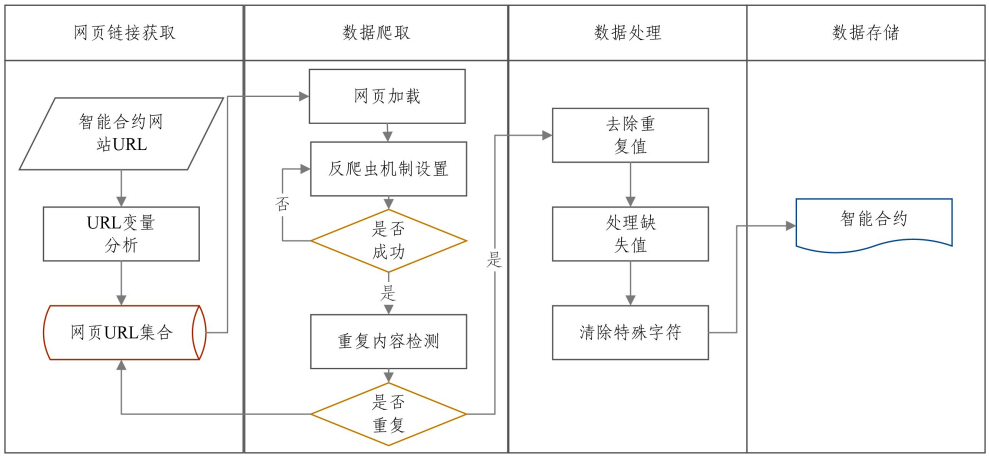


图 1 数据爬取流程

Fig. 1 Data crawling process

在 Solidity 智能合约漏洞修复任务中，实施数据增强技术是一种有效弥补微调数据集不足的方法。在通过引入变换，生成更多的训练样本，提升模型训练效果。在实践过程中用到的数据增强技术包括代码重排、变量重命名和常量调整等，具体流程如图 2 所示。

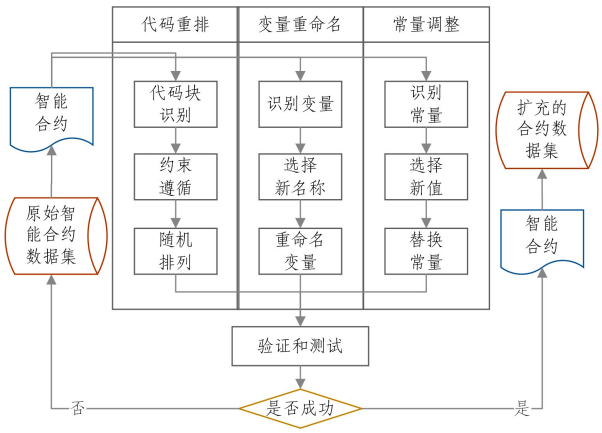


图 2 数据扩充流程

Fig. 2 Data augmentation process

### 3.2 T5 模型预训练

T5 模型在预训练过程中采用自监督学习方法。在数据输入阶段，输入合约部分内容被随机遮蔽，模型需要尝试从不完全的输入文本中预测出被遮蔽的部分。对于包含大量 Solidity 智能合约代码的预训练数据集，需要将其转换为模型能识别的格式。对于每个合约，需要使用遮蔽函数随机选择一部分代码片段将其遮蔽，并用特殊的标记，如[MASK]，使其成为模型的预测目标。

模型预训练阶段，将遮蔽处理过的 Solidity 智能合约输入到 T5 模型中。T5 模型基于 Transformer 的 Encoder-Decoder 结构，该结构包含编码器(Encoder)和解码器(Decoder)，能够有效地处理序列到序列的任务。在训练过程中，编码器负责将输入的合约代码进行编码，捕捉其中的语义和结构信息；解码器则根据编码器的输出和已知的部分合约代码，生成预测的遮蔽部分的内容。

训练时需要设置相应参数。基于文献和经验设置训练所需参数，如训练轮数、学习率和权重衰减等。在实验过程中，根据内存大小、实验结果进行调整，部分参数如表 3 所列。

表 3 T5 模型训练参数

Table 3 T5 model training parameters

| 参数名 | 参数值 |
|---|---|
| num_epochs | 5 |
| batch_size | 2 |
| learning_rate | $1\times10^{-5}$ |
| warmup_steps | 100 |
| weight_decay | 0.01 |

将模型生成的预测内容与实际原始合约代码的遮蔽部分进行比较，并使用损失函数衡量两者之间的差异。此处使用交叉熵损失函数来计算损失。交叉熵损失函数用于衡量模型生成的概率分布与实际的目标分布之间的差异，对于每个被遮蔽的部分，交叉熵损失可以通过以下计算式计算。

$$CrossEntropyLoss=-\sum_i y_i \log(\hat{y}_i) \tag{1}$$

其中，$y_i$ 是实际的目标概率分布中第 $i$ 个类别的概率，通常为 0 或 1，表示是否是该类别。$\hat{y}_i$ 是模型生成的预测概率分布中第 $i$ 个类别的预测概率。在预训练任务中，通常只有一个正确的预测，为 0 或 1，其余类别的概率为 0。假设只有一个被遮蔽的部分，损失函数可以简化为：

$$CrossEntropyLoss=-y\log(\hat{y}) \tag{2}$$

其中，$y$ 是实际的目标概率，通常为 1，表示预测目标的类别；$\hat{y}$ 是模型生成的预测概率。

损失函数衡量模型生成的预测概率 $\hat{y}$ 与实际目标概率 $y$ 之间的差异。模型的目标是最小化交叉熵损失，使得预测结果尽可能接近实际目标。

本文采用 AdamW 优化算法，如算法 1 所示。根据损失函数的梯度更新模型的参数，减小模型生成的遮蔽部分与实际原始合约代码之间的差异。通过参数更新，模型能够更准确地预测遮蔽部分的内容。

**算法 1** AdamW 优化算法

输入：初始参数 theta；学习率 alpha，一阶矩估计的指数衰减率 beta1，

[1] https://etherscan.io/address/{address}#code，其中{address}是具体的智能合约地址。

二阶矩估计的指数衰减率 beta2，数值稳定常数 epsilon，权重衰减因子 lambda_，梯度计算函数 gradient，终止条件 converged

输出：更新后的参数 theta

```
t=0,m=0,v=0
For i in converged:
    t+=1
    g_t=gradient(f,theta)
    m=beta1 * m+(1-beta1) * g_t
    v=beta2 * v+(1-beta2) *(g_t^2)
    m_hat=m /(1-beta1^t)
    v_hat=v/(1-beta2^t)
    theta=theta-alpha *(m_hat/(sqrt(v_hat)+epsilon)+lambda_ * theta)
End for
```

### 3.3　T5 模型微调

预训练结束后，利用不同漏洞的微调数据集，对 T5 模型进行微调以适应相应 Solidity 智能合约漏洞修复任务。总体流程如图 3 右半部分和算法 2 所示，包括处理漏洞和修复后的合约数据，生成对应的训练集和测试集；定义微调任务并设置超参数，采用 AdamW 优化器进行训练；使用训练损失和 BLEU(Bilingual Evaluation Understudy)分数评估模型性能，并在训练结束后导出 BLEU 分数最高的模型。微调后的 T5 模型能生成修复后的 Solidity 合约代码，提升漏洞修复的准确性和效率。

**算法 2**　SGD 优化算法

输入：初始参数 theta；学习率 alpha，训练数据集 X，标签 y，终止条件 converged

输出：更新后的参数 theta

```
theta=initial_theta
for t in converged:
    for i in range(len(X)):
        x_i=X[i]
        y_i=y[i]
        gradient=compute_gradient(theta,x_i,y_i)
        theta=theta-alpha * gradient
    End for
End for
```

T5 模型训练结束后会生成预测的修复代码。将模型生成的预测内容与实际正确合约代码进行比较，并使用损失函数衡量两者之间的差异。微调过程使用随机梯度下降算法(Stochastic Gradient Descent，SGD)，并结合梯度累积步数来更新模型参数，以减少损失函数。此过程在每个轮次中迭代执行。

在每个轮次结束后评估模型的性能。在评估过程中，生成修复代码并计算 BLEU 分数，以衡量生成的修复代码与实际修复代码之间的相似程度。

BLEU 分数是一种用于自动评估机器翻译质量的指标，其比较翻译结果与人工参考翻译之间的相似性。给定机器翻译的输出 $C$ 和参考翻译的集合 $R$，BLUE 分数的计算步骤如下：

对于每个 $n$-gram，统计 $C$ 中该 $n$-gram 在 $R$ 中出现的最大次数 $c$。令 $c$ 为 $C$ 中 $n$-gram 的计数，$r$ 为 $R$ 中 $n$-gram 的最大计数。

计算 n-gram 的精度(precision)：

$$precision=\frac{\sum_{all\cdot n\text{-}gram} c}{\sum_{all\cdot n\text{-}gram} r} \tag{3}$$

计算翻译长度惩罚项(Brevity Penalty)：

$$BP=\begin{cases}1, & \text{if}(|C|>|R|)\\ e^{\left(1-\frac{|R|}{|C|}\right)}, & \text{if}(|C|\leqslant|R|)\end{cases} \tag{4}$$

其中，$|C|$ 是机器翻译输出的词数，$|R|$ 是参考翻译的平均词数。

计算 BLEU 分数：

$$BLEU=BP\cdot\exp\left(\sum_{n=1}^{N} w_n\cdot\log(precision_n)\right) \tag{5}$$

其中，$N$ 是考虑的最大 $n$-gram 的阶数；$W_n$ 是权重，通常取 $1/N$。

BLEU 分数的计算中考虑不同阶数的 $n$-gram，且通过权重进行平均。为防止出现 0 的情况，使用对数形式进行计算。

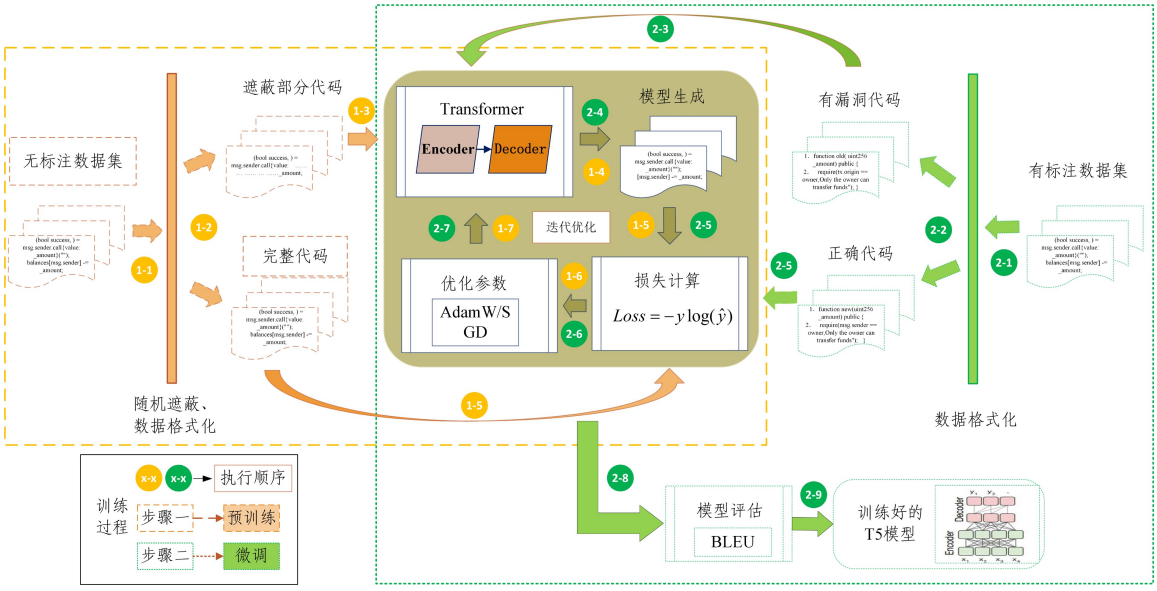


图 3　T5 模型训练流程

Fig. 3　T5 model training process

## 4 实验与分析

在 T5 模型的修复基础之上，训练针对不同类型漏洞的修复模型，构建 T5SCVR(T5 Smart Contract Vulnerability Repair)系统，并且构造实验数据集，通过实验对漏洞修复的各方面指标进行分析。系统整体修复流程如下：用户输入含有漏洞的 Solidity 智能合约，经漏洞检测子系统检测后，选择调用相应 T5 模型进行漏洞修复。修复的合约经过漏洞检测子系统验证无误后返回给用户。

在智能合约漏洞修复过程中，需要进行合约漏洞检测。然而，单一检测工具难以覆盖所有漏洞类型且准确性不足。为此，设计一个智能合约漏洞检测子系统，集成 Onsol[19]，Slither[20]和 Mythril[21]等多种漏洞检测工具，对 Solidity 智能合约进行全面检测。

### 4.1 实验数据集与评估指标

#### 4.1.1 实验数据集

在智能合约漏洞修复领域，目前缺少成熟的智能合约漏洞修复的数据集。为构建数据集，采用数据爬虫技术，用于从互联网上收集 Solidity 智能合约代码。使用数据增强技术，对采集到的数据进行处理，以增加数据的多样性和覆盖范围。

在 Solidity 智能合约漏洞修复任务中，实施数据增强技术是一种有效解决微调数据集不足的方法。通过引入变换，生成更多的训练样本，从而提升模型训练效果。在实践过程中使用到的数据增强技术包括代码重排、变量重命名和常量调整等。

预训练数据集包含 142599 条无监督智能合约数据，涵盖各种合约类型和功能，通过漏洞检测工具确保其正确性，以提高 T5 模型的语言理解和泛化能力。

微调数据集包含 4366 条有监督的智能合约数据，涵盖重入、整数溢出和短地址攻击等 8 种漏洞，每条样本数据包括漏洞合约、修复合约及相应标签。此数据集用于微调 T5 模型，使其能够学习漏洞特征和修复策略，提高漏洞修复的准确性和效率。

结合智能合约漏洞修复工具 TIPS 和 SGUARD 所使用的测试数据集，构建一个包含 187 条合约的测试数据集。该数据集涵盖 8 种 Solidity 智能合约漏洞，不同漏洞的测试合约数量因其出现频率和研究关注度的不同而有所差异。

#### 4.1.2 评估指标

为全面评估智能合约漏洞修复系统的性能，设置了多维度评估指标，并将本文提出的方法与其他常用漏洞修复工具进行对比。通过这些指标，可以深入了解本文提出的修复方法的优越性及其在实际应用中的性能表现。

具体评估指标包括：1)准确率，评估系统修复漏洞的成功率，计算式为正确修复合约数与总合约数之比；2)gas 消耗，评估修复后合约新增的 gas 消耗与原合约的消耗之比，以评估修复对程序性能的影响；3)新增代码量，分析修复后的合约新增代码行数，以评估修复引入的复杂性。

### 4.2 结果与分析

在漏洞修复准确率、gas 消耗、新增代码量方面，将 T5SCVR 与 TIPS，SGUARD，Elysium 等现有工具进行了对比分析。

#### 4.2.1 准确率

利用所构建的测试数据集对智能合约漏洞修复模型进行实验验证，结果如表 4 所列，在对 187 条智能合约进行测试时，成功修复 175 条合约，修复准确率达 93.6%。

表 4 T5SCVR 漏洞修复测试结果

Table 4 T5SCVR vulnerability repair test results

| 合约漏洞 | 漏洞数量 | 原始修复数量 | 优化后修复数量 | 优化后准确率/% |
|---|---|---|---|---|
| 重入漏洞 | 35 | 23 | 32 | 91.4 |
| 整数溢出 | 47 | 39 | 43 | 91.5 |
| 无保护的 SELFDESTRUCT 指令 | 5 | 5 | 5 | 100 |
| 短地址攻击 | 6 | 4 | 5 | 83.3 |
| 随机性不足 | 6 | 6 | 6 | 100 |
| 利用 tx.origin 授权 | 23 | 18 | 22 | 95.7 |
| 未声明函数可见性 | 5 | 5 | 4 | 100 |
| 未检查调用返回值 | 60 | 41 | 58 | 96.7 |
| 总计 | 187 | 141 | 175 | 93.6 |

通过对实验结果的进一步研究发现，利用 T5 模型对重入漏洞进行修复时，35 条测试合约中仅修复了 23 条，其中有 8 条测试合约的漏洞表现形式相同。用于微调训练的数据集未涵盖这种漏洞形式，导致模型无法进行修复。对重入漏洞数据集进行重构和微调训练，随后再次对新模型进行测试，发现更新后的模型成功修复了之前修复错误的 8 条合约。

与 TIPS 在相同的数据集上进行实验对比，TIPS 的智能合约漏洞修复准确率为 95.4%，本系统使合约漏洞修复准确率为 96.3%，优于 TIPS。进一步对测试所用数据集进行详细分析，发现其覆盖漏洞的表现形式不全。同一种漏洞的表现形式可以多种多样，表 5 列出了重入漏洞的 3 种表现形式。TIPS 定义的修复模板未包含第三种重入漏洞的修复策略，而本系统能成功修复第三种表现形式的重入漏洞。

表 5 重入漏洞不同表现形式

Table 5 Different manifestations of reentrancy vulnerabilities

```
Reentry -1:
1. function withdraw(uint256 _amount) public {
2.   (bool success,)=msg.sender.call{value:_amount}("");
3.   balances[msg.sender] -= _amount;}
Reentry -2:
1. function withdrawBalance(){
2.   if( !(msg.sender.call.value(userBalance[msg.sender])() ) ){
3.    throw;}
4.    balances[msg.sender] -= _amount;}
Reentry -3:
1. receive() external payable {
2.   (bool success,) = msg.sender.call{value:balances[msg.sender]}
     ("");
3.   require(success);}
4. function deposit() public payable {
5.    balances[msg.sender]+= msg.value;}
```

1)与 SGUARD 在相同的数据集上进行实验对比，结果如表 6 所列。

表 6　T5SCVR 与 SGUARD 的修复准确率

Table 6　Repair accuracy of T5SCVR and SGUARD

| 合约漏洞 | 漏洞数量 | T5SCVR | | SGUARD | |
|---|---|---|---|---|---|
| | | 修复数量 | 准确率/% | 修复数量 | 准确率/% |
| 整数溢出 | 22 | 20 | 90.9 | 10 | 45.5 |
| 利用“tx.origin”授权 | 17 | 16 | 94.1 | 11 | 64.7 |
| 汇总 | 39 | 36 | 92.3 | 21 | 53.8 |

SGUARD 可以修复重入漏洞、整数溢出和利用“tx.origin”授权漏洞 3 种合约漏洞，经过实验发现，SGUARD 对测试的 10 条智能合约没有成功修复，因此不对此漏洞进行实验对比，整数溢出和利用“tx.origin”授权漏洞的实验结果如表 6 所列。本文提出的系统总的准确率为 92.3%，而 SGUARD 总的准确率为 53.8%。同时，本系统的单个漏洞修复准确率也高于 SGUARD。

通过对实验结果的综合分析发现，本系统在处理大多数合约时表现出良好的修复效果，成功修复了大多数漏洞。在智能合约漏洞修复方面，本系统的表现优于 SGUARD，能够修复其不能修复的合约。

2）与 Elysium 在相同的数据集上进行实验对比，结果如表 7 所列。

表 7　T5SCVR 与 Elysium 的修复准确率

Table 7　Repair accuracy of T5SCVR and Elysium

| 合约漏洞 | 漏洞数量 | T5SCVR | | SGUARD | |
|---|---|---|---|---|---|
| | | 修复数量 | 准确率/% | 修复数量 | 准确率/% |
| 重入漏洞 | 30 | 28 | 93.3 | 19 | 63.3 |
| 整数溢出 | 17 | 16 | 94.1 | 15 | 88.2 |
| 未声明函数可见性 | 5 | 5 | 100 | 5 | 100 |
| 未检查调用返回值 | 60 | 58 | 96.7 | 47 | 78.3 |
| 汇总 | 112 | 107 | 95.5 | 86 | 76.8 |

使用包含重入、整数溢出和未检查调用返回值等漏洞的 112 条测试智能合约进行实验测试。本系统总的修复准确率为 95.5%，高于 Elysium 的 76.8%，并且在重入、整数溢出和未检查调用返回值等漏洞上的准确率也高于 Elysium。

对于重入漏洞和未检查调用返回值，Elysium 漏洞修复准确率较低。对其深入研究发现，对于某些表现形式相同的合约漏洞，Elysium 都不能成功修复。原因在于 Elysium 是模板和语义的修复技术相结合的修复工具，其修复策略的制订依赖于人工，修复策略考虑不全面。

4.2.2　gas 消耗

对修复后合约的 gas 消耗进行实验测试，并与原始合约的 gas 消耗进行比较，发现修复后的 gas 消耗与原始合约之间的差异范围极大。为更好地进行比较，将新增 gas 与原始 gas 消耗的比例进行实验对比，实验结果如图 4 所示。

通过对实验数据的详细分析，发现本系统修复后的合约新增 gas 消耗与原有合约的 gas 消耗比值平均为 8.6%，最高为 54.7%，最低为−12.9%，即与原始合约的 gas 消耗之比降低 12.9%。93%的修复后的合约的 gas 消耗与原有合约的 gas 消耗比值在 30%以下，并且该比例主要集中在(−10%，20%)这个范围内。这意味着修复后的合约在大部分情况下并未显著增加 gas 消耗。

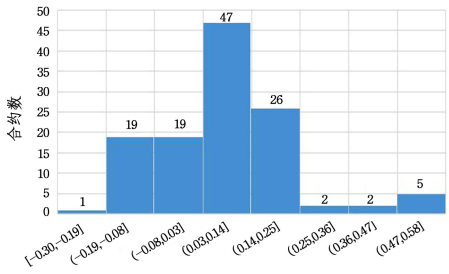


图 4　T5SCVR 新增 gas 占比

Fig. 4　Proportion of new gas added by T5SCVR

对于 SGUARD 修复的合约，计算新增的 gas 消耗占比，结果如图 5 所示，最高为 70%，最低为−23%，约 86%的合约在(−1%，43%)这个范围内。相比之下，本系统引入的 gas 消耗更少。

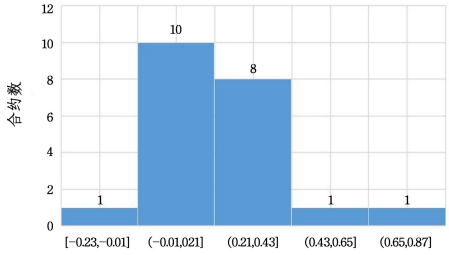


图 5　SGUARD 新增 gas 的占比

Fig. 5　Proportion of new gas added by SGUARD

4.2.3　新增代码量

图 6 给出了本系统新增代码的分布情况。与原始合约相比，修复的合约最多引入 8 行代码，最少比原来少 5 行代码，且 95%的修复合约新增行数不超过 4 行。这表明修复后的合约在大部分情况下未引入过多的新代码，有助于保持代码的简洁性和可读性。

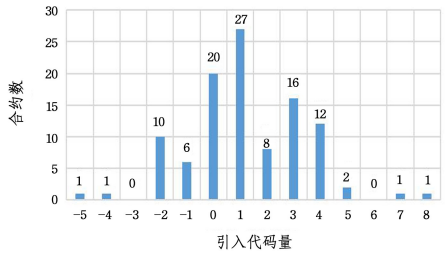


图 6　T5SCVR 引入代码量

Fig. 6　Amount of code introduced by T5SCVR

对 SGUARD 引入的代码量进行测试分析，结果如图 7 所示。

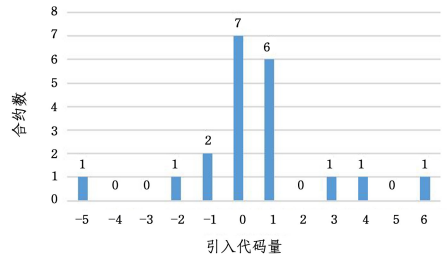


图 7　SGUARD 引入代码量

Fig. 7　Amount of code introduced by SGUARD

实验结果表明，与原始合约相比，修复的合约最多引入5行代码，最少比原来少2行代码，引入的代码量不超过5行。这表明SGUARD在修复合约时通常只需引入少量代码，能够有效地减少修复成本和代码复杂度。本系统与SGUARD在修复合约引入代码量方面性能相近。

**结束语** 本文将预训练语言模型T5应用于智能合约漏洞修复领域，以重入、整数溢出和随机性不足等8种合约漏洞为例，训练相应T5模型实现漏洞修复。实验结果显示，本系统修复类型全面，在漏洞修复准确率、时间消耗和gas消耗等方面都表现出较好的效果，修复准确率达94.6%，优于SGUARD，TIPS和Elysium等现有智能合约漏洞修复工具。

鉴于设备和时间等方面的限制，本研究仍存在改进空间。目前构建的数据集可能仅覆盖部分漏洞类型和场景，未来可进一步扩展数据集，使其涵盖更多不同类型的漏洞，提高模型的泛化能力和修复效果。同时，对漏洞修复模型进行进一步优化，探索更有效的机器学习算法和模型架构，以提高漏洞修复的准确性和效率。

## 参 考 文 献


[1] FAQIR-RHAZOUI Y, ARROYO J, HASSAN S. A comparative analysis of the platforms for decentralized autonomous organizations in the Ethereum blockchain[J]. Journal of Internet Services and Applications, 2021, 12: 1-20.

[2] GUPTA B C, KUMARN, HANDA A, et al. An insecurity study of ethereum smart contracts[C]// Security, Privacy, and Applied Cryptography Engineering: 10th International Conference. SPACE, 2020: 17-21.

[3] КОМЛЕВА Н О, ТЕРЕЩЕНКО О. Requirements for the development of smart contracts and an overview of smart contract vulnerabilities at the Solidity code level on the Ethereum platform[J]. Вісник сучасних інформаційних технологій, 2023, 6(1): 54-68.

[4] CHU H, ZHANG P, DONG H, et al. A survey on smart contract vulnerabilities: Data sources, detection and repair[J]. Information and Software Technology, 2023, 159: 107221.

[5] HE D, WU R, LI X, et al. Detection of vulnerabilities of blockchain smart contracts[J]. IEEE Internet of Things Journal, 2023, 10(14): 12178-12185.

[6] GAO C, YANG W, YE J, et al. sGuard+: Machine Learning Guided Rule-based Automated Vulnerability Repair on Smart Contracts[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(5): 1-55.

[7] KUSHWAHA S S, JOSHI S, SINGH D, et al. Systematic review of security vulnerabilities in ethereum blockchain smart contract [J]. IEEE Access, 2022, 10: 6605-6621.

[8] TAŞ R. Smart contract security vulnerabilities [J]. Erzincan University Journal of Science and Technology, 2023, 16(1): 196-211.

[9] NARAYANA K L, SATHIYAMURTHY K. Automation and smart materials in detecting smart contracts vulnerabilities in Blockchain using deep learning[J]. Materials Today: Proceedings, 2023, 81: 653-659.

[10] CHU H T, ZHANG P C, DONG H, et al. A survey on smart contract vulnerabilities: Data sources, detection and repair[J]. Information and Software Technology, 2023, 159: 107221.

[11] RODLER M, LI W, KARAME G O, et al. {EVMPatch}: Timely and automated patching of ethereum smart contracts[C]// 30th USENIX Security Symposium (USENIX Security 21). 2021: 1289-1306.

[12] ZHANG Y, MA S, LI J, et al. Smartshield: Automatic smart contract protection made easy[C]// 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2020: 23-34.

[13] JIN H, WANG Z, WEN M, et al. Aroc: An automatic repair framework for on-chain smart contracts[J]. IEEE Transactions on Software Engineering, 2021, 48(11): 4611-4629.

[14] FERREIRA TORRES C, JONKER H, STATE R. Elysium: Context-Aware Bytecode-Level Patching to Automatically Heal Vulnerable Smart Contracts[C]// Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses. 2022: 115-128.

[15] YU X L, AL-BATAINEH O, LO D, et al. Smart contract repair [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2020, 29(4): 1-32.

[16] NGUYEN T D, PHAM L H, SUN J. SGUARD: towards fixing vulnerable smart contracts automatically[C]// 2021 IEEE Symposium on Security and Privacy (SP). IEEE, 2021: 1215-1229.

[17] TOLMACH P, LI Y, LIN S W. Property-based automated repair of defi protocols[C]// Proceedings of the37th IEEE/ACM International Conference on Automated Software Engineering. 2022: 1-5.

[18] CHEN Q, ZHOU T, LIU K, et al. Tips: towards automating patch suggestion for vulnerable smart contracts[J]. Automated Software Engineering, 2023, 30(2): 31.

[19] CHEN R X, JIAO J, WANG R H. Intelligent Contract Vulnerability Detection System Based on Ontology Reasoning[J]. Computer Science, 2023, 50(10): 336-342.

[20] FEIST J, GRIECO G, GROCE A. Slither: a static analysisframework for smart contracts[C]// 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). IEEE, 2019: 8-15.

[21] SHARMA N, SHARMA S. A survey of Mythril, a smart contract security analysis tool forEVM bytecode[J]. Indian J Natural Sci, 2022, 13(75): 51003-51010.


**JIAO Jian**, born in 1978, Ph.D, professor, is a member of CCF (No. 28495M). His main research interests include network security and blockchain.

（责任编辑：何杨）