基于双模态交叉注意力机制的智能合约漏洞检测方法

陈锦富[1,2]，胡心怡[1,2]，蔡赛华[1,2]，闵玺润[1,2]
（1.江苏大学计算机科学与通信工程学院，江苏 镇江 212013；2.江苏省工业网络安全技术重点实验室，江苏 镇江 212013）

摘　要：针对智能合约漏洞检测中现有深度学习方法依赖单一模态进行特征提取、对上下文信息捕获不足导致检测准确率较低的问题，提出了一种基于双模态交叉注意力机制的智能合约漏洞检测方法，设计了特定的注意力机制，同时分析合约的源代码和字节码，实现源代码中的高级语义特征与字节码中的底层执行流程双向映射和互补增强，丰富特征表示。引入的残差连接有效地保持和传递原始特征信息，缓解深层网络训练中的梯度消失问题。在公开数据集上进行广泛测试，实验结果表明，所提方法相较基线提高了检测准确率2%以上；消融实验结果显示，跨模态特征融合和注意力机制的设计相互协同，显著提升检测性能。

关键词：智能合约；漏洞检测；深度学习；双模态；交叉注意力
中图分类号：TP311
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2025107

Smart contract vulnerability detection method based on Bi-modal cross-attention mechanism

CHEN Jinfu[1,2], HU Xinyi[1,2], CAI Saihua[1,2], MIN Xirun[1,2]
1. School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China
2. Jiangsu Key Laboratory of Security Technology for Industrial Cyberspace, Jiangsu University, Zhenjiang 212013, China

Abstract: To address the problem that existing deep learning methods for smart contract vulnerability detection rely on single-modal feature extraction and insufficient contextual information capture, leading to relatively low detection accuracy, a smart contract vulnerability detection method based on the Bi-modal cross-attention mechanism was proposed. A specific attention mechanism was designed that simultaneously analyzed both contract source code and bytecode, achieving bidirectional mapping and complementary enhancement between high-level semantic features in source code and low-level execution flows in bytecode, thereby enriching feature representation. Residual connections were introduced to effectively preserve and transmit original feature information, mitigating the vanishing gradient problem in deep network training. Extensive testing on public datasets demonstrates that the proposed method improves detection accuracy by more than 2% compared to baselines. Ablation experiments confirm that cross-modal feature fusion and the design of the attention mechanism work in synergy with each other, significantly improving the detection performance.

Keywords: smart contract, vulnerability detection, deep learning, Bi-modal, cross-attention

收稿日期：2025–03–10；**修回日期：**2025–06–03
通信作者：陈锦富，jinfuchen@ujs.edu.cn
基金项目：国家重点研发计划基金资助项目（No.2020YFB1005501）；国家自然科学基金资助项目（No.62172194, No.62202206, No.U1836116）；江苏省自然科学基金资助项目（No.BK20202001）；江苏省青蓝工程基金资助项目
Foundation Items: The National Key Research and Development Program of China (No.2020YFB1005501), The National Natural Science Foundation of China (No.62172194, No.62202206, No.U1836116), The Natural Science Foundation of Jiangsu Province (No.BK20202001), The Qinglan Project of Jiangsu Province

0　引言

区块链技术作为一种分布式账本技术，凭借其去中心化、不可篡改性、透明性和共识机制等核心特征，正在重塑全球数字经济格局[1]。智能合约作为区块链技术的关键创新应用，通过可编程性实现了合约条款的自动化执行，为金融科技、供应链管理、物联网以及数字身份认证等领域带来了显著的技术变革[2]。

然而，智能合约的固有特性，如其执行的确定性和不可逆性，使其面临着严峻的安全挑战[3]。以太坊作为首个支持图灵完备智能合约的公链平台，在推动去中心化应用（DApp, decentralized application）发展的同时，安全问题也日益凸显。其中，重入漏洞作为一种典型的漏洞类型，在 The DAO 事件中造成了约 6 000 万美元的资产损失，这一事件不仅导致了以太坊硬分叉，更引发了学术界和产业界对智能合约安全性的深入思考[4]。

当前对于智能合约漏洞检测的技术主要可以分为两大类：基于传统程序分析的方法[5-10]和基于深度学习的方法[11-17]。基于传统程序分析的方法主要使用静态程序分析和动态模糊测试等软件工程技术，通过形式化验证和预定义的漏洞特征模式进行检测，但在处理高维度状态空间和复杂交互逻辑时表现出显著的可扩展性限制[18]。基于深度学习的方法则充分利用深度神经网络在特征提取和模式识别方面的优势，在检测性能上取得了显著突破，特别是在处理语义级漏洞时展现出独特优势，但现有研究主要局限于源代码层面的分析[19]。

然而，当前大多数方法往往采用单一模态特征进行分析，这存在一些局限性。其中，仅基于源代码的分析无法捕获编译过程中的优化和变换，可能忽略编译器引入的潜在问题；仅基于字节码的分析虽能反映实际执行状态，但缺乏高级语义信息和开发者意图，难以全面理解合约的业务逻辑。此外，不同类型的漏洞在不同模态中的表现各异，单一模态分析往往导致检测盲区的产生。

为了更直观地说明多模态分析的必要性，考虑以下重入漏洞的实例。图 1 展示了一个以太坊虚拟机（EVM, Ethereum virtual machine）字节码片段，图 2 是其对应的智能合约示例。

```
1  JUMPDEST
2  PUSH1 0x00
3  …
4  SLOAD   // 加载余额
5  DUP1
6  ISZERO  // 检查余额是否足够
7  PUSH1 0x5d
8  JUMPI   // 如果余额不足，跳转
9  …
10 CALLER
11 …
12 CALL    // 执行外部调用
13 ISZERO
14 PUSH1 0x7a
15 JUMPI   // 如果调用失败，跳转
16 …
17 SSTORE
18 …
```

图 1　重入漏洞 EVM 字节码片段

```
1 function withdraw(uint256 _amount) public {
2     require(balances[msg.sender] >= _amount, "Insufficient balance.")
3     (bool success, ) = msg.sender.call{value: _amount}("");
4     require(success, "Transfer failed.");
5     balances[msg.sender] -= _amount;
6 }
```

图 2　重入漏洞智能合约示例

在字节码层面，CALL 指令与 SSTORE 指令的相对位置关系确实表明了重入风险。字节码分析能够准确反映实际执行流程和指令序列，但缺少高级语义信息，难以理解开发者原始意图。若没有源代码提供的上下文信息，难以区分这是设计意图还是意外疏忽。

相反，从源代码分析，可以识别出典型的“先调用后写入”模式，这是重入漏洞的经典特征。源代码分析提供了关键的上下文信息，帮助确认这确实是一个漏洞而非有意设计。

进一步考察时间戳依赖漏洞的例子。一个看似安全的彩票合约通常会使用多种熵源来生成随机数，并通过函数抽象和外部库调用来组织代码。在源代码分析中，如果只关注主合约而忽略外部库的实现，或者被抽象函数名所误导，例如，图 3 所示的代码使用了名为“secureRandom”的函数，很容易错过潜在的时间戳依赖风险。这是因为外部库中的函数可能在底层使用 block.timestamp，这种依赖在纯源代码分析中常被层层调用而掩盖。

```
1 function drawWinner(address[] memory players) public {
2     require(players.length > 0, "No players");
3     uint256 randomIndex = randomGenerator.secureRandom() % players.length;
4     winner = players[randomIndex];
5     //...
6 }
```

图 3　时间戳依赖漏洞智能合约示例

相比之下，在字节码层面，如图4所示，无论开发者如何封装或隐藏对时间戳的依赖，TIMESTAMP操作码的使用都会被明确地暴露出来。即使时间戳依赖被深埋在多层函数调用或外部库中，字节码分析也能够将所有调用关系展平为明确的指令序列，追踪到关键操作码的使用。

```
1  PUSH1 0x40
2  MLOAD
3  …
4  TIMESTAMP   // 获取区块时间戳 –关键操作码
5  CALLER
6  …
7  MSTORE
8  SHA3        // 计算哈希值用于随机数
9  …
10 JUMP
```

图4　时间戳依赖漏洞关键字节码

由上述2个例子可以看出，源代码和字节码对于漏洞的表征形式各有侧重。通过结合源代码与字节码的互补视角，可以显著提高漏洞检测的准确性和全面性。

智能合约字节码包含了合约的完整执行逻辑，包括编译器优化后的指令序列，能够捕获到源代码层面可能忽略的潜在漏洞，扩大漏洞检测的覆盖范围。字节码序列能够直观反映合约的执行路径和控制流，便于构建基于深度学习的漏洞检测模型，提取合约的行为特征，同时可以识别典型漏洞模式对应的字节码特征，提高检测准确性[20]。通过结合源代码分析和字节码分析，可以实现更加可靠的智能合约漏洞检测。

基于此，本文提出了一种创新的智能合约漏洞检测方法，该方法综合考虑了智能合约源代码和字节码的不同特征形式，通过源代码和字节码之间特征的相互学习，充分弥补了单一模态特征存在的局限性，从而提高了智能合约漏洞检测的效果。本文的主要贡献如下。

1) 提出了一种新颖的双模态交叉注意力机制（BCAM, bi-modal cross-attention mechanism），用于智能合约漏洞检测。该模型在统一框架下整合了智能合约的字节码和源代码2种模态数据，通过设计双向交叉注意力机制实现模态间的深度交互。双向交叉注意力机制能够使每个模态自适应地从对方模态中获取互补信息，从而生成更具判别性的多模态融合表征，有效提升了漏洞检测的性能。

2) 对传统Transformer架构进行了任务特定的改进与优化。具体而言，简化了多层编码器–解码器结构，同时保留了关键的多头注意力机制和位置编码模块，创新性地在交叉注意力层引入残差连接，实现了原始特征与交互特征的有效融合。架构的改进和优化使模型更专注于模态特征的交互与整合，更契合智能合约漏洞检测的特定需求。

3) 在公开数据集[12]上进行了系统性的实验评估，重点关注了重入漏洞和时间戳依赖漏洞2类典型漏洞，并与现有的自动化安全分析工具和基于深度学习的检测方法进行了全面比较。通过消融实验验证了模型各个组件的有效性。实验结果表明，所提出的BCAM方法在上述2类漏洞的检测任务上均实现了优于现有方法的性能。

1　相关工作

本节主要列举了现有智能合约漏洞检测工具的研究现状，简要介绍了实验中所涉及的对比方法。

1.1　传统智能合约漏洞检测方法

传统智能合约安全分析工具的发展主要建立在符号执行与形式化验证等技术基础之上[21]。

在静态分析方面，文献[5]提出的SmartCheck采用了基于XML的中间表示转换方法，结合Xpath模式匹配机制实现了精细化的漏洞识别。文献[6]提出的Oyente工具通过构建控制流图（CFG, control flow graph）并结合符号执行技术，实现了对智能合约内部逻辑的深度分析。但该工具在处理大规模合约时面临路径爆炸问题，导致检测效率降低。文献[7]提出了Securify，创新性地运用符号化分析依赖图技术，通过合规与违规模式的双向匹配实现了智能合约行为的自动化安全审计。不过，其对未知漏洞类型的识别能力受限于预定义模式的完备性。文献[8]设计的Slither分析工具首先将Solidity源代码转换为EVM字节码，继而将其转化为基于单一静态赋值（SSA, single static assignment）的专用中间表示，从而实现了对智能合约更为深入的静态分析。

在动态分析方面，文献[9]创新性地提出了面向以太坊智能合约的模糊测试框架ContractFuzzer，建立了一套科学的实际漏洞检测准则体系。该方法在测试用例生成方面缺乏针对性，可能导致低覆盖率问题。文献[10]指出Mythril工具将符号执行与具

体执行相结合，在EVM环境下执行合约字节码，通过符号执行技术实现了对程序状态空间的全面遍历。但该工具在处理复杂状态转换和大型合约时计算开销过大。

尽管现有自动化检测方法取得了显著进展，但仍面临自动化程度受限、检测效率低下、准确率有待提高等技术挑战。特别是在自动化程度方面，由于需要人工设计交互代理合约或其他干预机制，在一定程度上制约了这些方法的可扩展性与实用性。这些局限性亟待通过技术创新予以突破。

1.2　基于深度学习的智能合约漏洞检测方法

随着深度学习技术的快速迭代与日趋成熟，其在智能合约漏洞检测领域展现出显著的应用潜力。就单模态智能合约源代码而言，深度学习方法主要通过将其映射为文本序列表示或图结构表示，继而进行特征提取与分类任务。智能合约的代码表征方式具有多样性，近年来相关研究成果不断涌现。

基于序列的模型主要是将源代码转换为序列数据，并使用先进的深度学习架构进行处理。这类架构包括双向长短期记忆（BiLSTM, bidirectional long short-term memory）网络、带有注意力机制的双向长短期记忆（BiLSTM-ATT, bidirectional long short-term memory with attention mechanism）网络以及Transformer等。其中，文献[13]的研究具有代表性，其将智能合约视为合约片段，通过将变量和函数名称映射为符号名称，再通过词法分析将每个合约片段划分为一系列令牌，最后使用BiLSTM-ATT对智能合约序列进行处理来识别潜在漏洞。这种方法虽然能够很好地描述语句执行顺序，但在捕捉指令间的调用关系、数据流以及详细执行逻辑方面仍存在局限性。

为了更好地捕获代码的语法和语义信息，研究者们提出了将智能合约转化为图形结构的方法。这类方法主要利用图神经网络（GNN, graph neural network）进行漏洞检测，可以处理如代码图、语义图等多种图结构。例如，文献[15]提出的DR-GCN和TMP模型，创建了表示关键函数调用和变量的节点，并通过边来表示它们的时间执行轨迹，然后应用图卷积网络提取特征；文献[16]提出的Peculiar框架专注于使用关键数据流进行智能合约重入漏洞检测。

近年来，研究者们开始尝试更具创新性的融合方法。文献[17]结合了专家模式与深度学习，通过设计合约图模型并使用时间消息传播网络提取特征，显著提高了漏洞检测性能。文献[12]提出的跨模态相互学习框架SMS则更进一步利用源代码和字节码之间的互补信息，通过共享学习机制在只有字节码可用的情况下提升漏洞检测效果。

目前的智能合约漏洞检测方法仍面临着多重挑战。首先，在识别复杂漏洞方面，现有方法往往依赖固定规则或模式，容易产生误报和漏报，且难以覆盖所有复杂情况[22]。其次，这些方法高度依赖专家制定的硬性规则，这些规则不仅容易出错，还难以随新漏洞类型的出现而快速更新和扩展。最后，在智能合约数量快速增长的背景下，基于专家规则的方法在可扩展性和自动化程度上存在明显不足，难以有效整合分布式专家的知识[23]。

针对上述智能合约漏洞检测方法的不足，本文的研究动机如下。

1)现有智能合约漏洞检测技术虽然取得了进展，但在面对复杂的语义依赖和多样化的漏洞类型时，检测准确率仍有提升空间。

2) 单一模态特征表示存在局限性，单一模态进行特征提取和漏洞检测会导致特征表示不完整，无法全面捕获智能合约中的漏洞特征。源代码分析无法捕获编译过程中的优化和变换，字节码分析缺乏高级语义信息，两者各有局限性。

3) 现有方法对特征提取深度不足。尽管深度学习凭借其强大的学习能力在智能合约安全领域展现出良好的应用前景，但目前的深度学习方法在漏洞特征提取的深度和广度上仍明显不足，特别是在模态间信息交互方面有待加强。

为此，本文对智能合约源代码进行不同维度表征，通过引入交叉注意力机制使源代码和字节码2种模态的特征能够交互学习，从而使特征信息更加丰富，进而提高模型的检测效果。

2　BCAM方法

基于现有研究成果、理论基础与以上动机，本文提出了一种基于双模态交叉注意力机制的智能合约漏洞检测方法BCAM。该方法通过深度学习技术实现了源代码和字节码的双模态特征融合，提高了漏洞检测的准确性。

2.1 智能合约漏洞形式化

形式化定义。设智能合约源代码空间为SC，旨在提出一个自动化模型M用于函数级别的漏洞检测。对于任意函数$f \in F$，其中F为函数空间，模型输出预测标签$\hat{y} = \{0,1\}$。定义$\hat{y} = 1$表示函数f存在漏洞，$\hat{y} = 0$表示函数安全。

问题形式化。给定待检测的智能合约函数集合$F = \{f_1, f_2, \cdots, f_n\}$，其中$f_i$表示集合中第$i$个函数。定义特征提取过程$g: F \to G$将函数$f_i$映射到其对应的特征$g(f_i)$，特征提取函数$\phi: G \to R^d$将特征表示转换为$d$维特征向量。

任务建模。本文将智能合约漏洞检测建模为二分类优化问题。给定函数f_i，模型基于其图表示$g(f_i)$提取特征向量$\phi(g(f_i))$并结合训练得到的参数集θ进行预测。形式化表示为

$$\hat{y} = \text{argmax}_{c \in 0,1} P(c|\phi(g(f_i)); \theta) \tag{1}$$

其中，$P(c|\phi(g(f_i)); \theta)$表示在给定特征表示和模型参数下函数$f_i$属于类别$c$的条件概率。

本文聚焦于以太坊智能合约中最具代表性和危害性的2类漏洞：重入漏洞和时间戳依赖漏洞。这2类漏洞分别涉及外部依赖问题和合约执行流程控制，对智能合约的安全性构成重大威胁。

重入漏洞是智能合约生态系统中最具破坏性的安全漏洞之一。重入漏洞发生在合约状态更新滞后于外部调用的场景中。具体而言，当合约函数F在执行过程中调用外部合约函数E，且在E执行完成前未能及时更新关键状态变量时，攻击者可以利用这个时间窗口重复调用函数F，从而破坏合约的预期行为[24]。

时间戳依赖漏洞是另一类重要的智能合约安全隐患，其本质是合约逻辑过度依赖区块时间戳（block.timestamp）作为随机性或时序判断的来源。以太坊网络中的矿工可以在一定范围内操纵区块时间戳，这种依赖可能导致合约行为被预测或操纵[25]。

2.2 BCAM方法框架

如图5所示，本文提出的BCAM方法主要包含4个核心模块：数据预处理、双模态网络架构、交叉注意力机制、特征融合与分类。数据预处理主要用于实现源代码和字节码的特征提取，双模态网络架构用于构建并行的网络并实现源代码和字节码的标准化，交叉注意力机制实现了模态间的特征交互与增强，特征融合与分类用于完成最终的漏洞识别。

2.3 数据预处理

本文针对智能合约C的源代码S和对应的字节码B采用了不同的预处理策略，具体流程如图6所示。

对于源代码，已有的研究[26]表明，程序可以被表征为符号图，且能够保留丰富的结构和语义信息。参考已有的代码语义图，并用该图确定合约代码中的控制依赖和数据依赖关系。构建代码语义图$G_s = (V_s, E_s, T_s)$，该图包含了2种节点集$V_s = F \cup X$（F为关键函数调用节点集，X为变量节点集），以及3类边集$E_s = E_c \cup E_d \cup E_f$（$E_c$为

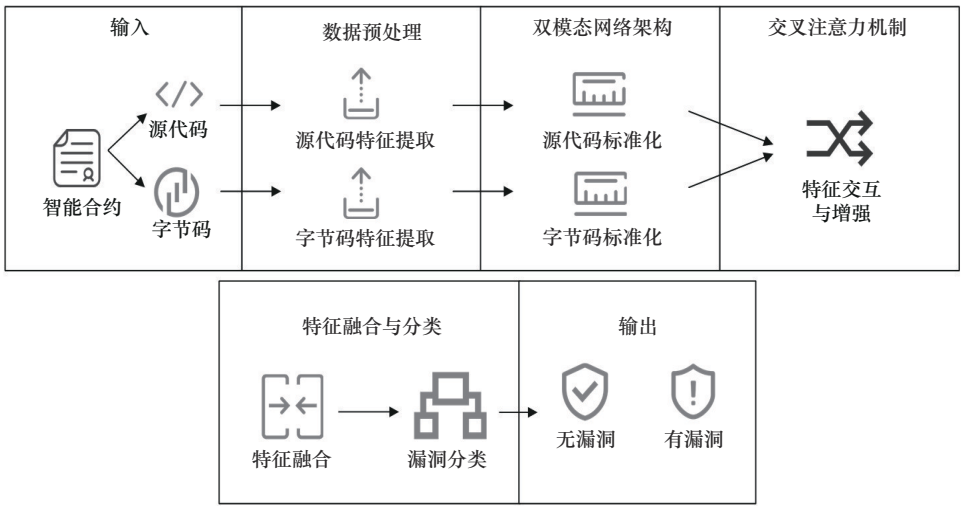


图5 BCAM方法框架

控制流边集，E_d为数据流边集，E_f为回退边集）。同时，每条边都有一个与它在代码中的顺序一致的时间顺序。

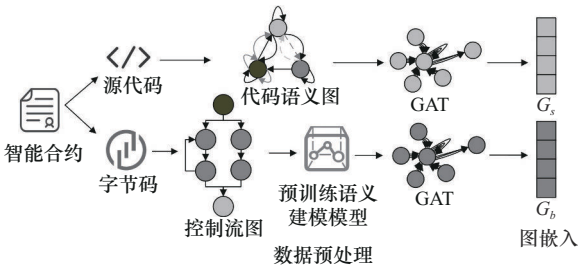


图6　数据预处理流程

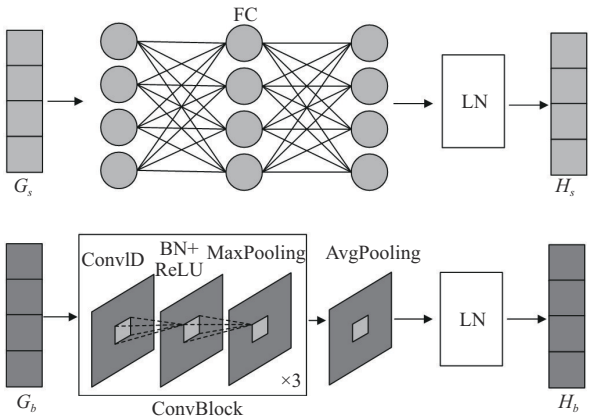


图7　双模态网络架构流程

对于字节码，使用公共编译器将给定的智能合约源代码转换为字节码，并使用现有自动化工具BinaryCFGExtractor来提取编译后字节码的控制流图。字节码的控制流图$G_b=(V_b,E_b)$由节点集V_b和边集E_b组成，$V_b=\{b_1,b_2,\cdots,b_n\}$表示由EVM指令序列组成的基本块，边集$E_b=E_{uj}\cup E_{tj}\cup E_{fj}$包含无条件跳转边集$E_{uj}$、条件为真跳转边集$E_{tj}$和条件为假跳转边集$E_{fj}$，用于捕获字节码级的控制流依赖关系。

在获得了这2类图之后，使用图注意力网络[27]学习源代码和字节码的高级图语义嵌入。具体来说，对源代码的代码语义图和字节码的控制流图分别采用图注意力网络进行处理，该过程包含消息传播和聚合2个阶段。在消息传播阶段，信息沿着图的边按照代码中的时序顺序进行传递，每个节点通过聚合其邻居节点的信息来更新自身的隐藏状态，同时使用注意力机制来计算不同邻居节点的重要性权重，这确保了模型能够识别并关注更重要的依赖关系。在聚合阶段，在完成所有边的遍历后，将所有参与节点的隐藏状态通过多层感知机进行转换和聚合，最终得到源代码和字节码各自的固定维度图语义嵌入表示G_s和G_b。2种不同的图嵌入方法允许模型同时从源代码和字节码2个层面捕获智能合约的特征，为后续的漏洞检测提供了更全面的特征表示。

2.4　双模态网络架构

在获得了数据预处理的2种模态的高级图嵌入之后，本节设计了一种新的双模态交叉注意力的网络架构，以平等对待字节码和源代码2个模态，针对源代码和字节码的不同特点，本文设计了不同的并行处理策略，具体流程如图7所示。

针对源代码的高级图嵌入，源代码模态分支利用全连接层（FC）和层归一化（LN, layer normalization）处理实现代码表示的高维度特征映射。这一处理过程可以表示为

$$H_s=\mathrm{LN}(\mathrm{FC}(G_s)) \tag{2}$$

该处理的一个重要作用是将源代码特征维度对齐到与字节码特征相同的维度空间，为后续的交叉注意力机制奠定基础。

针对字节码的高级图嵌入G_b，类似地，本文设计了一个语义提取器针对使用卷积块（ConvBlock, convolution block）结合最大池化（MaxPooling）和平均池化（AvgPooling）操作对字节码的高级图嵌入进行特征提取

$$H_b=\mathrm{LN}(\mathrm{AvgPooling}(\mathrm{ConvBlock}_n(G_b))) \tag{3}$$

其中，每个卷积块都包含卷积神经网络（CNN, convolutional neural network），并在每层CNN后采用批归一化（BN）、修正线性单元（ReLU）和最大池化，用于突出重要元素并且避免过拟合。

2个分支的输出特征H_s和H_b在维度上实现了自然对齐，为后续的模态交互提供了统一的特征表示空间，使2种模态的信息能够在相同的特征空间中进行有效的交互和融合。这种特征对齐的设计不仅简化了后续的交叉注意力计算，也有助于保持2种模态信息的平等性，避免了额外的维度转换开销。

2.5　基于位置编码的注意力增强

考虑到位置信息在序列数据处理中的关键作用，参考Transformer架构，在多头注意力机制中引入位置编码来保持和增强序列的位置信息[28]。具体而言，采用正余弦位置编码，通过三角函数的周期性特征为序列中的每个元素赋予独特的位置表

示。其数学表达式为

$$\mathrm{PE}(\mathrm{pos},2i)=\sin\left(\frac{\mathrm{pos}}{10\,000^{\frac{2i}{d_{\mathrm{model}}}}}\right) \tag{4}$$

$$\mathrm{PE}(\mathrm{pos},2i+1)=\cos\left(\frac{\mathrm{pos}}{10\,000^{\frac{2i}{d_{\mathrm{model}}}}}\right) \tag{5}$$

其中，pos表示序列中的位置，i表示维度索引，d_{model}是特征维度。

在提出的双模态漏洞检测框架中，位置信息的保留具有特殊的重要性。对于源代码模态，位置信息直接反映了程序语句之间的执行顺序和依赖关系；对于字节码模态，位置信息则体现为指令序列中的控制流和数据流模式。这2种模态的位置信息对于理解程序结构和发现潜在漏洞都起着关键作用。特别是在交叉注意力机制中，准确的位置表示能够帮助模型建立2种模态之间的精确对应关系，从而提升特征交互的质量。

通过将位置编码直接加入特征表示中，模型能够在进行跨模态注意力计算时充分利用序列的顺序信息。位置编码的加入不仅提升了模型对程序结构的理解能力，也显著增强了漏洞检测的准确性。

2.6　基于残差连接的多头交叉注意力机制

本节提出了一个结合位置编码的双模态交叉注意力架构，用于实现源代码和字节码特征的深度交互。该架构通过多头注意力机制和残差连接，实现了2种模态特征的互增强学习[29]。交叉注意力机制的具体流程和细节如图8所示。

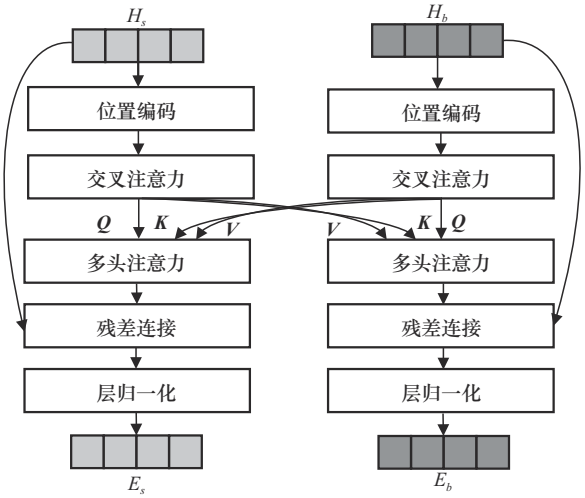


图8　交叉注意力机制的具体流程和细节

针对双模态网络架构的输出，在维度上对齐的特征H_s和H_b，首先引入位置编码来保持和增强序列的位置信息。具体而言，对于源代码流，特征嵌入H_s、位置编码PE的拼接被输入自注意力模块中建模交互，即

$$X_s=\mathrm{Concat}(H_s,\mathrm{PE}) \tag{6}$$

对于字节码流，同样将特征嵌入H_b与位置编码PE结合

$$X_b=\mathrm{Concat}(H_b,\mathrm{PE}) \tag{7}$$

在获得包含位置信息的特征表示后，使用多头注意力机制来实现跨模态特征交互。一般情况下，自注意力计算式为

$$\mathrm{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{8}$$

其中，$\boldsymbol{Q}$、$\boldsymbol{K}$和$\boldsymbol{V}$分别是查询矩阵、键矩阵和值矩阵，d_k是注意力头的维度，即$\boldsymbol{K}$的维度。

多头注意力机制的核心思想是将同一组查询、键和值通过不同的线性变换投影到h个不同的子空间中，分别进行注意力计算，然后将结果拼接起来并经过一个线性变换得到最终输出。具体来说，首先对$\boldsymbol{Q}$、$\boldsymbol{K}$、$\boldsymbol{V}$进行h次线性变换，得到h组不同的查询、键、值表示，然后对每个头进行注意力计算，最后将h个头的输出拼接并经过线性变换得到最终输出。

$$\mathrm{MultiHead}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})=\mathrm{Concat}(\mathrm{head}_1,\cdots,\mathrm{head}_h) \tag{9}$$

$$\mathrm{head}_i=\mathrm{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^{\boldsymbol{Q}},\boldsymbol{K}\boldsymbol{W}_i^{\boldsymbol{K}},\boldsymbol{V}\boldsymbol{W}_i^{\boldsymbol{V}}) \tag{10}$$

交叉注意力则使用一个模态的查询去匹配另一个模态的键值对。

具体地，源代码到字节码的注意力计算式为

$$\mathrm{Att}_{\mathrm{s2b}}=\mathrm{MultiHead}(X_s,X_b,X_b) \tag{11}$$

字节码到源代码的注意力计算式为

$$\mathrm{Att}_{\mathrm{b2s}}=\mathrm{MultiHead}(X_b,X_s,X_s) \tag{12}$$

其中，$\mathrm{Att}_{\mathrm{s2b}}$是源代码关注字节码的注意力输出，$\mathrm{Att}_{\mathrm{b2s}}$是字节码关注源代码的注意力输出。在源代码和字节码的跨模态特征融合中，多头注意力机制特别有效，这是因为不同的注意力头可以分别关注源代码和字节码中的不同特征模式，有助于建立2种模态之间的多层次对应关系，通过多个头的协同作用提高特征融合的质量和鲁棒性。

多头注意力机制的显著优势如下。首先，各注

意力头能够独立捕获输入序列的不同语义特征，从而形成多尺度特征表示，有效学习序列中的多样化依赖关系；其次，通过并行计算结构实现了功能冗余，降低了单一注意力头失效对整体性能的影响，同时集成多头输出可抑制噪声干扰，增强模型的表达能力与泛化性能；再次，各注意力头可为特定特征模式的检测器，实现对多层次关联信息的同步提取，构建更为丰富的特征表示空间；最后，多头结构支持高度并行化计算，通过降维策略有效控制计算复杂度，在维持模型表示容量的同时优化参数效率，显著提升训练与推理性能。

考虑到深度神经网络中梯度消失和特征传递的问题，在交叉注意力模块之后引入了残差连接机制。对于源代码到字节码的特征增强，表达式为

$$E_s = \mathrm{LN}(H_s + \mathrm{Att}_{\mathrm{s2b}}) \tag{13}$$

对于字节码到源代码的特征增强，表达式为

$$E_b = \mathrm{LN}(H_b + \mathrm{Att}_{\mathrm{b2s}}) \tag{14}$$

其中，E_s和E_b分别是增强后的源代码和字节码特征表示。通过残差连接，原始特征H_s和H_b可以直接与注意力输出$\mathrm{Att}_{\mathrm{s2b}}$和$\mathrm{Att}_{\mathrm{b2s}}$相加，然后经过层归一化处理。

这种设计通过残差连接保证了原始特征信息能够直接传递到后续层，同时提供了梯度反向传播的捷径来缓解梯度问题。更重要的是，它允许模型同时利用原始特征和注意力交互后的特征。层归一化的引入则帮助稳定训练过程，加快收敛速度。最终，增强后的特征E_s和E_b将用于后续的漏洞检测任务。结合了交叉注意力和残差连接的架构，模型能够有效地融合源代码和字节码的互补信息，提升特征表示的质量。

2.7　特征融合与分类

将2个模态充分交换信息后，采用特征拼接和非线性变换的方式进行特征融合，随后通过全连接层和Sigmoid激活函数完成最终的二分类任务。特征融合与分类过程如图9所示，详细介绍如下。

特征拼接。将增强后的字节码特征E_s和源代码特征E_b在特征维度上进行拼接

$$F_{\mathrm{cat}} = [E_s{:}E_b] \tag{15}$$

特征融合。通过全连接层和ReLU激活函数对拼接特征进行非线性变换

$$F_{\mathrm{fused}} = \mathrm{ReLU}(W_f F_{\mathrm{cat}} + b_f) \tag{16}$$

其中，W_f是特征融合全连接层的权重，b_f是相应的偏置项。

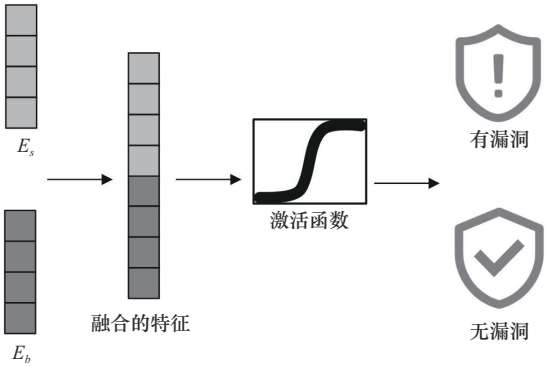


图9　特征融合与分类过程

特征拼接在特征维度上直接连接特征向量，简单地将增强后的字节码特征和源代码特征堆叠在一起。特征融合通过使用非线性变换整合拼接特征，通过应用全连接层和激活函数，创建了一个统一的表示，用于捕捉不同模态之间的关系。

二分类。使用全连接层将融合特征映射到二维空间，并通过Sigmoid函数得到最终的漏洞预测概率。

$$P(y) = \mathrm{Sigmoid}(W_c F_{\mathrm{fuesd}} + b_c) \tag{17}$$

其中，W_c是特征融合全连接层的权重，b_c是相应的偏置项。如果$P(y) > 0.5$，样本被分类为漏洞代码；否则，样本被分类为良性代码。

2.8　BCAM_Alg算法

BCAM方法的核心算法BCAM_Alg（BCAM algorithm）如算法1所示。该算法以智能合约源代码为输入，首先进行数据预处理获取语义图和字节码控制流图，然后利用图注意力网络提取特征，接着通过位置编码和双向交叉注意力机制实现源代码和字节码特征的深度交互，最后融合双模态特征进行漏洞检测，输出最终的漏洞标签。

算法1　BCAM_Alg算法

输入　智能合约源代码S

输出　漏洞标签$y \in \{0,1\}$

1) 进行数据预处理：获取智能合约源代码S的语义图G_s和字节码B；根据字节码B获取控制流图G_b

2) 使用图注意力网络对G_s和G_b进行特征提取。

3) 计算位置编码：

for pos ∈ [1,sequence.length] do

4) 利用式(4)计算得到PE[pos,2i]

5) 利用式(5)计算得到PE[pos,2i + 1]

6) end for

7) 位置编码增强：

利用式(6)计算得到X_s，利用式(7)计算得到X_b

8) for 每个注意力头$i \in [1,h]$ do

9) 计算源代码到字节码的注意力得分

获取查询矩阵$\boldsymbol{Q}_s = X_s W_i^{\boldsymbol{Q}}$

获取键矩阵$\boldsymbol{K}_b = X_b W_i^{\boldsymbol{K}}$

获取值矩阵$\boldsymbol{V}_b = X_b W_i^{\boldsymbol{V}}$

计算$\text{head}_i^{\text{s2b}} = \text{softmax}\left(\dfrac{\boldsymbol{Q}_s \boldsymbol{K}_b^{\text{T}}}{\sqrt{d_k}}\right)\boldsymbol{V}_b$

10) 计算字节码到源代码的注意力得分

获取查询矩阵$\boldsymbol{Q}_b = X_b W_i^{\boldsymbol{Q}}$

获取键矩阵$\boldsymbol{K}_s = X_s W_i^{\boldsymbol{K}}$

获取值矩阵$\boldsymbol{V}_s = X_s W_i^{\boldsymbol{V}}$

计算$\text{head}_i^{\text{b2s}} = \text{softmax}\left(\dfrac{\boldsymbol{Q}_b \boldsymbol{K}_s^{\text{T}}}{\sqrt{d_k}}\right)\boldsymbol{V}_s$

11) end for

12) 合并多头注意力结果

$\text{Att}_{\text{s2b}} = \text{Concat}(\text{head}_1^{\text{s2b}},\cdots,\text{head}_h^{\text{s2b}})$

$\text{Att}_{\text{b2s}} = \text{Concat}(\text{head}_1^{\text{b2s}},\cdots,\text{head}_h^{\text{b2s}})$

13) 应用残差连接和层归一化

利用式(13)计算得到E_s

利用式(14)计算得到E_b

14) 特征融合：

利用式(15)计算得到F_{cat}

利用式(16)计算得到融合的特征F_{fuesd}

15) 利用式(17)计算得到实验结果y

16) 返回检测结果y

3　实验

本节对本文提出的BCAM方法进行了广泛的评估，主要针对以下问题设计实验。

RQ1：与最先进的漏洞检测方法相比，本文提出的模型在重入漏洞和时间戳依赖漏洞上的检测效果是否更好？

RQ2：交叉注意力机制与特征融合是否起到了增强漏洞检测的效果？

RQ3：位置编码与残差连接的机制是否在模型中起到了增强检测效果的作用？

3.1　实验设置

1) 数据集

本文选取了公共数据集作为实验数据集[12]，数据集详情如表1所示，主要对重入漏洞和时间戳依赖漏洞进行实验。数据来源于3个渠道：以太坊平台（占比超过96%）、GitHub代码库以及分析智能合约的博客文章。

表1　　数据集详情

漏洞类型	有漏洞/个	无漏洞/个
重入漏洞	701	2 505
时间戳依赖漏洞	3 368	6 285

数据集包含了从42 910个智能合约中收集的相关函数。对于重入漏洞，680个合约中共有701个存在该漏洞的函数；对于时间戳依赖漏洞，2 242个合约中共有3 368个存在该漏洞的函数。实验选择了80%作为训练集，剩下的20%作为测试集。每个实验重复20次，记录平均结果。

2) 实验环境

实验是在配置了包括2个20核40线程CPU、128 GB RAM、2个Nvidia RTX3090显卡的服务器上进行的。服务器的操作系统为Ubuntu 20.04 LTS。参数设置如下，模型采用Adam优化器，学习率设置为0.001，隐藏层大小设置为256，epoch设置为50，batch size设置为64，weight decay设置为0.000 1。

3) 评价指标

本文根据准确率（Acc，Accuracy）、精确率（Pre，Precision）、召回率（Re，Recall）和F1值来评估方法的性能。其中，TP为模型预测为正类且实际也为正类的样本数，TN为模型预测为负类且实际也为负类的样本数，FP为模型预测为正类但实际为负类的样本数，FN为模型预测为负类但实际为正类的样本数。准确率、精确率、召回率和F1的计算公式如下所示。

准确率：模型预测正确的样本数占总样本数的比例，即

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{18}$$

精确率：模型预测为正类的样本中实际上是真正的正类的比例，即

$$\mathrm{Pre} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{19}$$

召回率：模型正确找到所有正类的比例，即

$$\mathrm{Re} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{20}$$

F1 值：精确率和召回率的调和平均数，用于综合考量模型在这 2 个方面的表现，即

$$\mathrm{F1} = 2 \times \frac{\mathrm{Pre} \times \mathrm{Re}}{\mathrm{Pre} + \mathrm{Re}} \tag{21}$$

3.2　对比实验与结果分析

为了验证 RQ1 这个问题，本节将 BCAM 模型与自动化检测工具（Smartcheck[5]、Oyente[6]、Mythril 和 Slither[10]）和深度学习方法（Rechecker[13]、GCN、TMP[15]、AME[17]、Peculiar[16]、SMS 和 DMT[12]、SCVHunter[14]）在重入漏洞和时间戳依赖漏洞检测上进行了对比实验，详细的实验结果如表 2 和表 3 所示。

表 2　重入漏洞检测对比实验结果

方法	Acc	Re	Pre	F1
Smartcheck	56.66%	16.21%	43.73%	24.74%
Oyente	67.60%	63.94%	46.15%	50.96%
Mythril	64.84%	78.07%	41.87%	57.31%
Slither	70.70%	75.85%	77.66%	76.37%
Rechecker	69.80%	69.61%	70.57%	69.68%
GCN	76.44%	73.54%	76.91%	75.67%
TMP	77.97%	77.40%	76.74%	77.02%
AME	78.12%	77.07%	79.07%	78.47%
Peculiar	92.37%	89.21%	89.96%	89.38%
SMS	83.85%	77.48%	79.46%	78.46%
DMT	89.42%	81.06%	83.62%	82.32%
SCVHunter	94.56%	90.38%	88.75%	89.55%
BCAM	92.92%	89.13%	87.28%	88.52%

由表 2 可知，在重入漏洞检测方面，BCAM 的准确率达到 92.92%，高于其他自动化检测工具，至少提升了 20%；与深度学习方法相比，优于 Rechecker(69.80%)、GCN(76.44%)、TMP(77.97%) 和 AME(78.12%)等。虽然 SCVHunter 表现出了优异的性能，其与 BCAM 相比在各项指标上略有优势，但两者性能差距相对较小；Peculiar 也达到了较高的准确率，但其仅支持重入漏洞的检测。从召回率和精确率角度分析，BCAM 的召回率达到 89.13%，精确率为 87.28%，F1 值为 88.52%，相比大部分方法都有明显提升。特别是与 SMS（召回率为 77.48%，精确率为 79.46%）和 DMT（召回率为 81.06%，精确率为 83.62%）相比，BCAM 在避免漏报和控制误报方面都表现得更加出色，表明 BCAM 能够更好地平衡检测的全面性和准确性。

表 3　时间戳依赖漏洞检测对比实验结果

方法	Acc	Re	Pre	F1
Smartcheck	48.75%	79.91%	49.06%	61.58%
Oyente	67.15%	60.23%	63.78%	62.36%
Mythril	61.21%	51.72%	55.30%	54.51%
Slither	69.37%	65.95%	70.47%	68.79%
Rechecker	—	—	—	—
GCN	73.79%	81.06%	78.63%	79.11%
TMP	80.29%	78.69%	77.57%	77.87%
AME	79.60%	76.96%	80.19%	77.15%
Peculiar	—	—	—	—
SMS	89.77%	91.09%	89.15%	90.11%
DMT	94.58%	96.39%	93.60%	94.97%
SCVHunter	91.20%	86.89%	84.58%	85.77%
BCAM	96.13%	97.27%	95.09%	96.18%

由表 3 可知，在时间戳依赖漏洞检测方面，BCAM 方法的准确率达到 96.13%，召回率为 97.27%，精确率为 95.09%，F1 值为 96.18%，领先于传统自动化检测工具，超越了其他支持此类检测的深度学习方法。与 SCVHunter 相比，BCAM 的表现要更好；相比性能较好的 DMT（准确率为 94.58%，F1 值为 94.97%），BCAM 仍然实现了提升。BCAM 在时间戳依赖漏洞上展现出更好的结果，体现了 BCAM 模型更强的通用性。

从各项指标的领先优势来看，BCAM 在复杂场景下的检测能力较强，不仅能够很好地避免漏报，还能将误报率控制在较低水平，这表明它在时间戳依赖漏洞这一类型上具有检测优势，能够更全面、更准确地识别该类漏洞。

基于表2和表3中的数据，BCAM比传统自动化检测工具和深度学习方法在重入漏洞和时间戳依赖漏洞这2类漏洞检测上都具有明显的优势。BCAM在这2类的漏洞检测中都保持了较高的指标，性能波动较小，各项指标都在85%以上，显示出很好的可靠性和泛化能力。

传统智能合约漏洞检测工具存在有限的检测精度，主要因为部分工具依赖低级指令信息进行分析或未能充分提取这些指令信息。同时，这些工具缺乏从源代码中学习高级语义信息的能力，导致检测准确性未达到理想水平。字节码等底层信息的获取依然存在挑战，进一步凸显了整合源代码高级语义的必要性，以实现更为全面且精确的漏洞识别机制。

虽然SCVHunter在重入漏洞检测上展现出了较好的表现，但BCAM在2类漏洞的检测上都保持了高水平表现，特别是在时间戳依赖漏洞检测上，充分证明了BCAM模型双模态架构和多维特征融合策略的有效性与适应性。BCAM方法采用交叉注意力机制与残差连接、位置编码等多维架构融合策略，其双模态交叉注意力网络的检测效能充分验证了源代码与字节码特征交互学习及信息融合的有效性。残差连接的引入有效保持了原始特征信息的完整性，并显著缓解了深层网络训练过程中的梯度消失问题，从而提升了模型的整体表现。对于重入漏洞这类高度依赖上下文信息的特定漏洞类型，考虑到函数调用序列对漏洞形成具有决定性影响，本文引入的位置编码机制增强了模型对序列依赖关系的理解能力，进而提高了重入漏洞的检测精确率。

3.3　消融实验与结果分析

针对RQ2，本节首先对交叉注意力机制和特征融合对检测效果的影响展开讨论。

基于表4和表5的消融实验数据，可以清晰地看到交叉注意力机制和特征融合策略对模型性能的影响。其中，Sourcecode、Bytecode表示仅使用单一模态（分别为源代码和字节码）进行处理，不应用交叉注意力和特征融合；Sourcecode_F和Bytecode_F表示在各自模态基础上应用交叉注意力进行特征增强，但不执行特征融合；BCAM代表完整的模型，结合了双模态输入、交叉注意力机制和特征融合策略。

表4　重入漏洞检测的特征融合消融实验结果

方法	Acc	Re	Pre	F1
Sourcecode	89.22%	74.81%	82.35%	78.40%
Bytecode	85.83%	77.10%	71.13%	73.99%
Sourcecode_F	92.81%	90.84%	83.22%	86.86%
Bytecode_F	90.81%	89.13%	84.72%	85.73%
BCAM	92.92%	89.13%	87.28%	88.52%

表5　时间戳依赖漏洞检测的特征融合消融实验结果

方法	Acc	Re	Pre	F1
Sourcecode	82.34%	87.97%	80.47%	84.05%
Bytecode	87.11%	89.10%	80.86%	84.76%
Sourcecode_F	94.05%	95.55%	93.66%	94.10%
Bytecode_F	94.89%	94.25%	93.65%	93.95%
BCAM	96.13%	97.27%	95.09%	96.18%

首先，仅使用单一特征（源代码或字节码）且不采用交叉注意力和特征融合的基础模型表现相对较弱。在重入漏洞检测中，仅使用源代码特征的模型准确率为89.22%，仅使用字节码特征的模型准确率为85.83%；在时间戳依赖漏洞检测中，这2个基础模型的准确率分别为82.34%和87.11%。

其次，引入交叉注意力机制（Sourcecode_F和Bytecode_F）后，模型性能得到了明显提升。以重入漏洞检测为例，使用源代码特征的模型准确率提升到92.81%，使用字节码特征的模型准确率提升到90.81%。这表明交叉注意力机制能够有效地促进不同特征之间的信息交互，帮助模型学习到更丰富的特征表示。同样的提升也体现在时间戳依赖漏洞检测中，2个模型的准确率分别提升到94.05%和94.89%。

最后，在交叉注意力的基础上进一步引入特征融合策略，模型性能达到了最优。在重入漏洞检测中，模型的准确率达到92.92%，召回率达到89.13%，精确率提升到87.28%，F1值达到88.52%；在时间戳依赖漏洞检测中，各项指标都实现了更大的提升，准确率达到96.13%，召回率为97.27%，精确率为95.09%，F1值为96.18%。这说明特征融合策略能够有效地整合经过交叉注意力增强的不同特征，从而得到更全面和鲁棒的特征表示。

消融实验的结果证明了BCAM框架中交叉注意力和特征融合策略这2个组件的有效性。交叉注意力机制能够促进不同特征之间的有效交互学习，

特征融合策略能够更好地整合这些增强后的特征，两者的结合最终带来了显著的性能提升。这也验证了模型设计的合理性，说明充分利用源代码和字节码的互补信息对提升智能合约漏洞检测的效果具有重要意义。

针对问题RQ3，本节对预训练模型、残差连接（RE, residual connection）和位置编码（PE, position encoding）进行了消融实验，实验结果如表6和表7所示。

表6　重入漏洞检测的各模块消融实验结果

方法	Acc	Re	Pre	F1
BCAM_BERT	87.36%	83.89%	78.28%	80.57%
BCAM_GCN	87.45%	83.93%	78.85%	82.38%
BCAM_GPT	88.06%	83.24%	79.59%	81.06%
BCAM_word2vec	89.27%	83.98%	84.79%	84.42%
BCAM_BERT_RE	91.54%	86.76%	81.96%	84.27%
BCAM_GCN_RE	91.15%	86.03%	81.28%	83.54%
BCAM_GPT_RE	91.07%	84.20%	82.16%	83.04%
BCAM_word2vec_RE	91.81%	88.16%	86.62%	86.05%
BCAM_BERT_ PE	91.22%	84.73%	82.22%	83.46%
BCAM_GCN_PE	89.82%	88.55%	83.33%	84.06%
BCAM_GPT_PE	91.02%	83.97%	82.09%	83.02%
BCAM_word2vec_PE	92.02%	84.05%	82.43%	82.91%
BCAM_BERT_RE_PE	91.80%	88.21%	81.86%	84.89%
BCAM_GCN_RE_PE	91.35%	86.68%	81.52%	83.96%
BCAM_GPT_RE_PE	91.52%	86.37%	82.12%	84.17%
BCAM_word2vec_RE_PE	92.92%	89.13%	87.28%	88.52%

在预训练模型的选择上，实验比较了BERT、GCN、GPT和word2vec这4种不同的预训练方法。在不添加RE和PE的基础版本中，word2vec表现最好（重入漏洞准确率为89.27%，时间戳依赖漏洞准确率为91.39%），BERT（重入漏洞准确率为87.36%，时间戳依赖漏洞准确率为92.82%）和GPT（重入漏洞准确率为88.06%，时间戳依赖漏洞准确率为86.16%）次之，GCN在时间戳依赖漏洞检测中表现较差（准确率为79.96%）。这说明相对简单的word2vec预训练方法能够较好地捕获智能合约代码的语义特征。

残差连接的引入对所有预训练模型都带来了显著提升。以BERT为例，添加RE后重入漏洞检测的准确率从87.36%提升到91.54%，时间戳依赖漏洞检测的准确率从92.82%提升到96.13%。这种提升普遍存在于其他预训练模型中，证明添加残差连接后能够有效地保持和传递原始特征信息，缓解深层网络训练中的梯度消失问题，从而提升模型的整体性能。

表7　时间戳依赖漏洞检测的各模块消融实验

方法	Acc	Re	Pre	F1
BCAM_BERT	92.82%	92.90%	90.58%	91.05%
BCAM_GCN	79.96%	86.36%	79.93%	81.28%
BCAM_GPT	86.16%	88.55%	87.07%	87.38%
BCAM_word2vec	91.39%	89.47%	96.57%	93.79%
BCAM_BERT_RE	96.13%	97.27%	95.09%	96.18%
BCAM_GCN_RE	83.73%	90.32%	81.12%	85.46%
BCAM_GPT_RE	88.58%	90.67%	88.65%	89.34%
BCAM_word2vec_RE	95.34%	93.99%	97.63%	95.69%
BCAM_BERT_ PE	95.47%	91.43%	93.97%	95.54%
BCAM_GCN_PE	81.86%	93.38%	76.76%	84.51%
BCAM_GPT_PE	86.63%	90.68%	88.68%	89.67%
BCAM_word2vec_PE	94.75%	96.05%	92.54%	93.56%
BCAM_BERT_RE_PE	96.04%	96.13%	94.06%	96.09%
BCAM_GCN_RE_PE	83.40%	90.74%	80.43%	85.26%
BCAM_GPT_RE_PE	88.92%	88.50%	90.63%	89.39%
BCAM_word2vec_RE_PE	94.89%	98.06%	92.90%	95.34%

位置编码的效果在2类漏洞检测中表现出明显差异。对于重入漏洞，添加PE后的性能普遍有所提升，例如，word2vec模型的准确率从89.27%提升到92.02%。考虑到重入漏洞的判断强依赖于合约代码的上下文信息，函数调用的先后顺序对漏洞的形成具有重要影响，可以认为位置编码能够帮助模型更好地理解这种序列依赖关系。

然而，在时间戳依赖漏洞检测中，PE的引入并未带来显著改善，有些情况下反而略有下降。由于时间戳依赖漏洞主要体现为上文对下文的单向影响，即漏洞的形成主要取决于某个时间戳操作对后续执行的影响，推测这种单向的依赖关系使完整的位置编码信息的重要性相对较低。

最终的BCAM模型采用word2vec预训练，并结合RE和PE，在重入漏洞检测中取得了最佳性能（准确率为92.92%，F1值为88.52%）。对于时间戳

依赖漏洞检测，观察到BCAM_BERT_RE的组合取得了最佳性能（准确率为96.13%，召回率为97.27%，精确率为95.09%，F1值为96.18%）。这种组合仅使用残差连接而不包含位置编码，验证了之前的分析：时间戳依赖漏洞主要依赖于上文对下文的单向影响关系，不需要完整的位置编码信息。

BERT预训练模型在添加残差连接后，相比其他预训练方法（如word2vec的准确率为94.89%，F1值为95.34%）表现更好。这可能是因为BERT的自注意力机制本身就擅长捕获序列中的长距离依赖关系，残差连接的加入进一步增强了这种能力，使模型能够更好地识别时间戳操作对后续执行的影响。这一结果也表明，对于不同类型的漏洞，最优的模型组件组合可能不同，需要根据漏洞的特性来选择合适的模型结构。

3.4 漏报分析

针对BCAM模型在重入漏洞和时间戳依赖漏洞检测上存在的漏报情况，本节对可能存在的情况进行了分析。

通过分析重入漏洞检测的漏报，发现了如下原因。首先是复杂调用序列导致的漏报，这类漏报涉及多层嵌套调用和间接重入攻击，模型难以捕捉完整的调用链路。尽管交叉注意力机制能够有效关联相近的源代码和字节码特征，但在长距离依赖关系上的表现仍有提升空间。然后是编码方式与主流模式存在较大差异产生的漏报，非常规的特征表现情况难以被检测出，导致模型产生漏报。

对于时间戳依赖漏洞检测，主要的漏报集中在以下情况。这类漏报通常将时间戳与其他条件组合使用，如地址检查、余额验证等，增加了依赖关系的复杂性。这类情况下，BCAM模型的特征融合模块表现出一定优势，但在条件组合过于复杂时仍存在识别困难。

对上述漏报情况的分析不难发现，BCAM模型在处理高度复杂或低频特征组合的样本时仍存在一定局限性。交叉注意力机制在处理相邻或直接关联的源代码-字节码对应关系上表现出色，但对于复杂组合的隐式关系依赖的效果有限。

3.5 额外漏洞类型验证

为了进一步验证BCAM模型的泛化能力，对整数溢出漏洞进行了初步分析。整数溢出漏洞在字节码和源代码中表现出不同的漏洞表现形式。在字节码层面，整数溢出表现为特定算术操作码的使用模式，如ADD、SUB等，而在源代码层面则体现为缺乏安全检查的整数运算。

初步实验结果表明，BCAM模型在整数溢出漏洞检测上也取得了较好的性能（准确率达到88.32%，高于选择的基线模型），进一步证实了多模态方法的有效性。通过结合源代码提供的变量类型和运算语义信息，以及字节码实际执行的算术操作，可以丰富漏洞的特征表示从而提高检测效果。

由于篇幅限制，本文未展示详细结果，但这部分工作将在未来的扩展研究中进行详细阐述，预期将展现更为显著的性能优势和实用价值。

3.6 真实合约环境有效性验证

为进一步验证BCAM方法在实际环境中的有效性，本文对Smartbugs数据集[30]进行了额外的验证实验。Smartbugs是一个在软件工程领域会议ICSE2020上发表的智能合约漏洞数据集，包含超过47 000个以太坊智能合约。该数据集的漏洞标签主要基于传统静态分析工具的检测结果，因此存在一定程度的漏标和误标情况。

从Smartbugs数据集中选取可能存在重入漏洞的合约子集，应用BCAM方法进行分析。实验目标是验证BCAM能否发现原有工具未能正确识别的漏洞实例。首先使用BCAM方法对目标合约进行检测，通过比较检测结果与原始标签的差异，挑选出漏标合约，最后对这些差异结果进行人工代码审计验证。

通过上述过程，BCAM方法成功发现了多个在原始数据集中标记有误的重入漏洞合约实例。已验证的一些标记有误的漏洞合约如表8所示。其中一个

表8 标记有误的漏洞合约

合约	原始标签	BCAM检测结果	漏洞函数	人工验证结果
0x0d945b31cb2fa8d6d9bf1bb2a5d603435922815e.sol	无漏洞	重入漏洞	sendEtherToNewContract()	确认存在重入漏洞
0x82458d1c812d7c930bb3229c9e159cbabd9aa8cb.sol	无漏洞	重入漏洞	batchSend()	确认存在重入漏洞
0x2f30ff3428d62748a1d993f2cc6c9b55df40b4d7.sol	无漏洞	重入漏洞	X2()	确认存在重入漏洞

漏标的典型案例如图10所示，这个案例存在典型的重入漏洞风险，主要体现在其使用了call.value()向外部地址发送以太币，并在调用外部合约后未更新状态变量。虽然该漏洞实例未被数据集正确标记，但BCAM方法能够成功识别，这证明了本文方法在检测已知类型漏洞方面的有效性和优越性，同时也揭示了仅依赖自动化检测工具生成数据集标签的局限。

```
1  function X2() public payable{
2      if(msg.value > 1 ether) {
3          msg.sender.call.value(this.balance);
4      }
5  }
```

图10　漏标的重入漏洞案例

未来，计划扩展BCAM方法到其他类型漏洞的检测与验证，进一步提高其在智能合约安全分析领域的应用价值。

4　结束语

本文提出了一种基于双模态交叉注意力的智能合约漏洞检测方法。该方法通过同时利用智能合约的源代码和字节码2种模态信息，结合交叉注意力机制实现了特征的深度交互与融合，显著提升了漏洞检测的性能。本文设计了一种新颖的双模态特征提取框架，通过图注意力网络分别处理源代码的语义图表示和字节码的控制流图表示，实现了对智能合约2种模态信息的有效编码。相比单一模态的特征提取方法，该框架能够更全面地捕获合约代码的结构和执行特征。通过引入位置编码保留序列信息，利用多头注意力实现模态间的特征交互，并通过残差连接融合原始特征与交互特征。这种设计使模型能够充分利用2种模态的互补信息，生成更具判别性的特征表示。在公开数据集上进行系统的实验评估的结果表明，BCAM方法在重入漏洞和时间戳依赖漏洞2类典型漏洞的检测任务上均取得了优异表现，各项评估指标显著优于现有的检测工具和方法。通过消融实验验证了模型各个关键组件的有效性。

未来的研究工作主要包括以下几个方向：一是对检测漏洞的种类进行扩展，不局限于重入漏洞和时间戳依赖漏洞，扩展模型对其他类型智能合约漏洞的检测，提高模型的泛化能力；二是探索更高效的特征提取和融合方法，进一步提升模型性能；三是研究模型的可解释性，为智能合约的安全审计提供更直观的分析依据。

参考文献：

[1] CHEN S J, MI H N, PING J, et al. A blockchain consensus mechanism that uses proof of solution to optimize energy dispatch and trading[J]. Nature Energy, 2022, 7: 495-502.

[2] GAI K K, ZHANG Y, QIU M K, et al. Blockchain-enabled service optimizations in supply chain digital twin[J]. IEEE Transactions on Services Computing, 2023, 16(3): 1673-1685.

[3] GAI K K, WU Y L, ZHU L H, et al. Privacy-preserving energy trading using consortium blockchain in smart grid[J]. IEEE Transactions on Industrial Informatics, 2019, 15(6): 3548-3558.

[4] ZICHICHI M, CONTU M, FERRETTI S, et al. LikeStarter: a smart-contract based social DAO for crowdfunding[C]//Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). Piscataway: IEEE Press, 2019: 313-318.

[5] TIKHOMIROV S, VOSKRESENSKAYA E, IVANITSKIY I, et al. SmartCheck: static analysis of ethereum smart contracts[C]//Proceedings of the 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Piscataway: IEEE Press, 2018: 9-16.

[6] LUU L, CHU D H, OLICKEL H, et al. Making smart contracts smarter[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2016: 254-269.

[7] TSANKOV P, DAN A, DRACHSLER-COHEN D, et al. Securify: practical security analysis of smart contracts[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2018: 67-82.

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

[9] JIANG B, LIU Y, CHAN W K. ContractFuzzer: fuzzing smart contracts for vulnerability detection[C]//Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE Press, 2018: 259-269.

[10] SHARMA N, SHARMA S. A survey of Mythril, a smart contract security analysis tool for EVM bytecode[J]. Indian Journal of Natural Products and Resources, 2022, 13(75): 39-41.

[11] 钱鹏, 刘振广, 何钦铭, 等. 智能合约安全漏洞检测技术研究综述[J]. 软件学报, 2022, 33(8): 3059-3085.
QIAN P, LIU Z G, HE Q M, et al. Smart contract vulnerability detection technique: a survey[J]. Journal of Software, 2022, 33(8): 3059-3085.

[12] QIAN P, LIU Z G, YIN Y F, et al. Cross-modality mutual learning for enhancing smart contract vulnerability detection on bytecode[C]//Proceedings of the ACM Web Conference 2023. New York: ACM Press, 2023: 2220-2229.

[13] QIAN P, LIU Z G, HE Q M, et al. Towards automated reentrancy detection for smart contracts based on sequential models[J]. IEEE Access, 2020, 8: 19685-19695.

[14] LUO F, LUO R J, CHEN T, et al. SCVHUNTER: smart contract vulnerability detection based on heterogeneous graph attention network[C]// Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE). Piscataway: IEEE Press, 2024: 2098-2110.

[15] ZHUANG Y, LIU Z, QIAN P, et al. Smart contract vulnerability detection using graph neural network[C]//Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence. Piscataway: IEEE Press, 2020: 3283-3290.

[16] WU H J, ZHANG Z, WANG S W, et al. Peculiar: smart contract vulnerability detection based on crucial data flow graph and pre-training techniques[C]//Proceedings of the 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE). Piscataway: IEEE Press, 2021: 378-389.

[17] LIU Z G, QIAN P, WANG X Y, et al. Combining graph neural networks with expert knowledge for smart contract vulnerability detection[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(2): 1296-1310.

[18] VIDAL F R, IVAKI N, LARANJEIRO N. Vulnerability detection techniques for smart contracts: a systematic literature review[J]. Journal of Systems and Software, 2024, 217: 112160.

[19] KHAN Z A, NAMIN A S. A survey of vulnerability detection techniques by smart contract tools[J]. IEEE Access, 2024, 12: 70870-70910.

[20] CHEN J C, CHEN C, HU J, et al. Identifying smart contract security issues in code snippets from stack overflow[C]//Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM, 2024: 1198-1210.

[21] CHEN H Y, ZHAO X F, WANG Y C, et al. SafeCheck: detecting smart contract vulnerabilities based on static program analysis methods[J]. Security and Privacy, 2024, 7(5): e393.

[22] CAI J, LI B, ZHANG T, et al. Fine-grained smart contract vulnerability detection by heterogeneous code feature learning and automated dataset construction[J]. Journal of Systems and Software, 2024, 209: 111919.

[23] YANG H W, GU X G, CHEN X, et al. CrossFuzz: cross-contract fuzzing for smart contract vulnerability detection[J]. Science of Computer Programming, 2024, 234: 103076.

[24] ZHANG L J, LI Y, GUO R, et al. A novel smart contract reentrancy vulnerability detection model based on BiGAS[J]. Journal of Signal Processing Systems, 2024, 96(3): 215-237.

[25] COLIN L S H, MOHAN P M, PAN J, et al. An integrated smart contract vulnerability detection tool using multi-layer perceptron on real-time solidity smart contracts[J]. IEEE Access, 2024, 12: 23549-23567.

[26] ALLAMANIS M, BROCKSCHMIDT M, KHADEMI M. Learning to represent programs with graphs[C]//International Conference on Learning Representations. Vancouver: ICLR, 2018: 1-17.

[27] VELIČKOVIĆ P, CUCURULL G, CASANOVA A, et al. Graph attention networks[C]//International Conference on Learning Representations. Vancouver: ICLR, 2018: 1-12.

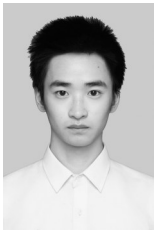
[28] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[J]. Advances in Neural Information Processing Systems, 2017, 30: 5998-6008.

[29] GU X, CHEN G, WANG Y F, et al. Text with knowledge graph augmented transformer for video captioning[C]//Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Piscataway: IEEE Press, 2023: 18941-18951.

[30] DURIEUX T, FERREIRA J F, ABREU R, et al. Empirical review of automated analysis tools on 47,587 Ethereum smart contracts[C]//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Piscataway: IEEE Press, 2020: 530-541.

[作者简介]

陈锦富（1978– ），男，江西赣州人，博士，江苏大学教授、博士生导师，主要研究方向为软件测试、软件安全和可信软件。



胡心怡（2000– ），男，江苏常州人，江苏大学硕士生，主要研究方向为区块链漏洞检测、软件安全测试。

蔡赛华（1990– ），男，江苏南通人，博士，江苏大学副教授、硕士生导师，主要研究方向为恶意流量检测、异常数据检测、软件安全测试。

闵玺润（2001– ），男，江苏无锡人，江苏大学硕士生，主要研究方向为智能合约漏洞检测、软件安全。