文章编号: 1673-3193(2025)01-0105-11 J. North Univ. China, Nat. Sci. Ed., 2025, 46(1): 105-115.

基于程序语义与度量的代码缺陷检测

卢 跃[1], 嵇友晴[1], 周礼亮[2], 吕 青[1], 张迎周[1]

(1. 南京邮电大学 计算机学院, 江苏 南京 210023; 2. 中国电子科技集团公司第十研究所, 四川 成都 610036)

摘 要: 软件中存在的代码缺陷严重影响了软件用户使用的体验感和安全性, 传统的代码缺陷检测方法存在准确率较低的问题, 而结合深度学习的现有方法的检测粒度较粗, 检测效果也不够理想。为此, 本文提出了一种基于程序语义与度量的代码缺陷检测方法。该方法采用基于LLVM IR的代码缺陷的兴趣点检测算法, 使用轻量级符号化程序切片工具SymPas获取与缺陷兴趣点相关的程序切片。通过预训练模型将程序切片代码片段转化为向量表示, 并融合指令级切片度量——认知复杂度度量, 深入分析了切片语句之间的关系和特征。通过构建混合模型ResCNN-GRU进行训练, 将提取的特征进行了有效融合和学习。实验结果表明, 本文利用符号化程序切片技术细化了漏洞检测的粒度, 在中间表示LLVM IR下融合的语义和度量信息能更好地表示缺陷代码语句间的关系和特征, 构建的混合模型一定程度上解决了时间序列问题以及样本数量不均衡问题, 相比其他先进方法, 本文方法的准确率达到了94.1%。

关键词: 预训练模型; 程序切片; 切片认知域; 残差网络; 卷积神经网络; 门控制神经网络

中图分类号: TP311 **文献标识码:** A **doi:** 10.62756/jnuc.issn.1673-3193.2023.10.0030

引用格式: 卢跃, 嵇友晴, 周礼亮, 等. 基于程序语义与度量的代码缺陷检测[J]. 中北大学学报(自然科学版), 2025, 46(1): 105-115.

LU Yue, JI Youqing, ZHOU Liliang, et al. Code defect detection based on program semantics and metrics[J]. Journal of North University of China(Natural Science Edition), 2025, 46(1): 105-115.

Code Defect Detection Based on Program Semantics and Metrics

LU Yue[1], JI Youqing[1], ZHOU Liliang[2], LYU Qing[1], ZHANG Yingzhou[1]

(1. College of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China;
2. The 10th Research Institute of China Electronics Technology Group Corporation, Chengdu 610036, China)

Abstract: Code defects in software seriously affect the experience and security of software users. Traditional code defect detection methods have the problem of low accuracy, while the existing methods combined with deep learning have coarse detection granularity and less than ideal detection effect. For this reason, this paper proposed a code defect detection method based on program semantics and metrics. A point-of-interest detection algorithm for code defects based on LLVM IR was designed, which used SymPas, a lightweight symbolic program slicing tool, to obtain program slices related to defective points of interest. The program slices code fragments were transformed into vector representations by a pre-trained model, and the instruction-level slicing metric, cognitive complexity metric, was fused to deeply analyze the relationships and features between the sliced statements. A hybrid model ResCNN-GRU was constructed for training to effectively fuse and learn the extracted features. The experimental results show that this paper refines the granularity of vulnerability

收稿日期: 2023-10-31

基金项目: 国家自然科学基金资助项目(62272214)

作者简介: 卢 跃(1995—), 男, 硕士生, 主要从事程序分析的研究。

通信作者: 张迎周(1978—), 男, 教授, 博士, 主要从事软件分析的研究。E-mail: zhangyz@njupt.edu.cn。

detection by using symbolic program slicing technique, the fused semantic and metric information under the intermediate representation LLVM IR can better represent the relationships and features between the defective code statements, and the constructed hybrid model solves the time-series problem as well as the unbalanced number of samples problem to a certain extent, and comparing with several advanced methods, the accuracy of this paper's method reaches 94.1%.

Key words: pre-training model; program slicing; slice cognitive domain; residual network; convolution neural network; gate control neural network

0 引 言

随着信息技术的不断发展,人们使用的软件越来越多。软件虽然带来了很多便利,但同时也面临着越来越多的漏洞威胁。软件漏洞已成为影响软件正常运行最大的隐患之一,尤其是在系统规模和复杂性迅速增长、新软件模式广泛应用的情况下,软件系统内外的各种非确定性不断增强,软件质量问题变得日益突出。一旦软件缺陷在航天、交通、国防等安全攸关领域被触发,所导致的后果和损失是极其巨大的。例如:在2017年5月12日全球爆发的Wanacry勒索蠕虫事件中,许多组织遭受了严重的数据损失和经济损失;在2017年6月的Petya变种勒索病毒攻击中,攻击者利用了Microsoft Office远程执行代码漏洞(CVE-2017-0199)进行投递,导致大量组织遭受了攻击;此外,近期还发生了Log4j和微软Exchange等“核弹级”漏洞事件,其对安全造成了巨大的威胁,应当给予足够的关注。

代码缺陷检测技术按照检测代码对象分为源代码分析和二进制分析;按照代码运行方式分为静态分析、动态分析和动静结合分析。动态分析需要运行系统代码,并通过比较实际的输出和预期的结果来判断系统是否存在问题,主要分为模糊测试、动态符号执行和动态污点分析等三种。静态分析不需要运行程序代码,可以直接对源码或二进制代码进行分析,通过专家定义的安全规则或深度学习模型检测代码缺陷。动静结合分析则是结合了动态分析和静态分析的优势。代码缺陷检测通常将源代码转换成token序列、树结构、图结构等表征形式进行分析。

近年来,随着深度学习的快速发展,越来越多的研究人员开始将深度学习应用于软件工程领域。Li等[1]首次提出了基于深度学习的漏洞检测系统VulDeePecker。该系统通过对源码中的关键点进行切片,将切片代码块定义为“Code Gadget”,并将这些“Code Gadget”转化为向量,作为深度学习模型的输入,相比其他方法,VulDeePecker具有更低的漏报率,但只能处理C/C++程序中与库/API函数调用有关的漏洞,只利用了由数据依赖引起的语义信息,没有考虑控制依赖。Yamaguchi等[2]从源代码中提取所有函数的抽象语法树(AST),将AST嵌入向量空间,使用机器学习方法分析这些函数结构模式组合中存在的漏洞。Li等[3]提出了一种名为SySeVR的框架,该框架专注于获取能够适应与漏洞相关的语法和语义信息的程序表示,适用于C/C++代码中与数组、算术表达式、指针、函数调用相关的四类漏洞类型。Pham等[4]则采用基于图的API来表征漏洞代码段,以检测共享库中的再现漏洞,但实验结果往往受到公共漏洞数据库完整性的限制。不同的编程语言和代码库可能具有不同的语法结构和代码规范,使得针对特定语言或代码库漏洞检测时往往需要进行调整,因此也限制了这些方法的可扩展性和通用性。

将图神经网络应用于代码的漏洞检测是一个研究热点。Zhou等[5]提出了一种基于图神经网络(GNN)的源代码漏洞检测模型Devign。该模型将源代码转化为抽象语法树(AST)、控制流图(CFG)和数据流图(DFG)三种图结构的联合图,使用word2vec模型对这些复合代码特征向量化,然后将向量作为GNN模型的输入来学习聚合特征。将整个函数作为检测对象,会导致粒度太大且对于大型函数无法高效地学习与检测。Cao等[6]提出了一种名为MVD的方法,它基于流敏感图神经网络(FS-GNN),用于检测与内存相关的漏洞。FS-GNN能够联合嵌入非结构化信息和结构化信息,以捕获与隐式内存相关的漏洞模式,该方法虽然缩小了检测粒度但只能检测与内存相关的漏洞。Wu等[7]提出了一个VulCNN模型,其借鉴了基于深度学习的图像分类方法,将函数的源代码转换为图像表示,将源代码解析和标记

化，并保留了关键的句法和语义信息，使模型能够有效地学习源代码的内部表示，这是首个利用图像分类方法进行漏洞检测的工作，但仍将函数作为检测粒度。Rabheru等[8]设计了DeepTective用于检测PHP源代码中的漏洞，其采用了一种新颖的混合技术，结合了门控循环单元和图卷积网络，从而能够利用句法和语义信息来检测SQLi、XSS和OSCI漏洞。DeepTective为了学习到源代码中的语法和结构特征，将其转换为一系列由门控循环单元（GRU）进行分析的标记。虽然图形模型能够更好地处理全局依赖信息和结构信息，但图结构表示源代码时也引入了复杂的计算和分析过程，图的构建和处理需要大量的计算资源和时间。

构建图的代价很大，因此对大型程序的研究就比较困难。从源代码中提取缺陷相关特征的全面性是决定代码缺陷检测效果的关键因素。代码粒度的选择也至关重要，常见的粒度有文件级、函数级、行级别等。常规基于深度学习的检测方法的文件级别粒度太大，容易学习到与缺陷无关的特征，而函数级别和行级别没有考虑相互之间的依赖关系。本文所使用的符号化切片的粒度，考虑了代码之间的数据依赖和控制依赖，具有丰富的语义特征。因此，符号化切片工具可以高效地对大型程序进行切片，提取与缺陷相关的特征，并结合预训练模型进行特征向量化，从而使最终预测的准确率更高。

综合上述分析，本文提出了一种基于程序语义与度量的代码缺陷检测方法，基于源码的中间表示并使用轻量级符号化程序切片工具SymPas来获取与缺陷兴趣点相关的程序切片，通过预训练模型CodeBERT和IR-BERT，将程序切片代码片段转化为向量表示，并融合新的指令级切片度量——认知复杂度度量，深入分析切片语句之间的关系和特征，并构建混合模型ResCNN-GRU来进行训练，将提取的特征进行有效融合和学习。

1 技术背景

1.1 程序切片

程序切片是一种分析和理解程序的技术。程序切片以切片准则（Slicing Criterion）为标准，从被测程序中抽取出满足准则要求的有关语句，忽略与此无关的语句，因此，该技术有利于故障定位、程序调试、软件测试等任务[9]。本文使用的是一种基于轻量级上下文敏感依赖分析的符号化程序切片方法SymPas（Symbolic Program Slicing）[10]。SymPas可以通过LLVM中间码IR实现过程间切片，无需构造易爆炸的SDG图或超图，切片的时间和空间成本大大降低，从而提高了对大型程序进行切片的效率。

1.2 预训练模型

为了让深度学习模型能够处理代码片段，需要将代码片段转换为向量表示，这种技术称为代码表示学习（Code Embedding），也叫做代码嵌入，其目的是将代码的语义信息保留在向量中，常用的有One-Hot编码、词嵌入等方法。本文使用到了CodeBERT[11]预训练模型与IR-BERT[12]预训练模型。预训练模型近年来已经应用到自然语言处理和编程语言等领域。CodeBERT能够在大规模数据集上训练出一个较好的模型，通过参数微调后的CodeBERT模型可在新任务上达到最优性能。如图1所示，图上半部分的add函数存在整形溢出的错误，图下半部分的add函数加入了if判断条件，可以避免整形溢出的缺陷，图中右半部分是CodeBERT生成的不同的语义信息。

源代码	代码嵌入

int add (int a){
int b = 2147483647;
b = b+ a;
return b;
}
	–0.125 3, 0.291 3,–0.036 5,...,–0.440 0,–0.301 2,0.396 3
	–0.129 7,0.297 7,–0.031 6,...,–0.447 7,–0.303 1,0.403 3
	–0.290 8,–0.539 1,0.006 3,...,0.298 0,–0.860 3,1.020 7
	......
	–0.129 4,0.297 5,–0.031 7,...,0.447 5,–0.303 0,0.403 0

int add (int a){
int b = 2147483647;
If(a<0){
b = b+ a;
}
return b;
}
	–0.112 9,0.288 8,–0.073 2,...,–0.365 2,–0.328 9,0.389 0
	–0.119 1,0.295 7,–0.067 9,....,–0.373 4,–0.332 1,0.395 2
	–0.199 5,–0.610 3,0.039 9,...,0.291 9,–0.798 6,0.930 2
	......
	–0.118 8,0.295 6,–0.068 0,...,–0.373 3,–0.332 0,0.395 0

图1 add函数通过CodeBERT的代码嵌入

Fig. 1 CodeBERT embedding by means of add function

本文还使用了IR-BERT预训练模型。不同编程语言的程序都可以被编译成LLVM IR指令，因此使用LLVM IR作为中间表示进行分析可以使方法更具有普适性和可扩展性。

1.3 深度学习网络

1.3.1 GRU门控制神经网络

门控制神经网络GRU由Cho等[13]提出，它是长短期记忆神经网络LSTM的一个变种，用于解决长期记忆和反向传播中的梯度消失等问题。GRU结构更简单，便于计算，更易于实现。GRU的内部结构如图2所示。

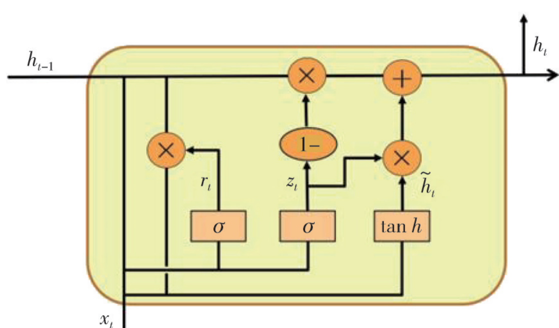

图2 GRU的内部结构

Fig. 2 Internal structure of GRU

图2中, x_t是当前时刻的输入, $\tilde{h}_t$是当前时刻的候选输出, h_{t-1}是上一时刻的输出, 也作为本时刻的输入, h_t是当前时刻的输出。模型的表达式为

$$r_t = \sigma(W_r \cdot [x_t, h_{t-1}]), \quad (1)$$

$$z_t = \sigma(W_z \cdot [x_t, h_{t-1}]), \quad (2)$$

$$\tilde{h}_t = \tan h(W \cdot [r_t * h_{t-1}, x_t]), \quad (3)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t。 \quad (4)$$

1.3.2 卷积神经网络

卷积神经网络[14](CNN)是一种在深度学习领域广泛使用的多层神经网络, 通过共享权值来解决随神经网络层数的加深而带来的参数过多问题。最典型的卷积网络由卷积层、池化层和全连接层组成。其中, 卷积层与池化层配合, 组成多个卷积组, 逐层提取特征, 最终通过若干个全连接层完成分类。

1.3.3 残差网络 (ResNet)

残差网络结构如图3所示, 它可以解决深层网络训练困难的问题。随着网络层数的不断加深, 目标函数越来越容易陷入局部最优解。残差单元通过跳层连接的形式实现, 即将单元的输入直接与单元的输出加在一起再激活。残差网络将拟合恒等映射函数转化为优化残差函数。当残差函数为零时就构成了恒等映射, 这样就避免了多余网络层产生的冗余, 使得网络在梯度下降的时候能够有效应对网络退化问题。

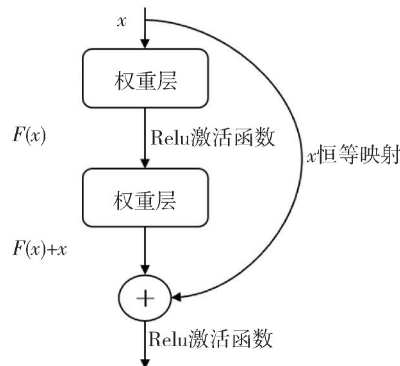

图3 残差网络结构

Fig. 3 The structure of ResNet

2 本文模型

2.1 总体框架

基于静态切片语义与度量的代码漏洞检测总体框架如图4所示。

该框架主要包括模型训练阶段和模型测试阶段。每个阶段的主要步骤分为: 1) 将源代码转化为LLVM IR, 用于后续分析和处理; 2) 兴趣点获取, 即根据兴趣点选取算法, 在LLVM IR层面选择与缺陷相关的兴趣点; 3) 获取切片语义, 即利用SymPas工具获取兴趣点的前向切片和后向切片, 使用CodeBERT和IR-BERT将LLVM IR指令转化为向量表示, 捕捉其语义信息; 4) 合并语义与度量信息; 5) 混合模型ResCNN-GRU进行分类。

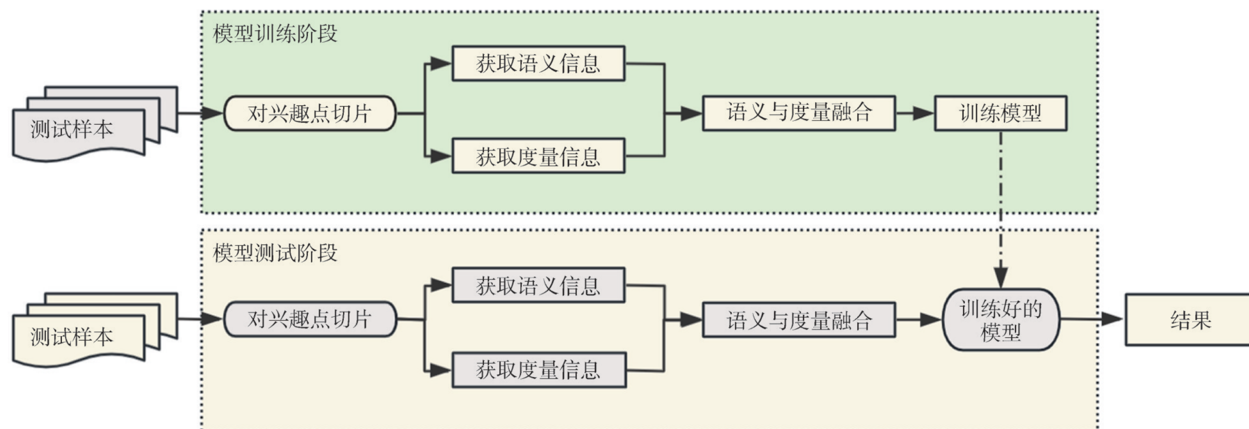

图4 总体框架

Fig. 4 Overall framework

2.2　兴趣点选取

在对代码进行程序切片时，需要以某个变量作为切点，进而获得其前向切片或后向切片，这个变量被称为切入点，可以提取代码的一般特征，特别是与漏洞有关的代码片段。因此，需要引入“程序关注点”或“程序兴趣点”的概念，可以将其看作一个漏洞的“中心”，或者暗示存在漏洞的代码片段，这里称为“兴趣点”。如图 5 所示，在 verify_cb 函数中，代码首先通 X509_STORE_CTX_get_ex_data 函数从 ctx 中获取 ssl 指针，然后在 if 语句中判断 ssl 是否为空，如果不为空，则继续调用 SSL_get0_peer_scts 函数对其进行解引用。如图 6 所示，在 LLVM IR 指令中，可以看到 icmp 指令比较了 X509_STORE_CTX_get_ex_data 函数返回的指针和空指针。如果指针不为空，if.then 块将被执行，其中 SSL_get0_peer_scts 函数对该指针进行了解引用，导致了空指针解引用漏洞。因此，指针 ssl 被选为兴趣点，以便后续提取与兴趣点相关的语义和度量信息。

```
int verify_cb(int ok, X509_STORE_CTX* ctx)
{
    char buf[256];
    X509* err_cert;
    int depth, err;
    SSL* ssl;
    ssl = X509_STORE_CTX_get_ex_data(ctx, SSL_get_ex_data_X509_STORE_CTX_idx());
    if (ssl)
        SSL_get0_peer_scts(ssl, NULL, 0);
    //...
}
```

图 5　verify_cb 函数

Fig. 5　verify_cb function

```
define i32 @verify_cb(i32% ok, % X509_STORE_CTX*% ctx) {
entry:
    % ssl = call i8 * @X509_STORE_CTX_get_ex_data(% X509_STORE_CTX * %ctx, i32 69)
    % cmp = icmp ne i8 * %ssl, null
    br i1 % cmp, label% if.then, label% if.end
if.then:
    % null = alloca i8*, align 8
    % num_null = alloca i32, align 4
    store i8 * null, i8**% null, align 8
    store i32 0, i32*% num_null, align 4
    call void @SSL_get0_peer_scts(i8 * %ssl, i8 * *%null, i32 * %num_null)
    //...
    br label % if.end
if.end:
    //...
}
```

图 6　verify_cb 函数对应的 LLVM IR 指令

Fig. 6　LLVM IR corresponding to verify_cb function

本文针对常见的漏洞类型总结了兴趣点选取准则中的指令，给出了指令中选取兴趣点的依据。

1）alloca 指令：将 alloca 指令的参数和返回的指针作为兴趣点。全局变量：如果 alloca 指令的参数是一个全局变量，则返回的指针作为兴趣点。返回值：如果 alloca 指令的返回值是一个函数的返回值，则返回的指针作为兴趣点。其他指针：如果 alloca 指令的参数是其他指针，则返回的指针作为兴趣点。

2）getElementPtr 指令：将 getElementPtr 操作的指针变量选为兴趣点指针参数。指针变量可以是指针类型、结构体、数组等。

3）赋值指令（store 和 load）：对于 store 指令，将其存储的值对应的地址标记为兴趣点；对于 load 指令，将其加载的地址对应的值标记为兴趣点。

4）对于调用指令（call），将其指令中调用的参数标记为兴趣点，如果函数返回值为指针类型，则将其标记为兴趣点。

5）对于算术指令（add、sub、mul、div 等），将其任何一个操作数标记为兴趣点。

6）对于比较指令（icmp 和 fcmp），将其任何一个操作数标记为兴趣点。

检测可能存在缺陷的兴趣点的步骤如算法 1 所示。

算法 1：基于 LLVM IR 的漏洞兴趣点检测算法

输入：程序 P

输出：兴趣点集合 K

```
1.  #将程序P通过编译器LLVM的Clang转化为LLVM IR
    指令
2.  llvm_ir ← compile_to_llvm_ir(P)
3.  #创建IR指令的函数调用图call_graph
4.  call_graph ← create_call_graph(llvm_ir)
5.  K ←set() #初始化空的兴趣点集合K
6.  #遍历函数调用图中的每个函数function
7.  For function in call_graph.functions do
8.    #遍历函数function中的每个基本块block
9.    For block in function.basic_blocks do
10.     #遍历基本块block中的每条IR指令
11.     For instruction in block.instructions do
12.       #判断该指令是否满足六大兴趣点准则
13.       If interesting_point_criteria(instruction) then
14.       #获取与该指令相关的变量,将其添加到兴趣点集合
          K中
15.       variables ← get_relate_variables(instruction)
16.       K.update(variables)
17.       End if
18.     End for
19.   End for
20.   End for
21.   Return K
```

首先，将程序 P 通过 Clang 编译器转化为 LLVM IR。其次，构建函数调用图，创建一个空的兴趣点集合 K。再次，通过遍历函数调用图中的每个函数和函数内的基本块，对基本块内的每条 LLVM IR，对指令进行六大兴趣点准则的判断。如果指令满足兴趣点准则，就从切片表中获

取与该指令相关的变量,并将这些变量添加到兴趣点集合K中。最后,返回兴趣点集合K。

2.3 语义特征提取

获取程序兴趣点后使用轻量级符号化程序切片工具SymPas获取每个兴趣点的程序切片结果——前向切片和后向切片。程序切片的结果是与兴趣点存在控制依赖和数据依赖的语句集合,故程序切片的结果包含兴趣点的控制依赖与数据依赖信息。为了让深度学习模型能够处理代码片段,需要将代码片段转换为向量表示。借助预训练模型,通过参数微调将程序切片的结果转为向量表示,预训练模型借助位置编码能够获取语句间的长距离依赖,故保留了与关键点相关的逻辑操作语义。将前向切片和后向切片的行号合并,按照行号从小到大的顺序将代码组合起来,通过CodeBERT预训练模型生成相应的向量。

在图7所示的示例代码中,以第22行代码为例,指针数组可能会发生越界的问题,根据兴趣点选取准则,data变量被选为一个兴趣点,考虑到其上下文的控制依赖信息和数据依赖信息,通过SymPas可以获取变量的前向切片和后向切片,如表1所示。

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <unistd.h>
5  void printDate(const char * d){
6    if(d!=NULL)
7    printf("%s\n",d);
8  }
9  void fund(){
10    char *data;
11    char dataBuffer[100];
12    char source[100];
13    memset(dataBuffer,'A',99);
14    dataBuffer[99]='\0';
15    while(1){
16       data = dataBuffer - 8;
17       break;
18    }
19    memset(source,'C',99);
20    source[99]='\0';
21    memmove(data,source,100*sizeof(char));
22    data[99]='\0';
23    printDate(data);
24  }
25  int main(void) {
26   fund();
27   return 0;
28  }
```

图7 示例代码

Fig. 7 Example code

将前向切片和后向切片的行号合并,按照行号从小到大的顺序将代码组合起来,通过Code-BERT预训练模型生成相应的向量。图8所示为data变量对应的IR指令切片,通过IR-BERT预训练模型微调后生成相应的向量。

表1 静态切片表

Tab. 1 Static slicing table

变量	后向切片	前向切片
d@printDate	{6,13,14,16,19,20,21,22,23,26}	{6,7}
data@fund	{10,11,12,13,14,15,16,19,20,21,22}	{23,6,7}
dataBuffer@fund	{11,12,13,14}	{16,21,22,23,6,7}
source@fund	{12,19,20}	{21,22,23,6,7}

```
label %6,%3 = icmp ne i8* %0 , null,
br i8* %3 , label %4 ,
%5 = call i32(i8*, ...)* @printf ( [4 x i8] [4 x i8] getelementptr inbounds
( [4 x i8]* @.str ,  i64 0, i64 0 ), i8* %0 )
br label %6,
%1 = alloca i8* , align 8
%2 = alloca [100 x i8] , align 16,
%3 = alloca [100 x i8] , align 16,
call void(metadata, metadata, metadata)* @llvm.dbg.declare ( i8** %1,
metadata* %data, metadata c"!28" ),
call void(metadata, metadata, metadata)* @llvm.dbg.declare ( [100 x i8]* %3, meta
data* %source, metadata c"!28" ),
%4 = getelementptr inbounds [100 x i8]* %2 , i64 0, i64 0,
call void(i8*, i8, i64, i1)* @llvm.memset.p0i8.i64 ( [100 x i8]* %4, i64 65, i64
99, i8 0 ),
........
%9 = getelementptr inbounds i8* %6 , i64 99,
store i64 0 , i8* %9 , align 1,
call void(i8*)* @printDate ( i8* %6 ),
```

图8 data变量对应的IR指令切片

Fig. 8 IR instruction slice corresponding to the data variable

2.4 度量特征提取

在软件工程中,度量是一种广泛应用的方法,可以帮助开发者预测可能存在的风险,并评估软件质量。传统的软件度量包括圈复杂度、Halstead度量、C&K度量、代码行数等[15]。已有的切片度量主要关注基于程序切片的内聚性度量和耦合性度量。然而,指令级切片不仅可以得到更精确的度量结果,还可以减少不同编程语言对程序分析的影响。由于指令级切片的分析粒度更细,它可以提供更精确的度量结果,进而更准确地分析程序的复杂性[16]。本文基于IR指令的前向切片和后向切片构建新型的切片度量指标,即基于切片的认知复杂度度量——切片认知域。具体公式为

$$IRsliceSpatial=\frac{\sum_{i\in B}\sqrt{(x_i-x_k)^2+(y_i-y_k)^2+(z_i-z_k)^2}}{len(B)}。\quad(5)$$

式中:B为前向和后向切片集合;K为关键点;$len(B)$

为前向切片和后向切片的数量；x_i为切片中第i个切片在CG图上的坐标；y_i为切片中第i个切片在CFG图上的坐标；z_i为切片中第i个切片在block上的坐标；x_k为关键点在CG图上的坐标；y_k为关键点在CFG图上的坐标；z_k为关键点在block上的坐标。

该度量的含义是指令行中切片语句离兴趣点的距离越大，了解该切片的数据流信息和控制流信息所需的认知努力越多，即编程时对于理解关键点操作逻辑的困难越大，故其存在缺陷的可能就越大。对程序的兴趣点进行前向切片和后向切片之后，会得到一定数量的切片指令，受到聚类思想的启发[17]，这些切片指令集和兴趣点距离越大，越不容易聚合在一起，存在缺陷的可能性越大。

为了精准地考虑距离，本文给出了每条指令的坐标概念，对IR文件生成其函数调用关系图（Call Graph，CG），CG图上的节点是一个函数，通过从上个函数生成其流程控制图（Control Flow Graph，CFG），通过层次遍历给CFG中的每个block一个坐标，给每个block中的指令一个坐标。最后，将度量特征和语义特征融合，作为深度学习模型的输入，如图4所示。

2.5 语义与度量信息融合

由2.3和2.4节分别得到语义特征向量$\boldsymbol{X}=\{x_1, x_2, x_3, \cdots, x_n\}$和度量特征向量$\boldsymbol{Y}=\{y_1, y_2, y_3, \cdots, y_m\}$。将语义信息$\boldsymbol{X}$和度量信息$\boldsymbol{Y}$进行合并连接作为语义与度量信息融合的结果，即$\boldsymbol{Z}=\boldsymbol{X}\oplus\boldsymbol{Y}=\{x_1, x_2, x_3, \cdots, x_n, y_1, y_2, y_3, \cdots, y_m\}$。特征提取与合并后将向量$\boldsymbol{Z}$提供给后续模型ResCNN-GRU进行学习和训练。

2.6 ResCNN-GRU模型

GRU拥有记忆特性，能够有效处理具有时间序列特性的问题。尽管相对LSTM结构有所简化，但是在网络结构加深的情况下，处理大量数据时的计算量依旧很庞大。通过引入CNN进行特征提取[18]，能够对低维数据的深层次特征进行提取，从而降低GRU部分训练所需的时间，并提升模型的训练结果。当网络深度增加到一定程度时，使用深度学习方法依旧面临着准确率降低的问题，这是由梯度消失现象导致的。在网络层数不断加深的同时，输入层附近的非线性参数训练将变得困难。因此，单纯的增加网络深度并不能直接提高模型效果。残差网络通过使用恒等映射的方法解决了网络深度加深时训练无法成功的问题。

本文最终构建的混合模型ResCNN-GRU结构如图9所示，基础结构由一系列残差块组成，每个残差块由卷积块CNN、门控循环单元和跳跃连接组成。卷积块CNN通过堆叠多层卷积、批归一化和激活函数的方式提取局部特征，从而增强模型的表达能力。GRU则是一种门控循环神经网络，它通过一系列的门控机制控制信息的传递和遗忘，从而可以更好地捕捉序列数据的长依赖关系。在每个残差块中，CNN和GRU之间通过跳跃连接进行连接，使得局部特征和长依赖信息可以更好地融合和传递。

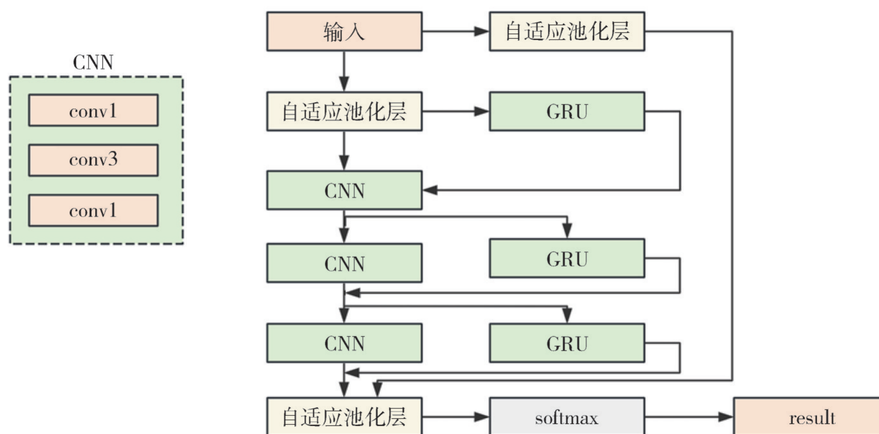

图9 基于残差结构的CNN-GRU模型

Fig. 9 CNN-GRU model based on residual structure

ResCNN-GRU定义了CNN卷积块，它由3个卷积层组成，每个卷积层后面跟随一个批量归一化层和一个ReLU激活函数层。第1个卷积层，卷积核大小为1；第2个卷积层，卷积核大小为3；第3个卷积层，卷积核大小为1。ResCNN-GRU通过一个由一维卷积层、批归一化层和ReLU激

活函数构成的卷积块对输入长序列向量进行处理，得到了一系列卷积特征。接着，使用一系列由卷积层、批归一化层和ReLU激活函数构成的卷积块对这些特征进行进一步的处理。其中，这些卷积块的设计使得网络可以在每一层对特征进行逐渐抽象和提炼，从而获取更加高层次的特征信息。在经过一系列卷积块的处理后，使用自适应平均池化层对特征进行降维，将降维后的特征输入到GRU模型中进行处理，以进一步提取时序信息，这样可以捕捉时间序列的局部和全局特征，从而提高模型的泛化性能。在模型中的每个GRU模型输出之后，将其与卷积块的输出进行残差连接，进一步提取特征信息。使用残差结构提高了模型的深度，有助于提高模型的性能，同时减轻了梯度消失的问题，提高了训练效率。整个模型的输出通过全局平均池化层(Global Average Pooling)进行降维，将卷积层的输出转换为定长的向量，然后通过全连接层进行分类。这种结构可以处理不同长度的输入序列，并提高模型的鲁棒性。最终使用全连接层(Linear)将特征映射到最终的分类结果上，并通过使用Focal Loss和Early stopping优化模型性能。这种结合能够很好地解决时间序列问题以及样本数量不均衡问题，并在保证模型精度的同时提高训练速度。具体的代码漏洞检测算法如算法2所示。

算法2： 基于静态切片语义与度量代码的漏洞检测算法

输入： 程序P

输出： 预测值prediction #0表示不存在漏洞，1表示存在漏洞

```
1. llvm_ir ←compile_to_llvm_ir(P)
2. #调用兴趣点选取算法
3. interest_points ← find_interest_points(llvm_ir)
4. For interest_point in interest_points do
5.   forward_slice ←SymPas.get_forward_slice(interest_point)
6.   backward_slice ← SymPas.get_backward_slice(interest_point)
7.   #合并前向切片和后向切片
8.   merged_slice ← merge_slices(forward_slice, backward_slice)
9.   #使用预训练模型将LLVM IR指令转化为向量表示
10.  merged_slice_vector ← Inst2vec.convert_to_vector
     (merged_slice)
11.  merged_result←
     merge_semantic_metric(merged_slice_vector, metric)
12. End for
13. #使用混合模型ResCNN-GRU进行训练
14. hybrid_model ← ResCNN_GRU()
15. hybrid_model.train(training_merged_result)
16. prediction← hybrid_model.predict(test_merged_result)
17. Return prediction
```

3　实验结果与分析

3.1　实验环境

表2　实验环境

Tab. 2　Experimental environment

环境	配置
操作系统	Windows 10
开发语言	Python
Python版本	Python3.7
Torch版本	1.11.0
Torchvision 版本	0.12.0
内存	16 GB
硬盘	512 GB

3.2　实验数据集

实验数据集是美国国家标准与技术研究院(National Institute of Standards and Technology, NIST)的软件保证参考数据集(Software Assurance Reference Dataset, SARD)[19]，本文中涉及到的缺陷类型包括CWE134(使用外部控制的格式字符串)、CWE121(堆栈缓冲区溢出)、CWE119(内存缓冲区边界内操作的限制不恰当)、CWE-399(资源管理漏洞)等。这些数据集中的代码分为两类：存在缺陷代码为bad，不存在缺陷代码为good。通过对数据集中的代码通过程序切片最终的testcase有83 762个，其中，与指针相关的错误有21 358个，与数组相关的错误有12 834，与API调用相关错误的有41 983个，与算数表达式相关错误有7587个，训练集、验证集、测试集的样本数量比为7:2:1。

3.3　评价指标

准确率Acc(R_{Acc})、误报率FPR(R_{FP})、漏报率FNR(R_{FN})、查准率(P)和$F1$分数($F1$ score)是5个常见的评价模型检测准确程度的指标。假设N_{FP}为本身不存在缺陷但被判定为存在缺陷的样本数量，N_{FN}为本身存在缺陷但被判定为不存在缺陷的样本数量，N_{TP}为本身存在缺陷同时也被判定为存在缺陷的样本数量，N_{TN}为本身不存在缺陷同时也被判定为不存在缺陷的样本数量，则Acc、FPR、FNR、P和$F1$的计算方法为

$$R_{Acc}=\frac{N_{TP}+N_{TN}}{N_{TP}+N_{TN}+N_{FP}+N_{FN}},\qquad(6)$$

$$R_{FP}=\frac{N_{FP}}{N_{FP}+N_{TN}},\qquad(7)$$

$$R_{FN}=\frac{N_{FN}}{N_{TP}+N_{FN}},\tag{8}$$

$$P=\frac{N_{TP}}{N_{TP}+N_{FP}},\tag{9}$$

$$R_{TPR}=\frac{N_{TP}}{N_{TP}+N_{FP}},\tag{10}$$

$$F1=\frac{2\times P\times R_{TPR}}{P+R_{TPR}}。\tag{11}$$

3.4 实验参数

模型使用的优化器Adam的学习率为0.001，损失函数为focal loss。利用early_stopping降低过拟合，early_stopping以验证集的loss为指标，当连续12个epoch发现验证集的loss不再下降时，就提前停止训练。

3.5 实验对比分析

本文所设计的实验用于回答以下3个研究问题：1）相对于源码表示，本文方法采用LLVM IR中间语言来表示语义信息和度量能否更好地检测代码的缺陷；2）本文方法提出的切片度量能否更好地体现代码的缺陷特征；3）本文方法相较于其他缺陷检测系统是否有更好的缺陷检测性能。

3.5.1 问题1实验结果及分析

针对问题1，从源码和LLVM IR中间语言两个角度进行实验，OneHot编码、Word2vec、CodeBERT是从源码层面进行代码嵌入，而IR-BERT是从LLVM IR中间语言层面进行代码嵌入，为了更好地将代码的语义信息保留在向量中，本文对比了OneHot编码、Word2vec、CodeBERT、IR-BERT方法，通过最终的结果指标来衡量它们的优劣，实验结果如表3所示。

表3 不同代码嵌入方法的实验结果

Tab. 3 Experiment results under different code embedding methods

方法	Acc/%	*P*/%	*F*1/%	FPR/%	FNR/%
OneHot	76.4	77.2	72.9	8.9	30.1
Word2vec	89.6	89.2	86.9	6.8	15.6
CodeBERT	93.7	92.1	93.3	5.0	9.2
IR-BERT	94.1	92.5	93.8	4.5	4.9

由表3可以看出，源码与IR中间语言预训练模型的结果要明显好于OneHot编码、Word2vec。OneHot编码没有考虑代码上下文的关系，而Word2vec利用神经网络对词的上下文训练得到词的向量化表示，考虑了代码的上下文，与OneHot编码方法相比，Word2vec方法的准确率提升了13.2百分点，精度提升了12百分点，效果更好。CodeBERT与IR-BERT均基于Bert思想，可以根据上下文信息生成不同的向量，这种预训练模型可在使用者自己的新任务领域中应用，与Word2vec相比，CodeBERT的准确率提升了4.1百分点，精度提升了2.9百分点。此外，将源码转化成LLVM IR中间语言表示，从IR中间语言的角度而言，相比CodeBERT，IR-BERT的准确率提升了0.4百分点，精度也提升了0.4百分点，漏报率降低了4.3百分点，同时，LLVM IR中间语言层面的方法不仅可以扩展到其他源码语言，还可以更好地检测代码的缺陷。

3.5.2 问题2实验结果及分析

针对问题2，本文进行了4组对比实验，选择4种不同长度的向量作为模型输入，并将切片语义特征和度量特征共同加入到模型中，结果如表4所示。

表4 不同神经网络的实验结果

Tab. 4 Experiment results under different neural networks

方法	Acc/%	*P*/%	*F*1/%	FPR/%	FNR/%
LSTM（语义）	87.5	85.6	81.8	11.8	21.4
LSTM（语义+度量）	89.1	89.6	85.7	8.9	17.8
GRU（语义）	90.4	93.7	90.1	8.3	13.4
GRU（语义+度量）	92.1	91.2	92.8	6.1	9.2
本文方法（语义）	92.5	91.1	89.8	6.0	11.4
本文方法（语义+度量）	94.1	92.5	93.8	4.5	4.9

由表4可以看出，加入度量特征后，预测缺陷的准确率分别提高1.6百分点，1.7百分点和1.6百分点，表明度量特征对于缺陷预测有良好的效果。由LSTM、GRU和本文方法在语义和切片融合度量两方面的实验结果可以看出：在LSTM网络中，准确率、精度、*F*1分数分别提升了1.6百分点，4.0百分点和3.9百分点，误报率、漏报率分别下降了2.9百分点和3.6百分点；在GRU网络中，准确率、*F*1分数分别提升了1.7百分点和2.7百分点，精度、误报率、漏报率分别下降了2.5百分点，2.2百分点和4.2百分点；在本文方法的网络中，准确率、精度、*F*1分数分别提升了1.6百分点，1.4百分点和4.0百分点，误报率、漏报率分别下降了1.5百分点和6.5百分点。综上

可以发现，在不同的神经网络模型中，加入度量特征可以有效提高模型预测缺陷的准确率。尽管通过程序切片可以考虑到程序的数据流和控制流信息，但是转化成向量之后，模型训练仍然存在一定的问题，本质上仍是代码相似性的对比，而本文提出的切片度量方法可以较好地预测代码缺陷。

3.5.3 问题3实验结果及分析

针对问题3，将本文方法与目前较为先进的漏洞检测方法 SySeVR[3]、VulDeePecker[1]、Flawfinder[20]、Checkmarx[21]进行对比，结果如表5所示。实验结果表明：本文提出的方法在准确率、精度、$F1$值、误报率和漏报率等指标上均表现优异。相较于 VulDeePecker 和 SySeVR，本文方法的准确率分别提高了5.0百分点和1.6百分点，精度分别提高了2.9百分点和1.4百分点，$F1$值分别提高了12.0百分点和4.0百分点，误报率分别降低了4.4百分点和1.5百分点，漏报率分别降低了12.9百分点和6.6百分点。这些改进得益于本文方法对模型的改进以及采用 focal loss 损失函数来解决正负样本不均衡的问题。此外，本文提取与漏洞相关的语义信息时，从LLVM IR的中间表示找到缺陷兴趣点，并进行切片，可以更好地体现代码的缺陷。本文方法还融合了基于切片的认知复杂度度量，该特征对于预测代码缺陷也有很好的效果，这也是本文在误报率和漏报率这两个指标上表现优异的原因之一。

表5 不同缺陷检测系统的实验结果

Tab.5 Experiment results of different defect detection systems

方法	Acc/%	P/%	$F1$/%	FPR/%	FNR/%
Flawfinder	69.8	60.1	59.8	26.6	40.5
Checkmarx	74.4	72.8	69.3	8.2	33.8
VulDeePecker	89.1	89.6	81.8	8.9	17.8
SySeVR	92.5	91.1	89.8	6.0	11.5
本文方法	94.1	92.5	93.8	4.5	4.9

4 结论

本文针对漏洞检测领域代码缺陷检测的问题，提出了一种基于程序语义和度量的代码缺陷检测方法。在研究源代码缺陷时专注于LLVM IR中间代码层面，同时利用预训练模型可以使本文方法能够适应不同的代码结构和编程语言，从而具有良好的适应性和实用性。为了有效融合和学习提取的特征，采用了混合模型ResCNN-GRU进行训练，充分利用特征的时序和空间信息来实现有效的漏洞检测。后续将对所提出的方法进行改进，研究代码的多模态表示，同时从动态切片的角度进一步提升代码嵌入的表征能力，从而增强深度学习模型对漏洞的检测能力。

参考文献:

[1] LI Z, ZOU D Q, XU S H, et al. VulDeePecker: a deep learning-based system for vulnerability detection[C]//The 25th Annual Network and Distributed System Security Symposium (NDSS), 2018: 234-258.

[2] YAMAGUCHI F, LOTTMANN M, RIECK K. Generalized vulnerability extrapolation using abstract syntax trees[C]//The 28th Annual Computer Security Applications Conference (ACSAC), 2012: 359-368.

[3] LI Z, ZOU D, XU S, et al. SySeVR: A framework for using deep learning to detect software vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4): 2244-2258.

[4] PHAM N H, NGUYEN T T, NGUYEN H A, et al. Detection of Recurring software vulnerabilities[C]//The IEEE/ACM International Conference on Automated Software Engineering, 2010: 447-456.

[5] ZHOU Y, LIU S, SIOW J, et al. Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks[C]//Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019: 10197-10207.

[6] CAO S, SUN X, BO L, et al. MVD: Memory-related vulnerability detection based on flow-sensitive graph neural networks[C]//Proceedings of the 44th International Conference on Software Engineering (ICSE), 2022: 1456-1468.

[7] WU Y, ZOU D, DOU S, et al. VulCNN: an image-inspired scalable vulnerability detection system[C]//Proceedings of the 44th International Conference on Software Engineering, 2022: 2365-2376.

[8] RABHERU R, HANIF H, MAFFEIS S. A hybrid graph neural network approach for detecting PHP vulnerabilities[C]//2022 IEEE Conference on Dependable and Secure Computing (DSC). IEEE, 2022: 1-9.

[9] PAN K, KIM S, WHITEHEAD J. Bug classification using program slicing metrics[C]//2006 Sixth IEEE International Workshop on Source Code Analysis and Manipulation, 2006: 31-42.

[10] ZHANG Y Z. SymPas：symbolic program slicing[J]. Journal of Computer Science and Technology，2021，36(2)：397-418.

[11] FENG Z Y，GUO D Y，TANG D Y，et al. CodeBERT：A pre-trained model for programming and natural languages[DB/OL].（2020-09-18）[2023-10-31]. http://arxiv.org/abs/2002.08155v4.

[12] GUI Y，WAN Y，ZHANG H，et al. Cross-language binary-source code matching with intermediate representations [C]//2022 IEEE International Conference on Software Analysis，Evolution and Reengineering（SANER），2022：601-612.

[13] CHO K，VAN MERRIENBOER B，BAHDANAU D，et al. On the Properties of neural machine translation：Encoder-decoder approaches [C]//Proceedings of the 8th Workshop on Syntax，Semantics and Structure in Statistical Translation. Stroudsburg：Associationfor Computational Linguistics，2014：103-111.

[14] HE K，ZHANG X，REN S，et al. Deep residual learning for image recognition [C]//IEEE conference on Computer Vision and Pattern Recognition（CVPR），2016：770-778.

[15] AGNIHOTRI M，CHUG A. A systematic literature survey of software metrics，code smells and refactoring techniques[J]. Journal of Information Processing Systems，2020，16(4)：915-934.

[16] 闫丽. 基于切片度量的软件模块复杂性研究[D]. 南京：南京邮电大学，2017.

[17] AHMED M，SERAJ R，ISLAM S M S. The k-means algorithm：A comprehensive survey and performance evaluation[J]. Electronics，2020，9(8)：1295.

[18] DHRUV P，NASKAR S. Image classification using convolutional neural network（CNN）and recurrent neural network（RNN）：A review[C]//Machine Learning and Information Processing：Proceedings of ICMLIP 2019，2020：367-381.

[19] National Institute of Standards and Technology. NIST Software Assurance Reference Dataset [EB/OL]. [2023-10-31]. https://samate.nist.gov/SARD.

[20] D'PEREIRA J，VIEIRA M. On the use of open-source C/C＋＋ static analysis tools in large projects [C]//2020 16th European Dependable Computing Conference（EDCC）. IEEE，2020：97-102.

[21] BASET A，DENNING T. IDE plugins for detecting in-put-validation vulnerabilitie[C]//IEEE Symposium on Security and Privacy Workshops，2017：143-146.