基于学习的源代码漏洞检测研究与进展

苏小红 郑伟宁 蒋 远 魏宏巍 万佳元 魏子越
（哈尔滨工业大学计算学部 哈尔滨 150001）

摘 要 源代码漏洞自动检测是源代码漏洞修复的前提和基础，对于保障软件安全具有重要意义.传统的方法通常是基于安全专家人工制定的规则检测漏洞，但是人工制定规则的难度较大，且可检测的漏洞类型依赖于安全专家预定义的规则.近年来，人工智能技术的快速发展为实现基于学习的源代码漏洞自动检测提供了机遇.基于学习的漏洞检测方法是指使用基于机器学习或深度学习技术来进行漏洞检测的方法，其中基于深度学习的漏洞检测方法由于能够自动提取代码中漏洞相关的语法和语义特征，避免特征工程，在漏洞检测领域表现出了巨大的潜力，并成为近年来的研究热点.本文主要回顾和总结了现有的基于学习的源代码漏洞检测技术，对其研究和进展进行了系统的分析和综述，重点对漏洞数据挖掘与数据集构建、面向漏洞检测任务的程序表示方法、基于机器学习和深度学习的源代码漏洞检测方法、源代码漏洞检测的可解释方法、细粒度的源代码漏洞检测方法等五个方面的研究工作进行了系统的分析和总结.在此基础上，给出了一种结合层次化语义感知、多粒度漏洞分类和辅助漏洞理解的漏洞检测参考框架.最后对基于学习的源代码漏洞检测技术的未来研究方向进行了展望.

关键词 软件安全；源代码漏洞检测；漏洞数据挖掘；漏洞特征提取；代码表示学习；深度学习；模型可解释性；漏洞检测
中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2024.00337

Research and Progress on Learning-Based Source Code Vulnerability Detection

SU Xiao-Hong ZHENG Wei-Ning JIANG Yuan WEI Hong-Wei
WAN Jia-Yuan WEI Zi-Yue
(*Faculty of Computing*, *Harbin Institute of Technology*, *Harbin* 150001)

Abstract Automatic detection of source code vulnerabilities is the precondition and foundation of source code vulnerability repair, which is of great significance for ensuring software security. Traditional approaches usually detect vulnerabilities based on the rules predefined by security experts. However, it is difficult to define detection rules manually, and the types of vulnerabilities that can be detected depend on the rules predefined by security experts. In recent years, the rapid development of artificial intelligence technology has provided opportunities to realize learning-based automatic source code vulnerability detection. Learning-based vulnerability detection methods are data-driven methods that use machine learning or deep learning techniques to detect vulnerabilities, among which deep learning based vulnerability detection methods have shown great potential in the field of vulnerability detection and have become a research hotspot in recent years due to their ability to automatically extract syntax and semantic features related to

收稿日期：2023-03-14；在线发布日期：2023-11-28.本课题得到国家自然科学基金项目(62272132)资助.**苏小红**(通信作者)，博士，教授，中国计算机学会(CCF)高级会员，主要研究领域为智能软件工程、软件漏洞检测、程序分析和软件测试等，E-mail：sxh@hit.edu.cn.**郑伟宁**，博士研究生，主要研究领域为软件漏洞检测.**蒋 远**，博士，助理教授，中国计算机学会会员，主要研究领域为程序分析、代码表示学习.**魏宏巍**，博士研究生，主要研究领域为软件数据挖掘、软件知识工程、基于搜索的软件工程、代码模式生成与搜索.**万佳元**，博士研究生，主要研究领域为软件漏洞检测和软件测试.**魏子越**，硕士，主要研究领域为智能合约软件漏洞检测.

vulnerabilities in source code to avoid feature engineering. This paper mainly reviews and summarizes existing learning-based source code vulnerability detection techniques, and provides a systematic analysis and overview of their research and progress, focusing on five aspects of the research work: vulnerability data mining and dataset construction, program representation methods for vulnerability detection tasks, traditional machine learning and deep learning-based source code vulnerability detection approaches, interpretable methods for source code vulnerability detection, fine-grained methods for source code vulnerability detection. Specifically, in the first part, we count existing publicly available vulnerability datasets, including their sources and sizes, and describe the challenges faced in building vulnerability datasets, as well as how to address these challenges. In the second part, we briefly introduce intermediate code representations and divide existing code representations applied in the field of vulnerability detection into four categories: metric based, sequence based, syntax tree based and graph based code representations. For each type of code representation method, we list some representative methods and analyze their advantages and disadvantages. In the third part, we introduce commonly used vulnerability detection tools and review coarse-grained vulnerability detection methods, including rule-based, machine learning based, and deep learning based vulnerability detection methods, and then analyze and discuss the characteristics, strengths and weaknessesof each type of vulnerability detection method. In the fourth part, we introduce interpretable methods that can further explain vulnerability detection results, briefly describe model self-interpretation methods, model approximation methods and sample feedback methods one by one, summarize their characteristics and discuss their strengths and weaknesses. In the fifth part, we first elucidate the problems and challenges posed by fine-grained vulnerability detection, and then provide a detailed description of existing representative methods for fine-grained vulnerability detection and their approaches to alleviate these challenges. Finally, we propose a source code vulnerability detection a framework that combines hierarchical semantic aware, multi-granularity vulnerability classification and assisted vulnerability understanding, and analyze its feasibility. We also prospect the future research directions for learning-based source code vulnerability detection techniques, such as the construction of large-scale, high-quality vulnerability datasets, techniques for detecting vulnerabilities in small or imbalanced samples, accurate and efficient vulnerability detection models, early detection techniques for vulnerabilities etc.

Keywords software security; source code vulnerability detection; vulnerability data mining; vulnerability feature extraction; code representation learning; deep learning; model interpretability; vulnerability detection

1 引 言

互联网是信息化时代不可或缺的基础设施[1]. 互联网技术在为人类带来了便利的同时,也给恶意分子提供了可乘之机. 近年来,黑客攻击[2]、数字资产盗窃[3]、用户隐私信息泄露[4]等网络安全事件频发,对信息系统的安全产生了严重的威胁. 作为网络空间核心组件的软件系统存在漏洞,是导致此类安全事件的根本原因[5].

软件漏洞(Vulnerability,也称脆弱性)是指在软件系统或产品的软件生命周期的各个层次与环节中,由于操作实体有意或无意的疏忽而产生的设计错误、编码缺陷和运行故障. 恶意主体可以利用软件漏洞实现获取更高级别系统权限、窃取软件中用户隐私数据等目的,从而危害软件系统的安全,并影响构建于软件系统之上的服务的正常运行[6]. 例如在2017年,Windows服务器消息块协议中的远程溢出漏洞引起了WannaCry勒索攻击[7],造成了全球性的互联网灾难.

2020年美国云视频会议企业Zoom发生大规模

用户的视频泄露事件，该漏洞使超过400万名Mac系统的Zoom用户受到影响，事件的起因在于Mac系统上的Zoom视频会议软件存在漏洞[8]. 2021年12月10日Apache开源项目Log4j被披露存在“核弹级”远程代码执行漏洞，导致攻击者可以通过构造恶意请求在目标服务器上执行任意代码，从而实施窃取数据、挖矿、勒索等行为[9]. 国际权威漏洞数据库CVE[10](Common Vulnerabilities & Exposures)和美国国家漏洞数据库NVD[11](National Vulnerability Database)中披露的统计数据如图1所示，可以看出近几年被披露的软件漏洞数量存在逐年上升的趋势，尤其是2017年后披露的漏洞数量是前几年的两倍多. 软件漏洞已成为软件与信息系统安全的重要隐患之一.

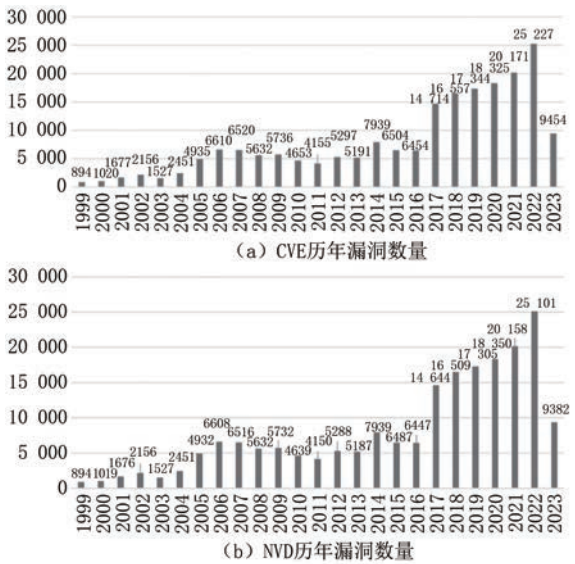

图1　在CVE和NVD中披露的漏洞数量

软件静态漏洞检测方法可以有效提高软件质量、减少软件安全漏洞并降低安全风险，已引起学术和工业界的广泛关注[12]. 依据分析对象可以将软件静态漏洞检测分为二进制漏洞检测和源代码漏洞检测[5]，本文主要对现有的源代码漏洞检测方法进行研究综述. 按照检测过程中使用的技术，又可以将源代码漏洞检测方法分为基于规则的漏洞检测方法[13-27]、基于传统机器学习的漏洞检测方法[28-44]和基于深度学习的漏洞检测方法[45-71]，后两种可以统称为基于学习的漏洞检测方法.

基于规则的源代码漏洞检测方法(例如某些开源或商业漏洞检测工具)依赖于安全专家定义的安全规则，但规则的局限性和不完善性往往会导致误报或漏报[71-73]，而制定出完备且实用的漏洞检测规则又需要较高的人工成本. 随着安全漏洞披露数量的不断积累，基于学习的源代码漏洞自动检测逐渐成为可能，这类方法能够从海量历史数据中自动学习漏洞模式，避免人工制定规则，因此成为当前软件和网络空间安全领域中最热门的研究方向之一.

本文对基于学习的源代码漏洞检测技术进行了系统的分析，着重介绍了漏洞数据集的挖掘和构建方法、面向漏洞检测任务的程序表示和程序表示学习方法、基于传统机器学习和深度学习的源代码漏洞检测方法、源代码漏洞检测的可解释方法以及细粒度的源代码漏洞检测方法. 通过分析现有方法，本文归纳总结了当前漏洞检测领域面临的挑战，并给出了一种结合层次化语义感知、多粒度漏洞检测和辅助漏洞理解的源代码漏洞检测参考框架. 最后，本文对未来的研究方向和发展趋势进行了展望.

2　文献统计

本文进行文献检索和筛选的过程如下：

检索数据库：对于外文文献，以谷歌学术为主，EI工程索引和SCI科学引文索引为辅. 对于中文文献，以中国知网为主，以万方数据和维普中文科技期刊为辅.

检索关键词：包括“源代码漏洞检测”、“漏洞检测可解释性”相关的中英文关键词；检索时间范围为2000年1月1日至2023年5月20日.

使用上述关键词按照年份在所列的检索数据库中进行检索，对检索结果进行人工核查，具体包括检查文献标题、关键词、摘要以及浏览文献内容，筛选出与本文研究主题相符的文献(即源代码漏洞检测相关的文献)，同时进行分类(包括文献出处、是否是基于学习的漏洞检测方法等)，若连续5页的网页检索结果列表中均未出现相符的文献，则该年份的检索工作完成. 此工作由三位安全领域的研究人员(3名博士研究生)完成，每人平均耗时75个小时.

基于上述步骤，本文最终汇总了1109篇与源代码漏洞检测相关的论文，其中有768篇研究型论文，186篇实证分析型论文和155篇综述型论文. 如图2所示，漏洞检测领域的文献数量随着时间推移呈现出波动性增长的趋势，且在近三年达到峰值(2023年由于只对1~5月间的文献进行了调研，所以数量相对较少)，这表明该方向已成为近几年的一个研究热点.

针对研究型论文，本文继续对其中基于传统机器

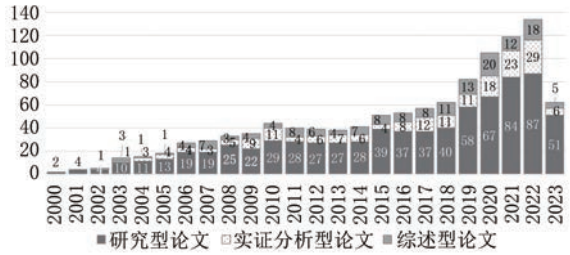

图2 源代码漏洞检测方法的文献分类统计(截至2023年5月20日)

学习和基于深度学习的源代码漏洞检测的文献数量进行了统计,结果如图3所示.可以看出,从2017年开始,深度学习技术逐渐应用于漏洞检测领域,2018年其论文数量就已经超过了基于传统机器学习的漏洞检测的文献数量.

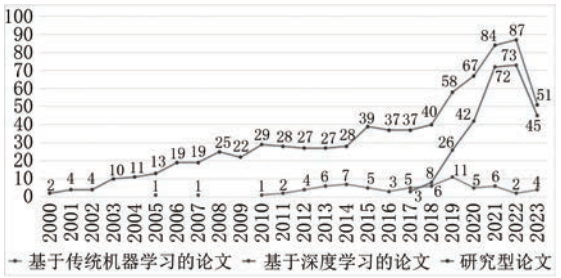

图3 源代码漏洞检测研究型论文信息汇总

进一步地,本文还对中国计算机学会推荐的A类国际学术会议和期刊列表中的文献及国内计算机三大学报文献进行了统计,结果如图4所示.从图中可以看出,漏洞检测仍是目前网络信息安全和软件工程领域的研究热点,并且在其他领域的期刊和会议中也有漏洞检测相关的研究成果.

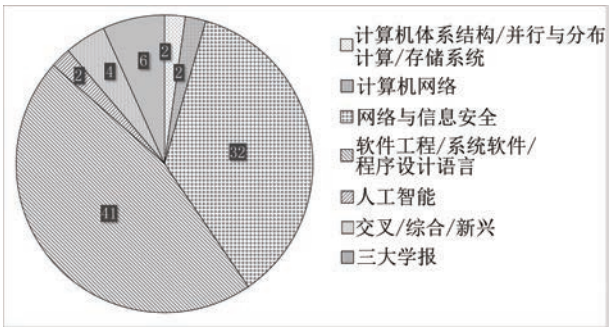

图4 源代码漏洞检测方法的顶级文献所属领域汇总

本文还对漏洞检测方法涉及的编程语言进行了统计,结果如图5所示.从图中可以看出,目前的源代码漏洞检测方法主要针对的是C/C++代码,Java、PHP、JavaScript等次之,其他编程语言和多语言的代码则涉及得较少.

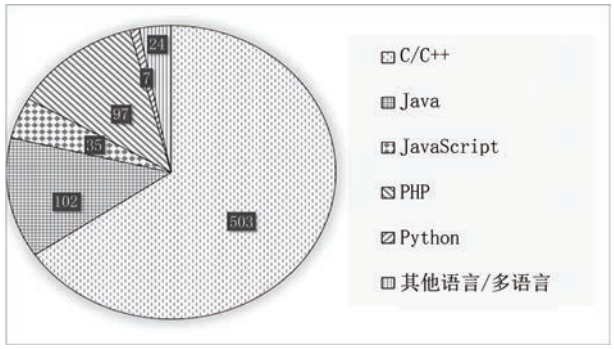

图5 源代码漏洞检测方法涉及的编程语言统计

3 相关定义、问题与挑战及研究内容

3.1 相关定义

定义1. 抽象语法树[28](Abstract Syntax Tree,AST).AST是源代码抽象语法结构的树形表示形式,是一种有序树结构的程序中间表示,其内部节点对应程序中的操作符,叶子节点对应于操作数(例如,常量或标识符).

定义2. 控制流图[49,74](Control Flow Graph,CFG).CFG是一个有向图,具有唯一的入口节点START和唯一的出口节点STOP,除了入口/出口节点外,其余中间节点表示程序中的语句或谓词表达式,其中谓词是指能够返回True或者False的操作(operation)[75],谓词表达式则是指包含了谓词的表达式.边表示语句间的控制流关系,也称为控制流边.另外,对于图中任一中间节点,均至少存在一条从入口节点到该节点,再从该节点到出口节点的路径.

定义3. 控制依赖[49,74](Control Dependency).若CFG中存在一条从节点A到节点B的有向路径,且路径中任何节点(不包括A和B)都被B后支配,并且节点A不被节点B后支配,则节点B控制依赖于节点A.这里,后支配是指若从节点C到出口节点的每条有向路径都包含节点B,那么节点B后支配节点C.需要说明的是,后支配不包括出口节点,并且一个节点不会对自己后支配.

定义4. 数据依赖[49,74](Data Dependency).若CFG中的节点A存在一条路径到它的另一个节点B,且在节点A处定义的值在节点B中使用,那么节点B数据依赖于节点A.

定义5. 程序依赖图[49,74](Program Dependency Graph,PDG).PDG是一个有向图,其节点和控制流图中的节点相同,但不包含入口/出口节点.PDG节

点相连的边表示节点之间存在的控制依赖关系和数据依赖关系.

定义6. 代码属性图[28](Code Property Graph, CPG). CPG是通过合并抽象语法树,控制流图和程序依赖图获得的程序的图形表示,其形式化定义为$CPG=(V,E)$,V是由控制流图和程序依赖图中的节点以及语法树中的语法结构节点组成的非空有限集合,E是有向边的集合,代表节点之间的控制依赖,数据依赖,控制流以及语法关系.

3.2 基于学习的源代码漏洞检测方法面临的问题及挑战

(1) 基于学习的源代码漏洞检测方法训练所需的漏洞数据集不足

与基于专家预定义规则的漏洞检测方法不同,基于学习的源代码漏洞检测方法需要足够的样本数据对模型进行训练来提高检测性能. 然而,由于真实项目的漏洞代码不易获取,因此目前仍缺乏高质量大规模的真实漏洞数据集. 使用小规模的数据集训练模型,极易导致模型过拟合,从而影响模型的泛化能力. 目前大部分研究都使用已公开的人工合成或半合成的漏洞数据集. 尽管这些数据易于获取,且规模较大,但是和真实项目相比,其在代码复杂度和漏洞模式多样性方面仍有显著的差距. 如前文所述,随着软件规模的增大,软件中可被利用的漏洞类型及数量也在不断增加. 新的漏洞利用和攻击方式的不断出现,使得基于学习的漏洞检测模型的泛化能力受到了越来越严峻的挑战,而扩充数据集对模型进行补充训练是提高泛化能力最直接有效的手段. 因此,如何构建高质量、大规模且类型足够丰富的漏洞数据集是基于学习的漏洞检测方法面临的一个挑战性问题.

(2) 基于学习的源代码漏洞检测方法在深层漏洞语义理解和复杂漏洞特征提取上的局限性

基于学习的源代码漏洞检测方法需要理解和学习程序语义,自动捕获代码中的漏洞特征,然而在训练过程中,由于硬件的限制,输入模型的代码向量表示需要限制在一个固定的长度,对于规模较大且漏洞语句分布在函数尾部的代码,其超出限制的代码信息将会被截断,使得模型无法学习代码的完整语义信息. 其次,程序中代码元素(例如,Token、语句等)往往存在着大量的上下文依赖关系,模型需要有选择地保留和学习其中更为重要的和漏洞相关的上下文依赖关系,以有效识别漏洞模式. 然而,真实项目中的漏洞模式通常较为复杂,使得基于学习的漏洞检测方法难以准确有效地学习代码的深层漏洞语义. 因此,如何构建能够提取复杂漏洞特征的检测模型是基于学习的漏洞检测方法面临的另一个挑战性问题.

(3) 基于学习的源代码漏洞检测方法的可解释性较差

长期以来,深度学习的黑盒问题一直是学术界的一大难题,对于基于学习的源代码漏洞检测方法更是如此. 粗粒度的源代码漏洞检测方法在识别出有漏洞的函数或代码段后,并不能提供更多关于漏洞的信息,而且模型本身的“黑盒”性使得模型的检测机制和检测结果都难以被充分解释. 因此,将基于学习的漏洞检测方法“白盒化”,使检测过程和检测结果具有可解释性,具有重要意义. 目前关于深度学习的可解释性研究已有了一定的发展,但将其应用于漏洞检测领域的研究仍然较少. 因此,提高基于学习的漏洞检测方法的可解释性是目前面临的一个具有较大挑战性的问题.

3.3 基于学习的源代码漏洞检测的研究内容

源代码漏洞检测是开发人员或安全专家以某种方式在源代码中发现已经存在但未暴露的漏洞的过程. 根据检测的粒度,可以将源代码漏洞检测方法分为粗粒度的漏洞检测和细粒度的漏洞检测. 其中,粗粒度漏洞检测指预测源代码文件、函数(或方法)或代码片段中包含漏洞的可能性,细粒度漏洞检测指预测源代码中可能引发漏洞的具体语句.

粗粒度的检测难度低、速度快、准确度高,但是检测结果缺乏可解释性,易延误漏洞修复的时机、增加修复成本. 因此,研究人员在粗粒度源代码漏洞检测的基础上,进一步提出了一些有助于理解粗粒度检测结果的可解释方法.

相比于粗粒度漏洞检测,细粒度漏洞检测的难度更大,但由于可直接定位到漏洞发生的语句位置,因此可以更好地辅助开发人员理解和修复漏洞.

基于学习的源代码漏洞检测技术通过传统机器学习或深度学习等人工智能技术对源代码进行分析、抽象和推理,使其能够从大量历史数据中自动或半自动地学习源代码中和漏洞相关的复杂语义特征以生成相应的漏洞模式,并应用于粗粒度或细粒度的漏洞检测任务中. 图6显示了现有的基于学习的源代码漏洞检测技术相关的各个研究内容之间的关系:

(1)漏洞数据集的挖掘和构建

从开源软件库或公开漏洞数据库中收集可疑的

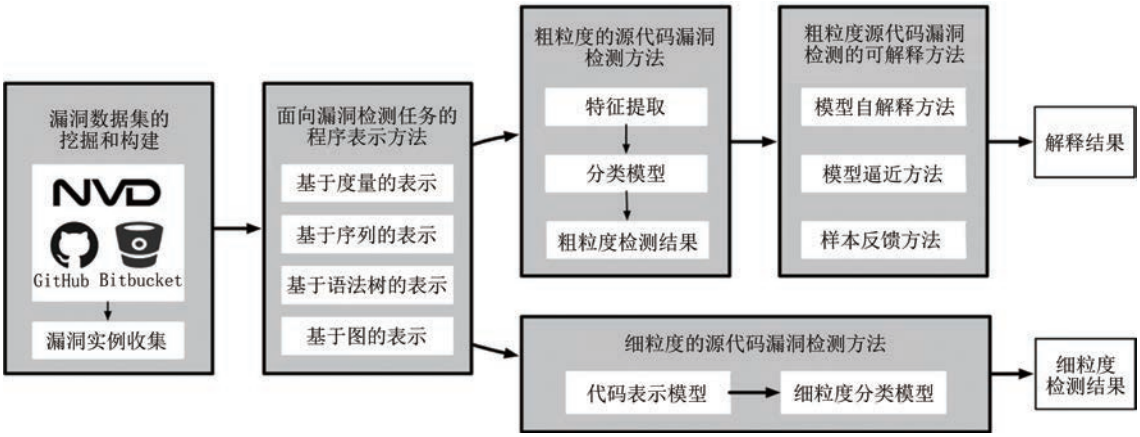

图6 现有基于学习的源代码漏洞检测方法各研究内容之间的关系

漏洞程序,以人工或自动化的方式标记漏洞程序,从而构建漏洞数据集.

(2)程序表示方法

对数据集中的源代码进行解析,生成合适的程序中间表示,例如基于度量、基于序列、基于语法树和基于图的程序表示形式.

(3)粗粒度的源代码漏洞检测

在得到程序的中间表示形式后,利用人工制定的规则提取软件度量特征,或者使用合适的深度学习网络从程序表示中提取与漏洞相关的语法和语义特征,将其送入分类器中,以二元分类方式预测待检测的源代码是否包含有漏洞.

(4)粗粒度源代码漏洞检测的可解释方法

对于检测为包含漏洞的源代码,通过可解释方法进一步给出概率化或细粒度的解释信息.常见的可解释方法主要分为模型自解释方法、模型逼近方法以及样本反馈方法等三类.

(5)细粒度的源代码漏洞检测方法

在得到程序的中间表示后,通过对源代码的程序中间表示进行表示学习,直接获得语句级的细粒度检测结果,给出漏洞语句的位置.

综上,基于学习的漏洞检测研究主要围绕如下五个难点问题展开研究:如何构建大规模且高质量的漏洞数据集?如何将代码解析成合适的程序表示?如何从程序表示中提取漏洞特征以实现粗粒度的漏洞检测?如何在粗粒度漏洞检测结果的基础上,获得具有可解释性的结果?如何基于程序表示对细粒度的漏洞特征进行建模以实现细粒度的漏洞检测?因此,本文将从这五个角度进行综述.

4 漏洞数据集的挖掘和构建方法

4.1 已公开的漏洞数据集

漏洞数据集的构建是实现基于学习的源代码漏洞检测和定位的前提和基础,基于学习的漏洞检测需要以高质量的漏洞数据为前提,数据集的规模和质量直接影响检测模型的泛化能力.有研究表明,提升训练数据集中所包含的漏洞类型和语法结构的多样性,有助于增强对未知漏洞的检测能力[6].

部分较为关键且公开的漏洞数据集如表1所示,漏洞数据集的公开推动了基于学习的漏洞检测技术的发展,但数据集构建仍面临以下挑战:

(1)数据集的样本不平衡问题

在最近的一项研究中,Yang等人[82]讨论了数据采样方法针对漏洞检测数据不平衡问题的有效性.具体来说,研究主要评估了四种数据采样方法,包括随机欠/过采样方法、SMOTE方法和OSS(One Side Selection)方法[83]对深度学习漏洞检测模型的有效性和学习代码漏洞模式的能力的影响,并得到了如下结论:首先,数据采样方法确实可以缓解漏洞检测中的数据不平衡问题.其次,过采样要优于欠采样.最后,对于原始样本的采样要优于对模型学习样本后生成的特征空间的采样.因此,未来可以重点针对原始样本的数据过采样方法进行研究.

在真实项目中,漏洞样本中的漏洞语句数量也相对较少,这导致了样本不平衡问题在软件细粒度漏洞定位任务上更为严重.尽管研究人员已经意识到该问题给漏洞定位任务所带来的挑战[84-87],但很少有研究给出相应的解决方案.软件故障定位是指定位程序中导致程序运行时出现错误的语句位置,

表1 已公开的漏洞数据集

标签粒度	数据集中漏洞与非漏洞实例之比	数据来源	文献
函数	320∶6166	NVD&CVE	CCS[45]
	457∶32 531	NVD&CVE	TII[76]
	83 710∶52 290	SARD	TDSC[77]
	479∶33 343	NVD&CVE	
	83 710∶52 290	SARD	ICICS[78]
	1471∶59 297	NVD&CVE	
	1471∶59 297	CVE	NCA[79]
	12 303∶21 057	SARD	ICSE[46]
	27 652∶31 313	GitHub	NIPS[47]
	1658∶16 511	Debian&Chromium	TSE[73]
代码段	17 725∶43 913	NVD&SARD	NDSS[48]
	56 395∶364 232	NVD&SARD	TDSC[49]
	43 119∶138 522	NVD&SARD	TDSC[50]
语句	13 437∶80 087	GitHub	WPC[80]
	5393∶6503	SARD	
	52 333∶730 160	GitHub	ICMLA[51]
	30 078∶461 795	Debian	
	18 653∶1 276 970	GitHub	ICSE-SEIP[81]

主要关注导致测试用例失败的运行时缺陷，也同样面临样本不平衡问题，即失败的测试用例较少. 虽然软件故障定位不同于漏洞定位，但考虑到二者任务上的相似性，软件故障定位领域中用于缓解样本不平衡问题的代表性方法依然可以被漏洞定位任务所借鉴.

一些文献通过对失败的测试用例进行克隆来扩充数据集，解决样本不平衡问题[88-89]. Xie 等人[90]提出了一种数据增强方法 Aeneas，该方法利用主成分分析（PCA，Principal Component Analysis）技术生成缩小的特征空间，通过条件变分自动编码器（CVAE，Conditional Variational Autoencoder）在缩小的特征空间中合成失败的测试用例，解决样本不平衡问题. 其优势在于利用 PCA 技术降低了特征空间的维度，简化了数据特征的表达方式，从而提高了数据合成的效率. Hu 等人[91]提出的 Lamont 方法也采用了类似的思路，使用线性判别分析（LDA，Linear Discriminant Analysis）技术降低特征空间的维度，然后利用 SMOTE 技术合成失败的测试用例以获得平衡的样本数据. Lei 等人[92]提出了一种基于类间学习的数据增强方法 BCL-FL，通过一个专门设计的数据合成公式将成功的测试用例和失败的测试用例进行混合，生成更贴近真实测试案例的失败测试用例. 因此，未来可以考虑借鉴上述用于增强或合成失败测试用例的思路，对漏洞样本进行增强，以解决漏洞样本不平衡的问题.

（2）数据集的数据质量问题

目前公开的漏洞数据集覆盖的编程语言和漏洞类型有限，使得数据集的泛化性较低，且因缺少完整的漏洞上下文使其只能表示有限范围内的漏洞模式. 例如，函数粒度的漏洞数据集无法提供跨函数漏洞代码的完整结构.

从同一项目中挖掘的漏洞还可能因跨版本或代码重用等原因而造成数据冗余问题，不同项目团队的开发人员代码风格的差异以及不同项目代码应用背景的不同，还使得跨项目的漏洞数据因数据分布的差异而难以学习[93].

此外，数据集的标签也往往存在噪声. 这是因为若采用手工标注，则标签的准确性取决于安全专家的专业知识，而使用静态分析工具进行标注，又会带来大量的错误标签，而且数据的时效性问题还可能使得数据集中存在潜在的尚未被发现或已被静默修复的漏洞，导致标签错误.

以上数据集相关的因素都显著增加了训练泛化能力更强的检测模型的难度，已成为漏洞检测模型性能提升的瓶颈问题.

（3）数据集的来源问题

目前大部分软件代码仍是不开源的，即便是公

开的漏洞报告大部分也不会公布漏洞代码，增加了研究人员从数据源中获取数据的难度．因此，目前的漏洞数据大部分来自于SARD（Software Assurance Reference Dataset）[94]的合成数据，少部分来自于NVD、CVE、GitHub[95]等真实项目的数据，还有部分数据集是二者的混合数据．

来自SARD的合成数据集中包含了人工合成或半合成的漏洞数据[73]，人工合成的数据集如SATE IV Juliet数据[96]是用模拟已知真实漏洞代码模式而人工合成的漏洞代码构造的公共漏洞数据集．合成数据具备样本数量多、类型多、噪声小、成本低等优点，被众多研究人员广泛使用．相比于真实代码，合成代码更简单、更独立，代码模式的变化更少，漏洞上下文更纯净，因此其漏洞特征更易于学习[78]．然而，合成数据与真实项目代码在复杂程度上存在较大差距，程序语法结构覆盖不全，无法揭示现实场景中真实的漏洞分布，无法反映真实项目场景，因此用其训练的模型难以准确检测真实代码中的漏洞．半合成数据集[73]是对真实代码进行简化和修改以服务于学术研究等目的的数据集．例如，SARD数据集中的testID：151 455为典型的半合成实例．由于在半合成数据集中，研究者往往会突出原始代码中有漏洞的部分，因此该数据集也不能完全代表现实世界的漏洞代码．

（4）数据集的标注问题

基于合成或半合成数据集训练的模型难以适应现实世界项目中复杂代码的漏洞检测场景，因此挖掘并标注真实软件项目中的漏洞代码实例以构建真实漏洞数据集势在必行．Dowd等人[97]发现，一个小时的安全检查平均仅可以覆盖500行代码，大多数现代软件系统一般会包含数百万行代码．因此，人工标注真实漏洞实例的成本较高，难以得到大量真实的漏洞实例．

针对该问题，部分研究聚焦于采用自动化的方式来标注漏洞数据集，例如利用静态分析工具的检测结果对开源项目代码进行漏洞标记[51,80-81]，但是静态分析检测工具的高误报率使其获得的漏洞标签可靠性较低．

更常用的漏洞数据自动标注方法是从国际权威漏洞数据库CVE入手收集有真实漏洞标签的数据，将CVE公开披露的每个漏洞数据条目中提供的代码提交（Commit）链接定位到开源代码仓库中安全相关的提交日志，再通过提交代码修复前后的差异分析分别提取出漏洞代码和补丁代码，从而创建漏洞数据集[45,48-50,76-79,98]．其中的一部分数据集[45,76,79]是由完全来自真实项目的漏洞实例构建的，但是其中包含的漏洞实例数量要远少于已公开的同时包含来自SARD和NVD的合成数据以及来自CVE的真实数据的混合漏洞数据集[48-50,77-78,98]．因为除了真实项目的漏洞实例外，另一部分数据集中还包括了从SARD和NVD数据库中抽取的人工合成漏洞代码．虽然使用程序切片技术将漏洞代码拆分为多个切片代码段，增加了数据集中漏洞实例的数量，但从同一个原始漏洞代码中抽取出来的不同切片代码段都具有相同的漏洞类型，因此这种以切片代码段为单位构建漏洞数据集的方法并不能为数据集增加新的漏洞类型．

也有研究尝试了半自动的漏洞数据集标注方法．例如，Zhou等人[47]采用先利用关键词筛选出和安全相关的代码提交、再由安全专家人工确认的方式，构造了4个全部来自实际开源项目的漏洞数据集，这些数据集比前面介绍的数据集包含了更多来自真实项目的漏洞实例，但目前只公开了其中的两个．Chakraborty等人[73]则是分别从Bugzilla[99]和Debian security tracker[100]中爬取有安全标签的故障（Bug）报告，再通过这些故障报告来搜集真实项目中的漏洞代码及补丁代码．

综上，目前尚缺乏被公认的可用于评估检测性能的公共基准漏洞数据集[78]，语句级标注的细粒度漏洞数据集尤为稀缺．其次，缺乏公认的高质量的真实漏洞数据集以及能够准确评估数据集质量的方法．数据的泛化性、冗余性、完整性以及标签的准确性和可靠性等问题使得漏洞数据集的质量还有待提升．再次，目前已公开的漏洞数据集大部分为混合数据集，其中包含真实漏洞实例的比例偏低，且存在漏洞类型多样性不足、漏洞上下文语法结构覆盖不够丰富等问题，不利于模型学习更丰富的漏洞模式，难以满足实际应用对模型检测未知类型漏洞的高泛化性需求，使得基于现有的数据集训练得到的漏洞检测模型难以在工业界实际应用．因此，缺少大规模、高质量、来自真实项目的源代码漏洞公共基准数据集，是当前漏洞检测面临的主要挑战之一．

4.2 漏洞数据挖掘方法

基于规则的方法，即利用关键字筛选和差异文件分析收集漏洞数据的方法，往往只能从代码仓库中抽取少量的漏洞实例．为了从GitHub等开源代码仓库中挖掘更多的漏洞数据以解决大规模高质量漏洞数据集的创建问题，一些研究者开始尝试采

用代码变更意图识别或安全相关的代码提交(Commit)识别方法来挖掘漏洞数据[47,73,81]. 表2列出了与故障或漏洞修复的代码提交识别相关的文献.

表2　Bug或漏洞修复的Commit识别的漏洞数据挖掘方法

任务	文献	研究机构
故障修复相关代码提交识别	ESEM[101] 2019	美国北达科他州立大学
修复的故障类型识别	JSS[102]2020	中国扬州大学
通过安全相关代码提交识别来挖掘漏洞数据	CCS[103]2015	德国波恩大学
	ESEC/FSE[104]2017	美国SourceClear公司
	ICSME[105] 2019	德国SAP安全研究所
	TIFS[52]2020	中国西北大学
	SQJ[106]2021	美国北卡罗来纳州立大学

其中，Zafar等人[101]设计了基于预训练BERT (Bidirectional Encoder Representation from Transformers)的故障修复相关代码提交识别方法，通过预测用户的代码提交是否包含故障修复意图，进而判断目标版本的代码是否包含故障. 为了进一步识别代码提交中涉及的具体故障类型，Ni等人[102]基于代码差异文件构造修复树，并使用基于语法树的卷积神经网络(Tree-based Convolutional Neural Network)来对代码提交的具体故障类型进行分类.

由于漏洞(漏洞是安全相关的缺陷)不同于一般意义上的缺陷，为了识别和漏洞修复相关(vulnerability-relevant)的代码提交，在借鉴上述故障修复相关的代码提交分类方法的基础上，Perl等人[103]提出基于文本挖掘和机器学习的安全相关代码提交识别方法，先使用文本挖掘技术从代码提交中提取修复意图特征，然后使用SVM对提取的特征进行分类，最后根据预测结果判断该代码提交是否和漏洞相关. 为了结合不同文本分类器在安全相关的代码提交识别任务上的优势，Zhou等人[104]利用逻辑回归组合6种不同的机器学习分类器，相比于使用单个文本分类器获得了更好的分类效果. Wang等人[52]则采用一致性预测评估多个分类模型的置信度，并采用投票策略组合置信度较高的多个分类器的预测结果，来提高模型的整体识别准确率. 相比于上述工作仅使用了代码提交文本信息的问题，Sabetta等人[105]还考虑了从代码中提取修复相关的代码特征，并利用文本分类器和差异代码分类器来综合评估当前用户的提交是否和漏洞修复相关. 考虑到代码仓库中的问题报告(issue report)也往往包含了大量的漏洞实例，Oyetoyan等人[106]开发了一种能够同时识别安全相关的问题报告和代码提交的方法，该方法结合了关键词筛选和TF-IDF (Term Frequency-Inverse Document Frequency)[107]方法. 其中，TF表示一个词在文档中出现的频率即词频，IDF为包含该词的文档频率倒数的对数即逆文档频率，TF-IDF是TF和IDF的乘积，代表该词对于文档的重要程度. 因此，通过计算TF-IDF可以得到对文档分类更重要的词即关键词. Oyetoyan等人使用关键词筛选和TF-IDF提取问题报告和代码提交中与安全相关的关键词，然后以此为特征利用机器学习算法识别问题报告和代码提交的内容是否与安全相关.

虽然开源代码仓库中存在大量可利用的数据资源，但是其中的提交日志和代码变更并非都与漏洞修复相关，部分提交日志甚至出现漏洞修复相关的提交和其他类型的提交混杂的情况，这种混杂的情况进一步增加了准确识别漏洞修复相关的提交日志的难度. 此外，由于一些复杂的漏洞通常不是一次性被修复，而版本跟踪的复杂性又进一步加大了准确识别漏洞被引入和被修复的版本的难度[93]. 因此，如何通过漏洞的版本溯源分析来挖掘漏洞代码还有待进一步研究. 此外，出于潜在商业敏感性以及安全和隐私方面的考虑，很多漏洞会被隐秘地修复，以避免攻击者对已公开漏洞的利用，这些都增大了从软件仓库中挖掘高质量真实漏洞数据的难度. 目前，以扩充高质量漏洞数据集为目的来研究安全相关代码提交识别方法的文献相对较少. 如何从多源异构的软件仓库中挖掘漏洞数据以构建大规模高质量的真实漏洞数据集，还有待深入研究.

5　面向漏洞检测任务的程序表示方法

尽管自然语言和程序编程语言之间有一些相似之处，但也存在明显的差异，从语言形式来看，程序

中包含了丰富而明确的结构信息，尽管这种结构也存在于自然语言中，但并不像程序中那样严格[108]. 此外，程序语义之间的相邻关系也和自然语言有所不同，例如，循环内外的语句虽然在距离上是相邻的，但从语义上来看并不相邻[109]，因此需要先将程序转换为合适的中间表示形式以建模这种语义信息[110]. 常见的代码中间表示形式有Token序列、抽象语法树(Abstract Syntax Tree，AST)、控制流图(Control Flow Graph，CFG)、数据流图(Data Flow Graph，DFG)、程序依赖图(Program Dependency Graph，PDG)等，这些不同的代码中间表示形式提供了不同的抽象级别[111]. 代码中间表示的抽象级别越高，其表示代码语义信息的能力越强[112]. 例如，Token序列能够体现代码的自然语序信息和词法统计信息，AST擅长表征特定的编程模式和语法结构的相似性[53,112]，而CFG和DFG能够比Token序列和AST表示更多的控制流和数据流信息，PDG则能更好地表示程序中变量之间的依赖关系等语义信息[113]. 其次，不同的代码中间表示对于特定的软件工程任务的重要程度有所不同，某一种代码中间表示可能只适合某一类或者某几类的软件工程任务. 因此，需要研究适合漏洞检测任务的程序表示方法.

早期的基于机器学习的漏洞检测方法(例如基于度量的程序表示方法)通常使用特征工程来人工提取漏洞相关的特征. 近年来，研究人员开始使用深度学习方法来自动提取漏洞特征. 由于程序在编译的不同阶段会有不同的程序表示形式. 因此，如图7所示，按照代码表示的组织形式，可将基于深度学习的程序表示方法分为三类：基于序列的程序表示方法、基于语法树的程序表示方法和基于图的程序表示方法. 程序表示方法的目标不同，适合提取的漏洞特征也有所不同. 表3列出了应用于漏洞代码检测任务的以上四类程序表示的代表性文献.

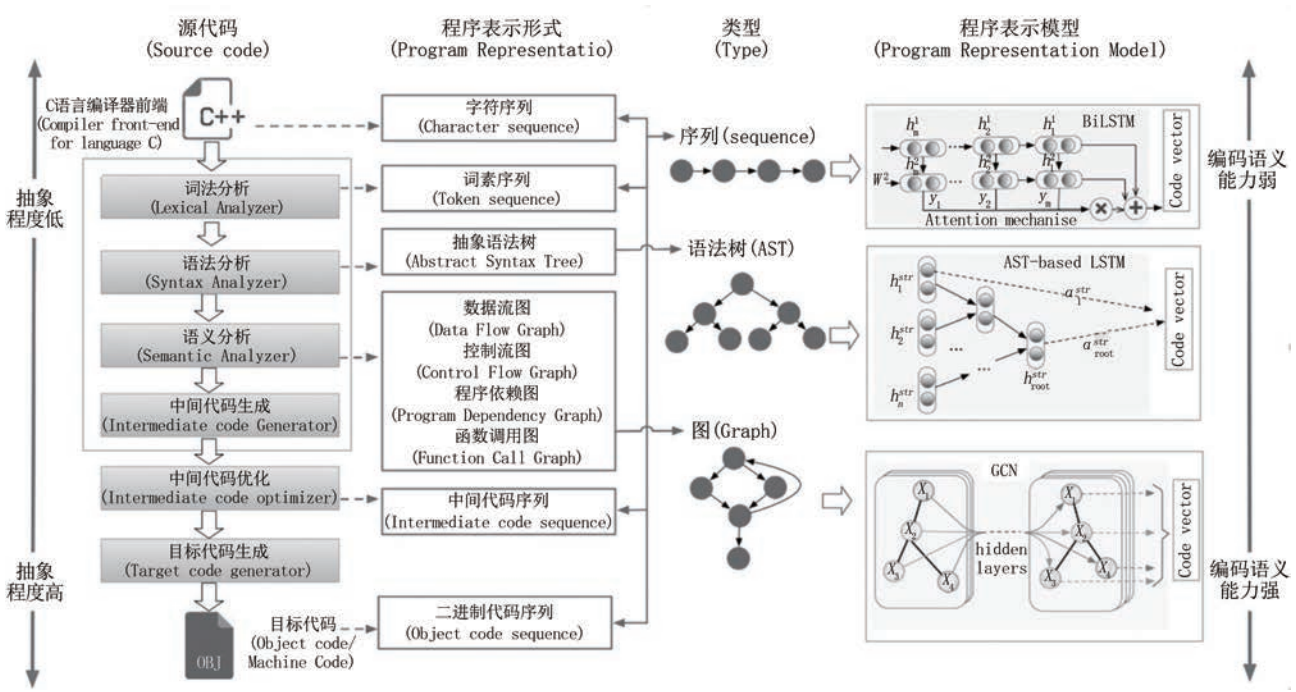

图7 不同抽象级别的代码中间表示形式与具有不同编码语义能力的程序表示模型的对应关系

5.1 基于度量的程序表示方法

软件度量是指通过对代码的质量、复杂性和可维护性等代码结构信息的度量，来识别软件中难以理解或难以维护的代码区域，以便后续更复杂的代码分析. 基于度量的漏洞检测方法的主要思路是根据软件度量构建一组代码特征值，然后将这组特征值作为机器学习模型的输入，从而对存在于结构度量和缺陷之间的复杂关联关系进行学习，并利用这些知识来预测新代码中出现缺陷的可能性.

在早期的工作中，研究人员使用的通常都是较为经典的软件度量. 例如，代码扰动(Code-churn)[29,31,37-38,103,115]、代码复杂度(Code-complexity)[29-32,37-39]、覆盖率(Coverage)[29,31]、依赖性(Dependency)[29,31]、组织(Organizational)[29,31]、开发者活动(Developer-activity)[36-37,103,114-115]等. 然而，已有的实证研究表

表3 程序表示方法的代表性文献

分类	子类	代码表示
基于度量	经典的软件度量	开发者活动[36,114]
		代码扰动、复杂度、覆盖率、依赖、组织[29,31]
		复杂度、代码扰动、故障历史[38]
		代码复杂度、代码扰动、开发者活动[37]
		代码扰动、开发者活动[103,115]
		复杂度[30,39,116]
	改进的软件度量	环复杂度、循环复杂度、函数依赖性、函数指针使用、控制结构的依赖性[117]
		切片软件度量[32,118]
基于序列	基于函数调用序列	函数调用序列[54,119]
	基于源代码序列	源代码文本Token序列[51,55,120]
	基于代码段序列	代码段Token序列[48-50]
	基于中间代码序列	中间代码Token序列[98]
	基于汇编代码序列	汇编代码Token序列[56,121]
基于语法树	语法树结构转化为序列	AST路径[122-123]AST Token序列[45,53]
	直接对语法树结构建模	CFAST[124]
基于图	图转化为序列	CPG[61-62]
	直接对图结构建模	复合图[47]; 程序图[52]; CCG[64]; 综合图[65]; XFG[68]; SPG[60]; 切片图结构[125]

明，经典的软件度量并不适用于源代码漏洞检测[31,39,126].

为了进一步提高基于度量的漏洞检测方法的准确性，使软件度量能够准确刻画漏洞代码的特征，研究者对软件度量的统计和提取方法进行了改进. 例如，除了代码复杂度等经典的软件度量外，Du等人[117]还提取了函数依赖性、函数指针使用和控制结构的依赖性等能够反映漏洞代码特征的软件度量，来辅助识别漏洞函数. 另一方面，函数内大量与漏洞无关的语句也对提取漏洞特征产生了干扰. 为了解决这一问题，Zagane等人[118]提出了对程序进行切片再从切片代码段上提取软件度量的漏洞检测方法. Salimi等人[32]的实验结果表明，在切片上提取的软件度量要比在函数上提取的度量能更准确地刻画漏洞特征.

5.2 基于序列的程序表示方法

基于序列的程序表示是将代码转化为Token序列，然后利用基于序列的深度神经网络对其进行建模，以生成代码的向量表示. 主要包括以下几种：

基于函数调用序列的方法：基于函数调用序列的表示方法通过解析源代码来提取其中的函数调用序列，再将其转化成向量表示. 例如，Grieco等人[119]通过N-gram[127]和word2vec[128]模型将其转化为低维的向量表示，Wu等人[54]则是通过为每个单词建立唯一的整数索引来生成代码的数值向量表示. 然而，这类方法的目标是希望能够实现快速有效检测漏洞的轻量级方法[126]，能够检测的漏洞类型有限.

基于源代码Token序列的方法：基于源代码Token序列的程序表示方法通过词法分析技术将代码转换为Token序列，然后利用词袋或N-gram或word2vec将代码文本的Token序列中的每个Token都转化为向量表示[51,55,120]. 然而，该类方法难以学习代码的高级语法和语义信息.

基于代码段Token序列的方法：所谓代码段，指利用语法和语义分析从代码中提取的语义相关但不一定连续的多行代码. 该类方法中最有代表性的工作是Li等人提出的以代码段(code gadget)来表征程序的漏洞检测方法VulDeePecker[48]、SySeVR[49]、μVulDeePecker[50]. 该类方法首先通过不同的方式提取代码段，随后将代码段表示为Token序列，再将其编码为向量表示. 具体地，VulDeePecker[48]利用静态分析工具(如Checkmarx)解析程序源代码，根据规则从代码中提取不安全库/API函数调用相关的漏洞候选关键点，采用程序切片技术从AST和PDG中提取和关键点有数据依赖关系的代码行组成代码段，然后将代码段转化为Token序列，再使用word2vec将其转化为低维的向量表示. 为了识别特定的漏洞类型，μVulDeePecker[50]扩展了VulDeePecker中的代码段表示，通过加入控制依赖相关的语句，使其包含更多的“全局”语义信息. 为了检测更多类型的漏洞，SySeVR方法[49]对VulDeePecker方法进行了改进，除库/API函数调用外，还根据语法特征提取了数组使用、指针使用、算

术表达式相关的漏洞候选关键点作为切片准则，使用程序切片技术从AST和PDG中提取与上述4种类型的漏洞候选关键点相关的语法和语义特征.

基于中间代码Token序列的方法：部分研究者聚焦于基于中间代码的Token序列的代码表示方法. 在漏洞检测领域最常用的中间代码是LLVM IR(Low-Level Virtual Machine Intermediate Representation)，其采用静态单赋值(Static Single Assignment，SSA)形式，可以确保每个变量均已定义并使用，并且只被赋值一次，因此基于中间代码的程序表示能更准确地编码与控制流和变量定义-使用(def-use)关系相关的代码语义特征[129]，相比于基于源代码的程序表示[48-49]能够捕获更准确的语义信息. 一个比较有代表性的工作是VulDeeLocator[98]，他们将SySeVR[49]中的源代码切片替换为由dg工具[130]生成的LLVM中间代码切片，以表示更多的代码语义.

基于汇编代码Token序列的方法：部分研究者基于汇编语言的Token序列对代码进行表示. 例如，Li等人[121]首先使用和SySeVR相同的方式提取源代码切片，在将源代码编译成汇编代码后，通过使用对齐算法找到与源代码切片中的语句对应的汇编代码来生成汇编代码切片. Tian等人[56]则使用符号执行和静态分析技术直接解析二进制程序中的控制流和数据流来生成汇编代码切片.

基于序列的程序表示方法能够体现程序的自然语序信息和词法统计信息，这些信息对于代码生成、代码补全、代码摘要、代码搜索、漏洞检测和缺陷修复等任务很有用. 但是，基于序列的程序中间表示对代码进行了“扁平化”处理，使得程序中的结构信息不能被有效地利用，因此还需要研究基于语法树和基于图的程序表示方法，以充分利用程序的结构信息建模源代码.

5.3 基于语法树的程序表示方法

基于语法树的程序表示方法一般指将代码表示为AST结构. 其中，部分研究将AST展平为序列的形式，而另一部分研究则直接基于AST的原始树结构提取漏洞语义.

将语法树结构转化为序列的优势在于可以直接使用基于序列的深度神经网络提取漏洞特征，其关键在于如何将从源代码中提取的AST转化为序列表示. Alon等人[129]通过连接根节点到叶子节点来形成路径，并为频繁路径赋予更大权重，以生成向量表示. 由于该方法能有效度量程序之间的语法结构的相似性，因此已成功应用在代码属性预测或者克隆代码检测任务上. 但是由于频繁出现的路径可能不是程序中最容易出现漏洞的路径，而有漏洞的路径也不一定是出现频率最高的路径，因此该方法并不适合漏洞检测任务.

在漏洞检测任务中，主要包括两类将AST转化为序列的方法，一类是将AST转化为AST路径的Token序列. 例如，Li等人[122]使用贪婪算法提取出尽可能覆盖AST中包含节点数量最少的长路径，并通过改变算法起点来提取可覆盖所有节点的多条长路径(这些长路径之间的节点可以互相重叠). 在获取长路径的序列集合后，通过word2vec将路径上下文中的Token转换为数值向量. Tanwar等人[123]则通过连接任意两个AST叶子节点来形成路径，用首尾叶子节点、路径节点和根节点一起构成路径的上下文，从AST中提取所有这样的路径来生成一个路径序列，再通过一个预先建立的词汇表将路径序列中的Token转换为数值向量. 另一类是使用深度优先遍历将AST转化为AST节点的Token序列[45,53]，然后使用word2vec等技术将序列中的Token映射为数值向量.

相比于将AST转化为序列的方法，直接对语法树结构进行建模的方法能够更充分地表示代码中的语法结构信息，而被广泛应用于克隆代码检测、代码分类、代码补全、代码生成、代码摘要以及代码属性预测等代码语义相关的任务中. 但是，有实证研究结果表明[47,59-60,131]，由于AST结构较为冗杂，和漏洞无关的节点比其他中间表示多得多，因此仅将AST作为代码的中间表示应用于漏洞检测任务并不能取得最佳的检测性能. 一些最新的研究是在AST的基础上，融入其他的代码中间表示来整合不同抽象层次的信息. 例如，将语句块的AST语法结构信息添加到CFG中[132]，或者在原始AST的基础上添加控制流信息生成一种新的代码表示CFAST(Control Flow Abstract Syntax Tree)[124].

5.4 基于图的程序表示方法

基于图的程序表示方法主要包括两种方式，将图结构转化为序列或者直接对图结构建模.

将图转化为序列的程序表示方法的重点在于如何遍历基于图的中间表示来形成序列. Duan等人[61]提出了一种基于CPG的漏洞检测方法VulSniper，在生成程序的CPG后，根据作者设计的编码规则对CPG中的节点进行编码生成一个三维的特征张量，该张量的前两维类似于邻接矩阵用于记录节点的位

置，第三维则描述了节点之间的关系类型，根据节点在AST中的位置和嵌套关系来确定节点在序列中的顺序，从而获得节点的特征向量序列．在VulSniper的基础上，作者还进一步提出使用程序切片技术来减少和敏感操作无关的语句，以减少程序表示中和漏洞无关的信息[62]．

相比于将图结构转化为序列后建模的方法，利用图神经网络直接对图结构进行建模能更好地学习基于图的代码表示中的节点之间的依赖关系以及程序的结构和语义信息，为源代码漏洞检测模型提供更准确的特征向量表示，主要包括两种思路，一种是通过集成多种代码中间表示形式来综合表征不同抽象层次的漏洞相关的语法语义信息[28,47]．另一种思路是采用程序切片技术从混合的代码表示中去除与漏洞无关的信息[60]．

第一种思路的代表性方法包括：Yamaguchi等人[28,35]提出的代码属性图(Code Property Graph，CPG)的程序表示，以综合表征程序的结构和语义信息．Zhou等人[47]在基于AST、CFG和PDG生成的CPG中加入自然语序信息，生成能综合表征程序结构和语义的复合图(Composite Graph)．Wang等人[52]通过结合从AST和PDG中提取的信息来生成一种称为程序图(Program Graph)的中间表示．Cao等人[64]提出了一种结合AST、CFG和DFG来综合表征程序的语法和语义信息的代码复合图(Code Composite Graph，CCG)，相比于前述文献中使用的CPG，CCG舍弃了代码中的控制依赖信息，而保留了数据依赖信息．Li等人[65]提出了一种包含12种连接关系的综合图(Comprehensive graph)，使其蕴含更多的程序语义信息，它几乎涵盖了joern静态分析工具所能提取的大部分边和节点．

已有研究[49,60,66]发现，程序表示中蕴含的信息并不是越多越好，大量和漏洞无关的信息反而会对模型学习漏洞模式产生负面影响．为了解决综合表征代码语义信息的图结构较为复杂且存在较多和漏洞无关信息的问题，另一种思路是采用程序切片技术从代码的图结构中间表示中去除与漏洞无关的信息[60,66,68]．例如，Cheng等人[68]提出一种结合程序切片和图神经网络的漏洞检测方法DeepWukong，基于程序切片技术从PDG上提取子图(Cheng等人称该子图为XFG)，但该方法仅以API/函数调用和操作符语句作为切片准则，因此仅能检测有限类型的漏洞．类似地，Cao等人[125]仅使用API/函数调用和指针操作作为切片准则，基于程序切片技术从结合函数调用图(Call Graph，CG)的扩展PDG上提取子图，该方法仅能检测内存相关的漏洞．为了提高对不同类型漏洞的检测能力，Zheng等人[60]提出了一种称为切片属性图(Slice Property Graph，SPG)的程序中间表示，在常用的4种切片准则基础上提出了函数参数和函数返回值两种与函数间数据传递相关的切片准则，以覆盖更多的漏洞候选关键点，然后基于这6种切片准则，使用过程间分析和程序切片技术，生成包含节点属性信息(语句内容和节点类型等)的SPG，以更准确地提取与漏洞候选关键点有数据依赖、控制依赖和函数调用依赖的图结构信息、节点属性信息和代码上下文信息，避免CPG中的大量漏洞无关语句节点对检测模型训练的不利影响．

最后从编程语言来看，目前基于学习的源代码漏洞检测方法中，面向多语言源代码的研究较少，并且这些方法主要使用基于度量和基于序列的程序表示[30,36,133-136]，因为基于这两种表示的特征提取无需借助特定语言的代码解析工具来构建复杂的代码中间表示，因此易于扩展到多种编程语言．例如，Zaharia等人基于Token序列的中间表示来提取C/C++和Java代码中的漏洞特征[133]．

由于基于语法树和图的程序表示能够提供代码中的结构信息，因此近年来这两种程序表示除了被广泛应用于C语言程序的漏洞检测外，也被应用于其他编程语言[124,137-140]．例如，Lin等人[140]针对PHP语言，使用PDG子图SDG(Sub-Dependence Graph)来表示漏洞代码中的依赖关系．但是这些表示方法需要针对不同编程语言使用不同的代码解析工具来获得语法树和图等代码中间表示．例如，解析C/C++代码可使用joern[141]或CppDepend[142]、解析Java代码可使用Javalang[143]或JavaParser[144]，解析JavaScript代码可使用Esprima[145]或Esgraph[146]，解析PHP代码可使用PHP-Parser[147]或PHP-Joern[148]等，这些工具使研究人员对源代码进行词法和语法分析的成本降低，进一步促进了基于语法树和图的程序表示的应用．随着适合多语言的代码解析工具(例如Tree-sitter[149])的出现，目前已有研究人员开始使用基于语法树和图的程序表示进行多语言的源代码漏洞检测[52,150]．

综上，程序的中间表示为面向漏洞检测任务的程序表示学习奠定了基础，但程序表示方法对不同粒度和抽象级别的语义表征能力仍有待提升．几种典型的程序表示方法的优势和劣势对比如表4所示．

表4 程序表示方法的优劣

分类	优势	劣势
基于度量的程序表示方法	绝大部分的软件度量简单且易于提取，适用于表征代码的统计度量信息	难以深度刻画代码中的漏洞特征
基于序列的程序表示方法	能够体现程序的自然语序信息和词法统计信息	将程序表示为线性序列使得程序中的结构信息未被充分利用
基于语法树的程序表示方法	能够体现代码语法结构信息	处理规模较大的程序时，语法树中包含的节点数远大于语句数，且树的层次结构特性需要递归处理，导致后续特征提取模型的时间复杂度较高
基于图的程序表示方法	能较好地保留代码的逻辑结构和依赖关系等复杂语义信息	处理规模较大的程序时，会生成边较为复杂的多重图，加大了后续模型提取特征的难度

其中，基于度量的程序表示方法适用于表征代码的统计度量信息，但是难以刻画漏洞代码的深层语义信息，基于序列的程序表示方法适合表示程序的自然语序和词法信息，但将程序表示为线性序列会丢失程序中的结构信息．基于AST的程序表示方法适合表示程序独特的语法结构信息，但在处理规模较大的程序时，由于AST中的节点数量远大于代码中的Token数量，使得模型难以学习语法树结构中的长距离依赖信息．基于图的表示方法能较好地保留代码的逻辑结构和依赖关系等复杂语义信息，但当程序规模较大时，该方法生成的图结构（例如CPG）较为复杂，使得模型的训练效率较低．一种有效的代码中间表示方法既要保留丰富和关键的漏洞语义信息，以提高其对漏洞语义信息的抽象和表征能力，又要最大限度地降低漏洞无关信息对表达漏洞特征的负面影响，并降低代码中间表示的复杂度．因此，程序表示方法如何全面、准确和有效地刻画漏洞代码在不同粒度和抽象级别上的语法和语义，仍有待深入研究．

6 源代码的粗粒度漏洞检测方法

程序分析是漏洞检测的基础技术，具体包括静态分析、动态分析和混合分析方法．其中，静态分析技术主要通过分析代码来检测漏洞，无需搭建系统运行环境对程序进行测试，但误报率较高[48]．动态分析通过运行程序来检测漏洞，虽然准确率较高，但很大程度上取决于测试用例的质量，且代码覆盖率较低，存在漏报的可能，而且有时研究人员只能从代码仓库中挖掘得到部分有变更的代码，很难得到完整的项目源代码，导致代码无法正常编译运行，只能依靠静态分析手段进行检测．动静结合的分析方法虽然可以在某种程度上缓解误报和漏报，但仍存在时间复杂度较高、检测效率较低的问题[151]．总之，传统的漏洞检测技术在效率和有效性方面仍有待进一步提升．近年来，机器学习和深度学习为漏洞检测任务提供了更有效的、自动化的解决方案．与传统技术相比，基于学习的方法可以学习潜在的、抽象的漏洞代码模式，并且能够显著提高模型的泛化能力[152]．因此，本文主要围绕基于学习的漏洞检测方法展开讨论和分析．

6.1 常用的漏洞检测工具和评价指标

基于规则的检测方法[13-27]检测漏洞的基本原理是通过提供一套语法，支持专业人员基于该语法用规则描述一个有缺陷的数据流或控制流，然后使用规则匹配方式，报告符合规则描述条件的漏洞代码．现有的商业或开源检测工具虽然其检测原理各不相同，但都利用了基于规则的检测方法．表5中列出了一些经典的漏洞检测工具．

表5 常用的漏洞检测工具

类型	代表性工具	检测语言
商业工具	Checkmarx[21]	20种语言
	Coverity[22]	19种语言
	Fortify[23]	25种语言
	CodeSonar[24]	C/C++/Java
	Klocwork[25]	C/C++/Java
	CodeSecure[26]	6种语言
	Helix QAC[27]	C/C++
开源工具	Flawfinder[13]	C/C++
	Cppcheck[15]	C/C++
	FindBugs[16]	Java
	ErrorProne[17]	Java
	Clang Static Analyzer[20]	C/C++
	RATS[14]	C/C++
	AUSERA[18-19]	Java/Kotlin/C/C++

漏洞检测任务为分类任务，因此可使用分类任务中的指标，表6列出了一些目前漏洞检测领域中较为常用的评估指标．

表6中大部分指标可通过TP（True Positive）、

表6 漏洞检测任务中常用的评估指标

评估指标	描述	计算公式
真阳性TP	模型检测为漏洞，实际也为漏洞的样本数量	——
假阳性FP	模型检测为漏洞，实际为非漏洞的样本数量	——
真阴性TN	模型检测为非漏洞，实际也为非漏洞的样本数量	——
假阴性FN	模型检测为非漏洞，实际为漏洞的样本数量	——
准确率Accuracy	模型检测正确的样本数量占测试数据总样本数量的比率	$Accuracy=\frac{TP+TN}{TP+FP+TN+FN}$
精确率Precision	模型检测为漏洞的样本中，真实漏洞样本所占的比例	$Precision=\frac{TP}{TP+FP}$
召回率Recall	测试数据的所有漏洞样本中，被模型预测为漏洞的样本所占的比例	$Recall=\frac{TP}{TP+FN}$
F1-Score	能够综合评估模型检测的精确率和召回率的指标	$F1=\frac{2\times Precision\times Recall}{Precision+Recall}$
漏报率FNR	模型检测为非漏洞，实际为漏洞的样本所占测试集中所有漏洞样本的比率	$FNR=\frac{FN}{TP+FN}$
误报率FPR	模型检测为漏洞，实际为非漏洞的样本所占测试集中所有非漏洞样本的比率	$FPR=\frac{FP}{TN+FP}$
ROC曲线	在横坐标为FPR，纵坐标为TPR(TPR等价于Recall)平面上所画的曲线	——
AUC	ROC曲线下与坐标轴围成的面积	——
IoU	设V为模型预测的漏洞语句位置集合，U为真实漏洞语句位置集合	$IoU=\frac{\lvert V\cap U\rvert}{\lvert V\cup U\rvert}$

TN(True Negative)、FP(False Positive)、FN(False Negative)计算得出，而较为复杂的ROC曲线(Receiver Operating Characteristic curve)、AUC(Area Under Curve)和IoU(Intersection over Union)指标则需要单独计算. 其中，ROC曲线需要通过遍历模型分类器的阈值来绘制. 具体来说，模型基于不同的分类器阈值预测的漏洞与非漏洞样本是不同的，因此可获得不断变化的FPR-TPR样本点，从而形成曲线，同时ROC曲线越陡，说明模型的性能越好. AUC表示ROC曲线和坐标轴围成的面积，其值一般介于0.5到1之间，且越接近于1，说明模型性能越好. IoU是一种被广泛应用于目标检测任务中的评估指标，被Li等人[98]引入细粒度漏洞检测任务中，通过计算真实漏洞语句位置集合和预测的漏洞语句位置集合之间的交并比来评估模型细粒度漏洞检测的性能. IoU的值介于0到1之间，且越接近1，说明模型性能越好.

6.2 基于传统机器学习的漏洞检测方法

基于机器学习的漏洞检测方法主要依靠专家手动提取特征，然后使用机器学习分类模型进行漏洞检测. 这些分类模型主要包括贝叶斯网络(Bayesian Network，BN)、朴素贝叶斯(Naive Bayes，NB)、决策树(Decision Tree，DT)、随机森林(Random Forest，RF)、逻辑回归(Logistic Regression)、K近邻算法(K-Nearest Neighbor，KNN)和支持向量机(Support Vector Machine，SVM)等. 根据其特征提取方法，又可将其分为三类：基于软件度量的方法、基于文本挖掘的方法、基于模式匹配的方法. 这些方法如表7所示.

基于软件度量的方法侧重于从代码中提取软件度量来构建一组特征，包括从经典软件度量转化而来的统计特征[29-33,36-38]，以及从改进的软件度量转化而来能进一步描述漏洞代码的特征[32]，然后将这些特征送入机器学习分类器进行漏洞检测.

基于文本挖掘的方法则是利用词法分析技术，从源代码的Token信息中提取统计特征，采用词袋技术生成源代码的数值向量[34,39]，再将其送入机器学习分类器中进行漏洞检测.

基于模式匹配的方法主要利用程序分析技术来提取漏洞特征和漏洞模式，并通过模式匹配技术来检测漏洞.

很多研究从不同形式的代码表示中提取结构化的特征作为代码模式(Code Pattern)送入机器学习分类器. 例如，Yamaguchi等人[40]通过代码分析技术提取源代码中的API/库函数调用序列，通过词袋和主成分分析技术将其转化为代码模式，然后基于距离函数进行模式匹配. 为了获取代码中的语法信息，该作者进一步提取了代码函数对应的AST，通过词袋技术和潜在语义分析(Latent Semantic Analysis，LSA)技术生成API模式[44]，通过对API

表7 基于传统机器学习的漏洞检测方法

特征提取	分类模型	优 势	劣 势
软件度量	BN[36] LR[29] DT/RF/LR/BN/NB[37] DT/RF/LR/BN[38] DT/RF/BN/NB[30] RF/LR/NB/SVM[31] RF/LR/NB/SVM[33] SVM[32]	软件度量指标在软件工程项目中很容易获得，且已形成了完整成熟的体系，基于软件度量构建漏洞检测模型不需要额外的专业知识	一般的软件度量指标并非专门为漏洞检测设计，无法直接反映漏洞特征，因此基于软件度量的漏洞检测模型准确率仍有待提高
文本挖掘	RF[39] DT/RF/NB/SVM/KNN[34]	易于获得源代码的文本特征	文本特征难以反映源代码的漏洞特征
模式匹配	PCA＋距离函数[40] NB/DT/MLP[41] NB/DT/MLP[42] k-means[43] 图遍历[28] CLC＋图遍历[35] LSA＋距离函数[44]	模式匹配方法利用特征工程或专家定义的漏洞特征检测漏洞，具有更好的可解释性	模式匹配方法不易提取代码中的深层语义特征，仅适用于检测与已知漏洞模式匹配的漏洞

使用模式进行匹配实现漏洞检测．除了基于序列和语法树的代码表示外，也有研究从基于图的代码表示中提取漏洞特征．例如，Shar等人[41,43]通过数据流分析来提取诸如输入输出数据的语句数量、类型等特征形成代码模式，然后利用机器学习分类器检测漏洞．除了使用单一图表示提取漏洞模式外，Yamaguchi等人[28,35]提出了通过综合多种图表示的CPG来提取函数调用信息（包括函数调用顺序、函数参数及返回值等数据流信息等），再通过完全链接聚类技术（Complete Linkage Clustering，CLC）对所有信息进行聚类分析以获得代码模式，最后利用代码模式在CPG上进行图遍历查询来检测漏洞．与基于软件度量和基于文本挖掘的代码特征相比，虽然从静态分析工具的代码解析结果提取的漏洞模式包含了更多的信息，但仍然无法提取代码的深层语义特征．

综上，基于机器学习的漏洞检测方法依靠专家人工定义和提取特征，特征工程的成本高，易出错，还有可能因缺少对深层语义特征的提取而降低检测性能．此外，数据噪声、模型过拟合/欠拟合等原因常使学习过程偏离预期目标，降低了基于机器学习的漏洞检测方法的性能．

6.3 基于深度学习的漏洞检测方法

相比于机器学习的漏洞检测方法，基于深度学习的漏洞检测方法能通过深度神经网络自动学习代码中隐含的漏洞语义特征，从而摆脱专家人工定义漏洞特征的桎梏，因此近年来受到了广泛关注[152]．现有的基于深度学习的漏洞检测研究大多集中在文件[51,53-54,64,69,153]、函数[45-46,52,59,61,63,65,67,105,154]或代码段[48-50,55-57,60,66,68,70-71]等粗粒度的级别上，即预测文件、函数或代码片段包含漏洞的可能性，仅有少量语句级细粒度漏洞检测的研究[84-87,98,125,153,155-158]，将在7.2节介绍．本节介绍粗粒度的漏洞检测方法，表8列出了基于深度学习的粗粒度漏洞检测方法及其使用的特征提取模型．

基于深度学习的漏洞检测方法大多遵循如图8所示的检测框架．首先将代码解析成合适的代码中间表示形式，并将其转换为适合深度学习模型的语义向量表示，然后构建合适的深度学习模型对将代码表示为包含深层漏洞语义的向量表示，最后通过分类器模型输出漏洞检测结果．

针对基于序列的程序中间表示（包括将语法树/图转化为序列），通常使用卷积神经网络（Convolutional Neural Networks，CNN）或者循环神经网络（Recurrent Neural Network，RNN）作为表示模型来提取漏洞特征．CNN的特点是可以捕获局部语义特征．例如，Russell等[51]人使用CNN从源代码函数的序列表示中学习并提取特征．

相对于CNN，RNN及其变体（GRU，LSTM等）在处理序列数据时具有较大的优势，擅长捕获代码的顺序语义信息和代码上下文中的长依赖信息．例如，Li等人提出的三种使用切片代码段作为中间表示的方法VulDeePecker[48]、μVulDeePecker[50]、

表8　基于深度学习的漏洞检测方法

模型	特征提取	优势	劣势
针对序列程序表示的深度学习模型	DBN[53] CNN[46,61] CNN/LSTM/CNN+LSTM[54] CNN/LSTM[51] BiLSTM[45,62] BiGRU[56,70-71] BiGRU/BiLSTM[48-50] BiGRU+TextCNN[55] CNN+BiLSTM[64] Transformer[58] BERT/CodeBERT[57]	能有效学习代码的顺序语义特征，对Token或语句间的依赖关系进行建模，基于Transformer的模型还可以有效提取长距离依赖关系	无法建模源代码中的结构语义和语法信息
针对语法树/图程序表示的深度学习模型	GCN/GAT[69] GGNN[154] GAT[69] BGNN[59] R-GGNN[52] GINN[63] FA-GCN[67] K-GNN[68] R-GCN[60,66]	可以直接建模代码中的结构语义和语法信息，能更准确地学习代码中的深层语义	难以学习代码的全局和顺序语义信息，模型的学习效率较低

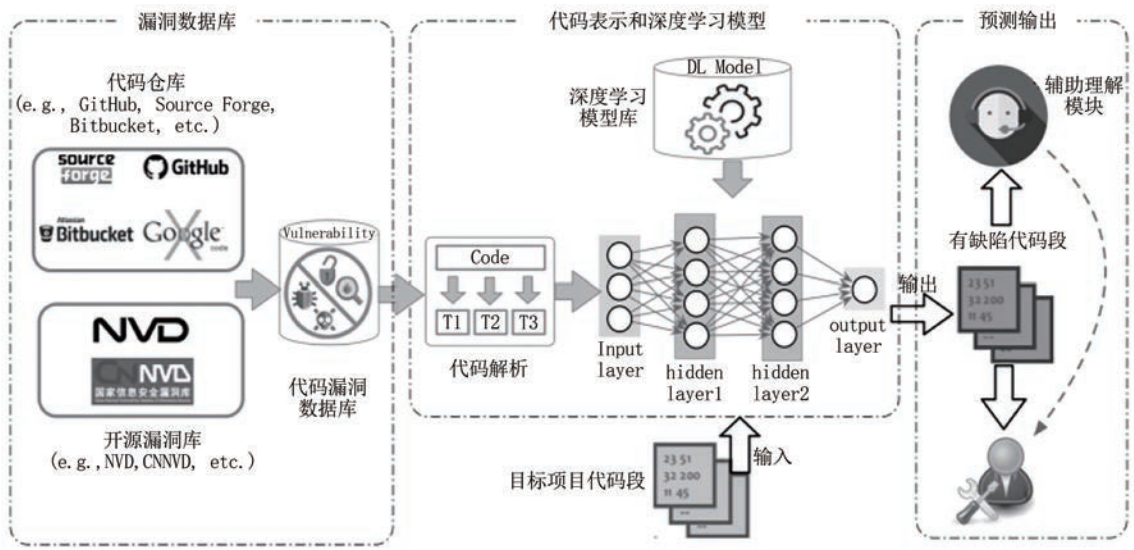

图8　基于深度学习的源代码漏洞检测框架

SySeVR[49]均使用了RNN及其变体作为漏洞特征提取的网络结构，结果表明BiGRU(Bidirectional Gated Recurrent Unit)和BiLSTM(Bi-directional Long Short-Term Memory)的效果更好，这是因为和单向GRU/LSTM相比，BiGRU和BiLSTM的结构由前向和后向网络组合而成，其每个时刻的输出由前向和后向两个RNN的输出共同决定，能够有效地学习代码的上下文(前向和后向)依赖信息[62]. 因此，许多研究都使用BiGRU[56,70-71]或BiLSTM[45,62]对代码进行表示学习.

为了结合CNN在提取局部语义特征和RNN在捕获时序依赖关系方面的优势，很多研究使用CNN与RNN的组合模型来实现漏洞模式提取，取得了比单独使用RNN和CNN更好的漏洞检测效果. 例如，Wu等人[54]对CNN、LSTM以及二者结合的三种漏洞特征提取方法进行了对比. 结果表明，先利

用CNN提取代码的局部语义特征，然后再通过LSTM学习代码上下文中的长依赖信息，相比于单独使用CNN和LSTM能够提取更多的漏洞语义信息. 为了更深入地刻画漏洞语义，Cao等人[64]还在使用CNN+BiLSTM组合模型中引入了离散傅里叶变换和注意力机制，通过傅里叶变换将源代码转换到频域，并通过注意力机制关注到代码中对漏洞检测更为重要的元素.

近年来，Vaswani等人[159]提出了一种完全由注意力机制和前馈神经网络构成的模型Transformer. 和传统的卷积神经网络以及循环神经网络相比，Transformer提出的多头自注意力机制能够直接计算任意两位置Token或语句之间的依赖关系，更好地捕获全局以及长距离的依赖关系. 此外，多头自注意力机制也可以更好地执行并行计算，提高模型效率，且结构也更为灵活，可以根据不同任务需要进行调整，也能够处理不定长的序列数据，从而更有效地保留输入特征信息. 目前，Transformer模型在自然语言处理领域取得了惊人的效果，鉴于其强大的序列特征提取能力，一些研究人员已经探索并利用Transformer模型构建漏洞检测模型. Thapa等人[57]在C/C++源代码漏洞数据集上对基于RNN及其变体和Transformer的模型进行的实证分析结果表明，BERT、CodeBERT等基于Transformer的模型性能优于BiLSTM、BiGRU等基于RNN及其变体的模型. 进一步地，Mamede等人[58]基于Transformer模型开发了一款IDE漏洞检测插件VDet. 在对JavaBERT[160]进行微调和评估后，将模型加载到服务器中，并与扩展程序建立通信，以实现对Java代码的漏洞检测.

最近，用于代码生成和分析的大语言模型凭借其优秀的性能成为了代码分析相关领域的研究热点，尤其是GPT（Generative Pre-trained Transformer）系列模型[161]，已被证明具有很强的理解和生成代码的能力. 例如，在Bubeck等人[162]的技术报告中，比较了GPT-4、GPT-3.5、GPT-3等模型在各种编码挑战中的性能，从结果来看，它们均取得了惊人的效果，GPT-4的表现甚至在一定程度上可以和人类媲美. 因此，研究人员开始将大语言模型用于漏洞检测，以探究其能否解决漏洞检测问题. Cheshkov等人[163]评估了ChatGPT和GPT-3模型检测真实项目中Java代码漏洞的性能，令人惊讶的是，无论是ChatGPT还是GPT-3模型，它们目前能够有效检测代码漏洞的能力有限. 但也有研究指出[164]，由于ChatGPT可以直接理解自然语言，因此其可以在检测过程中为安全专家提供更丰富且直观的漏洞信息，而非简单的分类结果，从而提高专家后续修复漏洞的效率. 然而，该研究还指出，ChatGPT还不能完全理解漏洞和非漏洞之间的细微差别，提供的信息并不总是准确或有用的，而且其高昂的部署及训练成本也对其使用造成了障碍. 总之，目前若要在漏洞检测任务中大规模地应用ChatGPT等模型，还需要进一步的完善和研究.

针对基于语法树的程序中间表示，虽然可以采用TBCNN[165]、Tree-LSTM[166]、AST-based LSTM[167]、ASTNN[168]等基于语法树的神经网络模型对语法树直接建模以捕获代码的结构特征，但是这些方法并不是专门为捕获漏洞特征而设计的，并且已有实证研究表明[47,59-60,131]，在漏洞检测任务上，仅使用AST作为中间表示并搭配相应的表示学习方法并不能取得满意的结果，通常是将AST和其他的基于图的中间表示结合生成复合图，并使用图神经网络进行表示学习.

针对基于图的程序中间表示，通常采用图神经网络（Graph Neural Network，GNN）直接对其进行建模，利用图级分类结果实现漏洞检测. GNN作为一种可直接学习图数据结构特征的深度学习模型，在与基于图的程序表示进行搭配时具有明显的优势，相比于其他先将程序“扁平化”处理再学习漏洞特征的检测模型，基于GNN的漏洞检测方法直接在复杂的代码结构上学习代码表示，能更好地捕获代码的语法和语义等信息，更适合复杂程序的漏洞语义理解，从而提高漏洞检测的性能. 图神经网络的核心思想是通过聚合来自局部邻域节点的信息学习中心节点的表示，根据采用的聚合技术的不同，研究人员开发出了不同的GNN模型.

最早被用于漏洞检测领域的是门控图神经网络（Gated Graph Neural Network，GGNN），Zhou等人[47]利用GGNN从代码复合图中提取漏洞特征，GGNN在图神经网络的基础上加入了GRU单元，使其能够同时学习图中的结构信息和顺序信息. 由于代码复合图综合了代码的多种中间表示形式，因此通常为异构图，为了能够学习其中不同类型的节点与边，Wang等人[52]使用了一种改进的R-GGNN（Relational Gated Graph Neural Network）模型来进行表示学习，和GGNN相比，R-GGNN对不同类型的节点和边赋予了不同的可学习权重，因此对异构图更有效. 图卷积神经网络（Graph Convolutional

Network，GCN）是GNN模型中最经典的一种，常被用作漏洞检测的基线模型[68-69]. 在一部分文献[60,66]中，一种改进的GCN模型R-GCN（Relational Graph Convolutional Network）被用作特征提取的主体模型. 和R-GGNN类似，R-GCN也是针对异构图进行表示学习的图神经网络. Li等人[67]则使用了另一种基于GCN的改进模型FA-GCN（Feature Attention Graph Convolutional Network）作为漏洞特征提取的主体模型，FA-GCN可以很好地处理具有异构特征的图（即并非所有语句均具有相同属性的图），并去除PDG中的潜在噪声[169]. 为了辅助模型着重学习程序中的与漏洞相关的信息，GAT（Graph Attention Networks）则是在图中引入了注意力机制，通过对节点之间的边赋予一定的权重，实现在程序表示学习中关注与漏洞相关的语法和语义信息. Li等人[65]和Ghaffarian等人[69]均使用了GAT作为表示模型，并在对比实验中获得了最优的结果. 原始GNN中使用消息传递技术来提高学习能力，但一层GNN仅能传递单跳邻居节点的信息，尽管堆叠多个GNN原则上可以扩大消息传递范围，但又存在计算成本过高和过度平滑的问题，这严重阻碍了GNN学习全局信息的能力[121]，因此，Wang等人[63]提出了图间隔网络（Graph Interval Neural Network，GINN）来实现对扩展控制流图的层次化表示学习，GINN首先根据控制流程图的结构提取图中的间隔区间，每次只计算区间内节点之间的信息传递，然后将区间聚合成“节点”再重新提取新的区间，直至图中所有节点均属于同一区间为止，此时可获得图的全局特征，随后再将特征分解，逐步将图恢复成初始结构，从而得到每一节点的隐藏特征，GINN可以避免因距离过远导致节点信息被稀释的问题，从而增强了对大图泛化的能力.

以上的代码表示模型侧重于学习程序的全局语义特征来实现漏洞检测，为了有效提取代码的局部和全局语法语义特征，一些研究成果使用了分段代码表示学习框架，即先在语句内提取局部的语法语义特征，然后再提取语句之间的全局语法语义特征. 由于同时考虑了代码局部和全局的语法语义信息，因此分段代码表示学习方法对程序漏洞特征的提取更为充分. 例如，针对基于序列的程序中间表示，An等人[71]采用两个BiGRU模型分别学习语句内的局部语义信息和语句间的全局语义信息，Yan等人[55]则分别使用BiGRU和textCNN来提取代码的局部和全局语义特征. 而针对基于图的程序中间表示，则是通过学习PDG[66]或SPG[60]的节点向量表示来捕获代码语句内的局部语义信息，通过学习PDG或SPG的节点（即语句）之间的依赖关系来捕获代码的全局语义信息. 其中，Li等人[67]同时使用Tree-LSTM和GRU学习PDG中语句节点的局部语法信息、语句内容、变量类型和上下文信息，并使用FA-GCN来提取全局语义信息，但该方法未考虑不同粒度的代码元素对理解漏洞语义的重要程度. 考虑到PDG中存在大量漏洞无关的节点，并且不同的语句节点以及语句中的Token对漏洞检测具有不同的重要程度，Zheng等人[60]在利用切片技术从CPG生成SPG的基础上，使用Token级的自注意力机制来学习漏洞相关的语句节点内各个Token之间的依赖信息，并使用带有节点级和子图级注意力机制的RGCN来从SPG中提取漏洞相关的全局语义信息.

现有的基于序列的程序表示模型虽然能够学习代码的顺序语义特征，但无法捕获代码的结构语义信息. 基于语法树的程序表示模型虽然能有效学习代码的语法结构特征，但无法提取语句之间结构语义特征. 而基于图的程序表示模型虽然能够有效学习代码的数据依赖和控制依赖等结构语义特征，但无法提取代码的顺序语义特征. 为此，Jiang等人[170]提出一种层次化语义感知的代码表示学习框架，该框架首先设计了一种新的代码表示形式即语义图，以同时表示语句内的语法信息和语句间的依赖信息. 在基于程序分析和程序切片技术生成代码的语义图的基础上，先利用Tree-LSTM从语义图的语句节点中提取和漏洞相关的局部语法语义特征，再利用其提出的一种新的代码表示学习网络即Graph-LSTM，按照语义图中节点的拓扑顺序提取代码中和漏洞相关的全局结构语义和顺序语义特征. 该模型的优势在于能够结合Tree-LSTM和Graph-LSTM的优势，更有效和高效地学习代码中的漏洞模式.

在对代码进行表示学习得到代码的特征向量之后，将其送入分类器模型即可获得代码的漏洞检测结果. 常用的分类器模型主要分成四类：一是仅使用全连接网络（Fully Connected Network，FCN）作为分类器网络[45-46,54,58-59,65]，二是使用FCN搭配Softmax函数作为分类器网络[48-50,52,55-56,60]，三是使用支持向量机、随机森林、逻辑回归、朴素贝叶斯等机器学习模型作为分类器[51,53]，四是单独设计分类器[47]. 其中，FCN搭配Softmax函数组成的分类器

是目前基于深度学习的漏洞检测方法最为常用的分类器.

综上,现有的面向漏洞检测的程序表示模型研究主要围绕如何提升程序表示模型在不同粒度和抽象级别上的语义建模能力和对代码的漏洞语义理解能力展开,并且主要集中于尝试使用不同的表示模型或综合利用现有模型的优势以更全面、准确地刻画和理解代码的关键漏洞语义,而对程序表示模型本身的复杂程度和学习效率关注较少. 例如,基于图的程序表示模型使用图嵌入技术直接在图上学习代码表示,能更好地捕获代码的结构语义信息,但当程序规模较大时,使用基于GNN的程序表示学习模型的学习效率偏低. 从语义学习能力的角度,单层GNN节点的感受野(receptive-field)仅限于单跳邻居,尽管堆叠多个GNN层原则上可以扩大感受野从而学习非相邻节点间的信息,但也会产生过平滑的问题,而难以准确学习漏洞代码的上下文信息[171]. 其中,过平滑问题是指使用多层GCN后,节点的区分性变得越来越差,节点的向量表示趋于一致,使得相应的学习任务变得更加困难,这个现象被称为过平滑[172]. 节点的感受野指的是节点在学习过程中能够聚合其他节点信息的范围. 因此,研究新的程序表示模型以兼顾程序表示模型的漏洞语义理解能力和模型的复杂度或学习效率,是未来值得关注和深入研究的一个问题.

7 粗粒度漏洞检测的可解释方法

基于深度学习的漏洞检测模型的黑盒性质使得用户无法理解它是如何做出预测的,并且粗粒度的漏洞检测结果难以辅助开发人员理解和分析漏洞的成因并快速修复漏洞. 因此,近年来针对粗粒度漏洞检测的可解释方法引起了研究者们的关注,这些方法在粗粒度漏洞检测结果的基础上,进一步给出代码元素对模型预测结果的贡献度或者对模型预测结果起到关键影响的代码元素(这些代码元素可以是token、语句或者程序依赖子图等),从而对检测结果进行解释. 从已有的研究来看,目前方法主要采用模型自解释方法、模型逼近方法或样本反馈方法对使用基于序列[173-175]、基于语法树[176-177]和基于图[178]程序表示的粗粒度漏洞检测方法的检测结果进行解释. 为了便于理解,本文以图9描述了三类方法对基于序列的程序表示的解释过程. 目前检索到的具有代表性的源代码漏洞检测的可解释方法如表9所示.

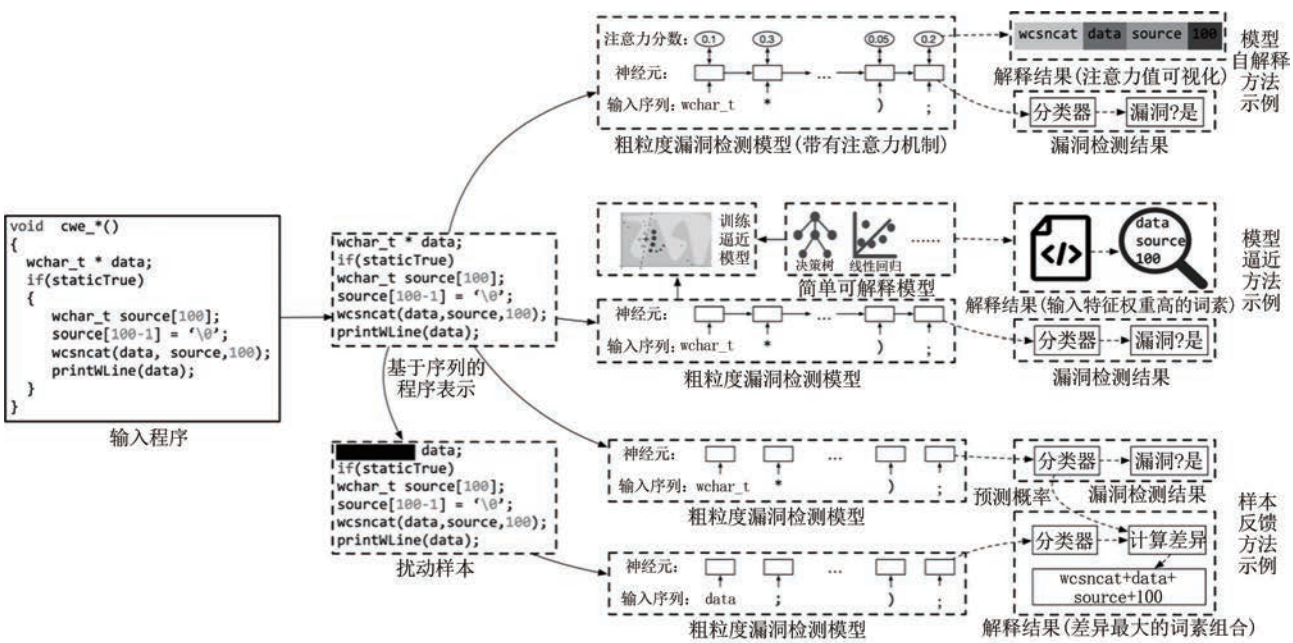

图9 粗粒度源代码漏洞检测的可解释方法示例

7.1 模型自解释方法

模型自解释方法主要从漏洞检测模型本身的角度来提供可解释性,相当于将黑盒模型的行为白盒化,通常需要研究漏洞检测模型的参数(如模型梯度)来确定输入的重要性或在检测模型中加入能起到解释作用的机制(如注意力机制).

典型工作是由Mao等人[176]、Gu等人[177]和Zou等人[173]提出的基于注意力的可解释漏洞检测方法,

表9 典型的源代码漏洞检测的可解释方法

方法类型	程序表示	解释方法	优势	劣势
模型自解释方法	基于语法树/基于序列	注意力机制[173,176-177]	检测过程自动提供了解释信息，解释成本较低	方法与待解释的模型相关，不具备普适性
模型逼近方法	基于序列	Gradient[174] IG[174] LRP[174] LIME[174] LEMNA[174] SHAP[174]	方法与待解释的模型无关，实现简单，适用性强，适配于大多数模型	代理模型只能局部逼近待解释模型，影响解释性能
样本反馈方法	基于序列	启发式搜索[175]	提取关键特征更为准确，解释能力强	解释成本较高
	基于图	GNNExplainer[178]		

这种方法在进行源代码的粗粒度漏洞检测的同时，利用模型学到的注意力权重来记录对漏洞检测起到关键作用的代码元素. Warnecke等人[174]介绍了6种解释方法，其中的三种方法是通过检测模型本身的参数来提供可解释性，包括梯度法及其改进的积分梯度法(Integrated Gradients，IG)以及逐层相关性传播法(Layer-wise Relevance Propagation，LRP). 其中，梯度法和积分梯度法是通过模型梯度的大小来确定代码元素对检测结果的贡献度. LRP将每个输入特征对预测结果的贡献定义为相关性分数，通过将预测结果(即相关性分数之和)根据模型的网络结构逐层反向传播到输入端来确定代码元素对检测结果的贡献度. 这类方法在检测过程中自动提供了解释所需要的信息，因此解释成本较低，然而，方法与待解释的漏洞检测模型相关，不具备普适性.

7.2 模型逼近方法

模型逼近方法利用了机器学习模型可解释性强的特点，通过训练更简单、更易理解的可解释模型，使其在性能上逼近基于深度学习的漏洞检测模型，为模型的检测结果提供解释信息. 例如，一种较为经典的模型逼近方法是Tang等人[179]使用的LIME方法，它通过构建局部近似的线性回归、决策树等可解释性更强的代理模型以分析单个被测样本的决策过程来提供局部的解释信息. Warnecke等人[174]介绍的另外三种方法中也采用了上述方式为模型提供可解释性，其中包括LIME以及LIME的改进方法LEMNA和SHAP. 这类方法将检测结果的解释与漏洞检测的黑盒模型分离，使方法与待解释的模型无关，具有实现简单、适用性强的优势，可以适配于大多数的漏洞检测模型，但在解释过程中代理模型只能局部逼近待解释模型，影响了解释性能.

7.3 样本反馈方法

样本反馈方法是将不同的掩码和已有的输入样本结合起来，通过干扰或掩盖样本中的局部成分信息(例如，Token、语句或者程序依赖图中的边或节点)构造扰动样本，将模型对扰动样本的检测结果作为样本反馈信息，以获取模型检测所依赖的重要信息，即通过对比样本扰动前后两次检测结果的差异来判断样本中的局部成分信息对漏洞检测的重要程度，将其用于解释模型决策产生的原因. 这类方法的典型工作是由Zou等人[175]提出的一种基于启发式搜索的模型解释方法. 该方法利用启发式搜索获取对模型分类影响较大的待测代码中的Token并按重要程度对其进行排序，通过判断删除某个特定的Token后漏洞预测概率是否显著变小，来判断该Token是否和漏洞相关，即通过确定样本中和漏洞相关的Token序列对模型的检测结果做出解释. 另外一项工作则是由Li等人[67]提出的一种细粒度可解释的漏洞检测方法IVDetect. 该方法由基于图神经网络的粗粒度漏洞检测模型FA-GCN和细粒度解释模型GNNExplainer[178]两个模块构成. 利用GNNExplainer掩盖待测程序的PDG中的边，通过对比掩盖前后的检测结果差异，来得到各个边的重要程度，将重要的边组成一个和漏洞相关的PDG子图，进而对检测结果做出解释.

相对于基于Token掩码的样本反馈法而言，GNNExplainer方法基于语句掩码进行样本反馈，虽然可以减小搜索空间，提高搜索效率，但对基于图神经网络的检测模型的细粒度解释过程仍需要较高的执行代价，并且该方法通过掩码改变了代码的图结构，进而对代码的表征发生了变化，有可能导致无法反映原始程序的语义信息.

样本反馈方法的优点在于通过预测概率的变化

获取的关键特征比通过注意力推测出的关键特征更为准确，适用面广，解释能力强，但因需要在较大的样本特征空间中进行搜索并进行多次漏洞检测和样本反馈，解释的成本较高.

综上，现有的主流的可解释方法主要侧重于从相关性的角度分析输入样本特征和模型预测结果之间的关联关系，以给出概率化或细粒度的解释信息，进而给出基于样本数据分析的因果关系解释，离辅助开发人员快速理解和修复漏洞的实际应用还有较大差距. 总之，目前对于深度学习漏洞检测模型的可解释方法的研究才刚刚起步，以下问题还有待深入研究.

(1) 漏洞检测可解释方法的效率有待提高

目前大多数基于深度学习的漏洞检测方法采用二元分类的方法，只能预测目标代码是否包含漏洞，这样的粗粒度检测结果不具备可解释性，现有的可解释方法能在一定程度上提供更细粒度的解释信息. 在目前常用的三类可解释方法中，解释能力最强的是样本反馈方法，但该类方法对粗粒度检测模型进行细粒度的解释需要额外的模型执行代价，并需要较高的搜索成本以找到使漏洞预测概率显著变小的代码元素集合. 例如，Zou等人[175]需要反复掩码漏洞样本中的Token或Token组合，不断重复漏洞检测过程才能给出Token或Token组合的重要程度. Li等人[67]使用的GNNExplainer也需要掩码PDG中的边形成新样本，然后进行重复检测，才能得到一个解释子图. Zou等人[173]的实验也证实了这一点，GNNExplainer在解释500个函数的耗时为10 595.3 s，远远超过了其他方法. 因此，这类解释方法的效率还有待提升.

(2) 目前可解释方法解释结果的可理解性有待提升

现有的解释方法聚焦于解释函数(或代码段)中的代码元素与漏洞相关的可能性. 例如，Li等人[67]可以提供一个包含漏洞语句的解释子图. 又如Zou等人[173]通过按照漏洞存在可能性对Token排序的方式来解释漏洞检测结果. 然而现有解释方法提供的解释信息难以有效辅助开发者理解漏洞的发生机理或成因，并快速定位和修复漏洞. 以图10所示漏洞实例为例，第27行为漏洞语句，但是仅根据第27行语句难以辅助开发人员解释漏洞的成因，在结合漏洞语句的相关上下文即第6、10、13、23行的语句进行分析后，才易于解释为什么第27行会被预测为漏洞语句. 因此，如何找到与漏洞语句共同作用进而引发漏洞产生的特定上下文，以辅助开发者对漏洞的产生进行有效的循证归因，是一个值得未来进一步研究的方向.

```
1  static long ec_device_ioctl_xcmd(struct cros_ec_dev *ec, void _user *arg)
2  {
3       long ret;
4       struct cros_ec_command u_cmd;
5       struct cros_ec_command *s_cmd;
6       if (copy_from_user(&u_cmd, arg, sizeof(u_cmd)))
7            return  -EFAULT;
8       if ((u_cmd.outsize > EC_MAX_MSG_BYTES) || (u_cmd.insize > EC_MAX_MSG_BYTES))
9            return  -EINVAL;
10     s_cmd = kmalloc(sizeof(*s_cmd) + max(u_cmd.outsize, u_cmd.inseze), GFP_KERNEL);
11     if (!s_cmd)
12          return -ENOMEM;
13     if (copy_from_user(s_cmd, arg, sizeof(*s_cmd) + u_cmd.outsize)) {
14        ret = -EFAULT;
15        goto exit;
16     }
17 +   if (u_cmd.outsize != s_cmd->outsize ||
18 +       u_cmd.insize != s_cmd->insize) {
19 +       ret = -EINVAL;
20 +       goto exit;
21 +   }
22     s_cmd->command += ec-> cmd_offset;
23     ret = cros_ec_cmd_xfer(ec->ec_dev, s_cmd);
24     /* only copy data to user land if data was received. */
25     if (ret < 0)
26          goto exit;
27 -   if (copy_to_user(arg, s_cmd, sizeof(*s_cmd) + u_cmd.insize))
28 +   if (copy_to_user(arg, s_cmd, sizeof(*s_cmd) + s_cmd->insize))
29          ret = -EFAULT;
30 exit:
31      kfree(s_cmd);
32      return ret;
33 }
```

图10 漏洞代码示例

8 源代码的细粒度漏洞检测方法

8.1 细粒度的源代码漏洞检测方法的问题及挑战

在漏洞检测任务中，粗粒度检测只能预测文件、函数或代码片段中包含漏洞的可能性，开发者往往还需要依靠自身的安全软件开发经验和调试经验，通过进一步的分析和调试才能定位到引发漏洞的语句，进而理解和修复漏洞. 软件的细粒度漏洞检测是指不仅要判断代码中是否包含漏洞，还要在确定代码包含漏洞的情况下，进一步定位到引发漏洞的细粒度漏洞结构(即引发漏洞的具体语句位置及相关语句之间的依赖关系). 细粒度漏洞检测不仅有助于降低人工确认和排查的工作量，还有助于增强漏洞检测结果的可解释性和易理解性，能够辅助开发人员更好地理解和修复漏洞. 因此，为了提高基于学习的漏洞检测技术的实用性，需要进行细粒度的漏洞检测. 然而，源代码漏洞的下列特性使细粒度的漏洞检测比粗粒度的漏洞检测更具挑战性.

(1) 漏洞信息分布的稀疏性和不连续性

在文件或函数中的数十、数百甚至数千条代码语句中，只有少数关键语句会引发源代码漏洞，即程序中存在大量与漏洞无关的语句，因此，漏洞信息的分布往往具有稀疏性和不连续性. 此外，程序代码具有的不同于自然语言文本的某些特殊性质[109](例如，程序上下文无关的语法结构，非上下文无关的名

称和类型约束，以及程序特有的数据流和控制流等复杂语义信息）使得程序没有明显的细粒度代码结构来描述漏洞信息的候选区域. 如图11所示，以目标检测为例，对于给定的任意一幅图像，目标检测能依据一定的算法和先验信息判断图像中是否存在指定类别的目标，如果存在，则返回目标的位置和类别的置信度. 软件细粒度漏洞检测与目标检测类似，都需要识别输入数据是否包含目标对象（图像目标或者漏洞模式），并定位出该对象的具体位置. 其区别在于：在目标检测任务中，图像目标往往是连续的且有明显区域边界的像素集合，目标信息密度较高，因此目标的关键特征易于被深度学习模型有效地提取，从而实现快速高效的检测和定位；而在细粒度漏洞检测任务中，构成漏洞模式的一组漏洞语句往往是稀疏和不连续的，不存在明显的区域边界，漏洞信息在程序中的分布密度较低，使得基于学习的漏洞检测方法难以有效地挖掘细粒度漏洞语义，并且大量无关的漏洞无关语句也增加了模型捕获细粒度漏洞模式的难度，从而导致漏洞定位的准确度较低. 因此，如何提高基于学习的细粒度漏洞检测方法对细粒度漏洞模式的识别能力是目前研究所面临的挑战性问题.

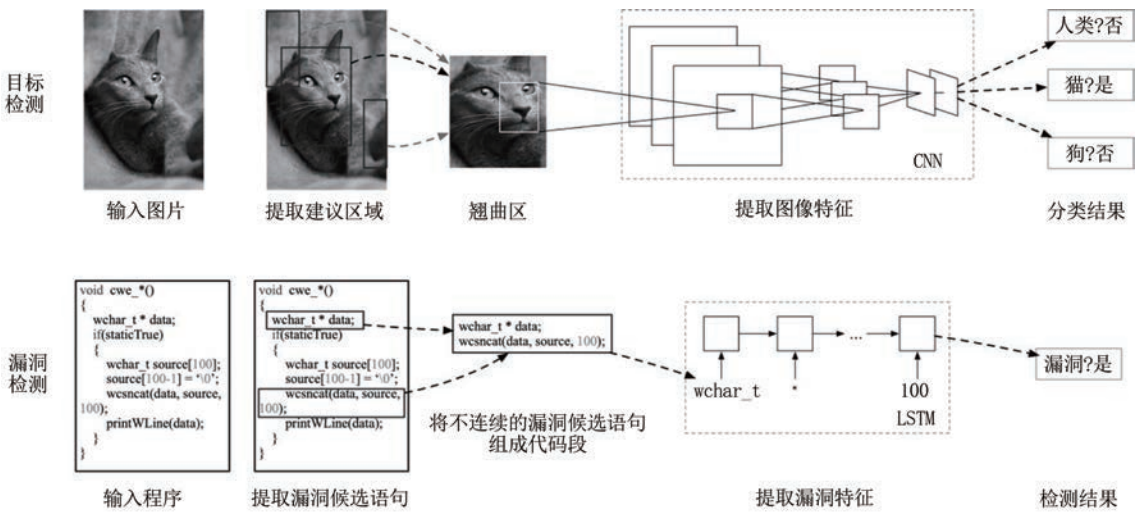

图11　漏洞检测和目标检测的过程对比

（2）漏洞语句间的上下文依赖性

一条语句是否为漏洞语句不仅取决于与其共同出现的语句，还取决于其所处的特定上下文（Context）. 源代码的上下文被定义为一组语义相关的语句，这些语句可能不相邻，但通常在控制流或数据流方面存在依赖关系[180].

仍以Li等人[67]所提供的漏洞代码为例，如图10所示，程序的第6行和第13行利用copy_from_user()通过指针参数arg从用户空间中获取数据. 在第6行将第一次从用户空间获取的数据（包括in_size和out_size的值，即旧值）存储在结构体变量u_cmd中，在第10行根据第一次获取的in_size和out_size的大小申请了一块内存缓冲区，并用指针s_cmd指向它. 在第13行第二次从用户空间获取数据（也包括in_size和out_size的值，即新值）保存到指针s_cmd指向的缓冲区中. 在竞争条件下，第二次获取的新值可能会被另一个用户线程更改，当使用不一致的in_size和out_size值时，将导致double-fetch漏洞. 若更改后的in_size值大于第一次获取的in_size值，由于第27行的copy_to_user()函数调用要将in_size大小的u_cmd中的数据复制到s_cmd指向的缓冲区中，所以前面的double-fetch漏洞会进一步引发缓冲区溢出漏洞. 为了修复此漏洞，在第17～21行增加了in_size和out_size的一致性检查和处理，同时修正了第27行的copy_to_user()函数调用语句的参数. 这一漏洞实例表明漏洞语句之间存在着共现和依赖关系，并且这种关系在解释缓冲区溢出漏洞的形成上起着关键作用. 识别出哪些函数中的哪些语句是潜在的漏洞语句以及它们如何相互作用而引发漏洞，是细粒度漏洞检测的根本任务.

如何提取全局的细粒度可解释的漏洞结构，而不是孤立地判定每条语句为漏洞语句的可能性，是细粒度漏洞检测需要深入研究的一个难点问题. 由于任务的难度和目标不同，目前适用于粗粒度的漏洞检测方法不能直接应用于细粒度的漏洞检测任务，因此，研究适合细粒度的漏洞检测方法具有重要

意义.

(3) 漏洞特征的多样性、复杂性和隐蔽性

随着现代软件的规模日益增大、功能和架构日益复杂,从最初的堆栈溢出型漏洞,到诸如跨站脚本、SQL注入等Web安全漏洞,再到心血漏洞等敏感数据泄漏漏洞,源代码漏洞的形态日趋复杂化和多样化[181],而且程序自身的语法和语义的复杂性和多样性使得漏洞在类型和模式上也呈现出复杂、多样等特征[12]. 此外,漏洞成因和形成机理的错综复杂使得各种类型的源代码漏洞特征难以从代码中显式地抽取出来,造成漏洞特征具有很强的隐蔽性,进一步增加了漏洞检测的难度.

以最近Log4j2框架中的远程代码执行高危漏洞CVE-2021-44228为例,由于该漏洞具有极强的隐蔽性,导致相关的漏洞语句未被一次性发现,使得2.0-2.14.1的所有版本均存在安全漏洞,直到2.15版本才被完全修复. 漏洞特征的多样性、复杂性和隐蔽性使得自动漏洞检测变得更加困难,也对模型的泛化能力提出了更高的要求.

8.2 细粒度的漏洞检测方法

目前,也有部分商业或开源的检测工具可以定位漏洞语句的位置. 例如,HelixQAC工具[27]不仅可以根据C/C++编码规范检查违背编码规则的代码,还可以识别潜在的安全漏洞,并根据漏洞风险的严重程度确定编码问题的优先级,使用过滤器(Filter)、抑制(Suppressions)和基线(baselines)定位到最关键的软件漏洞. 虽然依据专家预定义的规则能够定位到引发漏洞的语句位置,但是由于专家预定义规则依赖于人工设计并需要专家保证规则的正确性和完备性,因此具有较高的人工成本.

基于学习的漏洞检测方法可以摆脱对专家人工制定规则的依赖,图12描述了基于序列和基于图的两种典型的细粒度漏洞检测方法的检测过程,首先将源代码转化为程序表示,然后用代码表示模型学习程序表示中语句的特征向量,最后用分类器对代码语句进行分类,从而确定漏洞语句的位置. 目前具有代表性的细粒度漏洞检测方法及其使用的代码表示模型如表10所示.

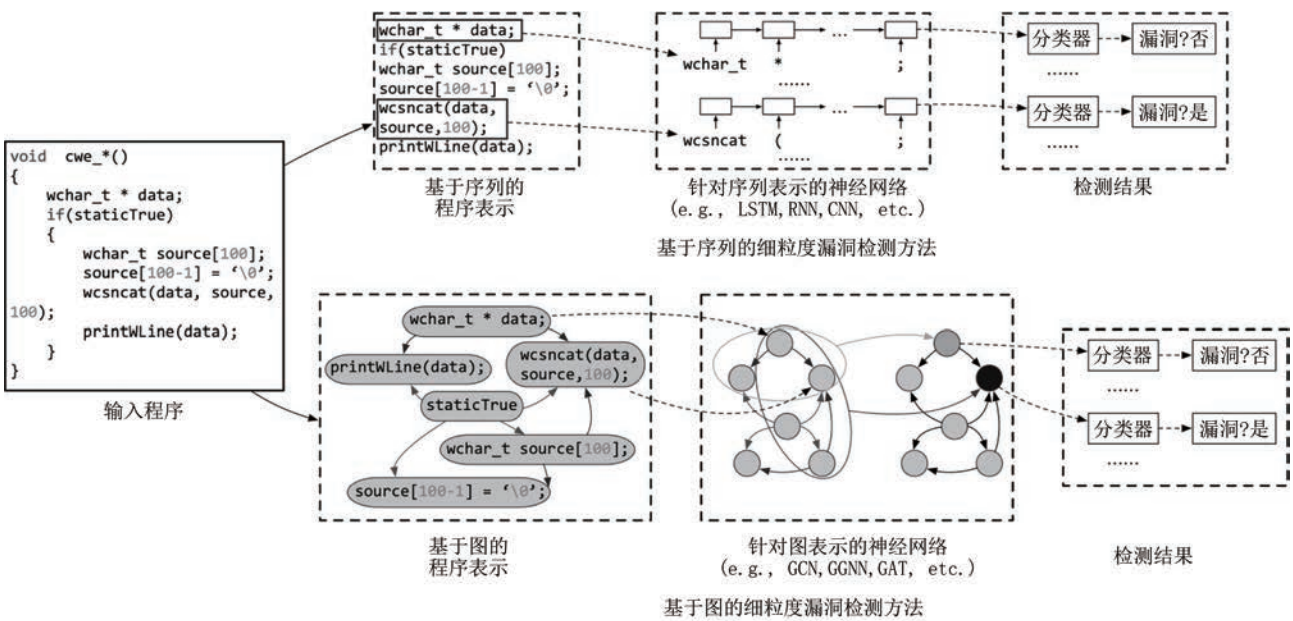

图12 典型的基于学习的源代码细粒度漏洞检测方法

其中,Choi等人[155]将缓冲区的访问操作语句作为查询,将代码段中的其他语句作为待查询的内容,从而将缓冲区溢出漏洞的检测问题形式化为问答(Query Answering,QA)任务,提出一种基于记忆网络(Memory Networks)的细粒度漏洞检测方法,利用对话系统中常用的记忆网络来捕获代码中的长距离顺序依赖关系,能够克服LSTM或CNN网络因输入序列过长而可能导致的遗忘问题,但该方法仅能识别不涉及跨函数和跨文件调用且上下文中不带有控制结构的缓冲区溢出漏洞. Sestili等人[156]对该方法进行了改进,使其能够检测上下文带有控制结构的缓冲区溢出漏洞,但该方法仍仅能识别缓冲区溢出这种特定类型的漏洞.

Tian等人[157]则将图像目标检测领域的边框回

表10 细粒度的漏洞检测方法

程序表示	代码表示模型	优势	劣势
基于序列的程序表示	记忆网络[155-156] BiLSTM/BiGRU[86,98] LSTM[153] Transformer[85] 边框回归[157]	易于提取语句之间的顺序关系和全局语义信息	难以对语句之间的数据依赖等结构关系进行建模，学习的语句向量表示中缺乏结构语义信息
基于图的程序表示	FS-GNN[125] GGNN+Transformer[84] CodeBERT+GAT[87] RGCN[158] Structure2vec[182]	通过图直接将语句映射到节点上，便于学习语句间的结构关系	受限于GNN网络，模型学习的语句向量表示缺乏全局和顺序语义信息

归思想引入到漏洞检测领域中，提出了一种基于边框回归的细粒度漏洞检测系统BBregLocator，采用边框回归方法训练模型，通过模型预测的边框（连续的漏洞语句块）来标记漏洞语句的位置，提高了对连续的漏洞语句块定位的准确率. 但该方法对非连续的漏洞语句的定位效果较差，已被作者列为后续改进的方向之一.

Wartschinski等人[153]提出了一种针对python语言的语句块的漏洞检测方法VUDENC. 该方法首先通过词法分析将代码拆分成Token序列，再通过滑动窗口按照一定步长将token序列分解成多个语句块. 为了获得用于语句块粒度的漏洞检测模型训练的数据，作者对每个程序包含的语句块进行了标记，若语句块中包含漏洞语句中的Token，则将其标记为“有漏洞”，否则标为“无漏洞”. 然后，使用word2vec将语句块转化为低维的向量表示，输入到LSTM中进行表示学习，最后通过由FCN组成的分类器网络得到检测结果.

Li等人[98]提出了基于LLVM中间代码表示的细粒度漏洞检测方法VulDeeLocator，在提取基于切片技术生成的LLVM中间代码切片代码段后，利用标准的BiGRU网络对切片代码段进行表示学习. 在模型训练阶段，利用multiply层去除漏洞代码段中非漏洞语句的Token向量，只保留和漏洞语句相关的Token向量，如果输入非漏洞代码段，multiply层不做任何处理，最后利用k-max和平均池化层，从上一层输出的Token向量中选择k个最大的元素，并取其平均作为模型预测该切片代码段有无漏洞的概率，用于计算交叉熵损失，并以此来指导模型参数的调整. 在预测阶段，在利用BiGRU获得每个Token的向量表示后，直接应用k-max和平均池化层从每个语句包含的Token向量中选择前k个最大的元素并取其平均作为该语句的特征值，如果特征值大于预设的阈值θ，则该语句被预测为漏洞相关的语句. 该方法能够充分利用LLVM中间代码的静态单赋值特性以捕获更多的与控制流和定值-使用关系相关的代码语义特征，在实现中间代码细粒度漏洞检测的同时，也提高了模型在粗粒度漏洞检测上的准确度. 但在真实项目测试集上的漏洞语句定位精度指标IoU只有32.6%，其检测性能仍有较大的提升空间. 蒋远[86]提出了一种序列生成模型Seq2SeqLocator，采用由BiGRU组成的解码器-编码器结构，学习代码段中每条语句的向量表示，再通过分类器对语句向量表示进行分类，从而实现细粒度的漏洞检测.

上述大多数细粒度的漏洞检测方法都使用了基于序列的深度神经网络，不利于提取代码的结构语义特征. 为了解决这个问题，近年来图神经网络开始被应用于细粒度的漏洞检测任务. 例如，Cao等人[125]提出的MVD方法和Dong等人[158]提出的SedSVD方法，其不同之处在于，MVD使用了流敏感的图神经网络（Flow-Sensitive Graph Neural Networks，FS-GNN），该方法仅能检测内存相关的漏洞. SedSCD则为了关注漏洞语句的更多上下文信息，减少漏洞无关信息的干扰，分析并归纳了10种可能和漏洞相关的中心节点，通过收集中心节点的邻居节点以及中心节点之间的关系来建立子图. 然后利用RGCN和MLP（Multi-Layer Perceptron）的组合来判断子图中的每个节点是否有漏洞，从而实现语句级的漏洞检测. Mirsky等人提出了一种可以精确定位漏洞语句并确定其漏洞类型的源代码漏洞检测方法VulChecker[182]. 该方法生成代码表示的过程和VulDeeLocator相似，均将代码编译成LLVM中间代码，然后使用程序切片技术来减少漏洞无关的

代码. 不同之处在于,VulChecker将代码编译成LLVM中间代码后,进一步生成其对应的图结构,称为ePDG. 然后,利用启发式方法根据不同漏洞类型提取其在代码中对应的潜在表现点(potential manifestation point),并以其为切片准则生成子图. 然后采用Structure2Vec模型[183]生成子图的节点嵌入. 最后,利用由全连接网络构成的分类器给出漏洞语句的分类结果.

近年来,开始有少量研究将大规模模型应用于细粒度漏洞检测领域. 例如,Fu等人[85]提出了一种基于Transformer的细粒度漏洞检测方法LineVul. LineVul利用BERT架构来捕获序列中的长依赖信息,并且通过架构中的自注意力机制进行语句级的漏洞定位. 具体而言,对于函数内的每一个Token,通过将BERT编码器中计算的每个Token的自注意力分数整合为每条语句的自注意力分数,并按分数对所有语句进行排序,从而确定漏洞语句的位置. 此外,也有研究将大规模模型与图神经网络相结合来同时捕获局部和全局的程序语义信息和结构信息. 例如,Hin等人[87]提出的LineVD方法通过预训练的CodeBERT提取每条语句内的语义信息,再利用GAT学习PDG语句节点间的数据和控制依赖关系得到语句的向量表示,然后通过MLP进行节点分类来实现细粒度漏洞检测. 考虑到GNN只能学习语句节点附近的局部上下文,而难以学习代码中更远的节点的语义信息[84],Ding等人[84]提出了一种基于集成学习定位漏洞语句的方法VELVET. 该方法利用GGNN和Transformer模型分别提取语句的局部和全局上下文,生成语句的局部和全局表示,然后通过MLP分别获得局部漏洞得分和全局漏洞得分,将二者的平均作为语句的集成漏洞得分,由于该方法仅将最后得分最高的节点作为漏洞节点,因此它只能检测单个漏洞语句,对于包含多个漏洞语句的程序会出现漏报. 在最近的一项研究中,Zhang等人[184]提出了ReVulDL,一种基于深度学习的两阶段智能合约漏洞检测方法. 在第一阶段,首先提取代码中有数据/控制依赖关系的代码片段,并利用数据流边将这些片段整合成链式结构,称为传播链(Propagation Chain,PC). 同时,提取Token序列、变量序列、Token位置序列、变量位置序列等关键信息. 然后,将上述信息送入GraphCodeBERT模型中以实现粗粒度的漏洞检测. 在第二阶段,在第一阶段构建的传播链和检测模型基础上,利用GNNExplainer[178]通过边掩码来寻找在传播链中能够解释漏洞的最小子传播链(minPC),根据数据依赖关系计算minPC中每个变量的可疑值,进而计算变量所属语句的可疑值,最后根据可疑值对所有语句进行排序,从而实现细粒度的漏洞定位.

上述方法大多需要将大规模模型和图神经网络结合才能同时捕获局部和全局的程序语义信息和结构信息,并且仅仅依据小规模的漏洞数据集难以训练具有上千万参数的大规模模型,导致漏洞特征语义提取不准确,而且由于不同模型的学习范式不同,不同类型的模型组合(例如Transformer+GGNN[84],CodeBERT+GAT[87])可能会导致训练复杂以及训练效率较低等问题. 此外,现有研究没有针对漏洞检测任务对大规模模型进行优化以解决深层漏洞语义难以提取的问题,也缺乏适合漏洞检测和定位的预训练任务用以解决大规模模型难以训练的问题. 另外,现有研究主要倾向于将其用于漏洞语义理解方面,较少有研究利用大规模模型来捕获程序的细粒度漏洞模式.

综上,虽然商业或开源的检测工具能实现语句级的细粒度漏洞检测,并且依据专家预先定义的规则得到的检测结果具有可解释性,但是专家预定义的检测规则的局限性和不完善往往会导致较高的误报率和漏报率[73,49]. 虽然基于深度学习的漏洞检测方法可以摆脱对专家人工制定规则的依赖,利用深度学习技术自动提取漏洞特征,可以学习代码中的漏洞模式,但目前大多数基于深度学习的漏洞检测方法主要是在文件、函数或代码片段等粗粒度水平上检测代码是否包含漏洞,仅有的一些细粒度漏洞检测方法,要么只能检测特定类型的漏洞,要么检测的准确率不高. 例如,文献[98]提出的VulDeeLocator方法其定位精度指标IoU仅为32.6%,MVD[125]在真实项目中检测的准确率(Accuracy)仅为67.6%,LineVD[87]在进行漏洞语句节点分类任务时其精确率(Precision)和召回率(Recall)仅分别为27.1%和53.3%,VELVET[84]的细粒度漏洞定位准确率(Localization Accuracy)仅为43.6%. 上述文献给出的实验数据说明目前最先进的基于深度学习的漏洞检测方法在真实项目上的细粒度漏洞检测的准确率较低. 这是因为在真实项目中漏洞语句所处的上下文语法结构和依赖关系更加复杂,漏洞类型也更加多样化,使得细粒度漏洞检测面临更大的挑战.

此外，目前大多数的细粒度漏洞检测方法仍然采用基于粗粒度标签计算的交叉熵损失函数作为目标函数来指导模型的训练，在模型训练的过程中缺少细粒度监督信号的指导，使其细粒度检测的准确率不高，很难满足实际应用的需要．实际上，细粒度漏洞检测的优化目标不应同粗粒度漏洞检测一样以整个代码段被预测为真实漏洞标签（1或者0）的概率最大化作为模型的优化目标，而应找到引发漏洞的语句集合，即让细粒度漏洞定位的评价指标 IoU（即真实的漏洞语句位置集合和预测的漏洞语句位置集合的交并比）达到最大化．由于不同的漏洞语句组合代表了不同的漏洞结构（例如语句的共现关系和条件依赖关系），因此，细粒度标签比粗粒度标签隐含了更多的与漏洞模式相关的信息．将细粒度标签中隐含的漏洞模式信息用于指导模型学习漏洞语句之间的共现关系和条件依赖关系，将有助于更准确地识别代码中的细粒度漏洞结构．

9　未来研究展望

9.1　结合层次化语义感知、多粒度漏洞检测和辅助漏洞理解的源代码漏洞检测参考框架

通过对已有文献方法存在问题的分析和总结，本文给出了一个面向多粒度漏洞检测和辅助漏洞理解的源代码漏洞检测参考框架，具体如图13所示．该框架建议包括：漏洞数据挖掘、漏洞数据表示、漏洞语义理解、多粒度漏洞检测以及漏洞辅助理解等5大模块．不同于一般的漏洞检测框架，该框架通过层次化语义感知的代码表示学习从代码中提取不同粒度和抽象级别的漏洞关键语义特征，实现代码段级和语句级的多粒度漏洞检测．该框架在提供语句级的细粒度漏洞检测结果的同时，还能进一步基于漏洞知识库或知识图谱为开发者提供用于辅助理解漏洞成因的漏洞知识和辅助修复漏洞的补丁知识．

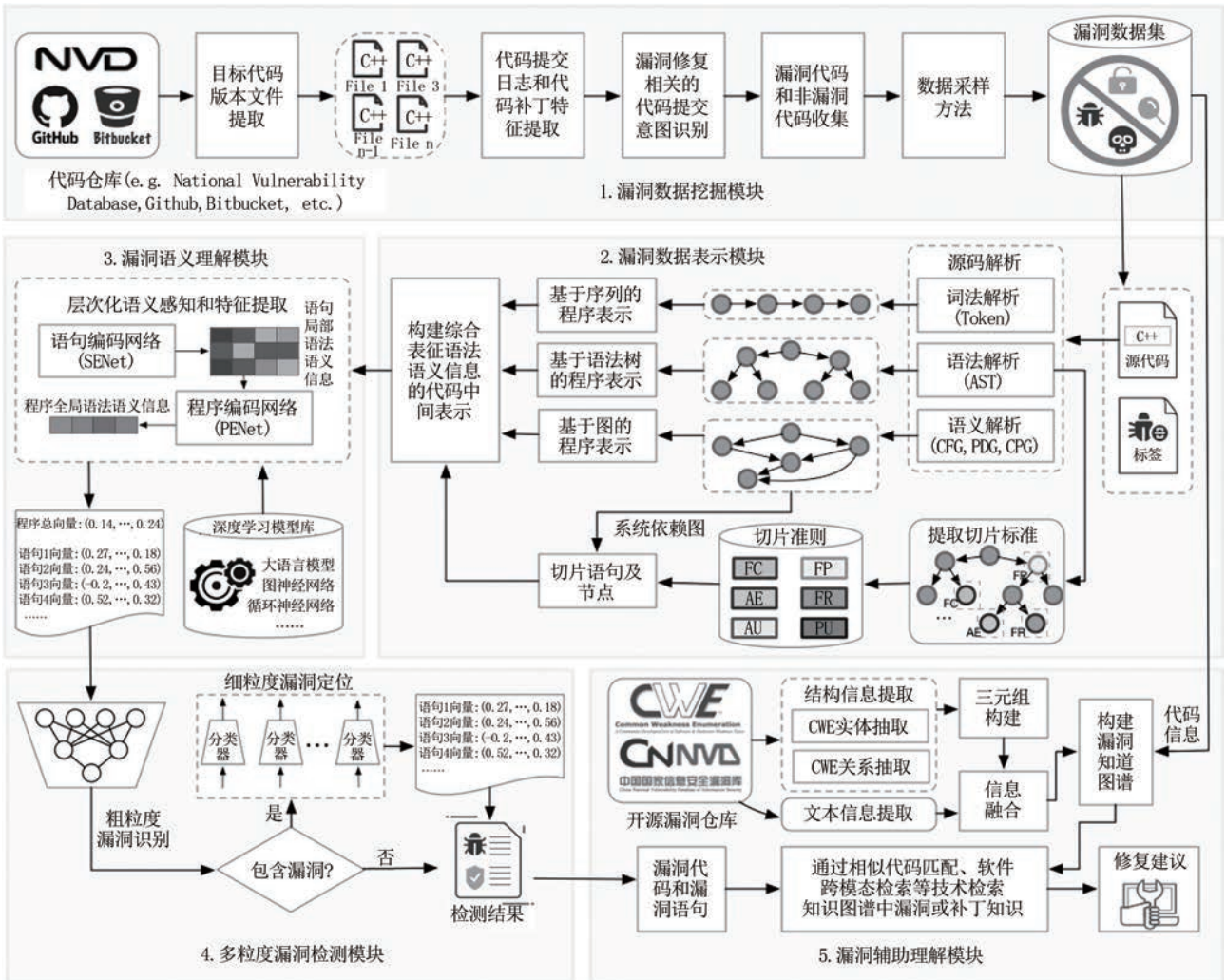

图13　面向多粒度漏洞检测和辅助漏洞理解的源代码漏洞检测参考框架

漏洞数据集构建模块的目标是通过挖掘多源异构软件仓库并联合使用漏洞数据挖掘、数据清洗和过滤以及漏洞数据增强等方法，来构建大规模高质量的真实漏洞数据集．考虑到联合挖掘NVD、CVE、NIST（National Institute of Standards and Technology）[185]和GitHub，并经过数据清洗和过滤后，收集到的漏洞代码有限且存在较为严重的样本不平衡问题，因此还可以考虑从Bugzilla、Stack Overflow[186]等其他来源挖掘更多的真实漏洞实例，在此基础上进一步采用数据增强技术以缓解样本不平衡问题．其基本思路为：首先挖掘代码仓库中目标项目各版本的代码文件，再对代码提交日志中的变更信息进行分析，利用代码提交意图识别技术提取和漏洞修复相关的代码提交，然后定位到修复前后的代码版本，根据差异文件收集代码中的漏洞代码以及对应的补丁代码，最后利用数据采样方法来改善收集到的样本，从而构建出漏洞数据集．

漏洞数据表示模块的目标是通过词法和语法解析等程序分析技术研究构建综合表征漏洞相关语法语义信息的代码中间表示．其基本思路为：首先，利用词法分析、语法分析和语义分析分别构建出源代码基于序列、基于语法树和基于图的程序表示．然后，根据抽象语法树提取出切片准则，利用切片准则和程序依赖图进一步地提取切片节点．最后，结合其他程序表示中的顺序关系、语法结构、控制依赖和数据依赖等信息，构建代码的中间表示．

漏洞语义理解模块的目标是研究层次化语义感知的代码表示模型，需要同时兼顾模型复杂度和对不同粒度和抽象级别的漏洞语义理解和表征能力．其基本思路为：首先构建语句编码网络，从漏洞数据表示模块生成的代码中间表示中提取程序语句内局部的语法和语义特征，得到每条语句的向量表示，然后通过程序编码网络进一步提取代码的全局语义特征，得到程序的向量表示，从而为实现多粒度漏洞检测奠定基础．在设计语句和程序编码网络时，除了传统深度神经网络外，也可以考虑使用大语言模型，即在经过预训练的基础上，通过漏洞语义理解任务对模型进行微调，来获得语句和程序的向量表示．

多粒度漏洞检测模块的目标是充分利用漏洞语义理解模块生成的包含了局部和全局的语法语义特征的语句和程序的向量表示，在预测函数或代码段包含漏洞的可能性的基础上，进一步定位到引发漏洞的漏洞语句．其基本思路为：首先将程序的向量表示送入粗粒度漏洞识别模型，判断整个源代码是否包含漏洞．若是，则进一步将语句向量表示送入细粒度漏洞定位模型，输出源代码中的疑似漏洞语句序列．

源代码漏洞的修复要求开发者具备专业的代码与网络安全知识，但并非所有的开发者都具备该能力，因此本文设计了漏洞辅助理解模块来辅助开发者快速理解并修复漏洞．其基本思路为：首先，通过挖掘软件仓库和漏洞仓库来构建面向漏洞与补丁知识搜索的漏洞知识图谱，具体来说，对于开源漏洞仓库，可以分别提取其中的结构信息（CWE实体和CWE关系）和文本信息，对于开源软件仓库，可以提取其中的代码信息，再通过上述信息来构建漏洞知识图谱．然后，以漏洞代码为查询搜索漏洞知识（如漏洞描述、漏洞类型和特征等），具体地，可以通过相似代码匹配、跨模态检索等技术检索知识图谱中的漏洞或补丁知识以辅助开发者理解漏洞，搜索补丁知识（如提交日志、补丁代码和差异文件等）以辅助开发者修复漏洞．

9.2 参考框架可行性分析

漏洞数据挖掘技术是目前漏洞数据收集最常用的手段，已公开的漏洞数据集[45-51,76-81]中的真实项目数据大多是通过CVE、NIST和GitHub的联合挖掘收集而来．虽然CVE中记录了大量的漏洞报告，但是大多数项目出于商业或隐私目的没有将代码开源，或出于安全目的将漏洞代码隐藏起来，导致通过上述方式收集到的漏洞代码数量仍然有限，样本失衡问题也比较严重．数据增强技术是解决数据失衡问题的有效手段，在其他领域已有较为成熟的应用．例如，Rao等人[187]采用相关过采样技术增加缺陷样本，Shen等人[188]采用基于半监督技术来扩充平衡样本．这些应对样本失衡问题的技术也可被漏洞样本生成所借鉴．例如，Chakraborty[73]和Liu等人[189]也采用过采样技术来生成新的漏洞样本，此外在计算机视觉领域常用的生成式对抗网络也可用于生成新的漏洞样本[190]．因此，本文的建议，利用漏洞数据挖掘和漏洞数据增强技术相结合的方法来构建更大规模的漏洞数据集，是可行的．

在漏洞数据表示模块，目前较为先进的漏洞检测方法是通过联合多种代码表示来表征程序中丰富的语法语义，虽然已有很多研究成果可供参考，如Yamaguchi等人[28]提出的CPG、Zhou等人[47]提出的复合图等，但是这类代码表示中往往包含了大量和漏洞无关的信息．为了避免漏洞无关信息对模型学习漏洞模式造成干扰，本文建议使用程序切片技术

来进一步优化代码表示，目前已有相关成果证明了切片技术在漏洞检测上的有效性，例如，Li等人[49]提出的SeVCs，Zheng等人[60]提出的SPG等．最后，CodeBERT等预训练模型已被证明在多个编程语言相关任务中的有效性[191-193]，并且可以通过微调适用于漏洞检测任务[194]．

在漏洞语义理解模块，已有研究证实了分段代码表示学习模型在克隆代码检测、程序分类等任务上的有效性，例如，An等人[71]使用两个BiGRU模型学习从语句到程序的语义信息，Yan等人[55]使用BiGRU和textCNN实现提取局部和全局的语义特征，Li等人[67]使用Tree-LSTM和GRU学习PDG中语句节点的局部语义信息，再使用FA-GCN学习程序的全局语义信息，以及Jiang等人[170]提出的层次化语义感知的代码表示学习框架等．因此，本文建议通过由语句编码网络和程序编码网络构成的分段代码表示学习框架来有效地捕获程序的局部漏洞语法结构和全局漏洞语义信息具有可行性．

在多粒度漏洞检测模块，尽管现有的漏洞检测技术大都是粗粒度或者细粒度的，但近期的一些研究已经证明了多粒度漏洞检测的可行性和有效性，例如Zou等人[173]所提出的多粒度漏洞检测方法，首先进行切片级的漏洞检测，然后进行函数级的漏洞检测，从而提高了函数级粗粒度漏洞检测的准确率．与该方法不同的是，本文建议的多粒度漏洞检测模型是先进行函数级或切片级的粗粒度漏洞检测，然后针对预测有漏洞的代码进行语句级的细粒度漏洞检测，该方法的最终目标是提高细粒度漏洞检测的准确率．

在漏洞辅助理解模块，本文建议通过构建漏洞知识图谱，将漏洞知识以图结构的形式整合起来，然后利用图搜索技术来提高漏洞知识和补丁知识的获取效率并增强可解释性．知识图谱本质是一种语义网络，提供了一种更好的组织和管理信息的途径，而漏洞知识图谱则是根据漏洞相关知识建立的．目前已有研究探索了漏洞知识图谱的构建方法，例如Han等人[195]基于通用缺陷枚举(Common Weakness Enumeration，CWE)数据库构建了CWE知识图谱，又如Qin等人[196]通过挖掘CVE数据库进一步扩展丰富了漏洞知识图谱的规模和内容．因此，构建漏洞知识图谱，并利用已有的代码搜索技术[197,198]从构建的漏洞知识图谱中搜索与漏洞代码相关的漏洞知识和补丁知识是可行的，该技术能够辅助开发人员理解和修复漏洞．

9.3 未来的研究方向和发展趋势

近年来，随着人工智能技术的发展，基于学习的源代码漏洞检测技术也得到了长足的发展，但是随着现代软件的规模日益增大、功能和架构日益复杂，源代码漏洞的形态也日趋复杂化和多样化，这些都对基于学习的源代码漏洞检测技术提出了严峻的挑战．通过总结已有的研究成果，本文提出了未来可能的研究方向和发展趋势：

(1) 大规模高质量的真实漏洞数据集构建技术

构建开放、共享、高质量的真实漏洞数据集，对于提高基于学习的漏洞检测模型的泛化能力和实用性至关重要．可以研究代码变更意图识别和安全相关的Issue Report或Bug Report识别方法，通过利用文本挖掘和程序分析技术从代码仓库或bug仓库中自动挖掘和标注真实的漏洞数据，也可以结合模糊测试和静态分析技术、探索新的漏洞数据挖掘方法和可信的漏洞自动标注方法．在构建并开放共享大规模高质量的真实漏洞数据集的基础上，还需要建立统一开放的测试平台，以便为分析和评估漏洞数据的质量提供支撑．

(2) 少样本和少标签条件下的漏洞检测技术

建立大规模且高质量的有可信标签的漏洞数据集是一项过程复杂且成本高昂的系统工程．真实漏洞样本的稀缺会导致训练集中的漏洞结构与类型不够丰富，从而限制了模型的泛化能力，而因此，为了提高少样本和少标签条件下的模型检测能力和泛化能力，需要研究面向漏洞检测的代码数据增强技术以及半监督或弱监督环境下的模型训练技术．这是一个现实而又重要的研究方向．

(3) 全面性和高效性平衡的漏洞语义理解技术

漏洞语义理解的关键在于如何从有限的样本数据中提取更为全面和关键的漏洞语义特征，不仅要有机融合多种形式的代码中间表示从不同的“视角”刻画代码的关键漏洞语义特征，还要利用层次化的代码表示学习从不同的“粒度”提取代码的关键漏洞语义特征，还可以考虑从不同的编译阶段(例如源代码、LLVM中间代码、汇编代码等)来学习不同程序模态的关键漏洞语义特征，结合程序切片技术实现多视角、多粒度、多层级的漏洞特征提取，以提高漏洞检测的性能，并在模型的有效性和高效性之间取得平衡，这是实现漏洞语义理解的一个未来发展方向．

(4) 细粒度的漏洞检测模型

相比于文件、函数、代码段等粗粒度的源代码漏

洞检测方法，语句级的细粒度漏洞检测可以为后续修复漏洞提供更多的指导信息．近年来，由于图形神经网络可以直接对代码的结构进行建模，能够有效利用程序的结构信息，因此提取代码的基于图的中间表示，通过图神经网络学习语句节点的特征，然后利用节点分类来定位漏洞语句，已成为比较常见的做法[87,125]．然而，图神经网络本身也有一些局限性．首先，图神经网络的计算成本很高，尤其是在处理大型图的场景．随着神经网络中层数的增加，存储图结构信息和计算特征所消耗的内存和其他资源呈指数级增长．其次，图神经网络模型容易出现过拟合现象．最后，图神经网络难以聚合距离较远节点之间的信息，因此难以学习程序中语句之间的长依赖关系．因此，突破原有模型的局限性，开发合适的检测模型仍是提高软件细粒度漏洞检测方法性能的一个未来发展方向．

(5) 漏洞检测可解释方法

漏洞检测模型和检测结果的可解释性也有助于开发人员理解漏洞的成因和发生机制，以便快速修复漏洞．现有的可解释方法侧重于解释代码中的哪些语法单元对模型检测漏洞起到了重要的影响或作用，但无法解释语法单元之间是如何相互作用而使得模型能够将其识别为漏洞代码，结合漏洞知识图谱中的外部安全领域知识，有望能够提供更多的解释，如何实现这一目标还有待今后更深入的研究．

(6) 安全漏洞的提前感知技术

只依赖NVD进行安全漏洞管理，将使开源软件(Open Source Software，OSS)用户处于极高的安全风险中，因此OSS用户急需提前感知安全漏洞的手段来进行提前防御．有研究发现，在漏洞被发现、确认和修复前，通常它们已在公开渠道中被讨论过，即从漏洞被讨论到漏洞被正式披露通常存在数天甚至数月的静默期时间差，在这个时间窗口期内，漏洞很可能会被攻击者所利用实施n-day攻击，尤其是对于高危类型的漏洞，更需要社区评论挖掘、漏洞舆情发现等技术手段来提前感知它们，这也是未来需要重点关注的研究内容之一，但是高危漏洞的样本更为稀缺和不平衡，无疑高危漏洞的提前感知将会面临更大的技术挑战．

(7) 利用领域知识强化漏洞检测模型

尽管基于学习的源代码漏洞检测模型在漏洞检测领域表现出了优秀的性能，但它们仍存在一些显著的缺陷，包括高度的数据依赖性、不稳定的泛化性能以及缺乏可解释性．另一方面，各类代码仓库中存在的大量安全相关的领域知识未被充分利用．充分利用这些领域知识，将在一定程度上弥补基于学习的漏洞检测模型的不足．例如，对于基于传统机器学习的漏洞检测方法，可以利用安全领域知识可以设计更适合机器学习模型的特征，降低人工特征工程的成本．再如，对于基于深度学习的漏洞检测方法，可以利用安全领域知识来解释模型的预测结果，或辅助开发者理解和修复漏洞．因此，将领域知识融入基于学习的漏洞检测模型，提高检测性能，也是未来值得深入研究的发展方向．

(8) 基于大语言模型的源代码漏洞检测方法

大语言模型是目前人工智能领域的研究热点，尽管已有部分研究显示，其在漏洞检测任务中的应用还需进一步完善和深入研究．但考虑到大语言模型在自然语言处理和代码分析等相关任务中的成功应用，它的潜力仍是毋庸置疑的．未来可以考虑基于提示学习技术改进大语言模型的漏洞检测结果．例如，增加对安全领域知识的利用，设计合适的源代码预训练任务．因此，基于大语言模型的漏洞检测是未来值得深入研究且很有前景的一个方向．

10 结 论

源代码漏洞已成为软件与信息系统安全的主要威胁，研究源代码漏洞检测技术对于减少软件安全漏洞、降低软件安全风险具有重要意义．基于学习的漏洞检测技术由于具有较低的人工成本、更高的检测效率等优势，近年来已成为漏洞检测领域的研究热点．本文通过深入调研软件工程方向的期刊和会议，围绕着漏洞数据集的挖掘和构建、面向漏洞检测任务的程序表示方法、常用的基于机器学习和深度学习的漏洞检测方法、源代码漏洞的可解释方法、细粒度的源代码漏洞检测方法等关键问题对已有的研究工作进行了系统梳理和总结，在此基础上，给出了一种结合层次化语义感知、多粒度漏洞分类和辅助漏洞理解的漏洞检测框架．最后，对基于学习的漏洞检测技术的未来研究方向和发展趋势进行了展望．

致 谢 感谢各位专家提出的宝贵意见，感谢国家自然科学基金项目(No. 62272132)的资助．

参 考 文 献

[1] Hu Chao-Jian. Vulnerability detection based on program analysis

[Ph. D. dissertation]. Beihang University, Beijing, 2012 (in Chinese)
(呼朝俭.基于程序分析的漏洞检测技术研究[博士学位论文].北京航空航天大学，北京，2012)
[2] Ettredge ML, Richardson VJ. Information transfer among internet firms: the case of hacker attacks. Journal of Information Systems, 2003, 17(2):71-82
[3] Mendell RL. Investigating Information-Based Crimes: Aguide for Investigators on Crimes Against Persons Related to the theft or Manipulation of Information Assets. New York, USA: Charles C Thomas Publisher, 2013
[4] Malandrino D, Petta A, Scarano V, Serra L, Spinelli R, Krishnamurthy B. Privacy awareness about information leakage: Who knows what about me? //Proceedings of the Workshop on Privacy in the Electronic Society. Berlin, Germany. 2013: 279-284
[5] Li Zhen, Zou De-Qing, Wang Ze-Li, et al. Survey on static software vulnerability detection for source code. Chinese Journal of Network and Information Security, 2019, 5(1): 1-14 (in Chinese)
(李珍，邹德清，王泽丽等.面向源代码的软件漏洞静态检测综述.网络与信息安全学报，2019，5(1)：1-14)
[6] Li Yun, Huang Chen-Lin, Wang Zhong-Feng, et al. Survey of software vulnerability mining methods based on machine learning. Journal of Software, 2020, 31(07): 2040-2061 (in Chinese)
(李韵，黄辰林，王中锋等.基于机器学习的软件漏洞挖掘方法综述.软件学报，2020，31(07)：2040-2061)
[7] Sun Hong-Yu, He Yuan, Wang Ji-Ce, et al. Application of artificial intelligence technology in the field of security vulnerability. Journal on Communications, 2018, 39(8): 1-17 (in Chinese)
(孙鸿宇，何远，王基策等.人工智能技术在安全漏洞领域的应用.通信学报，2018，39(8)：1-17)
[8] https://www.freebuf.com/vuls/208158.html
[9] https://github.com/cisagov/log4j-affected-db
[10] Common Vulnerabilities & Exposures. 2022, https://cve.mitre.org/.
[11] National Vulnerability Database. 2022, https://nvd.nist.gov/.
[12] Li Zhou-Jun, Zhang Jun-Xian, Liao Xiang-Ke, et al. Survey of software vulnerability detection techniques. Chinese Journal of Computers, 2015, 38(4): 717-732 (in Chinese)
(李舟军，张俊贤，廖湘科等.软件安全漏洞检测技术.计算机学报，2015，38(4)：717-732)
[13] https://dwheeler.com/flawfinder/
[14] https://code. google. com/archive/p/rough-auditing-tool-for-security/
[15] https://sourceforge.net/projects/cppcheck/
[16] http://findbugs.sourceforge.net/
[17] https://github.com/google/error-prone
[18] Chen S, Fan L, Meng G, Su T, Xue M, Xue Y, Liu Y, Xu L. An empirical assessment of security risks of global android banking apps//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Seoul, Korea, 2020: 1310-1322
[19] Chen S, Zhang Y, Fan L, Li J, Liu Y. AUSERA: Automated security vulnerability detection for android apps//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, USA, 2022: 1-5
[20] http://clang-analyzer.llvm.org/
[21] https://www.checkmarx.com/
[22] https://scan.coverity.com/
[23] https://www.microfocus.com/en-us/cyberres/application-security
[24] https://codesecure.com/our-products/codesonar/
[25] http://www.klocwork.com/
[26] https://phpmagazine.net/2007/04/armorize-codesecure-on-demand-php-source-code-analysis.html
[27] https://www.perforce.com/products/helix-qac
[28] Yamaguchi F, Golde N, Arp D, Rieck K. Modeling and discovering vulnerabilities with code property graphs//Proceedings of the Symposium on Security and Privacy. San Jose, USA, 2014: 590-604
[29] Zimmermann T, Nagappan N, Williams L. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista//Proceedings of International Conference on Software Testing, Verification and Validation. Paris, France, 2010: 421-428
[30] Moshtari S, Sami A, Azimi M. Using complexity metrics to improve software security. Computer Fraud & Security, 2013 (5):8-17
[31] Morrison P, Herzig K, Murphy B, Williams L. Challenges with applying vulnerability prediction models//Proceedings of the 2015 Symposium and Bootcamp on the Science of Security. Urbana Illinois, USA, 2015:1-9
[32] Salimi S, Ebrahimzadeh M, Kharrazi M. Improving real-world vulnerability characterization with vulnerable slices//Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering. New York, USA, 2020:11-20
[33] Younis A, Malaiya Y, Anderson C, Ray I. To fear or not to fear that is the question: code characteristics of a vulnerable function with an existing exploit//Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. New Orleans Louisiana, USA, 2016:97-104
[34] Scandariato R, Walden J, Hovsepyan A, Joosen W. Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering, 2014, 40(10):993-1006
[35] Yamaguchi F, Maier A, Gascon H, Rieck K. Automatic inference of search patterns for taint-style vulnerabilities//Proceedings of the Symposium on Security and Privacy. San Jose, USA, 2015:797-812
[36] Meneely A, Williams L. Strengthening the empirical analysis of the relationship between Linus' Law and software security//Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. Bolzano-Bozen, Italy, 2010:1-10
[37] Shin Y, Meneely A, Williams L, Osborne JA. Evaluating complexity, code churn, and developer activity metrics as

indicators of software vulnerabilities. IEEE Transactions on Software Engineering, 2011, 37(6):772-87

[38] Shin Y, Williams L. Can traditional fault prediction models be used for vulnerability prediction//Empirical Software Engineering.Baltimore. Maryland, USA, 2013, 18(1):25-59

[39] Walden J, Stuckman J, Scandariato R. Predicting vulnerable components: Software metrics vs text mining//Proceedings of the International Symposium on Software Reliability Engineering.Naples, Italy, 2014:23-33

[40] Yamaguchi F, Rieck K. Vulnerability extrapolation: assisted discovery of vulnerabilities using machine learning//Proceedings of the 5th USENIX Workshop on Offensive Technologies (WOOT 11). FranciscoSan, USA, 2011: 1-10

[41] Shar LK, Tan HB. Predicting common web application vulnerabilities from input validation and sanitization code patterns//Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. Essen, Germany, 2012:310-313

[42] Shar LK, Tan HB. Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. Information and Software Technology, 2013, 55(10): 1767-80

[43] Shar LK, Tan HB, Briand LC.Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis// Proceedings of the International Conference on Software Engineering (ICSE). San Francisco, USA, 2013:642-651

[44] Yamaguchi F, Lottmann M, Rieck K.Generalized vulnerability extrapolation using abstract syntax trees//Proceedings of the 28th Annual Computer Security Applications Conference. Orlando Florida, USA, 2012: 359-368

[45] Lin G, Zhang J, Luo W, et al. POSTER: Vulnerability discovery with function representation learning from unlabeled projects//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas, Texas, USA, 2017: 2539-2541

[46] Wu Y, Zou D, Dou S, Yang W, Xu D, Jin H. VulCNN: An image-inspired scalable vulnerability detection system// Proceedings of the International Conference on Software Engineering.Pittsburgh, USA, 2022: 21-29

[47] Zhou Y, Liu S, Siow J, Du X, Liu Y. Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks//Proceedings of Theadvances in Neural Information Processing Systems. Vancouver, Canada, 2019, 32:1-11

[48] Li Z, Zou D, Xu S, et al.VulDeePecker: a deep learning-based system for vulnerability detection//Proceedings of the 25th Annual Network and Distributed System Security Symposium. San Diego, USA, 2018:1-15

[49] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, SySeVR: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing, 2021:1-16

[50] Zou D, Wang S, Xu S, Li Z, Jin H.μVulDeePecker: A deep learning-based system for multiclass vulnerability detection. IEEE Transactions on Dependable and Secure Computing, 2019,18(5):2224-2236

[51] Russell R, Kim L, Hamilton L, et al. Automated vulnerability detection in source code using deep representation learning// Proceedings of the International Conference on Machine Learning and Applications (ICMLA). Orlando, USA, 2018: 757-762

[52] Wang H, Ye G, Tang Z, et al. Combininggraph-based learning with automated data collection for code vulnerability detection. IEEE Transactions on Information Forensics and Security, 2020,16:1943-58

[53] Wang S, Liu T, Tan L. Automatically learning semantic features for defect prediction//Proceedings of the 38th International Conference on Software Engineering. Austin, USA, 2016: 297-308

[54] Wu F, Wang J, Liu J, Wang W. Vulnerability detection with deep learning//Proceedings of the International Conference on Computer and Communications. Chengdu, China, 2017: 1298-1302

[55] Yan H, Luo S, Pan L, Zhang Y.HAN-BSVD: a hierarchical attention network for binary software vulnerability detection. Computers & Security, 2021, 108:102286

[56] Tian J, Xing W, Li Z. BVDetector: a program slice-based binary code vulnerability intelligent detection system. Information and Software Technology, 2020,123:106289

[57] Thapa C, Jang SI, Ahmed ME, Camtepe S, Pieprzyk J, Nepal S. Transformer-based language models for software vulnerability detection//Proceedings of the 38th Annual Computer Security Applications Conference. Austin, USA, 2022:481-496

[58] Mamede C, Pinconschi E, Abreu R. A transformer-based IDE plugin for vulnerability detection//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, USA. 2022:1-4

[59] Cao S, Sun X, Bo L, Wei Y, Li B.BGNN4VD: constructing bidirectional graph neural-network for vulnerability detection. Information and Software Technology, 2021, 136:106576

[60] Zheng W, Jiang Y, Su X. VulSPG: vulnerability detection based on slice property graph representation learning// Proceedings of the International Symposium on Software Reliability Engineering.Wuhan, China, 2021:457-467

[61] Duan X, Wu J, Ji S, Rui Z, Luo T, Yang M, Wu Y. VulSniper: focus your attention to shoot fine-grained vulnerabilities//Proceedings of the International Joint Conferences on Artificial Intelligence, Macao, China, 2019: 4665-4671

[62] Duan Xu, Wu Jing-Zheng, Luo Tian-yue, et al. Vulnerability mining method based on code property graph and attention BiLSTM. Journal of Software, 2020, 31(11): 3404-3420 (in Chinese)
(段旭, 吴敬征, 罗天悦等. 基于代码属性图及注意力双向LSTM的漏洞挖掘方法, 软件学报. 2020, 31(11):3404-3420)

[63] Wang Y, Wang K, Gao F, Wang L.Learning semantic program embeddings with graph interval neural network//Proceedings of the ACM on Programming Languages.Chicago, USA, 2020, 4: 1-27

[64] Cao D，Huang J，Zhang X，Liu X. FTCLNet：Convolutional LSTM with fourier transform for vulnerability detection//Proceedings of the International Conference on Trust，Security and Privacy in Computing and Communications（TrustCom). Guangzhou，China，2020：539-546

[65] Li M，Li C，Li S，Wu Y，Zhang B，Wen Y. ACGVD：Vulnerability detection based on comprehensive graph via graph neural network with attention//Proceedings of the International Conference on Information and Communications Security. Chongqing，China，2021：243-259

[66] Duan Ya-Nan. Research on software vulnerability detection method based on code property graph and graph convolutional neural network［Master dissertation］. Harbin Institute of Technology，Harbin，2020（in Chinese）
（段亚男. 基于代码属性图和图卷积神经网络的软件漏洞检测方法研究[硕士学位论文].哈尔滨工业大学，哈尔滨，2020）

[67] Li Y，Wang S，Nguyen TN. Vulnerability detection with fine-grained interpretations//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Athens，Greece，2021：292-303

[68] Cheng X，Wang H，Hua J，Xu G，Sui Y. DeepWukong：Statically detecting software vulnerabilities using deep graph neural network. ACM Transactions on Software Engineering and Methodology（TOSEM），2021，30(3)：1-33

[69] Ghaffarian SM，Shahriari HR. Neural software vulnerability analysis using rich intermediate graph representations of programs. Information Sciences，2021，553：189-207

[70] Zhao J，Guo S，Mu D. DouBiGRU-A：Software defect detection algorithm based on attention mechanism and double BiGRU. Computers & Security，2021，111：102459

[71] An W，Chen L，Wang J，Du G，Shi G，Meng D. AVDHRAM：automated vulnerability detection based on hierarchical representation and attention mechanism//Proceedings of the 2020 IEEE International Conference on Parallel & Distributed Processing with Applications，Big Data & Cloud Computing，Sustainable Computing & Communications，Social Computing & Networking（ISPA/BDCloud/SocialCom/SustainCom). Exeter，England，2020：337-344

[72] Yamaguchi F. Pattern-based vulnerability discovery［Ph. D. dissertation］. Georg-August University School of Science，Göttingen，2015

[73] Chakraborty S，Krishna R，Ding Y，Ray B. Deep learning based vulnerability detection：are we there yet. IEEE Transactions on Software Engineering，2021，1(01)：1-17

[74] Ferrante J，Ottenstein KJ，Warren JD. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems（TOPLAS），1987，9(3)：319-49

[75] Harvey B. Computer science logo style：symbolic computing. Bosten，USA：MIT Press；1997

[76] Lin G，Zhang J，Luo W，et al. Cross-project transfer representation learning for vulnerable function discovery. IEEE Transactions on Industrial Informatics，2018，14(7)：3289-3297

[77] Lin G，Zhang J，Luo W，et al. Software vulnerability discovery via learning multi-domain knowledge bases. IEEE Transactions on Dependable and Secure Computing，2019(99)：1-1

[78] Lin G，Xiao W，Zhang J，et al. Deep learning-based vulnerable function detection：Abenchmark//Proceedings of the International Conference on Information and Communications Security. Irbid，Jordan，2019：219-232

[79] Lin G，Xiao W，Zhang LY，Gao S，Tai Y，Zhang J. Deep neural-based vulnerability discovery demystified：Data，model and performance. Neural Computing and Applications，2021，33(20)：13287-300

[80] Dong F，Wang J，Li Q，et al. Defect prediction in android binary executables using deep neural network. Wireless Personal Communications，2018，102(3)：2261-2285

[81] Zheng Y，Pujar S，Lewis B，Buratti L，Epstein E，Yang B，Laredo J，Morari A，Su Z. D2a：A dataset built for ai-based vulnerability detection methods using differential analysis//Proceedings of the 43rd International Conference on Software Engineering：Software Engineering in Practice（ICSE-SEIP）2021：111-120

[82] Yang X，Wang S，Li Y，Wang S. Does data sampling improve deep learning-based vulnerability detection? Yeas! and Nays!//Proceedings of the International Conference on Software Engineering. Melbourne，Australia，2023：2287-2298

[83] Kubat M，Matwin S. Addressing the curse of imbalanced training sets：one-sided selection//Proceedings of the International Conference on Machine Learning. Nashville，USA.1997，97(1)：179-187

[84] Ding Y，Suneja S，Zheng Y，Laredo J，Morari A，Kaiser G，Ray B. VELVET：A novel ensemble learning approach to automatically locate vulnerable statements//Proceedings of the 2022 IEEE International Conference on Software Analysis，Evolution and Reengineering（SANER). Honululu，USA，2022：959-970

[85] Fu M，Tantithamthavorn C. LineVul：a transformer-based line-level vulnerability prediction//Proceedings of the 19th International Conference on Mining Software Repositories. Pittsburgh Pennsylvania，USA，2022：608-620

[86] Jiang Yuan. Research on Data-Driven Program Security Bug Identification Method.［Ph.D. dissertation］. Harbin Institute of Technology. Harbin，2022（in chinese）
（蒋远. 数据驱动的程序安全缺陷识别方法研究[博士学位论文].哈尔滨工业大学，哈尔滨，2022）

[87] Hin D，Kan A，Chen H，Babar MA. LineVD：statement-level vulnerability detection using graph neural networks//Proceedings of the 19th International Conference on Mining Software Repositories（MSR'22). Association for Computing Machinery). New York，USA.2022：596-607

[88] Zhang L，Yan L，Zhang Z，Zhang J，Chan WK，Zheng Z. A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization. Journal of Systems and Software，2017，129：35-57

[89] Zhang Z, Lei Y, Mao X, Yan M, Xu L, Wen J. Improving deep-learning-based fault localization with resampling. Journal of Software: Evolution and Process, 2021, 33(3):2312-2330

[90] Xie H, Lei Y, Yan M, Yu Y, Xia X, Mao X. A universal data augmentation approach for fault localization//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh Pennsylvania, USA, 2022:48-60

[91] Hu J, Xie H, Lei Y, Yu K. A light-weight data augmentation method for fault localization. Information and Software Technology, 2023:107148-107160

[92] Lei Y, Liu C, Xie H, Huang S, Yan M, Xu Z. BCL-FL: Adata augmentation approach with between-class learning for fault localization//Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Honululu, USA, 2022:289-300

[93] Croft R, Xie Y, Babar MA. Data preparation for software vulnerability prediction: asystematic literature review. IEEE Transactions on Software Engineering, 2022:1044-1063

[94] Software assurance reference dataset. 2022, https://samate.nist.gov/SRD/index.php.

[95] https://github.com/

[96] Okun V, Delaitre A, Black PE. Report on the static analysis tool exposition (sate) iv. NIST Special Publication, 2013, 500: 297-343

[97] Dowd M, McDonald J, Schuh J. The art of software security assessment: Identifying and preventing software vulnerabilities. London, UK: Pearson Education, 2006

[98] Li Z, Zou D, Xu S, Chen Z, Zhu Y, Jin H. Vuldeelocator: a deep learning-based fine-grained vulnerability detector. IEEE Transactions on Dependable and Secure Computing, 2021:1-17

[99] https://www.bugzilla.org/

[100] https://security-team.debian.org/security_tracker.html

[101] Zafar S, Malik M Z, Walia G S. Towards standardizing and improving classification of bug-fix commits//Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM). Porto de Galinhas, Recife, Brazil, 2019: 1-6

[102] Ni Z, Li B, Sun X, et al. Analyzing bug fix for automatic bug cause classification. Journal of Systems and Software, 2020, 163: 110538-110552

[103] Perl H, Dechand S, Smith M, et al. Vccfinder: finding potential vulnerabilities in open-source projects to assist code audits// Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. Denver, USA, 2015: 426-437

[104] Zhou Y, Sharma A. Automated identification of security issues from commit messages and bug reports//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn, Germany, 2017: 914-919

[105] Sabetta A, Bezzi M. A practical approach to the automatic classification of security-relevant commits//Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME). Madrid, Spain, 2018: 579-582

[106] Oyetoyan TD, Morrison P. An improved text classification modelling approach to identify security messages in heterogeneous projects. Software Quality Journal, 2021, 29(2): 509-53

[107] Ramos J. Using tf-idf to determine word relevance in document queries//Proceedings of the First Instructional Conference on Machine Learning. Washington, USA. 2003, 242(1):29-48

[108] Pane JF, Myers BA. Studying the language and structure in non-programmers' solutions to programming problems. International Journal of Human-Computer Studies, 2001, 54 (2):237-64

[109] Mou L, Li G, Zhang L, Wang T, Jin Z. Convolutional neural networks over tree structures for programming language processing//Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. Phoenix, USA, 2016: 30(1): 1-7

[110] Henkel J, Lahiri SK, Liblit B, Reps T. Code vectors: Understanding programs through embedded abstracted symbolic traces//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista, USA, 2018: 163-174

[111] Tufano M, Watson C, Bavota G, Di Penta M, White M, Poshyvanyk D. Deep learning similarities from different representations of source code.//Proceedings of the 2018 IEEE/ ACM 15th International Conference on Mining Software Repositories (MSR). Gothenburg, Sweden, 2018: 542-553

[112] Zhao G, Huang J, DeepSim: deep learning code functional similarity//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista, USA. 2018: 141-151

[113] Liu Fang, Li Ge, Hu Xing, et al. Program comprehension based on deep learning. Joural of Computer Research and Development. 2019, 56(08): 1605-1620 (in Chinese)
(刘芳, 李戈, 胡星等. 基于深度学习的程序理解研究进展. 计算机研究与发展. 2019, 56(08):1605-1620)

[114] Bosu A, Carver JC, Hafiz M, Hilley P, Janni D. Identifying the characteristics of vulnerable code changes: an empirical study// Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong, China, 2014:257-268

[115] Meneely A, Srinivasan H, Musa A, Tejeda AR, Mokary M, Spates B. When a patch goes bad: Exploring the properties of vulnerability-contributing commits//Proceedings of the 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Baltimore, USA, 2013: 65-74

[116] Shin Y, Williams L. An initial study on the use of execution complexity metrics as indicators of software vulnerabilities// Proceedings of the 7th International Workshop on Software Engineering for Secure Systems, Honolulu ,USA. 2011: 1-7

[117] Du X, Chen B, Li Y, Guo J, Zhou Y, Liu Y, Jiang Y. Leopard: Identifying vulnerable code for vulnerability assessment through program metrics//Proceedings of the 41st International Conference on Software Engineering (ICSE). Montréal, Canada. 2019:60-71

[118] Zagane M, Abdi MK, Alenezi M. Deep learning for software vulnerabilities detection using code metrics. IEEE Access, 2020 (8):74562-74570

[119] Grieco G, Grinblat GL, Uzal L, Rawat S, Feist J, Mounier L. Toward large-scale vulnerability discovery using machine learning//Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, USA, 2016: 85-96

[120] Hovsepyan A, Scandariato R, Joosen W, Walden J. Software vulnerability prediction using text analysis techniques// Proceedings of the 4th International Workshop on Security Measurements and Metrics, Lund, Sweden, 2012:7-10

[121] Li X, Feng B, Li G, Li T, He M. A vulnerability detection system based on fusion of assembly code and source code. Security and Communication Networks, 2021:1-11

[122] Li Y, Wang S, Nguyen TN, Van Nguyen S. Improving bug detection via context-based code representation learning and attention-based neural networks//Proceedings of the ACM on Programming Languages. Athens, Greece, 2019, 3:1-30

[123] Tanwar A, Manikandan H, Sundaresan K, Ganesan P, Chandrasekaran SK, Ravi S. Multi-context attention fusion neural network for software vulnerability identification. arXiv preprint arXiv:2104.09225, 2021: 1-13

[124] Zhang H, Bi Y, Guo H, Sun W, Li J. ISVSF: Intelligent vulnerability detection against Java via sentence-Level pattern exploring.IEEE Systems Journal, 2021: 16(1):1032-1043

[125] Cao S, Sun X, Bo L, Wu R, Li B, Tao C. MVD: Memory-related vulnerability detection based on flow-Sensitive graph neural networks//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh Pennsylvania, USA, 2022: 1456-1468

[126] Ghaffarian S.M., Shahriari H.R..Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. ACM Computing Surveys (CSUR), 2017, 50(4): 1-36

[127] Brown PF, Della Pietra VJ, Desouza PV, Lai JC, Mercer RL. Class-based n-gram models of natural language. Computational linguistics, 1992, 18(4):467-480

[128] Church KW. Word2Vec. Natural Language Engineering, 2017 Jan;23(1):155-62

[129] Alon U, Zilberstein M, Levy O, Yahav E. Code2vec: learning distributed representations of code.//Proceedings of the ACM on Programming Languages.Cascais, Portugal, 2019, 3:1-29

[130] dg, https://github.com/mchalupa/dg

[131] Siow JK, Liu S, Xie X, Meng G, Liu Y. Learning program semantics with code representations: An empirical study.// Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering. Honolulu, USA, 2022: 554-565

[132] Zhao Z, Yang B, Li G, Liu H, Jin Z.Precise learning of source code contextual semantics via hierarchical dependence structure and graph attention networks.Journal of Systems and Software, 2022, 184:111108-111125

[133] Zaharia S, Rebedea T, Trausan-Matu S. Source code vulnerabilities detection using loosely coupled data and control flows//Proceedings of the 2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). Timisoara, Romania.2019:43-46

[134] Binkley D, Moonen L, Isaacman S. Featherweight assisted vulnerability discovery. Information and Software Technology, 2022, 146:106844-106855

[135] Li R, Feng C, Zhang X, Tang C. A lightweight assisted vulnerability discovery method using deep neural networks. IEEE Access, 2019, 7:80079-80092

[136] Omar M. Detecting software vulnerabilities using language models. arXiv:2302.11773. 2023: 1-8

[137] Aladics T, Hegedűs P, Ferenc R. An AST-based code change representation and its performance in just-in-time vulnerability prediction. arXiv:2303.16591. 2023: 1-20

[138] Rabheru R, Hanif H, Maffeis S. DeepTective: detection of PHP vulnerabilities using hybrid graph neural networks// Proceedings of the 36th Annual ACM Symposium on Applied Computing. Virtual Event, 2021: 1687-1690

[139] Rabheru R, Hanif H, Maffeis S. A hybrid graph neural network approach for detecting PHP vulnerabilities//Proceedings of the 2022 IEEE Conference on Dependable and Secure Computing (DSC).Edinburgh, UK, 2022:1-9

[140] Lin C, Xu Y, Fang Y, Liu Z. VulEye: a novel graph neural network vulnerability detection approach for PHP application. Applied Sciences, 2023,13(2):825-847

[141] https://joern.io/

[142] https://www.cppdepend.com/

[143] https://github.com/c2nes/javalang

[144] https://javaparser.org/

[145] https://esprima.org/

[146] https://github.com/Swatinem/esgraph

[147] https://github.com/nikic/PHP-Parser

[148] https://github.com/malteskoruppa/phpjoern

[149] https://github.com/tree-sitter/tree-sitter

[150] Zhu B, Tan H. VuLASTE: long sequence model with abstract syntax tree embedding for vulnerability detection. arXiv: 2302.02345. 2023: 1-7

[151] Votipka D, Stevens R, Redmiles E, Hu J, Mazurek M.Hackers vs. testers: A comparison of software vulnerability discovery processes//Proceedings of the Symposium on Security and Privacy. San Francisco, USA, 2018:374-391

[152] Lin G, Wen S, Han QL, Zhang J, Xiang Y. Software vulnerability detection using deep neural networks: a survey. Proceedings of the IEEE, 2020, 108(10):1825-1848

[153] Wartschinski L, Noller Y, Vogel T, Kehrer T, Grunske L. VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python. Information and Software Technology, 2022,144:106809-106827

[154] Şahin CB. Semantic-based vulnerability detection by functional connectivity of gated graph sequence neural networks. Soft Computing, 2023, 2:1-7

[155] Choi M J, Jeong S, Oh H, et al.End-to-end prediction of buffer overruns from raw source code via neural memory networks//

Proceedings of the 26th International Joint Conference on Artificial Intelligence.Melbourne, Australia, 2017: 1546-1553

[156] Sestili CD, Snavely WS, VanHoudnos NM. Towards security defect prediction with AI. Pittsburgh, USA: Carnegie Mellon University Software Engineering Institute, Technical Report: AD1083878, 2018

[157] Tian J, Zhang J, Liu F.BBregLocator: avulnerability detection system based on bounding box regression//Proceedings of theInternational Conference on Dependable Systems and Networks Workshops (DSN-W). Xinzhu, China, 2021:93-100

[158] Dong Y, Tang Y, Cheng X, Yang Y, Wang S. SedSVD: statement-level software vulnerability detection based on relational graph convolutional network with subgraph embedding. Information and Software Technology, 2023, 158: 107168-107181

[159] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. Advances in neural information processing systems, 2017, 30: 1-11

[160] De Sousa NT, Hasselbring W. JavaBERT: Training a transformer-based model for the Java programming language// Proceedings of the36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW). Melbourne, Australia, 2021: 90-95

[161] Radford A, Narasimhan K, Salimans T, Sutskever I. Improving language understanding by generative pre-training. https://openai.com/research/language-unsupervised. 2018,6,11

[162] Bubeck S, Chandrasekaran V, Eldan R, Gehrke J, Horvitz E, Kamar E, Lee P, Lee YT, Li Y, Lundberg S, Nori H. Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv:2303.12712. 2023:1-155

[163] Cheshkov A, Zadorozhny P, Levichev R. Evaluation of ChatGPT model for vulnerability detection. arXiv preprint arXiv:2304.07232. 2023:1-6

[164] Aljanabi M, Ghazi M, Ali AH, Abed SA. ChatGpt: open possibilities. Iraqi Journal For Computer Science and Mathematics, 2023, 4(1):62-64

[165] Peng H, Mou L, Li G, Liu Y, Zhang L, Jin Z. Building program vector representations for deep learning//Proceedings of the International Conference on Knowledge Science, Engineering and Management. Chongqing, China, 2015:547-553

[166] Yang S, Cheng L, Zeng Y, Lang Z, Zhu H, Shi Z. Asteria: deep learning-based AST-encoding for cross-platform binary code similarity detection//Proceedings of the International Conference on Dependable Systems and Networks (DSN). Taipei, China, 2021:224-236

[167] Wei H, Li M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code//Proceedings of the International Joint Conference on Artificial Intelligence. Melbourne, Australia, 2017: 3034-3040

[168] Zhang J, Wang X, Zhang H, Sun H, Wang K, Liu X. A novel neural source code representation based on abstract syntax tree// Proceedings of the 41st International Conference on Software Engineering (ICSE). Montréal, Canada, 2019: 783-794

[169] Bo D, Wang X, Shi C, Shen H. Beyond low-frequency information in graph convolutional networks//Proceedings of the AAAI Conference on Artificial Intelligence. 2021, 35(5): 3950-3957

[170] Jiang Y, Su X, Treude C, Wang T. Hierarchical semantic-aware neural code representation. Journal of Systems and Software, 2022,191:111355-111375

[171] Wang G, Ying R, Huang J, Leskovec J. Multi-hop attention graph neural network. arXiv:2009.14332.2020: 1-11

[172] Li Q, Han Z, Wu XM. Deeper insights into graph convolutional networks for semi-supervised learning//Proceedings of the AAAI Conference on Artificial Intelligence. New Orleans, USA, 2018, 32(1):1-8

[173] Zou D, Hu Y, Li W, Wu Y, Zhao H, Jin H. mVulPreter: amulti-granularity vulnerability detection system with interpretations. IEEE Transactions on Dependable and Secure Computing, 2022, Aug(01): 1-12

[174] Warnecke A, Arp D, Wressnegger C, Rieck K. Evaluating explanation methods for deep learning in security//Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), Genoa, Italy, 2020:158-174

[175] Zou D, Zhu Y, Xu S, Li Z, Jin H, Ye H. Interpreting deep learning-based vulnerability detector predictions based on heuristic searching. ACM Transactions on Software Engineering and Methodology (TOSEM) - Continuous Special Section: AI and SE, 2021, 30(2):1-31

[176] Mao Y, Li Y, Sun J, Chen Y. Explainable software vulnerability detection based on attention-based bidirectional recurrent neural network//Proceedings of IEEE International Conference on Big Data (Big Data). Atlanta, USA, 2020: 4651-4656

[177] Gu M, Feng H, Sun H, Liu P, Yue Q, et al. Hierarchical attention network for interpretable and fine-Grained vulnerability detection//Proceedings of the IEEE INFOCOM Workshops. 2022: 1-6

[178] Ying Z, Bourgeois D, You J, Zitnik M, Leskovec J. Gnnexplainer: generating explanations for graph neural networks. Advances in Neural Information Processing Systems, 2019, 32: 1-12

[179] Tang G, Zhang L, Yang F, et al. Interpretation of learning-based automatic source code vulnerability detection model using vme. Lecture Notes in Computer Science, 2021, 12817: 275-286

[180] Ben-Nun T, Jakobovits AS, Hoefler T. Neural code comprehension: A learnable representation of code semantics// Proceedings of the Advances in Neural Information Processing Systems.Montreal, Canada, 2018, 31: 3585-3597

[181] Liu Jian, Su Pu-Rui, Yang Min, et al. Software and cyber security—asurvey.Journal of Software, 2018, 29(1): 42-68 (in Chinese)
(刘剑, 苏璞睿, 杨珉等.软件与网络安全研究综述.软件学报, 2018, 29(1): 42-68)

[182] Mirsky Y, Macon G, Brown M, Yagemann C, Pruett M, Downing E, Mertoguno S, Lee W. VulChecker: graph-based

vulnerability localization in source code//Proceedings of the 31st USENIX Security Symposium, Security. Boston, USA. 2022:1-18

[183] Dai H, Dai B, Song L. Discriminative embeddings of latent variable models for structured data//Proceedings of the International Conference on Machine Learning. New York, USA. 2016:2702-2711

[184] Zhang Z, Lei Y, Yan M, Yu Y, Chen J, Wang S, Mao X. Reentrancy vulnerability detection and localization: a deep learning based two-phase approach//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering.Rochester, USA. 2022:1-13

[185] https://www.nist.gov/

[186] http://www.stackoverflow.com/

[187] Rao KN, Reddy C. An efficient software defect analysis using correlation-based oversampling. Arabian Journal for Science and Engineering, 2018, 43(8):4391-411

[188] Shen P, Ding X, Mu X, Xu J. A software defect prediction method based on sampling and integration. Journal of Physics: Conference Series 2021, 1732(1):12002-12009

[189] Liu S, Lin G, Han QL, Wen S, Zhang J, Xiang Y. DeepBalance: deep-learning and fuzzy oversampling for vulnerability detection. IEEE Transactions on Fuzzy Systems, 2019, 28(7):1329-1343

[190] Shu R, Xia T, Williams L, Menzies T. Dazzle: using optimized generative adversarial networks to address security data class imbalance issue//Proceedings of the 19th International Conference on Mining Software Repositories. Pittsburgh, USA. 2022: 144-155

[191] Feng Z, Guo D, Tang D, Duan N, Feng X, Gong M, Shou L, Qin B, Liu T, Jiang D, Zhou M.Codebert: a pre-trained model for programming and natural languages. arXiv:2002.08155.2020 1-12

[192] Mashhadi E, Hemmati H. Applying codebert for automated program repair of java simple bugs//Proceedings of the 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR).Madrid, Spain. 2021,17:505-509

[193] Pan C, Lu M, Xu B. An empirical study on software defect prediction using codebert model. Applied Sciences, 2021, 11 (11):4793-4811

[194] Yuan X, Lin G, Tai Y, Zhang J. Deep neural embedding for software vulnerability discovery: comparison and optimization. Security and Communication Networks, 2022:1-12

[195] Han Z, Li X, Liu H, Xing Z, Feng Z. Deepweak: reasoning common software weaknesses via knowledge graph embedding// Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). Campobasso, Italy.2018:456-466

[196] Qin S, Chow KP. Automatic analysis and reasoning based on vulnerability knowledge graph//Proceedings of the Cyberspace Data and Intelligence, and Cyber-Living, Syndrome, and Health. Beijing, China, 2019:3-19

[197] Bui ND, Yu Y, Jiang L.Self-supervised contrastive learning for code retrieval and summarization via semantic-preserving transformations//Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval.2021:511-521

[198] Li X, Gong Y, Shen Y, Qiu X, Zhang H, Yao B, Qi W, Jiang D, Chen W, Duan N. CodeRetriever: unimodal and bimodal contrastive learning.arXiv:2201.10866.2022: 1-13

SU Xiao-Hong, Ph. D., professor. Her research interests include intelligent software engineering, software vulnerability detection, program analysis and software testing, etc.

ZHENG Wei-Ning, Ph. D. candidate. His research interest is software vulnerability detection.

JIANG Yuan, Ph. D., assistant professor. His research interests include program analysis and its application, code representation learning.

WEI Hong-Wei, Ph. D. candidate. His research interests include software data mining, software knowledge engineering, search-based software engineering, code pattern generation and search.

WAN Jia-Yuan, Ph. D. candidate. His research interests include software vulnerability detection and software testing.

WEI Zi-Yue, M. D. His research interest is smart contract vulnerability detection.

Background

Security and software engineering researchers have become increasingly interested in software vulnerability detection techniques in recent years. In general, software vulnerability detection techniques can be divided into three categories: dynamic analysis techniques, static analysis techniques, and hybrid analysis techniques. Static analysis is currently the fastest growing and most widely used

technology.

In the past decade, with the rapid development of artificial intelligence techniques, researchers have developed numerous methods for detecting vulnerabilities based on machine learning and deep learning. According to the technology deployed and the procedural representation they use, these vulnerability detection methods can be classified into different bases: Sequence, AST, and Graph. There is also a lot of work on detection tools evaluation and methods survey. These studies give us good directions on software vulnerability detection and good comprehensive set of hundreds of articles.

Many researchers are exploring ways to improve the performance and utility of data-driven software vulnerability detection techniques. Unfortunately, the cost of extracting and constructing the dataset and the detection efficiency and accuracy of methods have prevented the widespread use of these techniques in the industry at this time. Therefore, implementing intelligent data-driven vulnerability detection methods that can detect vulnerabilities in real-world scenarios is a hot topic in this field. On the other hand, the ultimate goal of vulnerability detection is always to help developers understand and fix vulnerabilities. Thus implementing fine-grained vulnerability detection and providing interpretable detection results is becoming increasingly important in this field. This paper presents a systematic review of data-driven software vulnerability detection techniques, focusing on methods for extracting and constructing vulnerability datasets, methods for learning program and program representations for vulnerability detection tasks, methods for detecting vulnerabilities based on machine learning and deep learning, and methods for detecting fine-grained interpretable software vulnerabilities. By analyzing existing methods, this paper summarizes current challenges in the field of vulnerability detection, and proposes a new framework for software vulnerability detection that combines hierarchical semantic awareness, multi-granularity vulnerability detection, and assisted vulnerability understanding, and forecasts future research directions and trends.

This research was supported by the National Natural Science Foundation of China (No. 62272132).