软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(11):4906−4952 [doi: 10.13328/j.cnki.jos.007376] [CSTR: 32375.14.jos.007376]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

深度学习驱动的软件漏洞预测: 问题、进展与挑战*

唐家昕, 王 璇, 赖 伟, 路则雨, 郭肇强, 杨已彪, 周毓明

(南京大学 计算机科学与技术系, 江苏 南京 210094)

通信作者: 周毓明, E-mail: zhouyuming@nju.edu.cn

摘 要: 软件漏洞是软件中易于被攻击利用的代码片段, 确保软件不易受到攻击是软件开发中必须重视的安全性需求. 软件漏洞预测是指对软件代码进行分析预测, 从而及时找出潜在的漏洞. 深度学习驱动的软件漏洞预测是近年来一个热门的研究领域, 时间跨度大、研究数目众多、研究成果丰厚. 为梳理相关研究成果、总结研究热点, 对2017–2024年间发表的151篇深度学习驱动的软件漏洞预测相关的文献进行综述, 总结相关文献的研究问题、进展以及遇到的问题与挑战等内容, 为后续研究提供参考.

关键词: 软件漏洞; 深度学习; 预测; 软件质量; 特征表示

中图法分类号: TP311

中文引用格式: 唐家昕, 王璇, 赖伟, 路则雨, 郭肇强, 杨已彪, 周毓明. 深度学习驱动的软件漏洞预测: 问题、进展与挑战. 软件学报, 2025, 36(11): 4906–4952. http://www.jos.org.cn/1000-9825/7376.htm

英文引用格式: Tang JX, Wang X, Lai W, Lu ZY, Guo ZQ, Yang YB, Zhou YM. Deep-learning-driven Software Vulnerability Prediction: Problems, Progress, and Challenges. Ruan Jian Xue Bao/Journal of Software, 2025, 36(11): 4906–4952 (in Chinese). http://www.jos.org.cn/1000-9825/7376.htm

Deep-learning-driven Software Vulnerability Prediction: Problems, Progress, and Challenges

TANG Jia-Xin, WANG Xuan, LAI Wei, LU Ze-Yu, GUO Zhao-Qiang, YANG Yi-Biao, ZHOU Yu-Ming

(Department of Computer Science and Technology, Nanjing University, Nanjing 210094, China)

Abstract: Software vulnerabilities are code segments in software that are prone to exploitation. Ensuring that software is not easily attacked is a crucial security requirement in software development. Software vulnerability prediction involves analyzing and predicting potential vulnerabilities in software code. Deep learning-driven software vulnerability prediction has become a popular research field in recent years, with a long time span, numerous studies, and substantial research achievements. To review relevant research findings and summarize the research hotspots, a survey of 151 studies related to deep learning-driven software vulnerability prediction published between 2017 and 2024 is conducted. It summarizes the research problems, progress, and challenges discussed in the literature, providing a reference for future research.

Key words: software vulnerability; deep learning; prediction; software quality; feature representation

随着时代和科技的进步, 软件在工业生产和日常生活中的重要性不断增加. 确保软件的准确性和正常运行、防止软件受到攻击, 已成为每位软件开发者和维护者必须重视的安全性需求. 软件漏洞是软件中易于被攻击利用的代码片段. 一旦软件受到攻击, 不仅可能引发系统故障、信息丢失、隐私泄露等严重后果, 同时生产者还需承担为易受攻击的系统开发和分发补丁、信用下降、市场价值下跌等困难, 导致巨大的经济损失. 例如, 发生于2017年的Equifax数据泄露事件由黑客利用软件漏洞入侵Equifax内部服务器导致. Equifax是美国3大信用评分报告机构之一, 拥有超过1.4亿美国公民的敏感个人信息. 在这起事件中, Equifax使用的Apache Struts框架存在一个软件漏洞 (CVE-2017-5638), 允许攻击者通过特制的HTTP请求执行任意代码. 由于Equifax在漏洞被报告后未及

* 基金项目: 国家自然科学基金 (62172205)
收稿时间: 2023-08-09; 修改时间: 2024-05-22; 采用时间: 2024-12-20; jos在线出版时间: 2025-05-14
CNKI网络首发时间: 2025-05-15

时应用 Apache 公司提供的安全补丁, 攻击者在 2017 年 5–7 月间利用这一漏洞访问了 Equifax 的内部系统, 并持续进行数据窃取. 这一事件导致了 1.47 亿人的信息泄露, 其中包括姓名、住址、身份证号, 以及护照、信用卡信息等隐私内容. 事件发生后, Equifax 受到了大量的集体诉讼和法律索赔, 最终被处以 7 亿美元的罚款和赔偿. 相关研究表明, 为单个设备修复漏洞的典型成本超过 250 美元; 漏洞的报告会导致生产者的市场价值下降 0.6%[1]. 随着现代软件规模和复杂性的增加, 攻击软件的方法也日益多样化. 手动查找代码中的漏洞是一项相当费时费力的任务. 然而, 通过自动预测软件代码中潜在的漏洞, 并快速准确地确定可能出现漏洞的位置, 可以帮助开发和维护人员将精力集中在最易受攻击的代码部分, 从而节省大量时间, 降低人工和动态检测的成本. 自动化软件漏洞预测技术主要包括统计学习和深度学习两类方法. 传统的统计学习方法通常将一系列代码度量、开发过程度量等人工设计指标作为预测变量来分析漏洞的位置[2]. 这类预测方法的实现难度较低, 可解释性良好. 然而, 由于上述指标与漏洞存在的相关性难以保证, 其预测效果存在较大的不确定性. 因此, 这类方法的实现中, 需要针对指标的有效性进行深入分析, 并不断试探, 才能获得优良的性能.

近年来, 随着机器学习技术的不断发展, 深度学习技术在软件漏洞预测领域开始得到广泛应用, 涌现出一系列深度学习驱动的软件漏洞预测方法. 这些方法利用现有的软件漏洞数据集进行模型训练, 自动学习漏洞代码的特征, 并根据这些特征进行漏洞预测[3]. 自 2017 年以来, 软件漏洞预测领域的研究热度呈现逐年上升趋势, 成为时下的一大热门领域. 本文采用系统文献综述方法, 对 2017–2024 年间发表的 151 篇深度学习驱动的软件漏洞预测相关文献进行综述, 全面梳理和总结软件漏洞预测的研究问题、研究进展以及面临的问题与挑战. 本文的主要贡献如下.

(1) 研究趋势分析: 对 151 篇使用深度学习进行软件预测的研究文献的发表时间、发表趋势、发表场所进行了总结分析.

(2) 研究内容分析: 在编程语言、代码表示形式、代码向量化方式、深度学习技术、漏洞数据集和性能评估指标等方面, 对现有的研究进行了详细分析和归纳, 总结了研究的现状、进展以及存在的问题.

(3) 研究技术总结: 根据所采用方法的特点和效果, 对现有研究中使用的代码表示形式、代码向量化方式、深度学习技术进行了分类汇总, 以便更好地理解各类方法在软件漏洞预测方面的应用情况.

(4) 问题及挑战总结: 对现有的研究中提出的问题及挑战进行了分类汇总, 同时在此基础上分析了深度学习驱动的软件漏洞预测领域未来的重点研究工作, 为进一步推动该领域的发展提供了有价值的参考.

本文的综述结果可能会对软件研究人员和质量保证人员产生积极影响, 特别是对软件漏洞的预测和缓解感兴趣的人群. 他们可以通过本文了解这一领域的最新进展, 从而致力于构建更有效、更稳定、更高效的软件漏洞预测模型. 进一步, 本研究的发现也可能为软件供应商、软件采购人员等关心软件的安全性和质量的人提供有益帮助, 有助于降低软件风险管理的成本.

本文第 1 节简述综述背景和相关工作. 第 2 节详细说明所使用的综述方法. 第 3 节介绍研究所针对的编程语言的情况. 第 4 节深入分析研究所使用的代码表示形式的情况. 第 5 节详细阐述研究所使用的代码向量化方式的情况. 第 6 节描述研究所使用的深度学习技术的情况. 第 7 节提供研究所使用的漏洞数据集的详细情况. 第 8 节回顾研究所使用的预测有效性评估指标. 第 9 节讨论现有研究中遇到的挑战及未来的研究工作. 第 10 节汇总本次综述的研究成果并给出展望. 第 11 节总结全文.

1 背景及相关工作

1.1 软件漏洞预测

软件漏洞预测是一类设计并构建模型, 自动化地对软件代码中可能存在漏洞的位置进行预测的任务. 这项任务的复杂性和挑战性促使人们在该领域进行了持久而深入的研究, 导致了许多针对不同问题、不同编程语言, 采用不同类型研究方法的文献的发表. 尽管各项研究使用了不同的技术和算法, 但常规的软件漏洞预测任务可以总结为以下几个主要步骤: 数据收集、代码转换、特征提取、分类模型训练和性能评估. 图 1 展示了这些主要步骤的流程概览.

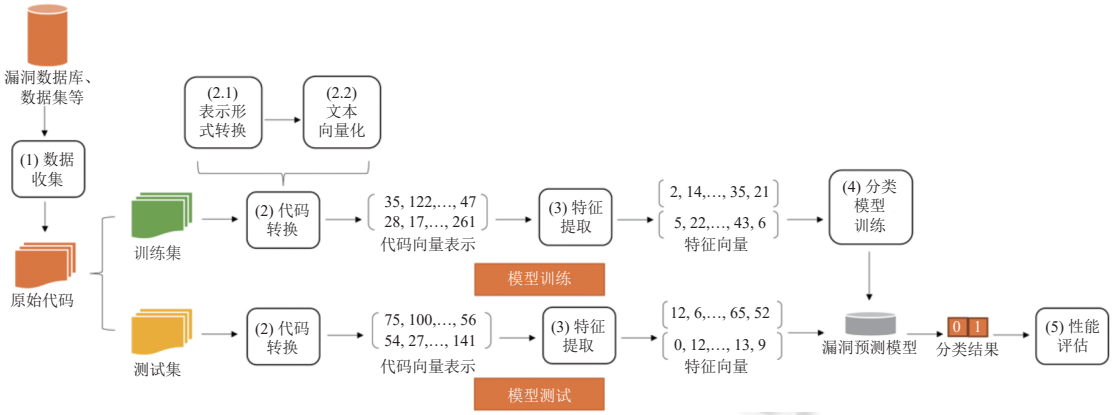


图 1 软件漏洞预测任务的主要流程概览

(1) 数据收集. 需要收集大量包含与不包含漏洞的软件代码及其对应的标签, 作为后续模型训练和模型评估过程所需的漏洞数据集. 数据的来源包括大型漏洞数据库、现有漏洞数据集、代码追踪系统的提交、软件错误追踪系统等. 在数据收集完成后, 数据集会被划分为训练集和测试集两部分. 在此之外, 可以留出一部分样本作为验证集.

(2) 代码转换. 将文本形式的原始代码转换为能够作为神经网络的输入的向量形式, 包括表示形式转换和文本向量化两个步骤.

● 表示形式转换. 将代码文本转换为某种适合于学习特征的表示形式, 例如语法树、控制流图、数据流图、代码切片等. 这一步骤是可选的.

● 文本向量化. 又称代码嵌入. 将代码中的自然语言标识符转换为数值, 从而将文本转换为能够作为特征提取模块的输入的向量形式. 在进行转换之前, 部分方法会将代码中用户定义的变量名及函数名替换为统一的通用名称, 例如将变量名 user_count、函数名 user_search 分别替换为 VAR1、FUN1. 可以使用映射、编码等传统方法进行向量化, 或使用单词转换模型、深度神经网络等转换模型自动学习每个标识符的表示方法. 这一步骤是可选的.

(3) 特征提取. 特征提取模型对代码的表示形式进行学习转换, 形成对应的特征向量. 主要的特征可以分为人工设计特征及表示学习特征两类. 人工设计特征通常包含传统的软件度量指标、文本特征等与代码中漏洞的存在相关的模式, 而表示学习特征则使用神经网络, 实现从代码中自动抽取特征.

(4) 分类模型训练. 分类模型能够对提取到的特征向量进行分类, 从而将代码映射为有漏洞或无漏洞. 常用的分类模型包括逻辑回归、决策树、随机森林等传统分类器, 以及卷积神经网络 (CNN)、长短期记忆神经网络 (LSTM)、图神经网络 (GNN) 等深度学习神经网络. 部分研究中, 表示学习与分类步骤使用同一神经网络进行. 分类模型构建好后, 在训练集与可选的验证集上对模型进行训练.

(5) 性能评估. 对测试集进行 (2)、(3) 中所述的转换操作, 将生成的特征向量送入 (4) 中训练好的模型进行分类, 得到分类结果, 使用性能评估指标对模型获得的性能进行评价. 最常用的指标为精度、召回率、$F1$ score 等分类性能评估指标.

随着现代软件的代码复杂程度的提高、漏洞利用方式的增加, 软件漏洞开始呈现涉及代码行数多、代码行之间依赖关系复杂、隐蔽而不易被发现等特点, 开展软件漏洞预测的重要性日益提升. 同时, 各类深度学习技术已经在多个其他领域内获得了成功应用, 并得到了广泛的认可[4−6]. 自软件漏洞预测领域诞生以来, 研究人员针对这一问题进行了持久而丰富的研究. 除尝试使用传统方法来预测漏洞之外, 还尝试将多种类型的深度学习技术引入这一领域, 同时对研究中遇到的主要挑战进行深入思考与探索, 不断发展和设计更强大、更高效的软件漏洞预测手段. 相关研究包括将经典的错误预测方法应用于漏洞预测[7]、解决数据不平衡问题的算法设计[8]、采用基于图神经网络[9]、注意力神经网络[10]的方法、多种深度学习技术的组合使用[11]、针对跨域漏洞预测进行研究[12]、对漏洞预测模型的可解释性进行研究[13]等. 总体来看, 软件漏洞预测领域的发展日新月异, 新的方法和技术不断涌现,

丰富了该领域的研究与应用.

1.2 相关研究工作

深度学习是机器学习的一个分支, 是一类基于深度神经网络的方法. 将深度学习技术应用于软件漏洞预测任务是一个新兴且备受关注的领域, 已经产生了大量的研究内容和丰富的研究成果. 一般来说, 只要在特征提取或分类步骤中使用了深度学习技术, 就可以将其视为深度学习驱动的软件漏洞预测方法.

为了深入理解各项研究所使用的方法、系统梳理各项研究的成果及遇到的挑战, 许多研究人员对这一领域的相关研究进行了调研及综述. 例如, Singh 等人[14]进行了简短的调研, 探讨了代码表示形式的转化、漏洞数据集的选择以及神经网络的设计等问题. Kubiuk 等人[15]对比分析了基于抽象语法树 (AST) 和程序依赖图 (PDG) 两种代码表示形式的漏洞预测方法, 并确定了最佳的神经网络结构. Chakraborty 等人[16]使用精心设计的现实世界漏洞数据集, 测试了现有性能优良的深度学习漏洞预测方法, 并发现这些方法的性能大幅降低. 为了解决这一问题, 他们提出了自己的漏洞预测模型. Alaoui 等人[17]则调查了深度学习驱动的 Web 应用程序漏洞预测问题, 研究了数据集的生成、漏洞预测模型的设计以及可预测的漏洞类型等关键问题. 而 Zheng 等人[18]回顾了最近 22 项采用深度学习来预测漏洞的研究, 并确定了 4 项对深度学习驱动的漏洞预测领域产生重大影响的研究工作, 将其他研究工作进行了分类. 这些调查研究为深度学习技术在软件漏洞预测领域的应用提供了有价值的见解, 促进了这一领域的发展进步.

上述综述文献描述了近年来深度学习驱动的漏洞预测领域的研究成果, 提出了有价值的发现和建议, 为后续研究指明了方向. 然而, 随着时间的推移, 这一领域正以前所未有的速度蓬勃发展, 并不断涌现新的研究成果. Wartschinski 等人[19]设计了一种基于源代码的文本表示的深度学习漏洞预测工具 VUDENC, 能够从 Python 代码库中自动学习易受攻击的代码特征. Zhang 等人[20]提出了一种基于权重图和深度学习的漏洞预测方法 VDBWGDL, 使用代码变体方法及控制流图切片获取易受攻击的关键字和关键语句. Zhou 等人[21]设计了一种基于代码属性图及图记忆力网络对漏洞函数进行预测的新方法 GraphEye. Cao 等人[22]提出了一种基于流敏感图神经网络 (FS-GNN) 的语句级内存相关漏洞预测方法 MVD, 通过联合嵌入非结构化信息 (即源代码) 和结构化信息 (即控制流和数据流) 来捕捉漏洞模式. Gong 等人[23]从源代码中提取并解析函数原型, 接着将函数原型拆分为具有语义含义的片段进行分类. Kim 等人[24]通过在易受攻击的代码数据集上微调预先训练的 BERT 神经网络, 提出了一种漏洞预测方法 VulDeBERT. Yan 等人[25]构建了一个名为 Juliet+的新漏洞数据集, 利用注意力机制实现了 4 个深度学习模型, 并使用 Hit@k 等评估指标分析其可解释性. Lin 等人[26]利用图神经网络, 提出了一种基于敏感函数及 PDG 切片 (子依赖图, SDG) 的漏洞预测方法 VulEye. Dong 等人[27]构建源代码的代码属性图, 基于中心节点对其进行切片, 构建出它们的子图并进行嵌入, 使用关系图神经网络进行学习. Wu 等人[28]针对二进制代码中的漏洞预测问题进行研究, 基于语言模型嵌入 (ELMo) 和 GRU 神经网络, 设计了一种同时考虑单词级和指令级特征的特征融合漏洞预测模型. 胡雨涛等人[29]对源代码进行规范化及切片, 使用图神经网络及漏洞解释器得到具体的漏洞代码行, 提出了一种切片级别的漏洞预测方法. 2023 年以来, 大型语言模型 (LLM) 爆发式发展, 涌现出一批使用 GPT 等人工智能技术进行软件漏洞预测的研究. 这些新的研究成果丰富了深度学习驱动的软件漏洞预测领域, 为这一领域未来的发展提供了新的方向和可能性.

近年来的各项新研究及其提出的新方法对这一领域的发展进步有着显著的推进作用, 但由于发表时间较晚, 未被各类综述收录. 此外, 现有的综述各有侧重点, 或篇幅较短, 或未能系统、全面地覆盖相关研究的研究问题、研究进展、研究中遇到的挑战等主要内容. 为此, 本文对 2017 年以来这一领域中发表的相关研究进行综述, 重点关注研究所针对的编程语言、研究所使用的代码表示形式、代码向量化方式、深度学习技术、性能评估指标、漏洞数据集, 以及研究中遇到的挑战等问题. 本文系统地总结软件漏洞预测任务的各个主要步骤, 对近年来的新研究进行概括, 从而能够为后续研究提供参考和指导, 帮助人们更好地了解这一领域的最新进展, 并为未来的研究提供明确的方向.

2 综述方法

本次综述包含确定搜索关键词及搜索范围、文献筛选、滚雪球、人工审查这 4 个步骤.

2.1 关键词及搜索范围

自 2017 年以来, 软件漏洞预测领域迅速发展, 针对深度学习驱动的软件漏洞预测这一研究问题的文献大量发表. 本文所调研的文献发表时间范围设定为 2017–2024 年. 在文献选择方面, 我们综合考虑文献数据库所属的领域、影响力、权威性等因素, 优先选取来自美国电气和电子工程师协会 (IEEE)、美国计算机学会 (ACM)、多学科数字出版机构 (MDPI)、Applied Sciences、软件学报、网络与信息安全学报等著名出版社的文献. 为了尽可能涵盖国内外相关研究, 我们使用百度学术、Google Scholar、IEEE Xplore、ACM Digital Library、中国知网、Springer Link 等检索平台对文献进行全面检索. 这样的综合检索策略能够确保获取到最具代表性和价值的文献, 为本文的综述提供可靠的研究依据.

漏洞是一类特殊的软件缺陷, 可被攻击者利用以对软件系统进行攻击, 导致栈溢出、root 权限窃取、数据丢失等问题. 与普通缺陷不同, 针对软件漏洞进行的研究文献应当包含“漏洞”或“有漏洞的”等关键词, 并需排除针对普通缺陷的研究. 漏洞预测任务中, 根据漏洞代码的特征, 将给定的代码分类为易受攻击或不易受攻击, 或定位易受攻击的片段. 因此, “漏洞预测”“漏洞探测”“漏洞发现”等关键词都与该领域相关. 另一方面, 深度学习是机器学习的子领域, 涵盖了各类线性神经网络、图神经网络以及部分机器学习方法, 用于特征提取和分类步骤. 因此, 包含“深度学习”“神经网络”“表示学习”“特征提取”等关键词的文献也可能与该领域相关. 为了综合考虑这两类关键词, 我们将它们相互连接形成中文及英文数据库的检索规则, 以便获取与软件漏洞预测相关的文献.

● 对英文数据库的检索规则: (“vulnerability detection” OR “vulnerability prediction” OR “vulnerability discovery” OR “vulnerability analysis” OR “vulnerability location” OR “vulnerable code” OR “vulnerable function” OR “vulnerable method”) AND (“deep learning” OR “neural network” OR “attention network” OR “representation learning” OR “feature extraction” OR “feature learning”).

● 对中文数据库的搜索规则: (“漏洞预测” OR “漏洞检测” OR “漏洞探测” OR “漏洞发现” OR “漏洞挖掘”) AND (“深度学习” OR “神经网络” OR “卷积网络” OR “特征提取” OR “特征学习” OR “表示学习”).

对于提供不同搜索模式的搜索平台, 我们根据不同平台的功能来调整搜索规则. 首先, 我们尽可能利用高级搜索功能来搜索标题中出现上述关键词的文献. 通过高级搜索, 可以更精准地筛选符合条件的文献, 提高搜索结果的相关性. 其次, 如果搜索平台不支持高级搜索, 或者不支持搜索标题中的关键词, 我们会选择尽可能与上述关键词匹配的文献. 这可能包括文献标题、关键词或者摘要中出现的相关词汇, 以确保尽量涵盖与软件漏洞预测相关的文献. 总体而言, 我们灵活运用不同搜索平台的功能, 以确保获取到与研究领域相关的文献, 并尽可能提高搜索结果的准确性和相关性.

2.2 文献筛选及滚雪球

搜索到的文献与研究领域的相关性受到许多因素的影响. 例如, 关键词“vulnerability”一词在其他领域 (如地理学、社会学等) 中也可能出现, 导致搜索结果包含与软件漏洞预测无关的文献. 此外, 文献中使用的特征提取和分类技术并非都属于深度学习技术. 因此, 搜索到的文献可能存在不符合本次综述内容的情况, 例如文献并非发表于计算机领域的期刊会议、文献的主要研究领域并非软件漏洞预测、文献所提出的主要方法并非使用深度学习技术预测软件漏洞等. 因此, 需要根据文献的出版机构、标题、关键词、摘要等内容进行初步筛选, 以确保文献的相关性和可信度. 综合考虑这些因素, 我们对搜索到的文献进行仔细筛选和评估, 确保只有符合文献搜索范围、与软件漏洞预测领域相关的高质量文献被纳入综述中. 这样的筛选过程有助于确保文献的质量和综述的准确性. 具体地, 我们将文献的纳入排除规则设置如下.

● 纳入规则: (1) 软件工程、信息安全、人工智能等相关领域的文献; (2) 研究内容为软件漏洞预测的文献; (3) 所用方法或所研究内容属于深度学习技术的文献.

● 排除规则: (1) 文献所使用的语言并非中文或英文; (2) 文献无法通过数字方式获取; (3) 文献并非完整版; (4) 文献为学位论文.

在初步筛选得到文献集后, 进行滚雪球操作. 具体做法是从每篇文献所引用的参考文献中选择未被当前文献集包含的文献, 并按照预先设定的纳入排除规则进行筛选. 这样可以获取被上述搜索规则遗漏的相关文献, 并将其

加入到文献集中. 这一滚雪球操作是递归进行的, 即对新纳入文献再次进行引用分析, 获取其引用的参考文献, 并再次进行筛选. 通过这种方式, 可以最大程度地扩充文献集, 确保覆盖尽可能多的相关研究. 滚雪球操作是一种有效的文献获取方法, 它允许我们发现一些潜在的重要文献, 尤其是那些不容易通过常规搜索规则获取到的文献. 通过逐步扩展文献集, 可以获得更全面、更有代表性的研究文献, 为综述提供更可靠的依据.

2.3 人工审查

在进行滚雪球操作后, 得到的文献可能存在一些问题, 例如主要研究内容并非软件漏洞预测、未对软件漏洞预测问题进行深入研究, 或者使用的主要漏洞预测方法不属于深度学习驱动的方法等, 这些文献不符合综述的目标. 为确保综述的准确性和可信度, 我们阅读这些文献并进行人工审查. 在人工审查过程中, 关注并提取文献所使用的代码表示形式、深度学习技术、漏洞数据集、性能评估指标等主要信息. 具体而言, 人工审查过程中我们关注两类信息: 基本信息和其他信息.

基本信息如下.

- 进一步审查文献的研究内容及结论, 根据软件漏洞预测是否为该文献的主要研究内容、文献是否深入研究了软件漏洞预测问题、文献中使用或调查的主要漏洞预测方法是否为深度学习驱动的方法等因素最终确定文献是否应包含在本综述中.
- 文献发表的期刊/会议在《中国计算机学会推荐国际学术会议和期刊目录》(CCF) 中对应的等级.
- 文献所研究的编程语言, 如 Python、Java、C++等.
- 文献所使用的代码表示形式, 如原始代码、抽象语法树、程序控制流图 (CFG) 等.
- 文献所使用的代码向量化方式, 如直接映射、编码、单词转换模型等.
- 文献所使用的深度学习技术, 如卷积神经网络、图神经网络、长短期记忆神经网络等.
- 文献所使用的性能评估指标, 如精度、召回率和 $F1$ score 等.
- 文献所使用的软件漏洞数据集及数据集的来源, 如 NVD、SARD、Juliet test suite 等.
- 综述类文献关注深度学习驱动的软件漏洞预测领域内的哪些问题.

其他信息如下.

- 文献所提出的在研究过程中遇到的问题与挑战、未来值得进行的研究工作.
- 综述类文献对各研究中存在的问题、挑战与未来研究工作的分析总结.

经过人工审查, 共获得了 137 篇英文文献与 14 篇中文文献, 其中包括 14 篇综述类文献 (综述类文献中有 1 篇还提出了自己的方法), 构成了本次综述的文献数据集. 每篇文献都包含上述内容中的一部分基本信息, 并可能包含一些其他信息.

2.4 文献数据集

所有 137 篇英文文献和 14 篇中文文献均以深度学习驱动的软件漏洞预测为主要研究内容. 由于文献出版时间、数据库更新等原因, 搜集到的 2024 年的文献数目较少. 图 2 对发表于 2017–2023 年间的文献进行统计, 展示了这些文献的发表时间, 并说明了它们所发表的期刊/会议在《中国计算机学会推荐国际学术会议和期刊目录》中的分类情况.

从发表时间来看, 发表于 2017 年、2018 年的文献分别仅有 3 篇、7 篇. 从 2019 年开始, 文献的发表数目大幅提升. 2019 年有 12 篇文献发表, 2020 年翻倍, 达到 24 篇. 这可能是由于深度学习技术大幅发展, 加之整个软件工程领域迎来了鼎盛的研究时期, 因而相关内容的研究大量涌现. 2021 年的文献数目相较 2020 年增加 5 篇, 达到 29 篇. 2022 年的文献数目为 27 篇. 2023 年的文献数目为 37 篇, 热度持续增加. 由于文献出版时间、数据库更新等原因, 搜集到的 2024 年的文献数目为 8 篇. 总体而言, 近几年研究人员对这一领域的兴趣呈现迅速上升趋势, 自 2019 年以来一直保持着较高的关注度.

从出版机构来看, 137 篇英文文献中来源较多的出版社包括: IEEE, 61 篇 [3,8,9,11−13,16,18,20,21,24,25,28,30−77]; ACM, 22 篇 [22,49−51,54,62−67,77−87]; 施普林格出版集团 (Springer), 14 篇 [10,14,23,88−98]; 爱思唯尔 (Elsevier), 22 篇 [19,27,99−118]; 多学科数

字出版机构 (MDPI), 5 篇[17,26,119–121]; 约翰威立国际出版集团 (Wiley), 5 篇[122–126]; 欣达维 (Hindawi), 3 篇[127–129]. 其余 16 篇文献[15,130–144]中, 有 3 篇文献[133–135]发表于 NDSS、USENIX Security Symposium 等国际著名顶级期刊及会议. 从 CCF 分类来看, 137 篇英文文献所发表的期刊/会议中被 CCF 推荐的共有 82 篇, 占比约 60%. 其中来自 A 类期刊/会议的文献有 27 篇, B 类有 30 篇, C 类有 25 篇. 发表在 B 类及以上高水平期刊/会议的文献共 57 篇, 占比约 42%, 说明这一领域的各项研究成果得到了学术界的广泛认可.

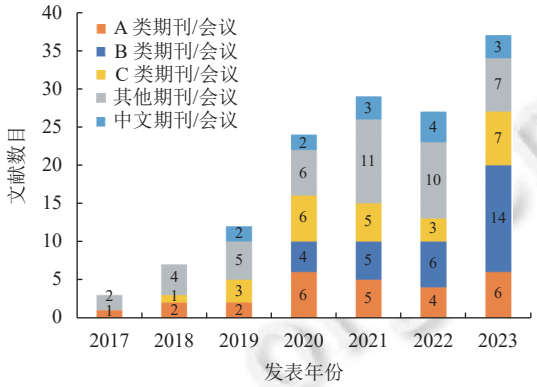


图 2 2017–2023 年间每年发表的文献数目及其所属的期刊/会议的 CCF 分类

14 篇中文文献[29,145–157]分别发表于电子与信息学报[152]、软件学报[29,148]、通信学报[156]、网络与信息安全学报[145,157]、信息安全学报[146]等国内知名高水平期刊. 可以看出, 绝大部分文献都来源于国内外著名的、权威的、有影响力的出版机构, 有着极高的地位和广泛的受众.

表 1 列示了 14 篇综述类文献的主要关注内容, 包括了各文献的发表时间、文献名称、调研范围、主要关注内容, 以及本综述与之相比所具有的独特之处和创新点. 尽管这些文献各自侧重点不同, 但总体上涵盖了代码的表示形式、向量化方法、深度学习技术、性能评估指标以及研究中遇到的问题与挑战等方面. 其中, 部分文献的调研范围包括基于传统机器学习方法的内容, 而本综述的调研范围专注于深度学习驱动的方法. 本综述的研究范围涵盖了 2017–2024 年间共 151 项深度学习驱动的漏洞预测研究, 而之前的综述文献则涉及不同的调研范围和关注点, 与本综述有显著不同.

表 1 各综述类文献主要关注的内容

发表时间	文献	作者	调研范围	主要关注内容	本综述的不同与创新点
2020	[14]	Singh等人	未说明	软件漏洞分析方法、机器学习技术、代码的向量化方式、漏洞数据集、深度学习技术	仅针对深度学习驱动的方法; 包含对编程语言、代码表示形式、问题与挑战等内容的综述
2020	[18]	Zeng等人	22项研究	4项游戏规则改变者、漏洞预测发展趋势、漏洞数据集、代码表示形式、深度学习技术、漏洞粒度、问题与挑战、未来的研究工作	包含更多的文献; 并非围绕部分研究而展开; 包含对编程语言、代码的向量化方式、评估指标等内容的综述
2020	[55]	Lin等人	未说明	软件漏洞的不同视角、机器学习技术、深度学习技术、代码表示形式、代码标记方式、漏洞粒度、挑战与未来研究工作	仅针对深度学习驱动的方法; 包含对编程语言、代码的向量化方式、评估指标等内容的综述
2020	[16]	Chakraborty等人	未说明 (主要针对4项研究)	代码表示形式、深度学习技术、漏洞数据集、模型实际性能、问题与挑战	包含更多的文献; 并非针对现有模型的实际性能; 包含对编程语言、代码的向量化方式、评估指标等内容的综述
2020	[125]	Abdallah等人	2014–2020年的28项研究	常规检测方法、机器学习技术、深度学习技术、漏洞数据集、漏洞分析目标、代码表示形式、问题与挑战、未来的研究工作	包含更多的文献; 仅针对深度学习驱动的方法; 包含对编程语言、代码的向量化方式、评估指标等内容的综述

表 1 各综述类文献主要关注的内容 (续)

发表时间	文献	作者	调研范围	主要关注内容	本综述的不同与创新点
2021	[30]	Alagöz等人	2020–2021年的4项研究	深度学习技术、漏洞数据集、性能评估指标、模型性能	包含更多的文献; 包含对编程语言、代码表示形式、代码的向量化方式等内容的综述
2021	[15]	Kubiuk等人	未说明	代码表示形式、深度学习技术 (主要针对基于AST和PDG的方法)	并非针对特定的代码表示形式; 包含对编程语言、代码的向量化方式、评估指标等内容的综述
2022	[17]	Alaoui等人	2010–2021年的63项研究	研究趋势和研究类型、框架及平台、深度学习技术、漏洞数据集、性能评估指标、网络攻击的类型、模型性能、研究重点、问题与挑战	包含更多的文献; 仅针对漏洞预测, 不包含网络攻击; 包含对编程语言、代码表示形式、代码的向量化方式等内容的综述
2022	[42]	Nong等人	2016–2020年的55项研究	开放科学中的可用性、可执行性、可再现性和可复制性	包含更多的文献; 并非主要针对开放科学领域进行研究; 包含对编程语言、代码表示形式、代码的向量化方式、评估指标等内容的综述
2022	[129]	Wu等人	未说明	代码表示形式、预处理方式、向量化方式、深度学习技术、漏洞数据集、挑战与未来研究工作	包含对编程语言、评估指标等内容的综述
2023	[63]	Steenhoek等人	2018–2022年的多项研究 (主要针对11项研究)	模型性能、漏洞数据集、模型学习到的漏洞特征	主要内容并非复现模型并评估其性能; 包含对编程语言、代码表示形式等内容的综述
2023	[144]	Harzevili等人	2011–2022年的67项研究	研究趋势、漏洞数据集、机器学习及深度学习技术、代码表示形式、向量化方式、漏洞类型、挑战与未来研究工作	包含更多的文献; 仅针对深度学习驱动的方法; 包含对编程语言、评估指标等内容的综述
2023	[98]	Zhu等人	近5年的48项研究	感知差距、漏洞数据集、代码表示形式、深度学习技术、特定场景优化、挑战与未来研究工作	包含更多的文献; 并非针对某一现象而展开; 包含对编程语言、代码的向量化方式等内容的综述
2024	[141]	Ding等人	未说明 (主要针对7项研究)	漏洞数据集、评估方法、模型性能、挑战与未来研究工作	主要内容并非探讨漏洞数据集存在的问题; 包含对编程语言、代码表示形式等内容的综述

后文图 3 按照不同分类方法显示了我们的综述组织各项研究的方式. 其中, 括号中的数字表示该类别包含的文献数目. 在第 3–9 节中, 我们将按照这一组织形式介绍上述文献的主要研究问题、研究进展以及研究所面临的挑战. 具体内容包括研究所针对的编程语言, 研究所使用的代码表示形式、向量化方式、深度学习技术、漏洞数据集、性能评估指标, 以及在研究过程中遇到的问题与挑战. 在第 10 节中, 我们将总结上述内容并探讨未来可能的研究方向.

3 编程语言的选择

几乎每一项涉及软件漏洞预测的研究都会专注于特定的编程语言. 本节将首先综合概述上述研究所涉及的编程语言及各编程语言被研究的次数, 接着从多个角度解释这一现象出现的原因, 最后分析上述研究在选择编程语言时面临的问题.

3.1 研究所针对的编程语言

在提出了自己的方法的 138 篇文献 (137 篇非综述类文献及 1 篇综述类文献) 中, 其中 2 篇[75,136]针对模型训练方法进行改进, 未涉及具体的编程语言; 其中另外 2 篇[34,62]不针对特定的编程语言. 故针对编程语言的研究共涵盖了 134 篇文献. 图 4 显示了针对每种编程语言的文献数目, 其中部分文献涉及多种编程语言. 可以看出, 134 篇文献中有 113 篇 (约 84%) 文献所针对的编程语言为 C/C++语言, 而针对其他语言的研究数目极少. 在其他编程语言中, 数目排在首位的是对汇编语言或二进制代码等非源代码形式进行漏洞预测的文献, 共 10 篇. 排在之后的是常用的应用程序开发语言 Java (9 篇)、常用的 Web 后端开发语言 PHP (6 篇). 针对 Python 语言、Swift 语言进行漏洞预测的文献则非常稀少, 而针对其余常用编程语言, 例如 C#、Go 等重要语言的文献暂未搜集到.

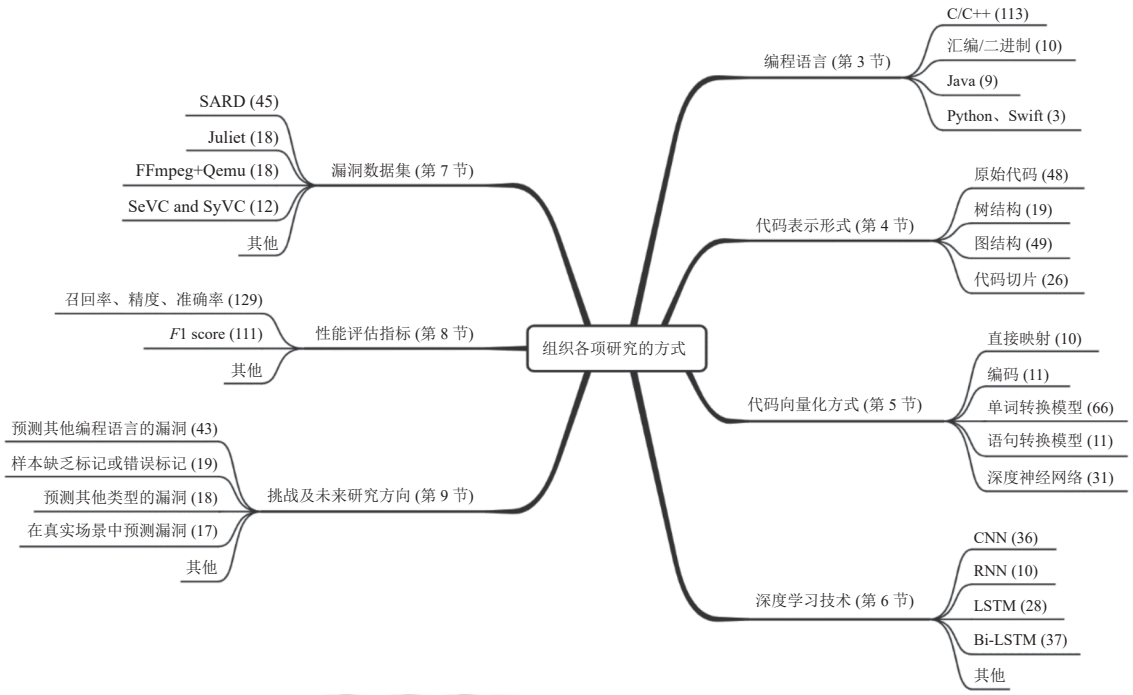


图 3　组织各项研究的方式

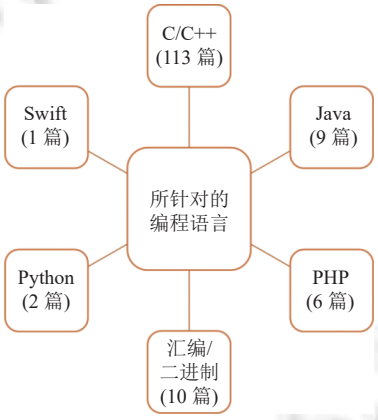


图 4　针对每种编程语言的文献数目

可以从语言特性、实际应用、漏洞分布这几个角度解释为何现有研究大量集中在 C/C++语言上.

从语言特性的角度来看, C/C++语言是偏向底层、抽象与封装相对较少的语言, 涉及大量易于导致漏洞的操作. (1) 整数运算. FFmpeg 是一个多媒体编码/解码库, 其中涉及大量的整数运算. 使用 C/C++语言编写的 FFmpeg 源代码中存在大量数值错误 (CWE-189) 或整数溢出 (CWE-190) 漏洞. 这与 C/C++语言在进行整数存储时存在数值范围限制, 且涉及有符号与无符号整数之间的转换及运算有关. 上述特性涉及复杂的运算规则, 可能出现编程人员无法预期的运算结果, 进而导致各种漏洞的产生. 而 Python 语言中不存在整数符号或范围限制, 从而避免了此类型的漏洞. (2) 数组、指针、动态内存等内存操作. 越界写入漏洞 (CWE-787) 是指程序将数据写入预期缓冲区边界之外, 覆盖相邻的内存区域. 这可能导致程序崩溃、执行任意代码等后果. C/C++语言中, strcpy 函数在进行字符串拷贝时不检查待拷贝字符串长度是否会超出数组边界. 若超出, 则导致越界写入漏洞. 而 Java 语言中存在的数

组边界检查将会阻止程序对数组范围之外的内存进行访问, 从而避免了此类型的漏洞. 空指针解引用 (CWE-476)、未初始化的指针访问 (CWE-824) 等漏洞均由程序对指针的管理不当导致; 重复释放 (CWE-415)、释放后使用 (CWE-416) 等漏洞均由程序对指针或动态内存的管理不当导致. 而 Java、Python 等语言中不存在对指针或动态内存的管理. (3) 进程管理、访问权限控制等内核操作. 此类操作涉及许多与资源、信号、系统状态相关的语句, 可能被攻击者所利用. 争用条件 (CWE-362)、对权限的不当处理 (CWE-280) 等漏洞均由此类操作导致.

从实际应用的角度来看, C/C++语言常常被用于实现相对底层的软件及系统. Linux 操作系统内核、PHP 语言官方解释器、安全通信协议 OpenSSL 等重要系统均由 C/C++语言编写. 若此类系统受到攻击, 可能导致严重的后果. 例如, 著名漏洞 Dirty COW (CVE-2016-5 195) 中, Linux 内核对写时复制 (copy-on-write, COW) 机制的处理不当, 使得攻击者能够利用一个竞争条件, 对系统中共享的只读页进行修改, 进而导致系统完全被攻击者所控制. 相比其余编程语言, 由 C/C++语言编写的大量底层代码中存在的漏洞往往导致更加严重的安全问题, 影响更为深远.

从漏洞分布的角度来看, 相比 Java、Python 等编程语言, C/C++语言编写的应用程序可能具有更高的漏洞数目及漏洞密度. 漏洞密度表示软件源代码中包含漏洞的数目与源代码行数的比例. 2021 年, Nikitopoulos 等人[158]收集了超过 40 种编程语言的应用程序源代码. 统计结果显示, 数据集中最常见的 5 种漏洞类型中, 4 种漏洞类型大量出现在 C 语言编写的应用程序中, 含量远多于 Python、PHP 等编程语言. 同时, 包含最多漏洞的 10 个应用程序中, 超过半数的应用程序由 C/C++语言编写. 同年, Venson 等人[159]对源代码中的安全相关代码 (PSC) 比例与漏洞密度之间的关系的研究表明, 在特定来源的源代码中, C++语言相比 Java 和 Python 语言具有更高的 PSC 比例, 从而具有更高的漏洞密度.

3.2 选择编程语言时存在的问题

本节汇总了上述文献在深度学习驱动的软件漏洞预测问题的研究中所针对的编程语言. 通过分析我们发现, 当前研究中编程语言的选择存在如下问题.

(1) 缺乏在其他主流编程语言上进行的研究. 现有的各项研究[3,38,83]所使用的编译器 Clang、代码分析器 Cppcheck、代码解析器 Joern 等工具主要针对 C/C++语言, 漏洞数据集的主要编程语言也是 C/C++. 在性能评估时, 常使用的基线模型也是基于 C/C++语言设计的. 许多主流编程语言缺乏经典而高性能的工具及漏洞预测基线模型. 例如, 现有的针对 Java 语言的研究使用软件指标、神经网络等作为基线方法[41], 或不使用基线方法[25]. 总体来看, 其他流行的编程语言所开发的项目同样需要进行漏洞预测, 但针对这些编程语言的研究数量很少. 如 Java、Python、Swift 等主流编程语言几乎没有相关研究, 而其他编程语言如 Go、C#、Ruby 等则完全空缺.

(2) 现有方法在其他编程语言上的泛化能力未知. 各编程语言具有独特的语法结构和应用场景, 不同编程语言中包含的漏洞代码可能具有完全不同的格式. 这限制了特定漏洞预测模型在其他编程语言上的泛化能力. 因此, 目前在 C/C++语言训练集上训练得到的模型是否能够在其他编程语言的测试集上取得优良性能仍然是未知的[32]. 即使重新进行训练, 这些模型是否适用依然是巨大的挑战. 这启示我们应当更多关注那些流行但未被深入研究的编程语言, 加强数据集收集、代码表示形式设计以及基线方法开发, 以提高使用这些编程语言编写的软件的质量.

(3) 针对汇编语言或二进制代码进行的漏洞预测需要得到更多重视. 我们能够获得的许多软件, 例如各类商业应用程序通常不是开源的[92], 源代码获取困难, 或者存在编程语言不流行、代码编写质量不佳等问题. 因此, 应当寻找能够便捷而准确地预测非源代码中存在的漏洞的方法. 然而, 汇编语言或二进制代码往往复杂, 指令间存在复杂的逻辑关系, 并且涉及与底层系统和硬件交互的问题. 从二进制代码中提取语义特征往往困难[42], 针对源代码的方法难以应用于二进制代码的漏洞预测. 同时, 构建汇编语言或二进制代码的漏洞数据集的难度较大. 对代码片段进行编译及反汇编的过程复杂[28], 对汇编语言或二进制代码进行准确标记也不易. 因此, 非源代码的漏洞预测问题需要采用全新的方法和思路、进行更深入的探索来解决.

4 代码的表示形式

由于常用的卷积神经网络、循环神经网络等深度学习模型通常只接受向量形式的输入, 故若希望从代码中提

取特征, 往往需要对原始代码进行一系列转换. 通常这一转换包括表示形式转换与文本向量化两个步骤. 在表示形式转换中, 代码被转换为某一包含数据流或控制流关系的、易于提取特征的形式. 在文本向量化中, 上述形式被编码为向量, 从而能够被送入神经网络. 本节首先汇总介绍上述研究所选择的代码表示形式, 接着分析这些文献在选择代码表示形式时存在的问题.

4.1 研究所使用的代码表示形式

在上述 138 篇文献中, 有 2 篇文献[78,91]提出的方法主要针对特征选择, 不涉及代码表示形式的转换; 另外还有 2 篇文献[75,136]着重于模型训练方法的改进, 未涉及具体的代码表示形式. 因此, 针对代码表示形式的研究共涵盖了 134 篇文献. 表 2 中展示了这些文献所使用的代码表示形式, 其中包括方法所属的类别、名称、对方法的说明以及方法的使用情况.

表 2 各项研究使用的代码表示形式汇总

类别	名称	说明	使用情况
原始代码	原始代码	直接对汇编指令、源代码片段、函数名称等进行指标计算、向量化等后续处理	[3,11,19,20,23,28,32,33,41,46,50,51,54,57–59,66,68,69,71,72,76,82,84,86,88–90,92–96,102,121,124–126,128,130,132,140,142,143,149,150,152,155]
树结构	抽象语法树 (AST)	由源代码进行一定的处理得到的树状中间形式	[8,10,12,13,34–36,44,45,91,97,107,109,112,113,122,125,131,146]
图结构	控制流图 (CFG)	包含代码中语句之间的控制流关系的图结构, 跨函数构建的CFG称为过程间控制流图 (ICFG)	[10,47,48,64,87,105,107,109,113,116,145,154]
	数据流图 (DFG)	包含代码中语句之间的数据流关系的图结构	[65,107,113,134,138]
	数据依赖图 (DDG)	包含代码中语句之间的数据依赖关系的图结构	[25]
	程序依赖图 (PDG)	包含代码中语句之间的数据依赖和控制依赖关系的图结构	[22,26,29,49,77,80,85,109,118,151,157]
	代码属性图 (CPG)	由AST、CFG、PDG组合而成的图结构	[16,21,27,43,56,60,73,103,108,115,127,137,139,148,153]
	代码复合图 (CCG)	由AST、CFG、DFG组合而成的图结构	[99]
	切片属性图 (SPG)	由DDG、CDG、函数调用依赖图 (FCDG)组合而成的图结构	[61]
	切片依赖图 (SDG)	由带函数调用的PDG切片而得的图结构	[117]
	特征依赖图 (FDG)	由AST、CDG、DDG组合而成的图结构	[156]
	系统依赖图 (SDG)	基于PDG和函数调用关系构造的图结构	[117,147]
	语义漏洞图 (SVG)	包含控制、数据、执行流边等多种边的图结构	[74]
	异常流图 (EFG)	包含与漏洞相关的关键程序元素及其依赖关系的图结构	[67]
	循环流抽象语法树 (LFAST)	由AST、CFG、DFG组合并进行简化而成的图结构	[106]
	ePDG	包含LLVM IR之间的数据流和控制流关系的图结构	[135]
代码切片	基于语义的源代码切片	基于关键点构建, 由一系列在数据流及控制流上相互关联的源代码行、线性IR行、汇编代码行或伪代码行组成	[9,24,31,35,37,39,52,53,69,70,76,79,101,104,114,120,125,133,142,146]
	基于语义的线性中间代码 (IR)切片		[38,83,119]
	基于语义的汇编代码切片		[100,110]
	基于语义的伪代码切片		[111]

不同代码表示形式包含不同的特征及信息. 我们按照所属类别对上述各表示形式进行介绍.

● 原始代码. 最直接的代码表示方法为不进行转换, 直接使用原始代码文本片段进行后续处理工作. 为了细化漏洞预测的粒度, 尽量去除无关代码, 可以选取更精细的片段进行特征学习, 例如选取代码中所有函数的名称、使用词法分析及语法分析提取出所有的方法等. 此类直接使用代码片段进行漏洞预测的方法是最简单、最便捷、处

理速度最快的方法.

● 树结构. 对源代码进行语法分析后得到的 AST 也可以用于进行漏洞预测. AST 使用树状结构表示源代码所具有的抽象语法结构, 通过树中的节点和边信息来描述源代码中包含的内容及结构, 能够指示源代码中各个标识符间具有的层次嵌套关系. AST 所具有的非线性结构使得其难以直接进行向量化, 通常需要使用遍历的方式将其转换为线性的文本形式. 常用的遍历方式包括深度优先遍历[35]、前序遍历[44]等. AST 与具体的文法、语言的细节无关, 屏蔽了源代码的编程风格、自然语言等可能会导致模型过拟合的信息. 同时其与机器无关, 避免了汇编代码中需要进行的底层操作, 从而适合于进行特征学习.

● 图结构. 图是一类包含节点及边的网状结构, 能够表示元素之间具有的关系. 基于图结构的代码表示形式包含许多信息, 适合用于漏洞预测. 图中的节点可以为源代码、线性 IR 或汇编代码片段, 节点之间的边则表示代码片段之间所具有的数据流、控制流关系. 常用的图结构包括 CFG、DFG、PDG、CPG 等. 另外有文献定义新的图结构. 例如 CCG、SPG、SDG 等. 一般地, 基于图结构的代码表示形式按照图节点进行向量化后, 使用图神经网络进行学习, 图中的边以邻接表、邻接矩阵等形式作为图神经网络的输入. 各类常用的图结构均与代码中的数据流、控制流等信息相关, 能够尽可能保留代码的结构特征, 更好地表达各代码行之间的关系, 从而常常能够拥有较好的漏洞预测性能.

● 代码切片. 基于图结构的代码切片是一种综合考虑了线性及图结构而得到的表示形式. 许多漏洞的产生与程序中特定类型的关键操作相关, 例如算术运算、指针操作、内存分配等. 此类操作通常和一系列与其存在依赖关系的语句共同作为漏洞的特征. 因此, 可以将包含关键操作的代码行作为中心, 构建各类图结构, 将与关键操作存在数据流、控制流关系的代码行包含在图中. 接着, 从原始代码中选取包含在图中的代码行, 从而得到一系列相互关联的代码行组成的代码切片. 将其视作包含敏感操作的代码片段, 进行后续处理工作. 此类方法最常使用到的图结构包括 PDG、CFG、DFG、CPG 等. 基于代码切片的方法能够最大程度减少无关代码的影响, 将漏洞代码具有的语法结构完整呈现给神经网络.

4.2 代码表示形式的使用现状

图 5 显示了使用每种类型的代码表示形式的文献数目, 其中部分文献使用了多种表示形式, 或将表示形式进行组合. 可以看出, 原始代码是使用数目最多的表示形式. 这可能与这一方法简单易行、无需复杂的数据处理步骤、能够利用简单的线性神经网络等因素有关. 与之类似的是基于 AST 的方法. 有 67 篇 (50%) 文献使用原始代码或 AST 进行漏洞预测.

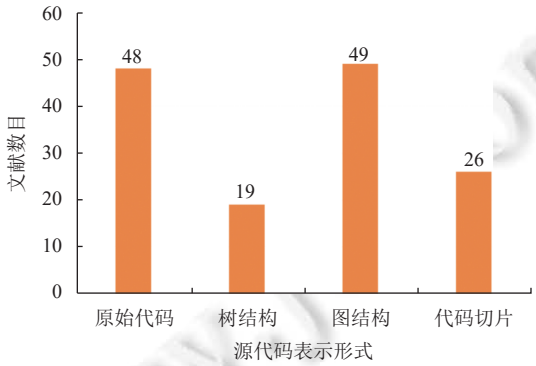


图 5 使用每种类型的表示形式的文献数目

截至目前, 上述搜索到的文献中已有 49 篇文献采用了基于图结构的方法, 占比约为 37%. 最早的使用基于图结构的方法的文献可以追溯到 2019 年. 图 6 显示了 2019–2023 年间使用基于图结构的方法的文献数量, 以及这些文献在各自年份发表的所有文献中所占的比例. 从图中可以观察到, 在 2021 年之前, 使用基于图结构的方法的文献很少. 但是自 2021 年起, 这类文献的数量和占比急剧上升, 直至 2023 年的 37 篇文献中有 21 篇采用了基于图结构的方法, 占比约为 57%. 可以看出, 近年来, 研究人员对基于图结构的软件漏洞预测方法越来越感兴趣. 这可能是

由于在大规模复杂数据集上进行的研究证明了基于图结构的表示形式与图神经网络相结合的方法能够获得出色的性能. 基于图结构的表示形式能够更好地捕捉代码中变量之间深层次的关联, 从而更有效地提取与漏洞相关的特征. 特别是在源代码中存在大量变量和复杂的控制结构的情况下, 需要使用设计精巧的图结构, 提取控制流和数据流信息, 才能对漏洞特征进行准确捕捉. 因此, 在软件漏洞预测领域中, 基于图结构的方法需要持续得到关注和重视.

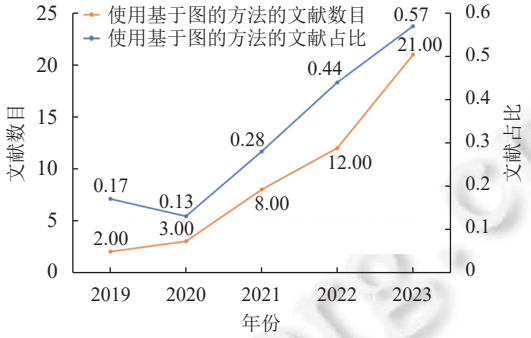


图 6 2019–2023 年间使用基于图的方法的文献数目及占比

4.3 选择代码表示形式时存在的问题

本节汇总了上述文献在深度学习驱动的软件漏洞预测问题的研究中所使用的代码表示形式. 通过分析我们发现, 当前研究中代码表示形式的选择存在如下问题.

(1) 基于标识符的表示形式存在关系缺失问题. 原始代码等基于标识符的表示形式常常会缺乏标识符之间的数据流及控制流关系[16]. 代码中标识符之间具有的数据流、控制流依赖关系在漏洞预测中常起到重要的作用, 许多漏洞基于这些关系而产生. 例如, 在缓冲区溢出漏洞中, 待分配内存大小的计算语句可能与内存分配语句相隔甚远, 但前者计算得到的数值会通过数据流流入后者. 因此, 此类漏洞必须通过数据依赖关系才能捕捉. 简单的表示方法, 例如仅能显示出代码的语法层次结构的语法树, 难以捕捉到此类漏洞所具有的内在特征, 从而导致相对不佳的预测结果[19].

(2) 对非线性结构的表示形式的遍历存在信息丢失问题. 对 AST、CFG 等非线性表示形式进行的遍历会丢失其中含有的分层结构信息. Zeng 等人[18]在研究中发现, 许多项研究中, 对 AST 和 CFG 等结构进行的遍历结束后, 所有节点被按照神经网络难以理解的线性次序排列. 这导致图中原有节点之间的层次结构、标识符间的连接关系、数据流及控制流等信息变得难以捕捉, 甚至丢失. 因此, 应当尽可能地使用图神经网络对非线性表示形式进行学习, 避免对其进行遍历操作.

(3) 许多表示形式包含与漏洞存在无关的信息. Wu 等人[129]在研究中发现, 源代码中常常包含许多与漏洞的存在无关的语句, 会影响模型对漏洞特征的学习; Zheng 等人[61]在研究中发现, AST 等结构的表示中包含许多与漏洞的存在无关的节点和边. 这对各类代码表示形式的有效性提出了挑战, 启示研究人员设计方法对其进行评估.

(4) 许多大型的复杂漏洞必须综合考虑复合代码语义才能进行预测. 现实世界中的软件漏洞往往复杂, 需要使用信息量大、表达能力强的代码表示形式, 综合考虑控制流、数据流、支配关系等众多因素进行推理. 使用简单的预测方法可能难以获得优良的性能. 例如, Zhou 等人[134]在研究中发现, 单独使用 AST 只能找到不安全的参数, 将 AST 与 CFG 相结合可以得到更好的结果; Wang 等人[40]在研究中发现, 简单地组合关系图所获得的效果较差; Cheng 等人[48]在研究中发现, 考虑数据依赖而忽略控制依赖的方法存在缺陷. 因此, 许多方法在现实世界的代码上所获得的准确性可能会不及预期.

5 代码的向量化方式

将代码表示形式转换为能够作为神经网络的输入的向量形式的过程被称为代码向量化. 部分文献[13]直接将

文本形式作为神经网络的输入, 将向量化作为神经网络学习的一部分进行联合训练; 另有部分文献[82,84,93]从代码文本中提取一系列文本特征或软件指标, 从而能够直接进行分类. 除此之外, 各类代码表示形式通常需要专门经历向量化步骤. 部分文献在进行向量化前, 还将代码中用户定义的标识符替换为统一的通用名称. 本节首先总结现有研究中使用的代码向量化方式, 接着分析现有研究在选择向量化方式时存在的问题.

5.1 研究所使用的代码向量化方式

在上述 138 篇文献中, 有 2 篇文献[78,91]提出的方法主要关注于特征选择, 没有涉及代码向量化. 另外还有 2 篇文献[75,136]着重于改进模型训练方法, 也未涉及具体的代码向量化方式. 因此, 针对代码向量化方式的研究共涵盖了 134 篇文献. 表 3 列出了这 134 篇文献所使用的代码向量化方法, 包括方法所属的类别、名称、对方法的说明以及方法的使用情况.

表 3 各项研究所使用的代码向量化方式汇总

类别	名称	说明	使用情况
映射	直接映射	直接将每个单词转换为其对应的数值	[8,12,28,34,35,47,58,72,104,122]
编码	one-hot编码	向量的某一位表示当前语句中是否包含这一单词	[3,33,78,117,152]
	特征张量编码	根据CPG的节点之间是否满足某种关系进行编码	[148]
	VecG	将CPG中的4个属性进行one-hot编码	[56]
	VecCPG	将CPG中的5个属性进行one-hot编码	[21]
单词转换模型	Word2Vec	词向量之间的距离与单词的语义相关	[8–12,16,19,25,27,29,31,35–40,43–45,52,53,60,61,73,79,83,88–90,95–97,99–104,106,108,110–116,120,125,127,128,130,132–134,137,139,145,146,149,153,155–157]
	GloVe	词向量之间的距离与单词的共同出现情况相关	[85,88,101,128,130]
	FastText	对Word2Vec中函数进行灵活使用, 提高了性能	[88,96,101,128,130]
语句转换模型	Sent2Vec	基于FastText, 能够对整个语句进行编码	[77,118]
	Doc2Vec	基于Word2Vec, 能够对文档中的每条语句进行编码, 识别文档的唯一性	[22,26,48,80,95,101,147,151]
	Node2Vec	一种用于图结构的节点嵌入方法, 使得嵌入空间中距离较近的节点在图中也具有相似的特性	[67]
深度神经网络	Transformer	一种基于注意力的神经网络, 能更好地捕捉长序列中的依赖关系	[107,138]
	BERT	将Transformer模型中的编码器进行组合而得到, 能够生成双向的语言表示	[68,74,95,131]
	CodeBERT	BERT神经网络的扩展, 能够捕捉自然语言与编程语言的语义联系	[49,50,62,66,69,70,105,126,128]
	ELMo	基于深层上下文, 能够生成单词的不同表示形式	[28,121]

在上述向量化方式中, 代码被看作是由一系列语句组成的文本, 而标识符则被视为单词. 各类方法通过建立单词或语句的映射关系, 将它们转换为向量形式. 我们按照所属类别对上述各向量化方式进行介绍.

● 直接映射. 最简单的向量化方式是直接将标识符映射到对应的数值表示. 可以通过单词查找表来实现映射. 例如, 建立一个单词查找表, 将每个出现的单词加入表中, 并按照它们在表中的索引将其转换为对应的数值或向量表示. 部分研究[34]在建表时还会考虑单词所属的类别, 以进一步优化向量化方法.

● 编码. 编码是一种广泛使用的向量化方法, 能够将文本、属性、关系等信息转换为数值形式. 在这类方法中, 代码语句或图中的单词、属性等内容按照规则被编码为数值. 有许多文献设计了自己的编码规则. 最简单的编码方式是 one-hot 编码. 在 one-hot 编码中, 向量的长度为语料库中的单词数目, 向量的某个位置表示当前语句中是否包含对应的单词. 由于 one-hot 编码可能需要巨大的向量长度, 各项研究几乎不直接使用 one-hot 编码, 而是在其基

础上设计自己的编码方法. 特征张量编码是一种应用于 CPG 的方法, 由段旭等人[148]于 2020 年提出. 在特征张量编码中, CPG 中的节点按照特定关系被编码为三维特征张量, 然后通过形状校正、线性变换等操作压缩映射为固定长度的特征向量. VecG 也是一种应用于 CPG 的编码方式, 由 Zhang 等人[56]于 2021 年提出. 在 VecG 中, CPG 中每个节点的标签、函数、常量和变量类型这 4 个属性被 one-hot 编码为固定长度的特征向量. VecCPG 是另一种应用于 CPG 的编码方式, 由 Zhou 等人[21]于 2021 年提出. 在 VecCPG 中, CPG 中每个节点的标签、运算符、API 函数调用、常量和变量类型这 5 个属性被 one-hot 编码为固定长度的特征向量.

● 单词转换模型. 随着深度神经网络技术的发展, 各类单词转换模型被提出, 并得到了广泛应用. 这些模型基于神经网络设计, 考虑单词之间的位置关系, 并通过大量语料的训练来学习单词的语义. 其中, 一系列高效、高性能的单词转换模型已经被研究人员提出, 包括 Word2Vec、GloVe、FastText 等. Word2Vec 模型是最常用的转换模型之一, 由 Mikolov 等人[160]于 2013 年提出. 它基于分布假设, 通过滑动窗口查看当前单词的周边单词, 不断更新当前单词的词向量. Word2Vec 模型能够保留代码的语义信息, 使得词向量之间的距离越近, 对应词的语义就越接近. GloVe 模型是一种基于单词共同出现的概率的转换模型, 由美国斯坦福大学的研究小组于 2014 年发表[161]. GloVe 模型能够反映整个语料库的统计信息, 兼顾全局信息和局部信息, 生成的词向量能够更好地捕捉全局语义关系, 在多个自然语言处理问题上取得了优良的性能. FastText 模型是 Facebook 公司于 2017 年发表的一种快速文本转换模型, 是对 Word2Vec 模型的一种扩展. FastText 模型将单词分解为字符序列进行处理, 考虑单词出现的先后次序, 提高了模型在语料库不足情况下的处理能力, 在保证效率的同时大幅缩减了训练时间[162,163].

● 语句转换模型. 除对单词进行编码外, 直接对语句、文档或图中节点进行编码的转换模型也得到了广泛的应用. Sent2Vec 模型是一种考虑单词及短语的嵌入形式的句子转换模型[164]. Sent2Vec 模型将滑动窗口大小扩展到整个句子, 通过将句子中的词向量进行组合, 生成高质量的句子表示形式. Doc2Vec 模型是一种基于 Word2Vec 的文档转换模型[165]. Doc2Vec 模型将整个句子或文档转换为一个固定长度的向量, 从而捕捉文档的语义信息. 相比于简单地拼接由单个标识符转换得到的向量, Doc2Vec 模型能够接受不同长度的句子作为训练样本, 从而提供更精确的代码语句表示. Node2Vec 模型是一种基于 Word2Vec 模型的图节点嵌入模型, 由 Grover 等人[166]于 2016 年提出. Node2Vec 模型通过将图中的节点嵌入到低维向量空间中, 使得在向量空间中距离较近的节点在图中也具有相似的特性.

● 深度神经网络. 部分研究直接将原始代码作为神经网络的输入, 将代码向量化作为神经网络特征学习的一部分进行联合训练. 除此之外, 一些具有单词转换功能的神经网络模型也适用于代码向量化. Transformer 模型是一种同时关注单词及其位置的序列模型, 由 Vaswani 等人[167]于 2017 年提出. Transformer 模型能够动态关注输入的不同部分并捕捉到其中存在的重要信息, 从而具有更优秀的单词转换能力. BERT 模型是 Transformer 模型中编码器的组合, 由 Devlin 等人[168]于 2018 年发表. BERT 模型能够生成双向的语言表征, 相比 Transformer 模型具有更好的编码能力. CodeBERT 模型由微软于 2020 年开发, 是一种 BERT 神经网络的扩展. 它是一个双峰模型, 输入中同时包含源代码和自然语言[169]. CodeBERT 模型能够捕捉自然语言与编程语言的语义联系, 获得整个函数及各语句的向量化表示. ELMo 模型是 Peters 等人[170]于 2018 年发表的一种上下文敏感的词嵌入模型. 不同于 Word2Vec 等传统模型, ELMo 模型基于双向语言模型, 能够根据同一单词在不同上下文中的使用情况及含义, 动态地生成其不同的表示形式.

5.2 选择代码向量化方式时存在的问题

图 7 展示了不同类型代码向量化方式在文献中的使用情况, 其中部分文献使用了多种代码向量化方式. 可以观察到, 共有 66 篇 (约 49%) 文献使用了单词转换模型进行向量化, 远超过使用其他向量化方式的文献数量. 其中, 有 65 篇文献使用了 Word2Vec 模型. 总体来看, Word2Vec 模型是这一领域中的文献最常用的代码向量化方式. 使用 FastText 模型及 GloVe 模型的文献均仅有 5 篇, 且几乎在所有使用 FastText 模型及 GloVe 模型的文献中, 都将它们与 Word2Vec 模型进行对比分析. 文本转换模型是最主要的代码向量化方式, 有 75 篇 (约 56%) 文献使用了单

词或语句转换模型进行转换. 接下来是深度神经网络驱动的方法, 有 31 篇 (约 23%) 文献使用深度神经网络进行向量化, 其中有 14 篇将向量化与特征提取步骤相结合进行联合训练. 有 11 篇文献使用了各种类型的编码方法, 使用直接映射及编码方法的文献共有 21 篇, 占比约为 16%.

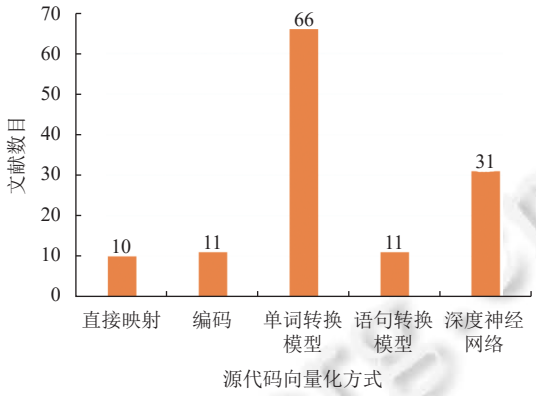


图 7 使用每种类型的向量化方式的文献数目

通过分析我们发现, 当前的研究中代码向量化方式的选择存在如下问题.

(1) 最常用的 Word2Vec 模型存在性能缺陷. Word2Vec 模型无法很好地捕捉目标单词与其上下文之间的相互依赖关系, 且难以反映整个语料库中单词的共同出现信息. 此外, 对于出现次数较少的单词, Word2Vec 模型的准确性较低, 可能会错过一些重要的语言关键字. 多项研究[101,128,130]已经表明, 在软件漏洞预测任务中, 许多情况下 FastText、CodeBERT 等模型具有比 Word2Vec 模型更高的准确性. 然而, FastText、CodeBERT 等模型尚未得到广泛使用. 因此, 有必要针对 Word2Vec 模型的弱点进行优化改进, 同时对 FastText 等其他单词转换模型进行更广泛、更深入的研究. 这可以提高代码向量化的准确性和效果, 改进漏洞预测任务的性能.

(2) 映射及编码方法忽略了有价值的信息. 在常用的映射及编码方法中, 多数方法假设单词之间相互独立, 忽略了单词的出现次序、单词之间的关系等有价值的信息. 然而, 这可能影响所得向量的质量. 例如, 许多漏洞的产生与特定类别语句及其出现次序相关, 若仅指示出语句中含有的标识符, 易导致包含与不包含漏洞的代码段所对应的向量相似, 难以分辨. Wartschinski 等人[19]在研究中发现, 简单的代码表示形式及向量化方式在过去获得了平庸的性能. 这对许多采用直接映射或编码的研究提出了挑战.

(3) 填充和截断向量的操作将导致信息丢失. 对于许多向量化方式来说, 标识符与数值或词向量是一一对应的关系. 由于每条语句包含的标识符数目不同, 不同长度的语句所生成的向量长度也会不同. 一种常用的解决方案为设定一个合适的标准长度, 超过这一长度的向量进行截断, 不足这一长度的向量在末尾进行填充. Zheng 等人[104]在研究中发现, 相比无需进行向量的截断填充操作的 CountVectorizer 工具, 进行这一操作的 Word2Vec 模型会导致性能下降, 这可能是由于上述向量截断填充操作导致了信息丢失; Yuan 等人[128]在研究中发现, 填充操作带来的与漏洞无关的信息会干扰模型对漏洞特征的判断. 这一问题对各类向量化方法的有效性提出了挑战.

(4) 对词汇表外单词的处理存在信息丢失问题. Wei 等人[121]在研究中发现, 未出现在训练集中的单词, 称为词汇表外 (OOV) 单词, 在使用 Word2Vec 进行向量化时, 会被省略或替换为特殊标识符. 这一操作会导致信息丢失. 不同来源的源代码中可能包含完全不同的单词. 若希望模型能够学习到健壮的、概括性好的源代码表示, 必须重视模型处理 OOV 单词的能力.

(5) 代码中的函数及变量名称可能产生噪音, 但也包含了一定的语义信息. 将代码中的标识符名称替换为标准名称能够防止模型学习无关特征, 但也可能会导致语义信息丢失. 总体来看, 各项研究对如何处理代码中的自然语言信息未达成一致. 部分研究使用自然语言信息作为特征的一部分, 部分研究则将其作为噪音进行删除处理.

Semasaba 等人[123]认为, 使用函数名称来表示函数功能是各项研究的限制之一. 而 Zheng 等人[104]则在研究中发现, 进行函数及变量名称替换工作对提高模型的漏洞预测性能没有帮助. 可以看出, 如何合理利用代码中的自然语言信息是一个值得研究的问题.

6 深度学习技术

随着深度学习技术的发展, 越来越多的深度神经网络开始被用于软件漏洞预测领域. 这些神经网络能够从原始代码对应的向量中寻找规律, 按照是否包含漏洞对它们进行分类. 本节首先汇总介绍现有研究所使用的深度学习技术, 接着分析这些研究在深度学习技术方面的使用现状, 最后分析现有研究选择深度学习技术时存在的问题.

6.1 研究所使用的深度学习技术

在上述 138 篇文献中, 有 1 篇文献采用了特征选择的方法来增强漏洞预测的可解释性[78], 另外还有 2 篇文献对模型训练方法进行了改进[75,136], 3 篇文献均未涉及具体的深度学习技术. 因此, 针对深度学习技术的研究共涵盖了 135 篇文献. 表 4 详细列出了这些文献所使用的深度学习技术, 包括方法所属的类别、名称、对方法的说明、方法通常用于处理的预测粒度以及方法的使用情况.

从神经网络的结构特点与功能来看, 各项研究所使用的神经网络大致可以分为卷积神经网络、循环神经网络、随机神经网络、注意力神经网络、双向神经网络以及图神经网络这几大类. 我们按照所属类别对上述各深度学习技术进行介绍.

表 4 各项研究所使用的深度学习技术汇总

类别	名称	说明	常用预测粒度	使用情况
卷积神经网络	卷积神经网络 (CNN)	使用卷积操作, 能够捕捉到处于不同位置的特征	代码片段、函数	[3,9,11,13,23,31,34,37,54,58,59,70,72,77,84,88,90,93–96,102,104,107,110,115,118–120,124,125,131,146,149,150,152]
循环神经网络	循环神经网络 (RNN)	具有反馈结构, 具有记忆输入上下文的能力		[3,20,23,37,54,59,81,83,100,119]
	长短期记忆神经网络 (LSTM)	对RNN的一种改进, 能够分析更大更复杂的文本		[9,13,19,31,32,41,46,51,57–59,67,83,85,88,90,91,94,96, 100,101,117,120,124,130,132,142,146]
	门控循环网络 (GRU)	一种简化版的LSTM, 具有更低的计算代价		[9,13,20,31,47,64,83,88,90,91,96,100,101,104,106,108,128,130,146,149]
双向循环神经网络	双向循环神经网络 (Bi-RNN)	输入序列分别以正序和逆序输入至两个循环神经网络进行特征提取, 能够保留过去和未来两方面的信息		[92,100,154]
	双向长短期记忆神经网络 (Bi-LSTM)			[8–13,25,31,33,35–39,46,52,79,81,86,88–90,95,96,100,103,104,111,120–122,125,130,133,146,148,155]
	双向门控循环网络 (Bi-GRU)			[25,28,31,38,44,45,52,79,85,88,90,100,102,110,125,130,146]
随机神经网络	随机神经网络 (SNN)	包含概率因素, 具有跳出局部最小寻找全局最小的能力	代码片段	[93]
	深度置信网络 (DBN)	基于受限玻尔兹曼机, 能够有效表征复杂数据结构		[37]
注意力神经网络	层次注意力神经网络 (HAN)	具有层次结构, 在单词和语句级别应用不同级别的注意力机制	代码片段	[25,53]
	Transformer	使用注意力机制, 具有更强大的并行计算能力及处理长序列的能力	代码行、代码片段	[52,60,112]
	BERT	基于Transformer, 采用了其中的编码器机制, 具有强大的自然语言处理能力	代码行、代码片段、函数	[20,24,46,50,68,95,126,142]
	GPT	基于Transformer, 通过自回归方式生产文本, 能够进行语言理解、翻译、问答等多种任务		[65,66,69,71,76,109,140,142,143]

表 4 各项研究所使用的深度学习技术汇总 (续)

类别	名称	说明	常用预测粒度	使用情况
图神经网络	图卷积神经网络 (GCN)	能够传递并聚合各个节点的特征, 能够学习到高质量的节点特征及节点之间的关联特征	代码行、代码片段、函数	[21,26,47,48,62,67,74,80,85,87,105,106,116,139,145]
	关系图卷积神经网络 (R-GCN)	对GCN的扩展, 能够处理包含了节点及边所属类型的图结构		[25,27,61,153]
	门控图神经网络 (GGNN)	存在数据的遗忘与保留机制, 具有更强的在图中长期传播信息的能力		[16,29,40,60,62,64,97,103,106,108,113,134,137]
	图注意力网络 (GAT)	在节点和边之间高效地并行化进行操作, 在处理复杂的图结构时具有更好的灵活性及表达能力		[21,43,49,56,73,80,147,154]
	双向图神经网络 (Bi-GNN)	具有前向和后向传播的能力, 在每个节点的表示中包含了更多上下文信息	函数	[99]

● 卷积神经网络. CNN 是一种具有广泛应用的经典神经网络模型, 是深度学习领域的基本模型之一. CNN 拥有多个卷积核, 能够从不同角度理解数据样本, 并通过卷积层进行特征提取. 非全连接的结构使得 CNN 能够专注于捕捉局部特征, 并在更高层次上对这些特征进行整合, 以获取全局信息. 同时, CNN 具备平移不变性, 能够捕捉数据样本中任意位置的特征. CNN 在文本分类、图像处理等领域获得了良好的效果. 在应用于软件漏洞预测问题时, CNN 具有的线性结构使得其适合于处理基于标识符的表示形式. CNN 能够学习标识符序列中潜在的漏洞细节, 并将这些细节进行组合, 从而找出代码中潜在的漏洞位置. 在预测粒度方面, CNN 能够处理从代码片段到文件的各种粒度. 其中, 文件粒度过于粗糙, 可能导致较大的人工查找工作量, 故常用的预测粒度为代码片段或函数. 由于 CNN 无法记忆输入数据, 故随着粒度的增大, CNN 可能遗忘之前的输入, 从而导致其性能受到影响. 另外, CNN 存在输入长度限制, 过长的代码序列作为输入时可能会被截断, 这也会对其性能产生影响. CNN 的简洁性及有效性使得其成为软件漏洞预测领域最常用的神经网络模型之一, 且其常用于与其他类型的神经网络进行组合, 以提高预测能力.

● 循环神经网络类. 主要包括 RNN、LSTM 和 GRU. RNN 是一种带有反馈结构的神经网络模型, 在每个时刻都包含一个状态. RNN 的当前状态不仅依赖于当前的输入, 还取决于之前的状态. 这使得 RNN 具备记忆输入数据的能力, 能够根据已有的数据来预测接下来可能出现的数据信息. LSTM 在 RNN 的基础上引入了输入门、输出门和遗忘门的结构. 通过控制这些门的开闭状态, LSTM 可以选择性地读取输入数据并遗忘之前的信息, 提取到当前状态所需的信息. 因此, 相比 RNN, LSTM 能够分析更长、更复杂的文本数据. GRU 是 LSTM 的一种简化版本, 仅包含复位门和更新门这两个门结构, 其内部状态向量和输出向量被合并为状态向量. 相比 LSTM, GRU 拥有更高的计算效率. 在应用于软件漏洞预测问题时, 各循环神经网络具有的记忆能力使其能够有效地捕捉标识符序列中包含的信息, 例如标识符的先后顺序、特定标识符是否出现等, 从而更加有效地学习潜在的漏洞细节. 在源代码中, 与漏洞的产生相关联的数据流可能跨越多条语句. 因此, 捕捉到在位置上间隔较远的语句之间的关系, 即远程依赖关系, 是进行漏洞预测的关键之一. 相比 RNN, LSTM 与 GRU 在这一问题上具有更出色的性能, 因而得到了更加广泛的应用. 在预测粒度方面, 各循环神经网络均能够处理从代码片段到文件的各种粒度. 相比 CNN, 其更加擅长处理长而复杂的文本数据, 性能更加稳定, 受到预测粒度的影响更小.

● 双向循环神经网络类. 主要包括 Bi-RNN、Bi-LSTM 和 Bi-GRU. 双向循环神经网络由两个相同的独立循环神经网络模块组成. 输入序列分别以正序和逆序输入至两个神经网络进行特征提取, 两个神经网络的输出组合得到该序列的最终特征表示. 在应用于软件漏洞预测问题时, 双向循环神经网络在任意时刻可以同时保留当前输入的过去和未来两方面的信息, 从而能够捕捉到双向的语义关系, 深入理解各单词的语义, 具有比单向神经网络更好的分析代码标识符序列之间的关系的能力. Bi-LSTM 具有强大的捕捉长期依赖关系的能力, 是软件漏洞预测领域内应用最为广泛的神经网络模型之一. 在预测粒度方面, 各双向循环神经网络均能够处理从代码片段到文件的各种粒度, 且处理长文本的能力相比循环神经网络有进一步的提升.

● 随机神经网络类. 主要包括 SNN 和 DBN. SNN 是一种模拟能量的特性而构造出的特殊神经网络. 传统神经网络中, 随着能量函数梯度下降, 网络单调地沿能量递减方向演化, 容易陷入局部最小值而无法收敛于全局最优解. SNN 中则包含概率因素, 网络总体上以较大概率向能量函数减小的方向演化, 但也能够在一定程度上暂时接受能量函数增大的结果, 从而更有可能跳出局部最小值. DBN 是一种由多层受限玻尔兹曼机堆叠而成的神经网络, 具有能够捕捉到高层次抽象特征、训练时更加稳定高效、能够有效表征复杂数据结构等优势. 目前随机神经网络在软件漏洞预测问题上的应用较为罕见. SNN 用于绑定两类不同的代码特征, DBN 则用于对代码片段进行分类. 两者的预测粒度均为代码片段.

● 注意力神经网络类. 神经网络的注意力机制是一种将模型的关注点集中到一部分输入或特征的能力, 允许模型动态地关注输入的特定部分, 从而能够更加有效地完成任务. 在应用于软件漏洞预测问题时, 带注意力的神经网络能够识别同一标识符在不同上下文中的重要性, 从而具有更优良的分类性能; 能够更好地识别对最终分类结果有影响的标识符或语句, 从而具有更好的可解释性. 在预测粒度方面, 注意力神经网络具有的聚焦能力使得其擅长在较细的粒度进行漏洞定位, 故其常用于进行代码片段或代码行级漏洞预测. 注意力神经网络主要包括 HAN、Transformer、BERT 和 GPT.

(1) HAN 是一种层次结构的神经网络, 由 Yang 等人[171]于 2016 年提出. HAN 包含单词级别和语句级别的注意力层, 在单词和语句级别分别应用注意力机制, 从而能够在构建文档的表示时区分出重要的内容. 在应用于软件漏洞预测问题时, 通过提取语句中更重要的标识符并进行信息聚合, HAN 可以学习到更细致的漏洞模式, 从而具有更高的准确性, 同时更加擅长于处理细小的预测粒度. 目前 HAN 在软件漏洞预测问题上的应用较为罕见, 用于对图中各个节点进行特征提取, 或对代码切片进行特征提取及分类.

(2) Transformer 是一种基于注意力机制的序列神经网络. Transformer 考虑输入单词之间的相关性, 将注意力应用于多组不同的输入矩阵, 能够学习到相同单词在不同上下文中的表示形式. 同时, Transformer 具有位置编码机制, 输入序列中的每个单词所对应的向量均显式包含其位置信息. 在应用于软件漏洞预测问题时, 相比 LSTM、GRU 等神经网络, Transformer 能够保持单词之间的距离关系, 从而更好地确定单词所具有的语义, 使得提取到的特征更加准确; 能够更容易地捕捉到远程依赖关系, 从而具有更强的处理长标识符序列的能力. 目前 Transformer 在软件漏洞预测问题上的应用较为罕见, 用于对代码片段进行特征提取及分类, 或在代码行级别进行漏洞定位.

(3) BERT 是一种基于 Transformer 的双向神经网络, 采用了其中的编码器机制. BERT 由多层 Transformer 编码器堆叠而成, 能够生成双向的语言表征, 具有强大的自然语言处理能力. 在应用于软件漏洞预测问题时, BERT 经过微调后即可使用, 且其相比 Transformer 具有更好的编码及特征提取能力. BERT 在软件漏洞预测问题上的应用多于 HAN 与 Transformer, 适用于代码片段或代码行级漏洞预测.

(4) GPT 是一种由 OpenAI 公司设计, 基于 Transformer 的生成式预训练模型. GPT 在经过大量文本数据的预训练后, 能够通过自回归的方式, 不断根据已生成的词汇对下一个可能出现的词汇进行预测, 从而生成高质量的自然语言文本. GPT 能够进行语言理解、翻译、问答等多种任务, 是目前在自然语言处理问题上性能最佳的模型. 在应用于软件漏洞预测问题时, GPT 能够根据提示对代码中是否存在漏洞进行判断, 以及对可能存在漏洞的语句进行定位.

● 图神经网络类. GNN 是一类专门用于处理图结构的神经网络, 能够接收格式化的图形数据 (包括节点信息、边信息、属性信息等) 作为输入, 在相互连接的节点之间传播信息, 聚合各个节点的特征, 从而学习到节点或整个图的表示形式. 在应用于软件漏洞预测问题时, GNN 能够对 DFG、CFG、CPG 等图结构进行学习. 在预测粒度方面, 由于各类图结构通常在代码片段或函数的基础上构建, GNN 通常被用于进行代码片段或函数级别的漏洞预测. 通过将漏洞定位到图中的节点, GNN 也能够进行代码行级漏洞预测. 上述文献中, 将图结构作为代码表示形式的文献均使用了某种结构的图神经网络, 主要包括 GCN、R-GCN、GGNN、GAT 和 Bi-GNN.

(1) GCN 是最常用的图神经网络模型, 由 Kipf 等人[172]于 2016 年提出. GCN 通常由多个卷积层堆叠而成, 每个卷积层对节点特征进行卷积操作, 通过卷积层的逐步聚合能够实现非邻接节点间的信息传递. 在应用于软件漏洞预测问题时, GCN 能够学习到高质量的节点特征及节点之间的关联特征, 适合于对图结构的代码表示形式进行

分类.

(2) R-GCN 是 Schlichtkrull 等人[173]于 2018 年提出的一种基于 GCN 的图神经网络模型. R-GCN 是对 GCN 的扩展, 主要用于处理包含多种类型的节点和边的复杂图结构. R-GCN 引入不同类型的关系, 根据关系对节点特征进行卷积操作, 能够捕捉到不同关系对节点特征的影响. 在应用于软件漏洞预测问题时, 可以将语句节点及边所属的类型作为标签嵌入构建的图结构中, 使用 R-GCN 进行处理. 这一操作能够补充与漏洞的存在相关的潜在信息, 从而有望获得更好的性能.

(3) GGNN 是 Li 等人[174]于 2016 年提出的一种基于 GRU 的图神经网络模型. GGNN 在 GCN 的基础上引入了门控机制以及时间步更新算法, 能够选择性地遗忘及记忆信息, 从而更好地捕捉图中节点间的关系, 更加有效地处理包含大量节点和边的复杂图结构. 在应用于软件漏洞预测问题时, 相比 GCN, GGNN 具有更加强大的表征能力及长期依赖关系处理能力, 更适合于学习图结构中包含的语义关系, 在大型复杂函数构建而成的图结构上有着更好的处理效果.

(4) GAT 是 Veličković等人[175]于 2018 年提出的一种基于注意力机制的图神经网络模型. GAT 能够为每个节点及其邻居节点分配不同的权重, 从而描述节点的重要程度, 对不同邻居节点的信息进行差异化处理. 在应用于软件漏洞预测问题时, 与 GGNN、GCN 等图神经网络模型相比, GAT 能够关注到与漏洞存在更加相关的节点, 能够在节点和边之间高效地并行化进行操作, 在处理复杂图结构时具有更好的灵活性及表达能力. 与注意力神经网络类似, 注意力权重对 GAT 的可解释性也有一定的帮助.

(5) Bi-GNN 是 Cao 等人[99]于 2021 年提出的一种基于 GGNN 的双向神经网络模型. Bi-GNN 结合了前向和后向的信息传播, 从而在每个节点的表示中包含了更多上下文信息. 目前 Bi-GNN 在软件漏洞预测问题上的应用较为罕见, 用于从 CCG 中提取并聚合节点特征.

6.2 深度学习技术的使用现状

图 8 显示了使用次数较多的深度神经网络被各文献使用的次数, 其中部分文献使用了多种神经网络, 或将神经网络进行组合. 可以看出, 大部分文献使用线性神经网络进行特征学习及分类. 在线性神经网络中, CNN 及 Bi-LSTM 是两种使用最广泛的神经网络模型, 分别被 36 篇、37 篇 (约 27%) 文献所使用. 这可能是由于各类线性深度神经网络在图像分类、目标检测、网络入侵检测等领域内取得了良好的效果, 且适合于处理由单词组成的文本, 因此有希望在基于文本的漏洞预测方法中得到应用. 由于存在漏洞的代码行可能受到位于其之前或之后的代码的影响, 故基于双向神经网络的方法的使用较为普遍. 使用各类图神经网络的文献共 44 篇, 占比约 33%, 且近年来得到了越来越多的重视. 各类结构精巧的图神经网络在学习到节点间的依赖关系后, 能够准确地基于一系列相互连接的节点预测到漏洞的存在. 基于图神经网络的方法的性能常常优于基于线性神经网络的方法. 因此, 基于图神经网络的方法需要得到持续的关注及重视. 使用 GPT 的文献共 9 篇, 可以看出研究人员正在努力将 GPT 应用于软件漏洞预测领域中.

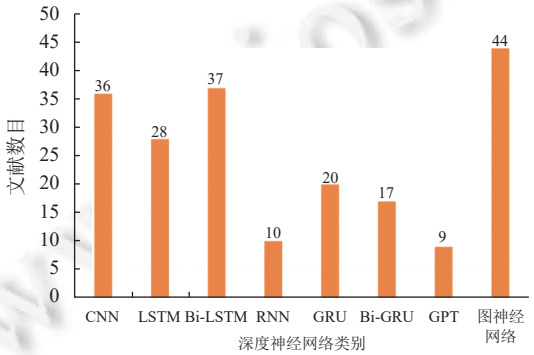


图 8 各项研究使用次数较多的深度神经网络

总体来看, 各项研究尚未在使用深度学习技术方面达成一致. Feng 等人[44]使用了基于 GRU 的方法, 他们认为, 与 LSTM 相比, GRU 能够取得相同的效果, 但运行速度更快. Wei 等人[121]则认为, 与 RNN 和 GRU 相比, LSTM 在跟踪长期依赖性方面表现更好, 因此他们使用了基于 LSTM 的方法. 而 Aumpansub 等人[79]在研究中发现, Bi-GRU 具有更高的准确性、精度和特异性, 在许多指标上要优于 Bi-LSTM, 拥有更强的漏洞预测能力. 这启示我们对各类深度学习技术进行更多的研究与评估.

6.3 选择深度学习技术时存在的问题

本节汇总了上述文献在深度学习驱动的软件漏洞预测的研究中所使用的深度学习技术. 通过分析我们发现, 当前的研究中深度学习技术的选择存在如下问题.

(1) 线性神经网络本质上学习的是与漏洞的存在不相关的特征. Chakraborty 等人[16]使用 Lemma 工具对一系列基于标识符的方法所学习到的特征进行重要性识别, 发现在许多情况下此类方法所认为的特征与存在的实际漏洞无关. 也就是说, 简单的线性神经网络可能忽略了内在的、重要的代码特征, 无法真正理解代码中漏洞产生的原因.

(2) 神经网络的能力存在一定限制. Wang 等人[40]在研究中发现, 常用神经网络的预测能力受到模型的深度和宽度的限制. 现实世界中的漏洞可能包含数目繁多的代码行和复杂的控制流结构, 需要使用更深更宽的神经网络以及更多的训练数据. 而大型神经网络的训练需要耗费大量的时间和资源, 且复杂的神经网络结构易于出现过拟合现象. Chakraborty 等人[16]在研究中发现, 对于图神经网络来说, 训练其需要的代价远高于训练各类线性神经网络的代价, 并且其在资源受限的场合中表现不佳.

(3) 现有的神经网络模型忽略了一些有用的源代码信息. Zhang 等人[56]在研究中发现, Bi-LSTM、GCN 等神经网络仅限于提取特征向量, 忽略了节点信息、节点间关系等有用的源代码信息. 事实上, 虽然这些经典的神经网络结构在图像识别、自然语言处理等领域具有良好的性能, 但代码的结构与图像、自然语言文本等信息完全不同. 常用的神经网络并不是为漏洞预测任务服务的, 难以保证准确地捕捉到源代码中指示漏洞存在的信息. Alaoui 等人[17]认为, 将应用程序安全方面与机器学习方面的专业知识联系起来, 设计出专门进行软件漏洞预测任务的深度神经网络, 是一个有趣的研究问题.

7 漏洞数据集

在深度学习模型构建完成后, 必要的步骤之一是在漏洞数据集上对其进行有效性评估. 为此, 研究人员建立了一系列包含代码漏洞的网站和数据集. 这些资源包含了大量来自真实软件的易受攻击代码片段, 或人工生成的特定格式代码片段, 不仅易于提取和处理, 还能够被转换为不同格式. 软件漏洞预测任务的最终目的是使用训练得到的模型发现真实项目中的潜在漏洞. 部分研究除在测试集上进行评估外, 还选取来自真实项目的无标签软件代码作为检测对象, 判断模型能否从中发现未知漏洞. 在本节中, 我们首先介绍上述研究所创建和使用的漏洞数据集, 接着总结各研究选择的真实检测对象及检测结果, 最后分析这些文献在选择漏洞数据集时所面临的问题.

7.1 研究所使用的漏洞数据集

不同研究所使用的数据集差异较大. 许多研究使用代码追踪系统、软件错误追踪系统 (如 NVD、GitHub、CWE 数据库) 中的漏洞代码, 或来自大型开源项目 (如 Linux kernel、FFmpeg) 的源代码. 部分研究基于上述资源构建了自己的数据集, 或使用先前的研究所使用的经典数据集. 表 5 详细展示了这些研究所构建和使用的漏洞数据集, 其中包括数据集的名称、发表时间、数据来源、所涉及的编程语言、数据集的粒度、数据集的来源文献、数据集被使用的次数, 以及数据集的使用情况. 其中, 原始文献来源是指创建该数据集的文献, 或该数据集的官方链接, 而使用情况则记录了除来源文献外直接采用该数据集的文献. 对于使用次数和使用情况栏位, 若为“—”, 则表示除来源文献外, 没有发现直接使用该数据集的文献.

不同漏洞数据集包含不同编程语言的代码. 我们按照数据集中包含编程语言的情况对数据集进行分类, 在每种类别内按照发布时间对数据集进行排序, 并按照这一次序对上述各数据集进行介绍.

表 5 各项研究所构建及使用的漏洞数据集汇总

名称	发表年份	数据来源	语言	粒度	来源文献	使用次数	使用情况
Software assurance reference dataset (SARD)	2005	含有150多种不同漏洞类型的合成代码和真实软件代码	C/C++、Java、PHP、C#	函数	[176]	46	[9,13,21,22,24,26,27,31,36–40,44–48,57,68,69,77,80,83,89,100,101,103,104,106,110,117–119,128,131,132,142,145–148,151,155–157]
Juliet test suite (J-DS)	2010	含有118种不同漏洞类型的合成代码	C/C++、Java、PHP、C#	函数	[177]	18	[3,25,28,44,51,54,56,60,72,84,87,88,102,111,130,135,137,143]
Firefox	2013	Firefox	C/C++	文件	[7]	1	[41]
POSTER	2017	FFmpeg、LibTIFF、LibPNG	C/C++	函数	[81]	2	[8,125]
Draper VDISC (R-DS)	2018	Juliet test suite、Linux Debian、GitHub	C/C++	函数	[3]	8	[34,54,72,84,113,125,137,138]
VulDeePecker (NDSS18)	2018	Linux kernel、Firefox等19种流行开源产品	C/C++	文件、代码片段	[133]	12	[35,75,76,78,93,96,113,114,120,122,125,152]
s-bAbI	2018	含有缓冲区溢出漏洞的合成代码	C/C++	文件	[178]	1	[137]
Transferable representation learning	2018	LibTIFF、LibPNG、FFmpeg等6个开源项目	C/C++	函数	[12]	7	[8,10,36,92,113,121,126]
Android and Firefox	2018	Android study dataset、Firefox	C/C++、Java	文件	[41]	—	—
NDSS18 binary dataset (ICLR 2019)	2018	NDSS18数据集删除重复函数并编译	汇编/二进制	函数	[179]	5	[28,87,92,102,154]
FFmpeg+Qemu (Devign)	2019	Linux kernel、QEMU、Wireshark、FFmpeg	C/C++	函数	[134]	18	[43,61,68,70,73,74,85,107—109,125,136,138,139,142,153,154,156]
SQLI-LABS	2019	SQLI-LABS	PHP	文件	[132]	—	—
Big-Vul	2020	Chrome、Linux等348种产品	C/C++	函数	[180]	12	[29,49,50,64,70,71,73,85,95,109,127,139]
Nine-projects	2020	Asterisk、FFmpeg等9个开源项目	C/C++	函数、文件	[88]	2	[128,130]
Code metrics dataset	2020	对来自SySeVR数据集的代码进行指标计算	C/C++	指标	[32]	1	[59]
FUNDED	2020	NVD、SARD、GitHub	C、Java、PHP、Swift	提交	[40]	1	[43]
SeVC and SyVC dataset (SySeVR)	2021	Linux kernel、Firefox等19种流行开源产品	C/C++	文件、代码片段	[37]	12	[32,52,53,61,69,78,79,94,124,125,142,150]
GitHub archive dataset (GH-DS)	2021	GitHub事件	C/C++	函数	[54]	—	—
D2A	2021	FFmpeg、OpenSSL、LibTIFF等多种开源产品	C/C++	函数	[181]	7	[43,60,68,74,116,127,142]
CodeXGLUE	2021	Devign数据集进行随机拆分	C/C++	函数	[182]	2	[62,105]
Reveal	2021	Linux Debian kernel、Chromium	C/C++	函数	[16]	10	[43,68,70,73,74,85,108,109,139,156]
DeepWukong	2021	SARD、lua、redis	C/C++	函数、代码片段	[80]	1	[115]
CVEfixes	2021	NVD、CVE	多种语言	文件、函数、提交	[183]	3	[76,117,143]

表 5 各项研究所构建及使用的漏洞数据集汇总 (续)

名称	发表年份	数据来源	语言	粒度	来源文献	使用次数	使用情况
SARD preprocessed	2021	SARD进行预处理	C/C++	函数	[46]	—	—
Draper processed	2021	Draper数据集进行数据平衡	C/C++	函数	[84]	—	—
ICLR19	2021	ICLR 2019删除重复样本	汇编/二进制	函数	[102]	—	—
PHP security vulnerability dataset	2021	phpMyAdmin、Moodle、Drupal	PHP	指标	[91]	—	—
LKSC+	2022	Linux kernel、GitHub、StackOverflow	C/C++	函数	[23]	—	—
Juliet+	2022	Juliet test suite进行重新标记	C/C++、Java、PHP、C#	函数	[25]	—	—
VulCNN	2022	NVD、SARD	C/C++	函数	[77]	1	[115]
MVD	2022	SARD、CVE	C/C++	代码片段	[22]	1	[74]
VUDENC	2022	GitHub	Python	提交、代码片段	[19]	—	—
DeepVD	2023	CVE	C/C++	函数	[67]	—	—
VulF	2023	GitHub、NVD	C/C++	函数	[74]	—	—
TrVD	2024	SARD	C/C++	函数	[112]	—	—

(1) 多语言程序漏洞数据集 (两种及以上语言)

SARD 是 2005 年发布的一个大型数据集, 是最著名、使用最广泛的软件漏洞数据集, 包含由 C/C++、Java、PHP 及 C#编写的合成代码, 以及从真实世界中收集到的开源软件源代码片段[176]. SARD 包含近 20 万个测试用例, 涵盖超过 150 种类型的软件漏洞. 大量现有的漏洞数据集是由 SARD 中的漏洞代码进行预处理、切片等工作构建得到的. SARD 是被使用最为广泛的数据集, 共被 45 篇文献所使用.

J-DS 是美国国家安全局的安全软件中心于 2010 年发布的一个大型漏洞数据集[177], 包含人工合成的约 5.7 万个 C/C++测试程序和 2.4 万个 Java 测试程序, 涵盖 118 种类型的软件漏洞. J-DS 中的每个示例都包含易受攻击版本和非易受攻击版本的代码. J-DS 的被使用次数仅次于 SARD, 被 18 篇文献所使用.

在 2018 年, Dam 等人[41] 利用已有的 Android study dataset 和 Firefox 数据集, 人工检索了来自 F-Droid 和 Android OS 的源代码文件, 并从 Firefox 代码存储库中提取了 Firefox 2.0 的源文件, 从而发布了一个包含超过 24 万个 Java 序列以及 1.1 万个 C/C++文件的大型数据集.

在 2020 年, Wang 等人[40]发布的多语言数据集 FUNDED 由 SARD、NVD 和 GitHub 上的开源项目构建而来, 包含 CWE2019 中定义的 top-5 到 top-30 最危险的安全漏洞. 他们使用 SARD 和 NVD 提供的修补版本作为无漏洞代码示例, 并通过补丁提交来获得 GitHub 上收集到的易受攻击样本的无漏洞版本.

在 2021 年, Bhandari 等人[183]发布了一种从 NVD 和 CVE 数据库中自动收集漏洞数据及其修复程序的方法, 并同步构建了一个大型多语言数据集 CVEfixes. CVEfixes 收集工具能够自动从 NVD 获取所有可用的 CVE 记录, 并从关联的开源存储库中收集易受攻击的代码及相应的修复程序. CVEfixes 数据集包含了从文件到提交等多粒度的漏洞代码及其无漏洞版本, 涵盖了大量 CWE 类型及开源项目.

在 2022 年, Yan 等人[25]发布了一个人工构建的数据集 Juliet+. 他们将 Juliet test suite 中的测试用例分为易受攻击和不易受攻击两部分, 并对源代码进行解析, 提取了其中的方法和调用信息. 然后, 他们根据函数名称, 基于正则表达式对其进行标记. 特别地, Yan 等人发现 Juliet test suite 中的部分代码行被错误标记, 因此他们使用源变量和接收变量对漏洞相关的代码行进行了重新标记.

(2) C/C++程序漏洞数据集

在 2013 年, Shin 等人[7]从 Mozilla 基金会安全公告 (MSFA) 收集了 Firefox 2.0 报告的漏洞信息, 与 Firefox 中的错误报告进行关联, 根据文件的修改情况识别出源代码中存在漏洞的文件, 发布了包含约 1.1 万个文件的 Firefox 数据集.

在 2017 年, 针对开源项目 FFmpeg、LibPNG 和 LibTIFF, Lin 等人[81]从 GitHub 上获取源代码, 从 NVD 和 CVE 数据库中收集数据, 根据 CVE 记录将对应函数标记为有漏洞, 将剩余的函数标记为无漏洞, 发布了包含超过 6 000 个函数的 POSTER 数据集.

在 2018 年, Russell 等人[3]发布了 Draper VDISC 漏洞数据集, 包含来自 Juliet test suite、Linux Debian 发行版和 GitHub 的数百万个 C/C++代码示例. 其中, 代码标签由 3 个开源静态分析器 Clang、Cppcheck 和 Flawfinder 生成, 并由人工映射到相应的 CWE 类型. Li 等人[133] 从 NVD 及 SARD 中收集数据, 重点考虑缓冲区漏洞和资源管理漏洞, 发布了一个涵盖 19 个流行开源产品的漏洞数据集 VulDeePecker. Lin 等人[12]从 NVD 和 CWE 数据库中收集数据并手动进行标记, 发布了一个涵盖 6 个流行开源软件、超过 3.2 万个函数的漏洞数据集. Sestili 等人[178]认为 Juliet test suite 太小太复杂, 无法用于软件漏洞预测, 因而提出了一个 C 语言源代码数据集生成器, 用于提供尽可能简单的、包含缓冲区溢出漏洞的源代码. 在此基础上, 他们构建了包含超过 3.8 万个文件的漏洞数据集 s-bAbI.

在 2019 年, 为了分析自己所设计的模型 Devign 的有效性, Zhou 等人[134]在 Linux 内核、QEMU、Wireshark 和 FFmpeg 中收集与安全相关的提交, 判断这一提交是否与漏洞修复相关, 进而从中提取易受攻击及非易受攻击的函数, 据此构建了一个大规模的漏洞数据集 Devign.

在 2020 年, Fan 等人[180] 抓取了 2002–2019 年的 CVE 数据库, 从中获取漏洞的描述性信息. 接着, 他们将其关联到相关的 GitHub 代码库链接, 提取与漏洞相关的代码更改, 得到了包含超过 25 万个函数、涵盖 91 种漏洞类型的数据集 Big-Vul. Zagane 等人[32] 在搜集到的 SySeVR 数据集的基础上, 对其中的每个代码切片计算 McCabe、Halstead 等 18 个代码度量指标, 删除冗余数据, 构建了一个包含超过 9.5 万个实例的漏洞数据集. Lin 等人[88]在 Asterisk、FFmpeg 等 9 个用 C 语言编写的流行开源软件项目的基础上, 基于 NVD 和 CVE 网页的描述进行标记, 构建了一个双粒度数据集, 包含超过 5 700 个文件和 6 万个函数.

在 2021 年, Li 等人[37]构建了包含 126 种漏洞类型的数据集 SySeVR, 其中有 1 500 个 C/C++文件来自于 NVD、1.4 万个 C/C++文件来自于 SARD. Mazuera-Rozo 等人[54]在 GitHub Archive 数据集的基础上构建了一个包含约 31.5 万对易受攻击/不易受攻击的 C/C++函数、涵盖 29 种不同漏洞类型的大型数据集. Zheng 等人[181] 从 FFmpeg、OpenSSL、LibTIFF 等多个开源 C/C++项目的多个版本中收集数据, 使用静态程序分析及差异分析技术收集其中的漏洞, 同时保留更多漏洞的详细信息, 构建了一个包含超过 130 万个函数样本的 D2A 漏洞数据集. Chakraborty 等人[16]构建了一个来源于 Linux Debian 内核和 Chromium 这两个开源项目、包含超过 1.8 万个函数样本的漏洞数据集 Reveal. Cheng 等人[80]收集数据并根据提交信息进行标记, 构建了一个包含函数和代码片段两种粒度的, 基于 SARD 以及两个真实开源项目 lua、redis 的大型数据集. Ziems 等人[46]发布了一个基于 SARD 的大型数据集, 包含超过 10 万个文件, 涵盖 123 种类型的漏洞. Alqarni 等人[84] 认为常用的 Draper 数据集存在数据不平衡问题, 因此采用欠采样方法对其进行优化, 构建出一个包含相同数目的有漏洞与无漏洞函数的大型数据集, 包含近 200 万个函数实例. Lu 等人[182] 随机打乱 Devign 漏洞数据集, 按照训练/验证/测试集分别为 80%/10%/10% 的比例对其进行拆分, 得到 CodeXGLUE 漏洞数据集.

在 2022 年, Gong 等人[23]构建了一个基于 Linux 内核源代码 (LKSC) 的漏洞数据集 LKSC+. LKSC 包含多种类型的函数, 适合训练分类模型. 由于 LKSC 中包含的某些种类的函数不足, Gong 等人对其进行了扩展, 插入了从 GitHub 和 StackOverflow 收集到的函数. 最终得到的 LKSC+数据集包含超过 3.4 万个函数. Wu 等人[77] 从 NVD、SARD 中收集数据, 构建了一个包含超过 1.3 万个有漏洞函数及近 2.7 万个无漏洞函数的大型漏洞数据集. Cao 等人[22] 从 SARD 及 CVE 数据库中收集数据, 构建了一个包含 10 个 C/C++开源项目、涵盖 13 种 CWE 类型和 1 200 个 CVE 记录的大型数据集.

在 2023 年, Wang 等人[67] 从 CVE 数据库中收集数据, 根据提交信息收集各函数的有漏洞及修复版本, 最终得

到了一个包含 303 个流行 C/C++项目, 涵盖大量 CVE 记录的大型数据集. Islam 等人[74] 从 GitHub 和 NVD 中收集数据, 将易受攻击的函数关联到对应的 CWE 类型, 最终得到的数据集中包含超过 14 万个函数, 涵盖超过 40 种 CWE 类型.

在 2024 年, Tian 等人[112] 在 SARD 的基础上, 使用 tree-sitter 处理代码, 识别易受攻击的函数及其修补的版本, 并将其与 CWE 类型相关联, 构建了一个包含超过 26 万个函数、涵盖 118 种 CWE 类型的大型 C/C++漏洞数据集.

(3) 汇编/二进制漏洞数据集

在 2018 年, Le 等人[179]从 NDSS18 源代码数据集中提取函数, 进行预处理过滤掉重复的函数, 并将其中的大部分编译为二进制文件, 得到包含超过 3.2 万个二进制函数的漏洞数据集 NDSS18 binary dataset (ICLR 2019). 具体而言, 他们使用 gcc/g++编译器进行编译, 在编译过程中捕获并解析错误消息, 使用 Joern 了解错误消息在源代码中的位置, 并修复相应的错误, 重复这一过程直到源代码能够编译为二进制文件为止.

在 2021 年, Yan 等人[102] 通过删除上述 ICLR 2019 数据集中的重复样本, 得到了一个新的二进制漏洞数据集 ICLR19. 数据集中包含 3 400 个有漏洞样本和 3 743 个无漏洞样本, 涵盖 CWE-119 和 CWE-399 这两种类型的漏洞, 并按照 6:1:3 分为训练集、验证集和测试集.

(4) PHP 程序漏洞数据集

在 2019 年, Fang 等人[132]构建了一个 PHP 数据集 SQLI-LABS. SQLI-LABS 是一个专门为学习和演练 SQL 注入漏洞 (CWE-89) 而设计的实验平台, 用户可以通过这一应用学习 SQL 注入漏洞的相关理论知识. SQLI-LABS 数据集中包含从这一应用中收集的 69 个与漏洞相关的示例文件.

在 2021 年, Şahin 等人[91]构建了一个带有软件度量的漏洞数据集, 包含 233 个经过验证的代码漏洞, 数据来源于 95 个版本的 phpMyAdmin 3.3.0、Moodle 2.2.0 和 Drupal 6.0.0. 对每个实例, 该数据集提供了代码行、圈复杂度、函数数目、扇入和扇出等特征信息.

(5) Python 程序漏洞数据集

在 2022 年, Wartschinski 等人[19] 利用 GitHub 上公开的 Python 项目创建了 VUDENC 漏洞数据集. 项目数据以包含安全相关修补程序的提交形式收集. 在这些提交中, 被修改或删除的代码以及与其最接近的代码被标记为易受攻击, 而其余代码和修复后的新版本则被标记为非易受攻击. 为了保持数据集的平衡, 数据被划分为代码块. 如果一个代码块与某个易受攻击的代码段重叠, 那么它被标记为易受攻击; 否则, 它被标记为不易受攻击. 通过这种方法, Wartschinski 等人收集了超过 1.4 万个漏洞修复提交. 最终生成的数据集中包含源代码片段和相关提交信息, 涵盖了 7 种不同类型的漏洞.

7.2 研究所选择的检测对象及检测结果

在模型训练完成后, 除使用测试集对模型性能进行评估外, 部分研究还将真实世界中的大型开源软件作为检测对象, 判断模型是否能够真正检测出其中未知的漏洞. 常见的做法为, 从代码仓库中收集来自开源软件的源代码, 使用训练好的模型进行检测, 人工确定模型报告出的漏洞是否真实存在. 上述过程模拟了将软件漏洞预测模型部署于实际开发流程中的情况, 能够很好地评估模型的可扩展性及实用性. 151 篇文献中, 有 8 篇文献按照上述过程进行了评估, 并检测出了至少 1 个从未被披露的未知漏洞. 部分未知漏洞在软件的后续版本中被开发人员静默修补, 最新版本中仍未修补的漏洞由作者报告给软件供应商. 表 6 显示了这 8 项研究所选择的检测对象, 以及研究检测出漏洞的情况.

表 6 各项研究选择的检测对象及检测结果汇总

发表时间	文献	作者	检测对象	检测结果
2018	[133]	Li等人	Xen、Seamonkey、Libav	检测出4个NVD中未报告的漏洞
2020	[38]	Li等人	FFmpeg、Wireshark、Libav	检测出2个NVD中未报告的漏洞
2020	[89]	Guo等人	SEACMS、ZZCMS、CMS Made Simple	检测出3个未知漏洞
2021	[37]	Li等人	Libav、Seamonkey、Thunderbird、Xen	检测出15个NVD中未报告的漏洞, 其中7个报告给供应商

表 6 各项研究选择的检测对象及检测结果汇总 (续)

发表时间	文献	作者	检测对象	检测结果
2021	[101]	Jeon等人	boringssl、mpc、expat	检测出4个未知漏洞, 其中2个报告给供应商
2022	[77]	Wu等人	Libav、Xen、Seamonkey	检测出73个NVD中未报告的漏洞, 其中52个报告给供应商
2023	[135]	Mirsky等人	来自GitHub的9个无标签项目	检测出1个未知漏洞
2023	[29]	胡雨涛等人	Libav、Xen、OpenSSL、httpd	检测出59个NVD中未报告的漏洞, 其中38个报告给供应商

除上述 8 项研究外, 其余研究所使用的测试集往往来源于现有数据集, 评估结果是基于已知漏洞得到的. 部分研究中, 提出的模型检测出了数据集中未报告的漏洞, 但研究人员并未对漏洞的真实性进行确认. 这一现状导致现有模型的性能是否能够扩展、现有模型在实际的漏洞预测任务上是否具有实用性仍然是无法确定的. 软件漏洞预测的最终目的是发现真实项目中的潜在漏洞. 因此, 尝试以检测出真实世界中的未知漏洞为导向进行软件漏洞预测任务, 是评估模型能够获得的实际性能最直接而有效的方式.

7.3 选择漏洞数据集时存在的问题

本节汇总了上述文献在深度学习驱动的软件漏洞预测问题的研究中所构建及使用的漏洞数据集. 通过分析我们发现, 当前的研究中漏洞数据集的选择存在如下问题.

(1) 现有的数据集在极大程度上偏向 C/C++语言. 大部分文献所获取的漏洞数据集来源于 NVD、SARD、CWE 数据库等著名漏洞数据库及数据集, 而这些数据库及数据集中包含的样本大部分为 C/C++语言所编写. 许多经典而常用的漏洞数据集, 例如 VulDeePecker[133]、Devign[134]、SySeVR[37]等数据集的编写语言均为 C/C++. 其余流行而常用的编程语言, 如 Java、Python 等语言的数据集数目稀少, 缺乏经典数据集, 且漏洞数据难以收集, 多数情况下需要研究人员自行构建. 而 Go、C#、Swift 等常用语言的数据集几乎无法搜索到. 上述对 C/C++语言的偏向严重限制了对其他编程语言进行的漏洞预测工作. 因此, 除 C/C++语言之外的漏洞数据集的构建应得到更多重视.

(2) 使用最为广泛的数据集存在较大问题. SARD 及 Juliet test suite 是使用最为广泛的漏洞数据集, 数目众多的研究从 SARD 或 Juliet test suite 中收集数据, 对模型性能进行评估. 然而, 上述数据集存在较大问题. Rabheru 等人[47] 的研究表明, SARD 中包含的函数相当简单, 且其中包含大量重复代码, 代码间仅有 1–2 行的差异. J-DS 是一个合成数据集, 而合成数据集往往难以代表真实世界中的软件代码[115]. 另外, Juliet test suite 中还存在代码被错误标记的情况[25]. 使用上述数据集进行研究可能导致模型的性能被错误估计, 甚至出现过拟合问题. 这启示研究人员选择更真实、更精确的漏洞数据集, 或对上述数据集进行去重、重新标记等步骤后再进行使用.

(3) 数据集的质量参差不齐, 部分数据集质量较低. Li 等人[119]在研究中发现, 现有的漏洞数据集中存在错误的标记和粗粒度的漏洞描述. 例如, Draper VDISC 数据集是通过传统的静态检测方法生成的, 这一方法存在误报率高的问题; NVD 中仅提供了漏洞代码和修复后版本之间的差异, 很难找到与安全相关的程序点. Chen 等人[184]在一项针对现有数据集 CVEFixes、Big-Vul 和 CrossVul 的研究中发现, 这些数据集的标签准确率非常低, 其中 Big-Vul 数据集的准确率仅为 25%. Jain 等人[185]在一项评估现有数据集质量的研究中发现, Big-Vul 数据集中存在将空函数映射到 CWE-416 的现象, 且部分函数被标记为有漏洞仅仅是由于其调用了另一个并非库函数的函数. 总体来看, 现有的漏洞数据集存在包含大量合成代码、标签大量错误、样本高度相似甚至重复、数据不平衡等问题. 这对使用这些数据集进行训练及测试得到的漏洞预测模型的真实性能提出了挑战.

(4) 高效的漏洞数据集获取手段缺乏. Russell 等人[3]在研究中发现, 来自 GitHub 的代码通常质量较低; 来自 Debian 及 GitHub 的代码及提交没有标记或包含了低质量的标记, 需要手动进行检查. Yan 等人[25]在研究中发现, 即使对于经验丰富的安全专家来说, 在现实世界中的漏洞数据集中标记与漏洞存在相关的重要代码段也是复杂且耗时的. Nong 等人[42]从开放科学的角度对深度学习驱动的软件漏洞预测进行研究, 发现只有低比例的研究为他们的方法提供了公开可用的数据集, 且数据集的可用性不佳. 总体来看, 漏洞数据集的获取存在漏洞代码搜集困难、标记困难、可用性差等问题.

(5) 数据集缺乏质量评估、更新迭代过程. 常用的漏洞数据集或多或少存在质量方面的缺陷, 但对数据集进行质量评估、对数据集进行去重、对数据集进行类平衡化、对数据集进行整合形成更大更全面的新数据集等工作存在较大程度上的缺乏. 例如, 上述文献中, 仅有 ICLR19 数据集[102]、ICLR 2019 数据集[144]、Juliet+数据集[25]、Draper processed 数据集[84]等少数数据集是经过删除重复样本、重新标记、数据平衡化等优化操作得到的. 此类优化操作有助于获得更公正、更全面的漏洞数据集, 需要得到重视.

(6) 在现有数据集上进行评估所得到的模型性能常常被错误估计. 现有漏洞数据集存在的各类问题对各项研究所设计的漏洞预测模型提出了挑战: 在存在质量缺陷的数据集上进行评估所得到的模型性能往往是不真实的. Nong 等人[42]在研究中得出结论, 由于常用的数据集存在数据不平衡、样本过于简单等问题, 深度学习驱动的软件漏洞预测技术的实际整体有效性可能被严重高估了. Chakraborty 等人[16]也认为, 现有数据集存在数据复制、数据不平衡等问题, 导致现有模型的性能被错误估计了. Nie 等人[186] 在研究中发现, 标签错误严重影响了主流漏洞预测模型, 最坏情况下平均 $F1$ score 下降 20.7%.

(7) 各项研究工作使用各自的数据集, 难以相互比较. Alaoui 等人[17]在研究中发现, 比较这一领域的研究工作几乎是不可能的. 这是因为大多数研究使用的漏洞数据集是自行构建的私人数据集, 即使它们使用经典而常用的公共数据集, 也会使用其中不同的部分. 另外, 这些研究对数据集进行的预处理也是不同的. Alaoui 等人认为, 需要标准的、真实的公共数据集, 促进研究工作之间的比较分析.

8 漏洞预测有效性评估指标

在完成模型的建立和测试后, 需使用一系列指标对漏洞预测模型的有效性进行评估. 评估指标的选择会直接影响评估结果. 若选取的评估指标不恰当, 将无法真实地描述模型在实际漏洞预测任务中所表现出的有效性. 本节首先综述上述研究中所采用的漏洞预测性能评估指标, 接着分析这些文献在评估指标选择方面存在的问题.

8.1 研究所使用的漏洞预测性能评估指标

表 7 显示了上述文献所使用的漏洞预测性能评估指标, 表中列出了评估指标的名称、计算公式、描述以及评估指标的使用情况.

表 7 各项研究所使用的漏洞预测性能评估指标汇总

名称	计算公式	描述	使用情况
Recall (召回率)	$Recall=\frac{TP}{TP+FN}$	有漏洞样本中预测正确的样本占比, 又称为*TPR* (真阳性率)、查全率	129篇文献至少使用了召回率、精度、准确率这3个指标之一
Precision (精度)	$Pre=\frac{TP}{TP+FP}$	预测为有漏洞的样本中实际有漏洞的样本占比, 又称为查准率	
Accuracy (准确率)	$Acc=\frac{TP+TN}{TP+FP+TN+FN}$	所有样本中预测正确的样本占比	
F_β	$F_\beta=\frac{(1+\beta)^2TP}{(1+\beta)^2TP+\beta^2FN+FP}$	召回率和精度在实际情况中的不同重要性 (β 通常取0.5、1或2)	$F1$ score: 被111篇文献所使用 $F2$ score: [33,96]
FPR (假阳性率)	$FPR=\frac{FP}{FP+TN}$	无漏洞样本中被预测为有漏洞的样本占比, 又称为误报率	[27,43,94,110,111,114–116,124,127,131]
FNR (假阴性率)	$FNR=\frac{FN}{TP+FN}$	有漏洞样本中被预测为无漏洞的样本占比	[27,43,94,110,111,114–116,124,127,131]
MCC (马修斯相关系数)	$MCC=\frac{TP\cdot TN-FP\cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	样本的预测类别与实际类别之间的相关系数	[3,35,37,43,108,138,150]
汉明距离	$D_{\text{Hamming}}(\hat{y},y)=\frac{1}{n}\sum_{j=0}^{n-1}I\left(\hat{y}_j\neq y_j\right)$	样本的预测类别与实际类别之间的不一致程度	[132]

表 7 各项研究所使用的漏洞预测性能评估指标汇总 (续)

名称	计算公式	描述	使用情况
IFN (*Informedness*)	$Informedness = Recall + Inverse\ Recall - 1$	量化预测模型对指定条件的知情程度, 表示预测相比随机预测而言多包含的信息量	[80]
MKN (*Markedness*)	$Markedness = Precision + Inverse\ Precision - 1$	量化预测模型对指定条件的标记程度, 表示预测模型的正确标记能力	[80]
Kappa系数	$k = \frac{p_o - p_e}{1 - p_e}$	用于检查一致性和衡量分类准确性	[132]
M_FPR、*M_FNR*、*M_F*1	$M\_FPR = \frac{1}{\lvert L\rvert}\sum_{l\in L}\frac{FP_l}{FP_l + TN_l}$ $M\_FNR = \frac{1}{\lvert L\rvert}\sum_{l\in L}\frac{FN_l}{TP_l + FN_l}$ $M\_F1 = 2\times\frac{M\_FPR\times M\_FNR}{M\_FPR + M\_FNR}$	每种漏洞类型对应的指标的算术平均	[39]
W_FPR、*W_FNR*、*W_F*1	$W\_FPR = \frac{1}{\sum_{l\in L}\lvert X_l\rvert}\sum_{l\in L}\lvert X_l\rvert\frac{FP_l}{FP_l + TN_l}$ $W\_FNR = \frac{1}{\sum_{l\in L}\lvert X_l\rvert}\sum_{l\in L}\lvert X_l\rvert\frac{FN_l}{TP_l + FN_l}$ $W\_F1 = 2\times\frac{W\_FPR\times W\_FNR}{W\_FPR + W\_FNR}$	每种漏洞类型对应的指标, 按照该类型样本数目在所有样本中占比的加权平均	[39]
AUC	ROC曲线下面积	模型对随机选择的正类评分高于随机选择的负类的概率	[28,32,41,43,45,48,49,69,72,80,85,87,92,95,132,135,140,142,154]
top-*k*准确率	$A@k = \frac{TP@k + TN@k}{TP@k + FP@k + TN@k + FN@k}$	在被预测为最可能存在漏洞的前*k*个代码或文件上进行指标计算	[60]
top-*k*召回率	$R@k = \frac{TP@k}{TP@k + FN@k}$		[36,81,88,90,121,126,128,130]
top-*k*精度	$P@k = \frac{TP@k}{TP@k + FP@k}$		[12,35,35,71,88,90,121,126,128,130]
*IoU*值	$IoU = \frac{\lvert U\cap V\rvert}{\lvert U\cup V\rvert}$	模型预测存在漏洞的样本中, 漏洞行位置的准确性	[157]
$\lvert V\rvert$值	$\lvert V\rvert = \frac{M}{N}$	样本中模型预测出的漏洞行数量的平均值	[157]
预测距离	$Distance = \lvert l_{\text{pred}} - l_{\text{true}}\rvert$	反映模型预测的漏洞行位置与真实位置之间的距离	[60]
FIRR (函数检查减少率)	$FIRR_{\text{A-to-B}} = 1 - \frac{N_{\text{B}}}{N_{\text{A}}}$	与使用方法B相比, 方法A可以节省的精力	[12]
Loss (损失率)	$H(p,r) = H(p) + D_{KL}(p\|\|r)$	可由交叉熵损失函数计算, 反映预测结果和真实结果之间的差距	[149]

不同的评估指标能够反映模型在不同方面的性能, 同时适用于不同的漏洞预测场景. 我们按照评估指标所适用的预测场景对指标进行分类, 并按照类别对上述各评估指标进行介绍.

(1) 二分类性能指标

准确率、精度和召回率这 3 个指标是广泛应用的经典二分类性能评估指标, 在漏洞预测领域的绝大多数文献中都有涉及, 它们从正面的角度评估模型正确分类的情况. 有 129 篇文献使用了上述指标中的至少一个来进行模

型性能评估, 并且其中大部分文献同时使用了多个指标.

F_β 是一项综合指标, 以加权平均的方式反映了召回率和精度之间的关系. 当 β 设置为 1 时, 召回率和精度被认为同等重要, 产生的 *F*1 score 是最常用的性能评估指标之一, 在 111 篇文献中得到了应用. 漏洞预测的主要目标是尽可能准确地预测出漏洞, 因此在分析预测结果时, 虽然误报会增加工作量, 但漏报可能会导致系统存在易受攻击的代码, 这是代价高昂且难以容忍的. 基于这一理念, 召回率的重要性超过了精度. 因此, 一些研究将 β 设置为 2, 从而使召回率的权重是精度的两倍.

除了从正面的角度评估模型的正确预测情况外, 还可以从负面的角度考虑模型出错的情况. 常见的此类指标包括假阳性率和假阴性率, 它们的计算也基于混淆矩阵, 能够反映出模型预测错误的样本所占比例. 与正面评估指标相比, 使用这些指标的文献相对较少.

马修斯相关系数 *MCC* 是一项综合的分类性能评估指标, 用于描述分类预测结果与实际结果之间的相关性. *MCC* 综合考虑了正面和负面两个角度, 在其计算过程中涵盖了所有 4 种样本类别, 因此被视为较为平衡的指标, 即使在数据不平衡的情况下也能得到较好的应用. *MCC* 的取值范围在−1~1 之间, 数值越大, 表示预测模型的性能越优秀.

汉明距离是一个类似于准确率的指标, 源自信息论, 用于度量两个相同长度字符串在对应位置上不同字符的个数. 在应用于分类问题时, 它表示样本预测的类别与实际类别之间的不一致程度, 即在所有样本中, 预测类别与实际类别不一致的样本所占比例. 值得注意的是, 这一指标的值等同于 1 减去准确率. 汉明距离的取值范围在 0–1 之间, 数值越小表示预测模型的性能越优秀.

为了克服传统评估指标的偏见性问题, Powers 等人[187]于 2011 年提出了两个分类性能评估指标 IFN 和 MKN, 用于对召回率和精度进行无偏修正. Powers 等人研究了多种传统评估指标, 并发现它们大多存在偏差. 因此, 他们引入了一系列包括 IFN 和 MKN 在内的新指标, 以优化传统评估方法. IFN 用于度量预测模型对特定条件的了解程度, 衡量在特定条件下模型的预测相对于随机预测所包含的额外信息量. MKN 是 IFN 的补充概念, 用于量化模型在特定条件下输出标签的可信程度, 表示模型预测所包含的可信信息比例. IFN 和 MKN 都具备无偏性, 能够消除标签偏见和数据不平衡等因素对评估结果的影响. IFN 和 MKN 的取值范围均在–1~1 之间, 且均与预测模型的性能呈正相关.

(2) 多分类性能指标

部分文献在进行漏洞预测时, 还考虑了漏洞所属的 CWE 类型, 将各个代码片段映射到无漏洞或某种特定类型的漏洞. 此时的漏洞预测问题是一个多分类问题, 需要使用多分类指标进行评估. Kappa 系数是一个基于混淆矩阵进行计算、能够用于多分类结果衡量的指标. Kappa 系数适合用于数据不平衡的情况, 它能够对偏重其中某些类别, 而在其余类别上预测结果较差的模型给出较低的分数. 只有在各类样本上均能取得较好的预测结果, 才能获得较好的 Kappa 系数. Kappa 系数的取值在−1~1 之间, Kappa 系数越大, 预测模型的性能越好.

M_FPR、*M_FNR*、*M_F*1、*W_FPR*、*W_FNR*、*W_F*1 是一系列对多类漏洞预测模型进行性能评估的指标. 为了按照漏洞的不同类型对预测模型的性能进行评估, 可以分别在包含各个类型的漏洞的样本上计算评估指标, 并将其进行综合, 得到最终的评估结果. 可以直接将所有评估结果进行算术平均, 得到 *M_**系列指标; 也可以按照包含各个漏洞类型的代码片段的数目占比对所有指标进行加权平均, 得到 *W_**系列指标. *M_F*1、*W_F*1 与预测模型的性能呈正相关, *M_FPR*、*M_FNR* 等指标则与预测模型的性能呈负相关.

(3) 排序性能指标

AUC 是一项经典的排序指标, 广泛用于分类模型的性能评估和比较. AUC 定义为 ROC 曲线下与坐标轴所围成的面积. ROC 曲线以假阳性率为 x 轴、召回率为 y 轴绘制, 表示分类阈值从小到大变化时模型的表现. 当模型将所有样本按照预测为存在漏洞的概率从大到小排序时, AUC 表示实际存在漏洞的样本排在前面的概率. AUC 的取值范围在 0–1 之间, 数值越大表示预测模型性能越佳.

top-*k* 召回率和 top-*k* 精度是在特定条件下对召回率和精度的计算方法. 真实的软件漏洞预测任务往往需要人工检查以最终确定漏洞的存在, 而最有效的方法是检查被模型预测为最易受攻击的代码部分. 因此, 在进行模型性

能评估时, 可以将所有代码或文件按照模型预测为存在漏洞的概率进行排序, 然后选取前 k 个样本进行指标计算. 这样可以在保证效率的同时对模型的性能进行有效评估.

(4) 行级性能指标

为了实现更精细的漏洞预测, 某些研究文献聚焦于漏洞在代码行级别的定位, 由此衍生出一系列用于评估漏洞定位性能的指标. 其中, *IoU* 值是一个表征漏洞预测模型对样本中漏洞代码行所在位置的预测的准确性的度量标准. *IoU* 值综合考虑了实际漏洞行与预测模型所标记的漏洞行的交集与并集比例, 衡量了模型对漏洞代码行的定位的准确性. 较大的 *IoU* 值意味着漏洞预测模型性能更佳.

$|V|$值是一个与 *IoU* 值相辅相成的指标, 用于描述漏洞预测模型对漏洞代码行的定位的精确性. 该指标表示各样本中模型预测出的平均漏洞行数, 定义为预测出的漏洞行总数与样本数目的比值. 在 *IoU* 值较大且$|V|$值较小的情况下, 模型能够更精确地定位漏洞, 从而提供更有效的预测信息.

预测距离是另一种衡量模型漏洞定位能力的标准. 针对在代码行粒度进行预测的漏洞预测模型, 预测距离定义为模型预测的漏洞位置与实际漏洞位置之间的行数差. 较小的预测距离表示预测模型性能更佳.

(5) 其他性能指标

FIRR 是 Lin 等人[12]在 2018 年提出的一个指标, 这一指标从工作量的角度比较不同方法的漏洞预测效果. *FIRR* 的含义为相对于方法 B, 方法 A 可以节省的工作量. 在期望找出数据集中一定数量的漏洞时, 召回率较低的方法需要检查更多的函数. 因此, 通过比较两种方法需要检查的函数数量, 可以比较它们的漏洞预测性能.

Loss 是机器学习领域常用的预测性能评估指标, 用于衡量预测结果与真实结果之间的差距. 预测模型的损失率可以通过多种损失函数进行计算. 在分类问题中, 常用的损失函数为交叉熵损失函数. 交叉熵损失函数通常用于衡量两个分布之间的差异, 它受到两个因素的影响: 预测结果的熵、预测结果与真实结果之间的 KL 距离. 交叉熵损失函数的值越小, 预测模型的性能越好.

8.2 选择性能评估指标时存在的问题

本节汇总了上述文献在深度学习驱动的软件漏洞预测问题的研究中所使用的性能评估指标. 通过分析我们发现, 当前的研究中性能评估指标的选择存在如下问题.

(1) 数据集的特点会影响评估指标的有效性. 由于某些评估指标自身的局限性, 其易受到数据集的特点的影响. 例如, 当数据集中存在的绝大部分样本均为无漏洞样本时, 简单地将测试集中的所有样本均预测为无漏洞即可实现很好的准确率和真阴性率, 但此时的模型并无漏洞预测能力. 当数据集不平衡时, Nguyen 等人[130]在研究中发现, 精度、召回率和 $F1$ score 这 3 个指标会低估模型的预测性能; Lin 等人[88]在研究中发现, 分类器会更加偏向于拟合占比多数的类别的数据分布, 并且在默认情况下对二元分类使用 0.5 作为决策的边界, 故使用精度和召回率等指标会低估预测模型的性能.

(2) 使用单一指标不足以准确评估模型的性能. Semasaba 等人[123]在研究中发现, 使用单一指标进行结果分析是深度学习驱动的软件漏洞预测领域研究的局限性之一. Alaoui 等人[17]在研究中发现, 一些研究使用单个性能指标, 例如准确率, 来评估其预测模型, 这在数据集不平衡的情况下是不够的. 他们认为, 重要的是确定研究人员应考虑的标准性能指标, 以确保所提出的模型准确而高效, 可靠且可重复.

(3) 现有方法所使用指标的评价范围有限. 现有研究大部分使用精度、召回率、准确率等正面评估模型性能的指标, 但很少使用假阳性率、假阴性率等负面指标. Alaoui 等人[17]在研究中发现, 很少有研究人员提供详细的误报报告. Chakraborty 等人[16]在研究中发现, 各项研究虽然报告了在实际项目中发现漏洞的情况, 但所使用的指标并未阐明假阳性和假阴性. 然而, 漏洞预测模型的误报和漏报的数量与开发人员在漏洞预测方面所需要做出的努力直接相关, 过多的误报和漏报情况会阻止开发人员对模型的使用.

9 现有挑战与未来研究工作

软件漏洞预测是一项极具复杂性和挑战性的任务. 本节首先介绍各项研究所提及的挑战, 然后总结深度学习驱动的漏洞预测领域中, 值得在未来研究中关注和探索的研究问题.

9.1 漏洞预测中的挑战

表 8 汇总了上述文献所提出的研究中遇到的问题及挑战. 表中列出了问题或挑战的类别、名称、描述以及问题或挑战的提及情况. 可以看出, 相关的问题及挑战主要集中在模型的预测能力、漏洞数据集的质量及获取、代码的表示形式、代码向量化方式、评估指标、模型的可解释性等方面.

各研究所提出的问题与挑战分布在软件漏洞预测工作的各个阶段. 我们按照所属的类别对上述各问题与挑战进行描述.

表 8 各项研究所提出的问题及挑战汇总

问题或挑战类别	问题或挑战名称	描述	提及情况
模型预测能力	预测其他编程语言的漏洞	使用训练好的模型预测其他编程语言、其他类型的漏洞	[8,15,19,22–24,27,31,32,34,35,37–39,43,47,48,54,57,59,61,64,68,70,73,80,88,95,96,98,101,106,108,110,112,114,115,117,119,128,130,133,139]
	预测其他类型的漏洞		[16,24,29,31,32,35,39,43,47,48,57,94,100,110,113,124,133,152]
	预测更细粒度的漏洞	在更细的粒度上进行漏洞预测	[18,33,37,39,54,64,56,96,97,98,103,114,129,131]
	超参数并非最优	模型的超参数可能不是最优	[10,43,49−51,54,70,99,104,106,114,125,151]
	跨域漏洞预测	使用训练好的模型预测来自其他项目的漏洞	[60,95,98,123,126,142,147]
	预测复杂的漏洞	预测出跨函数、跨文件的大型复杂漏洞	[12,16,19,21,26,36,43−45,49,56,70,89,95,125]
	语言模型并非软件漏洞预测模型	GPT等语言模型并非为软件漏洞预测而设计	[66,68,109]
数据集质量	合成数据集不具代表性	包含大量合成样本的数据集不具代表性	[16,25,40,41,42,47,55,59,66,77,106,115,118,129,137,143]
	在真实数据集上进行实验	在相对复杂的真实数据集或真实软件上进行实验	[16,17,19,22,25,38,44,45,51,55,99,100,114,128,129]
	样本缺乏标记或错误标记	数据集中样本缺乏标记或存在错误标记	[18,22,27,37,43,47,55,80,99,115,117,119,120,123,125,129,137,141,143]
	样本高度相似或重复	数据集中样本高度相似或出现重复	[16,47,55,104,141]
	数据不平衡	数据集中样本类别不平衡	[12,16–18,42,43,99,101,104,112,122,123,125,126,130]
	源代码难以编译或无法编译	数据集中的源代码难以编译或无法编译	[55,119]
	数据集过小	数据集中包含样本数目太少	[42,55,60,123,126]
	时间旅行	使用来自未来的数据作为训练集, 来自过去的数据作为测试集	[141]
数据集获取	数据集缺乏	漏洞数据集缺乏, 难以获取	[18,28,44,45,119,128,129,131]
	代码追踪系统质量较低	来自代码追踪系统的代码质量较低	[3]
	纠缠提交	漏洞修复提交同时还包含其他方面的修改	[16,47,54,66,95,141]
	修复提交出错	漏洞修复提交可能未修复漏洞	[19]
	丢弃可能包含信息的文件	丢弃了可能包含关键信息的非源代码文件	[41]
	编译及反汇编过程复杂	对源代码进行编译及反汇编的过程复杂	[28]
代码表示形式	语法代表性缺失	基于标识符的表示形式缺乏语法代表性	[16]
	遍历导致信息丢失	对非线性表示形式的遍历导致信息丢失	[18]
	表示形式包含无关信息	表示形式中包含许多与漏洞存在无关的信息	[43,48,61,129]
代码向量化方式	词汇爆炸	代码中可能出现的标识符数目几乎是无限的	[16]
	处理OOV单词导致信息丢失	对OOV单词的处理存在信息丢失问题	[121]
	填充截断向量导致信息丢失	填充和截断向量的操作将会导致信息丢失	[104,126,128]

表 8 各项研究所提出的问题及挑战汇总 (续)

问题或挑战类别	问题或挑战名称	描述	提及情况
评估指标	评估指标受到数据集的影响	数据集的不同特点会影响评估指标的有效性	[88,130]
	评估指标单一	使用单一的性能评估指标可能是不够的	[17,123]
	所用指标的评估范围及能力有限	使用的评估指标的评估范围及评估能力不足	[16,17,141]
可解释性	方法的可解释性	文献中提出的深度学习驱动的方法的可解释性	[18,27,31,34,37,38,40,55,68,95,112,121,127,134,137,151,152]
可用性	原始方法难以复现	原始方法可能难以复现	[41,50,61]
可判定性	判断漏洞是否可被利用	判断漏洞是否真正能够被攻击者利用	[101]
	判断漏洞是否适合预测	判断漏洞是否适合使用深度学习技术进行自动预测	[45]
预测成本	样本标记耗时长	对现实世界中的代码进行标记复杂且耗时	[25,141]
	模型训练耗时长	模型训练样本多、耗时长	[12]

(1) 模型预测能力方面的挑战

现有的软件漏洞预测研究大多都集中在 C/C++语言上, 而针对 C/C++语言设计的漏洞预测模型能否扩展到其他语言仍然是未知的[32]. 由于不同类型的漏洞具有不同的特征, 需要不同的分析方法, 使用训练好的模型未必能够预测出其他类型的漏洞[104]. 同时, 现有的许多漏洞预测方法的预测粒度都侧重于文件或函数层面, 无法准确定位到漏洞存在的具体位置[33]. 这使得软件工程师需要费时费力地分析整个文件或函数, 对易受攻击的语句进行人工定位, 提高了漏洞预测任务的成本[129].

由于来自不同软件项目的代码特征可能不同, 特征在不同项目之间缺乏可泛化性, 跨域漏洞预测问题会直接对模型的泛化性能造成较大影响, 甚至使得模型完全无法使用[42,95]. 进行跨域漏洞预测需要从不同功能的函数、不同编程风格的代码片段中提取共同点. 有至少 5 篇文献[12,35,36,43,81]对跨域漏洞预测问题进行了研究. 总体来看, 使用 RNN 进行迁移学习、尝试捕获漏洞所具有的内在模式、使用不同来源的数据集进行模型训练等研究方向均有助于跨域漏洞预测问题的解决.

在实际应用中, 由于超参数调优的困难性, 各项研究设计出的漏洞预测模型的超参数可能无法使模型获得最佳的性能[49]. 这限制了对模型能够获得的性能的评估及比较. 漏洞的复杂程度可能在一定程度上影响漏洞预测模型的性能. 部分自动化漏洞预测方法对具有复杂的数据流和控制流关系、与多个文件相关联的大型漏洞无能为力[89], 这限制了各个漏洞预测模型在工业生产中的实际应用.

近年来, 大型语言模型开始尝试被应用于软件漏洞预测领域. 然而, ChatGPT 等大型语言模型并非为了软件漏洞预测任务而设计, 故无法保证其训练集中包含大量与软件漏洞相关的内容[66]. 软件漏洞往往涉及代码中存在的细微差别, 而语言模型可能不了解代码, 从而无法对代码细节进行捕获[68]. 另外, 对语言模型进行提示、要求语言模型进行反思等操作均可能导致语言模型给出不同结果[109]. 这说明 GPT 等语言模型可能并未真正理解代码漏洞所具有的语义特征, 对其能否在软件漏洞预测任务上获得较高性能提出了挑战.

(2) 数据集质量方面的挑战

在现实中, 来自真实软件的漏洞数据集往往包含不同形式、不同风格、结构复杂的代码片段, 覆盖面较广, 学习难度较大. 而人工合成的数据集则通常由一批符合特定结构的代码经过一系列变换批量生成, 格式较为固定, 变化不大. 因此, 使用人工合成数据集训练得到的模型可能难以在真实数据集上拥有很好的泛化性能[129]. 总体来看, 人工合成的数据集无法代表真实情况、具有较低的质量等问题被较多文献[41,42,77]提及. 部分研究[25,51,100]认为, 需要将现有的工作在真实数据集上进行评估, 以确定能够获得的实际性能.

由于人工标记时的疏忽、自动标记时的错误等问题, 漏洞数据集中可能会存在被错误标记的样本, 从而导致模型的性能受到影响[125]. 数据集中样本的错误标记严重影响了现有数据集的质量, 缺乏带有标记的数据集一直是这一领域中的挑战[18]. 同时, 由于部分数据来源或数据获取操作中存在的问题, 数据集中可能存在大量相似甚至

重复的样本. 部分研究进行的预处理操作也会导致数据集中代码相似程度的提高. 这将导致模型的性能被过高估计, 甚至产生过拟合问题[16,104].

在真实的软件中, 有漏洞代码数目远小于无漏洞代码数目, 故正例远多于负例的数据不平衡是许多漏洞数据集中常见的问题. 这会导致模型倾向于拟合无漏洞代码, 同时导致许多预测指标无法反映模型的真实性能. 针对汇编语言或二进制代码进行漏洞预测的方法要求源代码能够进行编译. 然而, 许多数据集提供的源代码仅包含函数片段或存在各类语法错误, 具有较大的编译难度, 需要费时费力进行编译[119].

当数据集包含的样本数目太少时, 可能不足以训练出能够学习到漏洞代码内在模式的模型. 部分研究对数据集进行拆分时, 划分出的数据集较小, 从而使得其研究结果难以推广[60]. 现实世界中的软件漏洞预测是根据过去的数据预测未来的漏洞. 因此, 将来自新版本的数据作为训练集、旧版本的数据作为测试集是不符合实际的, 这一问题称为时间旅行问题[141].

(3) 数据集获取方面的挑战

来源于真实世界的数据集缺乏且难以获取一直是软件漏洞预测领域的一大问题. 通常, 需要从开源软件的代码追踪系统出发, 进行数据收集、漏洞代码定位、标记等工作, 才能获取到一批真实数据集. 这一过程通常费时费力. 在各项研究中, 常常存在多人研究团队花费大量时间进行代码标记的情况, 使得软件漏洞预测问题具有极高的成本.

许多数据集的来源为 GitHub、CVE 等代码追踪系统, 但上述数据库中包含的代码及提交通常质量较低[3]. 具有“漏洞修复”标签的提交可能并不仅完成了漏洞修复任务, 还对代码中的其他与漏洞无关的细节进行了修改, 直接使用此类提交进行漏洞代码的判定可能会导致错误, 这一问题称为纠缠提交问题[16]. 纠缠提交现象可能导致严重的标记错误, 与漏洞无关但被错误标记的函数在数据集中的占比最高可达 50%[184]. 各类漏洞修复提交可能不仅无法正确修补漏洞, 反而引入了新的漏洞, 需要仔细进行甄别[19].

通常的漏洞预测研究中都只保留了源代码文件, 但其他类型的文件 (例如 XML 文件) 中同样可能存在与漏洞相关的重要信息, 直接丢弃可能会导致错过某些特定类型的漏洞[41]. 在汇编语言、二进制代码的数据集的获取过程中, 通常需要对源代码进行编译及反汇编操作, 这一过程往往复杂, 这限制了针对汇编语言或二进制代码的漏洞预测任务的研究[28].

(4) 代码表示形式方面的挑战

对于基于标识符的代码表示形式来说, 标识符之间仅存在词汇依赖关系, 缺乏语法上的代表性和语义依赖关系, 后者通常在漏洞预测中起着重要的作用[16]. 对于各类非线性表示形式来说, 对其进行的遍历操作可能会丢失其中包含的信息[18]. 与变量及函数调用过程在语义上相关的代码段可能较大, 易导致表示形式中包含许多与漏洞的存在无关的内容, 这可能会影响神经网络的特征学习过程, 降低预测精度[61].

(5) 代码向量化方式方面的挑战

由于代码中可能出现的标识符数目几乎是无限的, 所有向量化方式均要处理词汇爆炸问题, 仔细思考如何缩小语料空间、高效地进行单词转换[16]. 在标识符处理的过程中, Word2Vec 等单词转换模型无法处理出现在训练集之外的 OOV 单词, 这可能会导致信息丢失[121]. 在向量生成后, 许多研究对生成的向量表示进行截断或填充, 以使得所有语句对应长度相同的向量, 这一过程可能存在信息丢失[104].

(6) 性能评估指标方面的挑战

漏洞数据集存在的数据不平衡、样本相似度高等各种问题将对性能评估指标造成极大影响. 当数据不平衡时, 许多评估指标都可能无法有效地体现出预测模型的实际性能情况. 此时使用单一的性能评估指标 (例如准确率) 对模型性能进行评估是不够的[17,130]. 各项研究所用的评估指标并未阐明模型的假阳性和假阴性情况, 评估范围有限, 存在一定的局限性[17]. 真实世界中的大部分代码并不包含漏洞. 因此, 软件漏洞预测的一个关键问题在于防止大量误报, 而 $F1$ score 无法反映这种不对称性[141].

(7) 可解释性方面的挑战

深度学习驱动的方法的可解释性问题是一个具有极大研究价值的研究领域. 深度神经网络通常被视为一个黑盒模型, 常用的漏洞预测方法在输出预测结果的同时并不会对结果进行解释, 包括为何样本被预测为有漏洞、样

本中的哪些标识符与漏洞的产生相关等, 这些问题值得进行深入的研究和探索. 有至少 9 篇文献[13,25,29,45,78,85,87,150,152]对解释深度学习驱动的漏洞预测方法的输出结果进行了研究. 总体来看, 许多潜在的研究方向与模型的可解释性问题有关, 包括使用注意力模型计算图中节点的注意力分数、探索如何将模型的注意力可视化、构建局部可解释性模型、使用特征选择方法选择与漏洞更相关的标识符、设计新型的可解释评估指标、设计简单的代理模型对模型行为进行模拟等. 这些研究方向均能够为模型可解释性问题的解决提供有益的启示.

(8) 可用性方面的挑战

许多研究中, 为了评估模型的性能, 所使用的基线方法为基于机器学习或深度学习的方法, 这些方法通常来自前人的工作. 然而, 此类方法未必具有良好的可用性. 许多文献未提供所设计出的模型的原始实现, 即使提供了, 所提出的方法也可能有着较大的复现难度. 另外, 复现出的方法所获得的性能可能与文中不一致. 此类问题严重限制了不同漏洞预测模型之间的比较[41,50].

(9) 可判定性方面的挑战

对软件漏洞的特点及性质的判断是一个值得探索的问题. 例如, 漏洞的可利用性是确定漏洞的重要性和影响力的主要标准. 判断一个漏洞是否具有较大的危害, 以及其是否易于被攻击者所利用, 有助于有选择地进行漏洞预测, 降低漏洞预测任务的成本[101]. 不同类型漏洞所需的预测信息量及预测难度不同, 判断哪些类型的漏洞适合使用深度学习驱动的技术进行自动预测, 也是未来的一个很好的研究课题[45].

(10) 漏洞预测成本方面的挑战

为了得到泛化性能好、性能优良的漏洞预测模型, 通常需要大量的标记样本来进行训练. 然而, 对现实世界中的样本进行标记通常复杂且耗时, 导致了极高的标记成本[25]. 数目繁多的样本同时导致了模型训练对系统环境的要求高、所需的时间长、占用内存大等现象, 导致了极高的训练成本[12]. 总体来看, 训练深度学习模型进行漏洞预测依然是一项代价高昂、费时费力的工作.

9.2 未来的重点研究工作

图 9 展示了在上述文献中被提及次数最多的 10 个问题或挑战. 可以观察到, 各项研究的关注点集中在模型的预测能力方面, 其中涵盖了超参数调整、预测其他语言的漏洞、预测其他类型的漏洞等. 其次, 数据集质量方面的问题也引起了关注, 包括样本标记缺失或错误、数据分布不均匀、真实数据集稀缺等. 此外, 模型的可解释性也是一个重要的挑战.

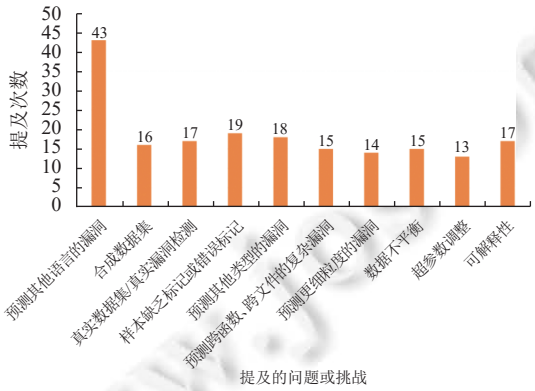


图 9 提及次数最多的 10 个问题或挑战

需要从多方面、多角度对深度学习驱动的漏洞预测领域的各类方法进行全面探索.

(1) 从编程语言来看, 值得进行的研究主要集中在将现有方法扩展到更多编程语言方面. 例如, 开展针对更多编程语言的漏洞预测工作、构建包含更多编程语言代码的漏洞数据集、挖掘不同编程语言内在的共同之处等工作均有助于实现现有方法的扩展. 最终的目标为实现现有的主流编程语言以及机器语言均能够便捷而准确地使用

深度学习技术进行漏洞预测.

(2) 从代码表示形式来看, 评估各类常用表示形式的有效性是一项值得进行的研究工作. 在表示形式中保留跨函数的代码信息、对包含不同信息的表示形式进行组合等工作有助于获得更加有效的代码表示形式. 最终的目标为构建出既适用于各类常用编程语言, 又能够准确地捕捉到大规模代码之间的内在关系, 且使得有漏洞与无漏洞代码易于区分的代码表示形式, 以供预测模型进行特征提取.

(3) 从代码向量化方式来看, 平衡代码中的自然语言信息与代码噪声, 对代码中的自然语言信息进行利用是一个值得探究的问题. 具体地, 应当在何种程度上对代码中的具体数值、变量名称等内容进行抽象这一问题值得更多的思考与探索. 探究不同语料库对不同的向量化方式的影响、更多地关注 FastText 等高效转换模型等工作有助于各项研究选择最合适的向量化方式. 最终的目标为设计出既能够很好地解决词汇爆炸、OOV 单词等问题, 又能很好地保留代码中包含的信息的高效转换方式.

(4) 从漏洞数据集来看, 当前最重要的工作为构建规模更大、包括更多编程语言、涵盖更多漏洞类型的漏洞数据集, 尤其是包含复杂漏洞的大型数据集. 数据集中样本的标记值得重视. 如何定位到包含漏洞的代码、如何解决纠缠提交问题、如何验证数据集中样本标记的准确性等研究问题都直接关系到数据集中的标记质量. 在设计方法解决数据不平衡问题时, 可以探索如何高效地平衡数据集、如何避免不平衡的数据集对预测模型的负面影响等问题. 在真实与合成数据集方面, 首要的是设计出能够高效地对代码库中的提交进行提取解析的方法, 从而能够迅速获得漏洞代码及对应的无漏洞版本, 构建真实数据集. 尝试在真实软件上进行漏洞预测并检测出未知漏洞, 是评估模型实际性能最直接而有效的方式. 如何合成出符合真实软件特点的漏洞代码段、如何高效利用合成数据集等研究问题均有助于对合成数据集进行利用, 缓解现有的数据集缺乏的问题. 最终的目标为设计出简便且高效地构建、评估、优化漏洞数据集的方法, 以及构建出研究人员都能认可并使用的标准数据集.

(5) 从深度学习技术来看, 使用包含不同编程语言的漏洞数据集进行模型训练、分析并提取不同编程语言的共同特征等工作有助于模型预测其他语言的漏洞. 使用包含不同漏洞类型的数据集进行模型训练、针对不同的漏洞类型训练神经网络等工作有助于模型预测其他类型的漏洞. 对提取到的不同维度的特征进行组合、探索更深更宽的神经网络、设计高效的超参数调优方法等工作有助于提高神经网络的性能. 使用注意力机制优化神经网络、探索神经网络内部结构的可解释性等研究问题与深度学习技术的可解释性相关. 对特征向量进行特征选择能够过滤与漏洞无关的内容, 进一步提升各类方法的性能. 使用迁移学习方法提高模型的跨域漏洞预测能力也值得作为重点研究内容. 大型语言模型在软件漏洞预测领域中能够获得的性能需要更多的探索. CNN、RNN 等不同种类的常见神经网络都有自己所擅长的应用领域. 在软件漏洞预测领域, 改造或设计出专门为软件漏洞预测领域服务的、具有强大的特征学习及泛化能力的神经网络是一项有希望实现的目标.

(6) 从性能评估指标来看, 一个主要的问题为确定当前的评估指标能够在何种程度上反映预测模型的实际性能. 这需要结合数据集中样本的数目、样本的相似程度、数据不平衡的程度等因素综合进行考虑. 确定评估指标的有效性和适用范围有助于确定不同的场景下应当选择何种评估指标. 组合使用多种评估指标, 并根据期望取得的预测效果赋予不同评估指标不同的权重能够提高评估结果的有效性. 使用统计显著性检验和效应大小检验等手段能够消除随机性导致的偏差. 最终的目标为设计出简便且高效地确定漏洞预测模型的真实性能的评估方法, 以及确定研究人员都能认可并使用的标准评估指标或指标组合.

总体来看, 目前深度学习驱动的漏洞预测领域还处于发展上升阶段, 各项研究间尚未在源代码表示形式、向量化方式、所使用的数据集及数据集的获取途径、所使用的深度学习技术、所使用的性能评估指标等方面完全达成一致. 在设计及评估漏洞预测方法的各个阶段均存在许多值得思考及探索的问题, 值得在未来的研究过程中得到关注.

10 研究结果及展望

深度学习驱动的软件漏洞预测是近年来一个新兴的、蓬勃发展的研究领域, 众多的研究团队对这一问题进行

了研究及探索, 取得了丰硕的成果.

在编程语言方面, C/C++语言是被研究次数最多的编程语言, 而 Java、PHP、Swift、C#等其余编程语言以及汇编、二进制等机器语言被研究的次数很少. 可以从语言特性、实际应用、漏洞分布这几个角度解释这一现象. 编程语言的选择存在大部分文献的研究主要集中在 C/C++语言上、现有方法在其他编程语言上的泛化能力未知、对于汇编语言或二进制代码等非源代码形式进行的漏洞预测研究需要更多的重视等问题. 未来的研究可以集中在将现有方法扩展到更多编程语言这一方面. 最终的目标为实现现有的主流编程语言以及机器语言均能够便捷而准确地使用深度学习技术进行漏洞预测.

在代码的表示形式方面, 常用的表示形式主要包括原始代码、树结构、图结构、代码切片等. 图结构主要包括 CFG、DFG、PDG、SPG 等, 其中均包含代码中存在的数据流或控制流等信息. 在所有表示形式中, 原始代码是使用最为广泛的表示形式之一, 基于图结构的方法的占比迅速上升, 正在得到越来越广泛的应用. 代码表示形式的选择存在标识符之间的数据流及控制流关系缺乏、对非线性结构的表示形式的遍历存在信息丢失问题、许多表示形式包含与漏洞存在无关的信息等问题. 未来的研究可以集中在分析现有表示形式的有效性、探索性能更优的表示形式等方面. 最终的目标为构建出既具有普适性、又具有高准确性的代码表示形式, 以及精确的、易于相互区分的有漏洞及无漏洞代码的表示形式.

在代码的向量化方式方面, 常用向量化方式主要包括直接映射、编码、单词转换模型、语句转换模型、深度神经网络等. 编码方法主要包括 one-hot 编码、特征张量编码、VecG 等, 单词转换模型主要包括 Word2Vec、GloVe、FastText 等, 语句转换模型主要包括 Sent2Vec、Doc2Vec、Node2Vec 等, 深度神经网络主要包括 Transformer、BERT、CodeBERT 等基于注意力的神经网络. Word2Vec 是使用最广泛的向量化方式, 但其存在性能缺陷. 向量化方式的选择存在填充和截断向量的操作可能导致信息丢失、对词汇外单词的处理存在信息丢失问题、映射和编码方法忽略了有价值的信息等问题. 未来的研究可以集中在平衡自然语言信息与代码噪声、选择最合适的向量化方式等方面. 最终的目标为设计出既能解决各种问题, 又能很好地保留代码中包含的信息的高效转换方式.

在深度学习技术方面, 常用的深度学习技术主要包括卷积神经网络、循环神经网络、注意力神经网络、双向神经网络等线性神经网络, 以及 GRU、GGNN、GCN 等图神经网络. 近年来, 各项研究开始尝试使用大型语言模型进行漏洞检测. 使用图结构作为代码表示形式时, 通常需要按照节点进行向量化后使用图神经网络进行学习. 大部分文献使用 CNN、Bi-LSTM 等线性神经网络, 使用图神经网络的方法正在得到越来越多的应用. 深度学习技术的选择存在线性神经网络本质上是在学习与漏洞存在不相关的特征、神经网络的能力存在一定限制、现有的神经网络模型忽略了一些有用的源代码信息等问题. 未来的研究可以集中在提高现有神经网络的性能、探索深度学习技术的可解释性、跨域漏洞预测等方面, 同时确定大型语言模型对漏洞的预测能力, 向着具有更强的特征学习及泛化能力的神经网络不断前进.

在漏洞数据集方面, 使用来源于 NVD、SARD、Juliet test suite 等大型漏洞数据库、数据集的代码, 或是从 GitHub、CWE 等常用的代码追踪网站提取代码并进行标记, 是各项研究获取数据集的主要途径. 在这一基础上, 部分研究构建了自己的大型漏洞数据集. 这些数据集具有不同的大小、粒度, 包含不同类型的软件漏洞. 部分研究使用先前研究所构建的经典数据集. SARD 及 Juliet test suite 是最广泛的数据集来源, 但其存在明显的质量问题. VulDeePecker、SySeVR、Devign、Big-Vul 等数据集为被使用次数较多的经典数据集. 少量研究在真实软件上进行检测, 并检测出了从未被披露过的未知漏洞. 数据集的选择存在现有数据集在极大程度上偏向 C/C++语言、数据集的质量参差不齐、高效的漏洞数据集获取手段缺乏等问题. 未来的研究可以集中在构建更大更细粒度的漏洞数据集、设计方法解决纠缠提交问题、对现有各数据集进行优化等方面. 最终的目标为设计出简便且高效地构建、评估、优化漏洞数据集的方法, 以及尝试构建出研究人员都能认可并使用的标准数据集.

在评估指标方面, 各项研究通常使用准确率、召回率、精度以及 $F1$ score 这 4 种指标, 在混淆矩阵的基础上对模型的性能进行评估. 另外, *FPR*、*FNR*、top-k 精度、top-k 召回率、*MCC* 等指标均得到了应用. 评估指标的选择存在数据集的特点会影响评估指标的有效性、使用单一指标不足以准确评估模型的性能、现有方法所用指标的评价范围有限等问题. 未来的研究可以集中在确定评估指标的有效性和适用范围、为不同评估指标赋予权重等

方面. 最终的目标为设计出能够简便且高效地确定漏洞预测模型的真实性能的评估方法, 以及确定研究人员都能认可并使用的标准评估指标或指标组合.

在研究遇到的问题及挑战方面, 各项研究提及的问题及挑战分布在模型的预测能力、漏洞数据集的质量及获取、代码的表示形式、可用性、可解释性等方面. 其中, 模型的预测能力、漏洞数据集、模型的可解释性这几个方面的问题与挑战被提及的次数较多. 数据不平衡、跨域漏洞预测、模型的可解释性等挑战被众多文献关注并进行了深入研究, 许多针对这些问题的新思路、新方法不断涌现. 未来的研究可以集中在各项研究普遍提及的问题与挑战上, 同时不断提出并思考新的问题, 持续推动这一领域的发展进步.

总之, 目前深度学习驱动的漏洞预测领域还处于发展上升阶段, 各个方面均存在着许多问题与挑战, 各项研究在许多关键性的问题上尚未达成一致. 这一领域中还有许多内容值得在未来的工作中进行探索. 我们期望, 随着更加简单易行、性能更加优良的方法的不断提出, 研究人员在各个主要步骤上均能够确定标准的操作. 最终能够建立起一套统一的深度学习驱动的软件漏洞预测框架, 使得后续这一领域的研究均能够易于进行、易于评估、能够取得更好的成果. 在能够实现精准漏洞预测后, 后续研究的重点是如何自动修复漏洞[188]和主动堵漏[189−191], 从而提升软件系统的安全性.

11 总 结

软件漏洞预测问题仍然是软件工程及安全领域的一大研究热点. 本文对深度学习驱动的软件漏洞预测问题进行了综述, 分析总结了 151 项研究所针对的编程语言、所使用的代码表示形式及代码向量化方式、所使用的深度学习技术、所使用的漏洞数据集及评估指标、研究过程中遇到的问题及挑战等内容, 分析了这一领域未来的重点研究工作, 为后续研究提供了方向上的参考.

References:

[1] Sen R, Choobineh J, Kumar S. Determinants of software vulnerability disclosure timing. Production and Operations Management, 2020, 29(11): 2532–2552. [doi: 10.1111/poms.13120]

[2] Shin Y, Meneely A, Williams L, Osborne JA. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE Trans. on Software Engineering, 2011, 37(6): 772–787. [doi: 10.1109/TSE.2010.81]

[3] Russell R, Kim L, Hamilton L, Lazovich T, Harer J, Ozdemir O, Ellingwood P, McConley M. Automated vulnerability detection in source code using deep representation learning. In: Proc. of the 17th IEEE Int'l Conf. on Machine Learning and Applications. Orlando: IEEE, 2018. 757–762. [doi: 10.1109/ICMLA.2018.00120]

[4] Renaud N, Geng CL, Georgievska S, Ambrosetti F, Ridder L, Marzella DF, Réau MF, Bonvin AMJJ, Xue LC. DeepRank: A deep learning framework for data mining 3D protein-protein interfaces. Nature Communications, 2021, 12(1): 7068. [doi: 10.1038/s41467-021-27396-0]

[5] Wang DY, Su JL, Yu HB. Feature extraction and analysis of natural language processing for deep learning English language. IEEE Access, 2020, 8: 46335–46345. [doi: 10.1109/ACCESS.2020.2974101]

[6] Jin XJ, Che J, Chen Y. Weed identification using deep learning and image processing in vegetable plantation. IEEE Access, 2021, 9: 10940–10950. [doi: 10.1109/ACCESS.2021.3050296]

[7] Shin Y, Williams L. Can traditional fault prediction models be used for vulnerability prediction? Empirical Software Engineering, 2013, 18(1): 25–59. [doi: 10.1007/s10664-011-9190-8]

[8] Liu SG, Lin GJ, Han QL, Wen S, Zhang J, Xiang Y. DeepBalance: Deep-learning and fuzzy oversampling for vulnerability detection. IEEE Trans. on Fuzzy Systems, 2020, 28(7): 1329–1343. [doi: 10.1109/TFUZZ.2019.2958558]

[9] Wang XM, Zhang T, Wu RP, Xin W, Hou CY. CPGVA: Code property graph based vulnerability analysis by deep learning. In: Proc. of the 10th Int'l Conf. on Advanced Infocomm Technology. Stockholm: IEEE, 2018. 184–188. [doi: 10.1109/ICAIT.2018.8686548]

[10] Kim J, Hubczenko D, Montague P. Towards attention based vulnerability discovery using source code representation. In: Proc. of the 28th Int'l Conf. on Artificial Neural Networks Artificial Neural Networks and Machine Learning. Munich: Springer, 2019. 731–746. [doi: 10.1007/978-3-030-30490-4_58]

[11] Cao DF, Huang J, Zhang XY, Liu XH. FTCLNet: Convolutional LSTM with Fourier transform for vulnerability detection. In: Proc. of the 19th IEEE Int'l Conf. on Trust, Security and Privacy in Computing and Communications. Guangzhou: IEEE, 2020. 539–546. [doi:

10.1109/TrustCom50675.2020.00078]

[12] Lin GJ, Zhang J, Luo W, Pan L, Xiang Y, De Vel O, Montague P. Cross-project transfer representation learning for vulnerable function discovery. IEEE Trans. on Industrial Informatics, 2018, 14(7): 3289–3297. [doi: 10.1109/TII.2018.2821768]

[13] Mao Y, Li Y, Sun JT, Chen YX. Explainable software vulnerability detection based on attention-based bidirectional recurrent neural networks. In: Proc. of the 2020 IEEE Int'l Conf. on Big Data. Atlanta: IEEE, 2020. 4651–4656. [doi: 10.1109/BigData50022.2020.9377803]

[14] Singh SK, Chaturvedi A. Applying deep learning for discovery and analysis of software vulnerabilities: A brief survey. In: Pant M, Sharma TK, Arya R, Sahana BC, Zolfagharinia H, eds. Soft Computing: Theories and Applications. Singapore: Springer, 2020. 649–658. [doi: 10.1007/978-981-15-4032-5_59]

[15] Kubiuk Y, Kyselov G. Comparative analysis of approaches to source code vulnerability detection based on deep learning methods. Technology Audit and Production Reserves, 2021, 3(59): 19–23. [doi: 10.15587/2706-5448.2021.233534]

[16] Chakraborty S, Krishna R, Ding Y, Ray B. Deep learning based vulnerability detection: Are we there yet? IEEE Trans. on Software Engineering, 2022, 48(9): 3280–3296.

[17] Alaoui RL, Nfaoui EH. Deep learning for vulnerability and attack detection on Web applications: A systematic literature review. Future Internet, 2022, 14(4): 118. [doi: 10.3390/fi14040118]

[18] Zeng P, Lin GJ, Pan L, Tai YH, Zhang J. Software vulnerability analysis and discovery using deep learning techniques: A survey. IEEE Access, 2020, 8: 197158–197172. [doi: 10.1109/ACCESS.2020.3034766]

[19] 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. [doi: 10.1016/j.infsof.2021.106809]

[20] Zhang X, Sun HY, He ZP, Gu MX, Feng JY, Zhang YQ. VDBWGDL: Vulnerability detection based on weight graph and deep learning. In: Proc. of the 52nd Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks Workshops. Baltimore: IEEE, 2022. 186–190. [doi: 10.1109/DSN-W54100.2022.00039]

[21] Zhou L, Huang MH, Li YJ, Nie YP, Li J, Liu YW. GraphEye: A novel solution for detecting vulnerable functions based on graph attention network. In: Proc. of the 6th IEEE Int'l Conf. on Data Science in Cyberspace. Shenzhen: IEEE, 2021. 381–388. [doi: 10.1109/DSC53577.2021.00060]

[22] Cao SC, Sun XB, Bo LL, Wu RX, Li B, Tao CQ. MVD: Memory-related vulnerability detection based on flow-sensitive graph neural networks. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 1456–1468. [doi: 10.1145/3510003.3510219]

[23] Gong HH, Ma SQ, Seyit C, Surya N, Xu C. Vulnerability detection using deep learning based function classification. In: Proc. of the 16th Int'l Conf. on Network and System Security. Denarau Island: Springer, 2022. 3–22. [doi: 10.1007/978-3-031-23020-2_1]

[24] Kim S, Choi J, Ahmed ME, Nepal S, Kim H. VulDeBERT: A vulnerability detection system using BERT. In: Proc. of the 2022 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Charlotte: IEEE, 2022. 69–74. [doi: 10.1109/ISSREW55968.2022.00042]

[25] Yan GQ, Chen S, Bail Y, Li XH. Can deep learning models learn the vulnerable patterns for vulnerability detection? In: Proc. of the 46th IEEE Annual Computers, Software, and Applications Conf. Los Alamitos: IEEE, 2022. 904–913. [doi: 10.1109/COMPSAC54236.2022.00142]

[26] Lin C, Xu YJ, Fang Y, Liu ZL. VulEye: A novel graph neural network vulnerability detection approach for PHP application. Applied Sciences, 2023, 13(2): 825. [doi: 10.3390/app13020825]

[27] Dong YK, Tang YE, Cheng XT, Yang YF, Wang SQ. SedSVD: Statement-level software vulnerability detection based on relational graph convolutional network with subgraph embedding. Information and Software Technology, 2023, 158: 107168. [doi: 10.1016/j.infsof.2023.107168]

[28] Wu GL, Tang HL. Binary code vulnerability detection based on multi-level feature fusion. IEEE Access, 2023, 11: 63904–63915. [doi: 10.1109/ACCESS.2023.3289001]

[29] Hu YT, Wang SY, Wu YM, Zou DQ, Li WK, Jin H. Slice-level vulnerability detection and interpretation method based on graph neural network. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2543–2561 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6849.htm [doi: 10.13328/j.cnki.jos.006849]

[30] Alagöz Zİ, Akleylek S. A brief review on deep learning based software vulnerability detection. In: Proc. of the 2021 Int'l Conf. on Information Security and Cryptology. Ankara: IEEE, 2021. 143–148. [doi: 10.1109/ISCTURKEY53027.2021.9654351]

[31] Li Z, Zou DQ, Tang J, Zhang ZH, Sun MQ, Jin H. A comparative study of deep learning-based vulnerability detection system. IEEE Access, 2019, 7: 103184–103197. [doi: 10.1109/ACCESS.2019.2930578]

[32] Zagane M, Abdi MK, Alenezi M. Deep learning for software vulnerabilities detection using code metrics. IEEE Access, 2020, 8:

74562–74570. [doi: 10.1109/ACCESS.2020.2988557]

[33] Li RH, Feng C, Zhang X, Tang CJ. A lightweight assisted vulnerability discovery method using deep neural networks. IEEE Access, 2019, 7: 80079–80092. [doi: 10.1109/ACCESS.2019.2923227]

[34] Bilgin Z, Ersoy MA, Soykan EU, Tomur E, Çomak P, Karaçay L. Vulnerability prediction from source code using machine learning. IEEE Access, 2020, 8: 150672–150684. [doi: 10.1109/ACCESS.2020.3016774]

[35] Liu SG, Lin GJ, Qu LZ, Zhang J, De Vel O, Montague P, Xiang Y. CD-VulD: Cross-domain vulnerability discovery based on deep domain adaptation. IEEE Trans. on Dependable and Secure Computing, 2022, 19(1): 438–451. [doi: 10.1109/TDSC.2020.2984505]

[36] Lin GJ, Zhang J, Luo W, Pan L, De Vel O, Montague P, Xiang Y. Software vulnerability discovery via learning multi-domain knowledge bases. IEEE Trans. on Dependable and Secure Computing, 2021, 18(5): 2469–2485. [doi: 10.1109/TDSC.2019.2954088]

[37] Li Z, Zou DQ, Xu SH, Jin H, Zhu YW, Chen ZX. SySeVR: A framework for using deep learning to detect software vulnerabilities. IEEE Trans. on Dependable and Secure Computing, 2022, 19(4): 2244–2258. [doi: 10.1109/TDSC.2021.3051525]

[38] Li Z, Zou DQ, Xu SH, Chen ZX, Zhu YW, Jin H. VulDeeLocator: A deep learning-based fine-grained vulnerability detector. IEEE Trans. on Dependable and Secure Computing, 2022, 19(4): 2821–2837. [doi: 10.1109/TDSC.2021.3076142]

[39] Zou DQ, Wang SJ, Xu SH, Li Z, Jin H. μVulDeePecker: A deep learning-based system for multiclass vulnerability detection. IEEE Trans. on Dependable and Secure Computing, 2021, 18(5): 2224–2236. [doi: 10.1109/TDSC.2019.2942930]

[40] Wang HT, Ye GX, Tang ZY, Tan SH, Huang SF, Fang DY, Feng YS, Bian LZ, Wang Z. Combining graph-based learning with automated data collection for code vulnerability detection. IEEE Trans. on Information Forensics and Security, 2021, 16: 1943–1958. [doi: 10.1109/TIFS.2020.3044773]

[41] Dam HK, Tran T, Pham T, Ng SW, Grundy J, Ghose A. Automatic feature learning for predicting vulnerable software components. IEEE Trans. on Software Engineering, 2021, 47(1): 67–85. [doi: 10.1109/TSE.2018.2881961]

[42] Nong Y, Sharma R, Hamou-Lhadj A, Luo XP, Cai HP. Open science in software engineering: A study on deep learning-based vulnerability detection. IEEE Trans. on Software Engineering, 2023, 49(4): 1983–2005. [doi: 10.1109/TSE.2022.3207149]

[43] Zhang CY, Liu B, Xin Y, Yao LW. CPVD: Cross project vulnerability detection based on graph attention network and domain adaptation. IEEE Trans. on Software Engineering, 2023, 49(8): 4152–4168. [doi: 10.1109/TSE.2023.3285910]

[44] Feng HT, Fu XT, Sun HY, Wang H, Zhang YQ. Efficient vulnerability detection based on abstract syntax tree and deep learning. In: Proc. of the 2020 IEEE Conf. on Computer Communications Workshops. Toronto: IEEE, 2020. 722–727. [doi: 10.1109/INFOCOMWKSHPS50562.2020.9163061]

[45] Gu MX, Feng HT, Sun HY, Liu P, Yue QL, Hu JL, Cao CJ, Zhang YQ. Hierarchical attention network for interpretable and fine-grained vulnerability detection. In: Proc. of the 2022 IEEE Conf. on Computer Communications Workshops. New York: IEEE, 2022. 1–6. [doi: 10.1109/INFOCOMWKSHPS54753.2022.9798297]

[46] Ziems N, Wu SE. Security vulnerability detection using deep learning natural language processing. In: Proc. of the 2021 IEEE Conf. on Computer Communications Workshops. Vancouver: IEEE, 2021. 1–6. [doi: 10.1109/INFOCOMWKSHPS51825.2021.9484500]

[47] Rabheru R, Hanif H, Maffeis S. A hybrid graph neural network approach for detecting PHP vulnerabilities. In: Proc. of the 2022 IEEE Conf. on Dependable and Secure Computing. Edinburgh: IEEE, 2022. 1–9. [doi: 10.1109/DSC54232.2022.9888816]

[48] Cheng X, Wang HY, Hua JY, Zhang M, Xu GA, Li Y, Sui YL. Static detection of control-flow-related vulnerabilities using graph embedding. In: Proc. of the 24th Int'l Conf. on Engineering of Complex Computer Systems. Guangzhou: IEEE, 2019. 41–50. [doi: 10.1109/ICECCS.2019.00012]

[49] Hin D, Kan A, Chen HM, Babar MA. LineVD: Statement-level vulnerability detection using graph neural networks. In: Proc. of the 19th Int'l Conf. on Mining Software Repositories. Pittsburgh: ACM, 2022. 596–607. [doi: 10.1145/3524842.3527949]

[50] Fu M, Tantithamthavorn C. LineVul: A Transformer-based line-level vulnerability prediction. In: Proc. of the 19th Int'l Conf. on Mining Software Repositories. Pittsburgh: IEEE, 2022. 608–620. [doi: 10.1145/3524842.3528452]

[51] Saccente N, Dehlinger J, Deng L, Chakraborty S, Xiong Y. Project Achilles: A prototype tool for static method-level vulnerability detection of Java source code using a recurrent neural network. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering Workshop. San Diego: IEEE, 2019. 114–121. [doi: 10.1109/ASEW.2019.00040]

[52] Wu TS, Chen LW, Du GWZ, Zhu CG, Shi G. Self-attention based automated vulnerability detection with effective data representation. In: Proc. of the 2021 IEEE Int'l Conf. on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking. New York: IEEE, 2021. 892–899. [doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00126]

[53] An WY, Chen LW, Wang JX, Du GWZ, Shi G, Meng D. AVDHRAM: Automated vulnerability detection based on hierarchical representation and attention mechanism. In: Proc. of the 2020 IEEE Int'l Conf. on Parallel & Distributed Processing with Applications,

Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking. Exeter: IEEE, 2020. 337–344. [doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom51426.2020.00068]

[54] Mazuera-Rozo A, Mojica-Hanke A, Linares-Vásquez M, Bavota G. Shallow or deep? An empirical study on detecting vulnerabilities using deep learning. In: Proc. of the 29th IEEE/ACM Int'l Conf. on Program Comprehension. Madrid: IEEE, 2021. 276–287. [doi: 10.1109/ICPC52881.2021.00034]

[55] Lin GJ, Wen S, Han QL, Zhang J, Xiang Y. Software vulnerability detection using deep neural networks: A survey. Proc. of the IEEE, 2020, 108(10): 1825–1848. [doi: 10.1109/JPROC.2020.2993293]

[56] Zhang HJ, Li YJ, Liu YW, Zhou NX. Vulmg: A static detection solution for source code vulnerabilities based on code property graph and graph attention network. In: Proc. of the 18th Int'l Computer Conf. on Wavelet Active Media Technology and Information Processing. Chengdu: IEEE, 2021. 250–255. [doi: 10.1109/ICCWAMTIP53232.2021.9674145]

[57] Xu AD, Dai T, Chen HJ, Ming Z, Li WN. Vulnerability detection for source code using contextual LSTM. In: Proc. of the 5th Int'l Conf. on Systems and Informatics. Nanjing: IEEE, 2018. 1225–1230. [doi: 10.1109/ICSAI.2018.8599360]

[58] Wu F, Wang JG, Liu JQ, Wang W. Vulnerability detection with deep learning. In: Proc. of the 3rd IEEE Int'l Conf. on Computer and Communications. Chengdu: IEEE, 2017. 1298–1302. [doi: 10.1109/CompComm.2017.8322752]

[59] Wang ZY, Guo JJ, Li HN. Vulnerability feature extraction model for source code based on deep learning. In: Proc. of the 2021 Int'l Conf. on Computer Network, Electronic and Automation. Xi'an: IEEE, 2021. 21–25. [doi: 10.1109/ICCNEA53019.2021.00016]

[60] Ding YRB, Suneja S, Zheng YH, Laredo J, Morari A, Kaiser G, Ray B. VELVET: A novel ensemble learning approach to automatically locate vulnerable statements. In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2022. 959–970. [doi: 10.1109/SANER53432.2022.00114]

[61] Zheng WN, Jiang Y, Su XH. Vu1SPG: Vulnerability detection based on slice property graph representation learning. In: Proc. of the 32nd IEEE Int'l Symp. on Software Reliability Engineering. Wuhan: IEEE, 2021. 457–467. [doi: 10.1109/ISSRE52982.2021.00054]

[62] Nguyen VA, Nguyen DQ, Nguyen V, Le T, Tran QH, Phung D. ReGVD: Revisiting graph neural networks for vulnerability detection. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. Pittsburgh: IEEE, 2022. 178–182. [doi: 10.1145/3510454.3516865]

[63] Steenhoek B, Rahman M, Jiles R, Le W. An empirical study of deep learning models for vulnerability detection. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 2237–2248. [doi: 10.1109/ICSE48619.2023.00188]

[64] Steenhoek B, Gao HY, Le W. Dataflow analysis-inspired deep learning for efficient vulnerability detection. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: IEEE, 2024. 177–189. [doi: 10.1145/3597503.3623345]

[65] Zhang CY, Liu H, Zeng JT, Yang KJ, Li YH, Li H. Prompt-enhanced software vulnerability detection using ChatGPT. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. Lisbon: IEEE, 2024. 276–277. [doi: 10.1145/3639478.3643065]

[66] Zhou X, Zhang T, Lo D. Large language model for vulnerability detection: Emerging results and future directions. In: Proc. of the 46th IEEE /ACM Int'l Conf. on Software Engineering: New Ideas and Emerging Results. Lisbon: IEEE, 2024. 47–51. [doi: 10.1145/3639476.3639762]

[67] Wang WB, Nguyen TN, Wang SH, Li Y, Zhang JY, Yadavally A. DeepVD: Toward class-separation features for neural network vulnerability detection. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 2249–2261. [doi: 10.1109/ICSE48619.2023.00189]

[68] Omar M. VulDefend: A novel technique based on pattern-exploiting training for detecting software vulnerabilities using language models. In: Proc. of the 2023 IEEE Jordan Int'l Joint Conf. on Electrical Engineering and Information Technology. Amman: IEEE, 2023. 287–293. [doi: 10.1109/JEEIT58638.2023.10185860]

[69] Omar M, Shiaeles S. VulDetect: A novel technique for detecting software vulnerabilities using language models. In: Proc. of the 2023 IEEE Int'l Conf. on Cyber Security and Resilience. Venice: IEEE, 2023. 105–110. [doi: 10.1109/CSR57506.2023.10224924]

[70] Zhang JW, Liu ZX, Hu X, Xia X, Li SP. Vulnerability detection by learning from syntax-based execution paths of code. IEEE Trans. on Software Engineering, 2023, 49(8): 4196–4212. [doi: 10.1109/TSE.2023.3286586]

[71] Fu M, Tantithamthavorn CK, Nguyen V, Le T. ChatGPT for vulnerability detection, classification, and repair: How far are we? In: Proc. of the 30th Asia-Pacific Software Engineering Conf. Seoul: IEEE, 2023. 632–636. [doi: 10.1109/APSEC60848.2023.00085]

[72] Seas C, Fitzpatrick G, Hamilton JA, Carlisle MC. Automated vulnerability detection in source code using deep representation learning. In: Proc. of the 14th IEEE Annual Computing and Communication Workshop and Conf. Las Vegas: IEEE, 2024. 484–490. [doi: 10.1109/CCWC60891.2024.10427574]

[73] Wen XC, Gao CY, Ye JX, Li YC, Tian ZH, Jia Y, Wang X. Meta-path based attentional graph learning model for vulnerability

detection. IEEE Trans. on Software Engineering, 2024, 50(3): 360–375. [doi: 10.1109/TSE.2023.3340267]

[74] Islam NT, De La Torre Parra G, Manuel D, Bou-Harb E, Najafirad P. An unbiased Transformer source code learning with semantic vulnerability graph. In: Proc. of the 8th IEEE European Symp. on Security and Privacy. Delft: IEEE, 2023. 144–159. [doi: 10.1109/EuroSP57164.2023.00018]

[75] Du QJ, Kun W, Kuang XH, Li X, Zhao G. Automated software vulnerability detection via curriculum learning. In: Proc. of the 2023 IEEE Int'l Conf. on Multimedia and Expo. Brisbane: IEEE, 2023. 2855–2860. [doi: 10.1109/ICME55011.2023.00485]

[76] Das Purba M, Ghosh A, Radford BJ, Chu B. Software vulnerability detection using large language models. In: Proc. of the 34th IEEE Int'l Symp. on Software Reliability Engineering Workshops. Florence: IEEE, 2023. 112–119. [doi: 10.1109/ISSREW60843.2023.00058]

[77] Wu YM, Zou DQ, Dou SH, Yang W, Xu D, Jin H. VulCNN: An image-inspired scalable vulnerability detection system. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 2365–2376. [doi: 10.1145/3510003.3510229]

[78] Zou DQ, Zhu YW, Xu SH, Li Z, Jin H, Ye HK. Interpreting deep learning-based vulnerability detector predictions based on heuristic searching. ACM Trans. on Software Engineering and Methodology (TOSEM), 2021, 30(2): 23. [doi: 10.1145/3429444]

[79] Aumpansub A, Huang Z. Detecting software vulnerabilities using neural networks. In: Proc. of the 13th Int'l Conf. on Machine Learning and Computing. Shenzhen: ACM, 2021. 166–171. [doi: 10.1145/3457682.3457707]

[80] Cheng X, Wang HY, Hua JY, Xu GA, Sui YL. DeepWukong: Statically detecting software vulnerabilities using deep graph neural network. ACM Trans. on Software Engineering and Methodology (TOSEM), 2021, 30(3): 38. [doi: 10.1145/3436877]

[81] Lin GJ, Zhang J, Luo W, Pan L, Xiang Y. POSTER: Vulnerability discovery with function representation learning from unlabeled projects. In: Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security. Dallas: ACM, 2017. 2539–2541. [doi: 10.1145/3133956.3138840]

[82] Pang YL, Xue XZ, Wang HY. Predicting vulnerable software components through deep neural network. In: Proc. of the 2017 Int'l Conf. on Deep Learning Technologies. Chengdu: ACM, 2017. 6–10. [doi: 10.1145/3094243.3094245]

[83] Zheng JY, Pang JM, Zhang XC, Zhou X, Li ML, Wang J. Recurrent neural network based binary code vulnerability detection. In: Proc. of the 2nd Int'l Conf. on Algorithms, Computing and Artificial Intelligence. Sanya: ACM, 2019. 160–165. [doi: 10.1145/3377713.3377738]

[84] Alqarni M, Azim A. Software source code vulnerability detection using advanced deep convolutional neural network. In: Proc. of the 31st Annual Int'l Conf. on Computer Science and Software Engineering. Toronto: IBM Corp., 2021. 226–231.

[85] Li Y, Wang SH, Nguyen TN. Vulnerability detection with fine-grained interpretations. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 292–303. [doi: 10.1145/3468264.3468597]

[86] Pechenkin A, Demidov R. Applying deep learning and vector representation for software vulnerabilities detection. In: Proc. of the 11th Int'l Conf. on Security of Information and Networks. Cardiff: ACM, 2018. 13. [doi: 10.1145/3264437.3264489]

[87] Li LT, Ding SHH, Tian Y, Fung BCM, Charland P, Ou WH, Song L, Chen CW. VulANalyzeR: Explainable binary vulnerability detection with multi-task learning and attentional graph convolution. ACM Trans. on Privacy and Security, 2023, 26(3): 28. [doi: 10.1145/3585386]

[88] Lin GJ, Xiao W, Zhang J, Xiang Y. Deep learning-based vulnerable function detection: A benchmark. In: Proc. of the 21st Int'l Conf. on Information and Communications Security. Beijing: Springer, 2020. 219–232. [doi: 10.1007/978-3-030-41579-2_13]

[89] Guo N, Li XY, Yin H, Gao YL. VulHunter: An automated vulnerability detection system based on deep learning and bytecode. In: Proc. of the 21st Int'l Conf. on Information and Communications Security. Beijing: Springer, 2020. 199–218. [doi: 10.1007/978-3-030-41579-2_12]

[90] Lin GJ, Xiao W, Zhang LY, Gao S, Tai YH, Zhang J. Deep neural-based vulnerability discovery demystified: Data, model and performance. Neural Computing and Applications, 2021, 33(20): 13287–13300. [doi: 10.1007/s00521-021-05954-3]

[91] Şahin CB, Abualigah L. A novel deep learning-based feature selection model for improving the static analysis of vulnerability detection. Neural Computing and Applications, 2021, 33(20): 14049–14067. [doi: 10.1007/s00521-021-06047-x]

[92] Nguyen T, Le T, Nguyen K, de Vel O, Montague P, Grundy J, Phung D. Deep cost-sensitive kernel machine for binary software vulnerability detection. In: Proc. of the 24th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining. Singapore: Springer, 2020. 164–177. [doi: 10.1007/978-3-030-47436-2_13]

[93] Filus K, Siavvas M, Domańska J, Gelenbe E. The random neural network as a bonding model for software vulnerability prediction. In: Proc. of the 28th Int'l Symp. on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems. Nice: Springer, 2021. 102–116. [doi: 10.1007/978-3-030-68110-4_7]

[94] Guo JJ, Wang ZY, Li HN, Xue Y. Detecting vulnerability in source code using CNN and LSTM network. Soft Computing, 2023, 27(2):

1131–1141. [doi: 10.1007/s00500-021-05994-w]

[95] Napier K, Bhowmik T, Wang SW. An empirical study of text-based machine learning models for vulnerability detection. Empirical Software Engineering, 2023, 28(2): 38. [doi: 10.1007/s10664-022-10276-6]

[96] Kalouptsoglou I, Siavvas M, Kehagias D, Chatzigeorgiou A, Ampatzoglou A. An empirical evaluation of the usefulness of word embedding techniques in deep learning-based vulnerability prediction. In: Proc. of the 2nd Int'l Symp. on Security in Computer and Information Sciences. Nice: Springer, 2022. 23–37. [doi: 10.1007/978-3-031-09357-9_3]

[97] Şahin CB. Semantic-based vulnerability detection by functional connectivity of gated graph sequence neural networks. Soft Computing, 2023, 27(9): 5703–5719. [doi: 10.1007/s00500-022-07777-3]

[98] Zhu YH, Lin GJ, Song LP, Zhang J. The application of neural network for software vulnerability detection: A review. Neural Computing and Applications, 2023, 35(2): 1279–1301. [doi: 10.1007/s00521-022-08046-y]

[99] Cao SC, Sun XB, Bo LL, Wei Y, Li B. BGNN4VD: Constructing bidirectional graph neural-network for vulnerability detection. Information and Software Technology, 2021, 136: 106576. [doi: 10.1016/j.infsof.2021.106576]

[100] Tian JF, Xing WJ, Li Z. BVDetector: A program slice-based binary code vulnerability intelligent detection system. Information and Software Technology, 2020, 123: 106289. [doi: 10.1016/j.infsof.2020.106289]

[101] Jeon S, Kim HK. AutoVAS: An automated vulnerability analysis system with a deep learning approach. Computers & Security, 2021, 106: 102308. [doi: 10.1016/j.cose.2021.102308]

[102] Yan H, Luo SL, Pan LM, Zhang YF. HAN-BSVD: A hierarchical attention network for binary software vulnerability detection. Computers & Security, 2021, 108: 102286. [doi: 10.1016/j.cose.2021.102286]

[103] Guo WB, Fang Y, Huang C, Ou HR, Lin C, Guo YY. HyVulDect: A hybrid semantic vulnerability mining system based on graph neural network. Computers & Security, 2022, 121: 102823. [doi: 10.1016/j.cose.2022.102823]

[104] Zheng W, Gao JL, Wu XX, Liu FY, Xun YX, Liu GL, Chen X. The impact factors on the performance of machine learning-based vulnerability detection: A comparative study. Journal of Systems and Software, 2020, 168: 110659. [doi: 10.1016/j.jss.2020.110659]

[105] Tang W, Tang MW, Ban MC, Zhao ZG, Feng MJ. CSGVD: A deep learning approach combining sequence and graph embedding for source code vulnerability detection. Journal of Systems and Software, 2023, 199: 111623. [doi: 10.1016/j.jss.2023.111623]

[106] Wang MK, Tao CQ, Guo HJ. LCVD: Loop-oriented code vulnerability detection via graph neural network. Journal of Systems and Software, 2023, 202: 111706. [doi: 10.1016/j.jss.2023.111706]

[107] Wang J, Xiao H, Zhong SW, Xiao YH. DeepVulSeeker: A novel vulnerability identification framework via code graph structure and pre-training mechanism. Future Generation Computer Systems, 2023, 148: 15–26. [doi: 10.1016/j.future.2023.05.016]

[108] Li X, Xin Y, Zhu HL, Yang YX, Chen YL. Cross-domain vulnerability detection using graph embedding and domain adaptation. Computers & Security, 2023, 125: 103017. [doi: 10.1016/j.cose.2022.103017]

[109] Lu GL, Ju XL, Chen X, Pei WL, Cai ZL. GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning. Journal of Systems and Software, 2024, 212: 112031. [doi: 10.1016/j.jss.2024.112031]

[110] Tao WX, Su XH, Wan JY, Wei HW, Zheng WN. Vulnerability detection through cross-modal feature enhancement and fusion. Computers & Security, 2023, 132: 103341. [doi: 10.1016/j.cose.2023.103341]

[111] Wang Y, Jia P, Peng X, Huang C, Liu JY. BinVulDet: Detecting vulnerability in binary program via decompiled pseudo code and BiLSTM-attention. Computers & Security, 2023, 125: 103023. [doi: 10.1016/j.cose.2022.103023]

[112] Tian ZZ, Tian BH, Lv JJ, Chen YP, Chen LW. Enhancing vulnerability detection via AST decomposition and neural sub-tree encoding. Expert Systems with Applications, 2024, 238: 121865. [doi: 10.1016/j.eswa.2023.121865]

[113] Fan YH, Wan CH, Fu C, Han LS, Xu H. VDoTR: Vulnerability detection based on tensor representation of comprehensive code graphs. Computers & Security, 2023, 130: 103247. [doi: 10.1016/j.cose.2023.103247]

[114] Chen JF, Lin W, Cai SH, Yin YM, Chen HB, Towey D. BiTCN_DRSN: An effective software vulnerability detection model based on an improved temporal convolutional network. Journal of Systems and Software, 2023, 204: 111772. [doi: 10.1016/j.jss.2023.111772]

[115] Cai WJ, Chen JL, Yu JP, Gao LP. A software vulnerability detection method based on deep learning with complex network analysis and subgraph partition. Information and Software Technology, 2023, 164: 107328. [doi: 10.1016/j.infsof.2023.107328]

[116] Song ZH, Wang JF, Yang KY, Wang JG. HGIVul: Detecting inter-procedural vulnerabilities based on hypergraph convolution. Information and Software Technology, 2023, 160: 107219. [doi: 10.1016/j.infsof.2023.107219]

[117] Wu BL, Zou FT, Yi P, Wu Y, Zhang L. SlicedLocator: Code vulnerability locator based on sliced dependence graph. Computers & Security, 2023, 134: 103469. [doi: 10.1016/j.cose.2023.103469]

[118] Zhang CY, Xin Y. VulGAI: Vulnerability detection based on graphs and images. Computers & Security, 2023, 135: 103501. [doi: 10.1016/j.cose.2023.103501]

[119] Li X, Wang L, Xin Y, Yang YX, Tang QF, Chen YL. Automated software vulnerability detection based on hybrid neural network. Applied Sciences, 2021, 11(7): 3201. [doi: 10.3390/app11073201]

[120] Li X, Wang L, Xin Y, Yang YX, Chen YL. Automated vulnerability detection in source code using minimum intermediate representation learning. Applied Sciences, 2020, 10(5): 1692. [doi: 10.3390/app10051692]

[121] Wei HW, Lin GJ, Li L, Jia HM. A context-aware neural embedding for function-level vulnerability detection. Algorithms, 2021, 14(11): 335. [doi: 10.3390/a14110335]

[122] Ban XB, Liu SG, Chen C, Caslon C. A performance evaluation of deep-learnt features for software vulnerability detection. Concurrency and Computation: Practice and Experience, 2019, 31(19): e5103. [doi: 10.1002/cpe.5103]

[123] Abdallah Semasaba AO, Zheng W, Wu XX, Akwasi Agyemang S. Literature survey of deep learning-based vulnerability analysis on source code. IET Software, 2020, 14(6): 654–664. [doi: 10.1049/iet-sen.2020.0084]

[124] Subhan F, Wu XX, Bo LL, Sun XB, Rahman M. A deep learning-based approach for software vulnerability detection using code metrics. IET Software, 2022, 16(5): 516–526. [doi: 10.1049/sfw2.12066]

[125] Abdallah Semasaba AO, Zheng W, Wu XX, Akwasi Agyemang S, Liu T, Ge Y. An empirical evaluation of deep learning-based source code vulnerability detection: Representation versus models. Journal of Software: Evolution and Process, 2023, 35(11): e2422. [doi: 10.1002/smr.2422]

[126] Zeng P, Lin GJ, Zhang J, Zhang Y. Intelligent detection of vulnerable functions in software through neural embedding-based code analysis. Int'l Journal of Network Management, 2023, 33(3): e2198. [doi: 10.1002/nem.2198]

[127] Song ZH, Wang JF, Liu SL, Fang ZY, Yang KY. HGVul: A code vulnerability detection method based on heterogeneous source-level intermediate representation. Security and Communication Networks, 2022, 2022(1): 1919907. [doi: 10.1155/2022/1919907]

[128] Yuan X, Lin GJ, Tai YH, Zhang J. Deep neural embedding for software vulnerability discovery: Comparison and optimization. Security and Communication Networks, 2022, 2022(1): 5203217. [doi: 10.1155/2022/5203217]

[129] Wu BL, Zou FT. Code vulnerability detection based on deep sequence and graph models: A survey. Security and Communication Networks, 2022, 2022(1): 1176898. [doi: 10.1155/2022/1176898]

[130] Nguyen HN, Teerakanok S, Inomata A, Uehara T. The comparison of word embedding techniques in RNNs for vulnerability detection. In: Proc. of the 7th Int'l Conf. on Information Systems Security and Privacy. SCITEPRESS, 2021. 109–120. [doi: 10.5220/0010232301090120]

[131] Xie PS, Wang JL, Wang H, Pan YC, Li XY, Feng T. Industrial Internet vulnerability detection method based on CBAM-CNN-SVM. Int'l Journal of Network Security, 2023, 25(3): 385–393. [doi: 10.6633/IJNS.202305_25(3).01]

[132] Fang Y, Han SJ, Huang C, Wu RP. TAP: A static analysis model for PHP vulnerabilities based on token and deep learning technology. PLoS One, 2019, 14(11): e0225196. [doi: 10.1371/journal.pone.0225196]

[133] Li Z, Zou DQ, Xu SH, Ou XY, Jin H, Wang SJ, Deng ZJ, Zhong YY. VulDeePecker: A deep learning-based system for vulnerability detection. In: Proc. of the 25th Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2018.

[134] Zhou YQ, Liu SQ, Siow J, Du XN, Liu Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 915.

[135] Mirsky Y, Macon G, Brown M, Yagemann C, Pruett M, Downing E, Mertoguno S, Lee W. VulChecker: Graph-based vulnerability localization in source code. In: Proc. of the 32nd USENIX Security Symp. Anaheim: USENIX Association, 2023. 367.

[136] Dou SH, Wu YM, Li WX, Cheng F, Yang W, Liu Y. Boosting the capability of intelligent vulnerability detection by training in a human-learning manner. arXiv:2112.06250, 2021.

[137] Suneja S, Zheng YH, Zhuang YF, Laredo J, Morari A. Learning to map source code to software vulnerability using code-as-a-graph. arXiv:2006.08614, 2020.

[138] Zhuang YF, Suneja S, Thost V, Domeniconi G, Morari A, Laredo J. Software vulnerability detection via deep learning over disaggregated code graph representation. arXiv:2109.03341, 2021.

[139] Wen XC, Chen YP, Gao CY, Zhang HY, Zhang JM, Liao Q. Vulnerability detection with graph simplification and enhanced graph representation learning. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 2275–2286. [doi: 10.1109/ICSE48619.2023.00191]

[140] Cheshkov A, Zadorozhny P, Levichev R. Evaluation of ChatGPT model for vulnerability detection. arXiv:2304.07232, 2023.

[141] Ding YRB, Fu YJ, Ibrahim O, Sitawarin C, Chen XY, Alomair B, Wagner D, Ray B, Chen YZ. Vulnerability detection with code language models: How far are we? arXiv:2403.18624, 2024.

[142] Omar M, Burrell D. From text to threats: A language model approach to software vulnerability detection. Int'l Journal of Mathematics

and Computer in Engineering, 2024, 2(1). [doi: 10.2478/ijmce-2024-0003]

[143] Khare A, Dutta S, Li ZY, Solko-Breslin A, Alur R, Naik M. Understanding the effectiveness of large language models in detecting security vulnerabilities. arXiv:2311.16169, 2023.

[144] Harzevili NS, Belle AB, Wang JJ, Wang S, Jiang ZM, Nagappan N. A survey on automated software vulnerability detection using machine learning and deep learning. arXiv:2306.11673, 2023.

[145] Chen H, Yi P. Code vulnerability detection method based on graph neural network. Chinese Journal of Network and Information Security, 2021, 7(3): 37–45 (in Chinese with English abstract). [doi: 10.11959/j.issn.2096-109x.2021039]

[146] Chen ZX, Zou DQ, Li Z, JIin H. Intelligent vulnerability detection system based on abstract syntax tree. Journal of Cyber Security, 2020, 5(4): 1–13 (in Chinese with English abstract). [doi: 10.19363/J.cnki.cn10-1380/tn.2020.07.01]

[147] Cheng JY, Wang BH, Luo P. Code vulnerability static detection method based on graph representation and MHGAT. Systems Engineering and Electronics, 2023, 45(5): 1535–1543 (in Chinese with English abstract). [doi: 10.12305/j.issn.1001-506X.2023.05.31]

[148] Duan X, Wu JZ, Luo TY, Yang MT, Wu YJ. Vulnerability mining method based on code property graph and attention BiLSTM. Ruan Jian Xue Bao/Journal of Software, 2020, 31(11): 3404–3420 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6061.htm [doi: 10.13328/j.cnki.jos.006061]

[149] Li YC, Cui YQ, Lv JF, Lai FG, Zhang P. Combined deep learning method for open source software vulnerability detection. Computer Engineering and Applications, 2019, 55(11): 52–59 (in Chinese with English abstract). [doi: 10.3778/j.issn.1002-8331.1809-0076]

[150] Liang SB, Zheng L, Zhong J, Hu Y. Source code vulnerability detection model based on convolutional neural networks. Communications Technology, 2022, 55(4): 493–499 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-0802.2022.04.013]

[151] Song ZT, Hu Y. Source code vulnerability detection method based on graph neural network. Communications Technology, 2022, 55(5): 640–645 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-0802.2022.05.014]

[152] Wang J, Kuang HY, Li RL, Su YF. Software source code vulnerability detection based on CNN-GAP interpretability model. Journal of Electronics & Information Technology, 2022, 44(7): 2568–2575 (in Chinese with English abstract). [doi: 10.11999/JEIT210412]

[153] Wen M, Wang RC, Jiang SJ. Source code vulnerability detection based on relational graph convolution network. Journal of Computer Applications, 2022, 42(6): 1814–1821 (in Chinese with English abstract). [doi: 10.11772/j.issn.1001-9081.2021091691]

[154] Wu B, Shen X, Geng JF, Liu B. Binary software vulnerability detection method based on composite neural network. Information Countermeasure Technology, 2022, 1(3): 57–65 (in Chinese with English abstract). [doi: 10.12399/j.issn.2097-163x.2022.03.005]

[155] Xia ZY, Yi P, Yang T. Static vulnerability detection based on neural network and code similarity. Computer Engineering, 2019, 45(12): 141–146 (in Chinese with English abstract). [doi: 10.19678/j.issn.1000-3428.0053136]

[156] Yang HY, Yang HY, Zhang L, Cheng X. Feature dependence graph based source code loophole detection method. Journal on Communications, 2023, 44(1): 103–117 (in Chinese with English abstract). [doi: 10.11959/j.issn.1000−436x.2023018]

[157] Zou DQ, Li X, Huang MH, Song X, Li H, Li WM. Intelligent vulnerability detection system based on graph structured source code slice. Chinese Journal of Network and Information Security, 2021, 7(5): 113–122 (in Chinese with English abstract). [doi: 10.11959/j.issn.2096-109x.2021088]

[158] Nikitopoulos G, Dritsa K, Louridas P, Mitropoulos D. CrossVul: A cross-language vulnerability dataset with commit data. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 1565–1569. [doi: 10.1145/3468264.3473122]

[159] Venson E, Lam TF, Clark B, Boehm B. Analyzing software security-related size and its relationship with vulnerabilities in OSS. In: Proc. of the 21st IEEE Int'l Conf. on Software Quality, Reliability and Security. Hainan: IEEE, 2021. 956–965. [doi: 10.1109/QRS54544.2021.00105]

[160] Mikolov T, Sutskever I, Chen K, Corrado G, Dean J. Distributed representations of words and phrases and their compositionality. In: Proc. of the 27th Int'l Conf. on Neural Information Processing Systems. Lake Tahoe: Curran Associates Inc., 2013. 3111–3119.

[161] Pennington J, Socher R, Manning C. GloVe: Global vectors for word representation. In: Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing. Doha: ACL, 2014. 1532–1543. [doi: 10.3115/v1/D14-1162]

[162] Joulin A, Grave E, Bojanowski P, Mikolov T. Bag of tricks for efficient text classification. In: Proc. of the 15th Conf. of the European Chapter of the Association for Computational Linguistics. Valencia: ACL, 2017. 427–431.

[163] Bojanowski P, Grave E, Joulin A, Mikolov T. Enriching word vectors with subword information. Trans. of the Association for Computational Linguistics, 2017, 5: 135–146. [doi: 10.1162/tacl_a_00051]

[164] Pagliardini M, Gupta P, Jaggi M. Unsupervised learning of sentence embeddings using compositional n-gram features. In: Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. New Orleans: ACL, 2018. 528–540. [doi: 10.18653/v1/N18-1049]

[165] Le Q, Mikolov T. Distributed representations of sentences and documents. In: Proc. of the 31st Int'l Conf. on Machine Learning. Beijing: JMLR.org, 2014. 1188–1196.

[166] Grover A, Leskovec J. Node2vec: Scalable feature learning for networks. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 855–864. [doi: 10.1145/2939672.2939754]

[167] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[168] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: ACL, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[169] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Findings of the Association for Computational Linguistics. ACL, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[170] Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L. Deep contextualized word representations. In: Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. New Orleans: ACL, 2018. 2227–2237. [doi: 10.18653/v1/N18-1202]

[171] Yang ZC, Yang DY, Dyer C, He XD, Smola A, Hovy E. Hierarchical attention networks for document classification. In: Proc. of the 2016 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. San Diego: ACL, 2016. 1480–1489. [doi: 10.18653/v1/N16-1174]

[172] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017.

[173] Schlichtkrull M, Kipf TN, Bloem P, van den Berg R, Titov I, Welling M. Modeling relational data with graph convolutional networks. In: Proc. of the 15th Int'l Conf. on Semantic Web. Heraklion: Springer, 2018. 593–607. [doi: 10.1007/978-3-319-93417-4_38]

[174] Li YJ, Tarlow D, Brockschmidt M, Zemel RS. Gated graph sequence neural networks. arXiv:1511.05493, 2015.

[175] Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. In: Proc. of the 6th Int'l Conf. on Learning Representations. Vancouver: OpenReview.net, 2018.

[176] National Institute of Standards and Technology. Software assurance reference dataset (SARD). 2005. https://samate.nist.gov/SARD

[177] National Institute of Standards and Technology. Test suites. 2010. https://samate.nist.gov/SARD/test-suites

[178] Sestili CD, Snavely WS, VanHoudnos NM. Towards security defect prediction with AI. arXiv:1808.09897, 2018.

[179] Le T, Nguyen T, Le T, Phung DQ, Montague P, de Vel OY, Qu LZ. Maximal divergence sequential autoencoder for binary software vulnerability detection. In: Proc. of the 7th Int'l Conf. on Learning Representations. New Orleans: OpenReview.net, 2019.

[180] Fan JH, Li Y, Wang SH, Nguyen TN. A C/C++ code vulnerability dataset with code changes and CVE summaries. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: IEEE, 2020. 508–512. [doi: 10.1145/3379597.3387501]

[181] Zheng YH, 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. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice. Madrid: IEEE, 2021. 111–120. [doi: 10.1109/ICSE-SEIP52600.2021.00020]

[182] Lu S, Guo DY, Ren S, Huang JJ, Svyatkovskiy A, Blanco A, Clement CB, Drain D, Jiang DX, Tang DY, Li G, Zhou LD, Shou LJ, Zhou L, Tufano M, Gong M, Zhou M, Duan N, Sundaresan N, Deng SK, Fu SY, Liu SJ. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In: Proc. of the 2021 Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks. 2021.

[183] Bhandari G, Naseer A, Moonen L. CVEfixes: Automated collection of vulnerabilities and their fixes from open-source software. In: Proc. of the 17th Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. Athens: ACM, 2021. 30–39. [doi: 10.1145/3475960.3475985]

[184] Chen YZ, Ding ZJ, Alowain L, Chen XY, Wagner D. DiverseVul: A new vulnerable source code dataset for deep learning based vulnerability detection. In: Proc. of the 26th Int'l Symp. on Research in Attacks, Intrusions and Defenses. Hong Kong: ACM, 2023. 654–668. [doi: 10.1145/3607199.3607242]

[185] Jain R, Gervasoni N, Ndhlovu M, Rawat S. A code centric evaluation of C/C++ vulnerability datasets for deep learning based vulnerability detection techniques. In: Proc. of the 16th Innovations in Software Engineering Conf. Allahabad: ACM, 2023. 6. [doi: 10.1145/3578527.3578530]

[186] Nie X, Li NK, Wang KL, Wang SG, Luo XP, Wang HY. Understanding and tackling label errors in deep learning-based vulnerability detection (experience paper). In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023.

52–63. [doi: 10.1145/3597926.3598037]
[187] Powers DMW. Evaluation: From precision, recall and *F*-measure to ROC, informedness, markedness & correlation. Journal of Machine Learning Technologies, 2011, 2(1): 37–63. [doi: 10.9735/2229-3981]
[188] Fu M, Tantithamthavorn C, Le T, Nguyen V, Phung D. VulRepair: A T5-based automated software vulnerability repair. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 935–947. [doi: 10.1145/3540250.3549098]
[189] Gu ZQ, Hu WX, Zhang CJ, Lu H, Yin LH, Wang L. Gradient shielding: Towards understanding vulnerability of deep neural networks. IEEE Trans. on Network Science and Engineering, 2021, 8(2): 921–932. [doi: 10.1109/tnse.2020.2996738]
[190] Lu H, Jin CJ, Helu XH, Du XJ, Guizani M, Tian ZH. DeepAutoD: Research on distributed machine learning oriented scalable mobile communication security unpacking system. IEEE Trans. on Network Science and Engineering, 2022, 9(4): 2052–2065. [doi: 10.1109/TNSE.2021.3100750]
[191] Tian ZH, Li MH, Qiu MK, Sun YB, Su S. Block-DEF: A secure digital evidence framework using blockchain. Information Sciences, 2019, 491: 151–165. [doi: 10.1016/j.ins.2019.04.011]

附中文参考文献:

[29] 胡雨涛, 王溯远, 吴月明, 邹德清, 李文科, 金海. 基于图神经网络的切片级漏洞检测及解释方法. 软件学报, 2023, 34(6): 2543–2561. http://www.jos.org.cn/1000-9825/6849.htm [doi: 10.13328/j.cnki.jos.006849]
[145] 陈皓, 易平. 基于图神经网络的代码漏洞检测方法. 网络与信息安全学报, 2021, 7(3): 37–45. [doi: 10.11959/j.issn.2096-109x.2021039]
[146] 陈肇炫, 邹德清, 李珍, 金海. 基于抽象语法树的智能化漏洞检测系统. 信息安全学报, 2020, 5(4): 1–13. [doi: 10.19363/J.cnki.cn10-1380/tn.2020.07.01]
[147] 程靖云, 王布宏, 罗鹏. 基于图表示和 MHGAT 的代码漏洞静态检测方法. 系统工程与电子技术, 2023, 45(5): 1535–1543. [doi: 10.12305/j.issn.1001-506X.2023.05.31]
[148] 段旭, 吴敬征, 罗天悦, 杨牧天, 武延军. 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. http://www.jos.org.cn/1000-9825/6061.htm [doi: 10.13328/j.cnki.jos.006061]
[149] 李元诚, 崔亚奇, 吕俊峰, 来风刚, 张攀. 开源软件漏洞检测的混合深度学习方法. 计算机工程与应用, 2019, 55(11): 52–59. [doi: 10.3778/j.issn.1002-8331.1809-0076]
[150] 梁树彬, 郑力, 钟杰, 胡勇. 基于卷积神经网络的源代码漏洞检测模型. 通信技术, 2022, 55(4): 493–499. [doi: 10.3969/j.issn.1002-0802.2022.04.013]
[151] 宋子韬, 胡勇. 基于图神经网络的源码漏洞检测方法研究. 通信技术, 2022, 55(5): 640–645. [doi: 10.3969/j.issn.1002-0802.2022.05.014]
[152] 王剑, 匡洪宇, 李瑞林, 苏云飞. 基于 CNN-GAP 可解释性模型的软件源码漏洞检测方法. 电子与信息学报, 2022, 44(7): 2568–2575. [doi: 10.11999/JEIT210412]
[153] 文敏, 王荣存, 姜淑娟. 基于关系图卷积网络的源代码漏洞检测. 计算机应用, 2022, 42(6): 1814–1821. [doi: 10.11772/j.issn.1001-9081.2021091691]
[154] 吴波, 沈璇, 耿君峰, 刘波. 基于复合式神经网络的二进制软件漏洞检测方法. 信息对抗技术, 2022, 1(3): 57–65. [doi: 10.12399/j.issn.2097-163x.2022.03.005]
[155] 夏之阳, 易平, 杨涛. 基于神经网络与代码相似性的静态漏洞检测. 计算机工程, 2019, 45(12): 141–146. [doi: 10.19678/j.issn.1000-3428.0053136]
[156] 杨宏宇, 杨海云, 张良, 成翔. 基于特征依赖图的源代码漏洞检测方法. 通信学报, 2023, 44(1): 103–117. [doi: 10.11959/j.issn.1000–436x.2023018]
[157] 邹德清, 李响, 黄敏桓, 宋翔, 李浩, 李伟明. 基于图结构源代码切片的智能化漏洞检测系统. 网络与信息安全学报, 2021, 7(5): 113–122. [doi: 10.11959/j.issn.2096-109x.2021088]

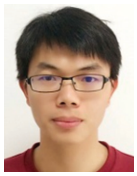
唐家昕(2000－), 男, 硕士生, 主要研究领域为软件分析与测试.

王璇(2002－), 女, 硕士生, 主要研究领域为静态分析.

赖伟(1999－), 男, 硕士生, 主要研究领域为程序自动修复补丁正确性评估.

路则雨(1996－), 男, 博士生, CCF 学生会员, 主要研究领域为软件分析与测试.

郭肇强(1994－), 男, 博士, 主要研究领域为软件分析与测试.



杨已彪(1987－), 男, 博士, 准聘助理教授, CCF 高级会员, 主要研究领域为软件分析与测试.

周毓明(1974－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件分析与测试.