计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202221046
60(7): 1467−1488, 2023

开源软件缺陷预测方法综述

田 笑[1,2] 常继友[3] 张 弛[2] 荣景峰[2,6] 王子昱[3] 张光华[3] 王 鹤[1,2] 伍高飞[1,2,4] 胡敬炉[5] 张玉清[1,2,6,7]

[1](西安电子科技大学网络与信息安全学院 西安 710126)

[2](国家计算机网络入侵防范中心(中国科学院大学) 北京 101408)

[3](河北科技大学信息科学与工程学院 石家庄 050018)

[4](广西密码学与信息安全重点实验室(桂林电子科技大学) 广西桂林 541000)

[5](早稻田大学情报生产系统研究科 日本 808-0135)

[6](海南大学网络空间安全学院 海口 570228)

[7](中关村实验室 北京 100094)

(tianx@nipc.org.cn)

Survey of Open-Source Software Defect Prediction Method

Tian Xiao[1,2], Chang Jiyou[3], Zhang Chi[2], Rong Jingfeng[2,6], Wang Ziyu[3], Zhang Guanghua[3], Wang He[1,2], Wu Gaofei[1,2,4], Hu Jinglu[5], and Zhang Yuqing[1,2,6,7]

[1](*School of Cyber Engineering*, *Xidian University*, *Xi'an* 710126)

[2](*National Computer Network Intrusion Protection Center* (*University of Chinese Academy of Sciences*), *Beijing* 101408)

[3](*School of Information Science and Engineering*, *Hebei University of Science and Technology*, *Shijiazhuang* 050018)

[4](*Guangxi Key Laboratory of Cryptography and Information Security* (*Guilin University of Electronic Technology*), *Guilin*,*Guangxi* 541000)

[5](*Graduate School of Information*, *Production and Systems*, *Waseda University*, *Japan* 808-0135)

[6](*College of Cyberspace Security*, *Hainan University*, *Haikou* 570228)

[7](*Zhongguancun Laboratory*, *Beijing* 100094)

Abstract Open-source software defect prediction reduces software repair costs and improves product quality by mining data from software history warehouses, using the syntactic semantic features of metrics related to software defects or the source code itself, and utilizing machine learning or deep learning methods to find software defects in advance. Vulnerability prediction extracts and tags code modules by mining software instance repositories to predict whether new code instances contain vulnerabilities in order to reduce the cost of vulnerability discovery and fixing. We investigate and analyze the relevant literatures in the field of software defect prediction from 2000 to December 2022. Taking machine learning and deep learning as the starting point, we sort out two types of prediction models which are based on software metrics and grammatical semantics. Based on the two types of models, the difference and connection between software defect prediction and vulnerability prediction are analyzed. Moreover, six frontier hot issues such as dataset source and processing, code vector representation method, pre-training model improvement, deep learning model exploration, fine-grained prediction technology, software defect prediction and vulnerability

收稿日期: 2023-03-30; **修回日期**: 2023-06-06

基金项目: 先进密码技术与系统安全四川省重点实验室开放课题 (SKLACSS-202205); 海南省重点研发计划项目 (GHYF2022010, ZDYF202012); 国家自然科学基金项目 (U1836210); 陕西省自然科学基础研究计划 (2021JQ-192); 广西密码学与信息安全重点实验室课题 (GCIS202123)
This work was supported by the Open Fund of Advanced Cryptography and System Security Key Laboratory of Sichuan Province (SKLACSS-202205), the Key Research and Development Program of Hainan Province (GHYF2022010, ZDYF202012), and the National Natural Science Foundation of China (U1836210), the Natural Science Basis Research Plan in Shaanxi Province of China (2021JQ-192), and the Program of Guangxi Key Laboratory of Cryptography and Information Security (GCIS202123).

通信作者: 张玉清(zhangyq@nipc.org.cn)

prediction model migration are analyzed in detail. Finally, the future development direction of software defect prediction is pointed out.

Key words software defect prediction; vulnerability prediction; machine learning; deep learning; metric; semantic and syntactic analysis

摘 要 开源软件缺陷预测通过挖掘软件历史仓库的数据，利用与软件缺陷相关的度量元或源代码本身的语法语义特征，借助机器学习或深度学习方法提前发现软件缺陷，从而减少软件修复成本并提高产品质量. 漏洞预测则通过挖掘软件实例存储库来提取和标记代码模块，预测新的代码实例是否含有漏洞，减少漏洞发现和修复的成本. 通过对2000年至2022年12月软件缺陷预测研究领域的相关文献调研，以机器学习和深度学习为切入点，梳理了基于软件度量和基于语法语义的预测模型. 基于这2类模型，分析了软件缺陷预测和漏洞预测之间的区别和联系，并针对数据集来源与处理、代码向量的表征方法、预训练模型的提高、深度学习模型的探索、细粒度预测技术、软件缺陷预测和漏洞预测模型迁移六大前沿热点问题进行了详尽分析，最后指出了软件缺陷预测未来的发展方向.

关键词 软件缺陷预测；漏洞预测；机器学习；深度学习；度量元；语法语义分析

中图法分类号 TP311

软件缺陷(software defect)，也称Bug或Fault，是由于开发过程或者维护过程中存在的错误而产生的导致计算机无法正常运行的错误或者功能性缺陷. 当在软件测试期间发现缺陷时，这些缺陷可能处于多种不同的状态，常见的缺陷状态信息如表1所示，

Table 1 Software Defect State Description

表1 软件缺陷状态描述

状态	描述
新建（New）	缺陷在测试中首次出现，并被质量工程师标记
待确认（Pending）	缺陷已被报告，并等待确认
开放（Open）	被确定为缺陷，等待被分配和修复
已分配（Assigned）	初步筛选后，被分配给适当的团队进行修复
拒绝（Rejected）	缺陷不需要修复或者不是缺陷
修复中（In Progress）	缺陷已被确认，并且开发人员正在处理修复
已修复（Fixed）	开发人员修改代码或者配置，并将缺陷标记为已修复
待测试（Test）	修复后的缺陷等待再次进行测试以验证修复是否有效
重新开放（Re-open）	经过修复并重新测试后，缺陷再次出现并被重新标记
已解决（Resolved）	缺陷已经修复，并且通过再次测试验证了修复的有效性
已关闭（Closed）	缺陷被确认为已解决，不需要进一步处理

如果缺陷没有被及时发现或者修复，则可能会影响软件系统的功能从而造成巨大的财产损失甚至威胁人身安全.

传统的软件缺陷检测方法，包括手动测试、自动化分析、静态分析、代码审查等[1]，但是由于人力资源和时间资源有限，传统的缺陷发现技术只能检测到较少的缺陷. 基于人工智能的软件缺陷预测技术包括机器学习和深度学习，可以预测更详细的软件缺陷信息. 软件缺陷的检测和预测方法对比如表2所示，软件缺陷预测可以在较少的时间内发现更多的软件缺陷，减少用于发现和修复缺陷的成本.

软件缺陷预测根据软件历史开发数据以及已发现的缺陷数据，借助机器学习等方法来预测软件项目中的缺陷数目和类型[2]. 软件缺陷预测模型可以确定哪些组件具有最大的安全风险，软件工程师可以根据模型预测结果做出风险管理决策，指导安全检查和测试，并确定软件安全防御工作的优先级.

早期研究者利用源代码度量从软件源代码中提取缺陷特征，静态分析样本中的缺陷数量，后来研究者定义与软件缺陷强相关的度量，提高分类器的效

Table 2 Comparison of Defect Detection and Defect Prediction Methods

表2 缺陷检测与缺陷预测方法对比

方法	类别	准确性	范围	时间	局限性
手动测试	缺陷检测	较为准确	较小	很多	可能出现人为错误
自动化分析	缺陷检测	基本准确	较大	适中	难以处理视觉、用户体验等问题
静态分析	缺陷检测	基本准确	较小	少	无法检测运行时行为和集成问题
代码审查	缺陷检测	较为准确	较小	很多	取决于审查者的经验和技能水平
人工智能	缺陷预测	基本准确	大	适中	取决于数据质量和技术

率. 基于软件度量的缺陷预测模型如图 1 所示，模型首先从开源公共仓库或者开源社区中抽取程序模块；然后定义与软件缺陷强相关的度量元，提取对应的度量特征从而构造出缺陷数据集；最后通过软件缺陷数据集训练软件缺陷预测模型，软件缺陷预测模型就能够预测目标程序模块的软件缺陷相关信息.

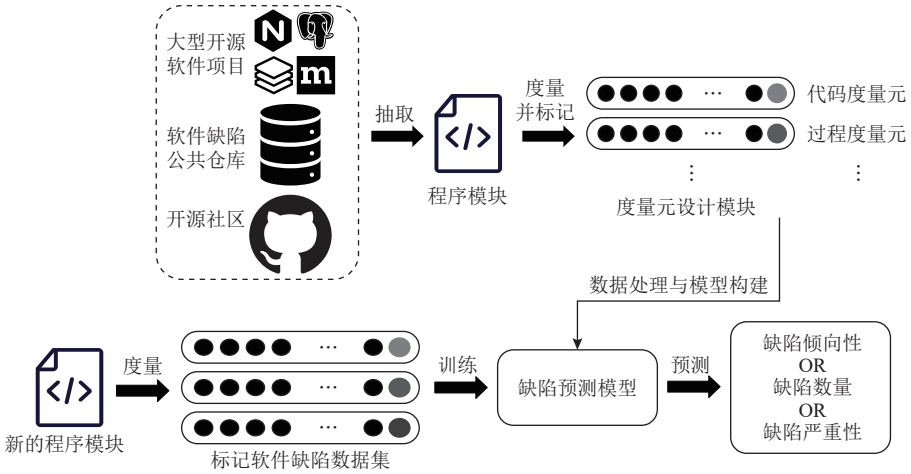


Fig. 1 Defect prediction model based on software metrics

图 1 基于软件度量的缺陷预测模型

基于软件度量的缺陷预测模型的缺陷预测方法较为简单，只需要手工定义与软件缺陷相关的度量元，但是该模型无法挖掘出与缺陷相关的更深层次信息，应用范围比较局限，模型的优劣取决于数据集和分类器.

随着深度学习的发展，研究者开始从复杂的代码中自动获取更深层次的特征，表征程序的语法语义信息. 基于语法语义的缺陷预测模型如图 2 所示，模型首先利用文本挖掘技术对源代码的标识符、函数名和运算符进行解析并标记，然后提取软件缺陷相关特征并抽象表示，最后通过编码器模型生成向量并送入神经网络预测模型进行训练.

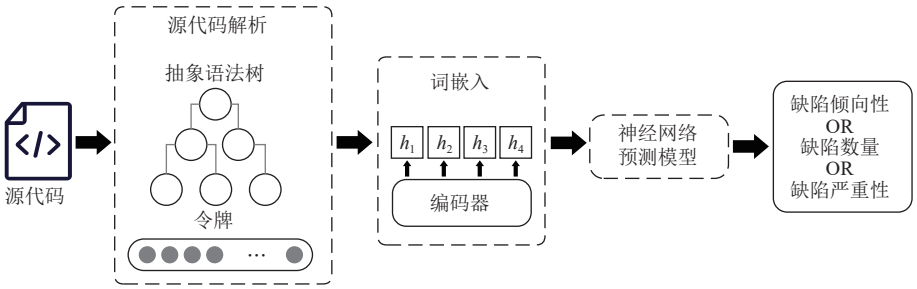


Fig. 2 Defect prediction model based on semantic and syntactic

图 2 基于语法语义的缺陷预测模型

漏洞是一种特定的软件安全缺陷，受软件缺陷预测研究的启发和推动，漏洞预测通过挖掘软件实例存储库来提取和标记代码模块，预测新的代码实例是否含有漏洞，减少用于发现和修复漏洞的成本. 与缺陷预测领域相同，漏洞预测研究领域也可以划分为基于软件度量的漏洞预测和基于语法语义的漏洞预测模型. 基于软件度量的漏洞预测模型使用人工设计的代码变更、耦合和内聚等度量，表征漏洞的相关信息[3]. 由于漏洞数量极少且种类繁多，这些特征不能完全理解代码本身的含义且预测效果较差，因此该模型逐渐被舍弃.

随着人工智能技术在各个领域的应用，人工智能技术被用于缺陷预测和漏洞预测任务，漏洞预测模型的效果因此得到了提升. 为了能够表征漏洞更多的信息，研究人员使用文本、抽象语法树(abstract syntax tree, AST)、代码属性图等多种方式提取软件代码中的特征，挖掘代码的语法语义信息，转化为向量的数字表示，随后使用机器学习或者深度学习方法预测项目中的漏洞. 细粒度的漏洞预测是漏洞领域的研究热点，为了提取更细粒度的特征，一些研究人员将漏洞函数表征为漏洞切片，使用注意力机制预测漏洞的位置信息.

本文使用 defect prediction, software defect prediction, fault prediction, vulnerability prediction 和 vulnerability detection 等关键字进行检索，收集并调研了 2000 年至 2022 年来自 IEEE Xplore、ACM、Digital Library、SpringerLink、ScienceDirect 和中国知网(CNKI)等国内外数据库和软件工程领域与网络安全领域的国际期刊和顶级会议上的论文，总结了开源软件缺陷预测已有的研究成果，并指出了该研究领域的未来发展趋势. 缺陷预测和漏洞预测各年份统计的相关文献数量如图 3 所示.

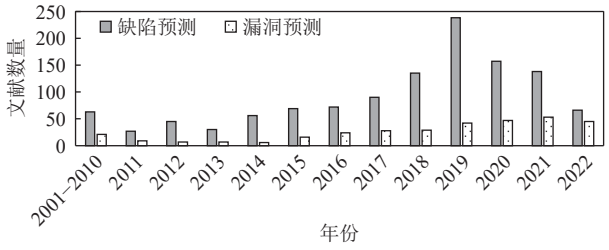


Fig. 3 Number of literatures related to defect prediction and vulnerability prediction

图 3 缺陷预测和漏洞预测相关文献数量

本文的主要贡献有 4 个方面：

1) 广泛收集并调研了 2000 年至 2022 年 12 月软件缺陷预测现有文献，从数据集、评价指标、模型构建 3 个方面分析了缺陷预测的研究现状；

2) 以机器学习和深度学习技术为切入点，总结了基于软件度量和基于语法语义 2 类预测模型；

3) 分析了软件缺陷预测模型和漏洞预测模型之间的区别和联系，探讨了缺陷预测和漏洞预测之间的可移植性；

4) 归纳和分析了缺陷预测技术领域的不足，指出了当前面临的机遇和挑战，给出了未来发展方向.

1 缺陷预测框架

软件缺陷预测技术包括静态缺陷预测技术与动态缺陷预测技术. 静态缺陷预测是目前主流的软件缺陷预测技术，它通过挖掘软件历史仓库，手工定义与软件缺陷相关的度量元和从源代码本身表征程序的语法语义信息 2 种方式，借助机器学习或深度学习方法提前发现软件缺陷. 动态缺陷预测通过研究软件缺陷随时间变化的趋势，预测软件的下一个版本或下一个生命周期中可能出现缺陷的数量. 基于动态软件缺陷预测技术的核心是数学模型，包括 Rayleigh 分布模型[4-5]、指数分布模型[6-7]、S 曲线分布模型[8-10]. 对于动态软件缺陷预测模型来说，需要对软件产品进行长时间的监测以获取软件的数据. 然而收集大量的开发数据需要耗费大量的成本，并且不同的数据集和动态缺陷预测模型会导致预测精度不尽如人意. 因此缺陷预测的大部分研究目前都集中在静态缺陷预测，后文的缺陷预测特指静态缺陷预测.

本文以机器学习和深度学习技术为切入点，介绍了基于软件度量和基于语法语义的 2 类缺陷预测模型，这 2 类缺陷预测模型框架如图 4 所示.

缺陷预测框架由 3 部分组成：

1) 从软件缺陷公共仓库和开源社区收集缺陷数据；

2) 定义与软件缺陷高度相关的度量元，提取程序模块的软件度量特征或者使用神经网络自动提取源代码的语法语义特征，利用机器学习或深度学习技术构建预测模型；

3) 使用性能评价指标对预测模型进行评价.

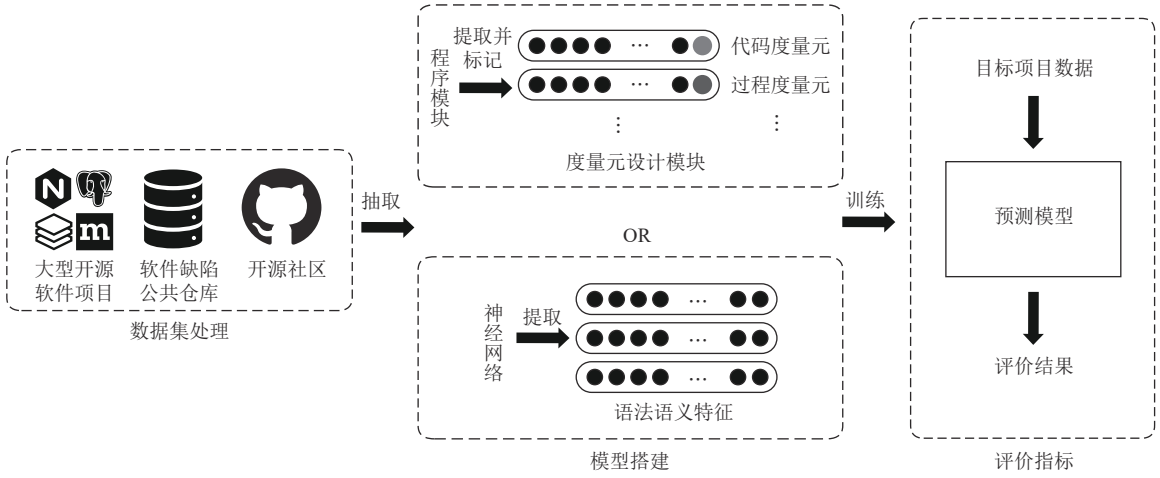


Fig. 4 Defect prediction framework

图 4 缺陷预测框架

下文从缺陷预测框架入手，分别介绍了缺陷数据、模型的评价指标以及缺陷预测模型的搭建.

2 缺陷数据集

软件缺陷预测模型的构建离不开缺陷数据集，但缺陷数据集中的维数灾难和类不平衡问题影响着软件缺陷预测模型的质量. 本节从缺陷数据集的来源和缺陷数据预处理 2 个角度对缺陷数据集进行分析.

2.1 缺陷数据集来源

软件缺陷数据集包含了软件项目全生命周期的缺陷数据信息，常常被用于软件缺陷预测的研究.

开源软件缺陷数据集主要来源于软件缺陷公共仓库、开源社区和大型开源软件项目. 国内外研究人员使用 NASA, PROMISE 等公共库进行了一系列研究，对软件缺陷预测的发展做出了贡献，这些公共库提供了包含了 Apache, Eclipse 等真实软件项目的数据集，经过验证和标注，具有较高的数据质量，通过对这些公共仓库数据的分析和实验，研究人员能够评估不同方法在缺陷预测方面的性能和效果. 随着越来越多的开源软件项目公开，一些研究者选择使用大型开源软件项目，如 Nginx, Redis 和 Gedit 等，评估不同的缺陷预测模型在预测软件缺陷方面的准确性和有效性. SourceForge, GitHub 等开源社区提供了与软件相关的特征，研究人员利用源代码、软件描述、错误修复、审查过程等开源软件提供的信息，发现深层隐藏的软件缺陷[11].

本文对 2000 年至 2022 年 12 月的论文进行了整理分析，软件缺陷公共仓库的数据来源及其统计信息如表 3 所示. 大部分研究人员使用 NASA, PROMISE 和 AEEEM 中的 KC3, PC4, CM1, PC3, KC1 和 PC1 等数据集，公共数据集的属性如表 4 所示，这些数据集发布在 2005 年左右，主要使用 C, C++, Java 等语言开发. 除了使用 NASA, PROMISE 中的数据集外，一些研究者使用 Nginx, Redis, Gedit 等大型开源软件，在软件缺陷预测领域开展一系列研究，开源软件项目缺陷数量统计如表 5 所示，文献 [12] 使用了 2 个连续版本的 6 个开源软件项目 Camel, Flume, Tika, Gedit, Nginx, Redis, 研究软件模块在文件级别的缺陷. 文献 [13] 使用了多个版本的 4 个开源软件项目 Gedit, Nagios Core, Nginx, Redis, 这些项目主要使用 C/C++语言开发，文献 [13] 的作者从缺陷修复报告中收集到了这些项目的缺陷信息. 此外，文献 [14] 探究 6 个项目在开源软件项目的不同版本中的代码变更，并分析了这些变更对产生缺陷的可能性，开源软件代码变更对缺陷的影响如表 6 所示.

Table 3 Public Warehouse Data Sources for Software Defect Modeling

表 3 软件缺陷模型的公共仓库数据来源

数据集	项目数	度量名称	文献数量	粒度	数据链接
NASA	13	代码度量	33	函数	http://promise.site.uottawa.ca/SERepository/datasets-page.html
SOFTLAB	5	代码度量	2	函数	https://github.com/bharlow058/AEEEM-and-other-SDP-datasets/tree/master/dataset/SOFTLAB
PROMISE	38	代码度量	41	类	https://zenodo.org/search?page=1&size=20&q=Marian%20Jureckzo&file_type=csv#
Relink	3	代码度量	7	文件	https://github.com/ai-se/HDP_pyjnius/tree/master/dataset/Relink
AEEEM	5	代码度量 过程度量	14	类	https://bug.inf.usi.ch/download.php
MORPH	9	代码度量	2	类	https://github.com/bharlow058/AEEEM-and-other-SDP-datasets/tree/master/dataset/MORPH

经过调查发现，NASA, PROMISE 等软件缺陷公共仓库的数据集由 LOC[15], Halstead[16] 等度量元的属性和包含缺陷的类标记组成. 大部分研究者使用 NASA, PROMISE 等软件缺陷公共仓库的数据集进行模型的比较和重现，一些研究者使用 Nginx, Redis, Gedit 等真实世界的开源软件，在软件缺陷预测领域开展一系列研究. 这些公开数据集往往具有类不平衡、维度过高、预测特征不足以及分类标签不足等缺点[17]. 数据集中维数灾难和类不平衡的问题可以通过特征选择[18]、降维[19-29] 等方法进行处理，但是特征不足和分类标签不足的问题则需要通过定义更细粒度的软件缺陷度量元来解决.

2.2 缺陷数据预处理

缺陷数据集中类不平衡[23-36]、维度过高[37-43]、特征不足[44-45] 以及分类标签不足[46-50] 等问题会降低软件缺陷预测模型的性能，在使用缺陷数据时对其进行预处理，可以提高缺陷预测模型的质量，部分缺陷数据预处理的方法如表 7 所示.

2.2.1 类不平衡

程序模块中 80% 的缺陷集中在 20% 的文件中，

Table 4 Attributes List of the Publicly Available Datasets

表 4 公共数据集属性列表

数据集	缺陷仓库	语言	属性	行数	缺陷行	缺陷率/%
CM1	NASA	C	22	498	49	9.84
JM1	NASA	C	22	10 885	8 779	80.65
KC1	NASA	C++	22	2 109	326	15.46
KC2	NASA	C++	22	522	105	20.11
KC3	NASA	Java	40	458	43	9.39
KC4	NASA	Perl	40	125	61	48.80
MC1	NASA	C++	39	9 466	68	0.72
MC2	NASA	C++	40	161	52	32.30
MW1	NASA	C	40	403	61	15.14
PC1	NASA	C	40	1 107	76	6.87
PC2	NASA	C	40	5 589	23	0.41
PC3	NASA	C	40	1 563	160	10.24
PC4	NASA	C	40	1 458	178	12.21
PC5	NASA	C++	39	17 186	516	3.00
ant-1.7	PROMISE	Java	21	745	166	22.30
ivy-2.0	PROMISE	Java	21	352	40	11.40
camel-1.6	PROMISE	Java	21	965	188	19.50
jedit-4.0	PROMISE	Java	21	306	75	24.50
log4j-1.2	PROMISE	Java	21	109	37	33.90
Lucene-2.4	PROMISE	Java	21	195	91	46.70
poi-2.0	PROMISE	Java	21	314	37	11.80
Synapse-1.1	PROMISE	Java	21	222	60	27.00
velocity-1.6	PROMISE	Java	21	229	78	34.10
Xerces-1.3	PROMISE	Java	21	453	60	15.20
tomcat	PROMISE	Java	21	858	77	8.90
Xalan-2.4	PROMISE	Java	21	723	110	15.20
EQ	AEEEM	Java	62	324	129	39.81
JDT	AEEEM	Java	62	997	206	20.66
LC	AEEEM	Java	62	691	64	9.26
ML	AEEEM	Java	62	1 862	245	13.16
PDE	AEEEM	Java	62	1 497	209	13.96

Table 5 Number of Defects List in Open-Source Software Projects

表 5 开源软件项目缺陷数量列表

来源	开源软件	版本数量	细粒度	代码行	缺陷数量
文献 [12]	Camel	2	文件	112 367	62.00
文献 [12]	Flume	2	文件	95 782	47.00
文献 [12]	Tika	2	文件	85 341	16.00
文献 [12]	Gedit	2	文件	60 441	18.50
文献 [12]	Nginx	2	文件	80 618	18.00
文献 [12]	Redis	2	文件	45 991	21.00
文献 [13]	Gedit	314	函数	2 012	58.96
文献 [13]	Nagios Core	93	函数	1 750	4.82
文献 [13]	Nginx	455	函数	1 975	6.17
文献 [13]	Redis	173	函数	2 350	57.31

缺陷的数量远少于非缺陷的数量，缺陷数量的不平衡分布会导致模型的预测性能受到严重影响.

20 世纪 90 年代，研究人员从数据层面解决类不平衡问题，包括随机欠采样（random under sampling, RUS）[20]、随机过采样（random over sampling, ROS）[21-22]，这些算法简单高效，但是效果取决于模型的训练算法. 文献 [23] 针对 ADASYN[24] 自适应过采样算法造成的噪声问题和 CURE-SMOTE[25] 层过采样算法分类准确性问题，提出了一种 AJCC-Ram 多层次过采样算法，使用 ADASYN 改进算法进行边缘采样，CURE-SMOTE 算法进行中心区采样，分层次进行样本生成，采用最近列表噪声识别（closest list noise identification, CLNI[26]）方法对噪声进行过滤.

文献 [27] 使用邻居清理学习（neighbor cleaning learning, NCL）去除不平衡数据集中的重叠样本，使用集成随机欠采样并重复多次，解决了缺陷数据集偏态分布的问题. 结果表明，与 CLNI、基于聚类的噪声检测（clustering-based noise detection, CBND）[28] 两种数据清洗算法相比，NCL 在 G-mean 和 AUC 评价指标上表现更优.

Table 6 Impact of Code Changes on Defects in Open-Source Software Projects

表 6 开源软件项目代码变更对缺陷的影响

开源软件	代码更改时间段	文件数量	每次更改的文件数	平均变更的代码行	代码更改诱发的缺陷率/%
Bugzilla	08/1998−12/2006	4 620	2.3	37.5	36
Platform	05/2001−12/2007	64 250	4.3	72.2	14
Mozilla	01/2000−12/2006	98 275	5.3	106.5	5
JDT	05/2001−12/2007	35 386	4.3	71.4	14
Columba	11/2002−07/2006	4 455	6.2	149.4	31
PostgreSQL	07/1996−05/2010	20 431	4.5	101.3	25

Table 7 Data Preprocessing Methods

表 7 数据预处理方法

来源	年份	数据集	数据预处理模型	分类方法	评价指标
文献 [23]	2022	AEEM，NASA	AJCC-Ram	XGBoost	*F*1-Score
文献 [27]	2018	PROMISE	NCL，RUS	Adaboost	PD，PF，G-mean，AUC
文献 [30]	2018	NASA，SOFTLAB，ReLink，AEEEM，MORPH	CTKCCA	逻辑回归	PD，PF，*F*-measure，G-mean，AUC
文献 [33]	2019	NASA，PROMISE	STr-NN+TCA	集成学习	*F*-measure，AUC，Recall，PF
文献 [34]	2022	NASA，AEEEM，Relink	BiGAN	随机森林、支持向量机、朴素贝叶斯	AUC，G-mean，*F*1-Score
文献 [39]	2021	NASA，PROMISE，AEEEM，ReLink	EWFS	决策树、朴素贝叶斯	*F*-measure，AUC
文献 [45]	2017	ReLink，AEEEM	FESCH	决策树、朴素贝叶斯、逻辑回归	Precision，Recall，*F*-measure，AUC
文献 [46]	2019	NASA	LSKDSA	逻辑回归	*F*-measure，AUC
文献 [49]	2018	MORPH	HAL，KCPA	逻辑回归	*F*-measure，G-mean，Balance
文献 [50]	2020	MORPH	CDS	随机森林、逻辑回归、朴素贝叶斯	*F*-measure，G-mean，Balance

文献 [29] 提出一种新的基于邻域的欠采样(neighbourhood based under-sampling, N-US)算法来处理类不平衡问题. N-US 首先识别缺陷实例的邻域，计算出邻域中的干净实例，并从有缺陷的实例的邻域中最近的位置消除干净实例. 结果表明，与 RUS 等基线方法相比，N-US 能够避免过度消除数据带来的负面影响，有效地提高了分类器的性能.

算法级方法通过直接修改训练机制解决类不平衡问题，算法级方法包括代价敏感学习方法、集成学习方法等. 代价敏感学习不是通过不同的采样策略创建平衡的数据分布，而是通过使用不同的代价矩阵来解决类不平衡问题，这些代价矩阵表示对特定数据实例进行错误分类的成本. 文献 [30] 提出了一种代价敏感转移核典型相关分析(cost-sensitive transfer kernel canonical correlation analysis, CTKCCA)方法，利用代价敏感技术，在迁移学习阶段使用转移核典型相关分析方法，分离有缺陷和无缺陷的实例. 结果表明，该方法在 PD, PF, AUC 等指标上均优于转移成分分析 (transfer component analysis, TCA)等基线方法.

对于机器学习来说，单个模型存在过拟合和缺乏泛化能力等缺点，集成学习通过组合多个分类器来处理类不平衡. 常用的集成学习方法包括 Boosting, Bagging, Stacking, Voting 以及随机子空间方法(random subspace method, RSM)等. 文献 [31] 使用随机森林(random forest, RF)、人工神经网络、K 最近邻作为模型的主要学习者，逻辑回归(logistic regression, LR)作为模型的次要学习者，从多个 Stacking 模型中选择了性能最好的模型. 文献 [32] 提出一种基于随机近似约简(random approximation reduction, RAR)的集成学习算法 ERAR, 该算法使用 RAR 扰动属性空间，以删除冗余和不相关的属性，采用重采样技术扰动实例空间. 为了解决学习器中数据集不平衡问题，ERAR-SMOTE 在 ERAR 中加入了方法 SMOTE, 对不平衡的缺陷数据进行预处理. 结果表明，ERAR-SMOTE 在 *F*1-Score, MCC, Recall 等指标上优于 RSM, Bagging 等基线方法.

近年来，研究人员使用深度学习解决缺陷预测中的类不平衡问题. 文献 [33] 将嵌入最近邻(stratification embedded in nearest neighbor, STr-NN)的分层思想引入了神经网络，在标记的训练数据集上训练集成学习分类器，用来搜索目标数据的伪标签，基于伪标签确定训练数据集中缺陷的数量. 文献 [33] 的作者使用最近邻(NN)平衡训练数据集，解决项目内缺陷预测(within-project defect prediction, WPDP)中的类不平衡问题. 对于跨项目缺陷预测(cross-project defect prediction, CPDP)的类不平衡问题，在 STr-NN 的基础上，引入 TCA 缓解源数据和目标数据之间的特征不一致问题. 结果表明，与 TCA, TCA+SMOTE 相比，TCA+STr-NN 具有更高的 AUC 和Recall.

文献 [34] 提出了一种基于双向生成对抗网络的软件缺陷预测模型(ADGAN-SDP)，从异常检测的角度解决类不平衡问题. 该模型使用双向生成对抗网络(bidirectional generative adversarial network, BiGAN)对无缺陷样本进行建模，并将无缺陷样本的损失值与预定义的阈值进行比较，从而预测当前样本是否含有缺陷. 结果表明，该方法在 AUC, G-mean, *F*1 等指标上优于代价敏感、过采样、欠采样、集成学习等方法.

混合方法将数据级技术、算法级技术与集成方法结合,用来处理类不平衡问题. 文献 [35] 分别使用采样、代价敏感、集成及混合形式构建软件缺陷预测模型. 结果表明,相比较于采样和代价敏感学习,集成学习实现了更好的预测结果.

传统的 Stacking 方法在处理类不平衡问题时没有考虑代价成本问题. 文献 [36] 提出一种 CSSG 方法,将 Stacking 方法与代价敏感学习结合. 结果表明,该方法能够很好地解决项目内缺陷预测中类不平衡问题.

讨论 1: 不平衡的数据集中可能存在类重叠问题,主要表现在少数有缺陷的训练样本与大多数无缺陷的训练样本实例重叠,增加了分类器学习缺陷特征的难度. 使用 NCL, CLNI, CBDN 等方法可以去除不平衡数据集中的重叠样本,提高缺陷预测的能力.

2.2.2 高维度数据

数据集中的冗余和无关特征会降低模型的预测性能,通过对高维度数据进行处理,能够缩短模型的训练时间并提高模型的预测性能,特征选择方法可以从高维度特征中选择与缺陷高度相关的特征从而实现特征降维. 20 世纪 90 年代后期,研究人员使用包装器[37] 和过滤器[38] 从高维度特征中选择与缺陷高度相关的特征. 基于过滤器的方法使用独立的学习算法评估数据集的特征,复杂度较低且不能保证学习算法的准确性. 基于包装器的特征选择方法使用预定的学习算法评估数据集特征的优劣,特征选择独立性较高,但是基于包装器的特征选择方法用于子集搜索评估和选择的数量未知,存在分类器过拟合等问题.

文献 [39] 改进了基于包装器的特征选择方法,设计了一种 EWFS 模型用来动态选择特征. 该模型首先使用熵度量对数据集中的特征进行排序,然后通过增量包装方法传递排序的特征,最后使用条件互信息最大化(conditional mutual information maximization, CMIM)选择所需特征. 结果表明,这种动态特征选择方法优于现有的元启发式方法和基于顺序搜索的包装法.

混合特征选择方法是基于过滤器和包装器的方法. 文献 [40] 提出一种混合特征选择方法,通过比较方案与理想解之间的相似度来进行排序的方法(technique for order preference by similarity to ideal solution, TOPSIS)为 5 种过滤方法计算分数,决策矩阵生成排序后的候选特征;使用基于 Rao 的优化的包装器算法寻找最优的候选特征. 结果表明,该方法优于 GA[41] 等特征选择方法.

文献 [42] 将代价矩阵结合到传统的特征选择方法,在特征选择阶段强调代价较高的样本,以解决特征选择阶段的类不平衡问题. 结果表明,基于代价敏感的特征选择方法在降低成本和解决不平衡问题上优于传统的特征选择方法.

文献 [43] 提出了 FECAR 特征选择框架,在特征聚类阶段,使用 k-medoids 聚类算法将特征划分为 k 个聚类,并根据 FC-Relevance 度量从每个聚类中选择相关的特征. 结果表明,该方法可以有效地识别并剔除冗余和不相关特征.

讨论 2: 特征选择方法是解决高维度数据的一种有效方法,在保证数据质量的同时选择突出特征. 基于包装器和过滤器的特征选择是 2 种传统的特征选择方法,虽然基于包装器的特征选择方法优于基于过滤器的特征选择方法,但是基于包装器的特征选择方法仍存在着搜索特征子集高时间复杂度等问题,未来应在减少其时间成本的同时,维持该方法的性能.

2.2.3 数据差异

构建性能优越的软件缺陷预测模型需要足够多的历史标注数据,但是现实中标注足够多的训练数据相当困难,数据差异问题普遍存在于跨项目缺陷预测、跨版本缺陷预测中(cross-version defect prediction, CVDP).

跨项目缺陷预测使用其他项目的缺陷数据来预测当前项目是否含有缺陷. 但不同项目的数据分布存在着差异,很难将通过源项目缺陷数据集训练的软件缺陷预测模型泛化到新的目标项目中.

文献 [44] 提出 TCA+方法,将 TCA 与数据归一化方法结合,但该方法性能不稳定. 随后,文献 [45] 提出了 FeSCH 方法,选择合适的特征缓解源项目和目标项目数据之间的分布差距,在特征聚类阶段使用 DPC 基于密度的聚类方法将原始特征划分为多个簇,在特征选择阶段设计了特征的局部密度(local density of features, LDF)、特征分布的相似性(similarity of feature distribution, SFD)、特征的类相关性(feature-class relevance, FCR)这 3 种策略将每个簇中的特征进行排序. 结果表明,FeSCH 相比于 TCA+等基线方法在 AUC 指标表现更优.

但是文献 [44−45] 所提方法没有考虑到源数据中的类标签信息,文献 [46] 提出一种域适应方法 LSKDSA,在子空间对齐(subspace alignment, SA)方法的基础上,使用源数据的可用类别标签,并且在域适应学习的过程中加入了判别信息,以减少源数据和

目标数据分布之间的差异. 结果表明, 该方法在 *F*-measure 和 AUC 等指标上优于 TCA+等基线方法.

由于开发环境等因素的影响, 各个项目的度量值会存在较大差异, 导致跨项目缺陷预测的精度较低. 文献 [47] 提出了一种基于特征选择和迁移学习的度量补偿软件缺陷预测方法, 首先使用 Pearson 计算特征和缺陷类别之间的相关系数, 使用相关系数选择训练时的特征子集; 接着使用 TCA 将源项目和目标项目的共同特征映射到潜在空间中; 随后, 使用度量补偿方法为原始度量值分配权重, 以提高目标项目之间的相似性. 结果表明, 该方法在 AUC 和 *F*-measure 这 2 个指标上优于传统的度量补偿技术.

跨版本缺陷预测同一项目、不同版本得到的数据分布存在着差异, 这种差异对跨版本缺陷预测模型的预测性能造成影响.

软件的演进会导致开发者在原先有缺陷的基础上引入新的缺陷, 文献 [48] 考虑了连续版本之间缺陷分布的变化, 提出一种主动学习方法. 但该方法依赖于先前版本的标记模块, 忽略了当前版本未标记模块的信息, 文献 [49] 在此基础上, 提出了一种混合主动学习(hybrid active learning, HAL), 从当前模块中选择最具有价值的未标记模块, 并与先前版本的标记模块组合, 共同构建混合训练集. HAL 模型使用基于核方法的主成分分析(kernel principal component analysis, KPCA)方法, 将混合训练模块的数据和剩余维标记模块的数据映射到高维特征空间.

但是文献 [48−49] 所述方法忽略了先前版本中数据分布差异的问题, 文献 [50] 提出了一种带有数据选择的跨版本缺陷预测模型 CDS, 该模型为噪声较少的历史版本以及数据集内部与测试集相似的文件分配更高的权重. 通过对比发现, CDS 在 *F*-measure 和 G-mean 指标上优于其他的基线模型.

讨论 3: 对于性能优越的软件缺陷预测模型来说, 大量的训练数据必不可少, 但是跨项目缺陷预测、跨版本缺陷预测中的数据差异问题影响了软件缺陷预测模型的性能, 未来研究人员的研究应着重构建高质量、平衡且无噪音的数据集, 改善软件缺陷预测模型的性能.

3 评价指标

本节首先统计了目前软件缺陷预测研究的评价指标, 分析了研究者最常用的评价指标, 各项常用评价指标的具体描述如表 8 所示.

Table 8 Common Evaluation Indicators and Their Descriptions

表 8 常用评价指标及其描述

评价指标	具体描述
Accuracy	模型预测正确的个数占实例总数中的比例
Precision，Correctness	模型预测有缺陷的实例中真实类别为缺陷所占的比例
Recall，TPR	模型预测有缺陷的实际数量占真实有缺陷中的比例
Specificity，TNR	模型预测无缺陷模块的实际数量占真实无缺陷中的比例
FPR	模型预测有缺陷的模块占真实无缺陷中的比例
FNR	模型预测无缺陷的模块占真实有缺陷的比例
AUC	ROC 曲线下面积、AUC 值越大，模型的有效性越好
MCC	观察到的分类与预测分类的比值
Balance	PF 的最佳截止点，ROC 曲线中 (0, 1) 点的归一化欧几里得距离
F-measure	是召回率和精确度之间的调和平均值
*F*1-Score，*F*2-Score	不平衡数据集学习的评价标准，表示精准率和召回率的组合
G-mean	Recall 和 Precision 的几何平均数
Error Rate	所有实例中错误分类的比率
AAE	平均绝对误差，表示预测值和实际值之间的绝对差
ARE	平均相对误差，表示预测值和实际值的绝对差与实际值的比值
Completeness	实际缺陷值与预测缺陷值的比值

本文的预测指标的研究数量占比如图 5 所示, 可以发现, Recall, Precision, AUC, *F*-measure 是最常用的评价性能指标. 经调查发现, 研究人员对于模型的评价指标也有争议: 文献 [51] 认为一个好的软件缺陷预测模型应该同时实现高召回率和高准确率; 文献 [52] 对 *F*-measure 和 Precision 进行了批判, 该文作者认为不是所有好的预测模型都需要高召回率和高准确率; 文献 [53] 发现使用 MCC, 而不是使用 *F*1-Score 作为软件缺陷预测的评价指标, 超过 1/5 的模型对比结果

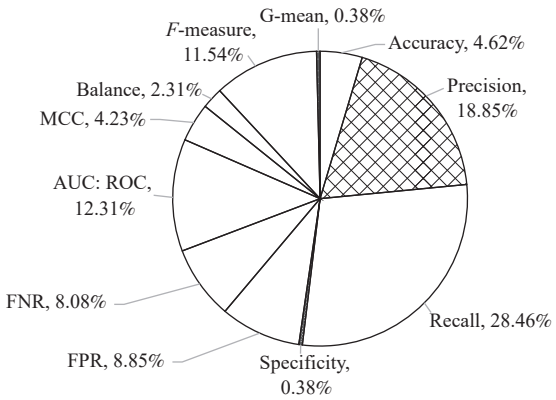


Fig. 5 Summary of evaluation indicators

图 5 评估指标统计

会发生改变.

软件缺陷预测研究的评价指标调查结果表明,不同模型在不同评价指标上的表现不同,研究者不应局限于单个指标评估不同模型的优劣,应结合多种评价指标,针对不同的模型采用多样性的评价指标进行评估.

4 缺陷预测模型

4.1 基于软件度量的缺陷预测

软件度量是在软件开发过程对软件进行数据收集、定义和分析的一个持续性定量化过程,描述软件产品或者开发过程的指标或者参数,以保证高效率、低成本和高质量的软件开发过程[54]. 度量元是最细粒度的软件度量属性,用于描述特定的度量特征,度量元主要分为代码度量元和过程度量元. 不同的度量元对软件缺陷预测模型的影响不同,为了提高软件缺陷预测模型的性能,一些研究者使用混合度量元搭建软件缺陷预测模型.

基于软件度量的缺陷预测模型如图 1 所示,模型首先从开源公共仓库或者开源社区中抽取程序模块,然后定义与软件缺陷强相关的度量元并提取对应的度量特征从而构造出缺陷数据集,最后通过软件缺陷数据集训练软件缺陷预测模型,软件缺陷预测模型就能够预测目标程序模块的软件缺陷相关信息.

如图 6 所示,早在 20 世纪 70 年代,传统的代码度量基于代码行(lines of code, LOC)[15],随后研究人员发现软件缺陷不仅取决于软件规模,还取决于软件复杂度,于是提出了 Halstead 度量[16]、McCabe 度量[55]. 20 世纪 90 年代,随着面向对象技术的发展,一些研究者开始关注模块的内聚性、耦合度等特征,所以使用面向对象度量元衡量软件规模和复杂度. 1994 年,Chidamber 和 Kemerer 基于面向对象程序的继承、耦合、内聚特征给出了 6 个类级别的 CK 度量标准[56]. 文献 [57] 提出了 MOOD 度量,包含 6 个度量标准. 2002 年,文献 [58] 提出了一套 QMOOD 度量,在系统和类的层次上评估软件设计. 2021 年,文献 [59]第一次将代码气味作为跨项目缺陷预测的特征,使用 RF、支持向量机(support vector machine, SVM)、多层感知机(multilayer perceptron, MLP)、决策树(decision tree, DT)、朴素贝叶斯(naive Bayes, NB)构建软件缺陷预测模型. 结果表明,基于代码气味训练的跨项目缺陷预测模型的性能相较于基于代码度量的缺陷预测模型提高了 6.5%,且基于代码气味训练的缺陷预测模型的表现效果优于代码气味和代码度量等混合度量指标训练的缺陷预测模型.

研究人员发现软件缺陷不仅与软件规模有关,还与软件开发过程的各阶段有关,因此提出了一系列的过程度量. 代码变更可以度量代码在不同版本之间的增加、删除或者修改量,它很容易从版本控制系统自动记录的系统更改历史中提取. 20 世纪 90 年代,文献 [60] 使用代码变更作为软件质量的衡量标准. 随后,文献 [61] 提出了 8 种相对代码变更指标,预测软件缺陷密度. 文献 [62] 使用了 LR, NB, DT 这 3 种分类模型,发现对于 Eclipse 数据,代码变更指标比代码度量能更好地预测缺陷.

代码度量和过程度量并不能很好地描述软件模块如何随项目演变而变化,而演化度量[63-65]是一种专门针对项目演变过程的度量元. 文献 [64] 使用类的年龄、类出现缺陷的可能性、类连续且不含有缺陷的周期来刻画软件演化过程中类的演化模式,研究结果发现,与代码度量和代码变更相比,演化度量元能更好地预测软件缺陷. 文献 [65] 将多个连续版本的文件度量按版本升序连接在一起,利用循环神经

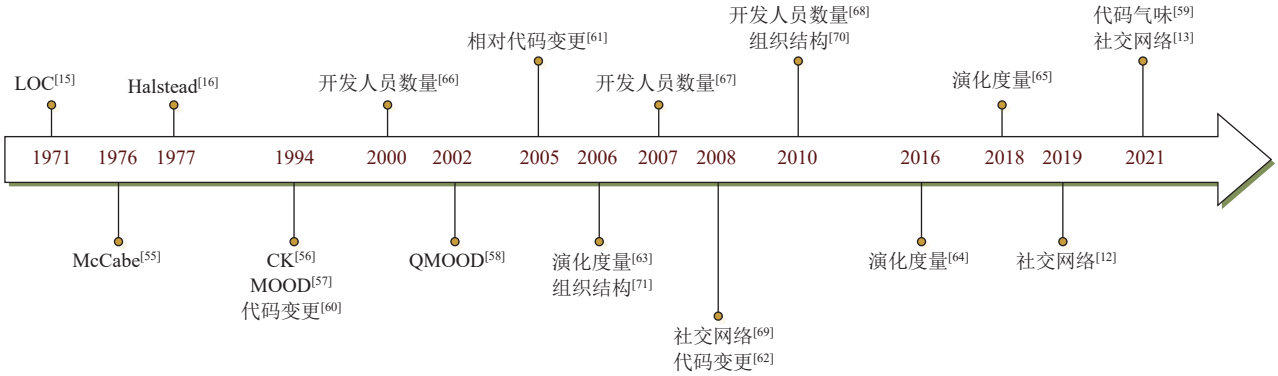


Fig. 6 Timeline of metrics development

图 6 度量元发展时间线

网络，提供了一个新的视角来解释文件如何随项目的发展而变化的趋势.

软件缺陷的产生与开发人员密切相关，早期研究人员仅仅将修改模块的开发人员数作为缺陷预测的度量元，但是后续研究发现软件缺陷与开发人员的数量无关[66-68]. 随后，一些研究人员从社交网络[12-13,69]分析开发人员的经验，从程序模块的所有权等角度分析开发人员与缺陷之间的关系. 文献 [69] 使用网络中心度来衡量开发人员贡献的碎片化程度，开发人员的数量越多，代码修改的次数也相应增多，导致软件含有的缺陷数量也越多，研究结果表明中心模块比网络周围的模块更容易产生软件缺陷.

随着全球软件开发的出现，研究人员发现组织结构[70-71]也影响着软件的质量. 文献 [71] 提出了 8 项量化组织复杂性的度量用于研究组织结构与软件质量之间的关系，实验结果表明，组织指标比传统代码变更、代码复杂性、代码依赖等指标的预测效果更佳. 2005—2010 年，软件缺陷预测结合各种度量元，使用各种机器学习算法提高模型的准确性和性能. NB[72]、贝叶斯网络（Bayesian network, BN）[73]、*K*-means[74-75]、AdaBoost[76]、DT[76]、LR[72-73,76] 等方法在此期间展现了较好的性能. 2010 年后，相比于传统的机器学习方法，一些新的机器学习[14,50,77-79]方法提高了软件缺陷预测的性能. 现有的跨版本缺陷预测模型通常只使用一个先前版本收集的数据进行模型训练，并没有考虑在具有多个先前版本的跨版本缺陷预测场景中数据分布差异方差和类重叠问题. 文献 [14] 提出了一种基于朴素贝叶斯的改进转移朴素贝叶斯，结果表明该模型在项目内缺陷预测和跨项目缺陷预测中的准确率、精度、PD 都优于转移朴素贝叶斯模型. 文献 [50] 提出了一种新颖的基于聚类的多版本分类器 CMVC，为每个先前版本分配适当的权重，并通过最小化目标函数来获得现有文件的预测标签，结果表明该方法优于 RF, LR, NB 等模型.

随着深度学习的应用，一些研究人员提出了基于深度学习[65,80-81]的缺陷预测模型. 文献 [80] 探索了 Siamese 网络的优势，提出了一种新的软件缺陷预测模型 SDNN. 与以往的方法相比，SDNN 使用 2 个完全相同的全连通网络学习最高相似度特征，并使用计量函数作为最高相似度特征之间的距离度量. 结果表明，在 *F*-measure 方面，SDNN 方法优于深度神经网络（deep neural network, DNN）、长短期记忆网络（long short-term memory, LSTM）、深度信念网络（deep belief network , DBN）、NB 和 LR 这 5 种方法.

文献 [81] 提出了一种 Defect-Learner 模型，从代码自然性的角度，将交叉熵作为一种新的软件度量方法引入到文件级缺陷预测. Defect-Learner 模型在学习阶段，构建了一个多层的 LSTM 网络，从输入的标记向量中学习隐含的语义特征. 在预测阶段，将交叉熵度量与其他度量特征结合，用来预测目标项目中的缺陷. 结果表明，使用结合交叉熵的 CK, QMOOD 等度量，能够改进软件缺陷预测的性能.

讨论 4: 本节按时间顺序归纳了代码度量和过程度量，图 6 展示了度量元发展历程，度量元的演变过程及其对比结果如表 9 所示. 从分析来看，软件的演化是一个动态的连续过程，新增模块或者修复历史缺陷都会产生新的缺陷，未来研究应该关注演化度量元，提高软件缺陷预测的性能.

观点 1: 目前大多数研究者是手工定义与软件缺陷相关的度量元，忽略了软件缺陷在软件生命周期中的产生方式，未来如何从存储库中自动提取与缺陷相关的度量元仍是研究的热点.

Table 9 Comparison of Metric Evolution

表 9 度量元的演进对比

来源	时间	度量元	机器学习/深度学习	评价指标	对比结果
文献 [59]	2021	代码气味 代码度量	RF，SVM，MLP，DT，NB	ROC，AUC，PR，*F*1-Score	代码气味优于代码度量且优于这两者的混合度量
文献 [62]	2008	代码变更 代码度量	LR，NB，DT	FP，Recall，PC	过程度量比代码度量更有效
文献 [64]	2016	演化模式度量元 代码变更度量元	NB、二元逻辑回归、J48 决策树	Precision，Recall，*F*-measure，ROC	与代码和代码变更相比，演化度量元有相对较好的预测性能
文献 [81]	2018	交叉熵	基于 LSTM 的循环神经网络（RNN）	Precision，Recall，*F*1-Score，AUC	交叉熵度量比 50% 的传统度量有更好的预测能力

4.2 基于语法语义的缺陷预测

基于语法语义的缺陷预测模型如图 2 所示，模型首先通过 AST 等技术对源代码进行解析从而提取语法语义特征信息，然后通过软件缺陷相关的词嵌入方法将语法语义特征编码为向量，最后将生成的向量数据集送入神经网络模型训练得到缺陷预测模型.

2015 年开始, 深度学习、人工神经网络在软件缺陷预测中得到广泛应用. 与传统的软件度量不同, 研究者使用卷积神经网络(convolutional neural network, CNN)、循环神经网络(recurrent neural network, RNN)、MLP、DBN、LSTM 和门控循环神经网络(gated recurrent unit, GRU)等深度学习算法, 通过 AST 等方式生成的标记向量提取特征, 从而挖掘软件程序中隐藏的语义和结构特征, 并将生成的特征数据反馈到神经网络中, 训练获得缺陷预测模型. 文献 [82] 提出了一种用于即时缺陷预测的深度学习方法, 利用 DBN 在一组初始变化特征中提取了表达特征, 在选定的特征上构建了缺陷预测模型. 但具有不同语义的程序文件可能具有相同的传统特征, 该方法无法区分不同语义的代码区域.

鉴于此, 文献 [83] 采用编辑距离相似度计算算法和 CLNI 剔除可能存在错误标签的数据, 并通过 AST 分析程序源代码获取语法信息, 最后将语法信息转换为特征向量并输入到 DBN 网络中从而建立预测模型. 结果表明, 该模型提高了项目内缺陷预测和跨项目缺陷预测的性能, 与传统特征相比, 该方法平均精度提高了 14.7%, 召回率提高了 11.5%, *F*1 提高了 14.2%. 文献 [84] 在文献 [82−83] 所述的文件级缺陷预测的基础上考虑了代码变更, 提出了一种启发式方法, 从代码变更片段中提取重要的结构信息和上下文信息; 对于变更级缺陷预测, 该方法从代码中提取标记, 并生成基于 DBN 的特征; 对于文件级缺陷预测, 使用源文件的完整 AST 生成 DBN 的语义特征. 结果表明, 在 *F*1 指标上, 该方法在文件级和变更级缺陷预测方面上都优于传统的 CPDP 和 WPDP 模型. 文献 [85] 在 DBN 模型提取语义特征的基础上, 将 CNN 学习到的特征与传统的缺陷预测特征相结合, 提出了一种基于 CNN 的缺陷预测框架, 从程序的 AST 中生成判别特征, 并且保留了程序的语义和结构特征. 结果表明, 在 *F*-measure 方面, 该方法比基于 DBN 的方法提高了 12%, 比传统的基于软件度量的方法提高了 16%. 文献 [86] 提出了一种基于注意力的 RNN 的软件缺陷预测(defect prediction via attention-based recurrent neural network, DP-ARNN)框架, 通过解析程序的 AST 并将语法信息转换为向量, 然后使用字典嵌入和词嵌入对向量编码并输入到 ARNN 网络中, 利用 ARNN 自动学习语法结构特征和语义信息内容, 最后通过注意力机制生成关键特征向量. 与 RNN 相比, DP-ARNN 在 *F*-measure 上平均提升了 3%, 在 AUC 上平均提升了 1%.

深度学习方法(CNN, DBN)使用 AST 生成的标记向量以提取深度学习特征. 文献 [87] 发现, CNN 和 DBN 结合传统软件度量和 TCA 算法, 可以将源项目中提取到的深度学习特征用于目标项目, 解决跨项目中分布不平衡问题. 实验结果表明, 将深度学习生成的特征和传统特征结合起来, 比纯 DBN 和 CNN 模型的表现效果要好. 文献 [88] 提出了一个新的跨项目缺陷预测方法 S^2LMMD, 首先通过在指定节点拆分原始 AST 构造联合学习语句级树 SLT, 从而捕获更精细的语义和结构信息; 然后使用双向门控循环神经单元(bidirectional gated recurrent unit, Bi-GRU)学习序列嵌入并生成更有效的语义特征; 最后使用最大平均偏差(maximum mean discrepancy, MMD)技术缓解跨项目中的数据分布差异, 提高缺陷预测性能. 实验结果表明, 在 AUC 指标上, S^2LMMD 优于 DBN 和 CNN. 源代码中的语法和不同级别的语义信息通常由树的结构表示, 文献 [89] 实现了一种基于树结构的 LSTM 网络(tree-structured LSTM network, Tree-LSTM), 使用 Tree-LSTM 匹配代码的 AST, 充分捕获源代码中的语法和不同级别的语义, 使用注意力机制定位源文件中可能存在缺陷的部分.

文献 [90] 提出一种基于图卷积神经网络(graph convolutional network, GCN)的缺陷预测模型. 首先该模型从源代码中提取了 7 种节点特征, 根据节点特征对控制流图(control flow graph, CFG)节点进行分类并生成节点特征值, 最后为 CFG 创建节点向量矩阵并将其传递给图卷积网络以自动学习特征. 结果表明, 该模型可以适应不同规模的软件, 在 AUC, *F*1-score 等指标上优于 CNN 等传统的缺陷预测模型.

缺陷特征隐藏在程序语义中, 但是 AST 不显示程序的执行过程, 它只代表源代码的抽象句法结构. 文献 [91] 采用 CNN 的思想, 将源代码转换为程序控制流程图, 采用多视图多层 CNN 从 CFG 中自动提取学习缺陷特征, 结果表明图学习可以提高基于传统特征和基于 AST 方法的性能.

文献 [92] 提出了一种基于增强代码属性图(augmented-code property graph, Augmented-CPG)模型 ACGDP, 用于捕获代码的语法、语义、控制流和数据流信息. 该模型在初始阶段从缺陷代码文件中获得 AST 和 CFG, 与代码的控制流和数据流合并, 生成 Augmented-CPG, 第 2 阶段提取候选缺陷区域(graph of defect region candidates, GDRC)定位可能存在的缺陷节点, 第 3 阶段从 GDRC 中获取代码嵌入, 使用图神经网络(graph neural network, GNN)学习 GDRC 中包

含的缺陷特征. 结果表明, 该模型在 Accuracy, Precision 等指标上优于 VulDeePecker[93] 模型.

代码注释是源代码的另一种视图, 帮助生成反映代码功能的语义特征, 并识别软件中含有缺陷的模块. 文献 [94] 提出了一种评论增强程序的卷积神经网络(convolutional neural network for comments augmented program, CAP-CNN)缺陷预测模型, 该模型在训练过程中将代码注释编码为语义特征, 结果表明 CAP-CNN 在 *F*-measure 指标上优于 CNN.

近几年, 一些改进的算法被提出来用于软件缺陷预测, 网络嵌入技术取得重大进展. 文献 [95] 提出了 Node2defect, 一种基于随机游走的网络嵌入技术的缺陷预测模型, 该模型能够自动学习软件类依赖网络的类结构特征并挖掘节点邻接信息和节点属性, 最后将学习到的特征与传统度量连接并输入到后续预测模型中. 结果表明, 该模型在 *F*-measure 指标上比传统的软件度量方法提高了 9.15%. 文献 [96] 提出了 GCN2defect, 一种基于 GNN 的缺陷预测模型, 该网络学习类依赖网络中节点的结构特征, 并对节点的语义和结构新信息进行端到端的学习. 文献 [96] 的作者引入了传统静态代码度量、复杂网络度量和网络嵌入 3 种类型的节点度量作为节点的属性, 与传统的软件度量和传统的网络嵌入相比, 该方法显著提高了缺陷预测性能. 文献 [97] 在此基础上提出了 CGCN 模型, 使用 CNN 捕获 AST 中的语义信息, GCN 捕获软件网络中的结构信息, CGCN 通过这 2 种方式获得联合特征, 并与传统特征结合用于训练分类器. 结果表明, 该模型优于 Node2defect 和 GCN2defect 等方法.

文献 [98] 使用 Transformer 模型自动学习序列的语义特征和句法结构特征, 预测代码的缺陷密度. 文献 [99] 考虑了源代码的语义和上下文信息, 采用 BERT 和双向长短期记忆网络(bidirectional long short-term memory, BiLSTM)模型预测缺陷, BiLSTM 通过 BERT 模型学习的 token 向量学习上下文的语义信息. 结果表明, 与现有的深度学习和传统特征相比, 该模型更能有效提取程序的语义信息.

讨论 5: 本节统计分析了基于语法语义的缺陷预测模型, 根据分析结果发现, 缺陷特征隐藏在程序语义中, 大多数研究者将源代码解析为 AST, 利用深度学习技术学习程序的语法和句法特征.

观点 2: 缺陷特征隐藏在程序语义中, 但是 AST 不显示程序的执行过程, 它只代表源代码的抽象句法结构, 未来研究者应使用源代码的多种表征方式, 帮助生成反映代码功能的语义特征.

5 漏洞预测

漏洞是一种特定的软件安全缺陷, 存在能够被攻击者利用并造成危害的安全隐患. 与软件缺陷研究领域不同, 漏洞的研究领域可以分为漏洞检测与漏洞预测 2 个任务场景.

漏洞检测和漏洞预测 2 个任务侧重的研究目的不同. 漏洞检测任务针对已知的漏洞, 在特定的软件中检测是否存在目标漏洞. 漏洞预测任务则主要负责从宏观角度进行数据上的统计, 预测一个代码文件中可能存在的漏洞数量, 以减少漏洞发现和修复的成本. 但随着人工智能尤其是深度学习方法的大量引入, 漏洞检测和漏洞预测模型愈发相似, 由于都使用大规模精细标注的数据集进行训练, 任务的目的和实现的方式也渐渐重合, 在近年的许多研究工作中已经不再区分这两者的概念.

本节以漏洞预测过程为切入点, 分别介绍了传统的基于软件度量的预测方法以及基于语法语义的预测方法.

5.1 漏洞预测模型

5.1.1 基于软件度量的漏洞预测

基于软件度量的漏洞预测模型通过人工设计软件特征表示软件漏洞相关信息, 再使用机器学习算法预测并识别漏洞相关的组件. 漏洞预测模型使用复杂度、代码变更、开发者活动、耦合、内聚等度量, 并使用 LR, SVM, J48, DT, RF, NB, BN 等机器学习算法预测漏洞相关的组件.

文献 [100] 对 Mozilla 中的 JavaScript 引擎进行了案例研究, 发现 9 种复杂性度量(包括 McCabe, SLOC 等度量)可以用于预测漏洞, 但是误报率较高. 文献 [100] 的作者发现, 漏洞函数和缺陷函数的复杂性度量仅在复杂度方面存在显著差异, 因此复杂度可以区分漏洞和缺陷函数. 文献 [101] 使用源代码行、代码变更等度量预测漏洞, 结果表明该回归树模型在最佳情况下实现了 100% 的召回率和 8% 的假阳率. 文献 [102] 研究了开发人员协作结构与漏洞之间的关系, 研究发现, 由 9 个或者更多开发人员更改的文件比少于 9 个开发人员更改的文件具有漏洞的可能性高 16 倍. 文献 [103] 研究了 3 种软件度量指标(复杂性、代码变更、开发人员活动)是否可以区分易受攻击的文件来指导安全检查和测试. 结果表明, 28 个度量中至少有 24 个可以区分 2 个项目的易受攻击文件和中性文

件, 这 3 种度量可以为安全检查和测试工作提供有价值的指导. 文献 [104] 研发出一个基于复杂性、耦合性和内聚性相关的漏洞预测框架, 该框架能正确预测 Mozilla Firefox 的大多数受漏洞影响的文件, 且误报率较低.

观点 3: 基于软件度量的漏洞预测研究提出了大量的指标, 试图尽可能完整地从软件代码中手动提取出漏洞相关的特征. 但这些人工定义的特征并不一定能够完全理解代码本身的含义, 因此在实际的漏洞预测中仍具有较多不足.

例如在图 7 中的 2 段 Python 代码, 从功能上来看都是从栈顶弹出 5 个数据, 但代码 1 在栈中数据不足 5 个时有向下溢出的风险, 代码 2 则没有. 但这 2 段代码具有相同的软件度量、代码标记和频率等, 仅依靠人工提取的指标难以对二者进行区分, 需要一种能够提取代码的语法结构特征和语义信息内容的算法.

```
1 x=0
2 if stack. is Empty():
3    while x<5:
4       y=stack. pop()
5       x=x+1
```

代码 1: A. py

```
1 x=0
2 while x<5:
3    if stack. is Empty():
4       y=stack. pop()
5       x=x+1
```

代码 2: B. py

Fig. 7 Code example

图 7 代码示例

5.1.2 基于语法语义的漏洞预测

基于语法语义的漏洞挖掘通过文本挖掘、AST 等方式提取软件代码中的特征信息, 转化为向量的数字表示, 随后使用机器学习(machine learning, ML)或深度学习(deep learning, DL)方法进行分类. 使用图算法或者注意力机制还可以具体定位可能出现漏洞的位置. 基于语法语义的研究主要集中在探究不同的特征提取方法和各种训练模型.

文献 [105] 提出了一种依赖于源代码文本分析的方法, 将每个文件转换为一个特征向量. 该方法将源代码的每一个字母组合都视为特征, 使用给定文件源代码中给定字母组合的计数值. 该研究在开源移动应用程序上实现了 87% 的准确率、85% 的精度和 88% 的召回率. 文献 [106] 将文件中的代码标记和频率作为代码的文本特征, 预测可能存在漏洞的组件, 该模型相比于 PHP 应用程序的软件度量模型实现了更高的召回率. 但是对词袋模型来说, 它忽略了代码之间的语法结构信息, 隐藏在程序中的语义信息可以帮助易受攻击的代码提供更丰富的表示, 从而改进漏洞预测模型的效果. 此后的研究引入了 AST 来表征句法结构, 文献 [107] 从代码中提取了 AST 并确定树的结构模式, 根据树的结构模式自动比较代码, 该方法仅通过检查少量代码库就能进行漏洞预测. 文献 [108] 使用 Antlr 从 C/C++源文件中提取 AST, 以每个函数为基本单元从 AST 中提取向量, 利用该向量训练分类器来预测缓冲区漏洞.

许多深度学习技术也开始应用于漏洞预测任务. 深度学习技术可以从复杂的代码中自动获取更深层次的特征信息, 表征程序的语法和语义特征. 文献 [109] 在令牌(token)级数据上, 采用深度神经网络, 结合 N-gram 分析和特征选择构造特征, 该模型能够在 Java Android 应用程序漏洞预测任务中实现高精度、高准确率和高召回率. 文献 [110] 利用 LSTM 捕获源代码中的上下文关系, 学习代码的语法语义特征, 结果表明该方法优于传统的软件度量等模型. 文献 [111] 研究了基于文本挖掘的漏洞预测中词嵌入算法的价值, 使用 word2vec 和 fast-text 两种模型学习代码标记之间的语法语义关系, 使用 CNN 和 RNN 模型预测文件中是否含有漏洞. 文献 [111] 的作者发现, 使用用于生成单词嵌入向量的算法时, 模型的 $F2$-score 增大, 且 word2vec 算法效果最佳. 基于深度学习的漏洞预测模型也成为了研究的热点.

文献 [93] 提出 VulDeePecker 系统, 在漏洞预测任务中引入了深度神经网络 BiLSTM, 同时采集了一个用于评估漏洞检测效果的数据集. 文献 [112] 建立了一套基于深度学习的漏洞预测系统, 能针对输入的文件或函数进行粗粒度预测, 也能针对新实例中的某一小代码块, 进行某几种漏洞的细粒度预测和定位. 文献 [113] 提出基于 GNN 的 Devign 模型, 提取了 4 种属性 AST、CFG、数据流图(data flow graph, DFG)和自然语言中的代码序列以完成对代码的表征. 文献 [114] 提出的 SySeVR 模型使用基于数据依赖的语义信息表征代码. 文献 [115] 提出了 IVDetect 模型, 提取代码中的数据和控制依赖关系生成 5 个维度的向量表示, 统一交由 RNN 模型训练学习, 实验结果表明, IVDetect 模型带来了精度的提升并能够定位到漏洞所在函数块. 基于此, 文献 [116] 提出了 LineVul 模型, 使用 CodeBERT 预训练模型完成代码的词向量转化, 使用行粒度数据集进行训练, 使用 GNN 捕获文本中的依赖关系, 模型得到更高预测准确性的同时能够定位漏洞所在代码行.

讨论 7: 从文本挖掘的角度来提取文件中的漏洞特征能够更充分地捕获代码的语法语义特征. 计算

机算力的提升和深度学习方法的引入也使得研究者得以从复杂冗长的代码中提取出丰富的特征. 这一领域的学者们主要围绕如何完成代码到连续值向量的转换以及深度学习模型的搭建对漏洞预测任务的优化提升展开研究. 通过总结归纳发现在代码特征表征阶段, AST 和 RNN 这 2 个思路被大多数研究者所选择, AST 可以很好地辅助理解代码中的逻辑结构, RNN 则可以在代码量巨大的文件中捕获长距离的依赖关系. 除了 AST 和 RNN, 近年来的工作开始逐渐转向大规模语言模型(例如 CodeBERT), 这些大规模语言模型在程序语言任务上进行了预训练, 因此可以更好地捕获和理解程序语言特征以应用于各项程序语言相关的下游任务.

观点 4: 即便使用了 AST 来表征代码的语法结构, 如何用向量尽可能充分地表征一个代码文件依旧是研究的热点所在. 当代码量过大时, RNN 对于代码中的依赖关系并不能充分捕获, 梯度消失问题依旧存在, 所以或许一种新的学习模型能够给漏洞预测领域的研究带来突破. 此外, 目前漏洞预测的结果较为单一, 无法全方位多角度地描述漏洞, 因此更细粒度的漏洞预测结果也是近年来研究的一个热点.

5.2 漏洞预测和缺陷预测的区别与联系

5.2.1 漏洞与缺陷的区别与联系

漏洞与缺陷不是 2 个完全分离的概念, 具体来说漏洞是一种特殊的安全性缺陷.

文献 [117] 表示软件缺陷是导致功能单元无法执行其所需功能的功能性缺陷, 软件漏洞是软件规范、开发或配置中的缺陷实例, 漏洞的执行会违反安全策略. 文献 [118] 认为漏洞是缺陷的子集, 漏洞是一种特定类型的安全性软件缺陷. 文献 [119] 表示, 漏洞是能够被人利用来进行入侵行为的缺陷, 漏洞的出现是人尝试攻击的结果, 而不是机器运行的结果.

根据文献的调研结果, 缺陷与漏洞之间的相似性与不同如表 10 所示. 漏洞与缺陷的产生都与硬件[120]、代码的复杂性[117-119] 以及编程人员能力[121] 有关, 且都会对软件造成巨大的影响. 文献 [120] 指出硬件在漏洞和缺陷的产生过程中扮演了重要的角色. 绝大多数的电子设备都采用典型的片上系统(system-on-chip, SoC)作为设备的核心部件, 它由多个知识产权(intellectual property, IP)组成, 包括处理器、内存和输入输出设备等. 为了降低 SoC 设计成本并且满足上市时间的要求, SoC 的开发涉及多个第三方公司的供应链. 然而, 第三方的 IP 依赖性也引起了硬件安全问题, 因为从不受信任的供应商收集的硬件 IP 带有硬件木马、后门等其他问题. 文献 [121] 指出软件开发者的经验和代码复杂性与缺陷密切相关, 该文作者对 2003—2011 年期间 Linux 的不同版本进行了分析, 研究发现尽管 Linux 的规模在不断增加, 但缺陷数量呈现减少的趋势. 此外, Block, Null 等类型缺陷仍然在新的文件中引入和修复, 但这些类型缺陷的影响各不相同. 与缺陷不同的是, 攻击者可以利用漏洞实施一系列攻击, 造成更为严重的后果.

Table 10 Differences and Connections Between Defects and Vulnerabilities

表 10 缺陷和漏洞的区别与联系

区别与联系	角度	缺陷	漏洞
区别	概念	软件或程序中存在的某种错误或隐藏的功能故障	软件在设计、实现、配置策略及使用过程中出现的缺陷, 它可能导致攻击者在未授权的情况下访问或破坏系统.
	来源	软件架构和设计	软件代码(源代码或二进制代码)
	产生原因	测试范围过小, 需求分析不精准, 团队职责不规范, 硬件配置、固件、处理器中的缺陷, 软件配置、操作系统中的缺陷	编程人员的能力, 硬件缺陷, 软件缺陷, 协议缺陷
	披露方式	软件存储库中会对缺陷进行披露, 缺陷数据的质量高于漏洞数据的质量	漏洞的披露会引发一系列的攻击, 开发人员和漏洞研究人员通常会限制公开披露漏洞的信息
	数量	较多	较少
联系	概念	漏洞是可能被攻击者利用从而实施入侵的软件缺陷	
	来源	与硬件、代码的复杂性以及编程人员的能力有关	
	影响	会对企业和人们的生活造成巨大的伤害	
	检测和预测方法	手动测试、自动化、静态分析、动态分析、代码审查、人工智能	

文献 [122] 的结果表明, 没有一套通用的指标可以有效地预测漏洞, 漏洞预测不像缺陷预测那么容易, 在相同的度量元下, 缺陷预测可以获得合理的精度和召回率, 但是漏洞预测的精度和召回率却不尽如人意. 文献 [123] 使用 3 种传统的缺陷度量构建了缺陷预测模型和漏洞预测模型, 测量缺陷预测模型

预测漏洞文件的准确性,结果表明传统的缺陷预测指标可以检测大部分易受攻击的文件,但是误报率较高. 文献 [124] 发现,缺陷预测和漏洞预测模型提供了相似的预测性能,当模型使用传统的软件度量时,两者可以互换使用,且缺陷和漏洞预测的性能在很大程度上受先前版本中报告的缺陷数量和漏洞数量的影响.

讨论 8:基于软件度量的缺陷预测和漏洞预测提供了相似的预测性能. 但是传统的软件度量不能识别语义不同的代码,因为大多数情况下 2 段代码具有相同的复杂度,但是受到的攻击可能不同. 未来研究者可以从语义角度自动提取缺陷和漏洞相关特征,探究缺陷预测模型在漏洞预测问题上的可移植性.

5.2.2 基于语法语义的预测模型

随着人工智能技术在各个领域表现出强大的预测能力,近年来安全研究领域也引入人工智能技术用于执行缺陷预测和漏洞预测任务.

缺陷和漏洞预测的研究需要提取不同的软件度量,基于人工智能技术的预测模型也逐渐有了类似的结构,包括向量表征、提取特征和 ML/DL 分类器训练等流程. 通过对缺陷和漏洞任务中的深度学习模型统计发现,CNN, RNN, DBN 等模型均是缺陷和漏洞预测任务中的常见方法,大量的研究工作用它们搭配不同的特征提取方法进行预测.

AST 能够高效地表征程序语言的语法语义,因此 AST 特征提取是目前最主流的缺陷预测和漏洞预测的向量表征方法. 由于软件代码平均长度较长,所以许多研究也开始使用 LSTM 模型来捕获长距离代码之间的语义依赖关系,基于 LSTM 模型的注意力机制被广泛用于定位具体缺陷和漏洞的位置. 随着 BERT 模型的出现,一些大规模语言模型加入数据流、控制流等深层次的程序语言依赖关系来帮助模型更好地捕获和理解程序语言的语义特征.

讨论 9:随着深度神经网络的可解释性逐步增强,面临具体问题时应当选择的网络模型也愈发明晰. 而缺陷与漏洞的预测模型越来越相似,也从某方面说明 2 个任务在实现思路上具有较强的共性,缺陷预测的可移植性问题也值得探究.

6 未来研究展望

本节通过梳理和归纳缺陷预测和漏洞预测任务的研究成果,总结了这 2 个研究领域现阶段面临的挑战与机遇,如表 11 所示.

Table 11 Opportunities and Challenges of Defect Prediction and Vulnerability Prediction Tasks

表 11 缺陷预测和漏洞预测任务的挑战与机遇

挑战	机遇
数据集的来源与处理	建立一个高质量平衡且无噪音的基准数据集
代码向量的表征方法	构建一种最大程度蕴含语法语义信息的表征方法
预训练模型的提高	利用在其他领域训练好的词向量嵌入提升模型性能
深度学习模型的探索	探索更适合具体预测任务的深度学习模型
细粒度预测技术	更加精确地定位缺陷和漏洞可能出现的位置
预训练模型的迁移	通过模型的迁移节约时间和资源成本

6.1 数据集的来源与处理

缺陷预测常用数据集及其属性如表 4 所示,研究者主要使用 NASA, PROMISE, AEEEM 缺陷公共仓库中的 CM1, JM1, KC1, KC2, KC3 等数据集进行缺陷预测相关研究,但这些缺陷数据集大多数已经无人维护.

随着深度神经网络(deep neural network, DNN)在缺陷预测模型中的发展,预测模型对于数据集质量的要求也越来越高. 这些公共数据集不包含缺陷相关的细节信息,不能给 DL 模型提供充分的特征信息. 因此,创建新的数据集、增加多标签预测是未来的研究点.

除了数据集的来源,数据集的标注也是一个值得关注的方面. 当前研究发现通过 GNN 或者注意力机制等方法能够具体定位到预测缺陷和漏洞出现的函数块或代码行,并且数据标注越细粒度,模型所能捕获的信息也越具体.

目前软件缺陷预测和漏洞预测使用的数据集往往具有类不平衡、维度过高、预测特征不足、分类标签不足等缺点. 类不平衡和维度过高问题可以通过研究者人工规范化或者降维处理来缓解,但特征和标签不足的问题则只能寻求更加高质量、平衡且无噪音的数据. 因此,为了更好地预测缺陷、改善资源分配和修复缺陷代码,研究者需要创建一个高质量的缺陷数据集.

6.2 代码向量表征方法

目前的代码向量表征研究尝试了包括变量名称和类型、AST、程序依赖图、数据依赖在内的多种方法来表征代码蕴含的语法语义信息.

除了单一表征方法外,许多研究也着眼于多维度的混合表征方法,并给模型带来了一定的性能提升. 其中,AST 能够出色提取代码中语法结构特征,因此被广泛应用于各种模型的代码表征阶段. 但如何在不丢失语法结构特征和语义信息内容的前提下,

将源代码嵌入到向量空间，从而更好地完成代码表征工作，仍需要研究者们继续探索.

6.3 预训练模型的提高

人工定义的各种代码表征方法似乎已经走到一个瓶颈，当前许多工作尝试了各种方法来提取代码中的特征，但都不能全面地捕获程序语言的全部语义信息，于是有的研究工作开始尝试引入 code2vec 模型.

近年来出现的 Glove 等词向量模型给代码表征带来了一定的启发. 最近人工智能领域也提出了许多诸如 CodeBERT 的大规模预训练程序语言模型，但它不能深入地捕获到程序语言的语义级别信息，它的后继者 GraphCodeBERT 加入了数据流的依赖关系，得到了更多的代码语义表征，并在多个程序分析相关下游任务中取得了 SOTA 成果. 可以预见，引入更好的预训练模型也会给漏洞挖掘领域的 DL 方法带来极大提升.

6.4 深度学习模型探索

由于软件代码的长度较长，代码中经常出现远距离的数据和控制依赖关系，因此尽可能完整地提取其中的语法结构特征和语义信息内容一直是相关研究面临的一个难题.

选择合适的深度学习模型就可以帮助研究者更好地捕获语法语义信息，目前主流的观点认为使用具有记忆门单元的 RNN 系列模型可以帮助提取部分远距离依赖关系的特征，但有些研究指出 RNN 在更长的代码文本中同样力有未逮. 因此，更适合缺陷和漏洞的预测任务的深度学习模型仍待研究者们进一步探索.

6.5 细粒度预测技术

为了在实际应用中减少查找、修复缺陷和漏洞的成本，预测任务中也产生了定位可能出现的缺陷或漏洞的位置需求，因此如何更加细粒度地定位缺陷和漏洞也是未来研究工作的突破方向之一.

通过各种图算法或者注意力机制，目前大部分预测工作可以对代码文件或者具体函数进行缺陷和漏洞定位，甚至还有部分研究工作实现了将缺陷和漏洞具体定位到代码行. 可以预见，细粒度预测技术会随着算法的改进和数据集的精细化而得到提升.

6.6 缺陷预测和漏洞预测模型的迁移

缺陷和漏洞都与代码或者设计的复杂性有关. 漏洞和缺陷之间的相似性使得能够用传统的缺陷预测指标进行漏洞预测. 如果缺陷预测模型可以用于漏洞预测，则不需要花费额外的时间和资源为漏洞创建单独的模型.

基于软件度量的缺陷预测模型在一定程度上提供类似于漏洞预测的功能. 一些缺陷预测模型可能会具有预测漏洞的性能，但是这种缺陷预测模型并不是专门为预测漏洞而产生的，如果要预测漏洞则需要对模型进行一定的改进. 未来，研究者可以开发一种通用模型区分缺陷与漏洞.

7 结束语

软件缺陷预测借助机器学习或深度学习方法提前发现软件缺陷，可以减少软件修复成本并提高产品的质量. 研究开源软件缺陷预测，能够提高对各类项目的软件缺陷的认识，更好地指导代码检测和测试工作. 本文通过调研分析软件预测研究领域相关文献，以机器学习和深度学习为切入点，梳理了基于软件度量和基于语法语义的 2 类预测模型. 基于这 2 类模型，分析了软件缺陷预测和漏洞预测之间的区别与联系. 最后，对软件预测的六大前沿热点问题进行详尽分析，指出软件缺陷预测未来的发展方向.

通过总结软件缺陷预测相关的研究，本文认为，未来的研究方向可以从 4 个方面展开：1)创建一个高质量的缺陷数据集有助于预测缺陷、改善资源分配和修复缺陷代码；2)构建一种最大程度蕴含语法语义信息的表征方法、利用其他领域训练好的词向量嵌入能够提高缺陷预测模型的性能；3)适应细粒度的预测技术可以更加精确地定位缺陷和漏洞可能出现的位置；4)开发一种通用模型区分缺陷与漏洞，可以更好地预测易受攻击的代码位置.

作者贡献声明：田笑负责设计研究方案及论文撰写和最终版本的修订；常继友负责调研分析、数据统计及画图；张弛负责部分论文的撰写；荣景峰和王子昱负责调研分析；张光华、王鹤、伍高飞、胡敬炉负责论文的整体修订；张玉清提出论文的整体研究思路，及最终论文的审核与修订.

参 考 文 献

[1] Pachouly J, Ahirrao S, Kotecha K, et al. A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools[J]. Engineering Applications of Artificial Intelligence, 2022, 111: 1−33

[2] Chen Xiang, Gu Qing, Liu Wangshu, et al. Survey of static software

defect prediction[J]. Journal of Software, 2016, 27(1): 1−25 (in Chinese)
(陈翔, 顾庆, 刘望舒, 等. 静态软件缺陷预测方法研究[J]. 软件学报, 2016, 27(1): 1−25)

[3] Gu Mianxue, Sun Hongyu, Han Dan, et al. Software security vulnerability mining based on deep learning[J]. Journal of Computer Research and Development, 2021, 58(10): 2140−2162 (in Chinese)
(顾绵雪, 孙鸿宇, 韩丹, 等. 基于深度学习的软件安全漏洞挖掘[J]. 计算机研究与发展, 2021, 58(10): 2140−2162)

[4] Trachtenberg M. Discovering how to ensure software reliability[J]. Radio Corporation of America Engineer, 1982, 27(1): 53−57

[5] Qian Lianfen, Yao Qingchuan, Khoshgoftaar T M. Dynamic two-phase truncated Rayleigh model for release date prediction of software[J]. Journal of Software Engineering and Applications, 2010, 3(06): 603−609

[6] Bustamante A, Bustamante B. Multinomial-exponential reliability function: A software reliability model[J]. Reliability Engineering & System Safety, 2003, 79(3): 281−288

[7] Zheng Yanyan, Xu Renzuo. An adaptive exponential smoothing approach for software reliability prediction[C]//Proc of 2008 4th Int Conf on Wireless Communications, Networking and Mobile Computing. Piscataway, NJ: IEEE, 2008: 1−4

[8] Yamada S, Ohba M, Osaki S. S-shaped reliability growth modeling for software error detection[J]. IEEE Transactions on Reliability, 1983, 32(5): 475−484

[9] Kececioglu D, Jiang S, Vassiliou P. The modified Gompertz reliability growth model[C]//Proc of Annual Reliability and Maintainability Symp (RAMS). Piscataway, NJ: IEEE, 1994: 160−165

[10] Ahmad N, Imam M Z. Software reliability growth models with log-logistic testing-effort function: A comparative study[J]. International Journal of Computer Applications, 2014, 75(12): 8−11

[11] Gong Lina, Jiang Shujuan, Jiang Li. Research progress of software defect prediction[J]. Journal of Software, 2019, 30(10): 3090−3114 (in Chinese)
(宫丽娜, 姜淑娟, 姜丽. 软件缺陷预测技术研究进展[J]. 软件学报, 2019, 30(10): 3090−3114)

[12] Li Yiyao, Lee S Y, Wotawa F, et al. Using tri-relation networks for effective software fault-proneness prediction[J]. IEEE Access, 2019, 7: 63066−63080

[13] Lee S Y, Wong W E, Li Yiyao, et al. Software fault-proneness analysis based on composite developer-module networks[J]. IEEE Access, 2021, 9: 155314−155334

[14] Zhu Kun, Zhang Nana, Ying Shi, et al. Within-project and cross-project software defect prediction based on improved transfer naive Bayes algorithm[J]. Computers, Materials and Continua, 2020, 63(2): 891−910

[15] Akiyama F. An example of software system debugging.[J]. IFIP Congress, 1971, 71(1): 353−359

[16] Halstead M H. Elements of Software Science (Operating and Programming Systems Series)[M]. New York: Elsevier Science Inc, 1977

[17] Shepperd M, Song Qinbao, Sun Zhongbin, et al. Data quality: Some comments on the NASA software defect datasets[J]. IEEE Transactions on Software Engineering, 2013, 39(9): 1208−1215

[18] Khoshgoftaar T M, Gao Kehan, Napolitano A, et al. A comparative study of iterative and non-iterative feature selection techniques for software defect prediction[J]. Information Systems Frontiers, 2014, 16(5): 801−822

[19] Li Zhiqiang, Jing Xiaoyuan, Zhu Xiaoke, et al. Heterogeneous defect prediction through multiple kernel learning and ensemble learning[C]//Proc of 2017 IEEE Int Conf on Software Maintenance and Evolution (ICSME). Piscataway, NJ: IEEE, 2017: 91−102

[20] Kubat M, Matwin S. Addressing the curse of imbalanced training sets: One-sided selection[C]//Proc of the 14th Int Conf on Machine Learning. San Francisco: Morgan Kaufmann, 1997: 179−186

[21] Kotsiantis S B, Pintelas P E. Mixture of expert agents for handling imbalanced data sets[J]. Annals of Mathematics, Computing & Teleinformatics, 2003, 1(1): 46−55

[22] Chawla N V, Bowyer K W, Hall L O, et al. SMOTE: Synthetic minority over-sampling technique[J]. Journal of Artificial Intelligence Research, 2002, 16: 321−357

[23] Yao ZhenDan. Research on unbalanced data classification algorithm in software defect prediction[D]. Harbin: Harbin Normal University, 2022(in Chinese)
(饶珍丹. 软件缺陷预测中不平衡数据分类算法研究[D]. 哈尔滨: 哈尔滨师范大学, 2022)

[24] He Haibo, Bai Yang, Garcia E A, et al. ADASYN: Adaptive synthetic sampling approach for imbalanced learning[C]//Proc of 2008 IEEE Int Joint Conf on Neural Networks (IEEE World Congress on Computational Intelligence). Piscataway, NJ: IEEE, 2008: 1322−1328

[25] Ma Li, Fan Suohai. CURE-SMOTE algorithm and hybrid algorithm for feature selection and parameter optimization based on random forests[J]. BMC Bioinformatics, 2017, 18(1): 1−18

[26] Kim S, Zhang Hongyu, Wu Rongxin, et al. Dealing with noise in defect prediction[C]//Proc of 2011 33rd Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2011: 481−490

[27] Chen Liu, Fang Bin, Shang Zhaowei, et al. Tackling class overlap and imbalance problems in software defect prediction[J]. Software Quality Journal, 2018, 26(1): 97−125

[28] Tang Wei, Khoshgoftaar T M. Noise identification with the k-means algorithm[C]//Proc of 16th IEEE Int Conf on Tools with Artificial Intelligence. Piscataway, NJ: IEEE, 2004: 373−378

[29] Goyal S. Handling class-imbalance with KNN (neighbourhood) under-sampling for software defect prediction[J]. Artificial Intelligence Review, 2022, 55(3): 2023−2064

[30] Li Zhiqiang, Jing Xiaoyuan, Wu Fei, et al. Cost-sensitive transfer kernel canonical correlation analysis for heterogeneous defect prediction[J]. Automated Software Engineering, 2018, 25(2): 201−245

[31] Yang Zhenyu, Jin Chufeng, Zhang Yue, et al. Software defect prediction: An ensemble learning approach[J]. Journal of Physics:Conf Series, 2022, 2171(1): 012008

[32] Jiang Feng, Yu Xu, Gong Dunwei, et al. A random approximate reduct-based ensemble learning approach and its application in

software defect prediction[J]. Information Sciences, 2022, 609: 1147−1168

[33] Gong Lina, Jiang Shujuan, Bo Lili, et al. A novel class-imbalance learning approach for both within-project and cross-project defect prediction[J]. IEEE Transactions on Reliability, 2019, 69(1): 40−54

[34] Zhang Shenggang, Jiang Shujuan, Yan Yue. A software defect prediction approach based on BiGAN anomaly detection[J]. Scientific Programming, 2022, 2022(1): 1−13

[35] Rodriguez D, Herraiz I, Harrison R, et al. Preliminary comparison of techniques for dealing with imbalance in software defect prediction[C]//Proc of the 18th Int Conf on Evaluation and Assessment in Software Engineering. New York: ACM, 2014: 1−10

[36] Eivazpour Z, Keyvanpour M R. CSSG: A cost-sensitive stacked generalization approach for software defect prediction[J]. Software Testing, Verification and Reliability, 2021, 31(5): e1761

[37] Kohavi R, John G H. Wrappers for feature subset selection[J]. Artificial Intelligence, 1997, 97(1-2): 273−324

[38] He Xiaofei, Cai Deng, Niyogi P. Laplacian score for feature selection[C]//Proc of the 18th Int Conf on Neural Information Processing Systems. Cambridge, MA, USA: MIT Press, 2005

[39] Balogun A O, Basri S, Capretz L F, et al. Software defect prediction using wrapper feature selection based on dynamic re-ranking strategy[J]. Symmetry, 2021, 13(11): 2166−2189

[40] Thirumoorthy K. A feature selection model for software defect prediction using binary Rao optimization algorithm[J]. Applied Soft Computing, 2022, 131: 109737−109753

[41] Bahaweres R B, Suroso A I, Hutomo A W, et al. Tackling feature selection problems with genetic algorithms in software defect prediction for optimization[C]//Proc of 2020 Int Conf on Informatics, Multimedia, Cyber and Information System (ICIMCIS). Piscataway, NJ: IEEE, 2020: 64−69

[42] Miao Linsong, Liu Mingxia, Zhang Daoqiang. Cost-sensitive feature selection with application in software defect prediction[C]//Proc of the 21st Int Conf on Pattern Recognition (ICPR2012). Piscataway, NJ: IEEE, 2012: 967−970

[43] Liu Shulong, Chen Xiang, Liu Wangshu, et al. FECAR: A feature selection framework for software defect prediction[C]//Proc of 2014 IEEE 38th Annual Computer Software and Applications Conf. Piscataway, NJ: IEEE, 2014: 426−435

[44] Nam J, Pan S J, Kim S. Transfer defect learning[C]//Proc of 2013 35th Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2013: 382−391

[45] Ni Chao, Liu Wangshu, Chen Xiang, et al. A cluster based feature selection method for cross-project software defect prediction[J]. Journal of Computer Science and Technology, 2017, 32(6): 1090−1107

[46] Li Zhiqiang, Qi Chao, Zhang Li, et al. Discriminant subspace alignment for cross-project defect prediction[C]//Proc of 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). Piscataway, NJ: IEEE, 2019: 1728−1733

[47] Chen Jinfu, Wang Xiaoli, Cai Saihua, et al. A software defect prediction method with metric compensation based on feature selection and transfer learning[J]. Frontiers of Information Technology & Electronic Engineering, 2022, 23(5): 715−731

[48] Lu Huihua, Kocaguneli E, Cukic B. Defect prediction between software versions with active learning and dimensionality reduction[C]//Proc of 2014 IEEE 25th Int Symp on Software Reliability Engineering. Piscataway, NJ: IEEE, 2014: 312−322

[49] Xu Zhou, Liu Jin, Luo Xiapu, et al. Cross-version defect prediction via hybrid active learning with kernel principal component analysis[C]//Proc of 2018 IEEE 25th Int Conf on Software Analysis, Evolution and Reengineering (SANER). Piscataway, NJ: IEEE, 2018: 209−220

[50] Zhang Jie, Wu Jiajing, Chen C, et al. Cds: A cross−version software defect prediction model with data selection[J]. IEEE Access, 2020, 8: 110059−110072

[51] Marcus A, Maletic J I. Recovering documentation-to-source-code traceability links using latent semantic indexing[C]//Proc of 25th Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2003: 125−135

[52] Menzies T, Dekhtyar A, Distefano J, et al. Problems with precision: A response to “comments on ‘data mining static code attributes to learn defect predictors’”[J]. IEEE Transactions on Software Engineering, 2007, 33(9): 637−640

[53] Yao Jingxiu, Shepperd M. The impact of using biased performance metrics on software defect prediction research[J]. Information and Software Technology, 2021, 139(11): 1−14

[54] Qiao Hui. Research on software defect prediction techniques[D]. Zhengzhou: Information Engineering University, 2013 (in Chinese)
(乔辉. 软件缺陷预测技术研究[D]. 郑州：解放军信息工程大学, 2013)

[55] McCabe T J. A complexity measure[J]. IEEE Transactions on Software Engineering, 1976, 2(4): 308−320

[56] Chidamber S R, Kemerer C F. A metrics suite for object oriented design[J]. IEEE Transactions on Software Engineering, 1994, 20(6): 476−493

[57] Brito E A F, Carapuça R. Candidate metrics for object-oriented software within a taxonomy framework[J]. Journal of Systems and Software, 1994, 26(1): 87−96

[58] Bansiya J, Davis C G. A hierarchical model for object-oriented design quality assessment[J]. IEEE Transactions on Software Engineering, 2002, 28(1): 4−17

[59] Sotto-Mayor B, Kalech M. Cross-project smell-based defect prediction[J]. Soft Computing, 2021, 25(22): 14171−14181

[60] Khoshgoftaar T M, Szabo R M. Improving code churn predictions during the system test and maintenance phases[C]//Proc of 1994 Int Conf on Software Maintenance. Piscataway, NJ: IEEE, 1994: 58−67

[61] Nagappan N, Ball T. Use of relative code churn measures to predict system defect density[C]//Proc of the 27th Int Conf on Software Engineering. New York: ACM, 2005: 284−292

[62] Moser R, Pedrycz W, Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction[C]//Proc of the 30th Int Conf on Software Engineering.

New York: ACM, 2008: 181−190

[63] Knab P, Pinzger M, Bernstein A. Predicting defect densities in source code files with decision tree learners[C]//Proc of the 2006 Int Workshop on Mining Software Repositories. New York: ACM, 2006: 119−125

[64] Wang Dandan, Wang Qing. Improving the performance of defect prediction based on evolution data[J]. Journal of Software, 2016, 27(12): 3014−3029 (in Chinese)
(王丹丹, 王青. 基于演化数据的软件缺陷预测性能改进[J]. 软件学报, 2016, 27(12): 3014−3029)

[65] Liu Yibin, Li Yanhui, Guo Jianbo, et al. Connecting software metrics across versions to predict defects[C]//Proc of 2018 IEEE 25th Int Conf on Software Analysis, Evolution and Reengineering (SANER). Piscataway, NJ: IEEE, 2018: 232−243

[66] Mockus A, Weiss D M. Predicting risk of software changes[J]. Bell Labs Technical Journal, 2000, 5(2): 169−180

[67] Weyuker E J, Ostrand T J, Bell R M. Using developer information as a factor for fault prediction[C]//Proc of Third Int Workshop on Predictor Models in Software Engineering. Piscataway, NJ: IEEE, 2007: 8−15

[68] Ostrand T J, Weyuker E J, Bell R M. Programmer-based fault prediction[C]//Proc of the 6th Int Conf on Predictive Models in Software Engineering. New York: ACM, 2010: 1−10

[69] Pinzger M, Nagappan N, Murphy B. Can developer-module networks predict failures?[C]//Proc of the 16th ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2008: 2−12

[70] Nagappan N, Murphy B, Basili V. The influence of organizational structure on software quality[C]//Proc of 2008 ACM/IEEE 30th Int Conf on Software Engineering. New York: ACM, 2008: 521−530

[71] Mockus A. Organizational volatility and its effects on software defects[C]//Proc of the 18th ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2010: 117−126

[72] Zhou Yuming, Leung H. Empirical analysis of object-oriented design metrics for predicting high and low severity faults[J]. IEEE Transactions on Software Engineering, 2006, 32(10): 771−789

[73] Pai G J, Dugan J B. Empirical analysis of software fault content and fault proneness using Bayesian methods[J]. IEEE Transactions on Software Engineering, 2007, 33(10): 675−686

[74] Seliya N, Khoshgoftaar T M. Software quality analysis of unlabeled program modules with semisupervised clustering[J]. IEEE Transactions on Systems, Man, and Cybernetics-Part A:Systems and Humans, 2007, 37(2): 201−211

[75] Catal C, Sevim U, Diri B. Clustering and metrics thresholds based software fault prediction of unlabeled program modules[C]//Proc of 2009 6th Int Conf on Information Technology: New Generations. Piscataway, NJ: IEEE, 2009: 199−204

[76] Arisholm E, Briand L C, Johannessen E B. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models[J]. Journal of Systems and Software, 2010, 83(1): 2−17

[77] Gyimóthy T, Ferenc R, Siket I. Empirical validation of object-oriented metrics on open source software for fault prediction[J]. IEEE Transactions on Software Engineering, 2005, 31(10): 897−910

[78] Zheng Jun. Cost-sensitive boosting neural networks for software defect prediction[J]. Expert Systems with Applications, 2010, 37(6): 4537−4543

[79] Shukla S, Radhakrishnan T, Muthukumaran K, et al. Multi-objective cross-version defect prediction[J]. Soft Computing, 2018, 22(6): 1959−1980

[80] Zhao Liuchang, Shang Zhaowei, Zhao Ling, et al. Siamese dense neural network for software defect prediction with small data[J]. IEEE Access, 2018, 7: 7663−7677

[81] Zhang Xian, Ben K, Zeng Jie. Cross-entropy: A new metric for software defect prediction[C]//Proc of 2018 IEEE Int Conf on Software Quality, Reliability and Security (QRS). Piscataway, NJ: IEEE, 2018: 111−122

[82] Yang Xinli, Lo D, Xia Xin, et al. Deep learning for just-in-time defect prediction[C]//Proc of 2015 IEEE Int Conf on Software Quality, Reliability and Security. Piscataway, NJ: IEEE, 2015: 17−26

[83] Wang Song, Liu Taiyue, Tan Lin. Automatically learning semantic features for defect prediction[C]//Proc of 2016 IEEE/ACM 38th Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2016: 297−308

[84] Wang Song, Liu Taiyue, Nam J, et al. Deep semantic feature learning for software defect prediction[J]. IEEE Transactions on Software Engineering, 2018, 46(12): 1267−1293

[85] Li Jian, He Pinjia, Zhu Jieming, et al. Software defect prediction via convolutional neural network[C]//Proc of 2017 IEEE Int Conf on Software Quality, Reliability and Security (QRS). Piscataway, NJ: IEEE, 2017: 318−328

[86] Fan Guisheng, Diao Xuyang, Yu Huiqun, et al. Software defect prediction via attention-based recurrent neural network[J]. Scientific Programming, 2019, 2019(4): 1−14

[87] Qiu Shaojian, Lu Lu, Cai Ziyi, et al. Cross-project defect prediction via transferable deep learning-generated and handcrafted features[C]//Proc of the 31st Int Conf on Software Engineering and Knowledge Engineering. Skokie: Knowledge Systems Institute Graduate School, 2019: 431−552

[88] Liu Wangshu, Zhu Yongteng, Chen Xiang, et al. S^2 LMMD: Cross-project software defect prediction via statement semantic learning and maximum mean discrepancy[C]//Proc of 2021 28th Asia-Pacific Software Engineering Conf (APSEC). Piscataway, NJ: IEEE, 2021: 369−379

[89] Dam H K, Pham T, Ng S W, et al. A deep tree-based model for software defect prediction[J]. ArXiv Preprint ArXiv: 1802.00921, 2018

[90] Šikić L, Kurdija A S, Vladimir K, et al. Graph neural network for source code defect prediction[J]. IEEE Access, 2022, 10: 10402−10415

[91] Phan A V, Le Nguyen M, Bui L T. Convolutional neural networks over control flow graphs for software defect prediction[C]//Proc of 2017 IEEE 29th Int Conf on Tools with Artificial Intelligence (ICTAI). Piscataway, NJ: IEEE, 2017: 45−52

[92] Xu Jiaxi, Ai Jun, Liu Jingyu, et al. ACGDP: An augmented code graph-based system for software defect prediction[J]. IEEE

Transactions on Reliability, 2022, 71(2): 850−864

[93] Li Zhen, Zou Deqing, Xu Shouhuai, et al. VulDeePecker: A deep learning-based system for vulnerability detection[J]. ArXiv Preprint ArXiv: 1801.01681, 2018

[94] Huo Xuan, Yang Yang, Li Ming, et al. Learning semantic features for software defect prediction by code comments embedding[C]//Proc of 2018 IEEE Int Conf on Data Mining (ICDM). Piscataway, NJ: IEEE, 2018: 1049−1054

[95] Qu Yu, Liu Ting, Chi Jianlei, et al. Node2defect: Using network embedding to improve software defect prediction[C]//Proc of 2018 33rd IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2018: 844−849

[96] Zeng Cheng, Zhou Chunying, Lv Shengkai, et al. GCN2defect: Graph convolutional networks for smotetomek-based software defect prediction[C]//Proc of 2021 IEEE 32nd Int Symp on Software Reliability Engineering (ISSRE). Piscataway, NJ: IEEE, 2021: 69−79

[97] Zhou Chunying, He Peng, Zeng Cheng, et al. Software defect prediction with semantic and structural information of codes based on graph neural networks[J]. Information and Software Technology, 2022, 152: 107057

[98] Yang Fengyu, Huang Yaxuan, Xu Haoming, et al. Fine-grained software defect prediction based on the method-call sequence[J]. Computational Intelligence and Neuroscience, 2022, 2022(8): 1−15

[99] Uddin M N, Li Bixin, Ali Z, et al. Software defect prediction employing BiLSTM and BERT-based semantic feature[J]. Soft Computing, 2022, 26(16): 7877−7891

[100] Shin Y, Williams L. An empirical model to predict security vulnerabilities using code complexity metrics[C]//Proc of the Second ACM-IEEE Int Symp on Empirical Software Engineering and Measurement. New York: ACM, 2008: 315−317

[101] Gegick M, Williams L, Osborne J, et al. Prioritizing software security fortification through code-level security metrics[C]//Proc of Workshop on Quality of Protection. New York: ACM, 2008: 31−38

[102] Meneely A, Williams L. Secure open source collaboration: An empirical study of Linus' law[C]//Proc of the 16th ACM Conf on Computer and Communications Security. New York: ACM, 2009: 453−462

[103] Shin Y, Meneely A, Williams L, et al. Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities[J]. IEEE Transactions on Software Engineering, 2010, 37(6): 772−787

[104] Chowdhury I, Zulkernine M. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities[J]. Journal of Systems Architecture, 2011, 57(3): 294−313

[105] Hovsepyan A, Scandariato R, Joosen W, et al. Software vulnerability prediction using text analysis techniques[C]//Proc of the 4th Int Workshop on Security Measurements and Metrics. New York: ACM, 2012: 7−10

[106] Scandariato R, Walden J, Hovsepyan A, et al. Predicting vulnerable software components via text mining[J]. IEEE Transactions on Software Engineering, 2014, 40(10): 993−1006

[107] Yamaguchi F, Lottmann M, Rieck K. Generalized vulnerability extrapolation using abstract syntax trees[C]//Proc of the 28th Annual Computer Security Applications Conf. New York: ACM, 2012: 359−368

[108] Meng Qingkun, Wen Shameng, Feng Chao, et al. Predicting buffer overflow using semi-supervised learning[C]//Proc of 2016 9th Int Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). Piscataway, NJ: IEEE, 2016: 1959−1963

[109] Pang Yulei, Xue Xiaozhen, Wang Huaying. Predicting vulnerable software components through deep neural network[C]//Proc of the 2017 Int Conf on Deep Learning Technologies. New York: ACM, 2017: 6−10

[110] Dam H K, Tran T, Pham T, et al. Automatic feature learning for predicting vulnerable software components[J]. IEEE Transactions on Software Engineering, 2018, 47(1): 67−85

[111] Kalouptsoglou I, Siavvas M, Kehagias D, et al. An empirical evaluation of the usefulness of word embedding techniques indeep learning-based vulnerability prediction[C]//Proc of Int ISCIS Security Workshop. Berlin: Springer, 2022: 23−37

[112] Ma Qianhua. Deep learning-based software vulnerability prediction[D]. Beijing: Beijing University of Posts and Telecommunications, 2020 (in Chinese)
(马倩华. 基于深度学习的软件源码漏洞预测[D]. 北京:北京邮电大学, 2020)

[113] Zhou Yaqin, Liu Shangqing, Siow J, et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks[C]//Proc of 33rd Conf on Neural Information Processing Systems (NeurIPS). San Diego, CA, USA: NIPS, 2019: 10197−10207

[114] Li Zhen, Zou Deqing, Xu Shouhuai, et al. SySeVR: A framework for using deep learning to detect software vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4): 2244−2258

[115] Li Yi, Wang Shouhua, Nguyen T N. Vulnerability detection with fine-grained interpretations[C]//Proc of the 29th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2021: 292−303

[116] Fu M, Tantithamthavorn C. LineVul: A transformer-based line-level vulnerability prediction[C]//Proc of 2022 IEEE/ACM 19th Int Conf on Mining Software Repositories (MSR). Piscataway, NJ: IEEE, 2022: 608−620

[117] Shin Y, Williams L. Is complexity really the enemy of software security?[C]//Proc of the 4th ACM Workshop on Quality of Protection. New York: ACM, 2008: 47−50

[118] Viega J, McGraw G R. Building Secure Software: How to Avoid Security Problems the Right Way, Portable Documents[M]. London: Pearson Education, 2001

[119] Gao Zhiwei, Yao Yao, Rao Fei, et al. Prediction model of vulnerabilities based on the type of vulnerability severity[J]. Acta Electronica Sinica, 2013, 41(9): 1784−1787 (in Chinese)
(高志伟, 姚尧, 饶飞, 等. 基于漏洞严重程度分类的漏洞预测模型[J]. 电子学报, 2013, 41(9): 1784−1787)

[120] Pan Zhixin, Mishra P. A survey on hardware vulnerability analysis using machine learning[J]. IEEE Access, 2022, 10: 49508−49527

[121] Palix N, Thomas G, Saha S, et al. Faults in Linux: Ten years later[C]//Proc of the 16th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2011: 305−318

[122] Zimmermann T, Nagappan N, Williams L. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista[C]//Proc of 2010 3rd Int Conf on Software Testing, Verification and Validation. Piscataway, NJ: IEEE, 2010: 421−428

[123] Shin Y, Williams L A. Can fault prediction models and metrics be used for vulnerability prediction?[R]. North Carolina, USA: North Carolina State University, Department of Computer Science, 2010

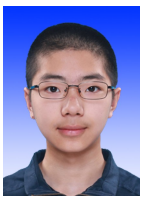
[124] Shin Y, Williams L. Can traditional fault prediction models be used for vulnerability prediction?[J]. Empirical Software Engineering, 2013, 18(1): 25−59

Tian Xiao, born in 1999. Master candidate. Her main research interest includes network and information security.

田　笑, 1999 年生. 硕士研究生. 主要研究方向为网络与信息安全.

Chang Jiyou, born in 1999. Master candidate. His main research interest includes network and information security.

常继友, 1999 年生. 硕士研究生. 主要研究方向为网络与信息安全.



Zhang Chi, born in 2002. Master candidate. His main research interest includes AI and security.

张　弛, 2002 年生. 硕士研究生. 主要研究方向为人工智能与安全.

Rong Jingfeng, born in 1986. PhD candidate. His main research interest includes network and information security.

荣景峰, 1986 年生. 博士研究生. 主要研究方向为网络与信息安全.

Wang Ziyu, born in 1998. Master candidate. His main research interest includes network and information security.

王子昱, 1998 年生. 硕士研究生. 主要研究方向网络与信息安全.

Zhang Guanghua, born in 1979. PhD, professor, master supervisor. His main research interest includes network and information security.

张光华, 1979 年生. 博士，教授，硕士生导师. 主要研究方向为网络与信息安全.

Wang He, born in 1987. PhD, lecturer, master supervisor. Her main research interests include cryptography, quantum cryptographic protocol.

王　鹤, 1987 年生. 博士，讲师，硕士生导师. 主要研究方向为密码学、量子密码协议.

Wu Gaofei, born in 1987. PhD, lecturer, master supervisor. His main research interest includes cryptography.

伍高飞, 1987 年生. 博士，讲师，硕士生导师. 主要研究方向为密码学.

Hu Jinglu, born in 1962. PhD, professor, PhD supervisor. His main research interest includes computational intelligence.

胡敬炉, 1962 年生. 博士，教授，博士生导师. 主要研究方向为计算智能.

Zhang Yuqing, born in 1966. PhD, professor, PhD supervisor. His main research interest includes information security.

张玉清, 1966 年生. 博士，教授，博士生导师. 主要研究方向为信息安全.