印地语

安全和通信网络卷2020，文章ID 8858010，16页https://doi.org/10.1155/2020/8858010 评论文章

*A软件漏洞检测、程序*

**修复和缺陷预测技术**

**沈志东**

# 还有斯珍教育部航天信息安全与可信计算重点实验室，

*武汉大学网络科学与工程学院，湖北武汉430079*

*通信地址应为沈志东；shenzd@whu.edu.cn*

2020年6月19日收到；2020年8月13日修订；2020年9月11日接受；2020年9月30日出版

学术编辑：路易吉·科波里诺

版权所有©2020沈志东和陈思。这是一篇根据知识共享署名许可证发布的开放获取文章，允许在任何媒体中不受限制地使用、分发和复制，前提是原创作品被正确引用。

开源软件以其开放性和灵活性在各个行业得到了广泛的应用，但同时也带来了潜在的软件安全问题。随着软件数量的大规模增加和复杂性的增加，处理这些安全问题的传统手动方法效率低下，无法满足当前网络空间的安全要求。因此，开发更多的智能技术来解决软件中潜在的安全问题是软件安全领域研究人员的一个重要研究课题。深度学习技术的发展为软件潜在安全问题的研究带来了新的机遇，研究者们相继提出了许多自动化方法。本文从软件漏洞检测、软件程序修复和软件缺陷预测三个方面对这些自动化技术进行了详细的评价和分析。同时，指出了这些研究方法存在的一些问题，给出了相应的解决方案，最后展望了深度学习技术在自动化软件漏洞检测、自动化程序设计中的应用前景

修复和自动缺陷预测。

介绍

# 1.  随着信息技术的飞速发展，软件在世界经济、军事、社会等各个领域发挥着重要作用。与此同时，软件中潜在的安全问题正在成为一个新出现的全球性挑战。软件漏洞是安全问题的根本原因之一。高技能的黑客可以根据自己的意愿利用软件漏洞做很多有害的事情，例如窃取用户的私人信息和停止关键设备。

在当今的多用户连续交互环境中，如果黑客使用某个时间节点发起攻击，其损失和成本难以预测。根据通用漏洞和暴露（CVE）组织[1]发布的统计数据，2000年发现的软件漏洞数量不到4600个，而目前发现的漏洞数量接近20000个。从图1可以看出，漏洞数量在过去三年达到了顶峰，这无疑增加了许多使用网络服务的计算机用户面临的威胁。因此，及时发现和修复软件问题对于安全研究人员来说至关重要。

图1：历年CVE漏洞数量。

|  |
| --- |
| 软件潜在的安全问题不仅是软件开发者面临的安全问题，而且关系到国家网络安全的发展。近几年来，工商业网络攻击现象不断增多，企业和个人面临的安全威胁也有所增加。2017年出现的勒索软件“WannaCry”是近年来影响最严重、覆盖面最广、后果最严重的网络攻击。因此，当面临网络安全问题带来的严峻挑战时，对自动化、可扩展、机器速度的漏洞检测、程序修补和缺陷预测技术的需求变得越来越迫切。 |

2016年，DARPA举办的CGC对我们产生了深远的影响。从CGC会议来看，机器确实可以取代人类白帽的一部分工作，甚至在某些方面，比如操作速度，机器比人类具有天然优势。特别是近年来，深度学习技术的快速发展为机器智能化解决软件安全问题提供了更多的机会。

在本文中，我们回顾了深度学习技术在软件安全研究中的应用，讨论了学术界和相关研究人员在软件安全研究中的努力，并展望了深度学习技术在软件安全领域面临的机遇和挑战。

总结本文的工作，主要贡献有三个方面：

首先，我们回顾了深度学习技术在软件漏洞检测、程序修补和缺陷预测方面的最新研究进展。表1和表2概述了这些技术。

其次，从软件漏洞检测、程序修补、缺陷预测等方面分析了各种技术的优缺点，并提出了今后研究的思路和解决方案。

最后，我们展望了现有的自动漏洞检测、自动程序修补和自动缺陷预测技术所面临的机遇和挑战，并为未来的研究人员提供一些参考价值。

自动软件漏洞检测

# 2.  传统的漏洞研究方法往往要求安全研究人员具备专业知识和丰富的实践经验，通用性较差，效率有限。目前，深度学习和自然语言处理技术的应用可以智能地处理漏洞信息，辅助安全漏洞研究，提高安全漏洞挖掘的效率。二进制漏洞检测方法检测精度高，实用性广，但难以跟踪程序上层代码的结构信息和类型信息。目前，二进制代码漏洞检测采用反汇编操作将二进制代码反汇编成汇编指令代码，利用程序分析技术提取关键的漏洞代码信息，并将其矢量化后输入神经网络进行训练。根据静态分析技术，将漏洞处理方法进一步分为基于代码相似度的漏洞检测方法和基于代码模式的漏洞检测方法。

2.1. 基于代码相似性的漏洞检测。

*代码相似性漏洞检测又称克隆检测，其核心思想是相似的程序代码具有相似的软件漏洞。在代码克隆检测技术不断发展的过程中，研究人员定义了四种常见的克隆类型：1型（T1）、2型（T2）、3型（T3）和4型（T4）[2]。Type-1（T1）克隆是相同的代码片段，可能包含空格、布局或注释的变化。Type-2（T2）克隆是允许标识符、文本、类型、空白、布局和注释变化的代码片段。与2型克隆相比，3型（T3）克隆允许进行额外的修改，例如更改、添加或删除语句。类型-4（T4）克隆还包括语义相同但语法不同的代码片段。*2.2. 基于语法的克隆检测方法。

*在克隆检测过程中，源代码的表示决定了信息提取的上限，进而限制了模型设计和相似性度量算法的选择，最终影响检测效果。传统的基于文本的表示方法，过程数据只涉及删除程序代码中的注释和空格，主要是基于文本相似度度量的方法来检测文本级克隆。然而，基于树的表示方法通过解释器分析程序代码来处理数据，其度量算法考虑了更多的程序结构信息，从语法层面检测代码克隆。*目前，许多学者都从代码的维度描述了代码克隆检测技术在软件安全领域的发展

表1：关于自动漏洞检测、自动程序修补和自动缺陷预测的深度学习技术总结。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 方法类型   |  |  |  |  | | --- | --- | --- | --- | |  | 有利条件 | 缺点 | 自动的 | | 漏洞检测  基于代码相似性的漏洞检测 | 基于代码模式的漏洞检测  基于源代码检测多个克隆类型；基于二进制编码的检测方法具有较高的检测精度 | 静态方法实现了更高的编码；动态覆盖方法检测速度更快  假阴性率高（来源）；分析复杂性（二进制） | 缺少运行时信息（静态）；低代码覆盖率（动态）  自动程序修补 | | 基于语法的程序修补 | 基于语义的程序修补  基于令牌的方法误差分析；简单的基于文本的方法生成更高质量的补丁静态方法实现了良好的修复效果动态方法并准确捕获 | 程序行为  可解释性差（令牌）；不可接受的程序行为（tex） | 有限类型的维修（静态）；高成本（动态）  自动缺陷预测 | | 项目内缺陷预测 | 跨项目缺陷预测  实时缺陷预测  端到端实施，准确预测有缺陷的程序模块 | 有效地集成数据集资源，以更好地促进新的项目开发实践  早期识别故障模块和细粒度分析，有效识别  缺陷  扩展性差 | 代码特征提取粒度过大  缺乏大量的训练数据来训练模型  表2：深度学习技术在软件漏洞检测、程序修复和缺陷预测方面选择的各种特征参数。 |   方法类型   |  |  |  |  | | --- | --- | --- | --- | | 美国有线电视新闻网 | RNN网络 | DNN网络 | 代码相似性vul检测 | | 迭代、退出、隐藏层、梯度率 | 隐藏层、网络深度、退出、批量大小、迭代 | 隐藏层、迭代、退出、层大小、学习率 | 代码模式vul检测 | | 过滤器大小、过滤器数量、隐藏层 | 辍学、批量大小、迭代、学习率、向量维数 | — | 语法程序修补 | | — | 学习率、批量大小、隐藏层、标记长度、隐藏单元、迭代、嵌入大小、梯度优化器 | — | 语义程序修补 | | — | 隐藏单元，梯度优化器， | 向量维数、学习率、迭代、退出、隐藏维数  — | 项目内缺陷预测 | | 过滤器数量、过滤器大小、隐藏节点、， | 批量大小、迭代、嵌入维度  辍学、向量维度、隐藏层、隐藏节点、批量大小、， | 迭代，学习率  隐藏层，隐藏节点，迭代，学习率，梯度优化器，批量大小 | 跨项目缺陷预测 | | 过滤器大小、过滤器数量、学习率、向量大小 | 辍学、隐藏\u层、隐藏\u节点、批次\u大小 | 迭代，隐藏层，隐藏节点 | 跨项目缺陷预测 | | — | — | 隐藏层、隐藏节点、激活功能 | 表示、模型设计、相似性度量算法和性能度量。然而，这些工作主要总结了早期的代码克隆检测方法，很少涉及深度学习技术在克隆检测技术中的应用。本文主要介绍了近年来深度学习技术在代码克隆检测技术中的研究进展，并从语法层面和语义层面对相关技术进行了阐述。 | |

语法层的代码表示主要考虑程序源代码的语法规则。通常，构造一个抽象语法树（AST）来分层显示程序的结构信息，然后进行进一步的检测处理。目前，基于深度学习的方法通过构造抽象语法树截取关键节点信息，将其转换为特征向量或哈希值，然后选择合适的神经网络模型进行训练，最后利用该模型实现克隆比较。

表3显示了语法克隆检测中深度学习的技术实现和技术特征。目前主流的方法是基于AST对程序代码进行特征化。Marastoni等人基于图像实现了T4型代码克隆检测。

表3：基于语法的克隆检测的技术特征。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 系统/写入程序   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 数据预处理 | 代码表示 | 网络 | 克隆类型 | 分类对象 | CCL学习者[3] | | 使用ANTLR和ASRParser解析每个方法 | 标记AST | DNN | T1–T3 | 方法对 | CLDH[4] | | 将每个代码片段解析为AST | AST | LSTM | T1–T4 | 代码片段 | 怀特等人[5] | | 使用ANTLR标记代码 | 橄榄树 | RtvNN | T1–T4 | 方法/文件级 | Marastoni等人[6] | | 利用tigress C模糊数据集 | 二值图像 | 有线电视新闻网 | T4 | 图像级 | 文献[3]提出了一种基于句子的深度学习代码克隆检测方法，定义了八种标记类型。首先，使用ANTLR[7]解析每个方法并提取C1–C3，使用EclipseAstParser[8]为每个方法创建一个抽象语法树以进一步提取C4–C8标记。然后，计算每种方法中每种类型的标记的频率和方法对之间的相似性，将其输入分类器进行训练，以检测给定代码库的克隆。为了评估CCLearner对DNN中的参数设置和所选特征的敏感性，我们对DNN的不同配置进行了实验，并调查了不同的特征集。据观察，CCLearner在DNN中使用2个隐藏层和300次训练迭代时效果最好。由于标记只揭示语法层面的代码关系，因此这种方法测量代码功能之间的相似性的能力是有限的。 | |

CcCurnter只使用源代码的词汇信息，不考虑代码结构信息，这使得很难检测功能克隆对。文献[4]提出了一种称为CDLH的端到端深度学习框架，该框架以有监督的方式捕获关于代码片段功能行为的词汇和语法信息，并通过哈希进一步将实值表示转换为二进制哈希代码。与CCLearner相比，学习的特征更加紧凑，显著提高了检测效率，节省了存储空间。同时，研究了不同长度的散列码对用F1值测量的CDLH克隆检测性能的影响，在8～48的不同码长范围内，CDLH的F1值对该范围内的散列码长度不敏感。

文献[5]综合考虑了同时用于建模的代码结构和标识符。与CCLearner基于令牌频率统计的工作不同，本文使用贪婪的方法在多个树结构之间进行转换来描述代码，并使用RtNN模型将代码片段中的相似项映射到相似的连续值向量。此外，该模型基于RvNN模型学习不同粒度级别的代码片段。在实验集中，ANTLR用于标记源代码，RNNLM工具包[9]用于为每个系统训练多个RTNN，改变隐藏层的大小和深度。然后，使用困惑[10]作为评估模型质量的代理，模型质量越高，准确性越好。

文献[6]提出了一种新的程序相似性检测方法，该方法将二进制代码转换为图像数据，输入训练模型的CNN网络。同时，作者使用Tigress C模糊算子[11]迭代变换文法对原始数据集进行扩展，以充分训练模型。由于CNN网络需要查看从二进制文件中提取的整个图像来完成每个文件的分类，因此二进制可执行文件的大小限制了系统的性能。

在实验阶段，CNN中的所有权重均采用标准偏差设置为0.1的正态分布随机值进行初始化，以避免在早期阶段陷入局部极小值，并进一步微调以下参数以提高模型性能：测试比率、梯度比率、限值、随机种子、，和规范。

为了证明相似性任务检测的各种代码表征方法组合的有效性，文献[12]实现了四种代码表示：标识符、抽象语法树、控制流图和字节码，并为每种表示训练了一个模型。实验结果表明，通过集成不同的模型，可以进一步提高代码克隆检测的准确性。

与文本和标记方法相比，基于语法的代码表示方法考虑了程序代码的结构信息，对于程序语句的序列转换和少量修改更具容错性。然而，基于树的方法仍然存在一些缺点。随着程序逻辑结构的增强，构造程序抽象语法树的复杂性增加，这进一步导致提取关键代码和向量变换操作的计算开销增加。

2.3. 基于语义的克隆检测方法。

*语义层的代码表示不仅考虑了源代码的语法，还利用了程序代码的控制流和数据流信息，解决了代码功能层的代码克隆检测问题。由程序生成的数据流图、控制流图和程序依赖图是当前主流的表示方法。*表4显示了语义克隆检测中深度学习的技术实现和技术特征。程序代码的特征是控制流和数据流的程序依赖关系图。众所周知的检测技术有DeepSim，它使用CFG和DFG节点信息；ZEEK使用代码符号化和哈希技术；CCDLC和SeNAMAMER等人使用PDG在更深的层次上考虑程序语义信息。

表4：基于语义的克隆检测的技术特征。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 系统/写入程序   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 预处理 | 代表 | 网络 | 克隆类型 | 分类对象 | CCDLC[13] | | Trimme，规范化方法块 | BDG-PDG-AST | 有线电视新闻网 | T3-T4 | 方法块 | Sheneamer等人[14] | | Trimme、nirmalize方法块 | AST-PDG-BDG | — | T1–T4 | 方法块 | 泽克[15] | | 将过程拆分为基本块 | 散列 | NN | T4 | 代码块 | DeepSim[16] | | 使用WALA分析字节码 | DFG CFG | DNN′ | T4 | 方法级 |  | |

该程序具有定义良好的语法，并且可以很好地表示抽象语法树。AST在语法级别的克隆检测中显示了良好的结果，但它不能有效地检测不连续的代码克隆。文献[13]实现了一种新的聚类功能，它将从PDG中提取的高级特征与从BDG中提取的低级特征有效地结合起来，紧密地反映了程序代码中数据之间的关系。在模型训练方面，采用丢包正则化和线性校正单元实现了CNN网络，并采用多线程小批量下降算法运行，实验结果表明CCDLC在代码混淆和语义克隆检测方面取得了良好的效果。与[14,17]相比，CCDLC系统进一步改进了数据预处理和特征向量转换过程。

为了更好地执行单词向量的嵌入学习，文献[15]引入了一种新的向量表示方法proc2vec。首先，将程序汇编代码拆分为基本块，并使用组成代码部分的链作为特征集，将链转换为数字，并将这些数字组合成表示相应代码的向量。与直接符号数字转换相比，数字向量矩阵更为稀疏，可以大大降低相似度的计算量。神经网络模型是一种常见的四层结构，它使用交叉熵代价函数、0.1的退出正则化、32的批量以及3次数据传递进行训练。

文献[16]基于控制流和数据流分析了各种变量和基本代码块之间的关系，并考虑了编码控制流和数据流信息的三种特征（变量特征、基本块特征以及变量和基本块之间的关系特征）。然后，通过预定义的编码规则生成高维稀疏二进制特征向量语义矩阵。语义特征矩阵将寻找同构子图的问题简化为检测相似模式的问题，并便于后续处理。模型训练的特征参数从层大小、历元、学习率、辍学率和L2正则化中考虑，其适当值通过参考经典模型中的参数设置来设置。与现有最先进的技术相比，DeepSim在召回率、准确率和时间性能方面都有显著的优势。

与一般的自然语言文本特征相比，基于语义的方法充分考虑了程序结构、序列和特殊的语法信息，深化了语义特征。然而，基于语义的表示技术需要使用生成器为每种编程语言生成程序依赖图，可扩展性差，很少涉及神经网络模型的选择和优化。

在现有的代码相似性检测方法中，通常采用四层网络结构模型。RNN和CNN在数据关系处理和特征提取方面有各自的优势，可以考虑它们的优势来设计一种新的网络结构。对于特征参数的选择，我们可以借鉴k-fold交叉验证的训练方法。

# 3.  基于代码模式的漏洞检测

基于代码模式的漏洞检测技术主要包括两个阶段。在训练阶段，利用控制流和数据流技术提取程序中的关键代码，并在当前主流工具（如word2vec）的帮助下将其转换为向量，然后将其输入适当的神经网络进行监督训练。在检测阶段，对新软件程序执行相同的数据处理，用于通过学习的模型检测现有漏洞。根据程序是否需要运行，基于代码模式的漏洞检测方法分为静态检测方法和动态检测方法，目前用于模型训练的网络结构包括CNN、RNN和LSTM[18–20]。

*3.1. 静态检测方法。*静态分析是指在不运行软件的情况下，通过构造抽象语法树和程序依赖关系图进行程序分析的过程，其分析对象通常是源代码或可执行代码。与可执行代码相比，源代码分析可以获得更多的语义信息，综合考虑执行路径上的信息，从而发现更多的漏洞，提高命中率。

|  |
| --- |
| 图2：代码静态分析和神经网络训练的原理。 |

图2显示了代码静态分析和神经网络训练的原理。整个过程包括以下步骤：样本代码构建、特征提取、词向量生成、神经网络模型训练和分类。其中，漏洞特征提取主要涉及如何选择合适的粒度来表示软件程序和漏洞检测。由于深度学习或神经网络以向量作为输入，我们需要将程序表示为对漏洞检测具有语义意义的向量。我们应该使用一些中间表示作为程序和向量表示之间的“桥梁”，向量表示是深入学习的实际输入。漏洞特征提取是通过CFG和PDG技术将程序转换为某种中间表示形式，能够保留（部分）程序元素之间的语义关系（例如，数据依赖和控制依赖）。词向量生成基于特征提取，应用最主流的词向量生成技术，将中间表示转换为向量表示，即神经网络的实际输入。神经网络训练分类包括训练和检测两个阶段。训练阶段以从历史代码库中提取的代码向量表示作为输入，其输出为模型参数微调的神经网络。在检测阶段，将从新软件程序中提取的码向量表示作为输入，输出为分类结果。

表5显示了静态代码漏洞检测中深度学习的技术实现和技术特征。VulDeePecker首次展示了深度学习技术在漏洞检测中的潜力；SySeVR和CPGVA从PDG开始，更好地利用了程序控制流程图；Lee等人从程序的二进制级开始研究汇编指令级代码检测问题。

传统的漏洞检测方法要求人类专家手动定义漏洞的特征，这既繁重又繁琐。此外，每个人对脆弱性特征的感知和经验水平都不同，因此很难达到预期的效果。

为了挖掘深度学习在漏洞检测中的潜力，文献[21]提出了VulDeePecker，这是一个基于深度学习的漏洞检测系统，它提出了代码小工具的概念，它是一些中间表示，可以保留（部分）程序元素之间的语义关系（例如，数据依赖性和控制依赖性）然后，中间表示可以转换为向量表示，这是神经网络的实际输入。考虑到程序代码中变量和语句之间的关系，作者使用BLSTM网络结构。这是因为程序函数调用的参数可能会受到早期统计的影响在模型训练期间，本文采用10倍交叉验证来训练BLSTM神经网络，并改变每个BLSTM神经网络的隐藏层数量，以观察对结果F1测度的影响。

VulDeePecker使用数据流分析生成代码小工具，未能实现控制流分析过程，检测到的漏洞类型有限。此外，选择用于模型训练的特征参数主要根据NLP任务中的经验以及如何选择更合适的特征参数来选择ers需要进一步研究。

针对VulDeePecker现有的缺点，文献[22]进一步考虑了程序控制流程，提出了基于语法、语义和向量的漏洞检测系统框架。与VulDeePecker提取代码小工具类似，数据流分析用于从程序中提取和生成SYVC，然后用于通过PDG生成程序片段并转换为SeVCs。学习BLSTM的主要参数是辍学率为0.2，批量大小为16，纪元数为20，输出维数为256，默认学习率为0.002，隐藏向量维数为500，隐藏层数为2。与VulDeePecker相比，SySeVR提出了自己的字向量生成算法，避免了t建立了word2vec模型，优化了神经网络模型特征参数的选择。

如前所述，通过考虑程序结构、数据流和控制流依赖关系，很难完全发现多个漏洞。文献[23]提出了一种新的源代码表示方法——代码属性图，它是源代码的一种新表示方法，将经典程序分析的概念，即抽象语法树、控制/数据流图和程序依赖图，合并为一个联合数据结构XTRAP考虑程序的各种流程图所可能的程序点序列，避免了传统PDG中不能考虑的程序依赖性。文中没有详细讨论网络结构特征参数对模型检测性能的影响，主要比较了状态-根据CNN（AlexNet、Lenet、Tcnn等）或RNN（LSTM、BLSTM、GRU等），与深入学习SARD[25]数据集的结果相关的art源代码评审方法。

为了更好地评价深度特征学习漏洞检测技术的性能，文献[26]对三种漏洞检测方法进行了比较，在软件漏洞检测的跨项目和类不平衡问题上的实验结果并不好，但为以后的研究提供了良好的研究思路。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 表5：静态代码漏洞检测中深度学习的技术特征。   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 系统/写入程序 | 分析对象 | 漏洞类型 | 矢量技术 | 网络 | 数据集 | | VulDeePeckke r[21] | 代码小工具 | 库/API函数 | word2vec | BLSTM | NVD SARD | | SySeVR[22] | 代码小工具 | 126种脆弱性 | 自定义算法 | 6深度神经网络 | NVD SARD | | CPGVA[23] | 码流 | CWE-78  CWE-90 | word2vec | 美国有线电视新闻网  （变量净额） | 特区 | | Lee等人[24] | 作用 | CWE-121 | 说明2VEC | 文本CNN | 朱丽叶 | |

此外，文献[24]在二进制程序漏洞检测的研究过程中，提出了指令2VEC模型对汇编代码进行矢量化。与基于word2vec模型的字向量生成方法相比，指令2VEC过程利用汇编代码中所有字的word2vec结果。同时，大多数指令使用一个字向量操作码和两个操作数，它们也有固定的长度。NLP使用高维向量来表示大量的字向量，但汇编代码不需要高维向量，因为字的数量很小，指令2VEC更好地保留了操作码、操作数和寄存器之间的潜在关系研究不同卷积核大小和隐藏层数量对模型性能的影响；文本CNN使用9种类型的过滤器，每种数量为128，以最大限度地提高性能。实验结果表明，在一定范围内增加CNN网络中的过滤器数量可以显著提高检测精度.

基于深度学习的静态漏洞检测方法从源代码层面分析程序代码的依赖性，并基于程序切片的思想对易受攻击的代码进行预处理。在现有研究中，每个样本代码的标记都需要手动进行，且模型仅适用于特定语言和类型的漏洞同时，本文现有的方法对不同特征参数的选择对神经网络模型的影响没有太多的解释。

*3.2.动态检测方法。*动态分析是通过运行特定程序并获取程序的输出或内部状态等信息（分析对象为可执行代码）来验证或发现软件漏洞的过程。与静态方法相比，动态方法分析漏洞以获得特定的操作这样，所分析的漏洞通常更准确，误报率更低。

静态分析工具已被证明在特定应用领域特别有效，如嵌入式系统或航空航天应用。然而，在更常见的软件上使用这些技术要困难得多。文献[27]提出了一种基于深度学习的动态分析软件漏洞检测方法，通过二进制执行、事件钩住、事件收集三个步骤以及VDiscover工具确定最终执行状态[28]。然后，使用zzuf工具实现数据标记。最后，将9872函数调用序列作为一种特征进行分析，以表示二进制程序在执行期间的模式。受数据量和数据预处理序列长度的限制，在实际检测中可能会遗漏某些特定类型的漏洞。

文献[29]从生成和选择智能种子开始，以减少对无用路径的探索。本文提出的解决方案是NeuFuzz，它使用深度神经网络从大量易受攻击和干净的程序执行路径中学习隐藏的漏洞模式。在在线引导模糊化过程中，使用预测模型el判断一条看不见的路径是否易受攻击，并根据预测结果对种子进行标记，然后添加到种子队列中，最后在下一个种子选择过程和种子变异过程中对易受攻击的种子进行优先级排序并分配更多的变异能量。

深度学习技术可以减少人的特征工程，有望在未来的研究过程中取代传统的漏洞检测方法，显著提高漏洞检测性能。然而，深度学习在漏洞检测方法中面临许多不足，其中模型往往是训练基在程序源代码上，这在大多数情况下是不可用的。此外，检测粒度和有限的模型进一步限制了检测效果。

为了进一步推动漏洞自动检测技术的发展，在未来的研究过程中，可以考虑对语法和语法相似的编程语言进行抽象建模，以实现同一模型的多语言之间的软件漏洞检测n可以考虑实施小块数据的学习过程，以解决缺乏大规模脆弱性数据集的问题。此外，神经网络结构的优化和特征参数的选择一直是相关从业者面临的问题[30]现有的代码漏洞检测技术通常基于10倍交叉验证训练方法来获得最佳参数，这种方法通常会影响大规模项目数据的训练时间和模型性能。

# 4.  软件程序自动修复

早期的自动修补技术主要用于防止蠕虫的传播，随着技术的发展，自动修补技术慢慢渗透到计算机软件安全的各个方面，自动程序修复技术可以帮助自动修复软件程序中的一些缺陷，从而起到保护软件安全的作用有效减少软件开发人员的程序调试时间。

*4.1.修补过程。*自动补丁技术分为以下三个阶段：软件故障定位阶段、补丁生成阶段和补丁评估阶段。

软件故障定位是程序自动修复的前提，主要用于识别程序中潜在缺陷或漏洞的位置。目前，常用的故障定位技术分为两类：静态故障定位技术和动态故障定位技术t定位技术[31–33]主要通过程序分析技术获取被测程序代码之间的控制依赖和数据依赖关系，从而确定和定位故障位置。动态故障定位技术[34]通过执行预选的测试用例获得程序的执行信息，并通过分析程序的执行流来定位缺陷语句在被测试程序中的位置。

构建补丁阶段通过检查程序代码结构并分析示例代码提交和修改信息，然后通过定义的操作修改缺陷声明，定义了一些操作规则。通常使用两种基于搜索和语义的补丁生成方法方法是在搜索空间中查找并验证补丁的过程，通常通过调整搜索空间大小和优化搜索策略来提高补丁生成效率。基于语义的补丁生成方法通过集成故障定位、约束求解和程序合成等技术来生成补丁。

补丁评估阶段评估生成的候选补丁，直到找到一个补丁，使程序能够执行所有测试用例。

# 5.  基于语法的修补技术

基于语法错误的程序修复技术通过学习代码语法特征来修改错误代码，以达到特定的语言规范。

*5.1.令牌级学习和修复技术。*标准LR解析器通常难以解决语法错误及其精确位置，甚至可能对实际错误的位置得出误导性结论。计算机程序的源代码是纯文本表示，类似于语法结构的自然语言，其中自然语言的发展技术和深度学习技术使基于学习的修复成为可能。

学习相似语法结构信息的神经网络可以增强程序修复的效果，文献[35]提出了一种提供语法错误反馈的技术，该技术使用递归神经网络（RNN）模拟语法有效的标记序列。对于给定的程序，首先基于RNN网络对一组语法正确的代码提交进行建模，使用前缀令牌序列查询RNN网络，以预测语法错误代码提交的有效令牌序列对。

在训练过程中，对具有1或2个隐藏层且每个具有128或256个隐藏单元的RNN和LSTM网络以及若干其他指标（如学习率、批量大小和梯度阈值）的实验进行了微调。本文重点分析添加更多隐藏单元的隐藏层实际上会降低数据集上网络的性能。目前的方法只使用前缀令牌序列来修复，并且不考虑令牌序列上下文之间的关系，这不能保证语义上的修复序列的正确性，同时修复多个错误。

文献[36]基于LSTM网络训练了两种相反的语言模型，该模型综合考虑了令牌上下文内容，以解决单个令牌语法错误。通过改变隐层神经元的数量来训练神经网络，同时保持每层神经元的数量不变，同时调整其他特征参数。在通过大量实例训练形成标记词汇的前提下，该模型可以保证定位误差和纠错位的准确性，但这反过来又限制了该模型的时间效率。此工具的目标受众是经验丰富的程序员，新手开发人员很难使用此工具。因此，数据主体、时间维度和可伸缩性问题限制了GrammarGuru的性能。

GrammarGuru方案仅限于错误代码和在同一领域的训练数据，其中文献[37]克服了这一问题，并使用n-gram模型和LSTM模型来纠正语法错误。对于hyperparameters的影响，本文测试了985种不同的配置，改变了隐藏层大小（50、100、200、300、400和1000）、上下文大小（5、10、15和20）以及优化器（RMSprop和Adam），Adam优化器的性能优于所有其他参数的影响。通过对Blackbox语料库的评估，实验结果表明，该语言模型能够在不进行语法分析的情况下成功地定位和修复手工编写的代码中的语法错误。

*5.2. 基于文本的学习和修复技术。*由于程序各个部分之间的依赖关系，即使是单个错误也可能需要对整个程序进行分析。修复精度是程序修复技术中的一个重要指标，以前的文献[35–37]基于令牌级精度定位和修复程序错误，其生成的错误补丁无法完全修复编程错误。

为了提高错误修复的效果，文献[38]提出了一种称为DeepFix的端到端解决方案，该解决方案综合考虑了令牌上下文文本，使用注意的多层序列神经网络捕获程序文本依赖性。本文使用Tensorflow中实现的基于注意的序列到序列结构，并进行5倍交叉验证，以准确评估我们的技术并选择最佳特征参数。迭代修复的有效性表明，如果网络无法在迭代中为程序生成修复，并且后续迭代不会针对该程序执行，因为再次应用相同的网络不会改变结果。

DeepFix以迭代的方式调用模型输出，逐个修复程序中的多个错误，但大多数修复只涉及减少编译错误，无法从根本上解决问题。文献[39]提出了DeepRepair学习框架，该框架使用程序中的冗余或重复代码，并基于代码相似性原则对代码库中的语句进行优先级排序和转换，以生成程序修复组件。本文选择word2vec来训练单词嵌入，用于初始化递归自动编码器的嵌入。Deep Repair的搜索策略使用基于嵌入的成分转换算法，与jGenProg相比，该算法在更短的时间内生成补丁，并找到更高质量的补丁[40]。

文献[41]提出了一种用于修复错误的系统跟踪器，它通过错误定位、抽象代码修复和具体的三模块代码修复来提高修复精度。作者进行了广泛的网格搜索以设置超参数，其最佳配置是根据顶部精度和顶部平滑精度的性能度量进行交叉验证后选择的。同时，TRACER采用的模块化方法侧重于一小部分句子，这大大提高了RNN框架提出相关局部更正的能力。

在现有的在线编程课程中，TRACER生成的程序修复比编译器内置的纠错功能更有教学价值。然而，TRACER使用基于编译器的启发式方法，存在错误定位不准确的问题。同时，示踪剂的输入数据不能考虑需要全局上下文信息的错误类型，这使得模型难以处理诸如打开/关闭缺失括号之类的类型错误。此外，TRACER和DeepFix使用监督学习技术来训练纠错模型。

文献[42]提出了一个强化学习框架RLAssist，它克服了基于编译器的跟踪器启发式。该模型在训练过程中不需要任何监督，其代理从原始程序学习生成候选修复。该模型是在A3C的开源实现2之后实现的，A3C通过实验为我们的任务提供了一个合适的特征参数配置。实验结果表明，事件数越大，错误消息的解决越多。同时，其他参数对误差修复度的比例也有不同程度的影响。

基于语法的程序修复技术仍然面临一些问题，一些方法仍然借用编译器来定位错误位置并验证错误修复。编译器通常无法提供准确的错误位置信息，这使得新手程序员很难根据编译器提供的信息执行准确的修复。此外，很难通过编译器来修复错误，从而改变程序的行为和结果，从而确保程序语句的语义有效性。

RNN网络在处理序列数据关系和生成文本数据方面具有更大的优势，由于程序修复过程涉及数据生成过程，因此经常选择其不同的网络结构模型来实现程序修复。在模型性能改进方面，通常通过优化以下特征参数来实现：学习率、批量大小、隐藏层、标记长度、嵌入大小梯度优化器、隐藏单元和迭代。在未来的修复技术中，研究者应该考虑程序文本中的上述信息，并开发更好的模型来定位和修复语法错误。

# 6.  基于语义的补丁技术

基于语义错误的程序修复技术通常指修改程序代码，使实际程序行为与程序员预期的行为基本一致。为了从语义层面修复程序中的错误，文献[31]提出了一种新的神经网络架构，该架构将生成和应用修复建议的非统计过程与评分和排序修复的统计过程分离。首先使用基于规则的处理器生成修复候选，然后使用新的神经网络结构对统计模型进行评分，以便不同的修复候选可以在概率空间中竞争，以更高的精度修复错误。在实验阶段，调整少量超参数，如训练时期、隐藏维度、嵌入大小和退出，以训练模型，这大大优于注意序列对序列模型。

文献[31]通过执行枚举搜索操作预测所有程序位置的修复，然后计算修复分数以选择最佳修复，这会在修复过程中导致训练/测试数据不匹配问题。在此基础上，文献[32]使用多头指针联合执行分类、定位和修复，可以实现特定变量的细粒度定位并完成修复过程。本文定义了两个参数来过滤预测，其中没有讨论特征参数对神经网络性能的影响。这一结果表明，该网络能够联合、高效和有效地执行定位和修复任务，而无需显式枚举。

为了克服语义补丁技术产生的补丁集合的语义不可理解性，文献[33]基于迭代三重聚类策略挖掘语义相关修复模式，使用抽象语法树、编辑操作树和代码上下文树进行模式表示，充分利用AST diff树中变更信息和令牌变更之间的相似性。对开源项目中收集的数千个软件补丁的评估表明，FixMiner生成的模式与相关补丁解决的错误的语义相关，足以生成正确率高的补丁，并且可以使用挖掘模型正确修复。

基于语法和语义的程序表示差异如下所示。

首先，在程序运行期间，程序中的语句通常不会按照向深度学习模型呈现相应标记序列的顺序进行学习。其次，程序中的控制依赖项和数据依赖项在程序语义中起着重要作用，但这些依赖项在标记和AST中并没有得到很好的表示。第三，从程序性能的角度来看，即使是相似的语法结构也可能导致程序语义的巨大差异。

文献研究[31–33]分析了程序的源代码，只能捕获序列数据之间的依赖关系，无法处理跳转或递归等关系。实时变量值的连续元组的程序状态可以准确地捕获这样的程序语义，同时也为神经网络模型提供自然的结果。文献[34]提出了一种新的语义程序嵌入系统，它将变量跟踪嵌入、状态跟踪嵌入和混合嵌入相结合，从程序执行轨迹中学习语义信息。每个跟踪模型中的所有编码器都有两个堆叠的GRU层，每个层中有200个隐藏单元，除了状态跟踪模型中的状态编码器有一个100个隐藏单元的单层。在预测学生常见错误模式方面，比较动态程序嵌入和基于语法的程序嵌入，在执行跟踪上训练的嵌入明显优于在程序语法上训练的嵌入。

目前，基于语义的程序修复技术主要通过构建程序AST来学习程序的语义特征，只能分析序列数据之间的依赖关系。与基于文法的程序修复技术一样，现有技术也基于RNN及其变体网络实现，因为程序修复过程涉及数据生成过程。然而，现有技术很少讨论特征参数对神经网络模型训练的影响，主要集中在算法优化上，这在一定程度上限制了现有技术的性能。

在未来的程序修复技术中，我们可以借鉴结合AST和PDG技术的漏洞检测技术，提升现有程序语义修复技术的性能，分析特征参数对模型性能的影响，进一步推动深度学习技术在程序修复技术中的应用。

# 7.  软件缺陷自动预测

软件缺陷是指产品描述、设计和编码阶段的任何缺陷。软件缺陷预测可以帮助软件开发人员快速发现错误，合理分配有限的资源，并确定测试工作的优先级。近年来，软件开发人员使用各种深度学习算法来分析软件缺陷并增加软件测试，以提高软件质量、降低软件成本和增强软件可维护性。

本节根据数据源和预测粒度的不同，介绍了近年来深度学习技术在软件缺陷预测中的最新研究成果，其内容主要包括项目内缺陷预测、跨项目缺陷预测和实时缺陷预测。

# 8.  项目内缺陷预测

项目内缺陷预测可进一步分为版本内缺陷预测和跨版本缺陷预测，前者使用软件系统的特定版本数据训练模型进行缺陷预测，后者使用软件系统的不同版本数据训练模型进行缺陷预测。

表6显示了项目缺陷预测中深度学习的技术实现和技术特征。Wang等人使用CLNI完成数据标记任务，树LSTM模型以无监督的方式进行训练；CAP-CNN综合考虑代码评审信息，实现代码缺陷预测，其他方法从网络模型入手，提高预测性能。

为了提高软件的质量，开发人员在测试和调试过程中投入了大量精力。然而，在大多数情况下，开发人员的资源和时间有限。在这种情况下，自动化软件缺陷预测技术可以更好地帮助他们发现错误并确定测试的优先级。

文献[43]采用编辑距离相似度计算算法[49]和CLNI[50]来消除具有潜在错误标签的数据，以消除数据噪声的影响，并使用AsAnalysis程序源代码来获取语法信息，将其转换为特征向量，输入DBN网络，建立效果预测模型。

在训练阶段，对三个参数进行微调，以训练有效的DBN来学习语义特征。网络结构越深，节点越多，这将增加模型的收敛时间，性能的提高将受到限制。此外，本文所提出的语义特征生成方法仅在计算机上进行了评估

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 表6：缺陷内预测方法的技术特征。   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 系统/写入程序 | 数据集集合 | 韵律学 | 特征生成 | 标记的数据 | | 王等人[43] | 许诺 | F1  回忆起 | 解析源代码，处理噪声，映射标记，通过DBN生成特征 | 克莱尼 | | Dam等人[44] | 三星 | F1  回忆起 | 解析源代码，映射AST节点，通过树生成特征-  LSTM | 模型  一代 | | DP-CNN[45] | 许诺 | F1  回忆起 | 解析源代码，提取和编码令牌，通过  有线电视新闻网 | 提供的存储库 | | SDNN[46] | 国家航空和航天局 | *F*1 AUC | 删除重复的实体，替换缺少的值，数据规范化 | 提供的存储库 | | CAP-CNN[47] | 许诺 | *F*1 | 分割源模块，通过预训练word2vec编码为矢量，通过CNN生成特征 | 提供的存储库 | | 英语学习者[48] | 12个开源项目 | F1  回忆起 | 删除注释，使用单词嵌入方法，生成特征  通过LSTM提供的项目 | | |

开源java项目，其在封闭源代码软件和用其他语言编写的项目上的性能未知。此外，所提出的语义特征生成方法仅在开源java项目上进行评估，其在封闭源代码软件和其他语言编写的项目上的性能未知。

最先进的方法[43]利用深度信念网络学习从程序AST中提取的标记向量的语义特征，这在缺陷预测方面优于传统的基于特征的方法。然而，它忽略了有关程序的结构信息，从而导致更准确的缺陷预测。文献[44]使用一种新的注意机制来实现基于树的LSTM网络，该网络利用树网络结构的强大预测能力来避免噪声数据的影响，并以无监督的方式进行训练。在模型训练中，为了防止神经网络中的过度拟合，考虑了退出，并将其与复杂度评价指标相结合，以选择最佳模型。三星开源项目已经证明了这种方法的有效性。

正如语音识别[51]和图像分类[52]中的深度学习研究人员所报告的那样，卷积神经网络（CNN）比DBN更先进，因为前者可以更有效地捕获局部模式。文献[45]借用了[43]的方法，选择AST上三种相同类型的节点作为标记，并将单词嵌入作为CNN体系结构的一部分。特征参数的微调是训练成功CNN的关键，它直接影响模型的收敛性。本文主要改变这三个参数的值：过滤器数量、过滤器长度和隐藏层中的节点数量，其最佳值通过验证实验获得。在对七个开源项目的评估中，实验结果表明，DP-CNN平均将现有技术方法改进了12%。

文献[46]探讨了暹罗网络的优势，提出了一种新的SDP模型，该模型将相似性特征学习和距离度量学习集成到一个统一的方法中。与之前的方法[43、49、50]相比，SDNN使用两个完全相同的连接网络来学习最高级别的相似性特征，并使用计量函数作为最高级别特征之间的距离度量。在实验过程中，作者探索了隐藏层数量对模型性能的影响，并从3个隐藏层中得到了最佳结果，这是通过实验得到的最佳模型。此外，还考虑了单元、小批量、学习率和辍学等因素，进一步提高了模型的精度。

以前的技术通常基于程序代码本身，而不考虑程序代码注释等信息。文献[47]提出了一种新的缺陷预测模型CAP-CNN，它是一种深度学习模型，自动将代码注释嵌入到源代码的语义特征中，用于软件缺陷预测。CAP-CNN结合源代码和注释进行软件缺陷预测，从而生成更具语义的特征表示，指示源模块的结构和功能。在本实验中，作者参考了传统的CNN网络结构模型和参数设置，没有进一步探讨各种特征参数对模型性能的影响。对广泛使用的软件缺陷预测数据集的实验结果表明，代码注释有助于提高缺陷预测性能。

从代码自然性的角度来看，文献[48]将交叉熵作为一种新的软件度量引入到文件级的典型缺陷预测中，它基于LSTM单元的RNN结构挖掘代码令牌集来捕获程序中的共模。同时，针对过拟合、梯度消失和梯度爆炸问题，本文采用了退出、梯度裁剪和自适应学习率等优化策略。在一系列对比实验中，对各种性能评价指标的微调进一步证实了各种特征参数在模型性能中的重要作用。实验结果表明，交叉熵度量比传统度量具有更好的区分性。

项目内缺陷预测技术主要关注文件级缺陷，它构建AST解析程序源代码，使用单词嵌入技术生成单词向量，并使用公共网络模型进行训练。对于项目内缺陷深度学习技术的每个特征参数，辍学是可用于防止模型过度拟合的关键参数，其他几个参数（如学习速率和隐藏层数）通常用于加速模型的收敛。

此外，项目内缺陷预测方法主要关注新文件是否包含缺陷信息，无法有效反馈引入缺陷的原因或定位缺陷的位置。在实际的软件工程实践中，很难确保在开源项目中训练的模型的可重复性和有效性。

# 9.  跨项目缺陷预测

受不同项目之间的开发过程和编程语言等因素的影响，以前的研究侧重于项目内部的缺陷预测。在实践中，新项目通常没有足够的缺陷数据来建立预测模型，因此跨项目缺陷预测是必要的，它使用来自其他项目的数据来训练模型以应用于新项目。

随着人工智能技术的飞速发展，特别是深度学习技术强大的学习和表示能力，为不同项目之间的缺陷预测研究提供了良好的应用基础。传统的研究侧重于人工设计编码程序的特征，简单地探索不同机器学习算法的性能，无法捕捉程序的语义差异，建立的预测模型精度较低。

为了有效弥补程序语义和缺陷预测特征之间的差距，文献[43]使用DBN自动学习从程序AST中提取的标记向量的特征，然后使用这些特征来训练缺陷预测模型。在实验阶段，本文探讨了隐藏层数量、每个隐藏层中的节点数量和训练迭代次数三个参数对模型精度、F1和召回率的影响。实验结果表明，与传统特征相比，语义特征显著改善了CPDP模式，适当的特征参数可以更好地提高模型的性能。

为了弥合程序语义和缺陷特征之间的差距，文献[44]开发了一种新的预测模型，该模型通过构建树形LSTM神经网络来捕获程序中相距很远的上下文信息，以匹配使用程序源代码的抽象语法树表示。在培训阶段，本文重点关注辍学对模型性能的影响。文献〔53, 54〕提出将深度神经网络应用于汇编指令序列而不是AST，它使用C++编译器将C++程序编译成汇编代码，然后应用卷积神经网络学习汇编指令序列的数据集，其预测效果明显优于基于AST的方法。

在以往的预测模型中，只关注程序代码数据，很少关注程序代码注释信息。文献[47]提出了一种新的缺陷预测模型CAP-CNN，它可以自动嵌入代码注释，生成软件缺陷预测源代码的语义特征。同时，该模型利用网络编码和注释信息的吸收，在训练过程中自动生成语义特征，有效地克服了程序中注释缺失的问题。对几个广泛使用的软件数据集的实验结果表明，注释特征可以提高缺陷预测性能。

现有的软件预测模型大多局限于源代码，但在预测工作中获得的文件通常是二进制可执行文件。由于软件版权和源代码保护的限制，缺陷预测研究界和第三方安全公司很难获得源代码。文献[55]提出了smali2vec方法来捕获APK中smali的特征，并使用深层神经网络进行训练。该模型侧重于微调三个参数，包括隐藏层的数量、每个隐藏层中的神经元数量和迭代次数，从而训练有效的DNN来预测APK中的缺陷。通过在我们的训练数据上使用这些参数的不同值进行实验来微调这三个参数，并从AUC值和错误率中找到参数的最佳配置。与项目内缺陷预测技术研究相比，基于深度学习技术的跨项目缺陷预测研究还太少，跨项目缺陷预测的研究过程有待于在今后的研究中进一步推进。

# 10. 实时缺陷预测

在实际场景中，我们很难应用软件系统给出的易出错性建议来克服这些问题，开发人员引入justin时间缺陷预测技术来解决这些问题。在即时缺陷预测中，可以在早期阶段识别容易发生故障的模块，并反馈给开发人员进行更改和修复。目前，即时缺陷预测技术是在代码更改级别进行的，它的细粒度分析使开发人员能够更有效地解决软件开发过程中遇到的问题。

文献[56]首先结合深度学习方法来提高即时缺陷预测的性能，该方法使用由三台受限Boltzmann机器和一个logistic回归分类器组成的深度信念网络来构建更深层的模型，以检测更具表现力的特征。在本文中，模型的神经网络结构是一种通用结构，其隐藏单元的数量基于一系列数字，使用类似于贪婪搜索的策略。同时，通过十倍交叉验证对模型的性能进行了评估，其成本效益和F1得分远优于以前的方法。

与文献[56]不同，文献[57]使用三个完全连接的反向传播（BP）神经网络来构建回归模型，而不是分类器模型。此外，模型神经网络回归（NNR）方法利用代码更改的十个数值度量，然后将它们反馈给神经网络，其输出指示测试中的代码更改包含bug的可能性。本文对模型特征参数值的选择没有太多的说明，主要介绍了如何基于交叉验证训练方法选择最佳模型。在给定检测资源的情况下，基于努力感知的即时缺陷预测可以更有效地识别缺陷数量。

文献[58]提出了一种新方法TLEL，它使用决策树和集成学习来提高即时缺陷预测的性能。在内层，将决策树和包相结合，构建随机森林模型。在外层，使用随机欠采样来训练许多不同的随机森林模型，并使用堆栈重新组合这些模型。本文介绍了在六个数据集上改变NTree和NLearner这两个参数的值对TLEL性能的影响，并基于成本效益和F1分数这两个评估指标，使用十倍交叉验证来评估TLEL的性能。

即时缺陷预测技术可以识别早期容易出现故障的模块，并将其反馈给开发人员进行更改和修复，其细粒度分析使开发人员能够更有效地解决软件开发过程中遇到的问题。缺陷预测问题是确定当前软件程序是否包含有缺陷的代码，在一定程度上可以看作是两类问题。在nlp和图像领域，各种CNN网络模型在分类任务中表现良好，在未来的研究中，应在实时缺陷预测应用中加以推广。

# 11. 未来的方向和挑战

在软件开发到部署的整个生命周期内，确保软件的可靠性至关重要，深度学习技术的应用加快了软件开发周期，并节省了人力和时间成本。然而，随着软件技术的不断发展，现有技术难以对日益增多的安全问题进行统一的分析和处理，这进一步使得深度学习技术在软件安全领域的应用面临一些挑战。如表7所示，将深度学习技术应用于软件安全领域仍然存在一些挑战。

*11.1. 特征生成。*选择的特征在神经网络训练过程中起着重要作用，其质量越高，模型训练效果越好。在漏洞检测中，通常使用基于标记、树和图的方法来提取程序源代码，然后进行特征映射和向量转换

表7：应用于软件安全研究的深度学习的机遇和挑战。

|  |  |
| --- | --- |
| 挑战 | 机会 |
| 工具审查 | 深度学习模型自动反馈 |
| 特征提取 | 树图结合模型 |
| 语义特征学习 | 细粒度程序特征表示 |
| 高假阴性和假阳性 | DL与静态、动态程序分析技术相结合 |
| 数据集 | 建立一个开源的统一数据集标准库 |
| 跨项目漏洞检测 | 迁移学习 |
| 代码度量 | 新代码属性 |

这些技术用于生成单词向量，作为深度学习模型的输入。目前主流的方法是对源代码进行分析，在特征选择中无法有效跟踪程序行为，这在一定程度上限制了模型的性能。分析二进制代码可以更好地理解程序的行为，从而使构建的模型更好地定位漏洞的位置。因此，有必要探索面向二进制的深度学习漏洞检测技术。

*11.2. 型号选择。*各种深度学习模型最初主要应用于计算机视觉和图像处理领域，在软件安全研究中的应用尚处于起步阶段，如何利用深度学习模型强大的学习能力来处理软件安全问题亟待解决。不同的深度学习模型对相同的数据具有不同的学习能力，如何选择合适的模型来学习特征数据需要进一步的探索。

*11.      3.数据集。*在当前的研究中，模型训练需要大量的数据，这些数据来自开源项目。开放源代码项目和封闭源代码项目之间的数据存在差异，这使得在开放源代码项目中培训的模型可能不适用于封闭源代码项目。此外，用于模型训练的数据常常存在数据类型不平衡的问题，这进一步限制了许多模型的性能。

*11.4. 业绩评价。*传统的漏洞检测需要人类专家手动定义特征，这既繁琐又耗时，并且经常面临高误报率和高误报率的风险。深度学习技术可以很好地处理自然语言任务，程序代码在一定程度上可以视为文本数据，我们可以借鉴文本数据的处理方法和评价指标，提高现有方法在自然语言特定任务中的性能。

*11.5.特征参数。*特征参数值的选择在各种深度学习技术中起着重要的作用，其参数值的设置不合理将极大地影响模型的性能。现有技术的许多特征参数值是根据经验选择的，其合理性尚未得到验证。对于不同的任务，为对应于相同网络模型的特征参数设置相同的值可能会产生不同的效果。

# 12. 结论

在软件系统开发过程中，提高软件质量和安全性是软件开发人员的共同目标。传统的检测方法需要领域专家花费大量的时间和精力创建特征工程，将深度学习技术与程序分析技术相结合，辅助软件安全研究，进一步推动自动检测技术的发展至关重要。本文详细介绍了深度学习技术在软件漏洞检测、软件程序修复、软件缺陷预测等方面的最新研究进展，并对其中存在的不足进行了阐述和讨论。纵观近年来自动化技术在软件安全领域的发展趋势，深度学习技术在未来软件安全自动化技术的研究中将发挥越来越重要的作用。对于大多数研究人员来说，深度学习技术是一个历史性的机遇，将促进软件工程技术研究的创新。

**利益冲突**

作者声明他们没有利益冲突。

# 致谢

这项工作得到了中国国家重点研发计划（授予2018YFC1604000）和湖北自然科学基金（批准号2017CFB663）的支持。

# 工具书类

[1] CVE，http://cve.mitre.org/. [2] S.Bellon，R.Koschke，G.Antoniol等人，“克隆检测工具的比较和评估”，IEEE软件工程学报，第33卷，第9期，第577-5912007页。

[3] 李磊、冯浩、庄文伟、孟新民和赖德，

“CClearner：基于深度学习的克隆检测方法”，《2017年IEEE软件维护和进化国际会议记录》（ICSME），第249-260页，IEEE，中国上海，2017年9月。

[4] H.Wei和M.Li，“利用源代码中的词法和句法信息监督软件功能性克隆检测的深层特征”，《第二十六届人工智能国际联合会议论文集》，第3034-3040页，澳大利亚墨尔本，2017年8月。

[5] M.White，M.Tufano，C.Vendome等人，“用于代码克隆检测的深入学习代码片段”，第31届IEEE/ACM自动化软件工程国际会议论文集，ACM，第87-98页，新加坡，2016年9月。

[6] N.Marastoni，R.Giacobazzi和M.Dalla Preda，“程序相似性的深度学习方法”，载于《第一届Symbiosis机器学习和软件工程国际研讨会论文集》，法国蒙彼利埃ACM，第26-35页，2018年9月。

[7] ANTLR，http://www.antlr.org/. [8] 使用JDT ASTParser解析单个Java FLE，http://www. 克里克。com/2011/11/使用jdt astparser解析Java文件/。

[9] T.Mikolov，S.Kombrink，A.Deoras，L.Burget和J.Cernocky，“RNNLM递归神经网络语言建模工具包”，ASRU&apos;11会议录，美国HI，Waikoloa，2011年12月。

[10] D.Jurafsky和J.Martin，《语音和语言处理》，皮尔森，伦敦，英国，第二版，2009年。

[11] ChristianCollberg，三峡河流漫流器/模糊器，2015年。

[12] M.Tufano，C.Watson，G.Bavota等人，“从源代码的不同表示中深入学习相似性”，摘自2018年IEEE/ACM第15届采矿软件存储库国际会议记录，IEEE，瑞典哥德堡，第542-553页，2018年5月。

[13] A.Sheneamer，“将聚类与语义克隆的深度学习分类相结合的CCDLC检测框架”，《2018年第17届IEEE机器学习与应用国际会议论文集》，IEEE，第701-706页，美国佛罗里达州奥兰多，2018年12月。

[14] A.Sheneamer，H.Hazazi，S.Roy和J.Kalita，“使用机器学习标记语义代码克隆的方案”，《2017年第16届IEEE机器学习和应用国际会议论文集》，IEEE，第981-985页，墨西哥坎昆，2017年12月。

[15] N.Shalev和N.Partush，“使用机器学习的二进制相似性检测”，《第十三届编程语言和安全分析研讨会论文集》，加拿大多伦多ACM，第42-47页，2018年10月。

[16] G.Zhao和J.Huang，“Deepsim：深度学习代码功能相似性”，摘自2018年第26届ACM欧洲软件工程会议联合会议记录和软件工程基础研讨会，美国佛罗里达州布埃纳维斯塔湖ACM，第141-151页，2018年11月。

[17] A.Sheneamer，S.Roy和J.Kalita，“语义代码克隆和模糊代码的检测框架”，专家系统与应用，第97卷，第405-420页，2018年。

[18] N.Kalchbrenner、E.Grefenstette和P.Blunsom，“用于句子建模的卷积神经网络”，2014年，https://arxiv.org/abs/1404.2188. [19] T.Mikolov，M.Karafiat，L.Burget等人，“基于递归神经网络的语言模型”，国际语音通信协会第十一届年会论文集，日本Makuhari，2010年9月。

[20] H.Sak，A.W.Senior和F.Beaufays，“用于大规模声学建模的长-短期记忆递归神经网络架构”，2014年。

[21]李志强，邹D.S.徐等，“VulDeePecker：基于深度学习的漏洞检测系统”，2018年，https://arxiv. org/abs/1801.01681。

[22]Z.Li，D.Zou，S.Xu等人，“SySeVR：利用深度学习检测软件漏洞的框架”，2018年，https://arxiv。org/abs/1807.06756。

[23]W.Xiaomang，Z.Tao，W.Runpu，X.Wei和H.Changyu，“CPGVA：基于代码属性图的深度学习脆弱性分析”，发表于2018年第十届先进信息通信技术国际会议论文集，IEEE，第184-188页，瑞典斯德哥尔摩，2018年8月。

[24]Y.J.Lee，S.H.Choi，C.Kim等人，“通过深入学习学习二进制代码来检测软件缺陷”，摘自《KSII第九届互联网国际会议（ICONI）2017年研讨会论文集》，美国宾夕法尼亚州匹兹堡，2017年7月。

[25] https://samate.nist.gov/SRD/. [26]X.Ban，S.Liu，C.Chen和C.Chua，“软件漏洞检测深入学习功能的性能评估”，《并发性和计算：实践与经验》，第31卷，第19期，第E51032019页。

[27]Wu，J.Wang，J.Liu和W.Wang，“深度学习的漏洞检测”，2017年第三届IEEE计算机与通信国际会议（ICCC）论文集，IEEE，第1298-1302页，中国成都，12月

[28]G.Grieco，G.L.Grinblat，L.Uzal等人，“利用机器学习进行大规模漏洞发现”，《第六届ACM数据和应用程序安全与隐私会议记录》，第85-96页，美国洛杉矶新奥尔良，2016年3月。

2017.

[29]王彦，吴子忠，魏魁等，“神经模糊：深度神经网络的有效模糊化”，IEEE Access，第7卷，第36340-363522019页。

[30]张若荣、李文华和汤明明，“深度学习回顾”，信息与控制，第47卷，第4期，第385-397页，2018年。

[31]J.Devlin，J.Uesato，R.Singh等，“使用神经符号转换网络修复语义代码”，2017年，https://arxiv。org/abs/1710.11054。

[32]M.Vasic，A.Kanade，P.Manatis等人，“通过联合学习定位和修复来修复神经程序”，2019年，https://arxiv。org/abs/1904.01720。

[33]A.Koyuncu，K.Liu，T.F.Bissyande等人，“Fixminer:挖掘自动程序修复的相关修复模式”，2018年，https://arxiv.org/abs/1810.01791. [34]K.Wang，R.Singh和Z.Su，“用于程序修复的动态神经程序嵌入”，2017年，https://arxiv.org/abs/ 1711.07163.

[35]S.Bhatia和R.Singh，“使用递归神经网络自动纠正编程分配中的语法错误”，2016年，https://arxiv.org/abs/1603.06129. [36]E.A.Santos，J.C.Campbell，A.Hindle和J.N.Amaral，“使用递归神经网络发现和纠正语法错误”，PeerJ，第5卷，文章ID e3123v1，2017年。

[37]E.A.Santos，J.C.Campbell，D.Patel，A.Hindle和J.N.Amaral，“语法和敏感性：使用语言模型检测和纠正语法错误”，载于2018年IEEE第25届软件分析、进化和再工程国际会议论文集（SANER），IEEE，第311-322页，意大利坎波巴索，2018年3月。

〔38〕R. Gupta，S.Pal，A. Kanade等人，“Deepfix：通过深入学习固定共同的C语言错误，”在第30届AAAI关于人工智能的会议上，旧金山，CA，美国，2017年2月。

[39]M.White、M.Tufano、M.Martınez、M.Monperrus和D.Poshyvanyk，“通过深度学习代码相似性对程序修复成分进行分类和转换”，摘自2019年IEEE第26届软件分析、进化和再工程国际会议记录（SANER），IEEE，第479-490页，中国杭州，2019年2月。

[40]M.Martinez，T.Durieux，R.Sommerard，J.Xuan和M.Monperrus，“java中真实缺陷的自动修复：缺陷4J数据集的大规模实验”，经验软件工程，第22卷，第4期，第1936-1964页，2017年。

[41]U.Z.Ahmed，P.Kumar，A.Karkare等人，“编译错误修复：针对学生程序，来自学生程序”，摘自2018年IEEE/ACM第40届软件工程国际会议记录：软件工程教育和培训（ICSE-SEET），IEEE，瑞典哥德堡，第78-87页，2018年5月。

[42]R.Gupta、A.Kanade和S.Shevade，“编程语言修正的深度强化学习”，2018年，https://arxiv。org/abs/1801.10467。

[43]S.Wang，T.Liu和L.Tan，“自动学习缺陷预测的语义特征”，2016年IEEE/ACM第38届软件工程国际会议（ICSE）论文集，IEEE，德克萨斯州奥斯汀，美国，第297-308页，5月

[44]H.K.Dam，T.Pham，S.W.Ng等人，“基于深层树的软件缺陷预测模型”，2018年，https://arxiv.org/abs/ 1802.00921.

[45]Li，P.He，J.Zhu等人，“通过卷积神经网络进行软件缺陷预测”，《2017年IEEE软件质量、可靠性和安全国际会议论文集》，IEEE，第318-328页，捷克共和国布拉格，2017年7月。

[46]L.Zhao，Z.Shang，L.Zhao等人，“用于小数据软件缺陷预测的暹罗密集神经网络”，IEEE Access，第7卷，第7663-76772018页。

2016.

[47]霍X，杨Y，李M.和詹D.-C.“通过代码注释嵌入学习软件缺陷预测的语义特征”，《2018年IEEE数据挖掘国际会议论文集》，IEEE，第1049-1054页，新加坡，2018年11月。

[48]X.Zhang，K.Ben和J.Zeng，“交叉熵：软件缺陷预测的新度量”，载于2018年IEEE软件质量、可靠性和安全（QRS）国际会议记录，IEEE，第111-122页，葡萄牙里斯本，2018年7月。

[49]G.Navarro，“近似字符串匹配的导游”，ACM计算调查，第33卷，第1期，第31-88页，2001年。

[50]S.Kim，H.Zhang，R.Wu和L.Gong，“缺陷预测中的噪声处理”，载于第33届国际软件工程会议论文集ICSE&apos;11，第481-490页，美国纽约，2011年。

[51]O.Abdel Hamid，A.-r.Mohamed，H.Jiang和G.Penn，“将卷积神经网络概念应用于语音识别的混合NN-HMM模型”，2012年IEEE声学、语音和信号处理国际会议记录，日本京都，2012年3月。

[52]A.Krizhevsky，I.Sutskever和G.E.Hinton，“具有深度卷积神经网络的图像网络分类”，神经信息处理系统进展论文集，内华达州太浩湖，美国，2012年12月。

[53]A.V.Phan和M.Le Nguyen，“预测软件缺陷的汇编代码上的卷积神经网络”，《2017年第21届亚太智能和进化系统研讨会论文集》，IEEE，第37-42页，越南河内，2017年11月。

[54]A.V.Phan，M.Le Nguyen和L.T.Bui，“用于软件缺陷预测的控制流图上的卷积神经网络”，《2017年IEEE第29届人工智能工具国际会议记录》，IEEE，第45-52页，美国马萨诸塞州波士顿，2017年11月。

[55]Dong，J.Wang，Q.Li，G.Xu和S.Zhang，“使用深度神经网络的android二进制可执行文件中的缺陷预测”，无线个人通信，第102卷，第3期，第2261-2285页，2018年。

[56]Yang，D.Lo，Xa.Xia，Y.Zhang和J.Sun，“实时缺陷预测的深度学习”，2015年IEEE软件质量、可靠性和安全国际会议论文集，IEEE，第17-26页，加拿大温哥华，2015年8月。

[57]L.Qiao和Y.Wang，“基于神经网络的努力感知和及时缺陷预测”，PLoS One，第14卷，第2期，文章ID e0211359，2019年。

[58]杨晓阳，罗德明，夏晓霞，孙杰，“TLEL：一种用于实时缺陷预测的两层集成学习方法”，信息与软件技术，第87卷，第206-220页，2017年。