面向代码克隆检测的多维源代码表征学习方法研究

王丹

2024 年 5 月

中图分类号：TQ028.1

UDC分类号：540

面向代码克隆检测的多维源代码表征学习方法研究

作　者　姓　名　王丹

学　院　名　称　计算机学院

指　导　教　师　马锐副教授

答辩委员会主席　** 教授

申　请　学　位　工学硕士

学　科　专　业　计算机技术

学位授予单位　北京理工大学

论文答辩日期　2024 年 5 月

Research on Multidimensional Source Code Representation Learning Method for Code Clone Detection

Candidate Name:	Wang Dan
School or Department:	Computer Science and Technology
Faculty Mentor:	Associate Prof. Rui Ma
Chair, Thesis Committee:	Prof. **
Degree Applied:	Master of Science
Major:	Computer Technology
Degree by:	Beijing Institute of Technology
The Date of Defence:	May，2024

面向代码克隆检测的多维源代码表征学习方法研究

北京理工大学

研究成果声明

本人郑重声明：所提交的学位论文是我本人在指导教师的指导下进行的研究工作获得的研究成果。尽我所知，文中除特别标注和致谢的地方外，学位论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京理工大学或其它教育机构的学位或证书所使用过的材料。与我一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此申明。

作者签名：_____　　签字日期：_____

关于学位论文使用权的说明

本人完全了解北京理工大学有关保管、使用学位论文的规定，其中包括：① 学校有权保管、并向有关部门送交学位论文的原件与复印件；② 学校可以采用影印、缩印或其它复制手段复制并保存学位论文；③ 学校可允许学位论文被查阅或借阅；④ 学校可以学术交流为目的, 复制赠送和交换学位论文；⑤ 学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

摘要

代码克隆检测是软件工程领域的重要任务，如何对源代码进行表征学习决定了对源代码表征抽取的程度，进而影响下游任务所能检测的精度。作为代码克隆检测任务的核心技术和研究热点，现有的代码表征学习研究存在诸多不足，例如对代码结构信息和语义信息利用不充分，特征表达不够完善；表征模型对数据集、模型结构和优化算法等多方面因素的要求高等，这些不足导致代码克隆检测效率较低。本文提出了一种多维源代码表征学习方法，旨在通过构建三个不同维度的代码表征模型，将源代码的语义信息表示为稠密低维实值向量，以在低维空间中高效计算实体和关系的语义联系，并通过特征融合得到多维特征，实现对代码信息的充分利用，以更加全面准确与智能化的方式提高代码克隆测试效率。

关键词：代码克隆检测；代码表征学习；深度学习

Abstract

Code cloning detection is an important task in the field of software engineering. How to learn representation of source code determines the degree of source code representation extraction, which in turn affects the accuracy that downstream tasks can detect. As the core technology and research hotspot of code cloning detection task, existing research on code representation learning has many shortcomings, such as insufficient utilization of code structure and semantic information, and incomplete feature expression; The representation model has high requirements for various factors such as dataset, model structure, and optimization algorithms, which leads to low efficiency in code cloning detection. This project proposes a multi-dimensional source code representation learning method, aiming to construct three different dimensional code representation models to represent the semantic information of source code as dense low dimensional real value vectors, efficiently calculate the semantic connections of entities and relationships in low dimensional space, and obtain multi-dimensional features through feature fusion to fully utilize code information and improve the efficiency of code cloning testing in a more comprehensive, accurate, and intelligent way.

Key Words: Code cloning detections; Code representation learning; Deep learning

目录

插图

表格

第 1 章　绪论

1.1　研究背景与意义

代码克隆，也叫代码复用，是指在软件系统中存在两个或两个以上的相似代码片段[1]，是软件开发中的常见现象。随着互联网时代的发展, 网络上各种开源项目越来越多样化，获取也更加便利。许多企业通过软件资源库、外部开源软件、软件产品线及开发框架等方式建立了多种多样的软件复用开发方法，同时开发人员自身也会通过多种方式大量复用已有的软件资源。在这些软件复用方法和资源的支持下，软件系统和软件产品大量引入了开源软件、网络资源、商业软件等第三方代码成分。这些第三方代码在多个软件系统中复制、传播和演化，给软件系统带来了软件质量的不确定性和风险，甚至导致漏洞的传播。

近年来，第三方代码中包含的漏洞数量呈现出快速增长的趋势。根据美国新思科技公司（Synopsys, Inc.）发布的《2023 年开源安全和风险分析报告》[2] 显示，在 2022 年审计的 1703 个代码库中，98% 的项目都包含开源代码，84% 的代码库包含至少一个已知开源漏洞，比 2022 年版的报告中增加了近 4%，有 48% 代码库中包含高风险漏洞。图1.1统计了 2018 年至 2022 年 Synopsys 审计代码库中开源代码及漏洞占比，从图中可以看出开源代码及漏洞数量整体呈上升趋势。

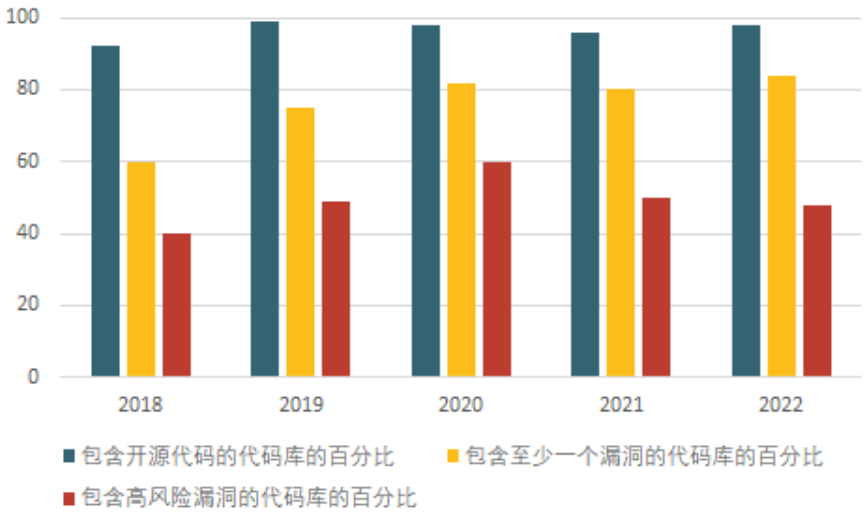

图 1.1　2018-2022 年 Synopsys 审计代码库中的开源代码及漏洞占比示意图

同时，Synopsys 统计了包含易受攻击组件的代码库占比，其中使用 JQuery 和

Lodash 两个最流行的开源组件的代码库占比达到了 47% 和 31%，其余组件占比如图1.2所示。一旦易受攻击组件出现安全问题，通常会导致软件遭受供应链攻击。据 Gartner[3] 预测，到 2025 年，全球 45% 的组织将遭受软件供应链攻击，比 2021 年增加三倍。因此，准确地检测代码克隆对于软件开发和维护是至关重要的。

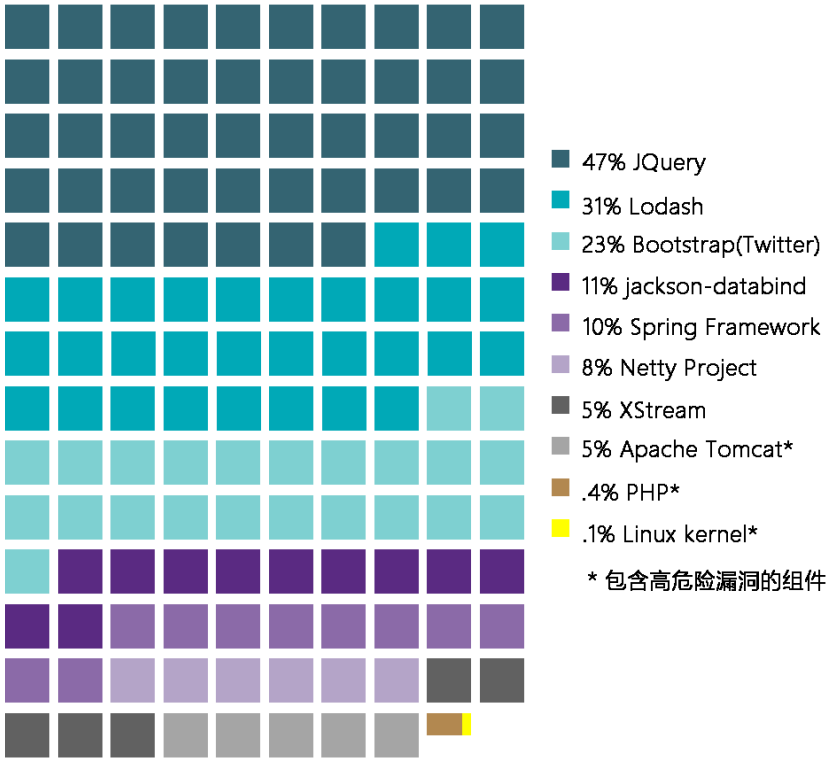

图 1.2　2022 年 Synopsys 审计代码库中包含易受攻击组件的百分比示意图

早期进行代码克隆检测通常采用人工检查并标注的方法，通过收集整理大量的代码逐行检查语法、语义结构，由人工复查筛选出正确的克隆代码并对其进行标注，由此形成了早期的代码克隆数据样本，例如，2015 年 Svajlenko[4] 等人提出了著名评估基准集 BigcloneBench，该数据集是由克隆领域三个专家评委花费 216 小时通过人工验证的方法从 IJaDataset[5] 中挖掘而来，总数据量达到 800 万，其背后的人工花费巨大。但利用人工的方法检测代码克隆效率低，成本高，并且无法保证准确率[6]，因此，有研究人员提出代码克隆检测技术，目的在于自动化定位软件系统中的代码克隆，并能够节约成本，减少出错风险[7]。

早期代码克隆检测技术通常将代码视为自然语言文本进行处理，通过文本相似性判断代码相似程度；随着编译技术的发展，研究者们将编译原理中的词法分析技术运

用到代码克隆检测领域；近年来，基于多维源代码表征学习的代码克隆检测技术已经引起了学者们广泛的兴趣，有研究人员从代码克隆检测与代码表征学习技术相结合这一方面进行了探索，试图从关键技术点入手，找到合适的结合点，以提高定代码克隆检测技术的效率和智能化程度。

1.2 研究现状与趋势

1.2.1 代码克隆检测技术

代码克隆检测技术，旨在自动化定位软件系统中的代码克隆，节省成本，减少出错风险，有助于更好地保证软件质量。目前已有的代码克隆方法大多需要对代码片段进行信息抽取，转换为中间表征，然后根据表征方式的不同计算不同代码片段之间的相似度，完成克隆检测任务。其具体流程如图1.3所示。

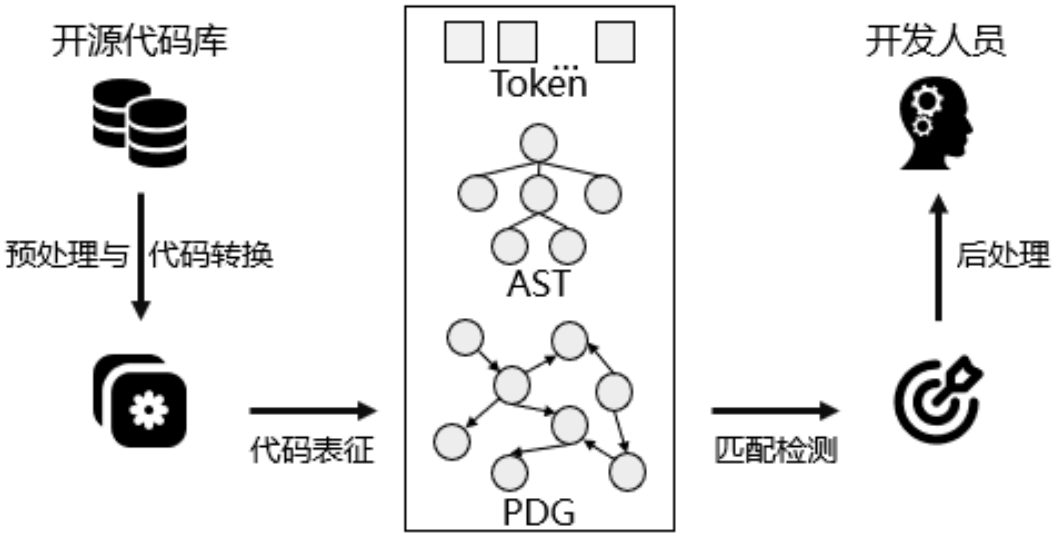

图 1.3 代码克隆检测流程

从图1.3可以看出一个完整的代码克隆检测过程通常包括预处理与转换、代码表征、匹配检测、后处理几个阶段。具体而言，一般的代码克隆检测从代码预处理与转换开始，首先删除与检测无关的空白行、注释、缩进等元素，并根据检测粒度将源代码划分为单独的片段，比如类、函数等；然后在代码表征步骤，将比较单元转换为相应的中间表示，常见的中间表示有：词法单元（Token）、抽象语法树（abstract syntax tree，AST）、程序依赖图（Program dependency graph，PDG）等；在匹配检测阶段，将根据得到不同的中间表示采用相应的匹配算法进行相似度计算，例如抽象语法树的比

较通常采用子树匹配算法，程序依赖图的比较则采用子图同构算法。此阶段将代码片段两两对比，以查找相似代码源片段，得到代码克隆对。最后在后处理阶段，通常会通过人工检测或者算法过滤掉错误的代码克隆，并以适当的方式呈现给开发人员提供帮助。

在这些步骤中，代码表征方式决定了匹配检测方法的预处理方式、模型设计、部署方式、运行效率，并影响最终结果[8]。比如将源代码表征为文本，其预处理过程主要为去除噪声，如空格、注释等，其比较算法可以利用文本相似的一系列方法，能够检测到语法相似的克隆代码；而如果表征为抽象语法树，则其预处理过程需要解释器的参与，相似比较算法更多地考虑了结构相似等，能够检测到语法层面相似的代码克隆。因此, 源代码表征方式是代码克隆检测的关键步骤。

1.2.2 代码表征学习

代码表征是对代码数值化的一种技术，把代码映射为一组连续的实值向量，提取隐藏在代码内部的属性，辅助程序员生成或分析代码，是代码克隆、代码推荐、代码剽窃等软件工程任务的核心技术和研究热点[9]。代码表征学习是指利用机器学习或深度学习技术从源代码中学习有效的表示，以便于后续的代码克隆检测等软件工程任务。在代码克隆检测中，代码表征学习可以用来提取代码片段更高层次的抽象特征表示，这些特征表示能够捕捉代码的语法、语义以及结构信息。通过学习到的代码表征，可以更准确地比较和识别不同代码片段之间的相似性，从而实现克隆代码的检测和管理，提高检测的准确性和鲁棒性。因此，代码表征学习为代码克隆检测提供了重要的技术支持，从而更有效地利用机器学习和深度学习技术来处理大规模的代码库，并提高代码克隆检测的性能和效率。

代码表征学习工作最早可追溯到 100 年前，基于传统机器学习的数据特征学习被广泛提出。1901 年，K.Pearson 提出最著名的主成分分析 (Principal Component Analysis，PCA) 方法[10]，即通过一个线性映射学习复杂高维数据的低维表示。1936 年线性判别分析 (Linear Discriminant Analysis，LDA) 被 Ronald A. Fisher 提出[11]，这是经典的有监督的表征学习方法，Fisher 判别准则也由此而来。随着代码表征学习工作的不断发展，神经网络逐渐被用于特征学习，但直到 1986 年，Geoffrey Hinton[12] 发现反向传播算法（BP Algorithm）可以在网络的隐藏层里学习到有用的关于输入数据的内在表征。2006 年，Geoffrey Hinton 提出贪婪分层预训练和深度神经网络微调[13] 的方法，从而

解决了困扰神经网络用于特征学习的两大难题：模型过拟合（Model Overfitting）和梯度扩散（Gradient Diffusion）。随着计算机计算能力的提升和深度神经网络结构的不断发展，人们更多地使用深度神经网络来更有效地提取数据的特征，用于后续的分类或预测。2013 年,Y Bengio 等人发表了关于表征学习的经典综述[14]。2016 年，Bengio 和 IGoodfellow 等人合著的《Deep Leanring》一书中也为表征学习专著一章[15]。

近些年来，源代码表征学习方法被用于代码克隆检测、代码推荐、代码剽窃等多个代码分析任务中，取得了一定的成就。根据源代码的抽象层次不同，现阶段代码表征学习工作可以分为基于 Token 的代码表征、基于树的代码表征、基于图的代码表征、基于语法和语义混合的代码表征四类。

（1）基于 Token 的代码表征

基于 Token 的代码表征通常利用词法分析器将代码中的词汇单元（Token）划分出来。这些词汇单元通常包含关键字、数字、标识符等。将代码表示为词汇单元序列之后，利用深度学习技术对其进行建模，学习代码序列中所包含的有效信息，如功能语义信息、语法结构信息等，最后生成具有丰富代码信息的表征向量，应用于后续的代码克隆检测任务中。

著名的 CCFinder[16]、CP-Miner[17] 等克隆检测工具都是基于 Token 级的，可以很好地检测完全相同的代码对以及参数化后的代码对克隆问题。其中，CCFinder 将源代码中的每一行单独转换为 Token 序列，根据转化规则对 Token 进行修改，将类型名、变量名、常量的标识符替换为指定的特殊 Token，最后利用后缀树匹配算法来查找代码克隆。CP Miner 增加了 Bug 检测，该工具的检测速度、检测精度相较于 CCFinder 有了很大的提高。

CCLearner[18] 是第一个使用神经网络在 Token 级别进行代码克隆检测的方法，它使用 BigCloneBench[4] 作为训练样本，抽取了其中方法级别的 Token 序列，从已有的标记数据集对 DNN 模型进行训练，学习保留字、类型标识符、方法标识符和变量标识符等八个特征并用于代码克隆检测。

Mikolov 等[19] 利用 Word2vec、GloVe、BERT 进行 Token 的预训练，通过无标注样本训练深度网络结构，使用标注样本进行模型参数微调，从而提升模型性能。其中 BERT[20] 是双向 Transformer 的编码器，通过遮蔽语言模型和下一句预测 2 种预训练目标来调整模型参数。

Feng 等[21] 提出多模态的预训练模型，利用不同模态的信息互补作用，有效提升

了模型的整体表征能力。CodeBERT 基于文档和代码，在自然语言和程序语言双模态下，利用 BERT 进行预训练，提取自然语言和程序语言之间的语义连接，为下游任务提供通用表示向量。

上述方法均在 Token 上进行代码的表征学习，力图充分提取代码中的属性信息。

（2）基于树的代码表征

抽象语法树 AST 是源代码的抽象语法结构的树状表示，可以有效地表示程序的语法及其结构，利用深度神经网络对抽象语法树进行建模得到其向量表示，根据该特征向量完成代码克隆检测任务，实现基于树的源代码表征。

White 等人[22] 提出了一种基于循环神经网络的代码表征方法，该方法将代码分为词汇以及句法两个层次。对于词汇级别的信息，该方法在代码的词汇单元序列上使用 RNN 神经网络进行建模。而对于代码的句法级别的信息，首先将代码转换为其对应的抽象语法树结构，之后将抽象语法树转换为其对应的满二叉树，最后将满二叉树转换为橄榄树，并在其上使用另一个 RNN 神经网络进行建模。该方法将这两个特征相结合作为整个程序的特征向量，根据该向量进行代码克隆检测任务。

Mou 等人[23] 提出了一种基于树的卷积神经网络模型（tree-based convolutional neural network，TBCNN）。该模型采用了“连续二叉树”的概念，直接在代码所对应的抽象语法树上进行卷积操作。在卷积操作之后获得了不同数目的 AST 结构特征向量，由于数目不同不能直接作为神经网络的输入，因此该方法还采用“动态池化”技术，最终将数目不同的特征向量转换为了一个向量。TBCNN 是一个通用的代码表征生成模型，所生成的向量能够包含代码片段中特有的代码模式，因而可以应用于不同的代码分析任务中。

Wei 等人[24] 提出了 CDLH(Clone Detection with Learning to Hash) 方法，该方法在抽象语法树的基础上使用 LSTM 模型，通过共享权重的方式学习两个代码片段之间的表征向量，然后利用特定哈希函数来将其编码为二进制哈希码，最后工具通过计算哈希码的汉明距离来检测代码克隆。

Chen 等人[25] 采用了基于树的卷积神经网络，该方法考虑到已有的方法仅仅使用抽象语法树会造成代码语义信息的丢失，因此该方法将 API 的调用信息作为补充信息合并到抽象语法树中，之后进行代码表征。在 POJ104[23] 和 BigcloneBench[4] 数据集上进行实验，在 F1 指标上获得了 0.39 和 0.12 的提升。

Zhang 等人[26] 提出了一种基于抽象语法树的神经网络代码表征方法，该方法将

代码转换为其对应的抽象语法树，然后将完整的抽象语法树分割为多个语句树。针对每个语句树，该方法设计了语句编码器用于将语句树转换为对应的语句表征向量，通过使用双向 GRU 神经网络对语句向量进行建模，对双向 GRU 层输出的隐含状态向量进行最大池化操作，以获得最显著的代码特征。该方法所生成的代码表征向量被应用于代码克隆检测任务中，在 POJ104[23] 和 BigcloneBench[4] 数据集上取得了当时最好的检测结果。

上述方法均在抽象语法树上进行代码的表征学习，力图充分提取代码中的结构信息。

（3）基于图的代码表征

程序依赖图 PDG 是程序的一种图形表示，所含结构信息最多，能够表示程序的控制依赖，数据依赖以及地址依赖等关系，是一种带有标记的有向多重图。通过将程序表示为图的形式使得模型能够更好地理解代码中不同部分之间的依赖关系。

Allamanis 等人[27] 考虑到代码中的长依赖问题，如在代码中变量的定义位置与使用位置之间的距离问题，提出了基于图的代码表征方法，旨在学习代码中的语法以及语义结构。首先将代码转换为对应的抽象语法树，之后通过不同的连接规则连接抽象语法树各个节点，获得了包含变量之间依赖关系在内的不同节点之间的关联关系；最后将构建好的代码图数据作为输入，输入到图神经网络中进行表征学习。

Lu 等人[28] 从代码中提取数据流与函数调用信息，将其融合到抽象语法树中，从而将代码构建为一个包含丰富信息的图结构表示。在传统的 GGNN 模型上引入了注意力机制，用于获得图中每个节点的重要程度，进而获得更具有区分度的代码表征向量。

Brockschmidt 等人[29] 同样在代码的抽象语法树上增加相应的边以构建代码图，代码图的构建方法与文献[27] 类似。之后采用图神经网络对程序的结构和数据流进行建模完成代码表征任务。

Ben-Nun 等人[30] 提出了一种与语言以及平台无关的代码表征方法 inst2vec。该方法首先使用编译器对代码进行编译，得到代码的中间表示。但由于该中间表示并没有包含代码之中的数据流信息以及控制流信息，因此该方法将数据流和控制流也融合到该中间表示中，进而构建了代码上下文流图。最后在所构建的图上使用循环神经网络进行建模，获得代码的表征向量。该向量在程序分类实验中的准确率取得了当时最好的效果。

Wang 等人[31] 考虑了仅仅使用代码的抽象语法树进行代码表征建模实际上仍然有代码结构上的缺失这一问题，构建了代码抽象语法树的图形表示 FA-AST，通过将抽象语法树各个叶子结点相连构建出适合图神经网络处理的数据，然后应用两种不同类型的图神经网络 GNN 来检测克隆。

DeFreez 等人[32] 提出了 Fun2Vec 方法，解决代码的路径爆炸问题，该方法采用随机游走算法，随机选择部分执行路径，捕获程序的层级结构，每条执行路径转换为一个标签序列，借助 Word2Vec 方法，把标签映射为连续实值向量，并通过神经网络训练函数的嵌入向量。

Duan 等人[33] 提出了一种无监督的程序代码表示学习技术 DEEPBINDIFF，依靠代码语义信息和程序控制流信息生成基本块嵌入，并且采用 k-HOP 贪婪匹配算法利用基本块嵌入发现最优的相似性结果。通过大量二进制文件和真实的 OpenSSL 漏洞对原型进行评估，结果表明 DEEPBINDIFF 相比于最先进的工具，跨版本和交叉优化级别都更优。

Kang 等人[34] 提出了一种基于门控神经网络的 CC-GGNN 方法来解决代码补全问题。CC-GGNN 通过从代码表示中获得有效的代码特征，提出了一种分类机制，通过使用已知的父节点对节点的表示进行分类，并在模型中构建训练图。实验结果表明，模型在数据集中最多优于最先进的方法 MRR 最多 9.2%，ACC 最多 11.4%。

上述方法均在图上进行代码的表征学习，力图充分提取代码中的语义信息。

（4）基于语法和语义融合的代码表征

基于语法与语义融合的模型，结合 AST、DFG、CFG、Token 序列，捕获程序的语法及语义结构信息。其中抽象语法树 AST 和 Token 序列反映了语法层面的信息，DFG、CFG 反映了语义层面的信息。

Tufano 等人[35] 采用四种不同的代码表征方法（即标识符、抽象语法树、字节码和 CFG）进行代码克隆检测，他们利用四种代码表示分别识别代码对的相似度，并计算平均值作为最终的相似度结果。

Fang 等人[36] 结合抽象语法树，控制流图和调用图来学习代码特征，融合了语法和语义信息。首先，该方法从源码中分析出方法之间的调用图、每个方法的抽象语法树以及每个方法的控制流图；然后，用调用图将找出每个功能的 AST 集合。通过 AST 集合抽取功能的语法信息；通过调用图组成每个功能的控制流图；通过控制流图抽取功能的语义信息；最后，将抽取出来的语义信息送入前馈神经网络得到分类结果。

Hua 等人[37] 结合标记、抽象语法树和控制流图三种方式实现检测目标。文中提出了使用注意力的代码克隆检测器（FCCA），这是一种基于深度学习的代码克隆检测方法，它通过保留多个代码特征，包括非结构化（以顺序令牌形式的代码）和结构化（以抽象语法树和控制流图形式的代码）信息，在混合代码表示的基础上进行代码克隆检测。将多个代码特征融合到混合表示中，该混合表示配备有注意力机制，有助于最终检测精度的重要代码部分和特征。

Dong 等人[38] 提出了一种基于 Token 和 AST 的代码表征方式，提取数量特征如 AST 中的 AST 树的高度、节点数以及标记中操作数的个数、字符串的个数等作为神经网络的输入进行检测。

亚芳等人[39] 提出了一种基于图像相似度的代码克隆检测技术。该方法区别于传统方式，从图像处理角度提出了一种基于图像相似度的新型代码克隆检测 (CCIS) 方法。首先对源代码进行移除注释、空白符等操作，以获取” 干净” 的函数片段，并将函数中的标识符、关键字等进行高亮处理；然后将处理好的源代码转换为图像，并对图像进行规范化处理；最后使用 Jaccard 距离和感知哈希算法进行检测，得到代码克隆信息。

Saini 等人[40] 提出了一种代码克隆检测框架 Oreo，该方法从程序的源代码中提取了包括被调用的外部方法的数量、变量的数量、语句的数量、循环的数量等 24 种度量，然后进一步从函数中抽取语义，并使用了基于哈希的方法进一步筛选，最后加入了深度学习的方法，将两个程序向量输入到孪生模型中来判断两个程序之间是否具有克隆关系。

SrcClone[41] 是一种基于切片的克隆检测方法，主要用于检测语法和语义代码克隆，该方法使用了轻量级、公开可用、可扩展的程序切片器。具体来说，SrcClone 使用切片信息计算基于切片的指标，这些指标可以用来计算某些切片向量以近似切片内部的结构信息，然后在定制的局部敏感哈希算法中使用这些信息来有效地散列和聚类相似的向量。如果两个代码片段对应的片段也相似，则 srcClone 认为它们在语义上相似。这些类似的切片表示候选克隆类。

总体而言，代码克隆检测是软件工程领域一项重要任务，如何对代码进行合适的表征是代码克隆检测的关键问题。代码表征学习决定了对源代码信息抽取程度的上限，决定了检测技术的预处理方法、模型设计、部署方式、运行效率，并会影响最终结果。面向代码克隆检测这一下游任务，代码表征学习研究存在以下不足：对代码结构

信息语义信息利用不充分，特征表达不够完善；表征模型对数据集、模型结构和优化算法等多方面因素的要求高等问题，这些不足严重制约着代码克隆检测技术的发展。因此，研究人员一方面通过对源代码进行充分利用，提出多维源代码表征方法，从而提高代码克隆检测能力；另一方面，通过研究更先进的算法来提高表征模型的自动化和智能化程度，也是目前重要的发展趋势。

1.3 研究内容

本文主要围绕如何将源代码表征学习技术应用到代码克隆检测领域，通过不同维度对程序表征进行学习，并基于学习得到的语义特征进行克隆对的判定，充分发挥代码表征学习技术检测代码克隆的能力。针对现有代码表征学习方法存在的对代码结构信息和语义信息利用不充分的问题，本文提出面向代码克隆检测的多维源代码表征学习方法 RLCCD，旨在通过构建三个不同维度的代码表征模型，将源代码的语义信息表示为稠密低维实值向量，以在低维空间中高效计算实体和关系的语义联系，并通过特征融合得到多维特征，实现对代码信息的充分利用，以更加全面准确与智能化的方式提高代码克隆测试效率。

本文的主要工作包括：

（1）提出面向代码克隆检测的多维源代码表征学习方法 RLCCD

本文提出了一种面向代码克隆检测的多维源代码表征学习方法 RLCCD，该框架主要针对代码表征关键步骤提出三个关键技术点，从 Token 序列、抽象语法树 AST、程序依赖图 PDG 三种不同维度对代码特征表示进行优化，分别形成了基于预训练辅助模型的 Token 表征学习、基于子树划分的抽象语法树表征学习、基于图过滤的程序依赖图表征学习三种方法，然后通过特征融合将三种维度特征整合为一个多维特征，实现对代码信息的充分利用，以更加全面准确与智能化的方式提高代码克隆测试效率。

（2）基于预训练辅助模型的 Token 表征学习

针对目前现有的基于 Token 的表征学习方法通常将代码表示为词汇单元，为了后续生成表征向量通常会将词汇单元规范化，丢失部分语法信息，出现在词汇表中不存在 Token 的难题，提出了一种基于预训练辅助模型的 Token 表征学习方法。该方法在模型训练之前，通过选取预训练辅助模型从代码语料库中学习基本单元的语法语义信息，以及这些单元之间的联系，最终给出一份单词-向量形式的词汇表，从而减少出现

集外词问题的概率。本文在 POJ104 数据集上的消融实验评估表明，预训练辅助模型方法能够提高代码克隆检测准确率。

（3）基于子树划分的抽象语法树表征学习

针对现有的基于树的表征学习方法通常将抽象语法树转换为完整二叉树，可能破坏源代码原有的语法结构，增加 AST 高度，丢失长期上下文信息，削弱了神经模型捕捉更真实和复杂语义的能力，导致梯度消失的难题，提出了一种基于子树划分的抽象语法树表征学习方法。该方法将每个大型的 AST 分割成小语句树序列，并通过捕获语句的词法和句法知识将每一个语句树都编码成一个向量，在得到一个语句向量序列后，将语句向量序列输入树卷积神经网络中生成代码片段的结构向量表示。本文在 POJ104 数据集上的消融实验评估表明，子树划分方法能够有效提取结构特征，提高代码克隆检测准确率。

（4）基于图过滤的程序依赖图表征学习

针对现有的基于图的表征学习方法通常将程序表征为有向多重图，继而采用图匹配算法将图中的控制流和数据流编码为一个紧凑的语义特征矩阵，矩阵中每个元素都是高维系数特征向量，所消耗的时间、空间开销巨大的难题，提出了一种基于图过滤的程序依赖图表征学学习方法。该方法通过收集 PDG 的简单特征来过滤掉明显不可能为克隆的 PDG 对。具体的，根据 PDG 的节点个数、控制边数、执行边数、数据边数、声明节点数、函数调用数、传入参数、传出参数等代表特征进行过滤，在大幅减少候选 PDG 对规模的同时，保证真正的克隆对不会被过滤掉而导致整体克隆检出率的降低。本文在 POJ104 数据集上的消融实验评估表明，图过滤方法能够有效减少时间、空间开销，提高代码克隆检测准确率。

（5）特征融合及实验验证

针对不同的特征采用不同的代码表征模型对代码片段进行多维源代码特征学习，从而提高对源代码特征提取的程度，并通过特征融合得到一个更能代表代码信息的多维特征，该多维特征能够在低维空间中高效计算实体和关系的语义联系，提高后续代码克隆检测任务的准确率。

本文选取了代码克隆检测领域常见的基准集 POJ104 进行实验验证，并与现有开源的 SourcererCC[42]、ASTNN[26]、SCDetector[43] 方法进行比较，主要通过召回率 (Recall)、精确度 (Precision) 和准确率（Accuracy）三个指标评价实验结果，实验结果验证了 RLCCD 的可行性和有效性。

1.4 论文结构

本文共由 6 章组成，具体的组织结构如下：

第 1 章 绪论部分首先对本文的研究背景与意义进行了阐述，并对代码克隆检测技术和代码表征学习技术的研究现状与趋势进行了分析总结，进而提出本文的主要研究内容，最后对全文的组织结构进行了介绍。

第 2 章 分析了代码表征学习领域的关键技术挑战，基于此，提出了本文的面向代码克隆检测的多维源代码表征学习方法 RLCCD，并对该技术的整体框架进行了介绍，进而根据所提框架简要论述了本文研究的关键技术，即基于预训练辅助模型的 Token 表征学习方法、基于子树划分的抽象语法树表征学习方法、基于图过滤的程序依赖图表征学习方法。

第 3 章 介绍基于预训练辅助模型的 Token 表征学习方法的设计与实现。首先，分析其研究动机，即目前 Token 表征学习面临的集外词问题，继而提出基于预训练辅助模型的方法设计，详细介绍该方法的设计思路和具体实现，最后，对该方法的有效性进行消融实验验证。

第 4 章 介绍基于子树划分的抽象语法树表征学习方法设计与实现。首先，分析其研究动机，即目前抽象语法树表征学习面临的梯度消失问题，继而提出基于子树划分的方法设计，详细介绍该方法的设计思路和具体实现，最后，对该方法的有效性进行消融实验验证。

第 5 章 介绍基于图过滤的程序依赖图表征学习方法设计与实现。首先，分析其研究动机，即目前图表征学习面临的计算开销大问题，继而提出基于图过滤机制的方法设计，详细介绍该方法的设计思路和具体实现，并给出了针对该方法有效性的消融实验验证。

第 6 章 介绍特征融合及本文研究框架 RLCCD 的实验验证。首先，针对特征融合的方法设计与具体实现进行了介绍。接着，对 RLCCD 框架的有效性进行了评估，通过与现有开源技术 SourcererCC、ASTNN、SCDetector 进行实验对比，验证 RLCCD 方法的有效性。

结论 对全文的研究进行了总结，并提出对未来工作的展望。

第 2 章　多维源代码表征学习方法总体设计

本章首先介绍代码表征学习面临的一些关键技术挑战，基于此提出本文的面向代码克隆检测的多维源代码表征方法框架 RLCCD 的研究方案，并就该框架的设计思路和总体架构进行详细阐述。此外也简要介绍了该框架包含的 3 个流程，即代码预处理、多维源代码表征学习、克隆检测任务实现。

2.1　面向克隆检测的代码表征学习关键技术挑战

目前已有的代码克隆检测方法大多遵循以下思路：（1）首先对代码片段进行预处理；（2）对处理好的代码片段进行代码表征，将其转换为中间表征；（3）根据表征的方式不同计算不同代码片段之间的相似度，完成克隆检测任务。在代码克隆检测中，源代码表征方式决定了后续克隆检测方法的算法选择。从目前多种维度的代码表征方法来看，现有的代码表征方式存在以下技术挑战：

（1）Token 维度代码表征存在集外词问题

基于 Token 的方法一般会将源代码表示为 Token 序列，然后通过词频统计，使代码文本形成一组由词频构成的数字向量，最后利用相似度计算的算法来计算两段代码的相似性。这类方法和自然语言处理（NLP）领域中常用来处理文本的方式很相似，产生一个规模巨大且稀疏的词汇表。但是，在大多数基于 token 的代码克隆检测工具中，通常会将 token 规范化，例如：将变量名用统一的标识符来代替。经过规范化的 token 产生的词汇表较小，导致模型学习能力有限，并且在训练过程中会出现未见过或未包含在词汇表中的词语。这些词语可能是用户自定义词、拼写错误、缩写、专有名词等。由于模型在训练阶段没有足够的信息来学习这些词语的表示，因此在实际应用中无法正确处理这些词语，从而导致模型的性能下降，这就是集外词（Out of vocabulary，简称 OOV）问题。集外词问题会对模型的性能和泛化能力造成影响，严重限制了代码表征的有限性。

（2）树维度代码表征存在梯度消失问题

基于树的方法将代码转换成相应的抽象语法树，然后通过树匹配算法计算其相似度。与自然语言处理领域的长文本类似，当上下文序列很长的时候，基于树的神经网络模型容易出现梯度消失的问题，即梯度在训练过程中变得越来越小，特别是当树非

常深的时候，模型会面临梯度消失问题。目前大多数基于树的代码克隆检测方法为了简化或者提高效率，通常会将生成的抽象语法树转换为完整的二叉树，在转换的过程中，不仅破坏了源代码原有的语法结构，也会增加树的高度，进一步削弱模型捕捉复杂语义的能力，导致检测性能下降。

（3）图维度代码表征存在算法开销大问题

基于图的方法会将源代码表征为程序依赖图或者控制流图，然后应用图匹配算法来寻找最大的相似子图。大多数基于图的代码克隆检测工具任务的核心是将图中的每个节点映射到一个低维、稠密的特征向量中，并将这些特征编码为特征矩阵，这一步通常需要大量空间开销。同时子图匹配算法是 NP 完全问题，计算成本过长，时间复杂度很高，因此图维度代码表征学习会存在算法计算开销大，可扩展性不好，检测结果召回率低等问题。

2.2 RLCCD 框架研究方案

针对上述提出的技术中面临的关键挑战，本文提出面向代码克隆检测的多维源代码表征方法 RLCCD。本节首先介绍 RLCCD 框架的研究思路及总体框架，然后根据流程介绍代码处理、多维源代码表征学习、克隆检测任务实现三个步骤，其中，多维源代码表征学习步骤中分别针对2.1节提出的 3 个关键挑战提出了对应的解决办法。

2.2.1 研究思路及总体框架

基于上述关键技术挑战，本文首先针对现有研究工作中代码表征学习方法存在的对代码结构信息和语义信息利用不充分、表征模型对数据集依赖过高等问题，提出面向代码克隆检测的多维源代码表征学习方法，旨在通过构建三个不同维度的代码表征模型，将源代码的语义信息表示为稠密低维实值向量，以在低维空间中高效计算实体和关系的语义联系，并通过特征融合得到多维特征，实现对代码信息的充分利用，以更加全面准确与智能化的方式提高代码克隆测试效率。

源代码的表征方式决定了信息抽取的程度和粒度，进而影响深度学习模型的精度和效率，是代码可读性评估的基石。考虑到经由不同表征方式处理所得到的信息通常具有互补性，因此本文将从字符级、词条级等不同维度构建合适的编码方案。

基于混合代码表示，通过保留多个代码特征，包括非结构化 (顺序令牌形式的代码) 和结构化 (抽象语法树和控制流图形式的代码) 信息。将多个代码特征融合成一个

混合表示，该混合表示具有关注机制，该机制关注重要的代码部分和有助于最终检测准确性的特征。

混合表示学习 (Hybrid representation learning, HRL)[15] -[18] 最近成为深度表示学习的一个很有前途的分支，它在将异构信息嵌入到统一的低维空间中表现出了强大的能力。生成的表示对于支持对复杂数据 (如源代码) 进行更好的建模特别有用。此外，注意机制的概念是从表征学习中产生的，目前正在发展成为一种重要的精度增强机制。除了在广泛的应用中提高性能外，注意机制在提高深度学习模型的可解释性方面也发挥着关键作用，有利于机器翻译 [19]、机器阅读理解 [20]、图像字幕 [21] 和文档分类 [22]

基于上述研究思路，本文提出的面向代码克隆检测的多维源代码表征方法 RL-CCD 框架如图2.1所示，由图可见，本文提出的基本框架与代码克隆检测的处理流程基本一致，并主要通过三个维度对代码表征环节进行优化。

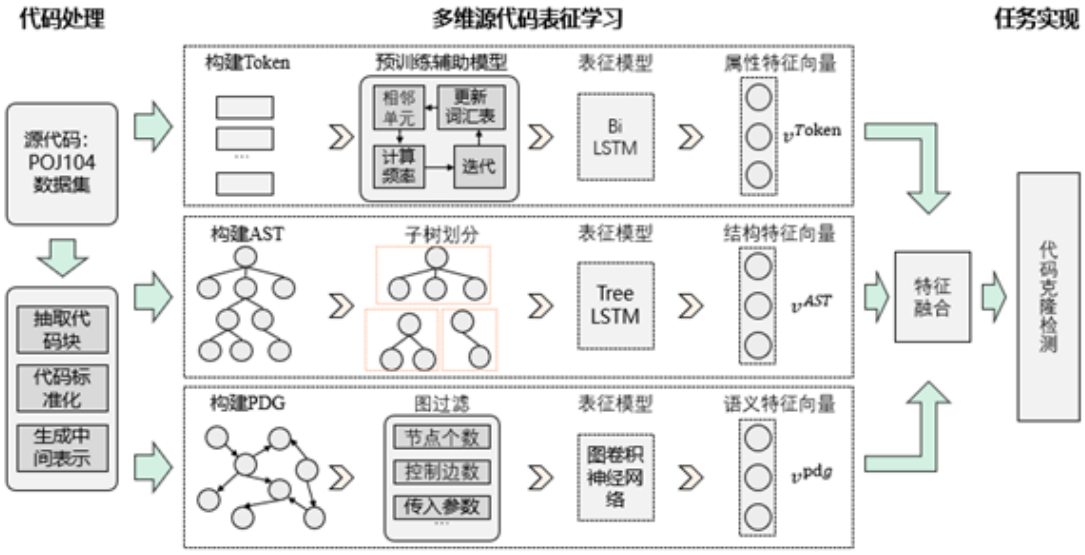

图 2.1　RLCCD 总体框架

下面则分别对代码处理、多维源代码表征学习、克隆检测任务实现的流程和关键技术点进行介绍。

2.2.2　代码处理

代码处理的目标是生成源代码片段对应的 Token 序列、抽象语法树和程序依赖图，主要包含 3 个流程：抽取代码块、代码标准化、生成中间表示。首先，使用 TXL

工具从源代码中提取出代码块。这里的源码来自 POJ104 公共数据基准集，而 TXL 是专门为软件分析和源代码转换任务设计的一个分析工具，可以很好地支持 C 语言；然后，对代码块中的代码进行代码标准化处理。具体的，首先需要去除代码块中的注释、空格和空行，然后根据一定的转化规则进行代码标准化；最后，基于标准化后的代码片段生成对应的中间表示：Token 序列、抽象语法树 AST 和程序依赖图 PDG。

2.2.3 多维源代码表征学习

RLCCD 框架的核心步骤是源代码表征学习，其目标是学习能够表示代码片段的连续向量，表现程序理解的认知层次，获取程序的语法、语义信息，创建程序更高抽象层次上的表示，它决定着对源代码信息抽取程度的上限，决定着检测方法的预处理方式、模型设计、部署方式、运行效率，并影响后续代码克隆检测任务所能检测的精度。下面从 Token 序列、抽象语法树 AST、程序依赖图 PDG 三种不同维度的代码特征表示出发，详细介绍研究方案并分析其优化改进。

（1）针对 Token 序列特征挖掘，提出预训练增强辅助模型提取属性特征，从而解决传统基于 Token 序列的方法存在的集外词问题；

（2）针对抽象语法树 AST 特征挖掘，提出子树划分的改进方法提取结构特征，从而解决传统基于抽象语法树的方法存在的梯度消失问题；

（3）针对程序依赖图 PDG 特征挖掘，提出过滤机制提取语义特征，通过收集 PDG 的简单特征来过滤掉明显不可能为克隆的 PDG 对，从而解决传统基于程序依赖图的方法存在的计算开销大问题。

（4）特征融合方法特征融合的目标是将提取到的属性特性、结构特征、语义特征合并，得到一个更能代表代码信息的多维特征，更具有判别能力。

按照具体的技术，特征融合包括特征拼接、特征求和（均值、pooling、加权求和）、特征之间对应元素相乘、特征之间求外积再送入神经网络、跳跃连接（skip）、反卷积、典型相关分析 CCA、注意力机制（包括 self-attention）加权求和、构图然后采用图神经网络等等。

2.2.4 克隆检测任务实现

克隆检测任务实现的目标是通过计算两个代码的向量距离来判断是否存在代码克隆。常见的向量距离包括欧式距离、曼哈顿距离等。由于代码克隆检测问题是一个

二分类问题，即给定两个代码片段，需要输出 0 或 1，0 表示它们之间不相似，1 表示相似。因此需要将向量距离 d 映射到 0 1 之间，这里可以通过 sigmoid 函数、tanh 函数等做映射，其函数值表示两个代码片段的相似度。然后设定阈值，当相似度大于阈值，则判定两个代码片段属于代码克隆，输出 1。

2.3 本章小结

本章首先分析了源代码表征学习在代码克隆检测过程中所面临的关键技术挑战，主要表现为 Token 集外词问题、树梯度消失问题、图计算开销大问题。针对上述提出的三个问题，提出了本文的方法 RLCCD，并介绍了其整体框架和处理流程，对其中的关键技术点进行了简要的论述。

第 3 章　基于预训练辅助模型的 Token 表征学习

本章主要对本文提出的基于预训练辅助模型的 Token 表征学习方法进行详细介绍，首先介绍其基本思想，接着阐述其方法设计，以及具体的实现过程，最后介绍实验验证过程和结果。

3.1　研究动机

本文针对现有存在的问题，

基于 Token 的代码表征通常利用词法分析器将代码中的词汇单元（Token）划分出来。这些词汇单元通常包含关键字、数字、标识符等。将代码表示为词汇单元序列之后，利用深度学习技术对其进行建模，学习代码序列中所包含的有效信息，如功能语义信息、语法结构信息等，最后生成具有丰富代码信息的表征向量，应用于后续的代码克隆检测任务中。现有方法大多都对 token 进行了规范化，比如将变量名用统一的标识符来代替，这样存在的问题是会丢失部分词法信息，但是如果不进行规范化，会存在集外词（Out-of-vocabulary，简称 OOV）的问题，即出现了在词汇表中不存在的 token。集外词 OOV 问题严重限制了代码表示的有效性。

针对上述问题，本文采用预训练增强的辅助模型得到词汇表，从而提高代码克隆检测的准确率。

3.2　方法设计

本节将主要介绍基于预训练辅助模型的 Token 表征学习方法设计与实现，首先介绍该方法的整体框架，并分别从预训练辅助模型、Token 表征学习的具体设计及实现。

3.2.1　框架概述

基于预训练辅助模型的 Token 表征学习：

3.2.2　预训练辅助模型

Word2vec 模型多次迭代维护词汇表，从而减少 OOV 问题

在模型训练之前，通过选取合适的模型从代码语料库中学习基本单元的语法语义信息，以及这些单元之间的联系，最终给出一份单词-向量形式的词汇表，从而减少出现集外词问题的概率。具体的，第一次迭代选择相邻的两个 token 组合为一个单元，查找出最频繁的 token 组合确定为一个组合单元，并将组合单元更新到词汇表中，然后二次迭代，每次迭代在原来的基本单元上再组合一个新的邻近 token 作为新的判断单元，每次迭代都会更新词汇表。在得到词汇表之后，根据词汇表获取每个单元的向量表示。

3.2.3 Token 表征学习

采用 BiLST 模型

将 token 序列对应的向量输入到 Bi LSTM 网络中，经过 Bi LSTM 学习哪些信息应该被记住哪些信息应该被遗忘，最终得到每个基本单元包含语义信息和上下文信息的向量表示。Bi LSTM 由一个正向的 LSTM 和一个反向的 LSTM 组成，主要思想是通过把序列向前、向后分别输入给两个独立的递归网络，这两个子网络连接到一个输出层，在每个词的输出部分把两个子网络的输出信息进行整合，这样网络就同时拥有了序列中每个词的过去时刻信息和未来时刻信息。Bi LSTM 可以捕获到序列前后的关系依赖，将代码片段的 Token 序列转换为可相互比较的向量。

3.3 实验验证

为了验证基于预训练辅助模型的 Token 表征方法的有效性，

3.3.1 实验设计

（1）实验环境本章的实验验证均运行于 Linux 系统下，其系统硬件配置如表3.1所示。

（2）实验数据集

为了验证预训练辅助模型在 Token 层面上表征学习的有效性，本文面向代码克隆检测任务对预训练辅助模型进行分析和评估，选取的实验数据为 POJ104 数据集。如表3.2所示，POJ104 数据集是一个基于 C 语言所构建的大型数据集。OJ 系统是一个以编程教学为目的公开评判系统，共存在 104 个编程问题，针对每个编程问题，学生们

表 3.1　实验环境配置

环境	配置
操作系统	Ubuntu 20.04
处理器	Intel Core i9-12900KF × 24
内存	31.1G
显卡	NVIDIA
磁盘	1TB

表 3.2　POJ104 数据集

代码	属性
Dataset	POJ104 数据集
Language	C
Program	52000
Classes	104
Max tokens	8737
Avg tokens	245

通过在线提交自己的代码来尝试解决，同时 OJ 系统将自动判断提交源代码的正确性和有效性。对于 OJ 系统中同一个编程问题来说，其所有正确提交的代码都为克隆代码，对于不同的编程问题所提交的代码，即为非克隆代码。POJ104 数据集针对每一个编程问题，均提供 500 个学生提交源代码，即共有 52000 个样本。

得到 POJ104 数据集后，本文首先对数据集进行初步筛选，去掉其中包含乱码的样本，共得到 51485 个源代码样本。然后对源代码进行预处理，删除样本中包含的空白行和注释等多余代码，并将数据集保存到一个 program.pkl 文件中，program.pkl 文件中一共包含 51485 行 ×3 列的数据集，每一行数据代表一个源代码样本，第一列为源代码 id，第二列保存源代码样本 code，第三列为源代码的标签 label，即属于哪一个编程问题。接着，本文随机两两组合同一标签 label 的源代码，组成 5200 个真克隆对，随机组合不同标签的源代码组成 44800 个假克隆对，一共给包含 50000 个克隆对，并将其保存到 oj_clone_ids.pkl 文件中，oj_clone_ids.pkl 文件中一共包含 50000 行 ×3 列的数据集，每一行数据代表一个克隆对样本，第一列为源代码 id1，第二列为源代码 id2，第三列为克隆对的标签 label，真克隆对标签为 1，假克隆对标签为 0。最后，依据随机种子将数据集按照 3：1：1 划分为训练集、测试集、验证集，其中的正负样本数如下表3.3所示。

（3）评估指标

表 3.3　本文预处理后的 POJ104 数据集正负样本数

数据集	真克隆对	假克隆对	克隆对数
训练集 train	3162	26838	30000
测试集 test	1022	8978	10000
验证集 dev	1016	8984	10000
总计	5200	44800	50000

表 3.4　分类问题的混淆矩阵

实际值	预测值	
	正样本 (P)	负样本 (N)
正样本 (P)	TP	FN
负样本 (N)	FP	TN

代码克隆检测问题是二分类问题，因此本文采用精确率（Precision）、召回率（Recall）、F1 值

为了对实验结果进行评价和分析，本文将用 ACC、Recall、F1-measure 和 MCC 四个指标来评估实验结果。其中使用了混淆矩阵中的 TP、FN、FP、TN（如表3.4）。

ACC：准确率是针对代码分类的预测结果而言，是分类任务最常见的评价指标。通常来说，准确率越高，分类器越好；ACC = TP + TN P + N （16）Recall：召回率表示的是样本中的正例中被预测正确的概率。主要由两部分组成，一部分是是把正类预测成正类（TP），另一部分就是把正类预测为负类（FN）。Recall = TP TP + FN （17）

采用召回率 (Recall)、精确度 (Precision) 和准确率（Accuracy）三个指标评价实验结果

3.3.2　预训练辅助模型消融实验结果

消融对比实验：体现改进的辅助模型的有效性，如图3.5 基于 Token 的 Bi LSTM 基于 Token 的 + 预训练辅助模型的 Bi LSTM

表 3.5　预训练辅助模型实验结果

对比	P	R	F1
基于 Token 的 Bi LSTM	0.xx	0.xx	0.xx
基于 Token 的 + 预训练辅助模型的 Bi LSTM	0.xx	0.xx	0.xx

3.4 本章小结

本章

第 4 章　基于子树划分的抽象语法树表征学习

本章主要对本文提出的基于子树划分的抽象语法树表征学习方法进行详细介绍，首先介绍其基本思想，其次阐述其具体方法设计与实现，最后进行实验验证。

4.1　研究动机

本文针对现有存在的问题，

抽象语法树 AST 是源代码语法结构的一种抽象表现形式，以树的形式包含了源代码中的语法信息和语法结构。利用深度神经网络对抽象语法树进行建模得到其向量表示，根据该特征向量完成代码克隆检测任务，实现基于树的源代码表征。现有的方法主要分为两类：一类是将 AST 转换为完整的二叉树，另一类是将 AST 直接视作二叉树，这两种方法都会或多或少破坏源代码原有的语法结构，而且在转换的过程会增加 AST 的高度，AST 树过高会导致梯度消失问题，可能会丢失长期上下文信息，削弱了神经模型捕捉更真实和复杂语义的能力。

4.2　方法设计

本节将介绍基于子树划分的抽象语法树表征学习方法设计与实现，

4.2.1　框架概述

基于子树划分的抽象语法树表征学习：

4.2.2　子树划分

针对上述问题，本文将每个大型的 AST 分割成小语句树序列，并通过捕获语句的词法和句法知识将每一个语句树都编码成一个向量。

4.2.3　抽象语法树表征学习

在得到一个语句向量序列后，将语句向量序列输入 Tree LSTM 网络中生成代码片段的结构向量表示。与标准 LSTM 结构类似，Tree-LSTM 神经网络中每个神经元都

表 4.1　抽象语法树子树划分实验结果

对比	P	R	F1
基于 AST 的 Tree-LSTM	0.xx	0.xx	0.xx
基于 AST 的 + 子树划分的 Tree-LSTM	0.xx	0.xx	0.xx

包括类似的输入门，输出门和隐层输出。不同的是 Tree-LSTM 单元中门向量和神经元状态的更新依赖于所有与之相关的子单元的状态，另外，Tree-LSTM 拥有多个遗忘门，分别对应当前节点的每个子节点，因此 Tree-LSTM 可以选择性地从子节点中获取信息，从而保存语义信息更加丰富的子节点的信息。通过 Tree LSTM 模型，本文能够学习到基于抽象语法树的结构向量，通过这种连续向量表现基于树的理解认知层次，获取程序的结构信息，创建更高抽象层次上的表示，从而提高后续代码克隆检测任务的精度。

4.3　实验验证

为了验证基于子树划分的抽象语法树表征学习方法的有效性，本文

4.3.1　实验设计

和 3.3.1 相同

4.3.2　抽象语法树子树划分消融实验结果

消融对比实验：体现 AST 子树划分的有效性

基于 AST 的 Tree-LSTM

基于 AST 的 + 子树划分的 Tree-LSTM

4.4　本章小结

本章

第 5 章　基于图过滤的程序依赖图表征学习

本章主要对本文提出的基于图过滤的程序依赖图表征学习方法进行详细介绍，首先介绍其基本思想，其次阐述其具体方法设计与实现，最后进行实验验证。

5.1　研究动机

本文针对现有存在的问题，

程序依赖图 PDG 是程序的一种图形表示，所含结构信息最多，能够表示程序的控制依赖，数据依赖以及地址依赖等关系，是一种带有标记的有向多重图。程序依赖图 PDG 结点代表语句，边代表依赖关系，依赖关系包括数据依赖和控制依赖。通过将程序表示为图的形式使得模型能够更好地理解代码中不同部分之间的依赖关系。现有的方法大多通过图匹配的方法，将 PDG 图中的控制流和数据流编码为一个紧凑的语义特征矩阵，其中每个元素都是一个高维的稀疏二值特征向量，通过寻找矩阵之间的相似模式来判定克隆代码。但这些方法通常需要消耗大量的时间和空间，计算开销大。

5.2　方法设计

本节将介绍基于图过滤的程序依赖图表征学习方法设计与实现，

5.2.1　框架概述

基于图过滤的程序依赖图表征学习：

5.2.2　图过滤机制

针对上述问题，本课题引入过滤机制，通过收集 PDG 的简单特征来过滤掉明显不可能为克隆的 PDG 对。具体的，根据 PDG 的节点个数、控制边数、执行边数、数据边数、声明节点数、函数调用数、传入参数、传出参数等代表特征进行过滤，在大幅减少候选 PDG 对规模的同时，保证真正的克隆对不会被过滤掉而导致整体克隆检出率的降低。

表 5.1　图过滤机制实验结果

对比	P	R	F1
基于 PDG 的 GCN	0.xx	0.xx	0.xx
基于 PDG 的 + 图过滤的 GCN	0.xx	0.xx	0.xx

5.2.3　程序依赖图表征学习

对于提取到的程序依赖图，本课题拟通过图卷积神经网络将其转换成向量。图卷积神经网络是一种特殊的前馈神经网络结构，为减少网络中参数个数，用卷积层来代替传统的全连接层，提高神经网络的训练效率，卷积神经网络可以提取信息最多的数据特征，生成一个固定大小的向量表示结构，从而挖掘深层次的语法和语义信息，在代码克隆检测任务中有较好的性能表现。

5.3　实验验证

为了验证基于图过滤的程序依赖图表征学习方法的有效性，本文

5.3.1　实验设计

和 3.3.1 相同

5.3.2　图过滤机制消融实验结果

消融对比实验：体现图过滤机制的有效性

基于 PDG 的 GCN

基于 PDG 的 + 图过滤的 GCN

5.4　本章小结

本章

第 6 章　特征融合及 RLCCD 框架验证

本章主要对面向代码克隆检测的多维源代码表征方法整体框架 RLCCD 进行介绍，同时进行实验评估及验证。具体地，RLCCD 是由上述基于预训练辅助模型的 Token 表征学习、基于子树划分的抽象语法树表征学习、基于图过滤的程序依赖图表征学习方法三种维度融合形成并进行实现的表征方法，最后通过与 SourcererCC、ASTNN、SCDetector 进行对比实验以验证该框架的有效性。

6.1　特征融合

特征融合

6.2　RLCCD 框架验证

本文的实验设计主要围绕以下 4 个方面的研究问题：

• RQ1：本文提出的预训练辅助模型策略是否优于基线方法？（见 3.3 节）

• RQ2：本文提出的子树划分策略是否优于基线方法？（见 4.3 节）

• RQ3：本文提出的图过滤策略能否优于基线方法？（见 5.3 节）

• RQ4：与现有代码克隆检测工具相比，RLCCD 表现如何？

6.2.1　实验设置

（1）系统环境

本章实验均在 Ubuntu 16.04 LTS（64 位）系统下进行。

（2）实验数据集

本章实验部分采用的数据集为代码克隆检测领域常见的基准集 POJ104，该数据集是一个基于 C 语言构建的大型数据集，包含了 104 个编程问题以及学生提交的对应问题的不同 C 语言源代码，在该数据集中，针对同一问题的不同源解法的代码被视为一个克隆对。

首先通过筛选，得到 51485 个源代码样本，然后随机生成 50000 个代码克隆对，其中包含 5200 个真克隆对，44800 个假克隆对。依据随机种子将数据集按照 3：1：1

表 6.1　RLCCD 实验结果

对比	P	R	F1
SourcererCC	0.xx	0.xx	0.xx
ASTNN	0.xx	0.xx	0.xx
SCDetector	0.xx	0.xx	0.xx
RLCDD	0.xx	0.xx	0.xx

划分为训练集、测试机、验证集，其中的正负样本数如下表。

6.2.2　对比工具

在对比整个方法的效果时，本文选取开源的 SourcererCC、ASTNN、SCDetector 方法进行比较。

SourcererCC：SourcererCC 是一种相对较新的基于 token 的克隆检测工具。该工具通过词袋模型，把收集的数据全部编码成词频信息，后将代码行转换成一个由词频构成的向量，通过向量的比较获取相似度。

ASTNN：一种基于神经网络的源代码表示方法。它将整个抽象语法树 AST 分解成一系列小型语句子树，并通过捕获语句的词法和语法信息将语句子树分别编码为向量，最后采用了 RNN 模型生成代码片段的向量表示。ASTNN 方法完整保留了抽象语法树的结构信息，能够检测到所有类型的代码克隆。

SCDetector：是基于令牌和基于图的方法的结合。给定一个方法源代码，我们首先生成 CFG，然后应用中心性分析将图转换为某些语义标记（即具有图细节的标记）。最后，这些语义标记被馈送到 Siamese 网络中，以训练模型并使用它来检测代码克隆对。

6.2.3　RLCCD 性能评估实验结果

对比实验：体现框架的有效性

6.3　本章小结

本章节主要对 RLCCD 框架进行了实验验证，以验证 RLCCD 的 XXX 能力。

结论

尽管模糊测试在漏洞挖掘方面取得了较好的成果，但其仍存在一定的随机性和盲目性，受限于生成种子的效率和质量。近年来，研究人员从模糊测试与智能化技术相结合这一方面进行了探索，试图从关键技术点入手，找到合适的结合点，以提高模糊测试技术的效率和智能化程度。

参考文献

[1] 乐乔艺, 刘建勋, 孙晓平, et al. 代码克隆检测研究进展综述 [J]. 计算机科学, 2021, 48 (S02): 14.

[2] Synopsys, Inc. 2023 Open Source Security and Risk Analysis Report. 2023. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html.

[3] Moore S. 7 Top Trends in Cybersecurity for 2022. April 13, 2022. https://www.gartner.com/en/articles/7-top-trends-in-cybersecurity-for-2022.

[4] Svajlenko J, Roy C K. Evaluating clone detection tools with BigCloneBench [C]. In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2015: 131–140.

[5] IJaDataset20. Ambient Software Evoluton Group. January 2013. http://secold.org/projects/seclone.

[6] Dang Y, Zhang D, Ge S, et al. Transferring Code-Clone Detection and Analysis to Practice [C]. In 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017: 53–62.

[7] Yang J, Hotta K, Higo Y, et al. Classification model for code clones based on machine learning [J/OL]. Empirical Software Engineering, 2015, 20: 1095–1125. https://api.semanticscholar.org/CorpusID:18892467.

[8] 陈秋远, 李善平, 鄢萌, et al. 代码克隆检测研究进展 [J]. 软件学报, 2019, 30 (4): 19.

[9] 谢春丽 □, 梁瑶. 深度学习在代码表征中的应用综述 [J]. 计算机工程与应用, 2021, 57 (20): 53–63.

[10] Pearson K. On lines and planes of closest fit to systems of points in space. [J]. PHILOSOPHICAL MAGAZINE, 1901, 2 (7-12): 559–572.

[11] Fisher R A. THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS [J]. Annals of Human Genetics, 2012, 7 (7): 179–188.

[12] Rumelhart D E, Hinton G E, Williams R J. Learning Representations by Back Propagating Errors [J]. Nature, 1986, 323 (6088): 533–536.

[13] Hinton G E, Osindero S, Teh Y W. A Fast Learning Algorithm for Deep Belief Nets [J]. Neural Computation, 2006, 18 (7): 1527–1554.

[14] Bengio, Yoshua, Courville, et al. Representation Learning: A Review and New Perspectives [J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2013, 35 (8): 1798–1828.

[15] Goodfellow I, Bengio Y, Courville A. Deep Learning [M/OL]. MIT Press, 2016. https://books.google.com.tw/books?id=Np9SDQAAQBAJ.

[16] Kamiya T, Kusumoto S, Inoue K. CCFinder: a multilinguistic token-based code clone detection system for large scale source code [J]. IEEE Transactions on Software Engineering, 2002, 28 (7): 654–670.

[17] Li Z, Lu S, Myagmar S, et al. CP-Miner: finding copy-paste and related bugs in large-scale software code [J]. IEEE Transactions on Software Engineering, 2006, 32 (3): 176–192.

[18] Jiang L, Su Z, Chiu E. Context-based detection of clone-related bugs [C/OL]. New York, NY, USA, 2007: 55–64. https://doi.org/10.1145/1287624.1287634.

[19] Pennington J, Socher R, Manning C. GloVe: Global Vectors for Word Representation [C/OL] // Moschitti A, Pang B, Daelemans W. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, October 2014: 1532–1543. https://aclanthology.org/D14-1162.

[20] Devlin J, Chang M-W, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [C/OL] // Burstein J, Doran C, Solorio T. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, June 2019: 4171–4186. https://aclanthology.org/N19-1423.

[21] Feng Z, Guo D, Tang D, et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages [J/OL]. ArXiv, 2020, abs/2002.08155. https://api.semanticscholar.org/CorpusID:211171605.

[22] White M, Tufano M, Vendome C, et al. Deep learning code fragments for code clone detection [J/OL]. 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), 2016: 87–98. https://api.semanticscholar.org/CorpusID:14867364.

[23] Mou L, Li G, Zhang L, et al. Convolutional Neural Networks over Tree Structures for Programming Language Processing [C]. In THIRTIETH AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2016: 1287–1293. 30th Association-for-the-Advancement-of-Artificial-Intelligence (AAAI) Conference on Artificial Intelligence, Phoenix, AZ, FEB 12-17, 2016.

[24] Wei H-H, Li M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code [C]. 2017: 3034–3040.

[25] Chen L, Ye W, Zhang S. Capturing source code semantics via tree-based convolution over API-enhanced AST [C/OL]. In Proceedings of the 16th ACM International Conference on Computing Frontiers, New York, NY, USA, 2019: 174 – 182. https://doi.org/10.1145/3310273.

3321560.

[26] Zhang J, Wang X, Zhang H, et al. A Novel Neural Source Code Representation Based on Abstract Syntax Tree [C]. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019: 783–794.

[27] Allamanis M, Brockschmidt M, Khademi M. Learning to Represent Programs with Graphs [J/OL]. ArXiv, 2017, abs/1711.00740. https://api.semanticscholar.org/CorpusID:3495200.

[28] Lu M, Tan D, Xiong N N, et al. Program Classification Using Gated Graph Attention Neural Network for Online Programming Service [J/OL]. ArXiv, 2019, abs/1903.03804. https://api.semanticscholar.org/CorpusID:73728621.

[29] Brockschmidt M, Allamanis M, Gaunt A L, et al. Generative Code Modeling with Graphs [J/OL]. ArXiv, 2018, abs/1805.08490. https://api.semanticscholar.org/CorpusID:46899514.

[30] Ben-Nun T, Jakobovits A S, Hoefler T. Neural code comprehension: a learnable representation of code semantics [C]. In Proceedings of the 32nd International Conference on Neural Information Processing Systems, Red Hook, NY, USA, 2018: 3589–3601.

[31] Wang W, Li G, Ma B, et al. Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree [C]. In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020: 261–271.

[32] DeFreez D, Thakur A V, Rubio-González C. Path-based function embedding and its application to error-handling specification mining [C/OL]. New York, NY, USA, 2018: 423–433. https://doi.org/10.1145/3236024.3236059.

[33] Duan Y, Li X, Wang J, et al. DEEPBINDIFF: Learning Program-Wide Code Representations for Binary Diffing [C]. In 27TH ANNUAL NETWORK AND DISTRIBUTED SYSTEM SECURITY SYMPOSIUM (NDSS 2020), 2020. 27th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, FEB 23-26, 2020.

[34] Yang K, Yu H, Fan G, et al. A graph sequence neural architecture for code completion with semantic structure features [J/OL]. Journal of Software: Evolution and Process, 2021, 34. https://api.semanticscholar.org/CorpusID:245628361.

[35] Tufano M, Watson C, Bavota G, et al. Deep Learning Similarities from Different Representations of Source Code [J/OL]. 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR), 2018: 542–553. https://api.semanticscholar.org/CorpusID:46902434.

[36] Fang C, Liu Z, Shi Y, et al. Functional code clone detection with syntax and semantics fusion learning [J/OL]. Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2020. https://api.semanticscholar.org/CorpusID:263876479.

[37] Hua W, Sui Y, Wan Y, et al. FCCA: Hybrid Code Representation for Functional Clone Detection Using Attention Networks [J/OL]. IEEE Transactions on Reliability, 2020, 70: 304–318. https://api.semanticscholar.org/CorpusID:226421066.

[38] Dong W, Feng Z, Wei H, et al. A Novel Code Stylometry-based Code Clone Detection Strategy [C]. In 2020 International Wireless Communications and Mobile Computing (IWCMC), 2020: 1516–1521.

[39] 王亚芳. 基于图像相似度检测代码克隆 [D]. [S. l.]: 内蒙古师范大学, 2020.

[40] Saini V, Farmahinifarahani F, Lu Y, et al. Oreo: detection of clones in the twilight zone [C/OL]. New York, NY, USA, 2018: 354–365. https://doi.org/10.1145/3236024.3236026.

[41] Alomari H W, Stephan M. Clone Detection through srcClone: A Program Slicing Based Approach [J]. Journal of Systems and Software, 2022, 184: 111115.

[42] Sajnani H, Saini V, Svajlenko J, et al. SourcererCC: Scaling Code Clone Detection to Big-Code [C]. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016: 1157–1168.

[43] Wu Y, Zou D, Dou S, et al. SCDetector: software functional clone detection based on semantic tokens analysis [C/OL]. New York, NY, USA, 2021: 821–833. https://doi.org/10.1145/3324884.3416562.

攻读学位期间发表论文与研究成果清单

[1] 高凌. 交联型与线形水性聚氨酯的形状记忆性能比较 [J]. 化工进展, 2006, 532 一 535.（核心期刊）

致谢

本论文的工作是在导师……。