计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210300310

代码克隆检测研究进展综述

乐乔艺　刘建勋　孙晓平　张祥平
湖南科技大学计算机工程与技术学院　湖南 湘潭 411100
(lqy1193278522@163.com)

摘　要　软件系统中两个或两个以上的相似代码片段被称为代码克隆(code clone)。有研究表明,代码克隆在软件系统中大量存在,并且随着时间推移不断增长。随着代码开源成为潮流,代码克隆占比越来越高。已有研究工作发现软件系统中的代码克隆是有害的,会导致系统稳定性降低,造成代码库冗余和软件缺陷传播等问题。为了提高代码质量,目前学术界和工业界已经提出了多种代码克隆检测方法,按照获取代码的信息程度不同分为基于文本、词法、语法、语义、和度量值5种方法,不同的方法具有不同的性能和应用场景。文中分析了软件克隆出现的原因及优缺点,对软件系统中的代码克隆问题进行了分类,评价了5种不同类型检测方法各自的优势,详细介绍了部分方法的核心思想、检测语言、验证所用数据集及检测效果等技术特征。文章最后总结了克隆检测技术所适用的不同应用场景,对代码克隆检测方法和应用的发展方向做出了展望。

关键词:代码克隆;克隆检测;克隆分析
中图法分类号　TP312

Survey of Research Progress of Code Clone Detection

LE Qiao-yi,LIU Jian-xun,SUN Xiao-ping and ZHANG Xiang-ping
School of Computer Engineering and Science,Hunan University of Science & Technology,Xiangtan,Hunan 411100,China

Abstract　In a software system,if there are two or more similar pieces of code,it is called code clone. Studies have shown that code cloning exists in a large number of software systems and is increasing with the passage of time. In the era of big data,open source code has been a trend,so the proportion of code cloning will be higher and higher. The existing related work thinks that the code cloning in software system is harmful,which will lead to the decrease of system stability,the redundancy of code base and the spread of software defects. This paper analyzes the reasons,advantages and disadvantages of software cloning,and classifies the code cloning in software system. In order to improve the code quality,many code cloning detection methods have been proposed in the academic and industrial circles. According to the degree of information obtained from the code,these methods can be divided into five types based on text,lexis,syntax,semantics,and metric. This paper analyzes and evaluates these five kinds of detection methods,and finds that all kinds of methods have no absolute advantages,and different methods can be applied to different fields according to different requirements. In the end,the paper summarizes the different application scenarios of the cloning detection technology,and forecasts the development direction of the code cloning detection methods and applications.

Keywords　Code clone,Code detection,Clone analysis

1　引言

在软件开发中,复制现有的代码片段并将它们粘贴到代码的其他部分中是一种常见的行为,是代码重用的一种体现[1-11],复制的代码称为代码克隆(code clone)[3]。事实上软件开发人员经常故意进行代码克隆,因为代码克隆具有一些潜在的好处[12]。Kapser等[12]在一个真实的电信系统上进行研究,发现多达71%的代码克隆其实对软件系统可维护性具有积极的影响。通过修改现有代码以满足新的需求,是一种简便的软件开发策略,在很多开源系统中得到实践。Cordy[13]提到,金融机构的软件系统可能更需要使用克隆技术,因为金融机构认为保证代码质量是维护软件时最重要的问题,软件错误的损失会使软件维护成本相形见绌。克隆代码是金融机构使用的一种风险最小化的方法,允许对代码进行单独维护和修改,一是可以利用已有的稳定代码,二是可以隔离问题代码。除此之外,由于代码克隆在软件开发中的高效率,急需上市或者面临市场竞争压力的公司往往认为时间的重要程度远远大于维护源代码的长期成本,也会选择代码克隆作为常用的开发手段[4]。

研究显示,一般软件系统中大约存在7%～24%的克隆代码,一些软件甚至达到50%[1]。即便是X Windows System等计算机领域公认的高质量软件系统仍存在19%的克隆代码,Linux内核源代码也存在15%～25%的克隆代码。

但是,过多的代码克隆将会对软件项目带来不利影响。已有研究[14]表明,代码克隆是一种不良做法(Bad Smell)。随着软件系统不断演化开发,未得到良好管理的软件系统会因

基金项目:国家自然科学基金(61872139)
This work was supported by the National Natural Science Foundation of China(61872139).
通信作者:刘建勋(ljx529@gmail.com)

为代码克隆造成代码库不断膨胀，逐渐增加软件系统维护成本[15]。除此之外，软件系统中的软件缺陷也会随着代码克隆蔓延至软件系统各处，导致软件系统中 Bug 蔓延。

研究[12]指出，在采取处理软件系统中代码克隆行动之前，应该先了解其代码克隆产生的原因。Baxter[16]认为成熟代码库中任何数量的克隆都可以用一个副本来替代，他们用宏替换克隆来重新构造系统，减少源代码的数量和方便维护。Rajapake 等[17]发现减少 Web 应用程序中的克隆不仅会对应用程序的可修改性产生负面影响，在初始开发阶段避免克隆还会导致大量开销。Mayrand 等[18]根据移除成本将代码克隆分为 8 个等级，研究指出移除克隆可能是一种昂贵的活动，不同等级的代码克隆应当采用不同的处理方式。Lague 等[19]在一个大型电信系统上评估了代码克隆删除与代码克隆避免的好处与不足。以上研究表明，应当采取不同的方式面对不同类型的代码克隆。

回顾代码克隆检测研究近 20 年的历史，我们可以将代码克隆检测问题简单地划分为两个子问题。第一个问题是如何找到代码克隆，第二个问题是找到代码克隆后该如何处理。第一个问题即代码克隆的检测研究，研究设计代码克隆检测方法找到软件系统中存在的不同类型的代码克隆。第二个问题即代码克隆的定性问题和应用问题，判断“好的”代码克隆、“坏的”代码克隆以及”需要观察的”代码克隆之间有何区别，如何界定，以及面对不同的代码克隆应该采用何种代码克隆检测方式进行检测。

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[20]，利用文本相似性判断代码的相似程度，这种方式也被称为基于文本的代码克隆检测技术。它通常直接在源代码上进行相似性判断，由于不用对代码进行转化等操作，因此具有较快的检测速度。基于文本的检测技术对于简单的复制粘贴行为具有良好的检测效果，但是对于复杂的修改行为，检测效果一般甚至完全检测不出。

随着编译技术的发展，研究者们开始将编译原理中的词法分析技术运用于代码克隆检测中。基于词法分析的检测方法[21]首先将源代码转换为词法单元序列，然后判断代码相似性。与基于文本的代码克隆检测技术相比，基于词法分析的代码克隆检测技术对格式变化、标识符重命名等产生的克隆代码具有更好的检测效果。该方法本质上还是将代码视为自然语言文本来进行处理，没有借助代码本身所特有的结构信息来提升检测效果。

由于基于词法分析的克隆代码检测技术无法保证检测结果的语法完整性，因此检测到的克隆代码片段可能会覆盖多个不完整的语法单元。针对这一问题，研究者尝试将语法分析运用到克隆检测过程中[22]。基于语法分析的检测方法使用与编译器类似的解析器前端将源程序转换为解析树或抽象语法树(Abstract Syntax Tree，AST)，然后基于抽象语法树进行克隆检测。

基于语义分析的克隆代码检测方法[23]通过静态程序分析技术抽取语义关系图结构，通过图相似度匹配来进一步理解源代码语义之间的相似性，因此相较于基于文本、词法分析和语法分析的检测技术能够挖掘更深层次的克隆代码。但是基于图的方法需要程序依赖图生成器，不同语言需要不同的生成器，同时图的相似性计算时间复杂度较高。

以上的方法都有着各自的优缺点，按照需求的不同，不同类型的方法可以检测相应的代码克隆。

本文采用以下流程完成对相关文献的获取，从而对代码克隆检测的研究现状进行综述。本综述调研了在公开的期刊及会议论文集、出版书籍中发表的代码克隆检测研究中提出的新模型、新算法，以及代码克隆检测在不同领域的应用技术。本文根据以上原则在以下文献库中进行检索

(1)本综述选用 ACM 电子文献数据库、IEEE 电子文献数据库、Springer Link 电子文献数据库及 Google 学术搜索引擎进行原始搜索。论文检索的关键词包括 code clone、clone detection、code similarity 等。同时在标题、摘要、关键词和索引中进行检索。

(2)本综述依据 CCF 推荐国际学术会议和期刊进行软件工程相关文献检索，相关会议及期刊包括 ICSE，TSE，ASE，FSC/ESEC 等。

(3)为避免遗漏相关研究，在之前两步的基础上，在每篇文献的参考列表中寻找与代码克隆检测相关的研究文献，并添加到相关文献。

基于上述选取原则和检索步骤，本文共选取 101 篇文献作为综述总结的相关参考文献。在这些文献中，提出代码克隆检测方法的新算法、新工具的直接相关文献有 62 篇，代码克隆领域综述文章有 6 篇，代码克隆检测方法应用于不同领域中的相关文章有 19 篇，其他为代码克隆的背景动机、可行性及应用场景提供支持的部分相关文献，有 14 篇。

本文的主要贡献可以总结为如下 3 点：

(1)本文通过对软件开发过程中不同类型代码克隆产生的原因、优缺点以及改进手段进行了研究分析，发现部分代码克隆是不可避免而且必要的，根据代码特征的不同，代码克隆的处理方式应当进行相应的调整。

(2)本文对不同的克隆检测方法进行了分析和评价，发现各类方法都有各自的优点，不同的方法按照不同的需求可以应用于不同的软件工程任务，如程序理解、剽窃检测、侵权调查、压缩代码、Bug 检测等。

(3)本文对现有的代码克隆检测研究进行了总结分析，对目前的代码克隆研究领域的发展趋势进行展望。

2 代码克隆及检测的背景和概念

2.1 基础定义

本文沿用了在以往代码克隆综述中的定义[1-11]。

代码片段：源代码的连续片段，$ps=(l,s,e)$。其中 l 为源文件，s 为代码片段开始的行，e 为代码片段结束的行。

克隆对：一对相似的代码片段，由$(f_1,f_2,\emptyset)$指定，包括相似的代码片段 f_1 和 f_2，以及它们的克隆类型 $\emptyset$。

克隆类：一组相似的代码片段。由元组指定$(f_1,f_2,\cdots,f_n,\emptyset)$，其中每对不同片段都是克隆对$(f_i,f_j,\emptyset)$，$i,j\in 1,\cdots,n,i\neq j$。

2.2 代码克隆背景介绍

软件开发过程中代码克隆是很常见的活动。不同于简单的复制粘贴，软件系统开发中的代码克隆往往有明确的目的，例如代码重构、防止逆向工程生成的模糊化代码、错误修复，甚至是软件剽窃。这些行为或多或少地影响软件系统性能[24]，主要体现在：1)如果在代码片段中检测到一个 Bug，应

该检查所有与之相似的片段是否存在相同的 Bug[25]；2）如果由于功能需求的更新导致新的代码片段增加了某些功能，为了优化程序，应该找到其原始代码并且通过重构或者覆盖等方式对原始代码进行处理；3）程序员在代码克隆时，没有考虑源自外部的代码是否存在安全漏洞、知识产权问题，这些代码随着项目的演化变得越来越难维护。因此，如何对软件系统中存在的代码克隆现象进行有效检测，成为了软件工程领域的一个重要问题。并且随着软件应用广泛应用于社会的各个生产生活领域，这个问题的重要性日趋明显。

由于目的不同，产生的代码克隆往往具有不同的特点，这些不同类型的代码克隆往往对原代码有着不同程度的修改，而这些代码修改往往会在一定程度上影响代码克隆检测工具的性能。因此，在指定的应用场景下，选对代码克隆检测方式是十分重要的。

2.3 代码克隆的原因

尽管从维护的角度看，代码克隆是一种不好的做法，但是程序开发人员经常在软件系统开发过程中使用代码克隆。本文列出了代码克隆在软件系统中大量出现的部分原因。

（1）代码克隆是一种节约时间和成本的开发模式：代码程序开发人员复制代码，很大程度上是因为他们知道已有的代码可以提供部分或是全部功能。在时间和程序员自身能力的限制下，代码克隆成为了唯一的、最好的出路[1]。

（2）软件系统缺乏维护：程序员通常会因为时间限制而将代码重构推迟到项目完成之后，随着时间的推移，存在大量新增的代码克隆，代码变得越来越难理解，使得代码克隆越来越难以移除、重构，增加了软件系统的维护成本[4]。

（3）代码的独立演化：分支是一种类似于重用的解决方法。为了满足不断变化的需求，降低对现有代码进行更改的难度，在短期内，软件开发人员希望代码能够独立地演化，用以减少代码更改对现有系统的影响。在分支结束之前，代码克隆会一直存在[12]。

（4）编程范例鼓励在编程中使用模板：结构模板和功能模板的使用也通常被认为是一种重用机制。由于编程语言的限制，程序开发人员通常很容易使用已编写好的模板，其目的是通过重用受信任的解决方案来防止出现错误[12]。

（5）Bug 避免：程序开发人员通常尽量不在现有代码中引入新的代码。因为引入新代码有可能会导致软件开发生命周期过长。此外重用现有代码比开发新的解决方案更容易，因为新代码可能会引入新的错误[27]。

（6）程序员记忆的副作用：程序员可能会重复一种常见的解决方案，不自觉地将克隆引入到软件系统内[1]。

2.4 代码克隆的优缺点

2.4.1 代码克隆的优点

（1）代码质量高。对于一个初学者而言，直接通过复制资深、专业程序员的代码来学习代码的标准编写格式是一种十分高效、常见的学习方法。已有成功开源项目一般为经过大量的程序员开发与修补后的状态，因此具有很好的借鉴意义[2]，其中克隆代码也是一种常见的程序开发方法。

（2）降低开发难度。同一个代码软件中，代码重用会提高代码效率，增强代码的可靠性以及一致性。在理想的状态下，一个项目由已有的、可重用的代码经过修改、重构、完成功能需求等步骤，能够将开发难度降到最低[5]。

（3）开发成本低。在程序开发过程中，需求分析、系统与详细设计、编码与测试等阶段都需要大量的人力、物力及财力。代码克隆后，可以提高开发效率，大大降低开发成本[4]。

2.4.2 代码克隆的缺点

（1）维护困难：如果在一个软件系统中有许多相同或几乎重复或复制粘贴的代码片段，一旦在一个代码克隆中发现了一个 Bug，则必须在所有地方检测并修复它。如果一个软件中许多位置都存在重复的 Bug，则增加了软件的维护难度[1]。

（2）Bug 产生：在程序开发人员进行复制粘贴（代码克隆）行为时，如果忘记对克隆的代码进行一致性修改，软件系统中可能就会引入新的 Bug。已有研究显示，代码克隆和软件中的 Bug 和不一致性直接相关[28]。

（3）Bug 传播：如果一个代码片段中含有 Bug，并且该 Bug 通过代码克隆粘贴在了不同的位置，那么相同的 Bug 将会出现在代码的各个位置，增加了代码克隆 Bug 传播的可能性[25]。

（4）版权保护以及剽窃问题：在大量代码开源的今天，代码克隆时，应该注意版权保护问题，遵守开源许可协议（GPL，COPYLEFT，Apache License 等）。如果在许可协议外克隆代码，则会导致剽窃问题[29]。

2.5 代码克隆的分类

按照克隆的程度可以将代码克隆分为以下 4 类。其中前 3 种类型都是基于文本的相似代码，具体代码以及其克隆分类如表 1 所列。第 4 种类型则是基于功能性的相似代码[2]。

表 1 代码克隆类型

Table 1 Type of code clone

原始代码	代码片段 1	代码片段 2	代码片段 3	代码片段 4

if (a==b)
{
c=a * b;
}
else{
c=a/b
};
	if (a==b)
	{
	//comment1
	c=a * b;
	}
	else{
	//comment2
	c=a/b
	};
		if (g==f)
		{
		//comment1
		h=g * f;
		}
		else{
		//comment2
		h=g/f;
		}
			if (a==b)
			{
			//comment1
			c=a * b;
			// New Stat.
			b=a−c;
			}
			else{
			//comment2
			c=a/b;
			}
				switch(true)
				{
				//comment1
				case a==b;
				c=a * b;
				//comment2
				case a! =b;
				c=a/b;
				}

	Type-1	Type-2	Type-3	Type-4

Type-1（完全克隆）：两个代码之间完全相同（除注释与空白之外）。

Type-2（重命名克隆）：除了变量、类型、文字和函数的名称进行了改动之外，两个代码片段相同。

Type-3（增删改克隆）：两个代码片段相似，在 Type-2 克隆的基础上，有一些语句添加、删除或修改，以及代码布局的修改。

Type-4（自实现克隆）：两个代码片段实现的功能相同，但是实现方式不同。

在软件系统的层面，我们将参照 Kapser[12] 的研究对在程序开发中的代码克隆进行更进一步的划分。

完全克隆：当某个特定的问题在软件系统中重复出现时，往往会出现完全匹配，但是该问题太小以至于无法建立有价值的抽象。横切关注点是一个典型的例子，在这种情况下，如果对克隆所依赖的模块设计进行更改，将对所有与它相关的克隆产生影响，因此程序的可维护性将受到负面影响。

独立型克隆：在程序开发过程中，被抽象出来的代码可能很难被修改。但是为了保持兼容性，我们期望新代码的演化将在某种程度上独立于原始代码，并且能随着操作系统的发展不断修改。在这种情况下，对新代码的更改对原始代码并没有任何影响。当新代码成熟时，对剩余的重复代码进行重构将是合理有效的。

模板化克隆：在需求明了，已有的代码可以实现所有需求时，常常会引入这种克隆。这种克隆产生的代码往往会和原始代码有着紧密的联系，由于代码量的增加，可维护性下降，而且所有副本的更改应当保持一致。

功能相似性克隆：当现有代码已经无法充分满足一组新的需求时，就会发生定制。为了系统的稳定性，现有代码不能被修改，因此将引入克隆。与前几种克隆不同，这种情况下的克隆往往是功能级相似的克隆，更难被发现。

2.6 代码克隆检测的研究简介

大量学者对软件开发过程研究之后发现，代码克隆在计算机程序中很多时候是有害。此后代码克隆检测技术便被应用于软件工程领域，并且取得了不错的结果。事实上，关于代码克隆是否应该被移除的问题，学术界并没有达成一致[12]。而对于不同的应用场景，判断两段代码是否为克隆的指标也并不相同。当前的困难在于自动检测工具并不能判断哪种克隆是值得删除的，哪些克隆是应该留下的。

在近 20 年内，代码克隆检测受到了学术界和工业界的广泛关注和重视，取得了很多进展。在代码克隆检测的研究领域中，研究者的主要目的是检测软件系统中存在的代码克隆，并且生成检测报告，以帮助软件开发人员对代码进行优化。按照其方法的代码中间表示不同，我们可以将这些方法分为基于文本、基于词法、基于语法、基于语义和基于度量值 5 种[4]，我们将在 3.5 节对这些方法进行归纳总结。

3 代码克隆检测方法

3.1 代码克隆检测方法的特征

在众多代码克隆检测方法中，有一些特征可以用于区分、描述不同的代码克隆检测方法。这些特征体现了克隆检测技术是如何工作的。我们将通过以下几个特征来区别、描述代码克隆检测方法。

(1)源代码规范化：在大多数方法中，预处理阶段会应用一些规范化操作来对代码进行处理(移除空白块、移除注释、对缩进进行标准化等)。一些方法可能会采用一些特殊的规范化方法[30]，Roy[31]在其提出的 Nicad 工具中提供了可供用户选择的“漂亮打印”处理，用于消除代码段之间的格式和布局差异，并且可以将语句的不同部分分成几行，这样就可以使用简单的行比较来找到语句的局部更改。

(2)源代码转换：在目前大多数的代码克隆检测方法中，除了部分克隆检测方法(基于文本的方法)直接将代码视作普通自然语言文本进行处理，其余方法都需要通过某种方法将源代码转换为特定的形式，如词法单元、抽象语法树、控制流图、数据流图等。

(3)代码比较粒度：在对代码进行检测之前，需要将源代码划分为一定大小的代码单元，这些代码单元就被称为粒度，根据粒度大小的不同可以分为固定粒度(即在预定义的语法边界内，如方法和块)和自由粒度(即没有语法边界)。采用固定粒度方式，如果粒度太大，则检测漏检率会很大，反之，如果粒度太小，则检测工作量会较大。因此，为实现更好的检测效果，在实际应用中，常使用自由粒度的方式进行克隆检测。

(4)比较算法：信息检索领域中有很多字符串比较算法，如后缀树算法、哈希散列比较算法、动态模式匹配算法等可以用于检测克隆代码。

(5)计算复杂性：一个代码克隆检测方法被应用为工具的前提是必须具有可扩展性，能够检测大型软件系统中数百万行源代码中的克隆片段。也就是说，工具的适用性不光体现在精确率、召回率上，还体现在其计算效率上。

(6)能检测到的克隆类型：一些技术检测 Type-1 克隆，而另一些技术检测 Type-1，Type-2 或 Type-3 克隆，甚至可以检测所有类型的克隆。由于代码克隆的检测难度从 Type-1 到 Type-4 逐步增加，因此代码克隆检测方法能检测到的最高类型代码克隆是评价代码克隆检测方法有效性的重要指标。

(7)适用代码语言：对于一些工具而言，由于要使用特定的代码处理方式，因此只能适用于特定的代码语言。但是研究者们仍然在追求一种在任何系统上工作且独立于编程语言的工具。

(8)克隆报告：有些工具将克隆对作为克隆报告返回给使用者，而另一些工具则将克隆类作为克隆报告返回给使用者。克隆类比克隆对更适合软件维护。报告时不进行过滤的克隆类比过滤后返回的克隆对更好，因为克隆类的使用减少了比较次数和需要报告的克隆数据量。

3.2 检测流程

Whale[32]首次提出了检测过程的两个阶段：转换代码格式和确定相似度。目前，大部分代码检测工具都会执行相同的检测流程，广义代码克隆检测方法的具体流程如图 1 所示。

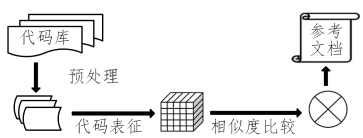

图 1 代码克隆检测流程

Fig. 1 Code clone detection process

预处理：此过程首先删除源代码中与比较过程无关的所有元素，将源代码标准化[31]。其次，它通过将源代码分成不同的片段(如类、函数、开始块或语句)将源代码转换为单元。这可以通过多种方式完成，包括词法或抽象语法树(AST)分析。这些单元用于检查是否存在直接的代码克隆关系。最后，这个过程将定义比较单位。例如，代码单元可以划分为令牌(token)比较。

代码转换：此过程将源代码转换为相应的中间表示(IR)以进行比较，从源代码中可以构造许多类型的表示。例如令牌流，其中每行源代码都转换为一个令牌序列。另一个常见的构造是 AST，在该结构中所有解析的源代码都被转换为抽象语法树或解析树，以便比较子树。除此之外，源代码中可以提取到程序依赖关系图(PDG)，用于表示控制和数据依赖关系。PDG 通常使用源代码中的语义感知技术比较子图。

匹配检测：此过程将每个转换后的代码片段与所有其他片段进行比较(比较粒度也可以被分为行、标记、抽象语法树(AST)的节点或程序依赖图(PDG)的节点)，以找到相似的源代码片段。

后处理：此过程将代码克隆检测结果整合并且输出一个参考文档，帮助代码开发人员进行程序开发[33]。常见的参考文档包含克隆对、克隆类等检测信息。

3.3 数据集与评测基准

早期的代码克隆检测方法通常是用自己的数据集进行评估，这些数据集缺乏统一性。为了解决这个问题，2007 年 Bellon 等[34]提出了第一批代码克隆检测基准，这个基准测试通过在两个较小的 C 程序和两个较小的 Java 程序上运行两个不同的代码克隆检测工具来收集。将这些结果与真实代码克隆的语料库进行对比，以创建代码克隆的数据集。Bellon 数据集在后续的工作中进行了很多次的扩展和优化。

2015 年，Svajlenko 等[35]提出了 BigcloneBench 评估框架，BigcloneBench 包含 800 万个 IjaDataset2.0 中经过验证克隆的代码。IjaDataset2.0 是一个包含 25000 个开源 Java 系统的大数据软件存储库。BigcloneBench 在项目内克隆与项目间克隆中都包含着所有 4 种类型克隆。

BigcloneBench 有两种克隆检测的方法[36]：

(1)直接作用于源代码库并报告克隆对，然后将报告的克隆对与标记的克隆对进行比较。该方法要求被评估的方法能够扩展到大规模的代码库，并且只能估计精度，因为可能会报告一些未标记的克隆对。

(2)直接作用于标记的克隆对，而未标记的函数被丢弃。这种方法简单，并且克隆检测器不需要考虑可伸缩性，对大型代码库进行了缩减。因此，该方法更适合于评估克隆检测器的分类能力，而不是可扩展性。

2016 年，Mou 等[37]公开了 OJClone 数据集，OJClone 包含 104 个编程问题以及学生为每个问题提交的不同 C 语言源代码。在 OJClone 中，解决同一编程问题的两个不同源代码被视为一个克隆对，因为它们实现了相同的功能，至少属于类型 3 克隆。

目前，大部分的检测工具覆盖率最高的语言仍然是 Java 和 C，造成这个结果的原因可能是常见的基准数据集只包含这两种语言代码。但是，Python，Javascript 等语言依然缺少公共基准测试集。

3.4 评价指标

在代码克隆检测研究中，不同检测方法针对不同问题，会有不同的评价指标，常见的评价指标包括以下几种。

准确率：所有预测正确(正类/负类)的比重。

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN} \tag{1}$$

精确率：正确预测为正的占全部预测为正的比例。

$$Precision=\frac{TP}{TP+FP} \tag{2}$$

召回率：正确预测为正的占全部实际为正的比例。

$$Recall=\frac{TP}{TP+FN} \tag{3}$$

F-Score($F1$)是 Precision 和 Recall 的加权调和平均，是 IR(信息检索)领域的常用的一个评价标准，用于评价分类模型的好坏。

$$F1=\frac{2TP}{2TP+FP+FN} \tag{4}$$

其中，TP(True Positive)是被模型预测为正的正样本数量；TN(True Negative)是被模型预测为负的负样本数量；FP(False Positive)是被模型预测为正的负样本数量；FN(False Negative)是被模型预测为负的正样本数量。由 TP，FN，FP，TN 构成的混淆矩阵如图 2 所示。

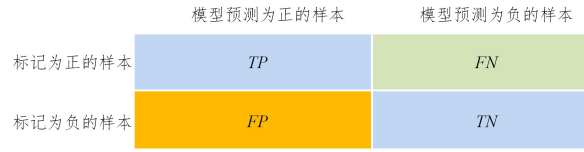

图 2 混淆矩阵

Fig. 2 Confusion matrix

准确率虽然可以判断出总的正确率，但是在样本不均衡的情况下，准确率并不能作为很好的指标来评价一个克隆检测器，因此使用精确率和召回率来作为克隆检测器的评价标准更均衡。为了平衡精确率和召回率，较为全面地评价一个代码检测工具，常常会引进 $F1$ 指数作为一个综合指标。

3.5 检测方法分类

目前代码克隆检测方法被主要分为 5 类[2]。已有研究表明[3]，每种克隆检测技术都有各自的优缺点。本文将在本节详细介绍各类方法中的典型方法，并对这 5 类代码克隆检测方法进行分析和比较，

3.5.1 基于文本的克隆检测方法

基于文本的克隆检测方法将源代码看作是字符序列，比较代码字符序列相似度并且返回字符串匹配结果集。这类方法易于实现，而且与语言无关。但是这类方法一般只能检测出 Type1，Type2 的代码克隆，对 Type3，Type4 代码克隆效果较差，不能很好地检测出在句法和语义层面上的代码修改。

Baker[20]在 1995 年提出了一种基于逐行比较的代码克隆检测工具 Dup。该工具用于定位大型软件中近乎相同的代码片段，解决代码开发过程中由于代码克隆产生的 Bug 传递现象。为了进行比较，Dup 首先从源代码文件中删除空白和注释，然后使用“参数化”的思想来检测 Type1 和 Type2 代码克隆。所有唯一标识符(例如变量名、方法名等)都被替换为唯一字符。此后，Dup 也被更新为使用“参数化”后缀树进行比较。

Duploc[38]于 1999 年被 Ducasse 提出，算法预处理阶段移除注释以及空白，将所有字符改为小写，通过删除代码克隆之间常见的结构来减少误报。它使用一个简单的字符串匹配算法来比较，逐行哈希，比较产生的结果存储在一个矩阵之中，并被可视化成一个点图。Koschke[54]的研究表明，Duploc 只能检测完全相同的复制粘贴片段。

2008 年，Nicad[31]被提出，Nicad 主要包括 3 个阶段：解析、规范化和比较。Nicad 先将给定粒度的代码片段(例如函数和块)解析成标准文本格式；然后将提取的片段进行规范化、过滤和抽象；最后使用优化的 LCS(Longest Common Subsequence)算法对提取的片段和归一化片段进行线性比较。与大多数方法不同，Nicad 直接构建克隆类。值得一提的是，Nicad 提供了一种增量检测模式，针对已分析过的系统，Nicad 只报告由于新版本而产生的克隆，而这种模式可以非常高效地进行代码克隆检测。

表 2 列出了基于文本的克隆检测方法及其特征，包括检测方法的核心思想、方法能检测到的代码语言、检测效果、方法所用验证数据集。其中检测效果代表方法能检测到的最高难度代码克隆类型，下同。

表 2 基于文本的代码克隆检测方法的特征

Table 2 Characteristics of text-based clone detection approaches

检测方法	核心思想	检测语言	检测效果	验证数据集
Dup[20]	对源代码预处理，引入了参数化匹配的概念，基于后缀树匹配	ALL	2	X Windows System
Nicad[31]	源代码标准化，使用 LCS 算法来进行简单的文本比较，与阈值进行比较	ALL	3	generated dataset
Duploc[38]	源代码标准化，简单的字符串比较算法，逐行哈希，使用点阵图可视化比较	ALL	2	generated dataset
SDD[39]	使用反向索引结构和 n 邻居距离算法检测克隆	ALL	3	开源软件系统

3.5.2 基于词法的克隆检测方法

基于词法的检测方法将代码转换成 token(符号、词汇)。将 token 序列视为抽象的代码表示。一般使用符合编译原理的符号序列进行标识。这种抽象的级别(粒度)可以根据预先定义的需要来决定抽取哪个层面的信息，抽取的粒度可以分为词汇、函数、语句等不同级别。相比于基于文本的代码表征方法来说，这种方法能够匹配到代码的特有信息，但从本质上来说，这种方法还是与基于文本表征的方法一样，单纯地将代码视为符号序列。基于符号的方法的缺点在于没有考虑代码中所包含的结构信息，比如，token 结构顺序发生改变时算法检测不到。同时，这类方法对代码语句修改比较敏感，容易漏检只有特定细微差别的代码克隆。图 3 给出了基于词法的代码表征的一般流程。

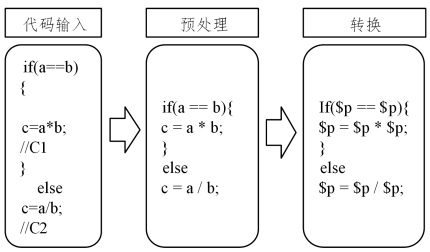

图 3 基于词法的代码表征流程

Fig. 3 Lexics-based code representation process

2002 年，CCFinder[40] 被提出，它通过一种转化规则将源代码语句转换为 token 流。与 Baker 早前工作相似，这种转换也包括了参数化的思想，同时能够移除一些无用的 token，利用前缀树和一种新的匹配算法来查找代码克隆。CCFinder 能够在几分钟内解析数百万行代码，针对 1—2 类型的代码克隆检测获得了很好的结果。此外，它还能够提取 C，C++，java 和 COBOL 的代码克隆。在 2007 年，该工具被扩展为 D-CCFinder[41]，可以用于非常大系统的分布式代码检测工具。

CP Miner[21] 作为 CCFinder 的继承者于 2004 年被推出。该工具在检测速度上取得很大提高，检测精度超过 CCFinder。同时，在代码克隆中增加了 Bug 检测。CP Miner 被应用到 Linux 和 apache webserver 上发现了当时未被检测出的 Bug。

CCLearner[27] 为第一个利用深度学习的基于 token 的克隆检测方法，其提出者认为保留字、类型标识符、方法标识符和变量标识符这样的术语可以有效地表征代码。他们使用 BigCloneBench 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模型进行训练，学习特征并用于代码克隆检测。

表 3 列出了基于词法的克隆检测方法及其特征。

表 3 基于词法的代码克隆检测方法的特征

Table 3 Characteristics of lexis-based clone detection approaches

检测方法	核心思想	检测语言	检测效果	验证数据集
CP-Miner[21]	频繁子序列挖掘来匹配克隆的代码段	C，C++	2	Linux-System Apache-webserver
CCFinder[40]	Token 标准化，基于后缀树匹配	C，C++，C，Java HTML，COBOL	2	工业级别代码
D-CCFinder[41]	用于非常大的系统的 CCFinder 分布式实现	C	N/A	FreeBSD 开源软件集合
FCFinder[42]	对每个代码文件解析生成一个标记序列，MD5 哈希散列，精确匹配	C	N/A	FreeBSD 开源软件集合
Boreas[43]	统计变量的数量，生成一个计数矩阵，并且计算余弦相似性	C，Java	3	JDK7，Linux 内核代码
SourcererCC[44]	优化的反向索引结构和过滤的启发式算法实现大规模克隆检测	Java，C，C#	3	Bigclonebench，IJA-Dataset
Iclones[45]	后缀树的标准化 token 比较	Java，C	2	经过验证的数据集
Javad[46]	利用卷积注意力神经网络对代码生成摘要描述，使用描述做为度量来计算相似性	Java	N/A	N/A
CCAligner[47]	利用滑动窗口和非精确匹配算法来计算相似性	C，Java	3	Bigclonebench generated dataset
CCLearner[48]	利用神经网络在 token 级别(token 频率)进行训练和检测	Java	3	Bigclonebench
CDSW[49]	词法分析，计算每个语句哈希，使用 Smith-Waterman 算法识别相似性哈希	C，Java	3	Bellon's benchmark
FRISC[50]	词法分析，计算语句哈希，折叠重复的语句，和阈值比较	C，Java	3	Bellon's benchmark

3.5.3 基于语法的克隆检测方法

基于语法的克隆检测方法考虑源代码语法规则,将源代码转换为其对应的抽象语法树。图4给出了基于语法的代码表征流程例子。抽象语法树(AST)是一个比较常用的将代码转换为树的一个方法,能够保留源代码中的语法信息。基于树的方法可以避免由于格式和句法问题引起的问题,因为基于树的方法关注于不同程序结构层面上的相似度。但是基于树的方法时间复杂度比较高,比如两个树有 N 个结点,那么它们相似度比较的时间复杂度上限为 $O(N^3)$。还有些方法使用编译器对源代码进行编译,但是部署成本高。基于树的方法的优点是能够考虑到源代码的结构特性,其缺点是不能识别出标识符和文本值的不同,并且基于树的方法计算开销大。

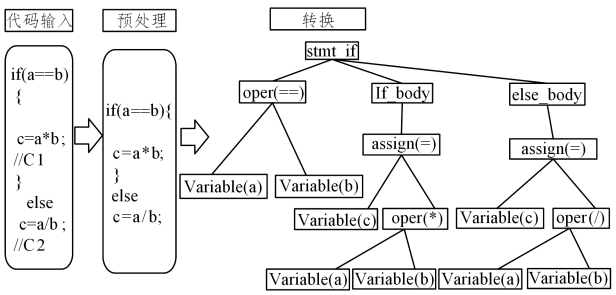

图4 基于语法的代码表征流程

Fig. 4 Syntax-based code representation process

1998年,CloneDR[16]工具被提出,它使用ASTs来比较源代码。CloneDR在源代码中提取的AST上使用树匹配方法查找代码片段的精确或接近精确的匹配。CloneDR仅限于检测1型和2型代码克隆。

2007年,Deckard[22]代码克隆检测工具引入了一种新的代码克隆检测方法。Deckard基于树,类似于CloneDR,但采用了一种创新的方法将AST转换为特征向量,然后通过向量聚类以减少比较次数。Deckard应用于大型存储库(包括Linux内核)时有着极好的结果,能够检测类型1－3的代码克隆。

2016年,White等[51]首次将深度学习应用于代码克隆检测,White等提出的基于树的深度学习代码克隆检测方法在398个文件级和180个方法级克隆对的检测中,正确预测率达到了93%,可以检测到所有4种类型的克隆,证明了深度学习在克隆检测领域中的高性能。

Wei等[52]将克隆检测描述为一个有监督的hash学习问题,并提出了一个端到端深度特征学习框架CDLH用于功能克隆检测,该框架利用基于AST的LSTM来挖掘词法和句法信息,通过对功能相似性的监督来指导特征学习。其研究表明无监督的训练方式在前3种类型上可以取得不错的效果,但是在很难检测到功能克隆对。CDLH在OJClone上的表现比较差,我们认为是由于OJClone中程序的变量名通常是无意义的,因此CDLH会丢失大量的词汇信息,只能捕获一些语法信息。

2019年,Zhang等[53]提出了一种名为ASTNN的方法。与现有基于整个AST的模型不同,ASTNN将每个大的AST分解为一系列小的语句树,并通过捕获语句的词法和语法知识将语句树编码为向量,采用双向RNN模型最终生成代码片段的向量表示。与大多数将抽象语法树转化成完全二叉树或是完全树的方法不同,ASTNN完整保留了抽象语法树的结构信息。实验结果也证实了该方法的优越性,足以检测到所有类型的代码克隆。

表4列出了基于语法的克隆检测方法及其特征。

表4 基于语法的代码克隆检测方法的特征

Table 4 Characteristics of syntax-based clone detection approaches

检测方法	核心思想	检测语言	检测效果	验证数据集
CloneDR[16]	利用哈希函数分割AST,再在子树之间进行比较	C	2	开源商业软件
Deckard[22]	利用解析器来生成解析树,用向量来获得树的结构信息,使用LSH进行聚类	C,Java	3	Linux内核代码
White[51]	通过递归自动编码器进行表示学习,以检测代码克隆。它们将每个源代码片段表示为标识符和文本类型的流(来自AST leaf节点)	Java	4	开源Java系统
CDLH[52]	一个端到端的克隆检测深度特征学习框架,该框架通过基于AST的LSTM同时学习哈希函数和代码片段的表示,以兼顾源代码的词法和语法方面	C,Java	4	Bigclonebench,OJClone
ASTNN[53]	将每个大的AST分解成一系列小的语句树,并通过捕获语句的词法和语法知识将语句树编码为向量。通过一个双向GRU模型最终产生代码片段的向量表示	C,Java	4	Bigclonebench,OJClone
Koschke[54]	抽象语法后缀树,子树匹配	C	2	Bellon's benchmark
WASTK[55]	WASTK首先将程序的源代码转换为一个抽象语法树,为每个节点分配相应的权重,然后通过计算两个抽象语法树的树核来获得相似性	C	N/A	由生成器生成的代码库
Wahler[56]	从XML中提取出AST,使用频繁项集收集技术	Java,C++,PROLOG	2	Java核心软件
Zeng[57]	利用加权RAE分析程序抽象语法树,提取程序特征并将函数编码成向量。使用TF-IDF对AST的不同类型节点进行加权	Java	3	Bigclonebench

3.5.4 基于语义的克隆检测方法

基于语义的方法不仅希望获得代码之中的结构信息,还试图获得代码中的语义信息,例如,代码的控制流和数据流。图5给出了基于语义的代码表征流程。作为基于语义表征方法的一种,基于图的检测技术可以同时考虑源代码的语法和语义信息,可以检测到非连续的代码克隆。

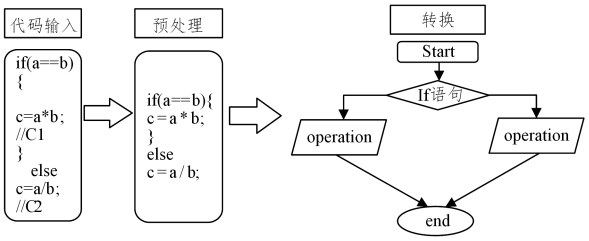

图 5 基于语义的代码表征流程

Fig. 5 Semantic-based clone representation process

基于图的检测技术使用程序生成的数据流图和控制流图来构成程序依赖图，数据流代表了源代码中数据的走向，控制流代表了逻辑的走向。然而与基于树的方法相似，基于图的方法对图进行表征，因此计算复杂度也很高。同时，基于图的方法需要一个程序依赖图的生成器，对不同语言需要不同的生成器。

Komondoor 等[58]提出的方法使用程序依赖图和程序切片来寻找代表克隆的同构 PDG 子图。这种方法的主要好处是可以找到不连续的、变量名重命名以及语句重新排序的代码克隆。然而其存在着运行时间过长、资源需求巨大等问题。

Higo 等[59]于 2011 年提出了一种基于 PDG 的增量代码克隆检测技术，用于解决基于 PDG 的检测技术耗时严重的问题。结果表明，增量技术可以用于基于 PDG 的检测技术，在不影响检测精度的情况下大幅度提高检测速度。

表 5 列出了基于语义的克隆检测方法及其特征。

表 5 基于语义的代码克隆检测方法的特征

Table 5 Characteristics of semantic-based clone detection approaches

检测方法	核心思想	检测语言	检测效果	验证数据集
Gabel[23]	将 PDG 转换为树来处理，重用 Deckard 匹配算法，LSH 聚类	C,C++	4	Linux 内核
CoP[29]	基于语义等价基本块的最长公共子序列检测方法	Binarycode	4	generated dataset
Komondoor[58]	使用程序切片来寻找同构的 PDG 子图	C	3	Unix 程序
Higo[59]	基于 PDG 的增量代码克隆检测技术	Java	3	开源 Java 软件
DeepSim[60]	将代码片段的控制流和数据流编码到语义矩阵中，在此基础上设计了深度神经网络来度量代码功能相似性	Java	4	GCJ dataset Bigclonebench
SCDetector[61]	将图视作社交网络，获得 token 节点的中心性信息，将获得语义信息的 token 放入神经网络中训练	Java	4	Bigclonebench Google Code Jam
Chen[62]	从 Android 应用程序包中提取方法并将每个方法转换为一个控制流图，映射至三维空间并且用“质心”来判断相似性	Android	N/A	安卓市场软件
Gemini[63]	面向二进制代码基于神经网络的嵌入计算框架，将二进制代码中的方法转换为控制流图，将控制流图表征为向量	Binarycode	N/A	generated dataset
CCGraph[64]	使用两阶段的滤波算法获取 PDG 的特征，基于图核的近似图匹配算法计算相似性	Java,C	4	Bigclonebench generated dataset

3.5.5 基于度量值的克隆检测方法

基于度量值的方法只对代码进行固定粒度的检测，基于度量值的方法并没有直接比较源代码，而是计算度量值的相似性。度量值包括函数名、声明、布局、控制流、返回值等。基于度量值方法的优点是准确率高，并且检测速度块。

1996 年，Mayrand[18]提出了一种基于度量值的代码克隆检测工具。该工具设计之初是为了检测学生提交代码中存在的抄袭现象，此后也被应用于大型的电信系统。研究表明克隆检测所获得的信息可以用于旧系统重大改造的计划阶段：如果存在大量克隆，并且这些克隆对于维护有着潜在风险，那么可以在重新设计任务之前把这些克隆给删除；而对于一些希望留下的代码克隆则不应该删除，甚至有时应该鼓励重用。

表 6 基于度量值的代码克隆检测方法的特征

Table 6 Characteristics of metric-based code cloning detection methods

检测方法	检测方法	检测方法	检测方法	检测方法
Mayrand[18]	从函数的名称、布局、表达式和控制流计算度量，计算相似性	C,C++	2	大型电信监控系统
Kontogiannis[65]	使用度量值的数值比较和使用最小编辑距离的动态规划(DP)算法	C	3	大型软件系统
Shawky[66]	度量置换寻找最佳度量序列，生成向量，聚类	C	N/A	N/A
CMCD[67]	使用计数矩阵比较代码段，计数矩阵由每个变量的计数向量构成	Java	2	JDK1.60_18
Oreo[68]	使用 24 种度量(包括声明变量的数量)表示一个代码片段，然后使用 48 维向量训练二进制分类器	Java	3	Bigclonebench
Davey[69]	忽略标识符和运算符，而是考虑关键字的频率、缩进模式和表示片段的每行的长度。这些特征向量被传递到一个自组织映射来检测克隆	N/A	N/A	N/A
Salwa[70]	收集功能性度量，分型聚类提取克隆类	C	N/A	开源 C 语言程序
Li[71]	将代码片段视为度量空间的一个坐标点，利用度量空间内的距离来度量代码的相似性	N/A	N/A	N/A

3.5.6 其他代码克隆检测方法

Chaiyong 等[72]提出了一种基于图像相似性的代码克隆检测技术 Vincent,该技术通过对原始源代码应用语法高亮和图像转换,将代码转化为图像。在经过高斯模糊过滤之后,对图像进行比较,生成克隆检测报告。Kim 等[73]提出了一种通过比较程序的内存状态来检测代码克隆的方法 MeCC。它首先使用路径敏感的基于语义的静态分析器来记录每个过程的内存状态,然后比较内存状态以确定克隆。该方法可以有效地检测语义克隆。由于静态分析器只能收集过程级的内存状态,因此 MeCC 并不能保证语义级别的代码克隆检测有效性。但是由于从程序中提取抽象的内存状态在时间以及内存上都是一项计算量很大的任务,所以该方法仍存在一些局限性。

表 7 其他代码克隆检测方法的特征

Table 7 Characteristics of other code clone detection methods

Vincent[72]	代码预处理,转换为图像,图像高斯模糊化,比较图像相似度	Java	2	generated dataset
MeCC[73]	通过语义静态分析技术,收集抽象内存状态,通过对这些内存状态的比较,判断代码相似性	Python,Java,SQL	4	generated dataset
Grant[74]	对原始的方法词法单元矩阵奇异值分解;独立分量分析,识别出与输入数据对应的新向量空间中的点,测量两个向量之间的有效距离来计算相似性	C	N/A	Linux 内核
Wahler[75]	将源代码的语句序列汇总为一个语句集,然后采用一种适用的数据挖掘算法对语句集中的高频语句进行搜索	C++,PROLOG,Java	2	JDK
Fang[76]	一种细粒度的代码克隆检测方法,将具有调用关系的方法视为功能实现,融合了语法和语义信息进行代码克隆检测	C++	4	OJClone
Tufano[77]	基于深度学习的多种表征方式结合的克隆检测方法	Java	4	generated dataset

表 8 不同类型克隆检测技术的对比

Table 8 Comparison of different types of cloning detection techniques

检测方法分类	中间表示形式	匹配算法	优点	缺点
基于文本	字符	字符匹配	算法实现简单,几乎可以检测所有语言的代码,具有很高的移植性。	不能识别程序中的语法,语义等信息,准确率并不高,只能检测 Type-1 和 Type-2 克隆
基于词法	token	LCS、后缀树、指纹算法、哈希索引	相对于文本而言,可以检测更多代码特有信息。而且相对于复杂的检测方法有着更低的时空复杂度	本质上来讲,还是将代码看作符号序列。不能识别程序的语法、语义等信息。依赖于代码行的顺序,一旦顺序发生了改变,就检测不到重复代码
基于语法	抽象语法树,解析树	子树匹配	可以识别程序的语法信息,准确率较高	构造 AST 以及 AST 的比较代价较大,识别不出程序的语义信息
基于语义	程序依赖图	子图匹配、程序切片	可以识别出程序的语义信息	构造 PDG 以及 PDG 的匹配代价较大,需要 PDG 生成器

3.6 克隆检测技术应用

(1)Bug 解决方案推荐:解释编译器错误和异常消息对新手程序员而言是一项挑战,展示其他程序员如何纠正类似错误的例子将有助于新手理解以及纠正其错误。Hartmann 通过推荐相似方法应用的 Bug 解决方案来解决项目中遇到的错误信息[78]。

(2)克隆相关 Bug 检测:已有研究表明,代码克隆与软件系统中的 Bug 和不一致性直接相关[27]。当程序开发人员忘记对变量名等进行一致性修改,或是克隆后代码 Bug 修复不完全,软件系统都将被引入新的 Bug[28]。通常将这些修改不一致或是存在 Bug 的代码克隆视作为易受攻击的代码,Seulbae 等在 2015 年提出了一种名为 VUDDY[79]的可扩展的代码克隆检测方法,能够在代码库中快速找到那些易被攻击的代码克隆。

(3)程序理解:Maletic 等利用程序的语义和结构信息来定义软件构件之间的语义相似度,将软件系统中的组件聚集在一起,使用软件系统的简单结构信息来帮助程序理解[80]。

(4)横切关注点的确定:在程序系统中部分代码在多个功能模块中重复的出现,这些代码即是横切关注点。日志功能就是横切关注点的一个典型案例,它往往横跨系统中的每个业务模块,而很多程序员在面对这种情况都会采用复制-粘贴。Bruntink 等在识别横切关注点代码上使用了 3 种不同的克隆检测技术[81],取得了不错的效果。

(5)克隆提醒:为了避免代码克隆对软件系统造成不好的影响,Vibran 等[82]在程序员写代码时识别克隆,并提醒程序员何处存在克隆。

(6)代码重构:通过使用代码克隆检测技术和代码重构技术替换常见代码,以减小代码大小,增加代码的可理解性[83-84]。

(7)版权保护及剽窃检测:源代码作为一种具有专利的知识产权,可能会被恶意软件非法盗用。通过克隆代码检测技术将具有知识产权的源代码与被测代码进行比对,判断是否存在抄袭现象,为知识产权维护提供依据[29]。Steven 等提出一种名为 ASM2VEC[85]的二进制代码克隆检测技术,能够获取汇编代码中丰富的语义信息,在面对代码混淆和代码模糊所带来的变化时,具有更强的鲁棒性。

3.7 已开源工具

我们在调研中发现,绝大多数检测工具及方法都没有开源。对于学术界和工业界来说,这无疑是一种巨大的损失。与未公开的工具相比,已开源的工具将被更快、更好地不断进行更新和改进。表 9 列出了我们统计的已开源方法和工具及其下载地址。

表 9 已开源方法

Table 9 Open source method

检测方法	下载链接
Deckard[22]	https://github.com/skyhover/Deckard
CloneTracker[33]	www.cs.mcgill.ca/~swevo/clonetracker
CCFinder[40]	http://www.ccfinder.net/index.html
SourcererCC[44]	https://github.com/Mondego/SourcererCC
ASTNN[53]	https://github.com/zhangj111/astnn
DeepSim[60]	https://github.com/parasol-aser/deepsim
MeCC[73]	http://ropas.snu.ac.kr/mecc
SHINOBI[86]	https://sdlab.naist.jp/projects/shinobi.html
Clone Merge[87]	https://krishnanm86.wixsite.com/clonerge
CtCompare[88]	https://github.com/awakecoding/ctcompare
CloneWorks[89]	https://github.com/jeffsvajlenko/CloneWorks
CodeEase[90]	https://github.com/shamsa-abid/CodeEase
CCFinderSW[91]	https://github.com/YuichiSemura/CCFinderSW
Nicad[92]	https://www.txl.ca/txl-nicaddownload.html
Clone-Differentiator[93]	https://sites.google.com/site/yinxingxue/home/projects/clonedifferentiator
Agec[94]	https://github.com/tos-kamiya/agec
LICCA[95]	https://github.com/tvislavski/licca
Mostaeen2018[96]	https://github.com/pseudoPixels/ML_CloneValidationFramework
BCFinder[97]	http://www.imm.dtu.dk/~sljo/bcfinder/gettingstarted.html
CCCD[98]	http://www.se.rit.edu/~dkrutz/CCCD/index.html
Simian	http://www.harukizaemon.com/simian/index.html
ConQAT[99]	http://www.conqat.org/
JCCD[100]	http://jccd.sourceforge.net
CloneCognition[101]	https://github.com/pseudoPixels/CloneCognition

4 代码克隆检测研究的主要挑战

虽然代码克隆检测的研究已经有超过 20 年的历史，在这期间也有非常多的研究者们贡献了非常多的成果，但是现有的研究成果与工业界的期望还是有着一定的距离。本节将从学术方面、工程实践、潜在风险 3 个方面阐述目前代码克隆研究中存在主要挑战。

4.1 科学问题

(1)代码克隆定性研究

对于代码克隆产生的原因、发展过程及影响，研究者们已经进行了定性分析[102]。Kim 等[104]从代码克隆系谱的角度出发，分析了两个开源软件系统的克隆系谱，得到了克隆并不总是有害的结论。他们的研究还表明，许多长期持续变化的克隆是不能局部重构的，这样的克隆不能通过重构从系统中移除。虽然众多研究者们在代码克隆产生的原因及影响上都进行了定性的分析，但在目前的代码克隆研究中，尚未有研究者可以进一步对代码克隆的特征以及处理方式进行分析，即面对不同的代码克隆，采取不同的优化方式。研究者们需要在代码克隆研究的定性问题中进行更一步的研究，针对不同的应用背景，对代码克隆产生的原因、特征以及影响进行分析，提高代码克隆检测技术在现实应用场景中的可行性。

(2)代码克隆检测技术研究

随着机器学习以及深度学习技术在代码克隆检测领域的普遍应用，曾经很难检测到的 Type-3 和 Type-4 克隆也能被检测到，并且有着很高的精确率和召回率。但是深度学习方法仍然存在着局限性，由于其训练时间过长，因此很难将这些方法工程化，将代码克隆检测方法和实际应用相结合。因此，提出一种在精确度、召回率、可扩展性、可移植性和健壮性都完美的克隆检测方法，依旧是我们所期盼的。

4.2 工程实践

代码克隆研究的实用性：代码克隆流程中包含预处理、代码表征、相似度比较和后处理。然而现有的研究大多数只针对前 3 个过程进行研究分析。而对于后处理阶段，大多数已公开的可使用的代码克隆检测工具仅仅是将代码相似片段映射在源代码中[103]。

Mostaeen 等[101]指出，在代码克隆检测工具使用过程中，检测器检测到的克隆有时并不被用户接受。用户常常需要从已有的克隆检测报告中手动地进行验证，这是一个耗时耗力的工作。针对不同的需求，如何有效地制定相应的克隆报告分析，需要未来工作的进一步分析与研究。

4.3 潜在风险

(1)对 AI 生成的代码进行有效克隆检测

随着人工智能的迅速发展，软件开发前沿理论研究包括群智软件和基于 AI 自动生成的软件。考虑系统的可维护性和稳定性，由这些技术开发出来的软件系统也需要做克隆检测，以适应新的发展需求。

(2)抗混淆的代码克隆检测方法

新兴的混淆技术例如识别密码算法已经被现代恶意软件广泛采用，以达到规避检测的目的，然而现有的代码相似性比较方法，无论是基于源代码还是基于二进制代码，大多不能抵抗混淆，因此自动检测非常困难[29]。未来代码克隆检测工具研究需要针对这一问题开展，以适应新的发展需求。

(3)跨语言代码克隆检测

以往工作已经广泛研究了相同编码语言编写的程序代码的克隆检测问题。但是跨编程语言的代码克隆检测仍未得到很好的研究[105]。事实上多种编程语言共同编写的系统也一直存在。

为了吸引更多开发者或是支持不同的平台，开源项目组织或商业公司倾向于采用多种语言实现其项目。跨语言代码克隆可能会对系统的可维护性产生负面影响，对代码的一致性修改在多语言开发的系统中也是一件耗时、耗力的任务。因此跨语言代码克隆检测是多语言软件项目维护中的重要部分。

(4)基准数据集的不完整

研究者们在验证自己提出的代码克隆检测工具的有效性时会通过寻找开源软件、通过修改、人工标注等方式对代码克隆进行修改以创建自己的数据集[7]。然而这样创建的数据集规模不够大、不足以形成统一的评价标准。为了解决这个问题，Svajlenko 等提出了 BigCloneBench[35]。BigCloneBench 从 IjaDataset-2.0 中提取了 800 万个经过验证的克隆集合，包含了 Type-1 至 Type-4 的代码克隆，既含有项目内克隆，也有项目间克隆。然而 BigCloneBench 也存在着缺陷，即如此大的数据集只有 43 个功能，这显然不符合现实中的软件系统情况。

除此之外，现有的基准数据集仅有 C 和 Java 两种语言，对于其他流行的编程语言而言，仍然缺少让大家认同的公共基准数据集。

5 解决思路

针对上一节代码克隆领域中存在的主要挑战，本节将讨论未来代码克隆检测的可能研究方向。

5.1 知识图谱的技术可以为代码克隆定性研究提供解决思路

近年来，开源运动如火如荼，大量软件企业积极参与开源软件生态系统的构建。持续跟踪、记录大型开源软件系统中的代码克隆相对较容易。研究者们可以利用知识图谱相关技术[107]记录代码克隆在软件系统中的产生、繁衍以及消亡的过程，并以此总结出代码克隆特征与处理过程之间的一般规律。

在目前的技术中，代码克隆检测技术已经能够检测到大多数代码克隆，但是仍不能完全应用在工业界之中。一方面是因为4.1节中提到的问题。另一方面是因为现有的克隆报告并不能直接被用户接受[108]。Xiao的成功很好地展现了开发者和用户之间的良好协作，未来的代码克隆检测工具需要考虑用户的需求，给出相应的代码克隆检测报告。

5.2 综合考虑代码的“额外信息”，并以此提出更准确的检测模型

除了代码本身包含的大量语义信息及结构信息外，研究者们可以利用起目前尚未被利用的“额外信息”，如注释信息以及API信息[100]。借助外部信息，可以增强模型对代码的表征能力，提高模型的准确率。

5.3 代码克隆检测工具开源的必要性

已有研究表明[9]，截至2019年，仅有22%的代码克隆检测工具是开源的，无论是对于学术界还是工业界，这无疑是一种莫大的损失。同时，相比于封闭的源代码而言，已经开源的工具正在不断地更新和改进。像SourcererCC这样的工具已经可以扩展到SoucererCC-i。未开源的工具往往仅在研究人员小组中得到改进，而开源工具则展示了许多不同研究人员群体之间的协作。开源工具为新研究者们提供思路，使其与产品使用者们保持联系，获取反馈，在此基础上加以改进。

5.4 采用新的算法可以减少对数据集的依赖

近年来，深度学习被引入到代码克隆检测领域[51]。基于深度学习的方法需要大量的、高质量的人工标注数据集，然而代码标注的过程十分的困难：理解代码并进行标注需要大量的专业知识，审查者们由于其先验知识不一，对代码克隆的判定标准也不一样。

因此，除了提出更准确的数据标注方法外[7]，尝试一些新的算法减少对标记数据集的依赖是可行的思路。人工智能领域内的一些研究指出，通过采用无监督、半监督以及强化学习等方法[110]可以大大减少对标记数据集的依赖。

结束语 随着IT信息技术发展，开源程序已经成为了一种潮流。在软件开发过程中，代码克隆普遍存在而且随着时间持续增多。已经有大量研究表明，代码克隆会给软件开发和维护带来不好的影响。本文通过仔细梳理发现并不是所有的代码克隆都对系统有害而需要被移除，为了提升代码质量，我们应该采用不同的方式应对不同类型的代码克隆。本文基于当前研究进度，对在软件开发过程中各种克隆产生的原因、优缺点以及克隆分类进行分析；同时介绍了目前已有各种代码克隆检测方法，从科学、工程以及潜在风险3个角度归纳总结了代码克隆检测研究目前存在的问题，最后给出了问题的可能解决思路以及未来的发展趋势。

参 考 文 献

[1] KIM M，BERGMAN L，LAU T，et al. An Ethnographic study of copy and paste programming practices in OOPL[C]// Proc. of 3rd International ACM-IEEE Symposium on Empirical Software Engineering (ISESE'04). 2004：83-92.

[2] ROY C K，CORDYJ R. A survey on software clone detection research[J]. Queen's School of Computing TR，2007，541(115)：64-68.

[3] BURD E，BAILEY J. Evaluating Clone Detection Tools for Use during Preventative Maintenance[C]// Proc. of the 2nd IEEE International Workshop on Source Code Analysis andManipulation. 2002：36-43.

[4] RATTAN D，BHATIA R，SINGH M. Software clone detection：A systematic review[J]. Information and Software Technology，2013：1165-1199.

[5] ZHANG D，LUO P. Survey of Code Similarity Detection Method and Tools[J]. Computer Science，2019，47(3)：5-10.

[6] SHENEAMER A，KALITA J. A survey of software clone detectiontechniques[J]. International Journal of Computer Applications，2016，137(10)：1-21.

[7] CHEN Q Y，LI S P，YAN M，et al. Code clone detection：A literature review[J]. Ruan Jian Xue Bao，2019，30(4)：962-980.

[8] KOSCHKE R. Survey of research on software clones. Dagstuhl Seminar Proceedings[J]. Schloss Dagstuhl-Leibniz-Zentrum für Informatik，2007.

[9] WALKER A，CERNY T，SONG E. Open-source tools and benchmarks for code-clone detection：past，present，and future trends[J]. ACM SIGAPP Applied Computing Review，2020，19(4)：28-39.

[10] SU X H，ZHANG F L. A survey for Management-Oriented Code clone Research[J]. Chinese Journal of Computers，2018，41(3)：628-651.

[11] HOU M，ZHANG L P. Research on Software Clone Detection Technology[J]. Computer Technology And Development，2019(8)：86-91.

[12] KAPSER C J，GODFREYM W. “Cloning considered harmful” considered harmful：patterns of cloning in software[J]. Empirical Software Engineering，2008，13(6)：645.

[13] CORDY J R. Comprehending reality-practical barriers to industrial adoption of software maintenance automation[C]// 11th IEEE International Workshop on Program Comprehension. 2003：196-205.

[14] FOWLER M，BECK K，OPDYKEW R. Refactoring：Improving the design of existing code[C]// 11th European Conference. 1997.

[15] UEDA Y，KAMIYA T，KUSUMOTOS，et al. Gemini：Maintenance support environment based on code clone analysis[C]// Proceedings Eighth IEEE Symposium on Software Metrics. 2002：67-76.

[16] BAXTER I D，YAHIN A，MOURA L，et al. Clone detection using abstract syntax trees[C]// Proceedings International Conference on Software Maintenance. 1998：368-377.

[17] RAJAPAKSE D，STAN JARZABEK S. Using server pages to unify clones in web applications：a trade off analysis[C]// 29th International Conference on Software Engineering(ICSE '07). IEEE Computer Society，2007：116-126.

[18] MAYRAND J，LEBLANC C，MERLO E. Experiment on the

Automatic Detection of Function Clones in a SoftwareSystem Using Metrics[C]// ICSM. 1996:244.
[19] LAGÜE B, PROULX D, MERLO E, et al. Assessing the benefits of incorporating function clone detection in a development process[C]// Proceedings of the International Conference on Software Maintenance 1997. 1997:314-321.
[20] BAKER B. On finding duplication and near-duplication in large software systems[C]// Proceedings of the 2nd Working Conference on Reverse Engineering (WCRE'95). 1995:86-95.
[21] 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.
[22] JIANG L, MISHERGHI G, SU Z, et al. Deckard: Scalable and accurate tree-based detection of code clones[C]// 29th International Conference on Software Engineering (ICSE'07). 2007: 96-105.
[23] GABEL M, JIANG L, SU Z. Scalable detection of semantic clones[C]// Proceedings of the 30th International Conference on Software Engineering. 2008:321-330.
[24] SAHA R K, ASADUZZAMAN M, ZIBRANM F, et al. Evaluating code clone genealogies at release level: An empirical study [C]// 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation. 2010:87-96.
[25] PRECHELT L, MALPOHL G, PHILIPPSEN M. Finding plagiarisms among a set of programs with JPlag[J]. Journal of Universal Computer Science, 2002, 8(11):1016.
[26] LI J, ERNST M D. CBCD: Cloned buggy code detector [C]// 2012 34th International Conference on Software Engineering (ICSE). 2012:310-320.
[27] JIANG L, SU Z, CHIU E. Context-based detection of clone-related bugs[C]// Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'07). 2007:55-64.
[28] MONDAL M, ROY C K, SCHNEIDERK A. Dispersion of changes in cloned and non-cloned code[C]// 2012 6th International Workshop on Software Clones (IWSC). 2012:29-35.
[29] LUO L, MING J, WU D, et al. Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection[J]. IEEE Transactions on Software Engineering, 2017, 43(12):1157-1177.
[30] RAGKHITWETSAGUL C, KRINKE J. Using compilation/decompilation to enhance clone detection[C]// 2017 IEEE 11th International Workshop on Software Clones (IWSC). 2017:1-7.
[31] ROY C K, CORDY J R. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C]// 16th IEEE International Conference on Program Comprehension. 2008:172-181.
[32] WHALE G. Plague: plagiarism detection using program structure[D]. University of New South Wales, 1988.
[33] DUALA-EKOKO E, ROBILLARD M P. Clonetracker: tool support for code clone management[C]// Proceedings of the 30th International Conference on Software Engineering. 2008: 843-846.
[34] BELLON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools[J]. IEEE Transactions on Software Engineering, 2007, 33(9):577-591.
[35] SVAJLENKO J, ROY C K. Evaluating clone detection tools with bigclonebench[C]// 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2015: 131-140.
[36] SVAJLENKO J, ROY C K. Bigcloneeval: A clone detection tool evaluation framework with bigclonebench[C]// Proc. ICSME. 2016:596-600.
[37] MOU L, LI G, ZHANG L, et al. Convolutional neural networks over tree structures for programming language processing[C]// Thirtieth AAAI Conference on Artificial Intelligence. 2016: 1287-1292.
[38] DUCASSE S, RIEGER M, DEMEYER S. A Language Independent Approach for Detecting Duplicated Code[C]// Proc. Int', l Conf. Software Maintenance. 1999:109-118.
[39] LEE S, JEONG I. SDD: high performance code clone detection system for large scale source code[C]// Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. 2005: 140-141.
[40] 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.
[41] LIVIERI S, HIGO Y, MATUSHITA M, et al. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder[C]// Proc. ICSE. 2007: 106-115.
[42] SASAKI Y, YAMAMOTO T, HAYASE Y, et al. Finding file clones in FreeBSD ports collection[C]// IEEE Working Conference on Mining Software Repositories (MSR 2010). 2010:102-105.
[43] YUAN Y, GUO Y. Boreas: an accurate and scalable token-based approach to code clone detection[C]// Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. 2012:286-289.
[44] SAJNANI H, SAINI V, SVAJLENKO J, et al. SourcererCC: Scaling code clone detection to big-code[C]// Proceedings of the 38th International Conference on Software Engineering. 2016: 1157-1168.
[45] GÖDE N, KOSCHKE R. Incremental clone detection[C]// 13th EuropeanConference on Software Maintenance and Reengineering (CSMR). 2009:219-228.
[46] GHOFRANI J, MOHSENI M, BOZORGMEHR A. A conceptual framework for clone detection using machine learning[C]// 2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI). 2017:810-817.
[47] WANG P, SVAJLENKO J, WU Y, et al. CCAligner: a token based large-gap clone detector[C]// Proceedings of the 40th International Conference on Software Engineering. 2018: 1066-1077.
[48] LI L, FENG H, ZHUANG W, et al. Cclearner: A deep learning-based clone detection approach[C]// 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). 2017:249-260.
[49] HOTTA K H, HIGO Y. Gapped code clone detection with lightweight source code analysis[C]// 2013 21st International Conference on Program Comprehension (ICPC). 2013:93-102.
[50] MURAKAMI H, HOTTA K, HIGO Y, et al. Folding repeated instructions for improving token-based code clone detection

[C]// 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation. 2012:64-73.

[51] WHITE M, TUFANO M, VENDOME C, et al. Deep learning code fragments for code clone detection[C]// 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). 2016:87-98.

[52] WEI H, LI M. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code[C]// IJCAI. 2017:3034-3040.

[53] ZHANG J, WANG X, ZHANG H, et al. A novel neural source code representation based on abstract syntax tree[C]// 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). 2019:783-794.

[54] KOSCHKE R, FALKE R, FRENZEL P. Clone detection using abstract syntax suffix trees[C]// 2006 13th Working Conference on Reverse Engineering. 2006:253-262.

[55] FU D, XU Y, YU H, et al. WASTK: a weighted abstract syntax tree kernel method for source code plagiarism detection[J/OL]. Scientific Programming. https://doi. org/10. 1155/2017/7809047.

[56] WAHLER V, SEIPEL D, WOLFF J, et al. Clone detection in source code by frequent itemset techniques[C]// 4th IEEE International Workshop on IEEE . Source Code Analysis and Manipulation. 2004:128-135.

[57] ZENG J, BEN K, LI X, et al. Fast code clone detection based on weighted recursive autoencoders[J]. IEEE Access, 2019. 7: 125062-125078.

[58] KOMONDOOR R, HORWITZ S. Using slicing to identify duplication in source code[C]// International Static Analysis Symposium. Springer. 2001:40-56.

[59] HIGO Y, YASUSHI U, NISHINO M, et al. Incremental code clone detection: A pdg-based approach[C]// 18th Working Conference on Reverse Engineering. 2011:3-12.

[60] ZHAO G, HUANG J. Deepsim: Deep learning code functional similarity[C]// Proc. ESEC/FSE. 2018:141-151.

[61] WU Y, ZOU D, DOU S, et al. SCDetector: Software Functional Clone Detection Based on Semantic Tokens Analysis[C]// 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020:821-833.

[62] CHEN K, LIU P, ZHANG Y. Achieving accuracy and scalability simultaneously in detecting application clones on android markets[C]// Proceedings of the 36th International Conference on Software Engineering. 2014:175-186.

[63] XU X, LIU C, FENG Q, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]// Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017:363-376.

[64] ZOU Y, BAN B, XUE Y, et al. CCGraph: a PDG-based code clone detector with approximate graph matching[C]// 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2020:931-942.

[65] KONTOGIANNIS, KOSTAS A. Pattern matching for clone and concept detection. Reverse engineering[J]. Automated Software Engineering, 1996, 3(1):77-108.

[66] SHAWKY D M, ALI A F. An approach for assessing similarity metrics used in metric-based clone detection techniques[C]// 2010 3rd International Conference on Computer Science and Information Technology. 2010:580-584.

[67] YUAN Y, GUO Y. CMCD: Count matrix based code clone detection[C]// 2011 18th Asia-Pacific Software Engineering Conference. 2011:250-257.

[68] SAINI V, FARMAHINIFARAHANI F, LU Y, et al. Oreo: Detection of clones in the twilight zone[C]// Proc. ESEC/FSE. 2018:354-365.

[69] DAVEY N, BARSON P, FIELDS, et al. The development of a software clone detector[J]. International Journal of Applied Software Technology, 1995, 1(3/4):219-236.

[70] ABD-EL-HAFIZ S K. A metrics-based data mining approach for software clone detection[C]// 2012 IEEE 36th Annual Computer Software and Applications Conference. 2012:35-41.

[71] LI, SUN J L. A metric space based software clone detection approach[C]// IEEE International Conference on Information Management & Engineering. 2010:393-397.

[72] RAGKHITWETSAGUL C, KRINKE J, MARNETTEB. A picture is worth a thousand words: Code clone detection based on image similarity[C]// 2018 IEEE 12th International Workshop on Software Clones (IWSC). 2018:44-50.

[73] KIM H, JUNG Y, KIM S, et al. MeCC: memory comparison-based clone detector[C]// Proceedings of the33rd International Conference on Software Engineering (ICSE 2011). 2011: 301-310.

[74] GRANT S, CORDYJ R. Vector space analysis of software clones [C]// 2009 IEEE 17th International Conference on Program Comprehension. 2009:233-237.

[75] WAHLER V, SEIPEL D, WOLFF J, et al. Clone detection in source code by frequent itemset techniques[C]// 4th IEEE International Workshop on Source Code Analysis and Manipulation. 2004:128-135.

[76] FANG C, LIU Z, SHI Y, et al. Functional code clone detection with syntax and semantics fusion learning[C]// Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020:516-527.

[77] TUFANO M, WATSON C, BAVOTA G, et al. Deep learning similarities from different representations of source code[C]// 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR). 2018:542-553.

[78] HARTMANN B, MACDOUGALL D, BRANDT J, et al. What would other programs do? Suggesting solutions to error messages[C]// Proc. SIGCHI Conference on Human Factors in Computing Systems. 2010:1019-1028.

[79] KIM S, WOO S, LEE H, et al. VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery[C]// 2017 IEEE Symposium on Security and Privacy (SP). 2017:595-614

[80] MALETIC J, MARCUS A. Supporting program comprehension using semantic and structural information[C]// ICSE '01. 2001: 103-112.

[81] BRUNTINK M, VAN DEURSEN A, VAN ENGELENR T, et al. On the use of clone detection for identifying crosscutting concern code[J]. IEEE Transactions on Software Engineering, 2005, 31(10):804-818.

[82] ZIBRAN M F, ROY C K. IDE-based real-time focused search for near-miss clones[C]// Proc. of the 27th Annual ACM Symposium on Applied Computing. 2012:1235-1242.

[83] HIGO Y, KAMIYA T, KUSUMOTO S, et al. ARIES: Refactoring support environment based on code clone analysis[C]// IASTED Conf. on Software Engineering and Applications. 2004: 222-229.

[84] ZIBRAN M F, ROYC K. A constraint programming approach to conflict-aware optimal scheduling of prioritized code clone refactoring[C]// 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation. 2011:105-114.
[85] DING S H H, FUNG B C M, CHARLAND P. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization[C]// 2019 IEEE Symposium on Security and Privacy (SP). 2019:472-489.
[86] KAWAGUCHI S, YAMASHINA T, UWANO H, et al. Shinobi: A tool forutomatic code clone detection in the ide[C]// 2009 6th Working Conference on Reverse Engineering. 2009:313-314.
[87] NARASIMHAN K. Clone merge an eclipse plugin toabstract near-clone c++ methods[C]// 2015 30thIEEE/ACM International Conference on Automated Software Engineering (ASE). 2015:819-823.
[88] TOOMEY W. Ctcompare: Code clone detection usinghashed token sequences[C]// 2012 6th InternationalWorkshop on Software Clones (IWSC). 2012:92-93.
[89] SVAJLENKO J, ROY C K. Cloneworks: A fast andflexible large-scale near-miss clone detection tool[C]// 2017 IEEE/ACM 39th International Conference onSoftware Engineering Companion (ICSE-C). 2017:177-179.
[90] ABID S, JAVED S, NASEEM M, et al. Codeease: harnessing method clonestructures for reuse[C]// 2017 IEEE 11th International Workshop on Software Clones (IWSC). 2017:1-7.
[91] SEMURA Y, YOSHIDA N, CHOI E, et al. Ccfindersw: Clone detection tool with flexiblemultilingual tokenization[C]// 2017 24th Asia-PacificSoftware Engineering Conference (APSEC). 2017:654-659.
[92] CORDY J R, ROY C K. The nicad clone detector[C]// 2011 IEEE 19th International Conference on Program Comprehension. 2011:219-220.
[93] XING Z, XUE Y, JARZABEK S. Clonedifferentiator: Analyzing clones by differentiation[C]// 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). 2011:576-579.
[94] KAMIYA T. Agec: An execution-semantic clonedetection tool [C]// 2013 21st International Conferenceon Program Comprehension (ICPC). 2013:227-229.
[95] VISLAVSKI T, RAKÍ C G, CARDOZO N, et al. Licca: A tool for cross-language clone detection[C]// 2018 IEEE 25th International Conference on SoftwareAnalysis, Evolution and Reengineering (SANER). 2011:512-516.
[96] MOSTAEEN G, SVAJLENKO J, ROY B, et al. on the use ofmachine learning techniques towards the design ofcloud based automatic code clone validation tools[C]// 2018 IEEE 18th International Working Conference onSource Code Analysis and Manipulation(SCAM). 2018:155-164.
[97] TANG W, CHEN D, LUO P. Bcfinder: Alightweight and platform-independent tool to findthird-party components in binaries [C]// 2018 25th Asia-Pacific Software Engineering Conference (APSEC). 2018:288-297.
[98] KRUTZ D E, SHIHAB E. Cccd: Concolic code clonedetection [C]// 2013 20th Working Conference onReverse Engineering (WCRE). 2013:489-490.
[99] HUMMEL B, JUERGENS E, HEINEMANN L, et al. Index-based code clone detection: incremental, distributed, scalable [C]// 2010 IEEE International Conference on Software Maintenance. 2010:1-9.
[100] BIEGEL B, DIEHL S. JCCD: a flexible and extensible API for implementing custom code clone detectors[C]// Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. 2010:167-168.
[101] MOSTAEEN G, SVAJLENKO J, ROY B, et al. Clonecognition: Machine learning based code clone validation tool[C]// Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE, 2019:1105-1109.
[102] AVERSANO L, CERULO L, DI PENTAM. How clones are maintained: An empirical study[C]// 11th European Conference on Software Maintenance and Reengineering (CSMR'07). 2007:81-90.
[103] DANG Y, KHAN S, ZHANG D, et al. Code clone notification and architectural change visualization: U. S[J]. Patent Application 12/972,535. 2012.
[104] KIM M, SAZAWAL V, NOTKIND, et al. An empirical study of code clone genealogies[C]// Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2005:187-196.
[105] NAFI K W, KAR T S, ROY B, et al. CLCDSA: cross Language code clone detection using syntactical features and API documentation[C]// 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2019:1026-1037.
[106] MATHEW G, PARNIN C, STOLEE K T. SLACC: simion-based language agnostic code clones[C]// Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. 2020:210-221.
[107] ZOU X. A survey on application of knowledge graph. Journal of Physics: Conference Series[J]. IOP Publishing, 2020, 1487(1): 012016.
[108] ANG Y, ZHANG D, GE S, et al. Transferring code-clone detection and analysis to practice[C]// Proc. ofthe 39th IEEE/ACM Int'l Conf. on Software Engineering in Software Engineering in Practice Track (ICSE-SEIP). 2017, 34(6):53-62.
[109] ABID S. Recommending related functions from API usage-based function clone structures[C]// Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019:1193-1195.
[110] ARULKUMARAN K, DEISENROTH M P, BRUNDAGE M, et al. Deep reinforcement learning: A brief survey[J]. IEEE Signal Processing Magazine, 2017, 34(6):26-38.

LE Qiao-yi, born in 1997, postgraduate, is a member of China Computer Federation. His main research interests include code big data and code clone detection.

LIU Jian-xun, born in 1970, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include big data, bussiness intelligence, service computing and cloud computing.