计算机科学
COMPUTER SCIENCE

二进制代码相似性检测方法综述

魏有缘, 宋建华, 张龑

引用本文

魏有缘, 宋建华, 张龑. 二进制代码相似性检测方法综述[J]. 计算机科学, 2025, 52(6): 365-380.

WEI Youyuan, SONG Jianhua, ZHANG Yan. Survey of Binary Code Similarity Detection Method[J]. Computer Science, 2025, 52(6): 365-380.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.240400003

二进制代码相似性检测方法综述

魏有缘[1] 宋建华[1,3,4] 张 龑[2,3]
1 湖北大学网络空间安全学院 武汉 430062
2 湖北大学计算机与信息工程学院 武汉 430062
3 智能感知系统与安全教育部重点实验室 武汉 430062
4 智能网联汽车网络安全湖北省工程研究中心 武汉 430062
(202221116012664@stu.hubu.edu.cn)

摘 要 代码相似性检测按照研究对象可分为源代码相似性检测和二进制代码相似性检测两种,常用于恶意代码识别、漏洞搜索、版权保护等场景。基于目前国内的互联网环境,程序通常以二进制文件的形式发布,大多数程序都无法直接获得源代码,因此在软件安全领域的相关研究中,二进制代码相似性检测的应用范围相对更广。从二进制代码相似性检测的定义和实现流程出发,按照代码表征形式将其分为基于文本字符、基于代码嵌入、基于图嵌入三大类,对经典的二进制代码相似性检测方法和近5年的新方法共19篇文献进行了整理,并根据多架构、Baseline、基准数据集和检测性能对各类方法进行了分析和总结。最后,结合新方法的发展分析了当前存在的问题和未来可能的研究方向。

关键词:二进制代码相似性检测;代码表征;软件安全;恶意代码识别;漏洞搜索
中图分类号 TP311.5

Survey of Binary Code Similarity Detection Method

WEI Youyuan[1],SONG Jianhua[1,3,4] and ZHANG Yan[2,3]
1 School of Cyber Science and Technology,Hubei University,Wuhan 430062,China
2 School of Computer Science andInformation Engineering,Hubei University,Wuhan 430062,China
3 Key Laboratory of Intelligent Sensing System and Security(Hubei University),Ministry of Education,Wuhan 430062,China
4 Hubei Provincial Engineering Research Center of Intelligent Connected Vehicle Network Security,Wuhan 430062,China

Abstract Code similarity detection can be divided into two types according to the research object:source code similarity detection and binary code similarity detection,which are commonly used in scenarios such as malicious code identification,vulnerability search,and copyright protection. Based on the current domestic Internet environment,programs are usually released in the form of binary files,and most programs cannot directly obtain source code. Therefore,in related research in the field of software security,the application scope of binary code similarity detection is relatively wider. Starting from the definition and implementation process of binary code similarity detection,according to the code representation form,it is divided into three categories:text character-based,code embedding-based,and graph embedding-based. The classic binary code similarity detection methods and the recent five years of research and development are compared. A total of 19 documents on new methods are sorted out,and various methods are analyzed and summarized based on multi-architecture,Baseline,benchmark datasets and detection performance. Finally,current problems and possible future research directions are analyzed based on the development of new methods.

Keywords Binary code similarity detection,Code representation,Software security,Malicious code identification,Vulnerability search

到稿日期:2024-04-01 返修日期:2024-10-03
基金项目:国家自然科学基金(62377009);湖北省重大攻关项目(JD)(2023BAA018);湖北省重点研发计划重点项目(2021BAA188,2021BAA184);湖北省高等学校人文社会科学重点研究基地绩效评价信息管理研究中心课题(2020JX01)
This work was supported by the National Natural Science Foundation of China(62377009),Major Program(JD) of Hubei Province(2023BAA018),Key R&D program of Hubei Province(2021BAA184,2021BAA188) and Hubei Province Project of Key Research Institute of Humanities and Social Sciences at Universities(Research Center of Information Management for Performance Evaluation)(2020JX01).
通信作者:宋建华(sjhhubu@126.com)

1 研究背景及意义

目前互联网上各种开源代码框架日益成熟，为了提高效率，在软件开发过程中也开始大量地使用互联网上的开源代码，即在软件开发过程中由于对源代码复用、修改、重构产生的文本相似或结构相似的代码使用，这种行为也被称为代码克隆[1](Code Clone)。这些开源代码不一定都经过了严格的筛选和检查，因此会对软件开发产生安全隐患，假如某段代码被人恶意留下后门，可能会危及用户的隐私安全，甚至造成社会危害。除此之外，未经许可的代码克隆可能还会涉及版权纠纷，导致原作者的合法权益受到侵犯。

2023 年 6 月 22 日，由 Cisco Talos 披露的 VMware DCERPC 调用请求未初始化内存堆溢出漏洞 CVE-2023-20892[2]被曝出。VMware vCenter Server 7.0.3.01000 中使用的开源 DCERPC 库的请求处理功能中存在堆溢出漏洞，可能导致使用未初始化的内存，使得不法分子可以利用此漏洞执行恶意代码，对系统和用户的安全造成威胁，而仅仅使用了开源 DCERPC 库代码的 Apple macOS 12.6.1 同样会受到影响。在软件安全领域，通过分析代码中的词法、语法、语义信息等，计算两段代码之间的相似性来快速识别漏洞代码，检测代码是否安全，这种方法就是代码相似性检测(Code Similarity Detection)。代码相似性检测是近年来软件安全分析技术中的重要一步，其根据代码特征检测代码中存在的恶意代码、已知漏洞或病毒，能有效降低代码被植入后门的风险。

人们生活中绝大多数的移动端或桌面端应用都是以二进制可执行文件的形式发布的，通常无法直接获得源代码，因此代码相似性检测也分为源代码相似性检测(Source Code Similarity Detection，SCSD)和二进制代码相似性检测(Binary Code Similarity Detection，BCSD)，二者的区别在于，源代码相似性检测是对源代码进行克隆检测，而二进制代码相似性检测是对二进制可执行文件进行反汇编后的机器代码进行克隆检测。程序的源代码和二进制代码在形式上通常具有较大的差异：源代码更多是由高级语言编写，如 C++，Python，Java 等，因此包含的程序信息更加丰富，且具备良好的移植性；二进制代码通常指二进制程序的反汇编代码，其没有源代码那样丰富的程序信息，在编译的过程中根据不同的编译器可能会有不同的优化策略，根据不同的平台也会对应不同的指令架构，这些干扰都使得二进制代码相似性检测的检测难度高于源代码相似性检测。

本文着重于对二进制代码相似性检测的研究，第 2 章简要介绍了二进制代码相似性检测概况，包括一些基础定义和相关术语的介绍；第 3 章主要介绍了二进制代码相似性检测的一般流程以及特征向量的相似性计算方法；第 4 章通过不同的代码表征形式，将二进制代码相似性检测的方法分为基于文本字符、基于代码嵌入和基于图嵌入 3 大类，并对相关方法进行了说明和汇总，同时按照不同的分类方法，对代码表征方法进行统计分析，包括多架构、Baseline 方法、基准数据集和性能分析等；第 5 章结合当前的发展状况，对当前面临的难题和未来可能的研究方向进行了总结。

2 基础概念

2.1 相关术语及其定义

2.1.1 基本块

一个二进制程序可以被划分为很多个基本块，每个基本块都由一组没有分支且能够最大限度地按顺序执行的指令序列组成。一个基本块通常只有一个入口和出口，入口通常对应基本块的第一条语句，出口则对应最后一条语句，一般为 jmp，ja，jz 等跳转指令。执行基本块中的代码时，只能从入口处按顺序依次执行到出口处。

图 1 为基本块的样例。

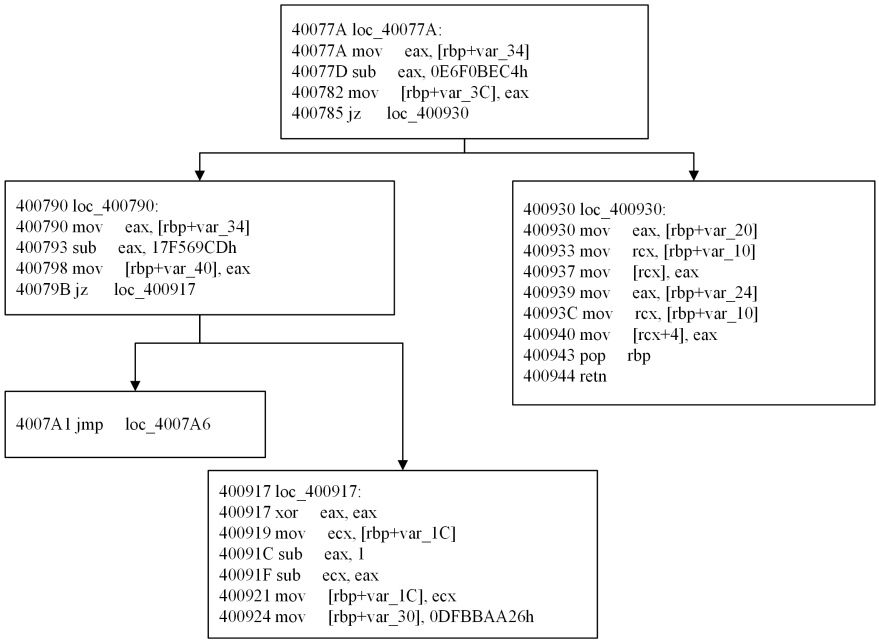


图 1 基本块的样例

Fig. 1 Example of basic block

2.1.2 代码片段和代码块

代码片段[3](Clone Fragment，CF)是一种模块化的设计，通常由多条具有实际意义的代码语句组成，属于原始代码的一个子集。

代码块(Code Block)是原始代码的一个组成部分，通常包含多行代码，由一对大括号{}包围，并且有明确的开始和结束。

2.1.3 二进制代码

与程序员在日常开发中使用的高级编程语言(如 C++，Java，Python 等)编写的源代码不同，二进制代码是计算机硬件直接能够理解和执行的代码形式[4]，通常用二进制中的“0”和“1”来表示，但通常对人类来说是不可读的。在二进制代码相似性检测的相关研究中，通常不是直接分析“0”和“1”形式的二进制代码，而是对二进制文件反汇编后，再对汇编代码进行分析。

一般情况下，源代码可以通过编译器或解释器转换为二进制代码[5]，但想要将二进制代码完全转换为源代码则是非常复杂和困难的。在源代码编译链接的过程中，很多高级编程语言的信息、结构和注释都被转换、优化或丢失，导致源代码被翻译成二进制代码的过程几乎是不可逆的[6]。目前有许多具有代表性的反汇编工具，如 IDA Pro，.NET Reflector，jadx-gui 等，虽然它们可以为二进制代码转换为源代码提供一些帮助，但想要还原出原始的、准确的源代码依然是极其困难的。

日常生活中的绝大部分程序无法直接获取源代码。由于编程语言种类众多，且在不同系统平台、不同指令集架构下的程序往往会有很大的区别[7]，从源代码角度进行相似性分析难度较大，但不同编程语言下的二进制代码通常都是比较类似的，因此二进制代码更适合跨编程语言、跨系统平台和跨指令集架构的代码相似性检测。图 2 为源代码到二进制可执行文件的编译过程。

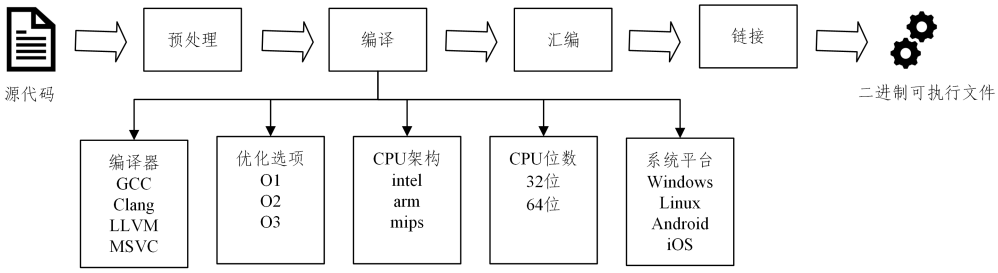


图 2 源代码到二进制可执行文件的编译过程

Fig. 2 Compilation process from source code to binary executable file

2.1.4 图相关结构

为了更好地利用二进制程序中具有语义信息的代码特征，同时考虑程序实际执行过程中的信息，通常会使用图相关的数据结构。二进制程序中比较有代表性且常用的图结构[8]有控制流图(Control Flow Graph，CFG)、函数调用图(Function-Call Graph，FCG)和程序依赖图(Program Dependency Graph，PDG)等。

控制流图是对程序控制流程的图形化表示。如果将程序看成由多个函数构成，则可以将程序中某个函数的控制流图视为由节点集合和边集合组成的图结构，其中每条节点代表一个语句或条件判断，每条边代表两个节点之间可能的控制流。

函数调用图是对程序中子程序或函数之间的调用关系的图形化表示。函数调用图是一个有向图，图中的节点通常有两类：一类是由程序设计者实现的本地函数，另一类则是外部函数或者系统、库函数。这些节点的集合代表程序中的所有函数，其中每一个节点代表程序中的一个具体的函数；而边的集合则代表程序中函数之间的调用关系，其中每一条有向边代表一个函数调用。

程序依赖图是对程序的控制依赖和数据依赖关系的图形化表示。程序依赖图是一个以控制流图为基础的、带有标记的有向多重图，其节点集合与控制流图中的节点集合相同，每个节点代表一个语句或条件判断；而在边的集合中，每一条边代表节点之间的控制依赖或数据依赖。

2.1.5 代码表征

二进制可执行文件是源代码进行预处理、编译、汇编、链接后的产物，通常无法直接进行代码相似性比较。使用代码表征[9](Code Representation)可以很好地解决这个问题。代码表征的主要目的是对代码进行数值化，即将源代码或汇编代码通过某种方式进行转换，形成一种能够用于比较、分析或学习的表示形式，通常为某种数据结构或特征向量。代码表征的优点在于，可以抽象化地体现出代码片段的语法特征或语义特征，例如将代码片段映射为一个多维空间向量，从而隐藏具体的语法和细节，最后根据代码表征形式使用不同的方法来进行相似性计算。

根据对代码信息的不同利用程度，可以将代码表征分为 4 个类别，表 1 列出了常见的 4 种代码表征形式。

表 1 常见的 4 种代码表征形式

Table 1 Four common code representation forms

分类	解释
基于文本的代码表征	将代码片段直接作为文本进行处理，代码被视为一串字符或单词的序列，通过标记、分词等方法进行代码表征，如后缀树(Suffix Tree)字符串匹配算法
基于词汇的代码表征	对代码进行词法分析，将代码片段表示为符号序列，通过符号序列构建代码表征，如语义 token
基于语法的代码表征	使用代码的语法结构来构建代码表征，抽象语法树(AST)就是一种常见的语法表征方式
基于语义的代码表征	将代码表示为具有语义信息的代码表征，通过对代码进行静态分析、数据流分析等，提取代码的语义信息，常见的提取信息有控制流图(CFG)等

2.2 代码克隆分类

二进制代码相似性检测不仅需要检测代码在文本层面的相似性，而且还要考虑代码在功能层面的相似性。例如，两段

代码使用的编程语言不同，并且使用了不同的语法结构、变量名或算法，但是它们解决了相同的问题并产生了相似的结果，就可以称其为功能层面的相似。代码在功能层面上的相似通常难以界定为代码抄袭，但在恶意代码检测、病毒样本识别等领域却有着重大意义，因为无论是恶意代码还是病毒，即使实现方式不同，它们的目的都是类似的：信息窃取、数据破坏或窜改、远程控制和后门等，所以它们通常具有相似的特征。

为了划分代码的相似程度，Bellon 等[10]定义了 Type-1 到 Type-4 这 4 种代码克隆类型来衡量代码的相似度。表 2 具体列出了 4 种代码克隆类型的定义。

从 Type-1 型到 Type-4 型克隆，两个代码片段间的差异越来越大，相似性检测的难度也逐渐增加。根据目前的研究结果，对于 Type-1 型和 Type-2 型克隆的相似性检测已经可以达到较好的效果，而对于 Type-3 型和 Type-4 型克隆的检测性能则有待提高。

从相似的范围来看，Type-3 型克隆的定义较为广泛，不仅包括对代码的增删改，还包括 Type-1 型和 Type-2 型克隆中的改变，但对标识符名称、类型名、空格和布局等修改的比例也没有做出明确限定[11]；而 Type-4 型克隆的定义则包含更多的可能，从而大大提高了相似性检测的难度。从代码的结构来看，Type-1 型、Type-2 型和 Type-3 型克隆都是语法层面的相似，而 Type-4 型克隆则是语义层面的相似。

表 2 代码克隆类型

Table 2 Code clone types

克隆类型	定义
Type-1 型克隆	两个代码片段仅仅只有空白部分、代码注释和代码的排版、布局发生了改变，其他部分完全相同
Type-2 型克隆	两个代码片段仅仅只有标识符名称、类型名和函数名发生了改变，代码结构和语法等其他部分都完全相同
Type-3 型克隆	允许对代码片段进行语句的添加、删除及修改，或者代码片段在标识符名称、类型名、空格、布局和注释方面进行不同程度的改变，其他部分完全相同
Type-4 型克隆	两个代码片段实现了相同或相似的功能，但在文本层面上不相似，语法结构和实现过程也不尽相同

图 3 给出了 4 种克隆类型的样例。与原始代码片段相比，在 Type-1 型克隆中，仅仅删去了注释 1 与部分空格；在 Type-2 型克隆中，将变量 i,a,b 更换为 j,c,d，其余部分不变；在 Type-3 型克隆中，删除了注释 2，同时增加了两个 puts() 操作和 count 计数器；在 Type-4 型克隆中，将 for 循环改写为 while 循环，并使用了不同的方法来实现与原本相同的功能。

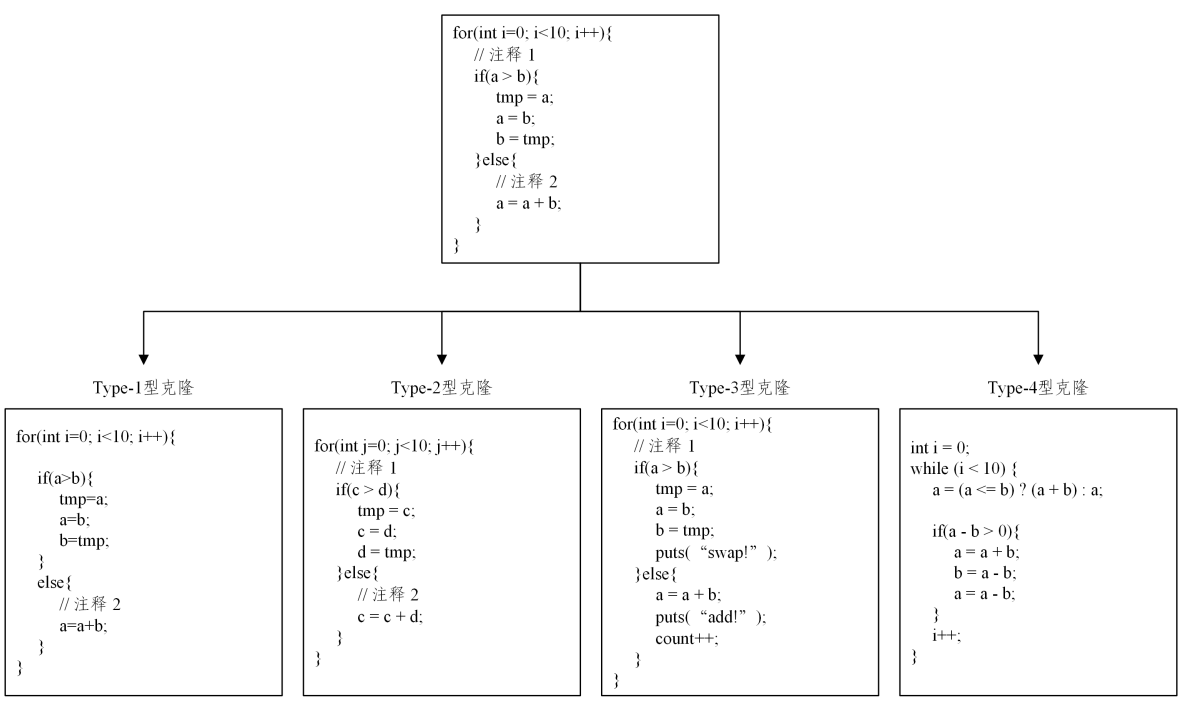


图 3 4 种克隆类型的样例

Fig. 3 Examples of four clone types

2.3 相似性评价指标

为了精准地衡量相似性检测模型的性能，研究者主要提出了 4 种评价指标[12]来量化实验方法的实际效果：精确率(Precision)、召回率(Recall)、准确率(Accuracy)、F_1 分数(F_1-Score)。这些评价指标的计算方式都与预测类别以及实际类别有关，预测类别为正表示被模型预测为代码克隆，实际类别为正则表示实际客观存在代码克隆。

按照预测类别和实际类别的不同情况可以分为 4 种类型：真阳性(True Positive，TP)、假阳性(False Positive，FP)、真阴性(True Negative，TN)和假阴性(False Negative，FN)。表 3 详细介绍了 TP，FP，TN，FN 这 4 种术语的具体含义。

表 3 TP，FP，TN，FN 的含义

Table 3 Meaning of TP，FP，TN and FN

术语	预测类别	实际类别	解释
真阳性(TP)	正	正	模型正确地将正类别样本分类为正类别
假阳性(FP)	正	负	模型错误地将负类别样本分类为正类别
真阴性(TN)	负	负	模型正确地将负类别样本分类为负类别
假阴性(FN)	负	正	模型错误地将正类别样本分类为负类别

精确率又称查准率，指预测类别为正且实际类别也为正的样本数量在所有预测类别为正的样本数量中所占的比例。

由于计算精确率的样本总数是所有预测类别为正的样本的数量，而预测样本往往带有主观性，因此精确率可以看成是从主观的角度来展示模型预测的效果。精确率如式(1)所示：

$$Precision=\frac{TP}{TP+FP} \tag{1}$$

召回率又称查全率，指预测类别为正且实际类别也为正的样本数量在所有实际类别为正的样本数量中所占的比例。由于计算召回率的样本总数是所有实际类别为正的样本的数量，而实际样本往往具备客观性，因此召回率可以看成从客观的角度来展示模型预测的效果。召回率如式(2)所示：

$$Recall=\frac{TP}{TP+FN} \tag{2}$$

准确率指模型在所有样本中正确分类的比例，即正确预测的样本数占总样本数的比例。准确率和精确率是两个不同的概念，准确率是全局性能指标，它衡量了模型对所有类别的分类准确程度；而精确率是局部性能指标，它关注的是模型在正类别上的预测准确性，即模型预测为正类别的样本中有多少是真正的正类别。准确率如式(3)所示：

$$Accuracy=\frac{TP+TN}{TP+FN+FP+TN} \tag{3}$$

F_1分数又称平衡 F 分数(Balanced F Score)。由于在实际应用中，精确率和召回率通常是一对相互制衡的指标，提高精确率可能会降低召回率，反之亦然。为了综合考虑这两个指标，F_1分数被定义为精确率和召回率的调和平均数，它的取值通常在 0～1。根据调和平均数的定义。F_1分数如式(4)所示：

$$\begin{aligned} F_1\text{-}Score &=\left(\frac{Precision^{-1}+Recall^{-1}}{2}\right)^{-1} \\ &=2\cdot\frac{Precision\cdot Recall}{Precision+Recall} \end{aligned} \tag{4}$$

有时为了突出模型的偏好，会赋予精确率和召回率不同的权重，于是将平衡 F 分数定义为如式(5)所示的通用形式：

$$F_\beta\text{-}Score=(1+\beta^2)\cdot\frac{Precision\cdot Recall}{\beta^2\cdot Precision+Recall} \tag{5}$$

当 $\beta=1$ 时，式(5)就变为F_1分数；当 $\beta\neq1$ 时，模型的偏好由 β 的取值范围决定，$\beta>1$ 时精确率占据的权重更大，$\beta<1$ 时召回率占据的权重更大。

3 二进制代码相似性检测

3.1 二进制代码相似性检测实现流程

Whale[13]在 1988 年最早提出了代码相似性检测的流程，他将代码相似性检测分为代码格式转换和相似度确定两个阶段。随着相关研究的不断发展，机器学习和深度学习等相关技术也逐渐被运用到二进制代码相似性检测领域，目前基本可以划分为 3 个阶段：预处理、代码表征和相似性计算。图 4 为二进制代码相似性检测的基本流程。

在预处理阶段，主要包括初始化和特征提取两个步骤。初始化需要剔除与相似性计算无关的信息，同时确定相似性检测的粒度；特征提取则是通过 IDA Pro 等工具进行逆向分析，提取二进制代码中的一些适用于相似性计算的典型特征，为了提高模型的准确性，通常会选取一些具有明显差异性的特征来分析。

代码表征是二进制代码相似性检测中的重要一步。为了更好地进行相似性比较，预处理阶段提取的代码特征需要选择合适的方法进行代码表征。在基于文本字符的表征中，通常将代码片段看作文本，指令、操作数或代码语句都是文本中的字符串；在基于代码嵌入(Code Embedding)的表征中，通常利用自然语言处理(Natural Language Processing，NLP)相关方法或 BERT 等预处理模型将代码特征表征为空间向量；在基于图嵌入(Graph Embedding)的表征中，通常使用二进制函数之间的控制流图、函数调用图、程序依赖图等图结构来提取关键特征，并将其表征为空间向量。

在相似性计算阶段，虽然模型选取和计算方法不是唯一的，但通常都是利用处理后的两个代码表征之间的相似性来衡量两个代码片段之间的相似性。在基于文本字符的方法中，通常利用字符串匹配算法来衡量相似性，而随着机器学习与深度学习的引入，大多数方法都选择空间向量作为代码表征，这种空间向量也称特征向量(Feature Vector)，最后通过计算特征向量之间的相似性来衡量代码片段的相似性。

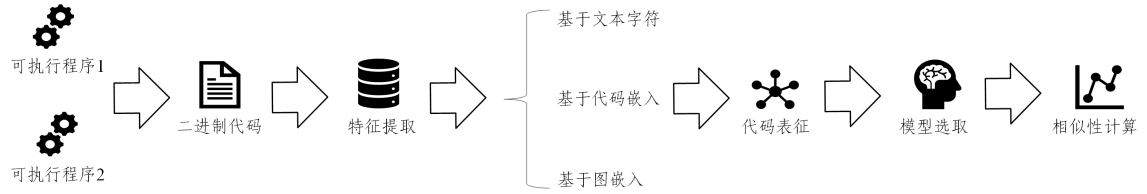


图 4 二进制代码相似性检测的基本流程

Fig. 4 Basic process of binary code similarity detection

3.2 特征向量的相似性计算方法

特征向量的相似性计算方法主要有闵可夫斯基距离(Minkowski Distance)、余弦相似度(Cosine-Similarity)和点积相似度(Dot Product Similarity)等。

闵可夫斯基距离是度量空间中两个向量之间距离的这一类方法的统称，表达式为：

$$d_{x,y}=\sqrt[r]{\sum_{i=1}^{n}|x_i-y_i|^r} \tag{6}$$

当 $r=1$ 时，称为曼哈顿距离(Manhattan Distance)，其几何意义为一维坐标轴上两个点之间的距离，表达式为：

$$d_{x,y}=\sum_{i=1}^{n}|x_i-y_i| \tag{7}$$

当 $r=2$ 时，称为欧氏距离(Euclidean Distance)，其几何意义为二维坐标系中两点之间的距离，表达式为：

$$d_{x,y}=\sqrt{\sum_{i=1}^{n}|x_i-y_i|^2} \tag{8}$$

当 r 趋近于∞时，称为切比雪夫距离(Chebyshev Distance)，该方法运将了极限的思想，用空间中所有的点之间的最长距离来代表两点之间的距离，表达式为：

$$d_{x,y}=\sqrt[\infty]{\sum_{i=1}^{n}|x_i-y_i|^{\infty}}=\max(|x_i-y_i|) \tag{9}$$

余弦相似度与闵可夫斯基距离不同，它并不是距离的度量，其几何含义是两个向量在空间中的夹角的余弦值，取值范围为[−1,1]。如果两个向量的方向完全相同则为 1，方向完全相反则为−1。余弦相似度只关心向量之间的角度而不关心向量的大小，因此不会受到向量长度变化的影响。图 5 为余弦相似度的几何表示，表达式如下：

$$\begin{aligned}\cos\theta&=\frac{x\cdot y}{|x|\cdot|y|}\\&=\frac{x_1\cdot y_1+x_2\cdot y_2+\cdots+x_n\cdot y_n}{\sqrt{x_1^2+x_2^2+\cdots+x_n^2}\cdot\sqrt{y_1^2+y_2^2+\cdots+y_n^2}}\\&=\frac{\sum_{i=1}^{n}x_i\cdot y_i}{\sqrt{\sum_{i=1}^{n}x_i{}^2}\cdot\sqrt{\sum_{i=1}^{n}y_i{}^2}}\end{aligned} \tag{10}$$

点积相似度与余弦相似度类似，它们都不是距离的度量。点积相似度的几何含义是两个向量的长度与它们之间夹角的余弦值的乘积，取值范围为$-\infty\sim+\infty$，两个向量的方向相反为负，方向相同则为正。点积相似度既考虑了向量的大小，又考虑了向量之间的角度，其表达式为：

$$\begin{aligned}Dot(x,y)&=\sum_{i=1}^{n}x_i\cdot y_i=|x|\cdot|y|\cdot\cos\theta\\&=\sqrt{\sum_{i=1}^{n}x_i{}^2}\cdot\sqrt{\sum_{i=1}^{n}y_i{}^2}\cdot\cos\theta\end{aligned} \tag{11}$$

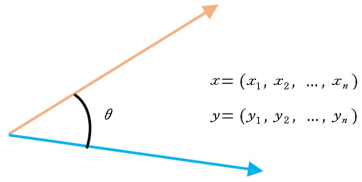


图 5 余弦相似度的几何表示

Fig. 5 Geometric representation of cosine similarity

4 二进制代码相似性检测方法的综合分析

本章按照代码表征形式对二进制代码相似性检测方法进行了系统的文献综述，这些方法的主题都是二进制代码相似性检测，其中包含该领域内的经典方法约 9 篇，以及近 5 年的新方法 10 篇。Genius 和 Gemini 作为图嵌入的代表性方法，常被用作实验的 Baseline，而将 Gemini 作为 Baseline 的经典开源方法 VulSeeker 则被不少实验忽视；SAFE 和 Asm2Vec 同样被众多方法当作 Baseline，是代码嵌入中的代表性方法，而 INNEREYE 和 αDiff 作为代码嵌入中的经典方法同样被忽视。这些被忽视的经典方法在未来的研究中应适当引起重视。

表 4 对筛选的文献进行了汇总，包括方法名称、发表时间、在文章中的引用编号、论文题目等信息。本章将以这些文献作为研究对象来进行具体的分类分析。

表 4 本文筛选的文献

Table 4 Literature screened for this paper

表征形式	方法/作者	时间	题目
文本字符	GitZ	2017	*Similarity of Binaries through re-Optimization*
	Zhang 等	2019	*Clone Detection Algorithm for Binary Executable Code with Suffix Tree*
代码嵌入	INNEREYE	2018	*Neural machine translation inspired binary code similarity comparison beyond function pairs*
	SAFE	2019	*Safe: Self-attentive function embeddings for binary similarity*
	Asm2Vec	2019	*Asm2vec: boosting static representation robustness for binary clone search against code obfuscation and compiler optimization*
	αDiff	2018	*adiff: cross-version binary code similarity detection with dnn*
	jTrans	2022	*jTrans: jump-aware transformer for binary code similarity detection*
	Jump-SBERT	2023	*Research on Binary Code Similarity Detection Based on Jump-SBERT*
	Wang 等	2023	*Binary Code Similarity Detection Method Based on Pre-training Assembly Instruction Representation*
	Li 等	2023	*Binary code similarity detection via attention mechanism and Child-Sum Tree-LSTM*
	FastBCSD	2023	*FastBCSD: Fast and Efficient Neural Network for Binary Code Similarity Detection*
	CEBin	2024	*CEBin: A Cost-Effective Framework for Large-Scale Binary Code Similarity Detection*
	BBTree	2023	*A binary code similarity analysis method based on code embedding*
图嵌入	Genius	2016	*Scalable Graph-based Bug Search for Firmware Images*
	Gemini	2017	*Neural network-based graph embedding for cross platform binary code similarity detection*
	VulSeeker	2018	*VulSeeker: a semantic learning based vulnerability seeker for cross-platform binary*
	IFAttn	2022	*IFAttn: Binary code similarity analysis based on interpretable features with attention*
	CI-Detector	2024	*Cross-Inlining Binary Function Similarity Detection*
	Kinable 等	2010	*Malware Classification based on Call Graph Clustering*

4.1 基于代码表征形式的检测方法

4.1.1 基于文本字符的表征

基于文本字符的表征通常在将二进制可执行程序进行反汇编得到汇编代码后，将代码片段视作文本，而操作指令和操作数则看作由一个个文本字符组成的字符串或字符序列。因此，比较两个代码片段的相似性，就等价于比较两个文本中的字符串的相似性。这类方法通常也会进行预处理，例如将反汇编后的代码划分为寄存器、内存地址、操作码和操作数等信息并提取，然后使用字符串模式匹配算法来进行字符串的相似性比对，如后缀树[14-15]等；或者根据基本块中的指令序列生成哈希值，利用哈希比较[16](Hash Comparison)进行相似性检测。这种表征方法由于没有考虑代码的语义信息，仅仅是针对代码片段进行文本层面的处理，因此通常只针对 Type-1 型、Type-2 型和 Type-3 型的代码克隆的相似性检测。表 5 为基于文本字符的表征方法的汇总。

David 等在 2017 年提出了一种可扩展的方法 GitZ[16]来计算二进制文件之间的相似性，这种方法将二进制文件分解为可比较的基本单元，即基本块数据流切片[17](Data-Flow Slice)，忽略了汇编代码中的操作数，使用 LLVM-IR 作为中间表示。为了使 GitZ 更好地支持多架构，其使用一种强大的中间表示形式 VEX-IR[18]来表示不同架构的二进制代码，并提供了一种将 VEX-IR 转换为 LLVM-IR 的转换引擎。在对

LLVM-IR 进行标准化处理后，将其索引为 MD5 哈希集，最后基于成对的哈希比较操作和全局上下文的应用来计算相似性得分。GitZ 提供了一种基于哈希的相似性计算方法，但像 LLVM-IR 这样的标准化处理并非每次都能够建立，因此后续工作可以尝试对该过程进行优化。

Zhang 等[19]于 2019 年提出了一种基于后缀树的二进制代码相似性检测方法，这种方法首先通过对二进制可执行程序进行反汇编得到指令序列；同时，为了减小源代码编译生成二进制可执行程序时由编译器的优化策略造成的影响，按照不同的操作类型将指令序列进行分类，然后用这些指令类型序列来构建后缀树，进而利用后缀树能高效处理字符串问题的特点来进行字符串匹配。由于考虑了汇编指令的语义信息，因此该方法对 Type-1 型到 Type-4 型克隆的相似性检测均有较好的表现。通过在 MIPS32 指令集的代码样本上进行测试，该方法在 Type-3 型和 Type-4 型克隆的检测上，性能优于 Nicad[20]。该方法的检测性能取决于后缀树的构建和沙砾指令的消除，因此后续工作可以针对后缀树构建的指令类型进行优化，以及提高沙砾指令检测的精度。

表 5 基于文本字符的表征方法的汇总

Table 5 Summary of text character-based representation methods

方法/作者	时间/引用	表征基础	相似性度量方法	主要技术	检测类型
GitZ	2017/[16]	指令序列哈希	基于成对的哈希比较和全局上下文来计算相似性得分	LLVM-IR、VEX-IR、哈希	Type-1 型 Type-2 型 Type-3 型
Zhang 等	2019/[19]	反汇编后的指令序列	利用后缀树进行字符串匹配	后缀树	Type-1 型 Type-2 型 Type-3 型 Type-4 型

4.1.2 基于代码嵌入的表征

基于代码嵌入的表征通常是将二进制代码经过特征提取得到语义、结构信息，然后将这些特征嵌入到一个向量空间，因此这种向量也称为特征向量。对特征向量经过一系列处理或训练后，通过计算两个特征向量之间的相似性来衡量代码片段的相似性。表 6 为基于代码嵌入的表征方法的汇总。

表 6 基于代码嵌入的表征方法的汇总

Table 6 Summary of representation methods based on code embedding

方法/作者	时间/引用	表征基础	相似性度量方法	主要技术	检测类型
BBTree	2023/[21]	基本块树	通过孪生神经网络来计算相似性	Word2Vec，LSTM，Bi-GRU，Siamese Network	Type-1 型 Type-2 型 Type-3 型
INNEREYE	2018/[24]	汇编指令、基本块	计算词嵌入和 LSTM 转换的高维数值向量之间的距离来衡量相似度	Word2Vec，LSTM，Siamese Network	Type-1 型 Type-2 型 Type-3 型
SAFE	2019/[25]	二进制函数中的指令序列	通过孪生神经网络来计算相似性	Word2Vec，RNN，Siamese Network	Type-1 型 Type-2 型 Type-3 型
Asm2Vec	2019/[30]	二进制文件的汇编代码	计算向量的余弦相似度	PV-DM	Type-1 型 Type-2 型 Type-3 型 Type-4 型
αDiff	2018/[35]	二进制函数的原始字节	合并函数内距离 D1、函数间距离 D2 和模块间距离 D3 来衡量函数之间的相似度	DNN，CNN	Type-1 型 Type-2 型 Type-3 型
jTrans	2022/[36]	二进制文件的汇编代码	使用余弦相似度来计算两个函数之间的相似性得分	BERT	Type-1 型 Type-2 型 Type-3 型
Jump-SBERT	2023/[38]	二进制函数	从余弦相似度、曼哈顿距离、欧式距离和点积相似度选出最优的计算策略	BERT，Siamese Network	Type-1 型 Type-2 型 Type-3 型
Wang 等	2023/[40]	汇编指令	通过孪生神经网络进行相似性比较	BERT、下一指令预测(NIP)、数据流向顺序预测(DOP)	Type-1 型 Type-2 型 Type-3 型
Li 等	2023/[41]	架构无关的抽象语法树	通过孪生神经网络和语法树相似度来衡量代码的相似度	Child-Sum Tree-LSTM、注意力机制	Type-1 型 Type-2 型 Type-3 型
FastBCSD	2023/[43]	汇编代码	通过孪生神经网络计算相似性	TextCNN，LSTM，MLP-Mixer	Type-1 型 Type-2 型 Type-3 型
CEBin	2024/[46]	二进制函数	通过可重用嵌入缓存机制缩小候选相似函数的范围，最后将一对函数作为输入并输出函数对之间的相似性	Transformer、WordPiece、可重用嵌入缓存机制	Type-1 型 Type-2 型 Type-3 型

1)使用树型结构的方法

这些方法使用树型数据结构来表示代码片段之间的关系，然后将代码片段映射到向量空间，通过向量之间的相似性来衡量代码片段之间的相似性。

Xiong 等[21]于 2023 年提出了一种基于基本块树(Basic Block Tree，BBT)的二进制代码相似性检测模型 BBTree。

BBTree 使用滑动窗口(Sliding Window)的思想将二进制函数划分为许多特定大小的基本块树,并通过前序遍历(Preorder Traversal,VLR)将每个基本块树处理为指令序列,在 BBTree 处理之前,通过将指令序列转换为操作码和 8 种不同的操作数类型来解决 OOV 问题。BBTree 包含 BBT 解析器(由 Word2Vec 模型和长短期记忆网络 LSTM 组成)和向量生成器(由 Bi-GRU 网络组成)两个结构,目的在于将每个 BBT 所形成的指令序列转换为空间向量,最后通过孪生神经网络来计算相似性。实验表明,在漏洞搜索任务中,其性能优于 SAFE。BBTree 的关键在于,利用滑动窗口获取稳定的局部特征,将 BBT 作为局部特征来解决 CFG 整体拓扑结构变化较大的问题,从而降低了编译环境对检测性能的影响。因此,在设计支持跨编译环境的检测方法时,可以考虑在特征提取阶段获取 CFG 中的稳定特征。

2)使用词嵌入的方法

这些方法通过词嵌入(Word Embedding)模型实现代码表征,例如由 Google 团队提出的无监督学习模型 Word2Vec[22],Doc2Vec[23](也称 Paragraph Vector)等,它们将代码片段视为文档(Document),将汇编指令视为单词(Word),从而对代码片段进行表征。

2018 年,Zuo 等[24]注意到二进制代码分析和自然语言处理存在着许多相似的特点,如语义提取、代码和文本的比较,因此借鉴 NLP 的思想实现了 INNEREYE。由于反汇编后的二进制文件通常以汇编语言表示,因此 INNEREYE 将汇编指令视为单词,将基本块视为句子。Zuo 等认为检测不同指令集架构的两个基本块在语义上是否相似的问题类似于确定不同人类语言的两个句子是否具有相似的含义,因此,他们根据神经机器翻译(Neural Machine Translation,NMT)可以很好地处理跨语言文本的特点,提出了一种新颖的基于神经网络的跨汇编语言基本块嵌入模型。其利用 NMT 中常用的词嵌入和长短期记忆网络(Long Short-Term Memory,LSTM)将基本块转换为高维数值向量,这样不仅能对基本块语义进行编码,还能捕获跨指令集架构的语义关系,最后通过计算两个高维数值向量之间的距离来有效地衡量相似度。INNEREYE 的灵感来自于专注处理各种自然语言文本的自然语言处理,也是早期将 NLP 应用于二进制代码相似性检测的经典方法之一。这也意味着,在不同领域中的方法可能存在着某些共同之处,在以后的研究中可以尝试将二进制代码类比为其他载体,从而使用其他领域的方法来尝试提升检测性能。

Massarelli[25]等于 2019 年提出了一种基于自然语言处理和自注意力神经网络的模型 SAFE(Self-Attentive Function Embeddings),这是一种对二进制可执行文件反汇编后的二进制函数进行代码嵌入的通用架构,不需要手动进行特征提取。它可以快速生成数百个二进制文件的嵌入,并且能够在 AMD64 和 ARM 等多个体系架构上工作。SAFE 将二进制函数中的指令序列视为自然语言,在第一阶段首先进行汇编指令嵌入[26](instructions2vec,i2v),通过 Word2Vec 自然语言处理模型将每个指令序列映射到实数向量,根据当前的指令使用 skip-gram[27]方法来预测周围的指令对 i2v 进行训练;第二阶段则使用 GRU 循环神经网络[28](GRU RNN)来捕获指令的顺序交互信息(Sequential Interaction),最后利用孪生神经网络[29](Siamese Network)输出两个实数向量之间的相似度得分。除了检测相似性,SAFE 还可以通过对相似的代码嵌入进行聚类,从而识别二进制函数的一般语义行为,例如识别二进制函数是否实现了加密算法、排序算法等功能,这也说明了可以将二进制代码相似性检测应用于生活中的其他场景。

Ding 等同样在 2019 年提出了一种汇编代码表示学习模型 Asm2Vec[30],其不需要汇编函数之间的正确映射等先验知识,只需要将汇编代码作为输入,找到并合并出现在汇编代码中的标记之间的丰富的语义关系。Asm2Vec 首先需要给定一个汇编函数存储库,然后使用基于原始 Word2Vec 模型扩展的 PV-DM[23]模型为每个存储库函数生成数值向量。但由于汇编代码结构不同于普通文本,因此其将汇编函数的控制流图建模为多个序列,并通过当前函数的向量和相邻指令提供的上下文来预测当前指令。同时,为了解决控制流图中原来的线性布局覆盖了一些无效的执行路径的问题,Asm2Vec 采用边缘采样(Edging Sampling)和随机游走(Random Walks)的方法生成序列。实验表明,在不同的编译器优化选项和混淆技术下,Asm2Vec 面对严重变化的汇编指令和控制流程图依然具有鲁棒性(Robustness),且可以应用于二进制文件、基本块或函数等不同粒度的汇编序列;但由于 Asm2Vec 是为单一汇编代码语言设计的,并且克隆搜索引擎与体系结构无关,因此不能直接用于跨架构的语义克隆检测。为弥补 Asm2Vec 仅支持单一编程语言的缺陷,后续研究可以考虑引入多种架构下的汇编语言,实现真正的架构无关。

3)使用神经网络的方法

这些方法使用神经网络的模型实现代码表征,例如 BERT[31](Bidirecional Encoder Representations from Transformers)等预训练模型将基本块或代码片段映射到一个向量空间。BERT 是一种基于双向 Transformers 的结构,其使用双向自注意力在大规模文本上进行预训练,可以捕捉到上下文中丰富的语义信息,类似结构还有基于 BERT 改进的 RoBERTa[32]、Sentence-BERT[33](SBERT)、PalmTree[34]等。

Liu 等[35]在 2018 年提出了一种使用深度神经网络(Deep Neural Network,DNN)来检测二进制代码相似性的模型 αDiff,其使用了 3 个语义特征来解决跨版本的二进制文件相似性问题,也适用于跨编译器和跨架构的二进制代码相似性检测。αDiff 直接作用于每个二进制函数的原始字节,并通过 DNN 来提取每个二进制函数的内部特征,将原始字节表示为矩阵,并使用一个修改后的卷积神经网络(Convolutional Neural Network,CNN)将其转换为多维空间向量,同时将 CNN 嵌入到孪生神经网络中使相似函数的向量彼此接近;通过提取二进制程序的函数内、函数间和模块间 3 种特征,分别计算函数内距离 D_1、函数间距离 D_2 和模块间距离 D_3,再将这 3 个距离进行合并来衡量两个二进制函数之间的相似度。αDiff 为 3 种特征分别计算了 3 种相似度,最后进行合并,这也为相似性的计算方式提供了新的思想,即不必局限于目前流行的空间向量的距离度量方法,可以针对性地使用其他相似性度量方式。

Wang 等[36]在 2022 年提出了一种基于 BERT 架构的方法 jTrans,其使用修改后的 Transformer 结构 Transformer-Encoder[37],将捕获指令语义信息的 NLP 模型与捕获控制流信息的 CFG 结合起来,并且是第一个尝试将控制流信息融合

到 Transformer 架构中的方法。jTrans 在对输入的二进制文件的汇编代码进行预处理时使其包含程序的跳转关系（控制流信息），使用 Transformer-Encoder 保证跳转的起点位置和终点位置的语义相似；然后使用掩码语言模型（Masked Language Model，MLM）[31]，根据相邻内容的上下文来预测遮蔽词的内容，使其更好地理解上下文之间的关系；最后根据监督学习任务微调模型，使用余弦相似度来计算两个函数 f_1 和 f_2 之间的相似性得分。jTrans 将控制流信息融合到 Transformer 架构中，大幅度提升了模型的检测性能。后续工作也可以尝试将其他数据特征融合进神经网络模型，以提高检测性能。

Yan 等[38] 在 2023 年提出了一种基于 SBERT 的方法 Jump-SBERT。为了捕获和学习二进制函数之间的跳转关系，其在 SBERT 的基础上引入了跳转识别训练机制，按照 α，β，γ 这 3 种概率且满足 $\alpha+\beta+\gamma=1$ 的条件进行跳转来学习上下文的直接跳转和间接跳转。这种机制使得 Jump-SBERT 可以学习到二进制函数的控制流和数据流等图信息，以此获取更加丰富的语义信息，并且使用标签替代的方法来解决未登录词[39]（Out-of-Vocabulary，OOV）问题。Jump-SBERT 解决了现有模型难以全面、精确地识别二进制函数的语义信息以及数据集单一的问题，其借鉴了 jTrans 中的跳转思想，将跳转作用于汇编指令，能够获取到直接跳转和间接跳转两种信息。但其跳转方式是基于概率的，因此该方法的跳转机制存在改进空间。

Wang 等[40] 于 2023 年提出了一种基于 BERT 和 Palmtree 的对汇编指令进行预训练表征的方法，其在 BERT 的基础上进行了跨架构的设计，同时针对跨架构提出了一种将符号与寄存器、地址结合的新分词方法，并考虑了指令的逻辑与出现的概率，使用下一指令预测（NIP）与数据流向顺序预测（DOP）替换上下文窗口预测（CWP）与 Def-Use 关系预测（DUP）进行表征，最后通过孪生神经网络进行相似性比较。该方法的关键在于分词操作，可见，如何优化分词的过程是提升检测性能的一个可行方法。

Li 等[41] 于 2023 年提出了一种基于 Child-Sum Tree-LSTM[42] 神经网络的方法。其引入了注意力机制且支持跨指令集和跨代码混淆。首先提取二进制代码的抽象语法树；然后利用基于注意力机制的 Child-Sum Tree-LSTM 神经网络对抽象语法树进行节点映射，经过孪生神经网络的训练后得到语义向量；最后通过语法树相似度来衡量代码的相似度。Li 等在 Child-Sum Tree-LSTM 的基础上引入注意力机制用于学习语义向量，最终在 Accuracy 性能上提高了 0.4%。可见，使用注意力机制来改进模型可以提高相似性的检测性能。

Huang 等[43] 于 2023 年提出了一种轻量级神经网络 FastBCSD。其首先对汇编代码进行处理，在预处理时使用分隔符将指令分解并基于频率进行过滤；然后将每条汇编指令视为一个动态向量，利用 TextCNN[44]，LSTM 和 MLP-Mixer[45] 进行训练；最后通过孪生神经网络计算相似性。FastBCSD 使用大量监督数据训练小型模型，可以在减少计算和时间的情况下达到与预训练模型相当的精度，同时可以应用于 x86，ARM 和 MIPS 等多架构环境。FastBCSD 使用动态指令向量编码的方法，解决了模型计算资源太大以及信息丢失的问题，实现了一种轻量级的检测方法，从而大幅减少了模型的参数。因此，提升模型性能可以从减少计算资源方面入手。

Wang 等[46] 于 2024 年提出了一种新的方法 CEBin。其以 Transformer 作为基础架构，首先使用 WordPiece[47] 算法在整个汇编代码数据集上训练分词器，并对汇编代码进行无损编码；然后通过一种可重用嵌入缓存机制（RECM）缩小候选相似函数的范围，从而高效地定位相似函数，比较模型将一对函数作为输入并输出函数对之间的相似性；最后通过分层推理框架提高准确性。CEBin 融合了基于嵌入和基于比较的方法，在最大限度减少开销的基础上，提升了检测性能，这说明多种检测方法之间相互补充有时也能得到不错的效果。

4.1.3 基于图嵌入的表征

基于图嵌入的表征利用代码的图结构等信息，将具有语义信息的代码图结构表征为空间向量，并用向量之间的距离来衡量代码之间的相似性。由于二进制程序的 CFG，FCG，PDG 等图结构通常具有丰富的语义信息，可以体现程序的执行流，因此可以用于检测 Type-4 型代码克隆。同时，为了更好地对二进制程序逆向进行分析，研究者开发了许多基于图结构的 IDA Pro 插件用于相似性分析，如 BinDiff[48]，Diaphora[49]，Kam1n0[50] 等。表 7 为基于图嵌入的表征方法的汇总。

表 7 基于图嵌入的表征方法的汇总

Table 7 Summary of representation methods based on graph embedding

方法/作者	时间/引用	表征基础	相似性度量方法	主要技术	检测类型
Genius	2016/[51]	CFG，ACFG	数值向量的每个维度表示原始特征与码本中某个类别的相似性距离	无监督学习、谱聚类、码本、二分图匹配、属性控制流图 ACFG	Type-1 型 Type-2 型 Type-3 型
Gemini	2017/[54]	CFG，ACFG	计算余弦相似度输出向量的相似度得分	Structure2vec，Siamese Network，属性控制流图 ACFG	Type-1 型 Type-2 型 Type-3 型
VulSeeker	2018/[56]	CFG，DFG，LSFG	计算余弦相似度与相似性阈值的关系	DNN，标记语义流图 LSFG	Type-1 型 Type-2 型 Type-3 型
IFAttn	2022/[57]	CFG，FCG	使用两个子网络共享权重和参数的孪生神经网络计算余弦相似度来识别克隆	Transformer，Siamese Network	Type-1 型 Type-2 型 Type-3 型
CI-Detector	2024/[59]	CFG，ACFG	使用 3 种内联模型分别处理 3 类不同的内联方式，最后聚合模型的结果得到最终的相似度	属性控制流图 ACFG，CNN	Type-1 型 Type-2 型 Type-3 型
Kinable 等	2010/[60]	FCG	通过最小图编辑距离来计算函数调用图之间的相似性	k-medoids 聚类、DBSCAN 聚类、图编辑距离、图同构	Type-1 型 Type-2 型 Type-3 型

1)使用控制流图的方法

Qian 等[51]受计算机视觉领域成果的启发,于 2016 年提出了一种将二进制代码控制流图生成特征向量来进行相似性检测和漏洞搜索的模型 Genius,其也是首次在二进制代码相似性分析领域引入机器学习中的图嵌入概念的方法,因为相比于控制流图,特征向量通常在不同架构的代码之间具有更好的鲁棒性。Genius 首先从二进制函数中提取属性控制流图(Attributed Control Flow Graph,ACFG)作为原始特征(Raw Features),ACFG 本质上是一种具有不同基本块级属性的 CFG。Genius 主要使用了统计特征和结构特征这两种基本块级属性,其中统计特征用于描述基本块内的局部统计,而结构特征用于捕获 CFG 内基本块的位置特征。然后使用无监督学习算法,通过相似性度量计算(Similarity Metric Computation)和聚类(Clustering)两个阶段来根据原始特征的训练集生成码本(Codebook)。相似性度量计算阶段通过二分图匹配(Bipartite Graph Matching)来实现,聚类阶段通过谱聚类[52](Spectral Clustering)来实现。最后通过特征编码[53](Feature Encoding)将函数的原始特征映射到数值向量,且数值向量的每个维度表示原始特征与码本中某个类别的相似性距离。Genius 首次引入机器学习中的图嵌入,并提出了 ACFG 的概念,为二进制代码相似性检测开辟了新的方向。可见,研究新的嵌入方法是提升检测性能的一种手段。

为了提高 Genius 的准确率和检测效率,Xu 等[54]在 2017 年提出了一种基于每个二进制函数的控制流图生成数值向量的新方法 Gemini。基于图匹配的相似性检测方法的效率都会受到图匹配算法(如二分图匹配)的效率的限制,因此Gemini 提出了一种基于深度神经网络的方法来生成二进制函数 ACFG 的嵌入。其设计了一个孪生神经网络结构,并与修改后的图嵌入神经网络 Structure2vec[55]结合,这种架构使得模型能够进行端到端训练。将两个二进制函数的 ACFG 分别作为模型的输入,使用聚合函数将所有特征向量表示为一个向量,再计算余弦相似度,输出它们的相似度得分。相比于 Genius,Gemini 在相似性检测的精度、生成特征向量所需的时间和总体的训练时间方面都有大幅度的提升。Gemini 使用了变种的 Structure2vec 来计算图嵌入,其效率比 Genius 更高。因此,改进嵌入的生成方式可以提升模型的检测性能。

Gao 等[56]于 2018 年提出了一种基于语义学习的跨平台检测模型 VulSeeker,其使用包含控制流图和数据流图(Data Flow Graph,DFG)的标记语义流图(Labeled Semantic Flow Graph,LSFG)。相比之前单独从 CFG 中提取特征的方法,使用 LSFG 可以捕获到更多的二进制函数的语义信息,同时有效地减少在不同平台下 CFG 不同结构的干扰。为了精确捕获函数语义信息,VulSeeker 参考 Structure2vec 神经网络,使用了一种包含 T 层迭代的语义感知 DNN 模型来处理结构化的 LSFG 数据,其将二进制函数内所有基本块顶点的嵌入向量聚合,通过 T 层迭代后,每个顶点的特征随着 LSFG 拓扑的迭代传播到其他顶点,使得二进制函数的每个基本块都具有相应的上下文语义信息。最后预先定义一个相似性阈值,通过计算余弦相似度与相似性阈值的关系来判断两个代码片段是否相似。实验表明,VulSeeker 在代码克隆检测的准确率和检测性能上高于 Gemini。VulSeeker 通过整合二进制函数的 CFG 和 DFG 来丰富函数的语义。其 Accuracy 性能相比 Gemini 提高了 12.14%,说明提升检测性能可以从丰富语义特征着手。

Jiang 等[57]于 2022 年提出了一种基于可解释特征和注意机制的二进制代码相似性模型 IFAttn,其利用注意力机制将可解释的基本特征转换为自适应语义特征。同时,该方法也是第一次使用注意力机制在功能层面上合并基本特征的方法。IFAttn 使用 TIKNIB[58]中定义的前语义特征(Presemantic Features),从 CFG 以及 FCG 等结构中提取随代码结构变化而变化的数值特征;然后通过 KFM 缩放点积注意力和多头注意力(Multi-Head Attention)机制的 Transformer 结构,将数值特征转换为维度为 50 的特征向量;最后使用两个子网络共享权重和参数的孪生神经网络计算余弦相似度来识别克隆。IFAttn 利用注意力机制来合并基本特征,本质上也是丰富了二进制函数的语义特征,再一次说明了丰富的语义特征可以提升模型的性能。

为了解决函数内联给检测精确度带来的影响,Jia 等[59]在 2024 年提出了一种针对交叉内联二进制函数(Cross-Inlining Binary Function)的相似性检测方法 CI-Detector。其使用 binary2source 函数来识别包含内联函数的二进制函数,然后将源函数视为桥接函数来构造交叉内联函数的映射,结合操作码和 CFG 得到 ACFG 后使用 GNN 将其表征为向量,并将交叉内联分为叶内联(Leaf-Inlining)、根内联(Root-Inlining)和内部内联(Internal-Inlining)3 类。针对不同的内联方式,使用不同的模型来计算相似度。最后将 3 种模型的结果聚合得到最终的相似度。不同于交叉优化、交叉编译和跨架构的场景,CI-Detector 是一种针对交叉内联函数的检测模型,这也为以后的研究方向提供了一种新的选择。

2)使用函数调用图的方法

Kinable 等[60]在 2010 年提出了一种基于函数调用图聚类的方法,其通过将代码样本表示为函数调用图,抽象出代码片段中的某些变化;然后利用图匹配(Graph Matchings)技术,通过图同构(Graph Isomorphisms)、检测最大公共子图(Maximum Common Subgraphs,MCS)以及查找最小图编辑距离(Graph Edit Distances,GED)来计算函数调用图之间的相似性(Graph Similarity);最后使用 k-medoids 和 DBSCAN 等多种聚类算法(Clustering Algorithms)将与恶意代码样本的函数调用图结构相似的代码视为恶意代码。该方法使用函数调用图来提取特征,表明在特征提取阶段除了提取 CFG 特征,还可以利用其他的二进制特征表现形式来进行提取。

4.2 多架构相似性检测

在数字化的时代背景下,互联网开发商陆续推出了各种不同平台架构的终端设备,如个人电脑、手机、路由器和其他嵌入式设备。为了更好地满足程序在不同终端下的使用需求,针对不同系统平台开发的程序通常会使用不同的指令集架构。根据可检测的架构种类可以将二进制代码相似性检测方法分为单一架构(Single Architecture)和多架构(Multiple Architectures)。考虑到不同架构之间的指令集差异和不同

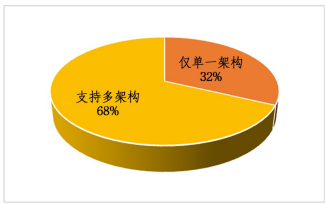
架构下的编译器优化等因素对相似性检测的影响，多架构的相似性检测的难度通常高于单一架构的相似性检测。多架构相似性检测和开源情况的汇总结果如表 8 所列，图 6 为多架构二进制代码相似性分析结果。

表 8 多架构、开源情况、Baseline、基准数据集的汇总

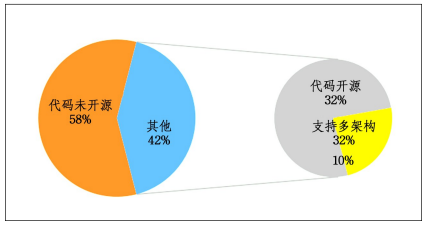
Table 8 Summary of multi-architecture, open source situation, Baseline and benchmark datasets

方法/作者	多架构	开源	Baseline	数据集
GitZ	是	否	ESH	来自开源项目的 9 个真实存在的易受攻击的程序、Esh 中公开可用的数据集、包含每个查询的 44 个真例的 500 k 程序语料库
Zhang 等	否	否	Nicad, SimCad	kernelv4.0.1, kernel v4.9.10
BBTree	否	否	Gemini, SAFE, Asm2Vec	BinKit, OpenSSL v1.0.1u
INNEREYE	是	否	—	thttpd v2.25b, atphttpd v0.4b, boa v0.94.13, lighttpd v1.4.30
SAFE	否	是	Gemini	gcc-5 编译的 openssl v1.1.1, gcc-6 编译的 postegres v9.6
Asm2Vec	否	否	Composite, Constant, Graphlet, Graphlet-C, Graphlet-E, MixedGram, MixedGraph, n-gram, n-perm, FuncSim-Search, PV(DM/DBOW)	公开可用的漏洞搜索数据集
αDiff	是	是	BinDiff, BinGo	GitHub 存储库、Debian 软件包存储库
jTrans	否	否	Gemini, SAFE, Asm2Vec, GraphEmb, OrderMatters, Genius	BinaryCorp
Jump-SBERT	是	否	jTrans	BinaryCorp
Wang 等	是	否	PalmTree	x86, arm 与 mips 架构下使用 9.4.0, 8.2.0, 7.3.0, 6.4.0, 5.5.0 版本 gcc 和 10.0, 7.0, 6.0, 5.0, 4.0 版本 clang 编译的 2.32, 2.35, 2.36 版本 binutils; 6.5, 6.7 版本 Coreutils; 1.33.1, 1.34.1 版本 busybox
Li 等	是	否	Child-Sum Tree-LSTM	BinKit
FastBCSD	是	是	Gemini, SAFE, Asm2Vec, GraphEmb, Order-Matters, jTrans	BinaryCorp
CEBin	是	否	Genius, Gemini, SAFE, Asm2Vec, GraphEmb, OrderMatters, Trex, GNN and GMN, jTrans	BinaryCorp, Cisco Dataset, Trex Dataset
Genius	是	是	—	32 位 x86, ARM, MIPS 下 BusyBox v1.21 和 v1.20, OpenSSL v1.0.1f 和 v1.0.1a, coreutils v6.5 和 v6.7, 公开可用的固件映像, CVE 漏洞数据集
Gemini	是	是	Genius	GCC v5.4 编译 OpenSSL v1.0.1f 和 v1.0.1u, Genius 使用的各种架构的 固件映像, 具有各种顶点数量的 ACFG 数据集, Genius 使用的漏洞数据集
VulSeeker	是	是	Gemini	GCC v4.9 和 v5.5 优化级别为 O0～3 的函数和基本块, Genius 使用的各种架构的固件映像
IFAttn	是	是	TIKNIB, Gemini, Asteria, Codee	BinKit
CI-Detector	是	是	Gemini, SAFE, GMN, Bingo, Asm2vec	BinKit
Kinable 等	否	否	—	F-Secure 公司提供的 194 个恶意样本

在筛选的 19 篇相关文献中，支持多架构检测的方法约占总体的 68%，其中有 7 种方法出自近 5 年，有 6 种方法已开源。例如 Genius, Gemini 等经典方法和 CEBin, FastBCSD 等近两年的新方法都适用于多架构的相似性检测，并且源代码已经公开。而 SAFE, Asm2Vec 等 6 种方法仅适用于单一架构的相似性检测，约占总体的 32%，其中开源的方法有 2 种。



(a)多架构模型百分比



(b)模型开源百分比

图 6 多架构二进制代码相似性分析

Fig. 6 Analysis of multi-architecture binary code similarity

根据统计结果可以发现，目前的二进制代码相似性检测方法基本由原来的仅支持单一架构转变为支持多架构。因此在当今的时代背景下，支持多架构的相似性检测也是未来研究发展中的一个必然方向。

4.3 常用 Baseline 方法

Baseline 方法指在模型实验中用来作为比较基准或起点的基本方法，其提供了一个基本的性能水平或参考，可以用来作为其他更复杂方法的对照，以此来评估这些方法的性能优劣或改进的效果。通过比较各种方法常使用的 Baseline 情况可以看出哪些方法更为经典、更加适合用于对比实验，可以为未来的研究提供基础和参考方向。本文筛选的 19 篇相关文献中 Baseline 方法的汇总结果如表 8 所列，图 7 为二进制代码相似性检测常用 Baseline 的分析结果。

在筛选的 19 篇相关文献中，主要统计了其中 2 篇文献以上被选作 Baseline 的方法。其中，Gemini[54] 被 8 篇文献选作 Baseline，占比最大；其次是 SAFE[25] 和 Asm2Vec[30]，各被 5 篇文献选为 Baseline；jTrans[36] 是 2022 年提出的基于 BERT 架构的新方法，被 3 篇文献作为 Baseline；而 Genius[51] 作为首次将机器学习中的图嵌入引入二进制代码相似性检测领域的经典方法，由于发布于 2016 年，且被 2017 年提出的 Gemini 进行了改进，因此仅被 3 篇文献选作 Baseline，但 Genius 依旧

是将图嵌入作为代码表征的经典方法之一。图嵌入模型GraphEmb[61]、OrderMatters[62]同样被3篇文献选为Baseline，图嵌入模型Bingo[63]和代码嵌入模型GMN[64]分别被两篇文献选为Baseline。

由统计结果可以看出，这些方法在二进制代码相似性检测领域中相对经典，得到了大多数研究者的认可并被选为实验的比较对象，可以作为未来研究的参考标准。

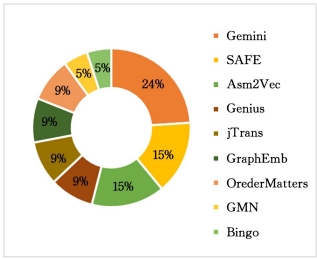


图7 二进制代码相似性常用Baseline分析

Fig. 7 Commonly used Baseline analysis for binary code similarity

4.4 基准数据集

数据集也是检验实验效果的一部分，因此数据集的选取在实验中同样重要。数据集通常可以分为开源数据集和闭源数据集两种。开源数据集指公开发布的、所有人都可以自行下载的数据集；闭源数据集则相反，这类数据集并没有公开发表或难以获取，因此无法进行实验验证，大多数可能并不具有普适性。筛选的相关文献中基准数据集的使用情况如表8所列，图8为二进制代码相似性常用基准数据集分析结果。

根据基准数据集的统计情况，大部分方法使用的是自编译的二进制数据集或将Linux等系统内核文件作为基准数据集。为了研究多架构下的相似性，这些二进制文件通常由不同指令集架构下不同版本的GCC采用不同的编译优化选项编译而来。例如，Genius通过GCC和Clang分别在x86，ARM，MIPS这3种架构下进行编译得到不同版本的BusyBox(v1.21和v1.20)，OpenSSL(v1.0.1f和v1.0.1a)和coreutils(v6.5和v6.7)二进制文件作为数据集，同时还使用了O0－O3这4种不同的优化选项。类似的二进制文件还有固件映像、postegres、thttpd和kernel等。

在公共的开源数据集中使用较多的是BinaryCorp[36]和BinKit[57]，它们各被4篇文献选为基准数据集。BinaryCorp在jTrans中被首次提出并作为基准数据集使用，是目前已知最多样化的数据集，其基于ArchLinux官方存储库和Arch用户存储库构建，包含编辑器、即时通信、服务器、浏览器、编译器等，其软件种类数以万计，共有48130个使用GCC和G++在不同优化级别编译的二进制程序。BinKit是一个综合数据集，在TIKNIB[58]中首次被提出，其提供了构建交叉编译环境的脚本，由243128个二进制文件组成，包含8个体系结构、5个优化级别(O0－O3和Os)以及13个编译器的1352个不同的编译器选项组合；2023年BinKit发布了2.0版本，将二进制文件扩充到371928个，并增加了1个优化级别Ofast和10个编译器的552个不同的编译器选项组合。BinKit相比于BinaryCorp，具备更多的跨架构、跨编译器和跨编译优化选项。另外，Kinable等[60]提出的方法使用了F-Secure公司提供的194个恶意样本，但他们没有公开数据集。

虽然这些数据集都是以二进制文件作为相似性检测的对象，但BinaryCorp和BinKit这样的公共开源数据集可以方便对实验数据集进行统一。自编译数据集则不同，不同版本下的GCC编译的二进制文件可能存在差异，使得实验数据集难以真正统一。另外，目前针对二进制代码相似性检测的开源数据集并不多，使用较多的是BinaryCorp和BinKit。由此可见，为了能够更好地对比模型的实际效果，二进制代码相似性检测的相关研究需要一个标准且完善的公共数据集，并且应当更多地选择统一的开源数据集进行实验，以更好地推进该领域在未来的研究。

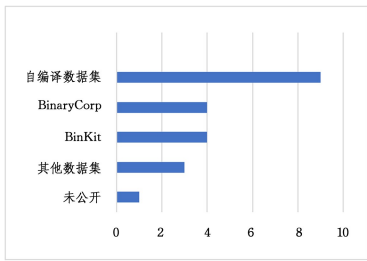


图8 二进制代码相似性常用基准数据集分析

Fig. 8 Analysis of common benchmark datasets for binary code similarity

4.5 模型的检测性能

由于模型实际性能受到数据集的影响，许多方法并未使用统一的数据集进行实验，且通过GCC编译的自编译数据集难以保证数据集的一致性，这些因素都会使模型的检测性能难以比较。因此，本节针对在同一数据集下的实验，将公共的开源数据集BinaryCorp作为基准数据集。

为了保证数据集的一致性和检测性能的准确性，本节在BinaryCorp数据集上针对不同的编译选项((O0，O3)，(O1，O3)，(O2，O3)，(O0，Os)，(O1，Os)和(O2，Os))，将二进制相似性检测领域近两年的新方法和经典方法进行对比，旨在突出近两年的新方法与传统方法在各项检测性能上的变化趋势，为今后的研究提供参考。本节选取8种检测模型在不同的编译优化选项下进行了性能分析，其中Gemini，SAFE和Asm2Vec等是经典的二进制相似性检测方法的代表，jTrans，Jump-SBERT和FastBCSD等是近两年的新方法的代表。

表9列出了BinaryCorp数据集下的二进制代码相似性检测性能分析结果。由于BinaryCorp数据集较大，因此实验均使用BinaryCorp数据集的一个子集BinaryCorp-26M，共包含约2600万个函数，函数池大小参数为$Poolsize=32$。实验环境为配备Intel Xeon 96核3.0GHz CPU，768GB RAM和8个Nvidia A100 GPU，系统为Ubuntu 18.04的Linux服务器[36]，以及配备Intel Xeon 10核2.20 GHz CPU，256 GB RAM和1个Nvidia 1080Ti GPU的硬件环境[43]。图9展示了BinaryCorp数据集下的二进制代码相似性模型检测性能分析结果。

表 9 BinaryCorp 数据集下的二进制代码相似性检测性能分析

Table 9 Performance analysis of binary code similarity detection under BinaryCorp dataset

方法	Recall						
	O0,O3	O1,O3	O2,O3	O0,Os	O1,Os	O2,Os	Average
Gemini	0.263	0.528	0.768	0.322	0.441	0.518	0.473
SAFE	0.770	0.902	0.951	0.795	0.891	0.891	0.867
Asm2Vec	0.314	0.789	0.940	0.362	0.716	0.800	0.654
GraphEmb	0.465	0.586	0.667	0.499	0.541	0.543	0.550
OrderMatters	0.417	0.740	0.903	0.481	0.692	0.677	0.652
jTrans	0.941	0.970	0.981	0.949	0.964	0.964	0.962
Jump-SBERT	0.960	0.974	0.958	0.959	0.972	0.966	0.963
FastBCSD	0.888	0.949	0.963	0.916	0.955	0.948	0.937

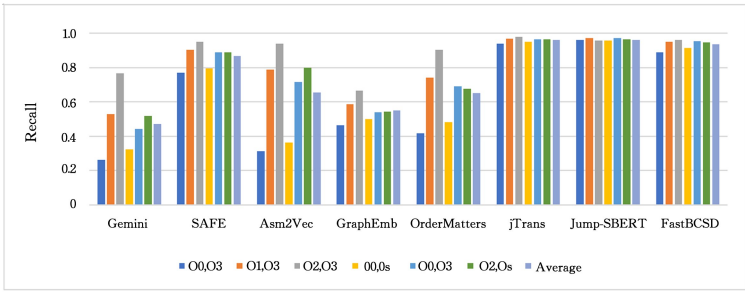


图 9 BinaryCorp 数据集下的二进制代码相似性检测性能分析

Fig. 9 Performance analysis of binary code similarity detection under BinaryCorp dataset

根据统计结果可知，在经典方法中，SAFE 对于不同编译选项下的检测性能明显优于其他经典方法，在 O1，O3 和 O2，O3 上的检测均高于 0.9，且平均召回率达到 0.867。而在近两年的新方法中，jTrans 在 Recall 指标上比经典方法中最接近的 SAFE 还高出 10%以上。jTrans 是第一个尝试将控制流信息融合到 Transformer 架构中的方法，其利用跳转关系来预测遮蔽词的内容，能够更好地理解上下文之间的关系，这也是 jTrans 优于经典方法的主要原因之一。FastBCSD 同样实现了与 jTrans 相当的性能，在 Recall 指标上都显著优于经典的检测模型，仅在 O0，O3 上的 Recall 低于 0.9，原因在于 FastBCSD 能够为二进制代码相似性检测任务构建大量的监督数据以及相对均匀的数据分布，通过使用大量监督数据训练小型模型可以实现与使用大型模型的预训练和微调方法相当的性能。而 Jump-SBERT 针对 jTrans 的跳转机制进行改进，在数据层面对汇编代码进行处理，使其可以识别直接跳转和间接跳转两种信息，因此在检测性能上与 jTrans 相似，同时采用了 4 种相似度计算方法，使 Recall 最高可达 0.963。

几乎所有的经典方法在 O0，O3 和 O0，Os 这类编译选项差异较大的检测上，其性能都明显低于平均值，而 jTrans，Jump-SBERT 和 FastBCSD 这些新方法由于在表征方式上具有更好的跨编译优化能力，因此在 O0，O3 和 O0，Os 这类编译选项差异较大的检测上较为稳定。但是，jTrans 等大多数检测方法的检测效果会随着函数池大小参数 Poolsize 的增大而下降，大多数现有解决方案的准确性会下降 20%左右[36]，因此未来的研究可以考虑提升 O0，O3 和 O0，Os 这类差异较大的编译选项上的检测性能；同时随着函数池的增大，需保证模型依然具有较为稳定的检测性能。

5 未来的挑战和可能的研究方向

目前在软件安全领域，二进制代码相似性检测依然是一个重要的理论基础，虽然在该领域已经有大量学者进行了长达几十年的研究，但依然有许多问题没有得到很好的解决。根据本文筛选的文献，将未来的挑战和可能的研究方向总结如下：

1）开发支持跨指令架构、跨平台的检测模型

不同的操作系统平台通常对应着不同的指令架构，如移动端的 ARM、桌面端的 INTEL 以及一些嵌入式设备的 MIPS。虽然同一种程序可能有着多个平台的版本，但目前能够支持跨指令架构、跨平台的二进制代码相似性检测方法尚不完善。因此，设计一种指令架构和平台无关的二进制代码相似性检测模型将是一个重要的研究方向。

2）考虑提高不同编译器优化下的检测效果

二进制代码经过源代码的编译、链接等操作，而不同编译器（如 GCC，Clang）甚至同一编译器的不同版本对源代码的优化配置都会有所差别，导致同一源代码得到的二进制代码也可能天差地别，尤其是在 O0，O3 和 O0，Os 这类编译优化选项差异较大的情景下，大部分方法的检测性能低于平均值，具体数值如表 9 所列。因此，实际中必须考虑到不同编译器优化带来的影响。虽然目前有许多方法具备跨编译器的检测能力，但实际效果还存在改进的空间。

3）设计并选择统一的公共数据集进行研究

二进制代码相似性检测已经经历了几十年的研究与发展，但并没有统一标准的、具有公信力的、完善且公开的数据集，甚至一些方法并未公开实验的数据集。虽然目前已经开源了统一标准的 BinaryCorp 和 BinKit 数据集，近些年也被 jTrans 和 FastBCSD 等较多方法使用，但这些数据集都是近五年才出现的，发布时间较晚，且这类开源、统一标准的数据集的数量较少。对于二进制代码相似性检测的不同实现方法，如果无法统一数据集，就难以比较不同方法之间的性能。因此，设计一个完善且有公信力的开源数据集并用于该领域

的研究也是很有必要的。另外，研究者也应当更多地选择统一的开源数据集进行实验。

4)针对 Type-3 和 Type-4 型克隆，寻找更精准提取语义信息的表征方法

在相似性计算阶段，基于文本字符、基于代码嵌入、基于图嵌入和基于符号序列的表征虽然各有与之相关的计算方法，但每种表征形式下的相似性计算方法都基本类似。相比之下，代码表征方法对于二进制代码相似性检测效果具有更大的影响。寻找能够更精准地提取到更多代码语义特征，能够对语义特征更好地进行代码表征的方法，也是提高模型检测效果的关键。

5)提升模型在面对代码混淆等保护措施时的检测性能

在软件安全领域，为了对抗二进制逆向分析，通常会对二进制可执行文件进行代码混淆、反调试、加密等处理，以此来增加反汇编的难度和成本。但目前的相似性检测研究大多只针对 Type-1 到 Type-4 型的克隆检测，包括恶意样本识别和漏洞搜索，通常只针对未经保护的源代码或二进制代码，因此缺乏对代码混淆等保护措施的考虑。针对这种场景，设计一种能识别代码保护措施并能对经过保护的二进制代码进行相似性分析的模型也是很有实际意义的。

结束语 本文以二进制代码相似性检测为主，筛选出了 19 篇相关实现方法，包括一些经典方法和近五年的新方法，并按照代码表征形式的不同，将二进制代码相似性检测的方法分为基于文本字符、基于代码嵌入、基于图嵌入 3 大类，同时对每类方法都进行了分析说明和汇总。最后按照多架构、Baseline 方法、基准数据集和检测性能进行分类统计，并针对当前二进制代码相似性检测领域的问题以及未来可能的研究方向进行了分析和讨论，为对本领域感兴趣的爱好者和研究者提供了一定的参考价值。

虽然本文分析了二进制代码相似性检测领域的经典方法和近五年的新方法，并得出了一些启发性的结论，但仍存在一些局限性及不足。为了使用开源且统一的数据集来保证模型性能的参考性，仅在 BinaryCorp 数据集下进行了性能分析，在其他数据集下的模型检测性能尚未得出结论。另外，为了更详细地分析二进制代码相似性检测领域中各种方法的实现原理和使用的 Baseline、基准数据集等，保证所筛选的方法具有一定代表性，仅从经典方法和近五年的新方法中选取了 19 篇相关文献，因此统计结果可能并不完善。希望本文可以为研究者指明哪些方法具有继续深入研究的价值和潜力，推进二进制代码相似性检测领域的深入发展。

参 考 文 献

[1] SUN X J, WEI Q, WANG Y S, et al. Survey of code similarity detection technology[J]. Journal of Computer Applications, 2024, 44(4): 1248-1258.

[2] NVD. CVE-2023-20892[EB/OL]. (2023-06-22) [2024-01-20]. https://nvd.nist.gov/vuln/detail/CVE-2023-20892/.

[3] ROY C K, CORDY J R, KOSCHKE R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach[J]. Science of Computer Programming, 2009, 74(7): 470-495.

[4] UL HAQ I, CABALLERO J. A Survey of Binary Code Similarity[J]. ACM Computing Surveys, 2021, 54(3): 1-38.

[5] XIA B, PANG J M, ZHOU X, et al. Research progress on binary code similarity search[J]. Journal of Computer Applications, 2022, 42(4): 985-998.

[6] ZHOU Z J, DONG R C, JIANG J H, et al. Survey on Binary Code Security Techniques[J]. Computer Systems and Applications, 2023, 32(1): 1-11.

[7] FANG L, WU Z H, WEI Q. Summary of Binary Code Similarity Detection Techniques[J]. Computer Science, 2021, 48(5): 1-8.

[8] LI Z, ZOU D Q, XU S H, et al. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2022, 19(4): 2244-2258.

[9] XIE C L, LIANG Y, WANG X. Survey of Deep Learning Applied in Code Representation[J]. Computer Engineering and Applications, 2021, 57(20): 53-63.

[10] 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.

[11] CHEN Q Y, LI S P, YAN M, et al. Code Clone Detection: A Literature Review[J]. Journal of Software, 2019, 30(4): 962-980.

[12] LE Q Y, LIU J X, SUN X P, et al. Survey of Research Progress of Code Clone Detection[J]. Computer Science, 2021, 48(S2): 509-522.

[13] WHALE G. Plague: Plagiarism Detection Using Program Structure[D]. Sydeny: University of New South Wales, 1988.

[14] MCCREIGHT E M. A Space-Economical Suffix Tree Construction Algorithm[J]. Journal of the ACM, 1976, 23(2): 262-272.

[15] UKKONEN E. On-line construction of suffix trees[J]. Algorithmica, 1995, 14(3): 249-260.

[16] DAVID Y, PARTUSH N, YAHAV E. Similarity of binaries through re-optimization[C]// The 38th ACMSIGPLAN Conference on Programming Language Design and Implementation. ACM, 2017: 79-94.

[17] DAVID Y, PARTUSH N, YAHAV E. Statistical similarity of binaries[C]// The 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI, 2016: 266-280.

[18] NETHERCOTE N, SEWARD J. Valgrind: a framework for heavyweight dynamic binary instrumentation[C]// The 28th ACM SIGPLAN Conference on Programming Language Design and Implementation. PLDI, 2007: 89-100.

[19] ZHANG L H, GUI S L, MU F J, et al. Clone Detection Algorithm for Binary Executable Code with Suffix Tree[J]. Computer Science, 2019, 46(10): 141-147.

[20] ROY C K, CORDY J R. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C]// 2008 16th IEEE International Conference on Program Comprehension. IEEE, 2008: 172-181.

[21] XIONG M, XUE Y X, XU Y. A binary code similarity analysis method based on code embedding[J]. Cyber Security And Data Governance, 2023, 42(3): 58-67.

[22] MIKOLOV T,CHEN K,CORRADO G,et al. Efficient estimation of word representations in vector space[C]// International Conference on Learning Representations. ICLR,2013.

[23] LE Q,MIKOLOV T. Distributed representations of sentences and documents[C]// The 31st International Conference on Machine Learning. PMLR,2014:1188-1196.

[24] ZUO F,LI X,YOUNG P,et al. Neural machine translation inspired binary code similarity comparison beyond function pairs [C]// Network and Distributed Systems Security Symposium. NDSS,2019.

[25] MASSARELLI L,GIUSEPPE A D L,PETRONI F,et al. Safe: Self-attentive function embeddings for binary similarity[C]// International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Cham:Springer,2019:309-329.

[26] MASSARELLI L,GIUSEPPE A D L,PETRONI F,et al. Investigating graph embedding neural networks with unsupervised features extraction for binary analysis[C]// The 2nd Workshop on Binary Analysis Research. BAR,2019.

[27] MIKOLOV T,SUTSKEVER I,CHEN K,et al. Distributed representations of words and phrases and their compositionality [C]// The 26th International Conference on Neural Information Processing Systems. NIPS,2013:3111-3119.

[28] LIN Z,FENG M,NOGUEIRA DOS SANTOS C,et al. A structured self-attentive sentence embedding[C]// International Conference on Learning Representations. ICLR,2017.

[29] BROMLEY J,GUYON I,LECUN Y,et al. Signature verification using a "Siamese" time delay neural network[C]// The 6th International Conference on Neural Information Processing Systems. NIPS,1994:737-744.

[30] 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]// The 2019 IEEE Symposium on Security and Privacy. IEEE,2019: 472-489.

[31] DEVLIN J,CHANG M W,LEE K,et al. Bert:Pre-training of deep bidirectional transformers for language understanding [C]// North American Chapter of the Association for Computational Linguistics. NAACL-HLT,2019:4171-4186.

[32] LIU Y H,OTT M,GOYAL N,et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach[C]// International Conference on Learning Representations. ICLR,2020.

[33] REIMERS N,GUREVYCH I. Sentence-BERT:Sentence Embeddings using Siamese BERT-Networks[C]// The 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing. EMNLP,2019:3982-3992.

[34] LI X,QU Y,YIN H. Palmtree:learning an assembly language model for instruction embedding[C]// The 2021 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2021:3236-3251.

[35] LIU B,HUO W,ZHANG C,et al. αdiff:cross-version binary code similarity detection with dnn[C]// The 33rd ACM/IEEE International Conference on Automated Software Engineering. IEEE,2018:667-678.

[36] WANG H,QU W,KATZ G,et al. jTrans:jump-aware transformer for binary code similarity detection[C]// The 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. ACM,2022:1-13.

[37] VASWANI A,SHAZEER N,PARMAR N,et al. Attention is all you need[C]// The 31st International Conference on Neural Information Processing Systems. NIPS,2017:5998-6008.

[38] YAN Y T,YU L,WANG T Y,et al. Research on Binary Code Similarity Detection Based on Jump-SBERT[J]. Computer Science,2024,51(5):355-362.

[39] PALMER D D,OSTENDORF M. Improving out-of-vocabulary name resolution[J]. Computer Speech & Language, 2005, 19(1):107-128.

[40] WANG T Y,PAN Z L,YU L,et al. Binary Code Similarity Detection Method Based on Pre-training Assembly Instruction Representation[J]. Computer Science,2023,50(4):288-297.

[41] LI T,WANG J S. Binary code similarity detection via attention mechanism and Child-Sum Tree-LSTM[J]. Cyber Security and Data Governance,2023,42(11):8-14,34.

[42] AHMED M,SAMEE M,MERCER R. Improving Tree-LSTM with Tree Attention[C]// 2019 IEEE 13th International Conference on Semantic Computing. ICSC,2019:247-254.

[43] HUANG C S,ZHU G B,GE G J,et al. FastBCSD:Fast and Efficient Neural Network for Binary Code Similarity Detection[J]. arXiv:2306.14168,2023.

[44] KIM Y. Convolutional Neural Networks for Sentence Classification[C]// the 2014 Conference on Empirical Methods in Natural Language Processing. EMNLP,2014:1746-1751.

[45] TOLSTIKHIN I,HOULSBY N,KOLESNIKOV A,et al. MLP-mixer:an all-MLP architecture for vision[C]// Proceedings of the 35th International Conference on Neural Information Processing Systems. Red Hook,NY:Curran Associates Inc.,2021: 24261-24272.

[46] WANG H,GAO Z Y,ZHANG C,et al. CEBin:A Cost-Effective Framework for Large-Scale Binary Code Similarity Detection [C]// The ACM SIGSOFT International Symposium on Software Testing and Analysis. ISSTA,2024.

[47] TAKU K. Subword regularization:Improving neural network translation models with multiple subword candidates[C]// The 56th Annual Meeting of the Association for Computational Linguistics(Long Papers). Association for Computational Linguistics,2018:66-75.

[48] Zynamics. com. BinDiff[EB/OL]. (2024-03-09) [2024-03-09]. https://www.zynamics.com/bindiff.html.

[49] Joxeankoret. Diaphora:A Free and Open Source Program Diffing Tool[EB/OL]. (2024-03-12) [2024-03-12]. http://diaphora.re/.

[50] DING S H H,FUNG B C M,CHARLAND P. Kam1n0:Mapreduce-based assembly clone search for reverse engineering[C]// The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD,2016:461-470.

[51] QIAN F,ZHOU R,XU C,et al. Scalable Graph-based Bug Search for Firmware Images[C]// ACM Sigsac Conference on Computer & Communications Security. CCS,2016:480-491.

[52] NG A Y, JORDAN M I, WEISS Y. On spectral clustering: analysis and an algorithm[C]// Proceedings of the 15th International Conference on Neural Information Processing Systems: Natural and Synthetic. Cambridge, MA: MIT, 2001: 849-856.

[53] CHATFIELD K, LEMPITSKY V S, VEDALDI A, et al. The devil is in the details: an evaluation of recent feature encoding methods[C]// British Machine Vision Conference 2011. NIPS, 2011.

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

[55] DAI H J, DAI B, SONG L. Discriminative Embeddings of Latent Variable Models for Structured Data[C]// The 33rd International Conference on International Conference on Machine Learning. ICML, 2016: 2702-2711.

[56] GAO J, YANG X, FU Y, et al. VulSeeker: a semantic learning based vulnerability seeker for cross-platform binary[C]// The 33rd ACM/IEEE International Conference on Automated Software Engineering. ACM, 2018: 896-899.

[57] JIANG S, FU C, QIAN Y K, et al. IFAttn: Binary code similarity analysis based on interpretable features with attention[J]. Computers & Security, 2022, 120: 102804.

[58] KIM D, KIM E, CHA S K, et al. Revisiting Binary Code Similarity Analysis Using Interpretable Feature Engineering and Lessons Learned[C]// IEEE Transactions on Software Engineering. IEEE, 2022: 1661-1682.

[59] JIA A, FAN M, XU X, et al. Cross-Inlining Binary Function Similarity Detection[C]// The IEEE/ACM 46th International Conference on Software Engineering. ICSE, 2024: 1-13.

[60] KINABLE J, KOSTAKIS O. Malware Classification based on Call Graph Clustering[J]. Journal in Computer Virology, 2010, 7: 233-245.

[61] MASSARELLI L, DI LUNA G A, PETRONI F, et al. Investigating graph embedding neural networks with unsupervised features extraction for binary analysis[C]// the 2nd Workshop on Binary Analysis Research. BAR, 2019.

[62] YU Z P, CAO R, TANG Q Y, et al. Order matters: Semantic-aware neural networks for binary code similarity detection[C]// The AAAI Conference on Artificial Intelligence. AAAI, 2020: 1145-1152.

[63] CHANDRAMOHAN M, XUE Y X, XU Z Z, et al. Bingo: Cross-sarchitecture cross-os binary search[C]// The 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016: 678-689.

[64] LI Y J, GU C J, DULLIEN T, et al. Graph Matching Networks for Learning the Similarity of Graph Structured Objects[C]// The 36th International Conference on Machine Learning. ICML, 2019: 3835-3845.

WEI Youyuan, born in 2000, postgraduate, is a member of CCF(No. T7853G). His main research interests include binary code similarity detection, malicious code identification and so on.

SONG Jianhua, born in 1973, Ph.D, professor, postgraduate supervisor, is a member of CCF (No. 27785M). Her main research interests include network and information security and so on.

(责任编辑:何杨)