arXiv:2601.00429v1 [cs.SE] 1 Jan 2026

论抄袭与软件抄袭

Rares Folea[1,2][0000−0002−4936−9082] 和 Emil Slusanschi[1][0000−0003−0222−1002]

[1] 布加勒斯特理工大学计算机科学与工程系，自动控制与计算机学院，罗马尼亚[2] 激光与加速器工程及应用博士研究生院（S.D.I.A.L.A.）

摘要。 本文探讨了自动检测软件相似性的复杂性，关注数字制品带来的独特挑战，并介绍了 Project Martial——一款用于检测代码相似性的开源软件解决方案。本文通过考察学术界和法律领域，列举了应对软件抄袭的一些现有方法，其中包括重要的诉讼和法院裁决；这些案例塑造了人们对商业应用程序中软件版权侵权的理解。此外，我们根据可用制品对检测挑战进行分类，并综述了文献中先前研究的技术，包括基于指纹识别、软件出生印记或代码嵌入的解决方案，同时举例说明其中一部分技术如何应用于 Project Martial。

关键词： 代码相似性、软件抄袭、软件指纹、软件出生印记、代码嵌入、抄袭检测

1 引言

从最广义的定义来看，抄袭[3] 是将他人的作品作为原创作品呈现，却未注明适当出处的行为。历史上有许多抄袭 案例，而有记载的最早一次使用*plagiarius* [31] 一词，发生在罗马诗人马提亚尔生活的年代。当时，他的诗歌和讽刺短诗被一字不差地抄录，并被一些籍籍无名的作者当作自己的作品发表。他被认为是第一个主张著作权的人；而在那个古代世界，知识产权和版权法不仅未得到执行，甚至根本无人提及。

软件抄袭只是抄袭的众多分支之一，但由于软件开发几乎完全处于数字化、快节奏的环境中，复制行为可以非常迅速地完成，因此与软件抄袭相关的行为数量正在迅速增加。抄袭软件的主要动机，是逃避从头开发新颖、原创解决方案所需付出的努力。

[3]这个词源自拉丁语 *"plagiarius"*，意为施刑者、压迫者或绑架者。

软件抄袭是一个普遍存在的问题，影响着学术界——学生可能会为了取得更好的成绩而抄袭——以及商业领域，在那里，知识产权诉讼的索赔金额可能高达数十亿美元。尽管涉及的利害关系大不相同，抄袭的核心行为始终相同：非法宣称自己是他人作品的作者。

著名抄袭案例会引起公众的广泛关注，软件抄袭也不例外。值得注意的是，这些争论和指控可能超出软件工程领域。一些广受关注的指控已被改编成电影，例如 *"Browser Wars"* 和 *"The Billion DollarCode"*。相比之下，入门计算机科学课程中的抄袭指控可能成为学生讨论的话题，引发有关道德以及大学委员会决策的讨论。

本文**第 2 节** 探讨软件抄袭相较于其他领域的独特之处。**第 3 节** 考察抄袭对学术界的影响，而**第 4 节** 讨论与抄袭相关的法律影响。**第 5 节**旨在根据源代码或二进制代码是否可用，对可能用于检测抄袭的解决方案类别进行分类；在**第 6 节**中，我们介绍一些用于自动检测的技术。最后，**第 7 节** 讨论未来研究的前景，并介绍 Project Martial，这是一种有助于检测代码相似性的开源软件解决方案。

2 与其他类型抄袭的比较

如何检测软件抄袭，以及它与其他类型的抄袭有何不同，这是一个值得探讨、却难以回答的问题。软件开发存在依赖关系，因此很难相信一套简单的规则就能涵盖所有可能的情况。正如后文所述的例子，美国最高法院在一项裁决中承认，仅复制代码的声明部分而不复制实现部分的情况，不应被视为抄袭。由此直接得出的结论是，该裁决指出，应用程序接口不构成版权侵权。这使软件行业对抄袭的认定具有独特之处，不同于音乐或文学等其他产业；在这些领域，作品的声明部分与实现部分之间并不存在明确的对应关系。

总的来说，对于这个问题的难度尚无共识 [10] ，关于与识别其他作品中的抄袭相比，检测软件抄袭是否容易，正反双方都有许多论据。

一方面，有观点认为，相较于软件，若两个来源是以自然语言文本写成，人们更容易判断它们之间的相似度。这是因为在人类语言中，语料库中的参考文献和引文可能会导致文本有 10% 的相似度，但这类文本通常不会被视为抄袭；而在软件开发中，类似的工作方法可能难以界定，因为“相同的理论知识”和“直接复制”之间的区别更难以确定，

尤其是在正确解决问题的方法寥寥无几的情况下 [29]。Ken Thompson 在1984年图灵奖演讲中展示了一个广为人知的代码重复案例，却没有引发抄袭质疑。 [33], 在提到他与 Dennis Ritchie 的合作时，他说，在两人长期合作期间，只发生过一次工作协调失误。Thompson 指出，他发现两人都“写出了同一个 *20* 行的汇编语言程序”。他总结道：“我比较了源代码，惊讶地发现它们逐字符完全一致”。

另一方面，认为软件抄袭可能更容易识别的论点，主要围绕二者之间的语言复杂性展开； [20]认为，人类语言文本包含实际上无限数量的可用词语，无论采用语料库内还是语料库外的抄袭方法均是如此，而代码抄袭则受限于给定编程语言的最小词汇表所提供的一组明确定义的关键字。

为了给这一问题增加一些复杂性，当出现无法获取可疑程序源代码、唯一可用资源仅为可执行二进制文件的情形时，便会产生一项额外约束。由于汇编指令对于按人类思维结构进行的分析而言要混淆得多，试图评估可执行文件的原创性，比研究源代码 [43]是一个困难得多的问题。

尽管如此，不提供源代码并非混淆分析的唯一可能方式。这也可以通过对原始代码进行迭代修改来实现，这些修改不会改变程序的行为，但会使其看起来不同。已开发出众多混淆技术，主要可分为控制导向和数据导向两类；从符号重命名或布尔拆分器等朴素技术开始（参见清单中的示例），进而发展到更复杂的改动，例如添加噪声指令、重排独立指令、循环展开（清单A1）、更改控制流条件分支、分支反转（清单A2）或聚合多条指令（清单A3）。这些以Go编程语言语法提供的示例，只是一些简单的学术样本，能够以某种方式改变 [4] 原始代码，使所得代码看起来大不相同。

此外，研究 [24, 42] 还探讨了通过添加一组混淆措施来保护软件所 encapsulated 的知识产权，这些措施主要旨在使仅能接触混淆结果的人员极难解码原始逻辑。这类复杂化技术既可应用于源代码级别，也可应用于二进制级别。正如

[4] 值得一提的是，某些修改（例如循环展开或向量化）具有多种功能。这些技术不仅可以用作混淆手段，也可以作为优化技术，使代码更加高效，而非刻意改变代码的外观。不过，本文仅研究以生成代码混淆为目的的修改。

上述介绍的技术一样，由于这些过程的结果是在不改变软件行为的情况下改变其外观，因此也可以用来误导软件抄袭检测工具，使其错误检测相似度 [35]。

鉴于近期基于大语言模型的解决方案的发展，关于软件抄袭必须提出一些特别的考量。这些解决方案已被证明能够有效处理大型代码片段。大型预训练语言模型已经被用于修改原始源代码，并且已被证明能够有效地进行修改，使其仍然不被检测到。

这在 [19]中有详细说明，该文得出结论认为，基于大语言模型的模型"具有生成复杂文本输出的巨大潜力，同时又不易被抄袭检测软件有效识别"。最后，使用这项新技术获得的初步结果似乎表明，与大型神经语言模型在生成令人信服的高质量释义方面所能达到的效果相比，大语言模型出现前的工具仅仅"触及了可能性的表面" [40]，并承认即使是人类，检测机器生成的混淆文本的能力"似乎也会随着模型规模的增大而下降，因为它们能够改变句子结构和词序，而不是仅仅替换单个词语"。

3 学术界

在学术界，抄袭被视为"逃避学习"[41]，因为它"使学生能够在尚未形成理解和批判性评价所提交材料的能力的情况下，就因评估内容的重要部分而获得学分。"，并不是罕见现象。由于计算机科学专业的学生人数仍在逐年增加，新的挑战也随之出现。Data USA指出，2020年授予的学位总数超过五万名毕业生 [38]，，年增长率为10%；英国计算机学会注意到， [1] 英国在2022年申请学习计算机科学的人数从前一年的14万增加到当年的近16万，这一趋势表明，人们对计算机科学相关学科的兴趣持续增长。

鉴于其规模，显然有必要改进自动化工具，以便识别可能的代码抄袭案例。正如 [10]所指出的："学生太多，教职员工太少 *[...]*。自动化抄袭检测将大有帮助"。

首先，各类作品 [16]都承认，使用现有工具识别论文和报告中的自然语言抄袭存在困难，但软件抄袭有其独特性。一项关于软件抄袭这一学术

[5] 大语言模型在更多类型的作业中表现出色，例如自由形式的自然语言论文，而不仅仅是修改软件程序；这也给其他领域的抄袭问题带来了额外挑战。[6] 幸运的是，新型基于大语言模型的工具也有其积极的一面。有证据表明，这些工具通过协助开发者（以及其他人），有助于 [12] 提高生产力。如今，OpenAI 的 GPT-4 [28], Google 的 Gemini [11] 以及其他工具，都能根据简单的提示生成数百行代码。

问题的调查 [10] 显示，只有 8% 的大学教职员工使用自动化软件方法尝试检测抄袭，而占绝大多数的 80% 仍依靠直觉和／或个人经验来应对这一问题。有人指出，对抄袭本质的理解过于复杂，可能导致解决方案“更多着眼于预防而非检测抄袭” [41]。

最常用的自动软件抄袭检测系统是 MOSS[7]。其维护者将其描述为“一种突出显示程序中值得更彻底检查的组成部分、从而为教师和教学人员节省大量时间的方法” [32]。文中强调，“在程序的特定部分经过检查之后，可疑代码最初是由 *MOSS* 还是由人工发现的，不应有所区别”并且“抄袭的论证应能独立成立”。多位研究者认同这一伦理理念，正如 [40] 所指出的，虽然自动抄袭检测可以“指出潜在的抄袭案例，但此类案例应由专家团队作出最终决定”，因为同一著作在 [40]中警告说“误报导致研究人员被错误指控的情况可能永远毁掉他们的职业生涯因此，“所有案例都应在作出任何最终裁决之前经过仔细评估”。

同样值得关注的是 Mossad [13], 等研究，这些研究表明，可以自动生成软件版本的变体，其唯一目的就是击败抄袭检测器。

学术界与产业界的核心区别，在于参与者在抄袭过程中愿意投入多少精力。在商业环境中，若需要应对侵犯版权的指控，往往会有更多资源和动机；而在教育环境中，“学生为掩盖抄袭而付出的努力可能要少得多” [18]。

4 法律层面与诉讼

科技行业的快速发展促成了法律领域一个相对较新的分支——专门处理技术相关诉讼。大多数诉讼都围绕确定某项商标或专利是否符合资格展开。知名商标案件既包括基于硬件的主张，例如苹果公司曾在中国主张拥有其 iPad 商标的所有权；软件相关主张也不例外。在企业界，此类知名案件和软件抄袭指控层出不穷。

软件著作权和抄袭指控很少能轻易裁定，其中大多数都需要经过审判才能作出判决；在某些特殊情况下，甚至还需要美国最高法院介入。

[7] MOSS 是“软件相似度测量”（Measure Of Software Similarity）的缩写，是一种用于判定代码相似性的自动化系统。https://theory.stanford.edu/%7Eaiken/moss/

Oracle America Inc 诉 *Hewlett Packard Enterprise Co* 案双方最终达成和解。争议涉及“*Oracle* 所拥有的 *Solaris* 操作系统遭未经授权的软件更新”，Hewlett Packard Enterprise 被指控“未经许可使用其受版权保护的软件，并压低 *Oracle* 的支持服务定价” [17], ，目的是降低新软件开发的成本。据针对电信公司 *Verizon*提起的诉讼指称，该公司在分发给消费者的无线路由器中使用了 *BusyBox* 编程，但未按 GPL [39]许可证的要求向客户提供 BusyBox 源代码。本案也以遵守许可证为条件达成和解。

在*Google LLC* 诉 *Oracle America Inc.* 案中，美国最高法院考量了多个因素，指出 Google 对 Java API 的使用具有转化性，而其复制的接口数量仅限于最低限度，这是为了让开发者能够运用从 Java 编程语言中积累的知识，在 Android 提出的新移动开发框架中开展工作。案件准备期间，Google 辩称，“程序之间的开放接口是我们如今使用的许多服务和产品，以及我们尚未想象出的技术的基石。如果 *Oracle* 胜诉，将颠覆科技行业一贯处理软件接口这一重要问题的方式。这将首次赋予版权所有者一种垄断权力，使其能够阻碍新实现和应用程序的诞生。此举还会让开发者和初创企业更难以更高成本开发出供人们使用的更多产品。” [3]与此相反，Oracle 认为，这场审判始于一个简单的主张：偷窃——无论能给窃贼带来多大便利——都是不可接受的[4]，并且“声称其盗用并明确商业使用市场上已有技术的行为， *somehow* 受合理使用原则保护”，这样的论点“几乎会彻底消除版权，因为几乎所有复制行为都将被视为 <<合理>> 使用”[4]。

陪审团认为，复制的代码部分“与不受版权保护的思想不可分割地结合在一起”，而 Google 仅复制了整个应用程序编程接口的 0.4%，这有利于认定构成合理使用。尽管如此，审判过程中证明，Android 平台并非 JavaSE 的替代品或竞争对手。陪审团中也有人对此次审判持异议，认为法院本应处理 Oracle 的代码是否可受版权保护这一问题，以及 Google 是否因复制代码而获得多项优势并严重损害了 Oracle 的合作关系。

5 类

如前文所述，识别软件抄袭的问题可能需要根据可用资源采取截然不同的方法。我们据此将问题分为两大类：能够访问源代码（**源代码**）的检测，以及只能访问二进制代码（二**进制代码**）的检测。请注意，虽然能够访问**源代码** 也意味着

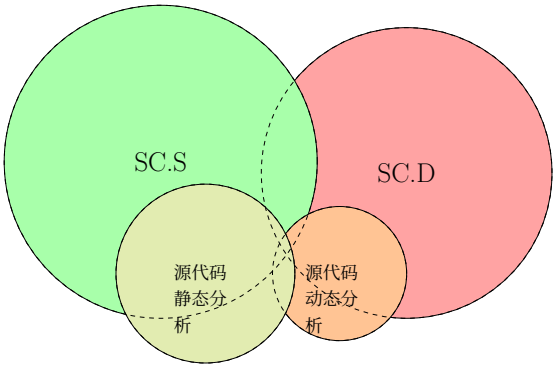

图1：基于问题类别的技术交集。

能够访问二**进制代码** 资源，只需获取源代码并进行编译（对于原生实现支持编译为机器语言的语言），或进行打包（对于解释型语言）即可；但在某些特殊情况下，用于将代码转换为机器可执行形式的工具可能无法获取、未公开或属于专有工具。这就引出了一个非常小众的问题类别：无法从源代码得到完全一致的二进制代码。

自然，在可以访问源代码时，分析代码的语义会容易得多，因为与二进制代码中的特定于机器的指令相比，高级编程语言对程序需要遵循的流程描述得更为清晰 [6]。图1展示了可用于应对各项挑战的技术交集。

因此，**源代码** 类问题可能有助于开发出更好的软件抄袭检测器，因为额外的源代码访问权限使得我们能够采用更先进的方法来处理这些问题。例如，这些方法可以分析代码的静态表示**源代码静态分析**，也可以通过监控程序的执行进行动态分析**源代码动态分析**。

类似地，我们可以将**BIN.S**定义为直接采用静态方法评估机器代码的一类方法。对称地，**BIN.D** 可定义为直接动态评估机器代码的一类方法，但它几乎与**SC.D**完全重叠[8]。

自动软件抄袭检测的问题远未找到通用的解决方案，但经过多年研究，学界在探索自动识别潜在抄袭案例的方法方面取得了一些进展。传统方法只有在发现相似性时才会判定为抄袭[9]。

[8] 我们发现的唯一差别在于：从源代码转换为二进制代码时，是否启用了调试跟踪信息和符号；若只能获取二进制代码，这些信息可能会缺失。[9]大多数解决方案都专注于 *identifying similarities* ，只有在发现相似度时才判定为抄袭；而 [43],的研究则采用了不同的方法。

6 自动抄袭检测技术

基于**指纹识别的** 方法可以自动发现程序相似性，但在大多数情况下，这些方法也缺乏令人信服的可解释性，无法确定某些程序为何相似。一种成熟的算法是 Winnowing[30], ，它是一种指纹识别解决方案，会将文件划分为 k 元组，也是 MOSS 软件的核心思想；不过，在其他研究中，指纹思想也有许多成功的应用，既用于**源代码** [9, 27, 6, 9, 27, 6], ，也用于二**进制代码** [6]。根据善意原则，仍需由人工审查检测引擎标记的代码片段，并判断该部分是否确实属于抄袭 [32, 40]。

技术	源代码静码静态分析			
代码分析（句法、语义）	✓			
基于 API 的	✓		✓	
分析		✓		✓
指纹	✓	✓	✓	✓
软件 软件出生印记	✓	✓	✓	✓
Code 嵌入	✓	✓	✓	✓
基于大语言模型	✓			

表 1：抄袭检测中使用的主要技术，以及已应用这些技术来检测抄袭的问题类别。它们已被应用于检测抄袭的问题。

软件诞生标记[10] 方法旨在克服软件指纹识别的一些缺点。与基于指纹的算法一样，软件行为印记的优势在于，它们既可应用于**源代码**，即源代码 [26, 9, 6, 35,25] ，也可应用于二**进制代码**，即仅提供二进制代码的情况 [6, 36, 35, 37, 23]。对于复杂程序，可以通过其他机制增强软件诞生标记技术，例如通过孪生网络监督构建过程 [34]，以捕捉多线程应用中机器指令非确定性调度所导致的扰动。

其出发点是 *"as long as we can find one dissimilarity, programs are semantically different"* *"if we cannot find any it is likely , a plagiarism case"*

但强调，。这是一种更具雄心的方法，但也有缺点：它缺乏可解释性，因此使得按照 [40] 方法论提出的人工分析明显更加困难。[10]软件行为印记通常被计算为一组相对不变的特征，这些特征从所分析的算法中提取，整体目标是唯一标识该程序。提取软件行为印记后，两个程序之间的相似度只需通过一个函数计算，该函数接收两个由此得到的软件行为印记并对其进行比较。

代码嵌入是另一种强大的机制，可用于相似度检测，即通过程序的数值向量表示对其进行分析。它们旨在以紧凑的格式捕捉代码片段的语义含义和句法结构，并有望在多维分析空间中将功能相似的代码片段彼此靠近。虽然大多数方法旨在解决**源代码**问题 [8], ，但也可应用于二**进制代码** [14] 类。

对于研究 **SC.S** 问题， [21, 7] 中的研究采用了一种基于**API-labeled dependency** 图分析的检测方法。

一个颇具前景但尚未得到探索的领域，是将大语言模型用作软件自动抄袭检测的技术。由于大语言模型在处理源代码方面已被证明具有良好的性能，这是一项很有前景的技术，可用于处理**SC.S** 问题。

请注意，最终应用程序可能会结合不同解决方案中的多种方法来分析相似度。只要这些方法假定问题出在**SC**中，那么所介绍的所有技术都可以应用：**SC.S**、**SC.D**、**BIN.S**、**BIN.D**。如果问题出在**BIN**中，则可用技术可以属于**BIN.S** 和**BIN.D**。现有主要方法的概要已汇总于表 1 中。

7 Project Martial：当前与未来的工作

作者在相似性检测领域的部分研究与贡献，着眼于寻找既能处理静态代码、又能执行动态分析的新方法（适用于**SC** 和**BIN** 两类）。Project Martial[11]是一个不断发展的实验性开源项目，旨在为检测软件抄袭提供自动化辅助。

ProjectMartial 提供了一个模块化框架，可轻松添加新的检测工具，也可微调现有模型的元参数，使其更好地适应需求。目前，该工具集包含三个独特的分析器，预计到 2024 年底和 2025 年初还会增加更多分析器。

主要分析器属于**源代码** 类别，并在对源代码进行静态分析时，运用自然语言处理技术计算注释中可读部分的相似度 [15]。该方法使用 Transformer（ RoBERTa [22] 和通用句子编码器 [5]）对注释内容进行嵌入，然后通过计算余弦相似度来检测潜在的相似内容。作为该分析器未来的工作方向，我们希望最终将这些技术扩展到没有人工注释的代码上，例如不带注释或其他指令的代码。在这一领域，基于大语言模型的新型代码分析技术已展现出巨大潜力，有望帮助为机器原生源代码添加以人类语言编写的注释；本质上，这相当于在应用现有模型之前对源代码进行预处理。

第二种分析器完全基于对机器可读注释（例如代码检查器指令）的检查，通过计算多维空间中的距离来

[11] https://github.com/raresraf/project-martial

计算估算的代码相似性，其依据是源代码的独热编码表示。 [15]。这种技术也属于**源代码** 类别。

Project Martial 中的第三种方法使用动态代码复杂度 [14]来检测软件抄袭，因此为 二**进制代码** 类别中的技术作出了贡献。我们计划扩展现有的软件行为印记（目前基于有限的度量指标子集，例如 CPU 周期、分支预测或缓存未命中统计），以捕获越来越多的性能指标，同时致力于优化相似度检测函数。

基于网络遥测检测相似性比较是另一个研究方向。初始方法侧重于寻找关系数据库之间的共同点，并通过检查从客户端网络数据包中提取的指纹，评估是否能够发现和比较数据库内部信息。通过记录和检查这些数据包中包含的独特特征——例如协议头、SQL 查询模式和特征签名——仅依靠有效载荷内容及通信方式来识别数据库之间的关联是可行的。

参考文献

1. BCS，T.C.I.f.I.：“今年申请英国计算机科学学位的人数创下纪录”。
https://www.bcs.org/articles-opinion-and-research/record-numbers-have-applied-for-uk-computer-science-degrees-this-year/（2022年），[在线；访问日期：1月29日。2023]。Biderman，S.，Raff，E.：使用预训练语言模型欺骗 MOSS 检测。载于：第31届 ACM 信息与知识管理国际会议论文集。第2933–2943页（2022年） 3. Blog，G.：“支持开放式创新的理由”。https://blog.google/outreach-initiatives/public-policy/case-for-open-innovation/（2020年），[在线；访问日期：1月29日。2023]。Blog，O.：“版权买者自慎”。
https://www.oracle.com/corporate/blog/oracle-vs-google-021220 2020 [29 2023] .html（），在线；访问日期：1月。。Cer，D.，5 Yang，Y.，Kong，S.y.，Hua，N.，Limtiaco，N.，John，R.S.，Constant，N.，Guajardo-Cespedes，M.，Yuan，S.，Tar，C.，等：通用句子编码器。arXiv 预印本 arXiv:1803.11175（2018年） 6. Cesare，S.，Xiang，Y.：软件相似性与分类。Springer Science & Business Media（2012年） 7. Chae，D.K.，Ha，J.，Kim，S.W.，Kang，B.，Im，E.G.：软件抄袭检测：一种基于图的方法。载于：第22届 ACM 信息与知识管理国际会议论文集。第1577–1580页（2013年） 8. Chen，Z.，Monperrus，M.：关于源代码嵌入的文献研究。arXiv 预印本 arXiv:1904.03061（2019年） 9. Chilowicz，M.，Duris，E.，Roussel，G.：用于源代码相似性检测的语法树指纹识别。载于：2009 IEEE 第17届程序理解国际会议。第243–247页。IEEE（2009年） 10. Chuda，D.，Navrat，P.，Kovacova，B.，Humay，P.：软件抄袭问题：学生视角。IEEE 教育学报 **55**（1），22–28（2011年） 11. DeepMind，G.：Gemini——Google DeepMind。https://deepmind.google/technologies/gemini
，
（访问日期：2023年12月15日）

12. Dell’Acqua, F., McFowland, E., Mollick, E.R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., Lakhani, K.R.：应对崎岖的技术前沿：人工智能对知识工作者生产力和质量影响的现场实验性证据。哈佛商学院技术与运营管理部工作论文（24-013）（2023） 13. Devore-McDonald, B., Berger, E.D.：Mossad：击败软件抄袭检测。《美国计算机协会程序设计语言会议论文集》**4**（面向对象编程、系统、语言与应用会议）（2020年11月）。https://doi.org/10 3428206 14 1145 3428206, 10 1145 . / https://doi.org/ . / . Folea, R., Iacob, R., Slusanschi, E., Rebedea, T.：基于复杂度的代码嵌入。载于：国际计算集体智能会议。第256–269页。施普林格（2023） 15. Folea, R., Slusanschi, E.：代码注释：一种识别源代码相似性的方法。《数学》**12**（7）（2024）。https://doi.org/10.3390/math12071073, https://www.mdpi.com/ 2227 7390 12 7 1073 - / / / . Foltynek, T., Dlabolova, D., 16 Anohina-Naumeca, A., Razı, S., Kravjar, J., Kamzola, L., Guerrero-Dib, J., Celik, O., Weber-Wulff, D.：测试抄袭检测支持工具。《高等教育教育技术国际期刊》**17**, 1–31（2020）17. “惠普企业在版权诉讼判决金额达3000万美元后与Oracle和解”。https://www.reuters.com/legal/litigation/hp-enterprise-settles-oracle-copyright-lawsuit-after-30-mln-verdict-2023-01-18/（2023）， [在线；访问日期：1月29日 2023] 18 。Joy, M., Luck, M.：编程作业中的抄袭。《IEEE 教育学报》**42**（2），129–133（1999）19. Khalil, M., Er, E.：ChatGPT会让你被抓到吗？重新思考抄袭检测。arXiv预印本 arXiv:2302.04335（2023）20. Lancaster, T., Culwin, F.：抄袭检测引擎的分类。《信息与计算机科学教学与学习创新》**4**（2），1–16（2005）21. Liu, C., Chen, C., Han, J., Yu, P.S.：GPLAG：通过程序依赖图分析检测软件抄袭。载于：第12届美国计算机协会知识发现与数据挖掘特别兴趣小组国际会议论文集。第872–881页（2006）22. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.：RoBERTa：一种稳健优化的BERT预训练方法。arXiv预印本 arXiv:1907.11692（2019）23. Lu, B., Liu, F., Ge, X., Liu, B., Luo, X.：一种基于动态操作码 n 元语法的软件诞生标记。载于：国际语义计算会议（ICSC 2007）。第37–44页。IEEE（2007）24. Madou, M., Anckaert, B., De Bus, B., De Bosschere, K., Cappaert, J., Preneel, B.：论源代码变换对二进制代码混淆的有效性。载于：软件工程研究与实践国际会议论文集（SERP06）。第527–533页。CSREA出版社（2006）25. Myles, G., Collberg, C.：通过全程序路径诞生标记检测软件盗窃。载于：信息安全：第七届国际信息安全会议（ISC 2004），帕洛阿尔托，美国，加利福尼亚州，2004年9月27日至29日。会议论文集7。第404–415页。施普林格（2004）26. Myles, G., Collberg, C.：基于 K 元语法的软件诞生标记。载于：2005年美国计算机协会应用计算研讨会论文集。第314–318页（2005）27. Narayanan, S., Simi, S.：使用基于指纹的距离度量方法进行源代码抄袭检测与性能分析。载于：2012年第七届国际计算机科学与教育会议（ICCSE）。第1065–1068页。IEEE（2012）

28. OpenAI：GPT-4 技术报告（2023年）29. Rosales, F., Garc´ıa, A., Rodr´ıguez, S., Pedraza, J.L., M´endez, R., Nieto, M.M.：编程作业中的抄袭检测。《IEEE 教育学报》**51**(2), 174–183 (2008) 30. Schleimer, S., Wilkerson, D.S., Aiken, A.：Winnowing：用于文档指纹识别的局部算法。载于：2003 年 ACM SIGMOD 数据管理国际会议论文集。第 76–85 页（2003）31. Seo, J. M.：马提亚尔作品中的抄袭与诗歌身份。《美国语文学杂志》第 567–593 页（2009）32. 抄袭检测。https://theory.stanford.edu/ aiken/moss/, （访问日期：2023年09月23日）33. Thompson, K.：关于信任信任的思考。《ACM 通讯》**27**(8), 761–763 (1984) 34. Tian, Z., Wang, Q., Gao, C., Chen, L., Wu, D.：通过孪生神经网络检测多线程程序中的抄袭。IEEE Access **8**, 160802–160814（2020年）35. Tian, Z., Zheng, Q., Liu, T., Fan, M., Zhuang, E., Yang, Z.：基于动态关键指令序列的软件行为印记进行软件抄袭检测。《IEEE 软件工程汇刊》**41**(12), 1217–1235（2015）36. Tian, Z., Zheng, Q., Liu, T., Fan, M., Zhuang, E., Yang, Z.：基于动态关键指令序列的软件行为印记进行软件抄袭检测。《IEEE 软件工程汇刊》**41**(12), 1217–1235（2015）。https://doi.org/10 2454508 37 1109 2015 . /TSE. . . Ullah, F., Wang, J., Farhan, M., Habib, M., Khalid, S.：采用机器学习方法检测多编程语言中的软件抄袭。《并发与计算：实践与经验》**33**(4), e5000（2021年）38. USA, D.：“计算机科学；科学、技术、工程和数学专业”。https://datausa.io/profile/cip/computer-science-110701, [在线；访问日期：29 2023] Jan. 。开源法律团体再次出手，起诉 Verizon，涉及 BusyBox——39 Computerworld。https://www.computerworld.com/article/2537947/open-source-legal-group-strikes-again-on-busybox–suing-verizon.html, （访问日期：2023年09月23日）40. Wahle, J.P., Ruas, T., Kirstein, F., Gipp, B.：大语言模型如何改变机器改写型抄袭。arXiv 预印本 arXiv:2210.03568（2022年）41. Warn, J.：抄袭检测软件：并非灵丹妙药！《高等教育研究与发展》**25**(2), 195–208（2006）42. Wu, Z., Gianvecchio, S., Xie, M., Wang, H.：Mimimorphism：一种二进制代码混淆新方法。载于：第 17 届 ACM 计算机与通信安全会议论文集。第 536–546 页（2010）43. Zhang, F., Wu, D., Liu, P., Zhu, S.：基于程序逻辑的软件抄袭检测。载于：2014 年 IEEE 第 25 届软件可靠性工程国际研讨会。第 66–77 页。IEEE（2014）

附录 A：混淆技术的代码清单

```
s := []整数{1, 2, 3, 4, 5, 6}
-     i := i i
    求和 +=s[i]
- }
```

```
s := []整数{1, 2, 3, 4, 5, 6}
+     i := i i
    总和 +=s[i]
+   总和
+   总和
+ }
```

代码清单 A1：循环展开混淆示例。

```
  s, err := f()
-
-   返回 nil
- }
-
```

```
  s, err := f()
+
+   //
+ }
+
```

代码清单 A2：分支反转混淆示例。

```
- if '0' <=          '9' {
-   c =       - 'A' + 10)
- } elif case 'A' <= c && c <= '
-   c =       - '0')
- }
- u
```

```
+
+   if '0' <= c
+     && c <= '9' {
+     返回 c     - '0'
+   if 'A' <= c && c <= 'F' {
+       返回 c     - 'A' + 10
+   }
+   返回 0
+ }
+ u := unhex
```

代码清单A3：通过将多条指令汇总到一个专用方法中，对原始代码进行混淆的示例。