软件漏洞数据集的数据质量

Roland Croft*†、M. Ali Babar*†、M. Mehdi Kholoosi*†
* 计算机科学学院、CREST、阿德莱德大学，澳大利亚， {firstname.lastname}@adelaide.edu.au
† 网络安全合作研究中心，澳大利亚

arXiv:2301.05456v1 [cs.SE] 13 Jan 2023

摘要——**在软件安全领域，利用基于学习的技术实现自动化软件漏洞检测一直备受关注。这些数据驱动解决方案依托用于训练和基准测试的大型软件漏洞数据集。然而，我们发现，目前人们并未充分考虑支撑这些解决方案的数据质量，因而影响了所产生结果的可靠性和价值。尽管人们对软件漏洞数据准备所面临挑战的认识日益加深，但针对软件漏洞数据质量可能造成的负面影响，相关研究仍然很少。例如，我们尚未确认漏洞标签是否正确或一致。本研究旨在弥补这些不足：我们考察了四个最先进的软件漏洞数据集的五项固有数据质量属性，以及相关问题对软件漏洞预测模型造成的后续影响。令人意外的是，我们发现所有受分析的数据集都存在某些数据质量问题。具体而言，我们发现现实世界数据集中有 20%–71% 的漏洞标签不准确，17%–99% 的数据点存在重复。我们观察到，这些问题可能对下游模型造成重大影响，既可能阻碍有效的模型训练，也可能抬高基准测试性能。我们倡导采取措施克服这些挑战。我们的研究结果将有助于今后更好地考量和评估软件漏洞数据质量。**

索引词—**软件漏洞、数据质量、机器学习**

I. 引言

软件漏洞检测是实现安全软件系统的一项重要任务 [1]。然而，传统的漏洞检测技术（例如基于规则的方法）在可扩展性和假阳性率方面表现不佳 [2]。因此，许多研究人员开始借助人工智能（AI）和机器学习（ML）的技术进步，以支持自动化软件漏洞检测 [3]。近期研究在这一方向上取得了巨大成功 [4]–[8],，其性能超越了传统方法 [9]。我们将这些基于学习的技术称为软件漏洞预测（SVP）。与任何数据驱动任务一样，SVP 高度依赖数据。为了学习漏洞的复杂特征，我们需要大量经过标注、区分为易受攻击或非易受攻击的代码数据集。

然而，软件漏洞数据收集并非易事 [10]。在现实世界中，软件漏洞的已标注样本很难获得，因为这类样本十分稀缺[11], 、文档记录不完善 [12], ，且仅限于已报告的漏洞 [13]。因此，许多研究人员投入了大量工作来构建大规模软件漏洞数据集 [14]–[17]。然而，我们发现，针对理解软件漏洞数据质量的相关工作却相对较少。尽管数据数量很重要，但有效的机器学习系统需要足够的数据质量 [18]。尽管人们日益认识到软件漏洞数据准备所面临的挑战 [10], ，但针对系统性理解这些挑战可能如何影响数据质量，并进而影响下游软件漏洞分析的可靠性，相关工作仍然相对有限。

对数据质量缺乏了解，是推进软件保障以防范漏洞的关键障碍。数据质量是任何数据驱动系统不可或缺的组成部分：垃圾进，垃圾出 [19]。某些数据偏差或错误信息可能会使基准测试性能结果产生误导 [20]–[22]。如果模型未使用公平且符合现实的数据进行训练，就可能无法泛化到现实场景 [17], [23], [24], 。

因此，我们着手了解软件漏洞数据集的数据质量本质。为此，我们重点关注数据本身所固有的固有数据质量属性。表 I 展示了我们系统分析的五项数据质量属性：准确性、唯一性、一致性、完整性和时效性。对于每项属性，我们都对其在现有数据集中的普遍程度进行了测量，并分析了所观察到问题的成因。我们的研究结果表明，即使是最先进的数据集，也存在相当严重的数据质量挑战。正如预期，我们发现这些问题会对最先进的软件漏洞预测模型的训练集和基准测试产生显著的负面影响。我们的研究结果具有重要意义：

- 数据质量问题可能会限制软件漏洞预测模型能够学习到的模式。我们发现，真实世界数据集的易受攻击类别在标签正确性方面可能存在严重问题。约有 20%—71% 的漏洞标签不准确。此外，最多有 47% 的标签存在不一致。这些问题会导致模型学习到错误或不充分的模式。

- 软件漏洞基准数据集可能导致评估性能虚高。一些数据集表现出较高的数据重复率，介于 17% 至 99% 之间。因此，数据泄漏会导致模型在使用标准测试设置时报告虚高的性能。移除此类重复项后，评估性能最高下降了 82%。

为了从基于学习的软件漏洞分析中获得最可靠的结果，我们必须考虑并解决数据质量问题。虽然可以通过基于规则的检测和

表 I ISO/IEC 25012 [25] 定义的固有数据质量属性。

属性	定义	解释
准确性	数据具有能够正确表示某一预期属性真实值的属性的程度 概念或事件。	正确标注。
唯一性	记录中不存在重复的程度。	没有重复值。
一致性	数据属性不存在矛盾且与其他数据保持一致的程度。	一致性标注。
完整性	与某个实体相关的主体数据，其所有预期属性和相关实例均具有取值的程度。	没有缺失值。
时效性	数据具有适当时效性属性的程度。	并非过时数据。

删除轻松解决一些已发现的质量问题，但其他问题无法通过这种方式补救。作为一个社区，我们必须专注于开发构建高质量软件漏洞数据集所需的知识和工具。我们的贡献有两方面：

1) 我们阐明了软件漏洞数据集中所观察到的数据质量问题的性质及其成因。同时，我们展示了这些问题对软件漏洞预测的相应影响。此类研究突出了数据质量问题以及缓解这些挑战的必要性。此外，我们的洞见有助于克服数据质量问题；对此，我们已在讨论部分提供了相关方向。

2) 我们提出并实施了用于衡量软件漏洞数据集数据质量的方法。这些工作可以帮助从业者考虑并开展数据质量评估。我们已提供本研究的复现包，以协助开展此类工作 [26]。

II. 背景与动机

A. 软件漏洞数据准备

软件漏洞预测（SVP）模型采用程序分析技术，从历史示例中自动学习软件漏洞模式 [27], [28]。由于源代码具有非结构化特性，研究人员发现，采用能够从程序语法和语义中学习的深度学习（DL）技术效果最佳 [3], [7], [8], [24]。

因此，SVP 是一个数据需求量很大的过程 [16]：模型需要一个包含已标注代码模块的大型训练数据集，并将这些模块标记为易受攻击或非易受攻击。然而，获取可靠的漏洞标签来源并非易事；没有预言机能够毫无差错地证明代码库中存在或不存在漏洞 [29]。因此，研究人员采用了多种标签来源，以弥补各自的不足。根据先前的分析 [10], [24], ，我们概述了四个主要标签类别：

- **安全厂商提供。** 安全供应商维护漏洞数据库，汇总来自各种安全公告的信息。这提供了一个标准化的已披露漏洞集合。示例包括国家漏洞数据库（NVD） [30],或 Snyk 漏洞数据库 [31]。漏洞记录通常会提供补丁链接，据此可以追溯并识别现实世界中的源代码和漏洞。

- **开发者提供。** 漏洞数据库可能无法完整记录项目 [32] 的所有漏洞。因此，研究人员可以通过开发历史或项目的问题跟踪系统，直接从开发者处收集漏洞修复提交。然而，这种方法需要额外付出努力，从开发工件中搜索安全相关缺陷。

- **工具生成。** 开发者或安全供应商提供的标签有一个主要局限：它们只能收集已报告的漏洞，这严重限制了可收集的样本数量。实际上，漏洞可能长期潜伏或未被发现 [33], ，这会限制数据集规模，并给各个模块带来大量标签噪声。为了解决这一问题，一些研究人员使用静态安全分析工具自动为源代码生成标签 [16], [34], [35]。这一过程严重依赖所用静态分析器的准确性，而这也是争议所在 [15]。

- **人工合成。** 最后，为了绕过其他标签来源的局限，可以根据已知的易受攻击模式人工创建易受攻击代码示例和注释。人工合成样本可确保标签正确性，但代价是源代码多样性降低 [24]。

尽管存在这些注意事项，无论选择何种数据集，研究人员都面临着数据准备方面的挑战 [10]。虽然最先进的软件漏洞预测模型在基准数据集上表现良好，但性能只能衡量模型拟合特定数据集的能力。性能良好并不保证模型能够泛化到现实场景 [36]。因此，数据质量问题会损害研究结果的可靠性和可信度。例如，先前的研究 [13], [33] 指出，由于对非易受攻击模块采用了不准确的标注机制，性能被夸大了。因此，软件漏洞预测模型的行业价值及其应用前景尚不确定 [37], [38]。本研究旨在揭示软件漏洞数据质量的现状，以便更好地了解使用这些数据集所得结果的可靠性和可信度。

B. 软件工程研究中的数据质量

训练数据是机器学习系统不可或缺的组成部分，对生成的模型有着重大影响。与传统软件系统不同，机器学习系统同时具有系统需求和数据需求 [18]。因此，数据质量正成为基于人工智能的软件工程研究的重要组成部分 [39]。软件工程数据和制品往往带有噪声，因为它们通常是在事后通过软件仓库挖掘收集的 [40]。数据和标签并非专门为研究目的而生成，

因此，研究发现，软件工程数据在数据准确性、相关性和数据来源方面存在问题 [40]。

目前，调查软件工程数据集数据质量特征的现有研究缺乏系统性且数量有限。数据质量可由多个维度界定，例如表 I 中列出的维度。现有研究通常仅分析语义或语法数据准确性以及噪声 [22], [41]–[45]。同样，Jimenez 等人 [33] 考察了软件漏洞数据集中的标签准确性。然而，这种方法无法提供完整的图景。为了做出明智的数据决策，有必要对软件工程领域的数据质量进行系统化、客观的调查。Croft 等人 [10] 对 SVP 研究人员所关注的数据质量问题开展了系统性文献综述。尽管这项工作提供了系统化的视角，但其观察结果并未结合实际软件漏洞数据集和软件漏洞预测模型加以验证。我们的工作旨在对软件漏洞数据集中的数据质量进行定量分析，并明确展示其对软件漏洞预测模型的影响。

此外，研究人员最近开始构建自动化清洗框架，以减少已发现的数据噪声问题并确保正确性 [22], [45]。这些框架基于对相关数据集中数据质量问题的深入理解。我们期望本研究所得的发现能够推动软件漏洞数据集清洗框架的创建。

III. 研究设计

为了了解最新技术水平的软件漏洞数据集的数据质量，我们通过分析若干数据质量属性来探讨两个研究问题（RQ）。图 1 展示了本研究采用的整体工作流程。我们将在第 III-B、III-C 和 III-D 节中分别介绍这三个流程。

A. 研究问题

我们的研究围绕以下研究问题展开：

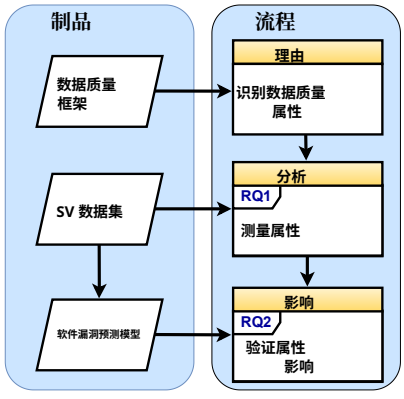


图 1. 整体研究设计。

- **研究问题 1：当前最先进的软件漏洞数据集中存在哪些数据质量问题？**我们的首要目标是让从业者了解软件漏洞数据集的数据质量现状及其性质。
- **研究问题 2：数据质量问题在多大程度上会影响下游的软件漏洞预测模型？**我们的第二个目标是验证数据质量的重要性。我们试图说明，观察到的数据质量问题可能会对软件漏洞预测的下游任务产生负面影响。

B. 确定数据质量属性

现有研究未考虑软件漏洞数据集的数据质量，这或许是因为缺乏对数据质量的系统定义，因而也缺乏相应的测量方法。由于数据质量涉及多个维度[46]，因此不易对其进行定义。并非所有维度都与特定情境相关[47]，，不同组织对质量属性的重视程度也各不相同 [18]。例如，并非所有组织都会优先考虑数据机密性。数据质量属性主要分为两类 [25]：**固有数据质量**，即数据本身内在具有的质量；或**系统依赖型数据质量**，即由外部因素和要求外在产生的质量。本研究聚焦于纯粹的固有数据质量属性，因为软件漏洞预测模型尚未在工业领域得到广泛应用 [37]。若没有相应的部署环境，就无法恰当地测量系统依赖型属性。从这个意义上说，我们关注的是数据本身，而非数据的使用方式。因此，我们的研究结果不受特定建模技术或特征的限制。

为识别固有数据质量属性，我们采用了标准化的数据质量框架 ISO/IEC 25012 [25]。该框架已用于软件工程 [47] 和机器学习[18], [48] 领域的数据质量评估。ISO/IEC 25012 概述了五项固有数据质量属性：准确性、一致性、完整性、时效性和可信度。由于在现有软件漏洞数据集中难以量化，我们未纳入可信度属性。可信度表示我们对数据集的信任程度，即对数据源或供应方真实性的信任。我们需要确保数据点不受污染，也不含虚假信息 [36]。鉴于这些数据集由同行评审研究或受尊重的政府组织[10], 制作，我们假定每个数据集的数据源和供应方具有一定可信度。此外，鉴于唯一性这一数据维度在现有软件工程研究 [49],中普遍存在，且先前的软件漏洞预测研究人员 [24]强调了其重要性，我们也将其纳入考量。表 I 总结了本研究选取用于分析的固有数据质量属性。

C. 属性测量

在分析中，我们为第 II-A 节所述的每种标签来源各收集了一个数据集。为确保收集的数据集能够恰当地代表最新技术水平，我们考虑了由高质量会议或

表 II 选定的最先进的数据集以供考察。

数据集	标签来源	# 函数	% 漏洞
Big-Vul [14]	安全厂商提供	188,636	5.78
Devign [15]	开发者提供	27,318	45.61
D2A [16]	工具生成	1,295,623	1.44
Juliet [50]	人工合成	253,002	36.77

期刊上发表的论文创建或使用的数据集；这些会议或期刊的质量由 CORE[1] 排名中的 A 或 A* 级别体现。新的数据集不断问世，以弥补以往数据集的不足。我们选取了截至 2022年3月最新发布的数据集。此外，我们还选择了包含适当元数据、说明标签获取方式的数据集。表 II 展示了我们选定的数据集。所有四个受检数据集都提供 C/C++函数的源代码。

- **Big-Vul** [14] 从 CVE Details 数据库中关联的补丁抓取易受攻击修复前的软件版本。补丁中行有变更的函数被标记为易受攻击。提交所涉及文件中的所有其余函数均被标记为非易受攻击。
- **Devign** [15]采用了类似的数据收集方法，直接从 GitHub 提交中抓取漏洞修复。研究人员使用关键词方法区分与漏洞相关和无关的提交。随后，他们手动筛选与漏洞相关的提交，以确保准确性。每个提交中所有相关函数都被收集，并归入相应类别。
- **D2A** [16]通过对六个开源代码库中漏洞修复提交前后的项目版本运行静态分析工具来收集源代码。修复后版本中消失的工具警告被归入易受攻击类别。非易受攻击类别则由其余工具警告构成。每条数据记录都表示一个包含原始漏洞位置的函数。我们提取了所有此类函数，以构建 D2A 数据集。
- **Juliet** [50]包含程序的合成示例，这些程序展示了各种已知的易受攻击代码模式。程序根据预定义的增强规则自动生成。每个程序都包含易受攻击版本和非易受攻击版本。Juliet 最初旨在测试静态和动态安全工具，但也被用于训练软件漏洞预测模型。源代码数据以文件形式提供，其中的函数被标注为易受攻击或非易受攻击。我们从数据中每个经过标注的部分提取了所有函数。

我们遵循 Nakajima 和 Nakatani [48] 提出的测量实践，使用 ISO/IEC 25012 框架评估人工智能训练数据需求。表 I 描述了各项属性的解释。我们计算数据集中满足相关特征的样本所占比例，以得出总体测量结果。因此，每项属性的测量值介于 0 和 1 之间，其中 1 表示不存在数据质量问题。我们在公式 1 中正式定义了这一测量方式，其中 N表示数据集中的样本数量，$dq(i)$ 若数据条目 i 满足相关特征，则返回 1。第四节将进一步介绍各项属性的具体测量细节。

$$Attribute = \sum_{i=1}^{N} \frac{dq(i)}{N} \quad (1)$$

D. 验证属性影响

最后，为了验证观察到的数据质量问题所造成的影响，我们研究了这些问题对最新技术水平的软件漏洞预测模型性能的影响。根据观察到的数据质量问题，我们要么在缓解这些问题后，测量重新训练的模型的性能变化；要么调整测试设置，以突出所关注的数据质量特征。更多详情请见第四节。

对于基准测试性能，我们在每个数据集上训练了一个模型，且未对数据进行任何预处理。然而，由于否则无法为 D2A 基准测试构建有效的分类器，我们移除了其中不一致的条目。除非另有说明，我们使用随机的 80:10:10 训练/验证/测试划分运行了所有实验，每次重复五遍，因为这是以往研究中的标准测试设置 [6]–[8]。

我们选择了 *LineVul* 软件漏洞预测模型 [7],，因为这是近期发表的模型，且已被证明在函数级和行级预测方面均优于所有先前的基线模型。LineVul [7] 依赖 CodeBERT [51]来获取能够捕捉词汇和逻辑语义的代码特征表示。CodeBERT 是一种基于 RoBERTa 架构 [52]、经过预训练且达到最新技术水平的代码嵌入模型。类似研究已证明 CodeBERT 在软件漏洞预测方面的有效性 [8], [13]。LineVul 使用基于 Transformer 的架构生成函数级预测。尽管*LineVul* 也能够在完成函数级预测后将预测定位到行级，但所选数据集提供的标签均为函数级。因此，我们以函数级粒度进行预测。

我们使用召回率、精确率和马修斯相关系数（MCC）评估模型性能。我们选择使用 MCC 作为整体性能指标，因为已有建议指出，在类似任务中应使用 MCC [53]。MCC 的取值范围为 -1 到 1，其中 1 为最佳值。

IV. 数据质量分析

表 III 展示了每个数据集的属性值。

A. 准确性

理由。 准确性定义了构成数据集的数据点的正确性。这在很大程度上与语义标签正确性有关；也就是说，被标记为易受攻击或非易受攻击的数据点是否确实与其标签相符。此前已有研究指出，在真实世界数据集中，非易受攻击标签并不可靠，因为这一类别没有真实标签来源 [10], [13], [33]。没有预言机能够可靠地确保一段

[1]http://portal.core.edu.au/conf-ranks/, http://portal.core.edu.au/jnl-ranks/

给定代码片段的安全性并确认其中不存在漏洞利用。因此，非易受攻击标签通常只是通过缺少易受攻击标签来收集的。因此，我们的分析仅限于易受攻击标签来源。我们重点调查了标记为易受攻击的数据点的标签正确性。

分析。我们根据每个数据集的标注机制，通过人工分析判断标签是否正确：漏洞是否准确反映了其来源所依据的漏洞报告或静态分析工具警告。从这个意义上说，我们并未验证漏洞是否确实可被利用，而是判断代码片段在功能上是否与各标签所报告的漏洞相关。我们采取以下步骤评估每条记录的标签正确性：

1) 首先，我们提取了与各数据集中的漏洞及修复提交相关的信息。除 Juliet 外，所有数据集都提供了 Git 修复提交 ID。Big-Vul 还提供了 CVE 编号，D2A 则包含静态分析工具跟踪信息。2) 我们阅读修复提交说明及其他可用信息（即 Big-Vul 中来自 NVD 的漏洞描述，以及 D2A 中的静态分析工具跟踪信息），以了解漏洞以及修复提交中的变更。3) 随后，我们检查了修复提交中相关函数的变更行，以及整个函数的代码，以了解变更行的上下文。根据对代码的理解，我们评估这些变更行在功能上是否与上一步获取的信息相关。4) 如果我们认为它们在功能上不相关，就会检查修复提交中的所有变更，找出根本性变更所在之处，以了解为何被标记的函数不相关。5) 之后，作者讨论了意见不一致的标签，并达成共识。

为便于进行人工审查，我们从每个数据集中随机抽取了 70 个样本进行检查（置信度为 90% +/- 10% [54]）。本文的两位作者分别独立开展了这项人工分析；他们每人都拥有两至五年的软件安全相关经验，经验来自学术界和工业界。两位评审者的科恩卡帕系数为 0.627 [55], ，这表明他们的一致性中等至较强。

我们的研究结果表明，真实世界数据集中存在标签不准确性。我们得到的准确性数值分别为 0.8（Devign）、0.543（Big-Vul）和 0.286（D2A）。我们没有发现合成 Juliet 数据集中存在不准确性，因为其中的漏洞案例是专门按照标签构造的，而非事后收集的。真实世界标注通过追溯漏洞标识符（通常是漏洞修复或警告）来定位原始代码片段。两位进行人工标注的作者记录了他们判断某个标签正确或错误的理由。我们对标签判断理由进行了主题分析 [56] ，以确定数据集不准确性的成因。表 IV 展示了各主题的占比。

• **无关代码更改**。真实世界数据集大多假设，漏洞修复所涉及的代码就是易受攻击代码。然而，漏洞修复提交不一定只包含补丁。风格更改、重构和代码迁移等非功能性更改会干扰数据标注过程。例如，这个修复示例[2]只是将一个常量值转换为等效的宏。同样，纠缠在一起的提交可能会并行实施其他无关更改 [57],，这些更改会被误解为易受攻击代码。

•**清理更改**。由于代码复杂，漏洞修复有时规模较大且涉及多个方面。提交中可能会进行辅助更改，以便更好地完成漏洞修复，例如添加、删除或修改变量、函数或参数。例如，在这个修复提交[3] 示例中，当 read_only 被设为 True 而非受保护的内存对象时，就会出现漏洞。清理更改会将值为 False 的 read_only 转换为空指针，以避免混淆。这些是与漏洞修复相关的功能性变更，因此我们不将其视为无关更改。尽管如此，它们并未指出底层可利用代码的位置，因此会产生假阳性标签。我们称之为清理更改，不过 Sejfia 等人也将其称为伴随更改 [58]。

•**漏洞修复识别不准确**。 如果标注机制未能识别出漏洞修复，那么后续的代码片段自然也不会是漏洞。像 Big-Vul 这样从外部漏洞报告中追溯漏洞修复的数据集，可能会在这一过程中引入错误。例如，我们发现，针对 *Chromium* 项目的大多数漏洞报告都未能正确追溯，因为该仓库并非托管在 GitHub 上。此外，尝试直接从提交历史中识别漏洞修复的数据集（Devign

2https://github.com/FFmpeg/FFmpeg/提交/8b2fce0d3f5a56c40c28899c9237210ca8f9cf75
3https://github.com/chromium/chromium/提交/673ce95d481ea9368c4d4d43ac756ba1d6d9e608

表 III 各数据集每个属性的测量值。

属性 \ 数据集	Big-Vul	Devign	D2A	Juliet
准确性*	0.543	0.800	0.286	1.000
唯一性	0.830	0.899	0.021	0.163
一致性	0.999	0.991	0.531	0.750
完整性	0.824	0.944	0.981	1.000
时效性	0.761	0.811	0.844	-

* 基于数据样本。

表 IV 真实世界数据集中的标签不准确性类型。

数据集	无关	清理	不准确
Big-Vul	25%	28.1%	46.9%
Devign	42.9%	21.4%	35.7%
D2A	0	0	100%

和 D2A）也可能充斥着错误。研究人员通常会尝试通过不准确且不可靠的关键词匹配方法来识别这些提交。最后，D2A 还借助静态分析工具来识别漏洞修复。这些工具会产生大量误报的漏洞警告。

总体而言，我们观察到，混杂提交会给当前的真实世界数据标注启发式方法带来问题 [57]。现有数据集假设漏洞修复中涉及的所有代码都是易受攻击代码，但实际提交往往杂乱无章 [59]。同样，含糊、笼统或不清晰的提交信息也会使漏洞修复识别变得困难 [12]。相比之下，正确的漏洞标签通常源自简单、聚焦且定义明确的漏洞修复提交。

此外，数据集中还包含一些样本，我们难以核实其标签或对标签达成一致。当漏洞位置处于灰色地带时，这种情况经常发生。例如，易受攻击代码片段的调用方是否也应被标注为易受攻击？Herbold 等人 [60]在调查混杂提交时也遇到了类似问题。另一种情况是，标签来源中的漏洞报告可能没有足够的信息，无法准确追溯相关内容。我们暂且将这些有歧义的案例标记为正确。然而，软件安全领域应致力于制定明确的定义，避免此类歧义案例，从而有助于确保标签正确性。

Devign 出现的问题相对较少，因为它是唯一一个创建者尝试对修复提交进行人工验证的数据集。然而，这种准确性保证是以数据集规模为代价的。由于人工验证工作艰巨，Devign 是这些数据集中规模最小的。尽管如此，Devign 仍存在一些不准确之处。大多数错误源于无关变更，例如重构或代码迁移，这可能意味着原作者没有检查此类情况。

Big-Vul 的准确性较低，因为该数据集在数据提取过程中使用的许多漏洞修复提交规模庞大、彼此纠缠或包含大量噪声。大多数错误都源于追踪修复提交时出现的不准确，尤其是在 Chromium 项目中。Big-Vul 中 36% 的易受攻击条目来自 Chromium 项目。

超过三分之二的 D2A 标签不准确。我们发现，这主要是因为静态分析工具发出的警告不可靠，而且漏洞提交标识符也不准确。D2A 数据提取器标记的大多数提交实际上并非漏洞修复，因为其中与安全性相关的词语往往被误解。例如，并非所有包含“memory”（内存）一词的提交都是在修复不安全的内存操作。大多数静态分析工具警告也是假阳性。静态分析工具通常会根据对警告的置信程度，给出警告可靠性的指示。例如，置信度高的整数溢出警告会明确指出整数数据类型和变量值，而不可靠的报告可能两者都未能指出。D2A 中包含的静态分析警告有超过 97% 属于可靠性最低的警告类别，因此往往不准确。不过，由于静态分析工具会尝试直接推断漏洞的位置，因此没有由无关变更或代码清理变更导致的假阳性。

影响。为了评估标签不准确性的影响，我们使用各数据集中的人工验证样本重新训练了每个模型，并将这些样本作为单独的留出测试集。我们比较了使用原始标签和人工修正标签时的模型性能。由于测试集中没有原先标记为非易受攻击的样本，因此无法计算马修斯相关系数。Devign、Big-Vul 和 D2A 的精确率分别下降了 29%、50% 和 80%；我们通过曼-惠特尼U检验 [61] ($p < 0.05$) 确认这些变化具有统计显著性。这是因为错误的易受攻击标签导致模型推断出该类别的错误模式。模型学到的易受攻击模式实际上属于非易受攻击模式。因此，在模型评估中，先前被视为真正例的样本变成了假阳性。相应地，我们发现模型召回率没有受到显著影响（通过曼-惠特尼U检验[61]检验），因为我们只发现了易受攻击类别的标签不准确性；假负例的数量没有变化。不过，这些影响仍然不容忽视，因为它们可能导致模型的假阳性率偏高，从而在实际使用中大幅增加检查工作量。

准确性对于某些现实世界数据集而言较为有限，因为这些数据集依赖于带有噪声且难以识别的漏洞修复提交。准确性问题会导致软件漏洞预测模型推断出类别之间的错误模式。

B. 唯一性

理由。唯一性不一定是固有的数据属性，因为现实世界的数据分布可能包含重复样本。然而，研究已证明，代码重复会对训练后的模型产生不利影响 [49]。重复样本可能导致模型偏向某些样本。当训练集与测试集之间存在重复时，性能值可能会被夸大 [24]。因此，确保数据集中的样本具有唯一性，有助于模型泛化到真实的数据分布 [62]。因此，我们将其视为一项固有属性，并决定研究缺乏唯一性会对软件漏洞预测产生何种影响。

相似或相同的代码片段称为代码克隆 [63]，共有四种主要类型 [64]:

1) **类型1：** 除了空白字符、布局和注释存在差异外，代码片段完全相同。2) **类型2：** 除了类型1克隆差异外，标识符名称和字面值也存在差异，但代码片段完全相同。3) **类型3：** 语法相似的代码片段，在语句级别存在差异。与彼此相比，这些片段增加、修改和/或删除了语句，此外还存在类型1和类型2克隆差异。4) **类型4：** 语法不同的代码片段，实现了相同的功能。

我们遵循标准做法，将类型*3* 代码克隆视为重复项 [49]。即使功能相似的代码片段，也会包含重复的模式和词元，从而对模型性能和评估产生不利影响。然而，对于软件漏洞数据集，轻微的功能性变更就可能造成易受攻击标签与非易受攻击标签之间的差异。典型的漏洞修复只会更改几行代码 [65]。模型必须能够捕捉不同预测类别之间这些细微的功能差异，以避免过高的假阳性率或假阴性率[24]。因此，我们只将标签相同（易受攻击或非易受攻击）的重复项视为代码克隆。

分析。 如果某条目与同一标签下的其他条目存在代码克隆关系，则该条目不具备唯一性。为了识别代码克隆，我们重新使用了 Allamanis[49]开发的代码重复检测器。由于函数比该工具最初设计用于处理的文件更小，我们将样本的最小词元数下调至五。该工具会输出重复簇，因为每个函数可能存在多个重复项。

我们发现所有数据集中都存在代码重复，但在 Big-Vul 和 Devign 数据集中出现的频率较低。我们得到的唯一性值分别为 0.830（Big-Vul）、0.899（Devign）、0.021（D2A）和 0.163（Juliet）。我们针对每个数据集随机抽取 30 个重复簇进行人工检查（Big-Vul 中有 74 个函数，Devign 中有 79 个函数，Juliet 中有 210 个函数，D2A 中有 2288 个函数），以了解重复代码条目出现的原因。通过主题分析[56]，我们发现真实世界数据集中代码重复主要有三种原因：

- **更新后的代码**。 所有真实世界数据集都会从同一代码仓库的多个版本中收集代码，以尽可能多地观察到漏洞。不同版本之间对代码进行的细微功能性或非功能性更新，主要会产生重复的代码片段。对于存在漏洞的情况，这类重复项可能意味着代码更新未能修复漏洞，或引入了新的漏洞。

- **相似函数集**。 一个代码文件可能包含一组简单的模块化函数。这些函数在变量名、逻辑和布局方面通常完全相同，但在功能上略有差异。例如，两个函数可能分别用于启动和停止进程；一组函数也可能各自对数据流执行不同的数学运算。

- **重命名的函数**。相同的函数可能会被复制并重命名，以便在不同文件和上下文中使用。

我们在图 2 中展示了这些成因。这些因素是源代码数据集所固有的，这是由于软件仓库中的代码在空间和时间上都具有重复性。

我们发现，重复现象在 D2A 数据集中特别显著。数据集中每个唯一函数平均有 57 个重复项。这是因为 D2A 在行级生成标签信息，然后将其抽象到函数作用域。如果独特的行被标记，同一个函数就可能被收录多次。因此，D2A 的标注过程引入了许多额外的完全重复项。超过 94% 的

表 V 唯一性问题对性能的影响。

数据集	有重复			无重复			变化 (马修斯相关系数)
	精确率	召回率	MCC	精确率	召回率	MCC	
Big-Vul	0.920	0.765	0.830	0.922	0.762	0.829	0.0% ↓
Devign	0.680	0.428	0.284	0.651	0.399	0.244	13.9% ↓
D2A	0.961	0.630	0.774	0.741	0.049	0.141	81.7% ↓
Juliet	0.939	0.945	0.909	0.962	0.799	0.814	10.4% ↓

D2A 数据集属于 1 型代码克隆。此外，构成 D2A 的六个代码仓库中有两个互为分支（*FFmpeg* 和*Libav*），这进一步造成了重复。D2A 缺乏唯一性，让人质疑该数据集规模的说法；该数据集在函数级提供的信息有限。

我们还发现 Juliet 中存在大量重复项，这是由于测试用例之间的细微差异所致。通过对控制流逻辑、内部函数调用或字面值进行细微更改，可以生成新的测试用例。此外，由于已修复语句采用固定的修正实现，非易受攻击的已修复语句可能会出现完全重复。

影响。 对于 SVP，重复项可能出现在训练集、测试集内，也可能跨越这两个集合。图 3 展示了这些重复类型。训练集内的重复项可能导致模型偏差 [62]，但很难通过模型性能 [49]来衡量这些方面。我们重点分析唯一性对模型评估的影响。我们按照 III-D 节中的说明，将每个数据集划分为训练集、验证集和测试集。随后，我们比较了移除或保留与测试集跨集合重复项时模型的评估性能。Allamanis [49] 发现，跨集合重复是软件工程研究中最显著的重复类型。

表 V 展示了移除已识别的重复条目后，软件漏洞预测模型的性能变化。我们观察到，跨集合重复是某些数据集的重要影响因素：Devign、D2A 和 Juliet 的总体评估结果（马修斯相关系数）有所下降，并且我们使用曼-惠特尼U检验 [61] ($p < 0.05$) 确认了这一变化具有显著性。使用 Big-Vul 数据训练的模型则未受到显著影响。这表明，缺乏唯一性并不总是会造成问题。

重复项可能导致评估设置中出现数据泄漏[49]；模型可以轻而易举地将测试集中同时也在训练集里重复出现的样本分类，从而夸大真实表现。我们观察到，对于所有数据集，重复对召回率的负面影响都大于对精确率的影响。

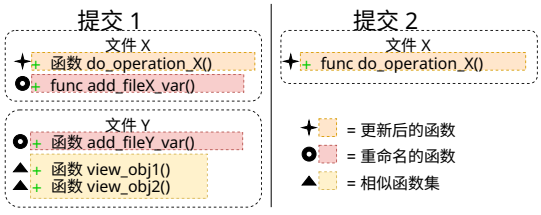


图 2. 三种主要代码重复原因的示例。

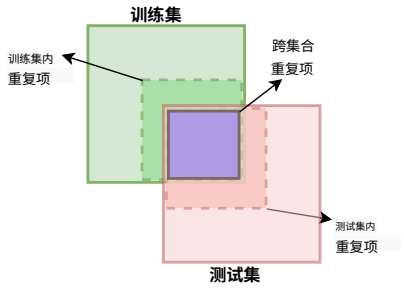


图 3. 机器学习模型中的重复项类型，改编自 Allamanis [49]。

移除跨集合重复项后，测试集中的简单样本也随之移除，主要减少了真正例。由于假负例的比例高于假阳性，这对召回率的影响更大。Devign（下降 7%）、D2A（下降 92%）和 Juliet（下降 15%）的召回率显著降低（通过曼-惠特尼U检验 [61]确认）；而移除重复项后，Big-Vul 和 Juliet 的精确率甚至有所上升。不过，除 Big-Vul 外，其他各数据集的整体表现（马修斯相关系数）仍然有所下降。

唯一性 问题存在于所有数据集中，这是由代码重复且不断增量演进的特性所致。重复代码片段可能会因数据泄漏而抬高整体评估性能。

C. 一致性

理由。 一致性意味着数据条目不应提供相互冲突的信息。对于软件漏洞数据集，这意味着相似的代码片段不应具有冲突标签。一段代码不可能既易受攻击又非易受攻击。然而，由于用于构建数据集的多个数据流，软件漏洞数据中可能会出现不一致 [66]。一致性对于模型训练和构建显然很重要，因为冲突标签会使任何试图区分两个类别的基于人工智能的模型感到困惑。

一致性与唯一性属性相关，因为我们再次检查了重复数据。不过，一致性衡量的是具有冲突标签的重复条目。由于细微的功能变更就可能形成易受攻击与非易受攻击代码片段之间的功能差异，因此我们只考虑了类型1代码克隆（完全匹配）。

分析。 如果某个条目没有任何带有冲突标签的重复项，则该条目是一致的。我们观察到 Big-Vul（0.999）和 Devign（0.991）的一致性值较高，而 D2A（0.531）和 Juliet（0.75）的值较低。我们手动检查了随机抽取的 30 个不一致簇，以确定漏洞标签不一致的原因。我们发现，标签不一致的原因因数据收集方法而异，且各有特点，以下将对此进行讨论。

对于 Big-Vul，不一致的标签是由源代码中存在的潜在漏洞产生的。该数据集的标注启发式方法假设，提交中未被明确修改的文件里的所有函数均为非易受攻击。然而，这些函数实际上可能包含开发者尚未知晓的漏洞。这些漏洞可能会在之后被报告并收集起来。图4展示了这一过程。尽管不一致案例的数量相对较少，但这些只是我们已知的潜在漏洞。实际上，我们无法全面掌握所有潜在漏洞。Croft 等人 [13]观察到，他们的数据集中潜在漏洞的数量至少是已知漏洞的两倍。

在 Devign 数据集中，不一致是由同时存在的代码分支造成的。漏洞修复提交可能只在一个分支中被识别，导致其他分支中的相同提交被视为非易受攻击。这主要是由于分支上的合并提交所致，因为合并提交可能包含漏洞修复，却没有对此进行说明。与 Big-Vul 的不一致标签类似，这意味着非易受攻击类别中存在错误标签，因为其他分支中的提交被错误地识别了。

用于推断 D2A 数据集标签的静态分析工具会产生过多警告。在 D2A 数据提取过程中，所有函数都会在每个经过分析的提交中接受扫描。因此，同一个函数可能会在不同的提交中收到静态分析工具发出的相同警告。如果某个提交修改了标记行，而其他提交没有修改，就会导致标签不一致。我们发现，这种情况在实践中很常见，这一点也体现在该数据集相对较低的一致性值上。

Juliet测试用例可能包含执行不安全操作的三级函数，例如向缓冲区写入数据。设置此类测试用例是为了帮助测试漏洞扫描工具跨函数追踪数据流的能力。尽管这些三级函数由于缺少必要的安全检查而易受攻击，但只有在向其传入特定值时才会发生漏洞利用。因此，该数据集的易受攻击和非易受攻击标注部分中都包含这些函数的重复副本。

从分析中我们发现，不一致样本主要反映了非易受攻击类别的数据收集流程存在不准确之处。这是因为该类别缺乏合适的标签来源或核查机制；该类别由.....构成由于缺少漏洞标签。

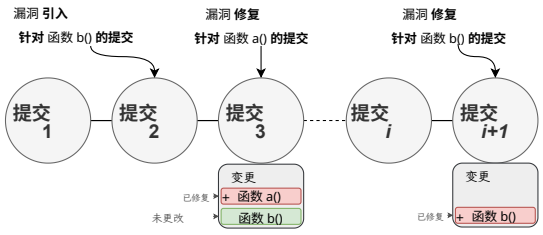


图 4. 潜在漏洞引发不一致的示例。函数 b 从提交 3 到提交 i+1 一直易受攻击，但仅在后者中被记录为易受攻击。

表 VI 一致性问题对性能的影响，与原始数据集设置的比较。

数据集	全部不一致（原始）			一致的测试集			一致的训练集和测试集		
	精确率	召回率	MCC	精确率	召回率	MCC	精确率	召回率	MCC
Big-Vul	0.902	0.774	0.826	0.919（↑）	0.774（-）	0.835（↑）	0.915（↑）	0.775（↑）	0.833（↑）
Devign	0.625	0.569	0.285	0.668 (↑)	0.500 (↓)	0.311 (↑)	0.653 (↑)	0.502 (↓)	0.289 (↑)
D2A	0	0	0	0 (-)	0 (-)	0 (-)	0.948 (↑)	0.599 (↑)	0.748 (↑)
Juliet	0.937	0.950	0.910	0.998 (↑)	0.985 (↑)	0.987 (↑)	0.999 (↑)	0.999 (↑)	0.999 (↑)

影响。 与唯一性一样，不一致性可能出现在训练集、测试集内部，或这两个集合之间，如图 3 所示。训练集中的不一致性会影响模型学到的模式，而测试集中的不一致性会影响模型评估。我们在影响分析实验中考虑了这两个方面。根据人工分析的结果，不一致簇中的非易受攻击类别条目有误，因此我们通过移除这些条目来消除不一致性。采用 III-D 节所述的实验设置，我们考虑了三种情形：保留所有不一致样例的原始情形；移除所有测试集内及跨集合不一致项、但训练集仍不一致的一致的测试集；以及移除所有不一致条目的完全一致数据集。我们针对每种设置分别训练并评估了一个模型。

表 VI 展示了对性能的影响。我们观察到，不一致性可能会影响模型评估：使用一致的测试集时，马修斯相关系数（MCC）性能有所提升。这是因为模型自然会对相同的输入作出相同的预测，从而导致一部分不一致条目被错误预测。因此，不一致样本会损害性能，因为缺少可区分的标签会阻止模型推断重要模式，或导致模型偏向错误的类别标签。在 D2A 中，不一致性十分普遍，除非使用一致的训练集进行训练，否则模型无法作出任何正确预测。我们观察到，对于不一致的簇，模型会默认预测其中最常见的标签；在 D2A 中，这一标签是非易受攻击类别。通过 Mann-Whitney U 检验 [61] ($p < 0.05$)，我们确认，消除不一致问题显著提升了受影响最严重的数据集（D2A和Juliet）的性能。

我们观察到，与召回率相比，一致性提高对精确率的正向影响更大。对于 Devign，使用一致的数据集时，召回率甚至有所下降（尽管整体性能仍有所提升）。这很可能是因为不一致簇更容易产生假阳性，因为每个数据集中非易受攻击样本的数量更多。

由于受影响条目相对较少，Big-Vul 和 Devign 的性能影响也相对较小。我们无法确认使用这些数据集时性能变化在统计学上是否显著。不过，我们认为，这些不一致实际上反映了这些数据集的非易受攻击类别中存在更大的问题。潜在漏洞或被错误分类的修复提交的数量很可能要多得多，但我们通过不一致的标签只能观察到较少的

表VII 数据集中各类缺失值的出现频率。

数据集	截断			空	声明	总计
	起始	End	Both			
Big-Vul	32,973	133	140	0	0	33,246
Devign	814	265	9	0	0	1,088
D2A	0	0	0	10,824	13,300	24,124

数量。Jimenez 等人 [33] 和 Croft 等人 [13] 均发现，被错误标记的潜在漏洞会显著影响下游软件漏洞预测模型。必须改进数据收集流程，以确保一致性。

一致性 问题源于缺少标签指示符，或缺少对无漏洞代码的检查。尽管测得的数值较小，但它们可能预示着更严重的问题。一致性可能是一个重大问题，会阻碍模型学习必要的模式。

D. 完整性

理由。 完整性既可以指数据集内信息的完整程度，也可以指单个数据条目中数值的完整程度。由于前者需要外部参考信息，我们聚焦于后者，因为它是数据固有的属性。对于漏洞数据集，如果源代码的内容未包含原始函数的全部代码，则可能存在信息缺失。

分析。 为检测信息缺失，我们通过分析 C/C++ 函数语法，自动检查代码片段是否不完整。我们发现，部分代码条目缺失或被截断。总体而言，Big-Vul、Devign、D2A 和 Juliet 的完整性值分别为 0.824、0.944、0.981 和 1.0。这些相对较高的数值表明，与其他数据质量属性相比，完整性问题出现的频率较低。信息缺失仅出现在所分析的四个数据集中三个数据集中。表 VII 展示了各数据集中不同截断类型的出现频率。我们未纳入 Juliet，因为其中没有任何条目存在信息缺失。

我们发现，函数开头的截断现象主要出现在 Big-Vul 数据集 中。函数定义的返回类型如果跨越多行，就会被截断，因为函数解析器通常从包含函数名称的那一行开始解析。我们还发现，Big-Vul 和 Devign 数据集中有少数函数被过早截断，缺失了具有功能性的代码行。由于我们无法访问用于生成这些数据集的

脚本，因此无法确定截断的确切原因。我们推测，源代码中的复杂结构会使所用的词法解析器混淆，从而影响其提取内容。例如，许多较早被截断的样本在字面量中包含额外的花括号（}）。

D2A 能够抵御截断，但其中包含空的缺失值，且未提供相应代码。这种情况出现在静态分析工具标记了代码文件中任何函数之外的行时。此外，D2A 包含 13,300 个单行函数声明，这些声明不含任何函数源代码。

影响。 为了了解信息缺失对软件漏洞预测模型的影响，我们为每个数据集留出一个包含完整条目的公共测试集。然后，将每个数据集剩余条目等分为两部分，构成两个训练集：一个包含不完整的数据值，另一个不包含。对于所有数据集，使用完整数据训练集时，马修斯相关系数的表现均略有提升。然而，使用曼-惠特尼U检验 [61]($p > 0.05$)，我们无法确认马修斯相关系数、精确率或召回率的任何变化具有统计显著性。尽管被截断信息的复杂程度可能各不相同，但这些信息似乎只占函数整体内容的一小部分，而且出现频率相对较低。不过，我们仍建议从业者今后确保软件漏洞数据的完整性，因为更严重的问题可能会造成更大的影响。

完整性问题可能在数据收集过程中出现，但这些问题很容易解决，且影响不大，因为它们造成的信息缺失相对较少。

E. 时效性

理由。时效性旨在确保数据集在时间特征上与其应用语境保持一致 [48]。在机器学习领域，这称为概念漂移 [67]：输入数据与目标变量之间的关系随时间发生变化的情形。由于漏洞和源代码具有不断演变的特性，漏洞数据集必须保持更新[68], [69]。我们将漏洞数据集中某条记录的日期定义为该漏洞通过数据集的标注机制报告的日期。时效性不适用于人工合成的 Juliet 数据集。

分析。时效性适用于整个数据集，而非单个数据点，因此我们选择了一种标准的非
用于概念漂移检测的上下文方法 [67]。我们使用詹森-香农散度度量 [70]来表示时效性，因为它衡量数据集中原始数据与当前数据之间的统计距离。该度量的公式见 [70]。为简化起见，我们分别将数据集最早和最新的半部分记为原始数据和当前数据。我们通过词元袋集合来表示漏洞数据的分布。我们使用词法分析器对集合中的所有源代码进行词元化处理，然后根据总频率对数值进行归一化，以获得每个词元出现情况的概率分布。

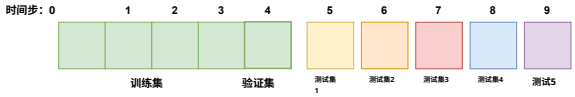


图5. 时效性影响实验设置。

由于詹森-香农散度度量衡量的是差异性，因此我们用 1 减去该值来计算时效性。我们得到的时效性数值分别为 0.761（Big-Vul）、0.811（Devign）和 0.844（D2A）。对于这一属性而言，这些数值相对较高，不太可能表明存在概念漂移。

影响。 我们采用了与 McIntosh等人 [69] 类似的实验设置，以确定漏洞数据是否属于移动目标。我们按日期对所有条目进行排序，然后将每个数据集划分为十个大小相等的分区。最早的四个分区用于训练软件漏洞预测模型，第五个分区用于调优，其余五个分区则分别作为测试集。图5展示了实验设置。然而，使用肯德尔秩相关检验 [71]($p > 0.05$)后，我们发现，随着训练集与测试集之间的时间间隔增大，模型在马修斯相关系数、精确率或召回率方面的性能并未显著下降。

时效性问题在软件漏洞数据集中并未出现。这些数据集展现出良好的数据时间分布，因为它们是在较长的时间范围内收集的。

五、讨论

由于数据准备困难，软件漏洞数据集尤其容易受到数据质量挑战的影响 [10]。现有的软件漏洞预测研究大多关注建模方面的进展，却往往忽视数据质量。因此，我们对固有数据质量属性的系统分析揭示了当前软件漏洞数据集存在的关键数据问题。当前，软件漏洞数据的数据预处理较为粗略或不一致 [10]。目前缺乏指导软件漏洞预测研究中数据清洗工作的相关方法。我们对现有数据集进行质量评估后，总结出以下经验教训：

- 谨慎重复使用现有数据集，务必先检查数据质量。
- 软件漏洞数据的唯一性较差，因此应避免采用会导致训练集与测试集之间出现大量重复的评估设置。
- 应根据不一致的原因，删除标签不一致的数据点。
- 应删除或补充信息缺失或不完整的源代码条目。

唯一性、一致性和完整性方面的问题可以通过基于规则的语法过滤器检测到，正如我们在本研究中所做的那样。因此，理论上，我们可以通过排除不符合质量属性的噪声样本来解决这些问题。然而，实际操作中可能并不那么容易，因为软件漏洞数据非常稀缺。SVP 需要大型数据集 [16]，因此移除噪声样本可能会使数据集规模过小。图6展示了我们能够为每个数据集自动检测出的

干净样本的比例。此外，我们通过人工观察发现，数据准确性问题十分严重，但目前还没有能够自动检测此类问题的方法。数据不准确可能会使干净条目的数量进一步减少20-71%。

因此，我们需要解决这些问题的根本原因。根据我们获得的发现，我们总结了当前主要数据质量问题的成因如下，并提出一些方向供研究人员探索，以帮助克服这些挑战：

- **自动数据收集往往会导致数据不准确**。我们发现，错误标记的漏洞主要源于漏洞修复识别不准确：要么选错了提交，要么选错了代码行更改。目前已有大量研究致力于使用基于机器学习的模型识别正确的漏洞补丁 [12], [72]。但目前仍缺乏用于识别正确漏洞修复代码行的语义过滤器或启发式方法。

- **源代码重复可能导致数据集缺乏多样性**。数据收集面临的一个根本问题是样本多样性和唯一性不足。虽然我们能够收集到的漏洞样本有限，但对于非易受攻击类别，我们可以选择样本。因此，我们建议开发数据收集启发式方法，以获取更多样化的非易受攻击代码样本。同样，我们也需要更好的合成数据生成方法。漏洞播种在这方面已展现出良好的效果 [17]，但这种技术仍依赖于其所推断种子的现实世界漏洞的数据质量。

- **未知漏洞可能导致标签不一致**。 标签不一致问题源于非易受攻击类别存在的潜在问题。Big-Vul 样本中包含未检测出的漏洞，而 Devign 和 D2A 均包含未记录的漏洞。这尤其棘手，因为我们缺乏非易受攻击代码的标签来源。半监督语义过滤器已显示出减少非易受攻击标签噪声的潜力 [13], [73]。将合成数据集用作训练数据集时，需要明确界定其使用指南。

VI. 有效性威胁

构念效度：我们对数据质量的分析可能无法完美地体现目标属性。我们的分析依据相关领域的标准实践以及现有的软件漏洞数据实践知识而形成 [48] [10]。通过领域专家开展需求引导，有助于在未来改进这些主张 [18]。部分属性需要进行人工分析，这也可能构成一项限制，因为人工分析可能带有偏差或存在不准确之处。我们使用了两位独立评估者，以尽量减少此类影响。我们将 CORE 排名作为数据集筛选标准，尽管 CORE 期刊排名现已不再使用。我们认为这些排名仍然适用，因为它们在数据收集日期前仅两个月才停止使用。

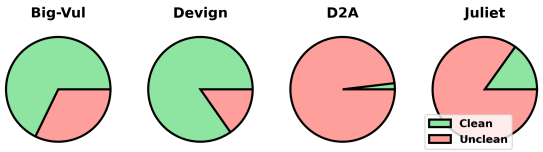


图 6. 可自动检测的数据集中，干净样本与不干净样本的比例。

内部效度： 混杂因素可能会影响我们的影响分析实验结果。我们分别分析了每个数据属性，因此每项实验期间都存在其他数据质量问题。还需要进一步研究累积考察数据质量属性的情况。

外部效度： 我们的分析仅限于四个最先进的数据集。测量范围也仅限于包含适当元数据的数据集。例如，由于这一问题，我们无法研究 ReVeal 数据集 [24] 。同样，我们仅使用一个 SVP 模型进行了影响分析。该模型已被证明达到了最新技术水平 [7]。此外，我们考察的是固有数据质量属性，因此无论使用何种模型，这些问题都依然存在。

VII. 结论

我们系统地考察了四个最先进的软件漏洞数据集的五项数据质量属性，以帮助提高依赖这些信息的下游数据驱动任务的有效性和可信度。我们的研究结果表明，一些软件漏洞数据集容易出现数据质量问题，尤其是在数据准确性、唯一性和一致性方面。我们发现，真实世界数据集中有 20%–71% 的漏洞标签不准确，导致性能变化最高达 65%。此外，有 0%–47% 的标签不一致，在最极端的情况下，这完全阻碍了模型训练。

数据质量需要持续关注和分析。我们建议未来的研究人员和从业者利用我们提供的方法，更有效、细致地考量数据质量。此外，我们强调数据质量的重要性，以及克服我们观察到的质量问题的必要性。最后，我们呼吁进一步研究依赖系统的数据质量属性，以满足特定的运行需求。

VIII. 数据可用性

我们已通过复现包 [26]提供数据和分析脚本。

致谢

本研究得到了网络安全合作研究中心有限公司的支持，该中心的活动部分由澳大利亚政府的合作研究中心计划资助。

参考文献

[1] G. McGraw，《软件安全》，IEEE Security & Privacy，第2卷，第2期，第80–83页，2004年。[2] H. Shahriar和M. Zulkernine，《缓解程序安全漏洞：方法与挑战》，ACM计算调查（CSUR），第44卷，第3期，第1–46页，2012年。[3] G. Lin、S. Wen、Q.-L. Han、J. Zhang和Y. Xiang，《使用深度神经网络进行软件漏洞检测：综述》，Proceedings of the IEEE，第108卷，第10期，第1825–1848页，2020年。[4] Z. Li、D. Zou、S. Xu、X. Ou、H. Jin、S. Wang、Z. Deng和Y. Zhong，《Vuldeepecker：一种基于深度学习的漏洞检测系统》，arXiv预印本arXiv:1801.01681，2018年。[5] Z. Li、D. Zou、S. Xu、H. Jin、Y. Zhu和Z. Chen，《Sysevr：一种使用深度学习检测软件漏洞的框架》，IEEE Transactions on Dependable and Secure Computing，2021年。[6] Y. Li、S. Wang和T. N. Nguyen，《通过细粒度解释进行漏洞检测》，载于第29届ACM欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2021年，第292–303页。[7] M. Fu和C. Tantithamthavorn，《Linevul：基于Transformer的行级漏洞预测》，载于2022年IEEE/ACM第19届国际软件仓库挖掘会议（MSR），2022年，第608–620页。[8] D. Hin、A. Kan、H. Chen和M. A. Babar，《Linevd：使用图神经网络进行语句级漏洞检测》，arXiv预印本arXiv:2203.05181，2022年。[9] R. Croft、D. Newlands、Z. Chen和M. A. Babar，《基于规则的方法与基于学习的方法用于静态应用安全测试的实证研究》，载于第15届ACM/IEEE实证软件工程与度量国际研讨会（ESEM）论文集，2021年，第1–12页。[10] R. Croft、Y. Xie和M. A. Babar，《软件漏洞预测的数据准备：系统性文献综述》，IEEE软件工程汇刊，2022年。[11] T. Zimmermann、N. Nagappan和L. Williams，《大海捞针：预测Windows Vista中的安全漏洞》，载于2010年第三届国际软件测试、验证与确认会议。IEEE，2010年，第421–428页。[12] J. Zhou、M. Pacheco、Z. Wan、X. Xia、D. Lo、Y. Wang和A. E. Hassan，《大海捞针：自动化挖掘未公开的漏洞修复》，载于2021年第36届IEEE/ACM国际自动化软件工程会议（ASE）。IEEE，2021年，第705–716页。[13] R. Croft、M. A. Babar和H. Chen，《用于安全缺陷的噪声标签学习》，arXiv预印本arXiv:2203.04468，2022年。[14] J. Fan、Y. Li、S. Wang和T. N. Nguyen，《Ac/c++ 代码漏洞数据集及其代码变更和CVE摘要》，载于第17届国际软件仓库挖掘会议论文集，2020年，第508–512页。[15] Y. Zhou、S. Liu、J. Siow、X. Du和Y. Liu，《Devign：通过图神经网络学习全面的程序语义实现有效的漏洞识别》，神经信息处理系统进展，第32卷，2019年。[16] Y. Zheng、S. Pujar、B. Lewis、L. Buratti、E. Epstein、B. Yang、J. Laredo、A. Morari和Z. Su，《D2a：一个通过差分分析为基于人工智能的漏洞检测方法构建的数据集》，载于2021年IEEE/ACM第43届国际软件工程会议：软件工程实践（ICSE-SEIP）。IEEE，2021年，第111–120页。[17] Y. Nong、Y. Ou、M. Pradel、F. Chen和H. Cai，《通过神经代码编辑生成逼真的漏洞：一项实证研究》，载于第30届ACM欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2022年，第1097–1109页。[18] A. Vogelsang和M. Borg，《面向机器学习的需求工程：来自数据科学家的观点》，载于2019年IEEE第27届国际需求工程会议研讨会（REW）。IEEE，2019年，第245–251页。[19] H. Sanders和J. Saxe，《垃圾进，垃圾出：所谓出色的机器学习模型如何被劣质数据搞砸》，Blackhat论文集，第2017卷，2017年。[20] D. Arp、E. Quiring、F. Pendlebury、A. Warnecke、F. Pierazzi、C. Wressnegger、L. Cavallaro和K. Rieck，《机器学习在计算机安全中的注意事项》，载于USENIX安全研讨会论文集，2022年。

[21] H. J. Kang、K. L. Aw 和 D. Lo，《检测自动静态分析工具产生的误报：我们还差多远？》，arXiv 预印本 arXiv:2202 05982，2022 年。。[22] L. Shi、F. Mu、X. Chen、S. Wang、J. Wang、Y. Yang、G. Li、X. Xia 和 Q. Wang，《我们是否建立在坚实的基石之上？论数据预处理对代码摘要的重要性》，载于第 30 届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2022 年，第 107–119 页。[23] J. He、L. Beurer-Kellner 和 M. Vechev，《基于学习的缺陷检测器中的分布偏移》，arXiv 预印本 arXiv:2204.10049，2022 年。[24] S. Chakraborty、R. Krishna、Y. Ding 和 B. Ray，《基于深度学习的漏洞检测：我们到达目标了吗？》，IEEE软件工程汇刊，2021 年。[25] ISO/IEC，《系统与软件工程——系统与软件质量要求与评价（SQuaRE）——数据质量模型》，2008 年。[26] R. Croft、M. A. Babar 和 M. Kholoosi，《“软件漏洞数据集的数据质量”的复现包》，2023 年。[在线]。可访问：https://figshare.com/articles/software/Reproduction_Package_for_Data_Quality_for_Software_Vulnerability_Datasets/20499924 [27] S. M. Ghaffarian 和 H. R. Shahriari，《使用机器学习和数据挖掘技术进行软件漏洞分析与发现：综述》，ACM计算调查（CSUR），第 50 卷，36，2017 年 4 月，第 1 期，页码——。[28] H. Hanif、M. H. N. M. Nasir、M. F. Ab Razak、A. Firdaus 和 N. B. Anuar，《软件漏洞的兴起：软件漏洞检测分类体系与机器学习方法》，网络与计算机应用期刊，第 103009 页，2021 年。[29] G. M. Weinberg，《完美软件及其他关于测试的幻象》。Dorset House出版社，2008 年。[30] NIST，《国家漏洞数据库》。 [在线]。可访问：https: //nvd.nist.gov/ [31] Snyk，《Snyk 漏洞数据库》。 [在线]。可访问：https: //security.snyk.io/ [32] A. Anwar、A. Abusnaina、S. Chen、F. Li 和 D. Mohaisen，《清理 NVD：全面的质量评估、改进与分析》，IEEE Transactions on Dependable and Secure Computing，2021 年。[33] M. Jimenez、R. Rwemalika、M. Papadakis、F. Sarro、Y. Le Traon 和 M. Harman，《预测软件漏洞时考虑真实世界标注的重要性》，载于 2019 年第 27 届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2019 年，第 695–705 页。[34] S. Moshtari、A. Sami 和 M. Azimi，《使用复杂度度量指标提升软件安全性》，《计算机欺诈与安全》，第 2013 卷，第 5 期，第 8–17 页，2013 年。[35] R. Russell、L. Kim、L. Hamilton、T. Lazovich、J. Harer、O. Ozdemir、P. Ellingwood 和 M. McConley，《使用深度表征学习自动检测源代码中的漏洞》，载于 2018 年第 17 届 IEEE国际机器学习与应用会议（ICMLA）。IEEE，2018 年，第 757–762 页。[36] J. M. Zhang、M. Harman、L. Ma 和 Y. Liu，《机器学习测试：综述、发展现状与展望》，IEEE软件工程汇刊，2020 年。[37] P. Morrison、K. Herzig、B. Murphy 和 L. Williams，《应用漏洞预测模型所面临的挑战》，载于 2015 年安全科学研讨会与训练营论文集，2015 年，第 1–9 页。[38] A. Sotgiu、M. Pintor 和 B. Biggio，《基于可解释性的漏洞发现机器学习调试》，载于第 17 届国际可用性、可靠性与安全会议论文集，2022 年，第 1–8 页。[39] G. Liebchen 和 M. Shepperd，《软件工程中的数据集与数据质量：八年之后》，载于第 12 届软件工程预测模型与数据分析国际会议论文集，2016 年，第 1–4 页。[40] M. F. Bosu 和 S. G. MacDonell，《实证软件工程中的数据质量挑战分类体系》，载于 2013 年第 22 届澳大利亚软件工程会议。IEEE，2013 年，第 97–106 页。[41] S. Kim、H. Zhang、R. Wu 和 L. Gong，《应对缺陷预测中的噪声》，载于 2011 年第 33 届国际软件工程会议（ICSE）。IEEE，2011 年，第 481–490 页。[42] K. Herzig、S. Just 和 A. Zeller，《这不是缺陷，而是特性：错误分类对缺陷预测的影响》，载于 2013 年第 35 届国际软件工程会议（ICSE）。IEEE，2013 年，第 392–401 页。

[43] M. Shepperd、Q. Song、Z. Sun 和 C. Mair，《数据质量：关于 NASA 软件缺陷数据集的一些评论》，IEEE软件工程汇刊，第 39 卷，第 9 期，第 1208–1215 页，2013 年。[44] X. Wu、W. Zheng、X. Xia 和 D. Lo，《数据质量至关重要：安全漏洞报告预测中数据标签正确性的案例研究》，IEEE软件工程汇刊，2021 年。[45] Z. Sun、L. Li、Y. Liu、X. Du 和 L. Li，《论构建高质量神经代码搜索训练数据集的重要性》，载于第 44 届国际软件工程会议论文集，2022 年，第 1609–1620 页。[46] F. Sidi、P. H. S. Panahy、L. S. Affendey、M. A. Jabar、H. Ibrahim 和 A. Mustapha，《数据质量：数据质量维度综述》，载于 2012 年国际信息检索与知识管理会议。IEEE，2012 年，第 300–304 页。[47] F. Gualo、M. Rodr´ıguez、J. Verdugo、I. Caballero 和 M. Piattini，《基于 ISO/IEC 25012 的数据质量认证：行业经验》，《系统与软件杂志》，第 176 卷，文章编号 110938，2021 年。[48] S. Nakajima 和 T. Nakatani，《SQuaRE 数据质量模型的人工智能扩展》，载于 2021 年 IEEE 第 21 届国际软件质量、可靠性与安全会议伴会（QRS-C）。IEEE，2021 年，第 306–313 页。[49] M. Allamanis，《代码重复对代码机器学习模型的不利影响》，载于 2019 年 ACM SIGPLAN 编程与软件新思想、新范式与反思国际研讨会论文集，2019 年，第 143–153 页。[50] T. Boland 和 P. E. Black，《Juliet 1.1 C/C++ 和 Java 测试套件》，IEEE 计算机体系结构快报，第 45 卷，第 10 期，第 88–90 页，2012 年。[51] Z. Feng、D. Guo、D. Tang、N. Duan、X. Feng、M. Gong、L. Shou、B. Qin、T. Liu、D. Jiang 等，《CodeBERT：用于编程语言和自然语言的预训练模型》，arXiv 预印本 arXiv:2002.08155，2020 年。[52] Y. Liu、M. Ott、N. Goyal、J. Du、M. Joshi、D. Chen、O. Levy、M. Lewis、L. Zettlemoyer 和 V. Stoyanov，《RoBERTa：稳健优化的 BERT 预训练方法》，arXiv 预印本 arXiv:1907.11692，2019 年。[53] J. Yao 和 M. Shepperd，《评估软件缺陷预测性能：为何马修斯相关系数至关重要》，《软件工程评估与评价》会议论文集，第 120–129 页，2020 年。[54] W. G. Cochran，《抽样技术》。John Wiley & Sons，2007 年。[55] J. Cohen，《名义量表的一致性系数》，《教育与心理测量》，第 20 卷，第 1 期，第 37–46 页，1960 年。[56] V. Braun 和 V. Clarke，《在心理学中使用主题分析》，《心理学定性研究》，第 3 卷，第 2 期，第 77–101 页，2006 年。[57] K. Herzig 和 A. Zeller，《纠缠代码变更的影响》，载于 2013 年第 10 届软件仓库挖掘工作会议（MSR）。IEEE，2013 年，第 121–130 页。[58] A. Sejfia、Y. Zhao 和 N. Medvidovi´c，《识别软件补丁中的伴随更改》，载于第 29 届 ACM 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，2021 年，第 304–315 页。[59] K. Herzig、S. Just 和 A. Zeller，《纠缠代码变更对缺陷预测模型的影响》，《实证软件工程》，第 21 卷，第 2 期，第 303–336 页，2016 年。[60] S. Herbold、A. Trautsch、B. Ledel、A. Aghamohammadi、T. A. Ghaleb、K. K. Chahal、T. Bossenmaier、B. Nagaria、P. Makedonski、M. N. Ahmadabadi 等，《关于缺陷修复提交中代码纠缠的细粒度数据集与分析》，《实证软件工程》，第 27 卷，第 6 期，第 1–49 页，2022 年。[61] H. B. Mann 和 D. R. Whitney，《关于检验两个随机变量中一个是否在随机意义上大于另一个》，《数学统计年刊》，第 50–60 页，1947 年。[62] Y. Zhao、L. Li、H. Wang、H. Cai、T. F. Bissyand´e、J. Klein 和 J. Grundy，《样本重复对基于机器学习的 Android 恶意软件检测的影响》，ACM 软件工程与方法学汇刊（TOSEM），第 30 卷，第 3 期，第 1–38 页，2021 年。[63] Q. U. Ain、W. H. Butt、M. W. Anwar、F. Azam 和 B. Maqbool，《代码克隆检测系统综述》，IEEE Access，第 7 卷，第 86 121–86 144 页，2019 年。[64] H. Sajnani、V. Saini、J. Svajlenko、C. K. Roy 和 C. V. Lopes，《SourcererCC：将代码克隆检测扩展至大规模代码》，载于第 38 届国际软件工程会议论文集，2016 年，第 1157–1168 页。[65] V. Piantadosi、S. Scalabrino 和 R. Oliveto，《开源项目中的安全漏洞修复：Apache HTTP Server 的案例研究

和 Apache Tomcat，”载于*2019*年第*12*届*IEEE*软件测试、验证与确认会议（*ICST*）。IEEE，2019年，第68–78页。[66] R. Croft、M. A. Babar 和 L. Li，“不同数据源中软件漏洞严重性不一致性的调查”，载于*2022*年*IEEE*国际软件分析、演化与再工程会议（*SANER*）。IEEE，2022年，第338–348页。[67] J. Gama、I. ˇZliobait˙e、A. Bifet、M. Pechenizkiy 和 A. Bouchachia，“概念漂移适应综述”，*ACM*计算调查（*CSUR*），第46卷，第4期，第1–37页，2014年。[68] T. H. M. Le、B. Sabir 和 M. A. Babar，“基于概念漂移的自动化软件漏洞评估”，载于*2019*年*IEEE/ACM*第*16*届软件仓库挖掘国际会议（*MSR*）。IEEE，2019年，第371–382页。[69] S. McIntosh 和 Y. Kamei，“诱发修复的变更是否是移动目标？即时缺陷预测的纵向案例研究”，*IEEE*软件工程汇刊，第44卷，第5期，第412–428页，2017年。[70] B. Fuglede 和 F. Topsoe，“詹森-香农散度与希尔伯特空间嵌入”，载于国际信息论研讨会，*2004*年。*ISIT 2004*。会议论文集。IEEE，2004年，第31页。[71] M. G. Kendall，“一种新的秩相关度量”，*Biometrika*，第30卷，第1/2期，第81–93页，1938年。[72] A. D. Sawadogo、T. F. Bissyand´e、N. Moha、K. Allix、J. Klein、L. Li 和 Y. Le Traon，“Sspcatcher：学习捕获安全补丁”，《实证软件工程》，第27卷，第6期，第1–32页，2022年。[73] A. Garg、R. Degiovanni、M. Jimenez、M. Cordy、M. Papadakis 和 Y. LeTraon，“从已知信息中学习：如何利用含噪历史数据进行漏洞预测”，*arXiv*预印本 *arXiv:2207.11018*，2022年。