计算机科学
COMPUTER SCIENCE

基于可解释性人工智能的软件工程技术方法综述

邢颖

引用本文

邢颖. 基于可解释性人工智能的软件工程技术方法综述[J]. 计算机科学, 2023, 50(5): 3-11.

XING Ying. Review of Software Engineering Techniques and Methods Based on Explainable Artificial Intelligence [J]. Computer Science, 2023, 50(5): 3-11.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.221100159

基于可解释性人工智能的软件工程技术方法综述

邢 颖
北京邮电大学人工智能学院 北京 100876

摘 要 在信息处理与决策方面,人工智能(AI)方法相比传统方法表现出了优越的性能。但在将AI模型投入生产时,其输出结果并不能保证完全准确,因此AI技术的"不可信"逐渐成为AI大规模落地的一大阻碍。目前人工智能被逐步应用到软件工程中,其过度依赖历史数据和决策不透明等弊端愈发明显,因此对决策结果做出合理的解释至关重要。文中对可解释性人工智能的基本概念、可解释模型的评估进行了详细阐述,探讨了软件工程与可解释人工智能结合的可行性;同时调研了相关文献,对软件工程中的恶意软件检测、高风险组件检测、软件负载分配、二进制代码相似性分析这4个人工智能的典型应用方向做出分析,讨论如何通过可解释AI揭示系统输出的正确程度,进而提高系统决策的可信度;最后展望未来软件工程与可解释人工智能相结合的研究方向。

关键词:可解释人工智能;软件工程;恶意软件检测;代码相似性分析
中图法分类号 TP311

Review of Software Engineering Techniques and Methods Based on Explainable Artificial Intelligence

XING Ying
School of Artificial Intelligence, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract In terms of information processing and decision-making, artificial intelligence(AI) methods have shown superior performance compared to traditional methods. However, when AI models are put into production, their output results are not guaranteed to be completely accurate, so the "unreliability" of AI technology has gradually become a major obstacle to the large-scale implementation of AI. As AI is gradually applied to software engineering, the drawbacks of over-reliance on historical data and non-transparent decision-making are becoming more and more obvious, so it is crucial to provide reasonable explanations for the decision results. This paper elaborates on the basic concepts of explainable AI(XAI) and the evaluation of explanation models, and explores the feasibility of combining software engineering with explainable AI. Meanwhile, it investigates relevant researches in software engineering, analyzes the four typical application directions of XAI, namely, malware detection, high-risk component detection, software load distribution, and binary code similarity analysis, to discuss how to reveal the correctness of the system output, thereby increasing the credibility of the software system. This paper also gives insights into the research direction in combining software engineering and explainable artificial intelligence.

Keywords Explainable artificial intelligence, Software engineering, Malware detection, Code similarity analysis

1 引言

近年来,随着人工智能(AI)技术飞速发展,人工智能的应用也迎来了新浪潮。然而,许多AI系统都不能向用户解释它们的自主决策和行为。对于国防、医学、金融和法律等领域的许多关键应用程序来说,解释对于用户理解、信任和有效管理这些新的人工智能系统至关重要。

过去几年间,在某些商业逻辑对技术友好或者伦理法规暂时稀缺的领域,人工智能取得了技术和商业上的快速突破。而当深度学习商业化被运用到某些对技术敏感、与人的生存或安全关系紧密的领域,如自动驾驶、金融、医疗和司法等高风险应用场景时,原有的商业逻辑在进行技术更替的过程中就会遇到阻力。究其原因,是模型透明性的缺失,人们无法理解模型的决策逻辑,因而无法信任其做出的决策。为此,学术界于2004年提出了可解释人工智能(Explainable Artificial Intelligence,XAI)[1]的概念。为使用户理解、信任和管理新一代人工智能系统,2016年10月美国国防部高级研究计划局(Defense Advanced Research Projects Agency,DARPA)启动了"可解释的人工智能"项目[2]。

在软件工程领域,人工智能可以准确并快速地处理数据,还可以保证数据的完整性,具有很大优势,因此,在软件工程中人工智能有很高的应用价值。基于此,本文将重点关注可解释人工智能与软件工程的结合,讨论其中的几个典型应用场景,并对二者未来可能的结合方向和发展前景进行了展望。

到稿日期:2022-11-19 返修日期:2023-02-02
通信作者:邢颖(xingying@bupt.edu.cn)

2 相关内容概述

本节将梳理可解释人工智能的发展及其评估方式，并针对人工智能在软件工程领域的应用进行探讨。

2.1 可解释人工智能概述

XAI这个术语最早是由Van等[1]于2004年提出，用于描述他们的系统在模拟游戏应用中解释AI控制实体行为的能力。虽然这个术语相对较新，但可解释性问题从20世纪70年代中期就已经存在了，具体来自于当时研究人员研究专家系统[3]的解释。然而，随着人工智能在机器学习[4](Machine Learning，ML)上的巨大进步，解决这一问题的速度已经放缓。人工智能研究的重点已经转移到实现模型和算法上，开始强调预测能力，而解释决策过程的能力则退居其次。

近年来，XAI的话题再次受到学术界和实践者的关注。图1为使用谷歌趋势展示的2004－2022年之间的数字变化，说明了研究者对XAI的研究兴趣明显复苏。从技术上讲，对于XAI没有标准和普遍接受的定义。在DARPA于2016年提出Explainable Artificial Intelligence(XAI)[2]项目之后，XAI的概念也开始被普遍接受。而更早出现的是可理解的机器学习(Interpretable Machine Learning，IML)[4]的概念。总体来说，XAI的研究范围包含IML。

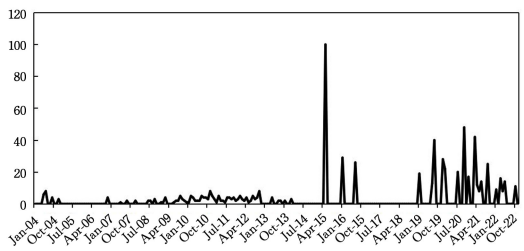

图1 XAI的谷歌研究兴趣趋势图

Fig. 1 Trend about XAI from Google Research Interest

根据DARPA[2]的描述，XAI的目标是在保持高水平的学习性能的同时，产生更多可解释的模型；并使人类用户能够更多地理解、适当地信任、有效地管理新一代人工智能伙伴。正如FAT-ML研讨会[3]所提出的，在ML中启用可解释性的目标是确保算法决策以及驱动这些决策的任何数据能够以非技术术语向最终用户和其他利益相关方解释。Yeung等[5]将XAI视为“打开ML黑箱的一种创新”和“一种创建模型和技术的挑战，这些模型和技术既准确又能提供良好的、值得信赖的解释，满足客户的需求”。这些关于人工智能可解释性的不同观点是由研究领域和关心的问题不尽相同导致的。

此外，有两个专业用语经常混淆：可理解性(Interpretability)和可解释性(Explainability)。可理解性主要指向人类提供可理解的能力；可解释性指使用解释作为人类和智能模型之间的接口，作为模型代理能够被人类所理解[6]，倾向于指代原模型不可理解，而需要构造事后模型进行解释。在以往的研究中，经常会出现这两个概念混用的情况[7]。

文献[8]从受众的角度给出了XAI的定义：针对特定的用户，XAI指可以提供细节和原因以使模型能够被简单、清晰地理解的技术。宽泛地讲，XAI技术指所有能够帮助开发者或者用户理解人工智能模型行为的技术[9]，包括原模型自身可解释和模型需要事后解释两种类型。针对智能决策受益者，人工智能可解释指可以给不同背景知识的用户以简单、清晰的方式，对智能决策过程的根据和原因进行解释的方法，其目标是将黑盒人工智能决策转化为可解释的决策推断，使用户能够理解和相信决策[10-13]。

2.2 人工智能可解释性评估

许多文献强调了可解释性评估的必要性，但也指出了评估指标缺乏的现实[14-16]。文献[15]调研了381篇XAI相关文献，其中只有5%的研究尝试对XAI进行评估。类似地，文献[17]发现，78%的关于决策支持系统的研究缺乏结构化的可解释性评估工作。只有系统科学地评估XAI，才能提高其可靠性和实用性，并推动相关的研究和应用。具体来说，可解释性评估的目的包括：1)为解释方法之间的比较提供科学、有效的评价标准；2)评价XAI是否实现了预期的目标[18]。

文献[19]将可解释性评估分为3类：1)基于应用(Application-ground)：在实际应用场景下，由用户(尤其是专业人员)评估可解释性；2)基于人(Human-ground)：设计简化的任务，利用基于用户实验获得的评价指标来评估解释性；3)基于功能(Function-ground)：无需用户参与，通过可解释代理模型或量化指标来评估可解释性，例如，决策树的深度、模型预测的不确定等。基于应用的评估是最理想的，因为它评估了XAI在实际应用中用户对解释的反馈，然而用户的参与导致评估成本较高，且评估结果依赖于所选的专业人员的领域。基于功能的评估无需人的参与，但其评估结果的有效性难以保证，因为量化指标可能并不能很好地反映可解释性。基于人的评估是一种折中方法，比基于应用的评估成本低，但比基于功能的评估更有效。

文献[20]根据用户是否参与评估，将可解释性评估分为主观评估、客观评估两类。主观评估利用用户或专家反馈来评估可解释性；客观评估利用客观评估指标来量化评估可解释性。以上基于应用的评估和基于人的评估属于主观评估，而基于功能的评估属于客观评估。

2.3 软件工程中的人工智能

软件工程(Software Engineering，SE)起源于19世纪80年代，当时查尔斯·巴贝奇(Charles Babbage)制造了一台能够进行各种数学计算的机器。巴贝奇的目的是消除人类手工计算时产生的固有错误[21]。许多人把这个程序称为第一个软件程序。软件，从它最初的阶段开始，旨在帮助人类自动化那些需要一定水平的“智能”。

在计算机科学中，SE领域是一个主导工业领域[22]，研究者对SE过程的各个方面都有大量研究。随着该领域的发展，许多重要的问题需要得到一致的回答，例如如何估计一个项目的时间和成本，何时停止测试系统，如何定位和消除错误，如何增量开发和重新设计，如何重构代码，如何将需求转化为特性等。与其他工程学科类似，构建软件遵循定义良好的过程被称为生命周期。文献[23]介绍了许多不同的生命周期，许多已经在工业中得到了应用。

生命周期中的每个阶段都有自己的挑战、缺点和最佳实践，因此，每个阶段都有自己的研究领域。同时，另一个代表机器智能的研究领域人工智能(AI)也在蓬勃发展。许多

研究者声称这两个研究领域(SE 和 AI)没有充分的相互作用或重叠[21-25]。在许多分支领域中,人工智能的目标是解决问题、表示智能、工程知识、识别模式、从经验中学习,并最终回答艾伦·图灵提出的著名问题:"机器能思考吗"?而根据人工通用智能的最新定义[24-25],人工智能具有在复杂环境中执行复杂任务的能力。2008 年,Rech 等声称 AI 和 SE 的学科有许多共性,两者都处理来自真实世界对象的建模,如业务流程、专家知识或流程模型[24]。

在为了解决软件工程领域的一些问题而不断探索的过程中,各种基于 AI 的数据挖掘技术被用于从软件生命周期获得的数据中提取有用的知识。软件生命周期的所有阶段都会产生大量的数据[26],通过对这些庞大数据的挖掘,可以成功地发现和识别用于生产重用的组件。在这个过程中,集成 AI 可以改进软件开发的整个过程,使其具有可重用性,并支持标准的自动化,进而促进软件智能化。Feldt 等提出了 AI 在 SE 应用级别的分类法,该分类法根据 AI 应用点、使用的 AI 技术类型和允许的自动化级别对应用程序进行分类[27]。这些都证明了人工智能在软件工程领域是一个非常有前景的研究领域,对其进行深入探索对于解决各种软件工程相关的问题是有意义的。

谢涛团队[28]针对人工智能在软件工程中的应用做过详细举例,如一旦软件系统开始运行,开发人员就会面临大量的调优选择。那时,基于多目标遗传算法的工具可以找到并配置该软件。而一旦调优算法配置和执行,就可以通过人工智能工具对软件进行监控和优化,从而减少软件所需的云资源。此外,在共享软件时,该团队发现构建测试套件非常有用,可以检查是否有人为的更改对系统造成了损害。对于大型测试套件,在云环境中运行速度可能会很慢,成本也很高。为了降低成本,并更快速地向开发人员反馈,人工智能工具可以学习如何对最有可能失败的测试进行优先级排序,一旦系统开始运行并且测试用例结果可用,那么 AI 就可以为软件开发过程提供很多支持,同时 AI 工具也可以自动查找和修复有 bug 的代码[29]。因此,利用人工智能进行知识发现在软件工程领域的自动化软件重用和设计中具有重要意义[30-32]。

人工智能技术在软件工程中也有创新应用,如软件开发项目的时间表以及信息技术管理、软件应用开发和软件安全[33]。Raza 讨论了软件开发中的人工智能技术,以及基于人工智能的系统风险管理[34]。

3　软件工程中的可解释人工智能

基于上一节的相关介绍,本节分别针对恶意软件检测、高风险组件检测、软件负载调配、代码相似性分析这 4 个比较常见的典型场景,对可解释人工智能在软件工程中的应用进行具体的论述。同时,可解释人工智能技术与软件工程的结合涉及软件需求、代码生成、项目管理、软件测试、智能运维、以人为中心的软件开发等很多内容,由于篇幅限制,本节将不再分别描述。可以肯定的是,可解释人工智能将在软件工程中发挥越来越重要的作用。

3.1　恶意软件检测中的 XAI 应用

恶意可执行程序指未经用户同意而设计的侵入或破坏计算机系统的程序,这些程序已经严重威胁到计算机系统的安全。根据恶意代码的传播方式,通常可以将其分为以下几类[35-37]:病毒、后门、间谍软件、木马和蠕虫。由于恶意可执行程序会造成巨大的损失和破坏,因此,对恶意软件的检测已经成为软件安全领域亟待解决的问题之一。

目前,对抗恶意软件最重要的防线是反病毒程序,它们大多都使用基于签名的方法来识别威胁[38-39]。签名是一个简短的字节串,对于每个已知的恶意软件都是唯一的。然而,在签名的提取和生成过程中,这些签名很容易被绕过。因此,快速分析已知恶意软件和未知恶意软件样本的变体是至关重要的。

早在 2010 年,金山杀毒实验室的病毒分析师就总结称,每天需要分析的样本约为 7 万个,他们称之为"灰名单"[40]。这表明,需要一种自动、有效和健壮的工具来对"灰色列表"进行分类,对于分类决策需要做出解释性判断。

最近有一些研究应用数据挖掘和机器学习技术来尝试检测新的恶意可执行程序[41-45]。这些技术的性能在很大程度上取决于用于描述可执行程序和分类器[46]的特性集。"可解释的字符串"不再是简单的可打印字符串[42-43],而是来自应用程序编程接口调用的输入表中[44],它可以反映程序代码段和携带语义解释的字符串行为,这些语义解释可以反映攻击者的意图和目标。可解释字符串具有很好的静态特性,因为它们不仅可以解析恶意可执行程序的可能行为,还可以捕获恶意软件作者的意图和目标。

文献[47]中收集了 39838 个可执行程序,其中 8320 个被称为良性可执行程序,另外 31518 个是 4 种不同类型的恶意程序(即后门程序、间谍软件、木马程序和蠕虫程序)。所有样本均来自金山杀毒实验室。从这些可执行文件中,总共提取了 963556 个可解释字符串。

为了开发一个基于可解释字符串的自动高效恶意软件检测系统,文献[48]提出必须解决以下 4 个问题:可扩展性、泛化、效率和有效性。针对上述挑战,文献[48]利用支持向量机(Support Vector Machine,SVM)[49]开发了基于可解释字符串的恶意软件检测系统(String Based Malware Detection System,SBMDS),对文件样本进行分类并预测恶意软件的确切类型。SBMDS 由 3 个主要模块组成:特征解析器、基于支持向量机的分类器和恶意软件检测器[50-54]。特性解析器用于从文件示例中提取可解释的字符串。支持向量机已被证明是有效的分类器,它能够处理大的特征空间并具有良好的泛化性。然而,支持向量机在大规模数据上是不可行的,因为它的训练复杂度高度依赖于数据集[47]的大小。集成分类技术在这方面很有优势,因此使用基于支持向量机的集成学习来应对可扩展性的挑战。

恶意软件检测分为 4 个主要步骤:1)通过特征解析器构造可解释的字符串;2)进行特征选择,选择与不同类型恶意软件相关的信息字符串;3)采用基于支持向量机的集成方法构建分类器;4)进行恶意软件检测。

上述系统能够对恶意软件及其变体的分类决策做出解释性判断,通过可解释字符串解析恶意可执行程序的可能行为,通过特征解析器提取每个文件的可解释字符串,并预测一个

程序是否是恶意的以及确切类型。系统在可扩展性、泛化、效率和有效性方面取得了较好的性能。

3.2 高风险组件检测中的 XAI 应用

在大型系统[55-57]中，少数软件组件要对不成比例的大量故障负责。因此，如果能够识别出可能产生大量故障的组件，就可以将验证和测试过程集中在这些组件上，以最小的成本优化软件系统的可靠性。为了达到这一目的，需要建立定量模型来预测哪些组件可能包含最高密度的故障。然而，构建这样的模型是一项艰巨的任务。在软件工程中，收集的数据通常是较少的、不完整的和异构的[58]。这为模型构建和解释带来了几个问题，例如数据集小、模型不准确、产生异常值等。因此，需要对高风险组件(那些在系统或验收测试期间高概率产生故障的组件)进行可靠的分类，并帮助理解这种高风险的原因。这样可以深入了解软件开发过程，及时采取补救行动，并在未来做出更好的过程决策。

文献[59]表明开集识别算法(Open-Set Recognition, OSR)可以被用作逻辑回归或分类树的替代方法来生成软件系统中的风险经验模型，并且它可以产生更准确的结果。本文讨论了与解释生成的模型相关的问题，具体介绍如何用 OSR 在 Ada 系统中识别高风险组件的特征，并分析在软件开发过程中高风险组件是如何引发故障的。文献[59]提出了一种进化版本的 OSR 算法(OSR 的一个早期版本被应用于项目成本估算[58])，旨在使 OSR 模型能更准确、更容易解释。具体而言，改进算法从 3 个方面提高了模型的可解释性和精度。首先，它提供了一种自动处理解释变量范围离散化的机制。其次，为 OSR 提供了处理谓词的能力，使模型能够引出在 OSR 的前一个版本中不可见的变量组合的影响。最后，为识别模式之间的相似性提供了支持，这有助于用户解释模型。这几个方面都有助于 OSR 处理的模型能够很好地解释变量之间的相互依赖性和相互作用。此外，与文献[58]相比，文献[59]将 OSR 建模技术应用于 Ada 组件的低或高风险分类问题中，而不是项目成本估计(在连续范围内的预测)。

对一个软件系统的所有部分进行同等的测试和验证工作并不是非常有效，特别是在资源有限和调度紧张的情况下。因此，需要能够区分低/高故障频率组件，以便在需要时集中测试/验证工作。这种策略可以检测到更多的故障，从而提高整个系统的可靠性。从一般的角度来看，OSR 方法是一种成功地集成了统计学的数据分析框架以及针对特定软件工程需求的经验建模的机器学习方法，它也为模型解释提供了支持。

3.3 软件负载调配中的 XAI 应用

随着现代软件的可配置性不断提高，巨大的配置空间和这些配置参数之间的复杂交互[60-63]使得用户在调优性能方面面临着巨大的挑战。吞吐量[64]、延迟[65]和能耗[66]等配置对应用程序性能都会造成很大的影响。因此，帮助调优应用程序配置已经成为一个重要而富有挑战性的研究领域。

为了高效地配置应用程序，机器学习(ML)方法被应用于建模配置参数和性能之间的复杂关系。其中大多数工作是对应用程序的配置参数进行随机采样，用这些样本训练学习器，预测未采样配置的性能，进而将应用程序部署在具有最佳预测性能的配置中[67-68]。这些学习器中最常用的方法有神经网络[69]、随机森林[70]，以及高斯过程回归[71-72]等方法，然而此类方法均是黑箱，这意味着用户无法看到其内部工作[73]，且有两个实际限制需要解决。

(1)难以整合先验知识。以前的基于 ML 的工具几乎没有能力利用用户之前收集到的信息，这些工具必须从头开始，生成大量的随机配置并收集它们所对应的系统性能。考虑以下两种具体的情况，如图 2 所示。

场景 1　由于一些错误的配置设置，工作负载 X 运行缓慢[61,74]。为了找到更好的配置，用户可以从头随机采样新的配置，为每个配置运行 X 以收集性能数据，并构建一个新的 ML 模型。为了避开这个繁琐的过程，用户可以利用已知的配置(尽管速度很慢)来构建一个可以推断出更好配置的模型。

场景 2　工作负载 Y 被设置为在硬件 A 上的最优配置下工作，由于某些硬件升级或调度问题，硬件 A 不再可用，Y 必须部署在硬件 B 上。使用之前的方法，用户将随机采样 B 上的配置，收集性能数据，并构建 ML 模型[75-77]进行配置搜索。新方式是整合先验知识，通过已知在硬件 A 上快速运行的配置，开发一个模型来进行从 A 到 B 的配置空间外推。可以看出整合先验知识比从头开始使用随机采样能获得更好的性能。

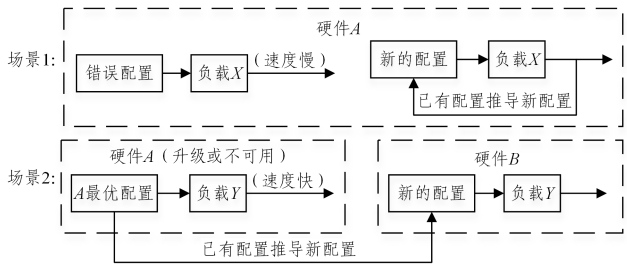

图 2　两类数据收集场景

Fig. 2　Two types of data collection scenarios

(2)缺乏可解释性。之前的工作只评估黑盒模型的预测精度，但很难向用户解释[78]。再次考虑两个场景：

场景 1　用户希望了解导致低性能的底层因素——从软件应用程序到硬件资源——从而避免未来出现类似的问题[79-81]。

场景 2　用户希望与学习者进行交互，了解为什么之前的配置比较慢，以及如何实现更快的配置。他们还希望这些知识能在不同的硬件和工作负载上得到推广[68]。

为了解决这些问题，文献[82]提出了 Gil 和 Gil+配置外推工具，它们结合了先验知识，以实现高性能，并提供可解释性。为了整合先验知识，Gil 和 Gil+使用现有的配置信息作为初始训练样本，然后推断出更高性能的配置。实现可解释性的关键是迭代构建近似于真正非线性函数的线性模型，这样最重要的配置参数可以被解释为线性模型中它们的权重以及随着模型迭代更新这些权重的变化的组合。为了帮助理解应用程序和它所运行的计算机系统之间的关系，Gil+用一个层次模型对 Gil 进行了扩展，该模型连接了应用程序级配置、底层系统指标(如上下文切换和缓存丢失)和最终的应用程序性能，如图 3 所示。

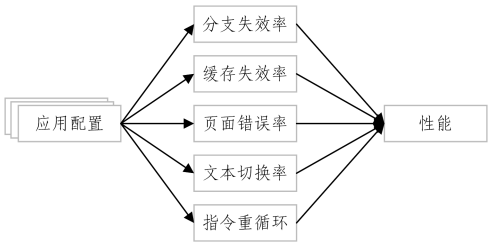

图 3　与应用程序配置、底层系统指标和性能相关的 Gil+层次模型

Fig. 3　Gil+ hierarchical model related to application configuration, underlying system metrics and performance

文献[82]中还开发了图形化工具来可视化这些关系,以帮助用户解释学习结果,并提供了关于它们如何实现最高性能的应用程序配置的图形化解释。另外,文献[82]的实验说明 Gil 和 Gil+的应用性能与最佳黑盒学习器相当,并且优于那些从头开始的随机采样训练,这证明了将先验知识纳入学习过程的好处。此外,图形化可视化工具使用户能够与学习者进行交互,并解释应用程序级配置、低级系统指标和性能之间的关系,以此达成软件工程将机器学习技术应用于性能建模,实现先验知识和模型的可解释性。

3.4　二进制代码相似性分析中的 XAI 应用

程序员重用现有代码来构建新软件,通常的做法是从另一个项目中找到源代码,然后根据自己的需要复用这些代码[83]。没有经验的开发人员甚至从 Internet 复制和粘贴代码样例,以简化开发过程。这一趋势对软件安全和隐私有着深刻的影响。当程序员从现有项目中获取了一个有 bug 的函数副本时,即使最初的开发人员修复了它,这个 bug 也不会改变。

由于源代码不易获取,因此二进制代码相似性搜索成为关键技术[84]。遗憾的是,二进制代码相似性检测也并不容易。因为二进制代码缺乏高级抽象,例如从二进制代码中很难确定内存单元是表示整数、字符串还是另一种数据类型。而且,确定精确的函数边界也是一个挑战[85-86]。因此,检测二进制代码之间的相似性一直是许多领域的研究课题,如抄袭检测[87-88]、作者身份识别[89],以及漏洞发现[90-101]。尽管研究者对二进制代码相似度分析(Binary Code Similarity Analysis,BCSA)的研究兴趣激增,但在这一领域进行新的研究仍然具有很大的挑战性,原因如下。

首先,大多数方法只关注最终结果,而不考虑方法背后的精确推理。二进制文件在不同的体系结构、编译器类型或编译器版本之间是完全不同的,因此在相同程序[91-95,99,102-106]的跨架构和跨编译二进制文件上应用 BCSA 技术是一个研究趋势。这些方法旨在度量两个或多个看似不同的二进制文件之间的相似性。这些二进制文件是由针对不同指令集的不同编译器生成的。为此,许多方法都设计了基于机器学习的方法来提取二进制文件的语义,这些方法都假设它们的语义不在编译器或目标体系结构之间发生变化。然而,没有一种现有的方法能够清楚地证明这种基于语义的复杂分析的必要性,因为大多数现有的方法都利用的是无法解释的机器学习技术。此外,这些研究不能解释为什么 BCSA 算法只在某些基准上工作,而在其他基准上不工作。

其次,关于 BCSA 的现有研究都使用自己的基准来评估所提出的技术,这使得在彼此之间的方法比较变得困难。此外,复现以前的结果通常是不可行的,因为大多数研究人员并不会透露他们的源代码和数据集。

基于上述分析,文献[107]首先对术语进行精确定义,并对以往文献中使用的特征进行分类,以统一术语,建立 BCSA 的知识库;然后构建了一个全面和可复制的 BCSA 基准,以帮助研究人员轻松扩展和评估他们的方法;最后设计了一个可解释的特征工程模型,并进行了一系列的实验,以研究编译器及其配置和它们的目标架构对生成的二进制文件的语法和结构特征的影响。

文献[107]将所构建的基准称为 BINKIT,它包含了各种现有基准。其中包括 8 个体系结构、9 个不同的编译器、5 个优化级别等。BINKIT 包含 243 128 个不同的二进制文件和 36 256 322 个函数,这些函数是在 51 个真实的软件包中为 1 352 种不同的编译器选项组合构建的。文献中还提供了一个自动化脚本,帮助扩展 BINKIT 以处理不同的体系结构或编译器版本。使用各种编译器选项交叉编译软件包非常具有挑战性,因为其中涉及许多环境问题。据该文献介绍,BINKIT 是第一个可复制和可扩展的 BCSA 基准。

同时,文献[107]中设计了一个可解释的 BCSA 模型,其本质是计算 BCSA 特征值之间的相对差异。然后构建了一个名为 TIKNIB 的 BCSA 工具。通过 TIKNIB,Kim 等发现了 BCSA 领域的几个误区,也为未来的研究提供了新的见解。

首先,BCSA 目前的研究是建立在一个假设之上:二进制文件在不同的体系结构、编译器类型或编译器版本之间是完全不同的。然而,文献中的研究表明,事实并非如此。例如,该文中演示了一个简单的数字函数,比如函数中传入/传出调用的数量,在为不同体系结构编译的二进制文件中非常相似;文中还介绍了其他的一些基本特性,这些特性在编译器类型、编译器版本甚至过程内混淆方面都能保持健壮。这些发现表明具有这些简单特征的 TIKNIB 可以达到与最先进的 BCSA 工具相当的准确率,如依赖于复杂深度学习模型的 VulSeeker[108]。

其次,大多数研究人员专注于从二进制文件中向量化特征,而不是恢复编译过程中丢失的信息,如变量类型。然而,实验结果表明,专注于后者对 BCSA 是非常有效的。具体来说,在所有基准测试中,带有恢复类型信息的 TIKNIB 达到了 99%以上的准确性,这是非常好的结果。这一结果强调,从二进制文件中恢复类型信息与为 BCSA 开发一种新的机器学习算法同样重要。

4　未来发展方向

随着人工智能在软件工程中的快速发展,可解释人工智能 XAI 在软件工程中的应用受到越来越多的关注。本文介绍了 XAI 的历史和基本思想,并探讨了其在软件工程中的一些典型应用。

当前软件工程中的 XAI 研究仍然处于早期发展阶段,一些研究工作有待进一步开展,未来的研究方向包括:

(1)软件工程中可解释性的客观评估方法。相比主观

评估方法，客观评估方法较少，这是因为：1）有些可解释的特性是概念性的，与用户主观感受相关（如满意度），难以进行客观量化；2）有些可解释的特性目前还缺乏可靠的量化方法。客观评估可以实现 XAI 的快速、自动评估，弥补主观评估成本高的不足，是可解释性评估的未来发展方向。

（2）可解释性评估的统一标准。可解释性评估标准需考虑两方面的因素：1）软件工程 XAI 的评估目标不同，不同类型的指标具有不同的解释需求，因此需要根据应用领域和用户类型来划分可解释评估工作，针对不同的大类分别建立一套标准；2）软件设计者或用户可能不清楚需要何种类型、何种程度的解释，因此，需提供可解释性评估列表，引导软件工程 XAI 的评估向着规范化方向发展。

（3）可解释性在软件工程中的细分应用类型。对于人工智能和软件工程结合的问题，由于细分方向较多，目前还难以给出一致性较好的解释方法。从大的方面来说，软件需求、代码生成、项目管理、软件测试、智能运维、以人为中心的软件开发等很多内容都涉及可解释人工智能；从小的方面来说，例如软件测试，就存在缺陷预测、恶意检测等多个细分方向，因此可以探求不同应用的共性，针对一类应用研究解释性的系统方法。

（4）可解释性在安全方面的考虑。解释可能会给 XAI 和用户带来安全隐患，这个问题在软件的许多方面都存在：1）解释方法往往会揭示底层模型和训练数据信息，展示软件底层代码，其展示的信息可能包含模型和用户信息，从而导致隐私泄露，因此，需要考虑 XAI 的隐私性；2）解释中包含的信息可能会被恶意利用，进而发现模型漏洞和脆弱点，实施对 XAI 的恶意攻击，因此解释需要考虑安全性因素。

结束语 本文概述了可解释性人工智能的基本概念，并结合软件工程的相关问题做了相关调研，从几个典型方向介绍了软件工程中可解释人工智能的应用。尽管目前在软件方面的可解释人工智能已经取得了一定的进展，但在评估指标、评估方法、应用领域、安全考量上仍有很多可探索的空间。在未来，随着人工智能在软件的测试和维护等任务中的应用逐步扩展，其可解释性方法和指标将会更加全面、客观。

参 考 文 献

[1] VAN T, FISHER W, MANCUSO M. An explainable artificial intelligence system for small-unit tactical behavior[C]// Proceedings 16th Conference Innovative Application Artificial Intelligence. 2004: 900-907.

[2] GUNNING D, AHA D. DARPA's explainable artificial intelligence(XAI) program [J]. AI Magazine, 2019, 40(2): 44-58.

[3] ALAMEDA P X, REEDI M, CELIS E, et al. FAT/MM'19: 1st International Workshop on Fairness, Accountability, and Transparency in Multi-Media[C]// The 27th ACM International Conference. 2019: 2728-2729.

[4] GUY T V. NIPS Workshop on Imperfect Decision Makers 2016: Preface[C]// Neural Information Processing Systems. 2016: 1-3.

[5] YEUNG C, HO D, PHAM B, et al. Enhancing Adjoint Optimization-based Photonics Inverse Design with Explainable Machine Learning[J]. ACS Photonics, 2022, 9(5): 1577-1585.

[6] AUGELLO A, INFANTINO I, LIETO A, et al. Towards A Dual Process Approach to Computational Explanation in Human-Robot Social Interaction[C]// International Joint Conference on Artificial Intelligence. 2017: 1-6.

[7] IERACITANO C, MAMMONE N, HUSSAIN A, et al. Anovel explainable machine learning approach for EEG-based brain-computer interface systems[J]. Neural Computing and Applications, 2021(3): 1-14.

[8] ESCALANTE H J, GUYON I, ESCALERA S, et al. Design of an explainable machine learning challenge for video interviews [C]// 2017 International Joint Conference on Neural Networks (IJCNN). Anchorage, AK, USA, 2017: 3688-3695.

[9] CHANDER A. Explainable AI: The New 42? [C]// Proceedings MAKE-Explainable AI. 2017: 295-303.

[10] WU Y N. Discuss the application of AI in software engineering [J]. Network Security Technology & Application, 2021(8): 52-54.

[11] LU Y. Application of Software Engineering Methods in Computer Software Development[J]. Software, 2022, 43(8): 176-178.

[12] LIU X. Application of AI in software engineering [J]. Computer & Network, 2021, 47(3): 48.

[13] MEIR K, MARK L. Artificial Intelligence Methods for Software Engineering [M]. World Scientific Publishing Company. 2021: 6-15.

[14] ADADI A, BERRADA M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence(XAI) [J]. IEEE Access, 2018, 6: 52138-52160.

[15] BARREDO A, DIAZ-RODRIGUEZ N, DEL SER J, et al. Explainable Artificial Intelligence(XAI): Concepts, Taxonomies, Opportunities and Challenges Toward Responsible AI [J]. Information Fusion, 2020, 58: 82-115.

[16] HOFFMAN R, MUELLER S, KLEIN G, et al. Metrics for Explainable AI: Challenges and Prospects [J/OL]. XAI Metrics, 2018: 1-50. (2018-12-11) [2023-03-06]. https://arxiv.org/abs/1812.04608.

[17] NUNES I, JANNACH D. A Systematic Review and Taxonomy of Explanations in Decision Support and Recommender Systems [J]. User Modeling and User-Adapted Interaction, 2017, 27(3): 393-444.

[18] MARKUS A, KORS J, RIJNBEEK P. The Role of Explain ability in Creating Trustworthy Artificial Intelligence for Health Care: a Comprehensive Survey of the Terminology, Design Choices, and Evaluation Strategies [J]. Journal of Biomedical Informatics, 2021, 113: 1-11.

[19] DOSHI-VELEZ F, KIM B. Towards a Rigorous Science of Interpretable Machine Learning [J/OL]. (2017-02-28) [2023-01-30]. https://arxiv.org/abs/1702.08608.

[20] VILONE G, LONGO L. Notions of Explainability and Evaluation Approaches for Explainable Artificial Intelligence [J]. Information Fusion, 2021, 76: 89-106.

[21] TURING A. Computing machinery and intelligence [J]. Mind, 1950, 59(236): 433-460.

[22] SORET B, JOSHI P, JAGTAP V. Use of artificial intelligence in

software development life cycle e a state of the art review [J]. Advanced Compute Engineering Communication Technology, 2015,4:2278-5140.

[23] MARCOS E. Software engineering research versus software development [J]. ACM SIGSOFT Software Engineering Notes, 2005,30(4):1-7.

[24] RECH J, ALTHOFF K. Artificial intelligence and software engineering: status and future trends [J]. Kunstliche Intelligenz, 2004,18:5-11.

[25] PRERSSMAN R, MAXIM B. Software Engineering: A Practitioner's Approach [M]. McGraw-Hill, Incorporated, 2014.

[26] TANGSRIPAIROJ S, SAMADZADEH M. A Taxonomy of Data Mining Applications Supporting Software Reuse [J]. Intelligent Systems Design and Applications. Advances in Soft Computing, 2003,23:303-312.

[27] FELDT R, NETO F, TORKAR R. Ways of applying artificial intelligence in software engineering[C]// Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering(RAISE '18). 2018:35-41.

[28] CARLETAN A, HARPER E, MENZIES T, et al. The AI Effect: Working at the Intersection of AI and SE [J]. IEEE Software, 2020,37(4):26-35.

[29] ZHU J H, ZHENG W, YANG F Y, et al. Software quality prediction based on ant colony optimization back-propagation neural network[J/OL]. (2023-01-12) [2023-03-06]. http://kns.cnki.net/kcms/detail/51.1307.tp.20230111.1318.004.html.

[30] ZHAO H J. Analysis of Software Testing Method of Dynamic Weighted Combined Neural Network Model [J]. Electronic Technology, 2022,51(4):70-72.

[31] LI Z, CUI Z Q, CHEN X, et al. Deep-SBFL: defect location method of deep neural network based on spectrum [J/OL]. (2022-11-15) [2023-03-06]. http://www.jos.org.cn/jos/article/abstract/6403.

[32] XU H R, WANG Y J, HAUNG Z J, et al. Compiler Fuzzing Test Case Generation with Feed-forward NeuralNetwork [J]. Journal of Software, 2022,33(6):1996-2011.

[33] MOHAMMADIAN M. Innovative Applications of Artificial Intelligence Techniques in Software Engineering [C]// Artificial Intelligence Applications and Innovations. AIAI 2010. IFIP Advances in Information and Communication Technology, 2010.

[34] RAZA E. Artificial intelligence techniques in software engineering(AITSE) [C]// International Multi-Conference of Engineers and Computer Scientists. 2009:18-20.

[35] ADLEMAN L. An abstract theory of computer viruses(invited talk) [C]// CRYPTO '88: Proceedings on Advances in cryptology. 1990:354-374.

[36] FILIOL E. Computer Viruses: from Theory to Applications [M]. Berlin; Springer Science & Business Media, 2006.

[37] MCGRAW G, MORRISETT G. Attacking malicious code: report to theinfosec research council [J]. IEEE Software, 2000, 17(5):33-41.

[38] FILIOL E. Malware pattern scanning schemes secure against black-box analysis [J]. Computer Virology, 2006,2(1):35-50.

[39] FILIOL E, JACOB G, LIARD M. Evaluation methodology and theoretical model for antiviral behavioural detection strategies [J]. Computer Virology, 2007,3(1):27-37.

[40] YE Y, LI T, CHEN Y, et al. Automatic malware categorization using cluster ensemble [C]// Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2010:95-104.

[41] CHRISTODORESCU M, JHA S, KRUEGEL C. Mining specificationsof malicious behavior[C]// Proceedings of ESEC/FSE07. 2007:5-14.

[42] KOLTER J, MALOOF M. Learning to detect malicious executable in the wild[C]// Proceedings of KDD. 2004:470-478.

[43] SCHULTZ M, ESKIN E, ZADOK E. Data mining methods for detection of new malicious executables. In: Security and privacy [C]// Proceedings of 2001 IEEE Symposium. 2001:38-49.

[44] SUNG A, XU J, CHAVEZ P, et al. Static analyzer of vicious executables(save) [C]// Proceedings of the 20th Annual Computer Security Applications Conference. 2004:326-334.

[45] WANG J, DENG P, FAN Y, et al. Virus detection using data mining techniques[C]// Proceedings of IEEE International Conference on Data Mining. 2003:71-76.

[46] REDDY D, PUJARI A. N-gram analysis for computer virus detection[J]. Computer Virology, 2006,2:231-239.

[47] YE Y, CHEN L, WANG D, et al. SBMDS: an interpretable string based malware detection system using SVM ensemble with bagging [J]. Journal in Computer Virology, 2009,5(4):283.

[48] DIETTERICH T. Machine learning research: Four current directions [J]. AI Magazine, 1997,18(4):97-136.

[49] BREIMA L. Bagging predicators [J]. Machine Learning, 1996, 24:123-140.

[50] DANG X, GONG D, YAO X, et al. Enhancement of Mutation Testing via Fuzzy Clustering andMulti-population Genetic Algorithm [J]. IEEE Transactions on Software Engineering, 2021, 48(6):2141-2156.

[51] WANG S Y, SUN Y Q, SUN J Z. Detection of Bad Smell in Code Based on BP Neural Network[J]. Computer Engineering, 2020,46(10):216-222,230.

[52] JIANG Y, WANG S, WU K Q, et al. Software defect prediction model based on quantum immune clonal BP algorithm[J]. Journal of Southwest Minzu University(Natural Science Edition), 2022,48(5):537-542.

[53] LI E H. Research on Accurate Prediction of Android Software defects based on Hybrid Neural Network[J]. Automation & Instrumentation, 2022,8:33-36,41.

[54] CUI M T, LONG S L, JIANG Y, et al. Research of Software Defect Prediction Model Based on Complex Network and Graph Neural Network[J]. Entropy, 2022,24(10):1373.

[55] BASILI V, PERRICONE B. Software errors and complexity: an empirical investigation[C]// Communications of the ACM. 1984:42-52.

[56] MUNSON J, KHOSHGOFTAAR Y. The detection of fault-prone programs [J]. IEEE Transactions on Software Engineering, 1992,18(5):423.

[57] SELBY R, PORTER A. Learning from examples: generation and

evaluation of decision trees for software resource analysis [J]. IEEE Transactions on Software Engineering, 1988, 14(12): 1743-1757.

[58] BRIAND L, BAILI V, THOMAS W. A pattern recognition approach for software engineering data analysis [J]. IEEE Transactions on Software Engineering, 1992, 18(11): 931-942.

[59] BRIAND L. Developing interpretable models with optimized set reduction for identifying high-risk software components [J]. IEEE Transactions on Software Engineering, 1993, 19(11): 1028-1044.

[60] NORBERT S, ALEXANDER G, CHRISTIAN K. Performance-influence models for highly configurable systems[C]// Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015: 284-294.

[61] KASHI V V, NACHIAPPAN N. Characterizing cloud computing hardware reliability[C]// Proceedings of the 1st ACM symposium on Cloud computing. 2010: 193-204.

[62] XU T L, JIN L, FAN X P, et al. Hey, you have given me too many knobs! understanding and dealing with over-designed configuration in system software[C]// Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015: 307-319.

[63] XU T Y, JIN X X, HUANG P, et al. Early detection of configuration errors to reduce failure damage[C]// 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016.

[64] ADAM B, GEORGE P, ANA K, et al. IX: A Protected Dataplane Operating System for High Throughput and Low Latency [C]// 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014: 49-65.

[65] DANA V, ANDREW P, GEOFFREY J, et al. Automatic database management system tuning through large-scale machine learning[C]// Proceedings of the 2017 ACM International Conference on Management of Data. 2017: 1009-1024.

[66] YI D, NIKITA M, HENRY H. Generative and multi-phase learning for computer systems optimization[C]// Proceedings of the 46th International Symposium on Computer Architecture. 2019: 39-52.

[67] MILTIADIS A, EERL T, PREMKUMAR D, et al. A survey of machine learning for big code and naturalness[J]. ACM Computing Surveys. 2018, 51(4): 1-37.

[68] WAN Z Y, XIA X, LO D, et al. How does machine learning change software development practices? [J]. IEEE Transactions on Software Engineering, 2019, 47(9): 1857-1871.

[69] ENGIN Ï, SALLY A, RICH C, et al. Efficiently exploring architectural design spaces via predictive modeling[J]. ACM SIGOPS Operating Systems Review, 2006, 40(5): 195-206.

[70] NARDI L, SOUZA A, KOEPLINGER D, et al. HyperMapper: a Practical Design Space Exploration Framework[C]// 2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). 2019: 425-426.

[71] YI D, RISI K, JONATHAN E. Multi-resolution Kernel Approximation for Gaussian Process Regression[C]// Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Long Beach, California, USA, 2017: 1-9.

[72] ATEFEH M, ANINDA M, BENJAMIN C, et al. Bayesian Optimization for Efficient Accelerator Synthesis[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2020, 18(1): 1-25.

[73] FINALE D B, EEN K. Towards a rigorous science of interpretable machine learning[J]. arXiv: 1702.08608, 2017.

[74] YIN Z N, DING Y, ZHOU Y Y, et al. How do fixes become bugs? [C]// Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. 2011: 26-36.

[75] JIANMEI G, KRZYSZTOF C, SVEN A, et al. Variability-aware performance prediction: A statistical learning approach [C]// 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2013: 301-311.

[76] CHRISTIAN K, ALEXANDER G, NORBERT S, et al. The interplay of sampling and machine learning for software performance prediction [J]. IEEE Software, 2020, 37(4): 58-66.

[77] FLAVIO M, CHRISTIAN K, MARCIO R, et al. A Comparison of 10 Sampling Algorithms for Configurable Systems[C]// Proceedings of the 38th International Conference on Software Engineering (Austin, Texas). 2016: 643-654.

[78] MOLNAR C. Interpretable machine learning[M]. Lulu. com, 2020.

[79] CHI L, SHU W, HENRY H. Statically inferring performance properties of software configurations [C]// Fifteenth EuroSys Conference (EuroSys'20). 2020: 1-16.

[80] VAN C K, JALEEL A, EECKHOUT L, et al. Scheduling heterogeneous multi-cores through performance impact estimation (PIE) [J]. ACM SIGARCH Computer Architecture News, 2012, 40(3): 216-224.

[81] SHU W, CHI L, HENRY H, et al. Understanding and auto-adjusting performance-sensitive configurations [J]. ACM SIGPLAN Notices, 2018, 53(2): 154-168.

[82] DING Y, PERVAIZ A, CARBIN M, et al. Generalizable and interpretable learning for configuration extrapolation[C]// Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering. 2021: 728-740.

[83] REISS S. Semantics-based code search[C]// Proceedings of the International Conference on Software Engineering. 2009: 243-253.

[84] 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.

[85] SHIN E, SONG D, MOAZZEZI R. Recognizing functions in binaries with neural networks[C]// Proceedings of the USENIX Security Symposium. 2015: 611-626.

[86] BAO T, BURKET J, WOO M, et al. ByteWeight: Learning to recognize functions in binary code[C]// Proceedings of the USENIX Security Symposium. 2014: 845-860.

[87] LUO L, MING J, WU D, et al. Semantics-based obfuscation-re-

silient binary code similarity comparison with applications to software plagiarism detection[C]// Proceedings of the International Symposium on Foundations of Software Engineering. 2014:389-400.

[88] ZHANG F,WU D,LIU P,et al. Program logic based software plagiarism detection[C]// Proceedings of the IEEE International Symposium on Software Reliability Engineering. 2014:66-77.

[89] MENG X,MILLER B,JUN K. Identifying multiple authors in a binary program[C]// Proceedings of the European Symposium on Research in Computer Security. 2017:286-304.

[90] PEWNY J,SCHUSTER F,BERNHARD L,et al. Leveraging semantic signatures for bug search in binary programs[C]// Proceedings of the Annual Computer Security Applications Conference. 2014:406-415.

[91] ESCHWEILER S,YAKDAN K,GERHARDS-PADILLA E. discovRE:Efficient cross-architecture identification of bugs in binary code[C]// Proceedings of the Network and Distributed System Security Symposium. 2016:58-79.

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

[93] GAO J,YANG X,FU Y,et al. VulSeeker:A semantic learning based vulnerability seeker for cross-platform binary[C]// Proceedings of the ACM/IEEE International Conference on Automated Software Engineering. 2018:896-899.

[94] DAVID Y,PARTUSH N,YAHAV E. FirmUp:Precise static detection of common vulnerabilities in firmware[C]// Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems. 2018:392-404.

[95] CHANDRAMOHAN M,XUE Y,XU Z,et al. Bin-Go:Cross-architecture cross-os binary search[C]// Proceedings of the International Symposium on Foundations of Software Engineering. 2016:678-689.

[96] FENG Q,WANG M,ZHANG M,et al. Extracting conditional formulas for cross-platform bug search[C]// Proceedings of the ACM Symposium on Information,Computer and Communications Security. 2017:346-359.

[97] SHIRANI P,COLLARD L,AGBA B. Binarm:Scalable and efficient detection of vulnerabilities in firmware images of intelligent electronic devices[C]// International Conference on Detection of Intrusions and Malware,and Vulnerability Assessment. Springer,2018:114-138.

[98] XUE Y,XU Z,CHANDRAMOHA M,et al. Accurate and scalable cross-architecture cross-os binary code search with emulation [J]. IEEE Transactions on Software Engineering,2018,45(11):1125-1149.

[99] LIU B,HUO W,ZHANG C,et al. αdiff:Cross-version binary code similarity detection with DNN[C]// Proceedings of the ACM/IEEE International Conference on Automated Software Engineering. 2018:667-678.

[100] DING S,FUNG B,CHARLANG P. Asm2Vec:Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization[C]// Proceedings of the IEEE Symposium on Security and Privacy. IEEE,2019:472-489.

[101] MASSARELLI L,LUNA G,PETRONI F,et al. SAFE:Self-attentive function embeddings for binary similarity [C]// International Conference on Detection of Intrusions and Malware,and Vulnerability Assessment. Springer,2019:309-329.

[102] PEWNY J,GARMANY B,GAWLIK R,et al. Cross architecture bug search in binary executables[C]// Proceedings of the IEEE Symposium on Security and Privacy. IEEE,2015:709-724.

[103] FENG Q,ZHOU R,XU C,et al. Scalable graph-based bug search for firmware images[C]// Proceedings of the ACM Conference on Computer and Communications Security. 2016:480-491.

[104] ZUO F,LI X,ZHANG Z,et al. Neural machine translation inspired binary code similarity comparison beyond function pairs [C]// Proceedings of the Network and Distributed System Security Symposium. 2019:1-15.

[105] MARASTONI N,GIACOBAZZI R,DALLA PREDA M. A deep learning approach to program similarity[C]// Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis. ACM,2018:26-35.

[106] REDMOND K,LUO L,ZENG Q. A cross-architecture instruction embedding model for natural language processing-inspired binary code analysis [J/OL]. (2018-12-23)[2023-01-30]. https://arxiv.org/abs/1812.09652.

[107] KIM D,KIM E,CHA S K,et al. Revisiting Binary Code Similarity Analysis using Interpretable Feature Engineering and Lessons Learned[J]. arXiv:2011.10749,2022.

[108] JIAN G,XIN Y,YING F,et al. VulSeeker-pro:enhanced semantic learning based binary vulnerability seeker with emulation [C]// The 2018 26th ACM Joint Meeting. 2018:803-808.

XING Ying, born in 1978, Ph. D, is a senior member of China Computer Federation. Her main research interests include software testing and deep learning.

(责任编辑:何杨)