软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007059]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

深度学习框架测试研究综述*

马祥跃[1], 杜晓婷[2], 采 青[1], 郑 阳[3], 胡 峭[3], 郑 征[1]

[1](北京航空航天大学 自动化科学与电气工程学院, 北京 100191)

[2](北京邮电大学 计算机学院 (国家示范性软件学院), 北京 100876)

[3](华为技术有限公司 可信理论、技术与工程实验室, 广东 深圳 518129)

通信作者: 郑征, E-mail: zhengz@buaa.edu.cn

摘 要: 随着大数据和计算能力的快速发展, 深度学习技术取得巨大突破, 并迅速成为一个具有众多实际应用场景和活跃研究课题的领域. 为了满足日益增长的深度学习任务开发需求, 深度学习框架应运而生. 深度学习框架作为连接应用场景和硬件平台的中间部件, 向上支撑深度学习应用的开发, 帮助用户快速构造不同的深度神经网络模型, 向下深度适配各类计算硬件, 满足不同算力架构和环境下的计算需求. 作为人工智能领域的关键基础软件, 深度学习框架中一旦存在问题, 即使是一个只有几行代码的缺陷都可能导致在其基础上构造的模型发生大规模失效, 严重威胁深度学习系统安全. 作为第 1 篇以深度学习框架测试为主题的研究性综述, 首先对深度学习框架发展历程和基本架构进行介绍; 其次, 通过对 55 篇与深度学习框架测试研究直接相关的学术论文进行梳理, 对深度学习框架缺陷特性、测试关键技术和基于不同测试输入形式的测试方法这 3 个方面进行系统分析和总结; 针对不同测试输入形式的特点, 重点探究如何结合测试关键技术来解决研究问题; 最后对深度学习框架测试尚未解决的难点问题进行总结以及对未来值得探索的研究方向进行展望. 可以为深度学习框架测试研究领域的相关人员提供参考和帮助, 推动深度学习框架的不断发展成熟.

关键词: 深度学习框架; 测试; 缺陷; 实证研究

中图法分类号: TP311

中文引用格式: 马祥跃, 杜晓婷, 采青, 郑阳, 胡峭, 郑征. 深度学习框架测试研究综述. 软件学报. http://www.jos.org.cn/1000-9825/7059.htm

英文引用格式: Ma XY, Du XT, Cai Q, Zheng Y, Hu Z, Zheng Z. Survey on Testing of Deep Learning Frameworks. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7059.htm

Survey on Testing of Deep Learning Frameworks

MA Xiang-Yue[1], DU Xiao-Ting[2], CAI Qing[1], ZHENG Yang[3], HU Zheng[3], ZHENG Zheng[1]

[1](School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China)

[2](School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China)

[3](Trustworthiness Theory, Technology & Engineering Lab, Huawei Technologies Co. Ltd., Shenzhen 518129, China)

Abstract: As big data and computing power rapidly develop, deep learning has made significant breakthroughs and rapidly become a field with numerous practical application scenarios and active research topics. In response to the growing demand for the development of deep learning tasks, deep learning frameworks have arisen. Acting as an intermediate component between application scenarios and hardware platforms, deep learning frameworks facilitate the development of deep learning applications, enabling users to efficiently construct diverse deep neural network (DNN) models, and deeply adapt to various computing hardware, meeting the computational needs across different computing architectures and environments. Any issues that arise within deep learning frameworks, which serve as the fundamental software

* 基金项目: 国家自然科学基金 (61772055, 61872169); 中央高校基本科研业务费专项 (2023RC06)
收稿时间: 2023-05-04; 修改时间: 2023-07-03, 2023-08-21; 采用时间: 2023-09-25; jos 在线出版时间: 2024-01-24

in the realm of artificial intelligence, can have severe consequences. Even a single bug in the code can trigger widespread failures within models built upon the framework, thereby posing a serious threat to the safety of deep learning systems. As the first review exclusively focuses on the testing of deep learning frameworks, this study initially introduces the developmental history and basic architectures of deep learning frameworks. Subsequently, by systematically examining 55 academic papers directly related to the testing of deep learning frameworks, the study systematically analyzes and summarizes bug characteristics, key technologies for testing, and methods based on various input forms for testing. The study explores how to combine key technologies to address research problems. Lastly, it summarizes the unresolved difficulties in the testing of deep learning frameworks and provides insights into promising research directions for the future. This study can offer valuable references and guidance to individuals involved in the research field of deep learning framework testing, ultimately promoting the sustained development and maturity of deep learning frameworks.

Key words: deep learning (DL) framework; testing; bug; empirical study

1 引 言

信息时代涌现出的海量数据不断推动着智能技术的创新. 2006 年, Hinton 等人[1]首次提出深度神经网络 (deep neural network, DNN) 的概念, 从此开启深度学习 (deep learning, DL) 时代. 之后, 伴随海量数据存储成本的快速下降和图形处理器 (graphics processing unit, GPU) 等基础硬件计算性能的不断提升, DNN 所具备的强大能力得以充分发挥, 逐渐成为引领人工智能浪潮的关键技术[2]. 在深度学习技术发展的初始阶段, 研究人员在设计搭建不同功能的 DNN 模型时往往需要编写大量的重复代码. 因此, 为了提高工作效率, 更好地实现深度学习任务, 研究人员开始将开发 DNN 模型的必要过程编写成算子 (operator) 和应用程序接口 (application program interface, API) 供其他人直接调用, 从而大幅降低了深度学习技术的门槛. 在不断积累发展的过程中, 涌现出越来越多的用于构建 DNN 模型的 DL 框架, 比如 TensorFlow[3]、PyTorch[4]和 MindSpore[5]等.

如图 1 所示, DL 框架逐渐占据对应用场景和硬件适配的双向主导权, 起到承上启下的作用——向上承接海量应用场景, 例如图像处理[6], 语音识别[7], 甚至是自动驾驶系统[8]等安全关键领域; 向下适配各类硬件基础设施, 满足不同硬件架构和部署环境下的计算需求. 以 2023 年华为发布的 PanGu-Σ[9]大语言模型 (large language model, LLM) 为例, PanGu-Σ 参数量多达 1.085 万亿, 在完成各种中文语言理解任务时的性能达到业界先进的水平. 这类大规模复杂深度学习模型的稳定运行, 正是依赖于 MindSpore 深度学习框架的极优设计与正确实现.

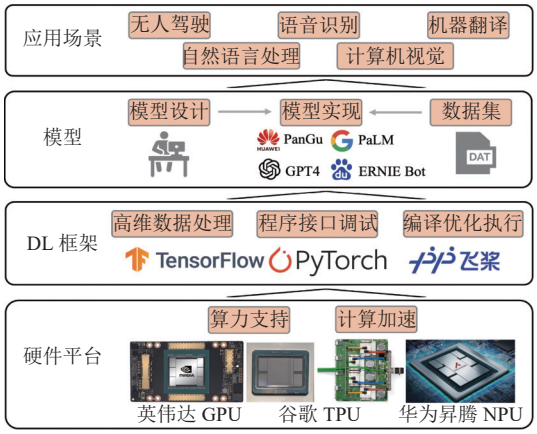

图 1 深度学习框架应用示意图

相比于 DL 框架的广泛应用, DL 框架中潜在的问题往往容易被忽视, 其安全性和质量难以得到保障. 例如, 2016 年谷歌的一辆自动驾驶汽车因为内部深度学习系统对路况的错误假设而低速撞向公交车的侧面, 引发车祸[10]; 2.3.24 以下版本的 PyTorch 采用了不安全的 YAML 文件加载方式[11], 攻击者利用这个安全漏洞可以将恶意代码添加到经过预训练模型的配置文件中, 导致模型性能降低或资源中断. DL 框架作为人工智能领域的关键基础

软件, 其安全性和质量尤为重要. 但目前 DL 框架相关的测试研究还比较滞后, 其缺陷特性难以明确, 测试方法和评价指标尚未系统化, 并且尚没有以 DL 框架为研究对象的综述论文发表.

作为第 1 篇 DL 框架测试研究综述, 本文共调研分析 55 篇与 DL 框架测试研究直接相关的论文 (相关论文来源已上传至 https://github.com/maxy-635/DLFrameworkSurvey). 在了解 DL 框架缺陷特性的基础上, 本文主要关注如何高效地生成测试输入 (测试输入生成问题), 如何判断 DL 框架的输出行为是否符合预期 (测试预言问题), 以及如何评估 DL 框架测试的有效性和充分性 (测试评估问题) 这 3 个研究问题. 本文梳理现有的 DL 框架测试研究论文, 聚焦基于不同测试输入形式的 DL 框架测试方法, 总结如何运用模糊测试等关键技术来解决框架测试中的上述 3 个研究问题. 本文可以帮助 DL 框架研究领域的研究人员尽快了解本领域的最新进展和亟待解决的难题, 同时有助于 DL 框架开发和维护人员了解目前 DL 框架中存在的缺陷特点, 掌握最先进的 DL 框架测试技术. 具体来讲, 本文主要贡献如下.

(1) 系统化梳理针对 DL 框架缺陷特性的实证研究工作, 总结 DL 框架缺陷的类别及其特性.

(2) 分析 DL 框架测试亟需解决的研究问题及其对应测试技术, 总结不同测试技术的应用情况和局限性.

(3) 分析基于不同输入形式的 DL 框架测试方法, 探究每篇论文中如何运用具体测试技术解决研究问题.

(4) 总结目前 DL 框架测试尚未解决的难点问题, 并对未来的研究方向给出建议.

2 深度学习框架背景知识

本节简要介绍深度学习框架的背景知识, 包括其发展历程和基本架构.

2.1 发展历程

深度学习框架和深度学习算法之间相互依赖的良性循环推动了深度学习框架和工具的快速发展. 如图 2 所示, 本文将 DL 框架发展历程总结为 3 个阶段, 并总结归纳不同阶段 DL 框架的技术特征.

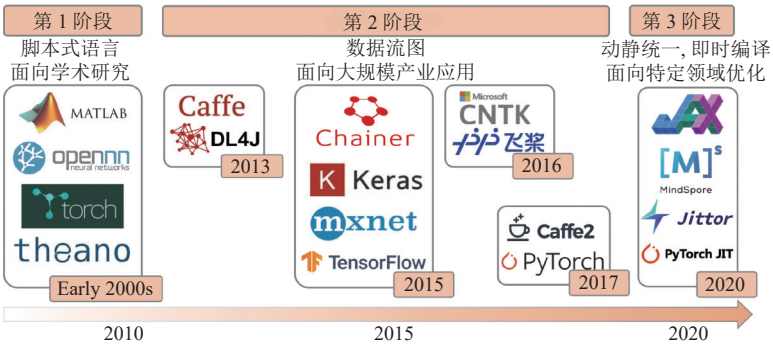

图 2 深度学习框架发展历程

21 世纪初至 2010 年, 第 1 阶段的 DL 框架以脚本式语言为技术特征, 主要面向部分学术研究领域. 神经网络的概念被提出之后, 一些可以用于描述和设计 DNN 的工具开始出现, 比如 Matlab[12]、OpenNN[13]、Torch[14]和 Theano[15]等. 这些早期的开发工具使用时主要采用脚本式编程, 通过简单配置文件的形式构造神经网络, 其有限的灵活性难以满足深度学习技术的快速发展需求.

2012 年开始, 第 2 阶段的 DL 框架开始以数据流图 (dataflow graph, 也称计算图) 为技术特征, 通过张量 (tensor) 在算子之间的流动完成模型训练的任务, 主要面向大规模产业应用. 谷歌人工智能研究团队采用静态图编程范式研发的 TensorFlow 至今仍是最流行的 DL 框架之一. Facebook 人工智能研究团队采用动态图编程范式, 在 Torch 的基础上开发出 PyTorch. 从此, TensorFlow 和 PyTorch 分别成为 DL 框架“数据流图时代”两种编程范式的代表. 同期被研发出的 DL 框架还有 Caffe[16]、Chainer[17]、Caffe2[18]、CNTK[19]、MXNet[20]和 Keras[21]等, 开发人员可以使用这些框架快速搭建复杂的 DNN 模型.

2019 年至今, 第 3 阶段的 DL 框架逐渐发展成以“动静统一, 即时编译”为技术特征, 主要面向特定领域优化. 第 3 阶段的 DL 框架普遍采用动静统一的编程范式, 可以将动态图和静态图的优点相结合, 兼顾编程的灵活性和计算的高效性. 华为面向端、边、云场景研发了 MindSpore, 通过即时编译的形式实现编译阶段中间表达式自动差分的转换. 值得注意的是, 第 3 代 DL 框架面向特定领域进行优化, 通过将特定领域语言 (domain-specific language)[22]嵌入到 Python、Java 和 C++等传统编程语言中. 例如 JAX[23]将 NumPy 的优势与硬件加速相结合, 适合高性能机器学习领域研究.

2.2 基本架构

深度学习框架架构各有不同, 但都具备一些共有的基本功能和特性. 如图 3 所示, 本文参考相关资料[24], 将 DL 框架从底层实现到应用分为 4 个层级, 由低到高分别是算子实现、计算图 (computation graph) 优化、API 调用和 DNN 模型构建. 其中, DNN 模型是 DL 框架的直接应用, 部分测试研究工作将 DNN 模型作为测试 DL 框架的输入, 验证 DL 框架实现是否存在缺陷, 因此在本节中一并进行介绍.

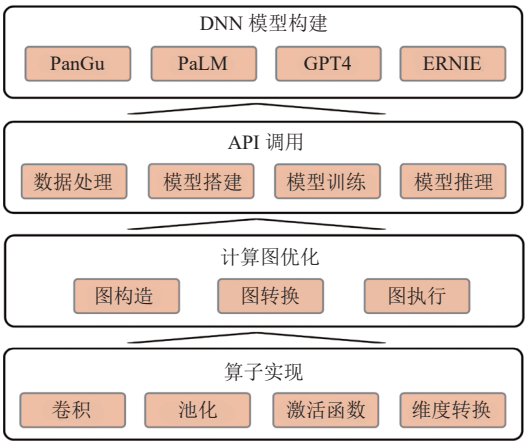

图 3 深度学习框架组成及应用架构

(1) DL 框架提供了构建 DNN 模型的工具和接口. 开发人员可以利用 DL 框架提供的 API 来自定义 DNN 模型的结构, 设置优化算法和选择损失函数等, 使模型在实际任务中得到应用. 此外, DL 框架可以将设计出的 DNN 模型转化为计算图的形式进行优化.

(2) API 是 DL 框架中一组函数、类和方法的集合, 用于定义高级的抽象关系, 可以方便地被用户直接调用来构建、训练和部署 DNN 模型. 根据 DL 任务的工作流程, 可以将 API 其分为 4 类.

1) 数据处理: 实现数据清洗、数据转换和数据归一化等功能, 使其满足 DNN 模型的相应规范.

2) 模型搭建: 构建模型结构并为模型配置一组超参数, 比如用户可以调用 Keras 中 Sequential 类 API 设计 DNN 模型的每一层.

3) 模型训练: 实现反向传播算法, 自动微分和权重更新等功能.

4) 模型推理: 帮助用户快速部署预训练的 DNN 模型, 包括预测精确度等评估指标的功能实现.

(3) 计算图是 DNN 模型在后端执行时的一种中间表示 (intermediate representation), 其中的每个节点 (node) 代表算子, 边 (edge) 代表数据. 数据输入在计算图中“流动”并调用相关基本算子完成数据转换和计算任务. 可以将计算图相关功能实现分为 3 类.

1) 图构造: 对 DNN 模型进行分析, 获取网络层之间的连接拓扑关系以及参数变量设置等信息, 将完整的模型描述编译为可被后端计算硬件调用执行的代码文本.

2) 图优化: 实现常量折叠、公共子表达式消除和算子融合等, 以提高计算性能.

3) 图执行: 根据计算图结构和计算依赖关系对算子进行调度, 对图进行切分来获取子图以进行分布式执行, 提

高计算效率.

(4) 算子是 DL 框架中最基本的组件, 例如卷积算子、池化算子等. 在计算图中, 算子将张量等参数作为输入, 用于执行特定的计算操作.

3 本文综述方法

本节内容介绍如何围绕 DL 框架测试研究展开综述, 包括论文收集、论文分析和本文整体的综述结构这 3 方面内容.

3.1 论文收集

由于 DL 框架测试是一个软件工程和人工智能交叉的研究领域, 很难直接利用少数几个关键词就得到覆盖内容较全面的论文. 为了对该研究问题做系统地调研和梳理, 本文采用关键词组合检索策略和“滚雪球”查找策略[25] (snowballing strategy) 收集 DL 框架测试研究相关论文, 具体步骤如图 4 所示.

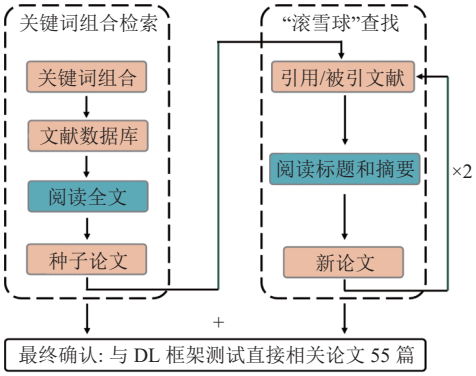

图 4 论文收集方法

本文采取关键词组合检索策略, 通过以下研究对象和研究方法关键词的相互组合在重要的文献数据库 (Web of Science、IEEE Xplore 和 arXiv) 中检索出备选论文, 之后通过阅读全文将与研究主题不相关的论文去除, 得到和研究内容直接相关的种子论文 36 篇.

① deep learning [framework|library]+[exploratory|empirical|comprehensive] study

② deep learning [framework|library]+[bug|fault|issues|problem]+understanding

③ deep learning [framework|library]+[fuzz|metamorphic|differential|mutation] testing

④ deep learning [API|operator|graph]+[fuzz|metamorphic|differential|mutation] testing

本文进一步采取两次“滚雪球”查找策略, 首先通过查阅种子论文的引用文献和被引文献, 得到更多新论文. 其次, 分别由两位同事通过阅读论文标题和摘要进行判断, 去除与研究主题不相关的论文, 当审阅意见不一致时, 通过阅读正文内容并进行讨论决定是否将该篇论文纳入综述范围. 通过重复两遍“滚雪球”策略, 共检索到 17 篇相关论文.

为使综述内容更加全面和准确, 本文在检索论文过程中, 如果在种子论文中发现高引文章, 则根据其第一作者或通讯作者追踪其相关研究内容的论文. 除国际学术会议与期刊外, 本文对《计算机学报》《软件学报》《计算机研究与发展》《计算机工程与科学》等国内计算机和软件工程领域高质量科技期刊的论文进行人工检索, 作为国内相关研究工作来源的补充. 最后, 截止到 2023 年 9 月底, 本文最终确定出与 DL 框架测试研究直接相关的论文 55 篇.

3.2 论文分析

为更好地了解深度学习框架测试研究的国内外研究现状, 本节对调研到的论文特征进行统计和分析. 如图 5

所示, DL 框架测试研究论文大多在近两年发表 (2023 年部分论文尚未公开), 说明 DL 框架测试研究尚处于初期且正处于快速发展阶段, 未来会受到更多研究人员的关注.

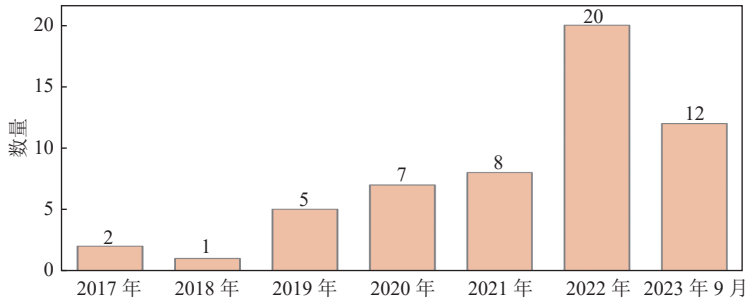

图 5　论文发表的年份分布

本文对已发表论文所属的出版物及出版物类型, 第一单位及其所在国家进行统计分析. 如图 6 所示, 与期刊相比, 会议中收录的研究工作较多, 占比超过 60%; 大部分论文均发表在由中国计算机学会评级为 A 和 B 的会议或期刊中, 具有一定的权威性. 例如, 软件工程国际会议 (international conference on software engineering, ICSE) 是软件工程领域公认的旗舰学术会议. 此外, 中国和美国是对 DL 框架研究工作最多的两个国家; 美国的伊利诺伊大学香槟分校发表论文数量最多, 其次是中国的南京大学、天津大学和北京航空航天大学等高等院校.

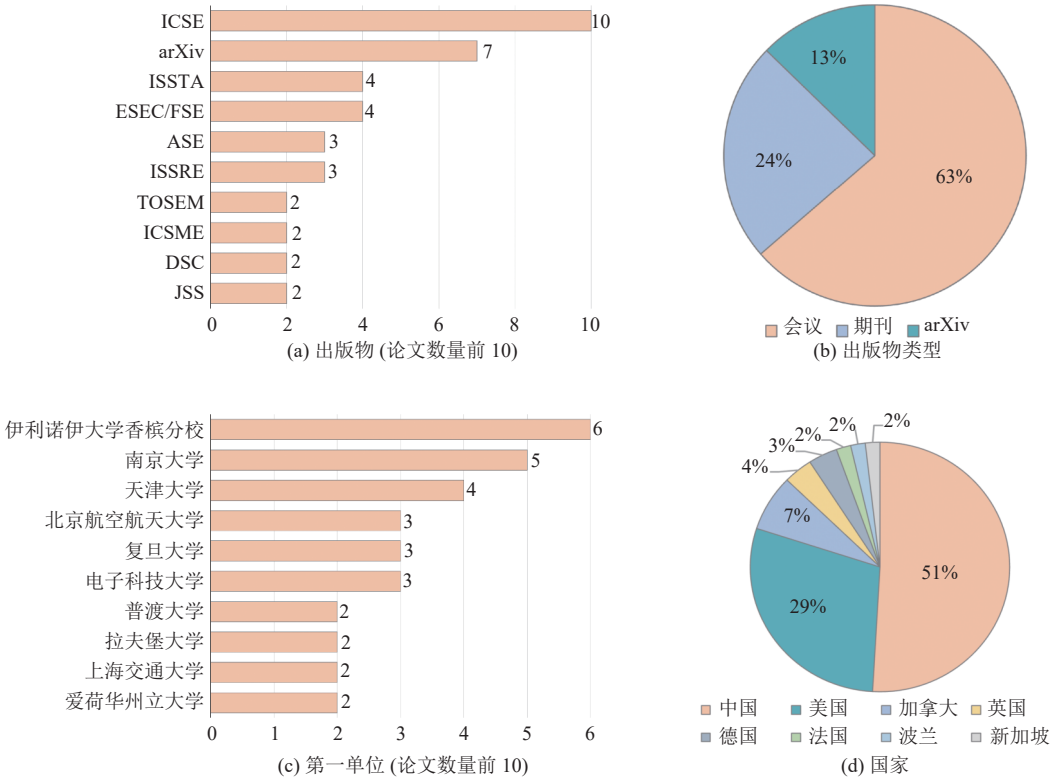

图 6　论文特征分布

3.3　综述结构

作为第 1 篇 DL 框架测试研究综述, 本文在梳理总结 DL 框架缺陷特性的基础上, 主要考虑如何高效地生成

测试输入 (测试输入生成问题), 如何判断 DL 框架的输出行为是否符合预期 (测试预言问题), 以及如何评估 DL 框架测试的有效性和充分性 (测试评估问题) 这 3 个关键问题. 本文通过梳理现有的 DL 框架研究论文, 根据不同的测试输入形式, 将现有研究工作总结为 3 类, 分别是: 以 DNN 模型为输入的系统测试, 以计算图为输入的系统测试和以特定参数为输入的组件测试. 此外, 本文在对每篇论文进行分析时重点考虑如何根据不同测试输入形式的特点, 结合具体测试关键技术来解决测试研究问题.

本文后续章节对 DL 框架测试研究做综述分析, 具体组织结构如图 7 所示. 本文第 4 节对 DL 框架缺陷特性实证研究进行总结, 主要包括实证研究方法论述和缺陷特性的总结分析两部分内容, 其中后者分为面向一般类别缺陷的整体性研究和面向特定类别缺陷的针对性研究两部分; 第 5 节主要对 DL 框架测试中重点关注的测试输入生成、测试预言和测试评估 3 个研究问题及其对应的测试关键技术进行分析; 第 6 节从测试方法的总体设计角度考虑, 根据不同的测试输入形式, 将现有研究工作进行分类、分析和总结; 第 7 节对 DL 框架未来研究方向进行展望; 第 8 节对本文全篇内容进行总结.

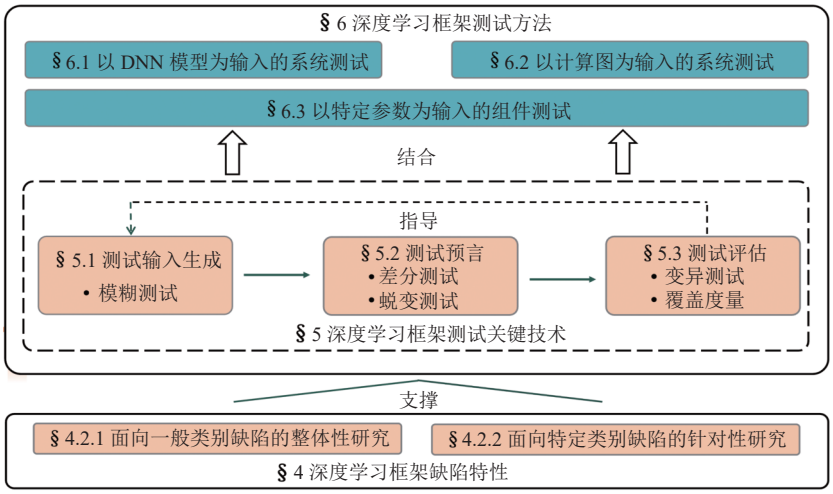

图 7 深度学习框架测试研究综述结构

4 深度学习框架缺陷特性

软件缺陷指存在于软件中的那些不希望或不可接受的偏差, 其结果是软件运行于某一特定条件时出现软件故障[26]. 软件系统往往规模庞大, 功能复杂, 在其开发和维护过程中, 不论开发和测试人员有多么丰富的经验, 都不可避免地存在缺陷. 缺陷会对软件系统的可靠性造成重大威胁, 尤其是对于安全关键软件, 一旦出现故障会造成巨大的经济和安全损失.

DL 框架作为人工智能领域中的一类软件, 同样会存在不同特征的缺陷. 只有对 DL 框架内部缺陷有足够充分的认识, 研究人员才可以设计出更有针对性的缺陷检测与定位方法. 因此, 理解缺陷特性是提高 DL 框架质量, 保证 DL 应用安全稳定的基础. DL 框架缺陷特性是本文首要关注的内容, 本节内容首先对现有工作普遍采用的实证研究方法进行论述; 其次, 本节内容将现有工作分为面向一般类别缺陷的总体性研究和面向特定类别缺陷的针对性研究两类进行总结分析. 如图 8 所示, 目前面向特定类别缺陷的研究占比 65%, 面向一般类别缺陷的研究占比 35%. 相关研究成果将有助于开发人员采取相对应的措施减少缺陷, 同时也有助于研究人员有针对性地设计缺陷检测和定位方法.

4.1 研究方法论述

实证研究 (empirical study) 具有鲜明的直接经验特征, 从大量的数据和经验事实中通过科学归纳总结出具有

普遍意义的结论或规律, 并经过严格的检验. 如图 9 所示, DL 框架缺陷特性的实证研究一般包括从缺陷数据库中抓取缺陷数据, 根据缺陷性质打标签, 对真实的缺陷数据集进行分析这 3 个步骤. 通过实证研究方法, 可以从不同方面了解 DL 框架缺陷特性, 比如缺陷产生的根本原因 (简称缺陷根因), 缺陷产生的影响, 缺陷修复模式等.

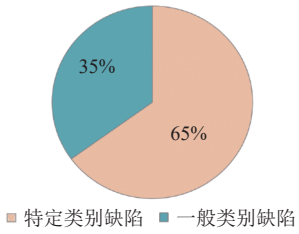

图 8 缺陷特性相关论文分布

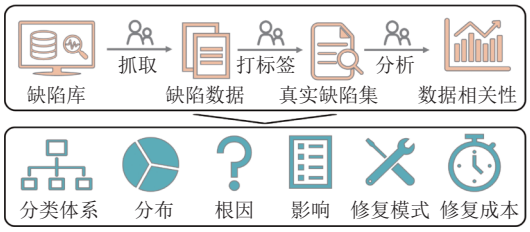

图 9 缺陷特性实证研究

(1) 抓取缺陷数据

本文通过梳理现有论文发现, DL 框架实证研究中抓取的缺陷数据来源于 GitHub[27]、Stack Overflow[28]和通用漏洞披露 (common vulnerabilities and exposures, CVE)[29]这 3 个缺陷库. 其中, GitHub 是一个面向开源及私有软件项目的托管平台, 缺陷数据是由开发者提交的, 因此缺陷数据的质量和数量都与开发者相关. Stack Overflow 是一个面向程序员的问答网站, 缺陷数据是由程序员提交的, 因此缺陷数据的质量和数量都与程序员相关. CVE 是一个公共数据库, 主要包含已知系统安全方面漏洞的详细信息, 可以帮助用户在各自独立的各种漏洞数据库中和漏洞评估工具中共享数据. 在 DL 框架的使用和测试过程中, 用户和开发人员可以通过在 GitHub 平台中提交缺陷报告的形式反馈他们所遇到的缺陷, 或将难以调试的缺陷提交到 Stack Overflow 问答社区中去讨论, 经过确认的缺陷或安全漏洞会被上传到 CVE 中. 在实证研究中, 研究人员一般通过设计网页爬虫程序来获取缺陷数据, 包括提交者对缺陷的描述, 缺陷报告的提交时间和关闭时间以及开发者和用户的讨论等.

(2) 缺陷打标分类

缺陷报告中的内容是开发人员和用户在使用软件的过程中遇到的问题, 包含了软件使用和测试过程中的重要缺陷数据. 研究人员一般通过阅读并理解开发人员和用户提交的缺陷报告, 依据一定的判断条件对缺陷特性进行判断并打标签, 完成缺陷分类.

通过人工阅读缺陷报告的方式所需时间和人力成本较高, 有部分工作研究如何对缺陷报告自动分类. Long 等人[30]为了对 DL 框架中非功能性缺陷进行研究, 在分层注意力网络 (hierarchical attention network)[31]基础上提出 MHNurf. MHNurf 可以利用 GitHub 中缺陷报告的文本内容、代码段和评论等多维度语义信息自动确认其中是否包含非功能性缺陷. 实验对比显示, MHNurf 的性能要优于传统缺陷报告分类器.

(3) 缺陷信息分析

研究人员在对缺陷分类结果进行分析总结之后, 通常会采用统计学中的相关性分析指标来评估不同分类结果之间的相关性. Du 等人[32]采用 lift function[33]分析不同类型缺陷之间, 缺陷根因和缺陷影响之间的相关性. 两个非独立类别 A 和 B 之间的 $lift$ 值可以通过公式 (1) 计算:

$$lift(A,B)=\frac{P(AB)}{P(A)P(B)} \tag{1}$$

其中, $P(A)$ 和 $P(B)$ 分别表示类别 A 和 B 的概率, $P(AB)$ 表示一个缺陷同时属于类别 A 和 B 的概率. 假设类别 A 和类别 B 分别代表某类缺陷症状和缺陷根本原因, 如果 $lift(A,B)$ 值大于 1, 则类别 A 和类别 B 是正相关的, 说明属于类别 A 的缺陷症状很有可能是由类别 B 中的缺陷根本原因造成的.

Chen 等人[24]利用斯皮尔曼秩相关系数 (Spearman's rank coefficient of correlation)[34]评估同一缺陷特性在不同框架中的分布是否具有共性. 该研究对不同缺陷根因和缺陷影响的数据分布计算斯皮尔曼秩相关系数, 计算结果均大于 0.8. 由此证明, 无论从缺陷根因还是缺陷影响来看, 4 个 DL 框架 (TensorFlow、PyTorch、MXNet 和 Deep-

Learning4J) 具有高度的共性, 因此同一种测试方法理论上是完全可以适用于不同 DL 框架的.

4.2 缺陷特性分析

研究人员采用第 4.1 节中论述的实证研究方法, 可以从缺陷报告分析出多种缺陷特性, 包括不同缺陷类型的缺陷根因、缺陷影响和缺陷修复模式等. 本文将现有工作分为面向一般类别缺陷特性的总体性研究和面向特定类别缺陷的针对性研究两类.

4.2.1 一般类别缺陷特性

一般类别缺陷是指按照一组缺陷分类依据, 对 DL 框架中的一般性缺陷进行综合性分析, 而不是对某些特定缺陷进行具体分析. 如表 1、表 2 所示, 为明确相关含义, 本文综合考虑现有工作, 将以缺陷根因和缺陷影响两种缺陷特性为分类依据的一般缺陷类别进行简要总结.

表 1 以缺陷根因为分类依据的缺陷特性总结

类别	数据失配	环境/配置	并发运行	内存	兼容性	语义
说明	数据在计算和转换过程中发生的形状或类型不匹配	依赖库、对系统功能产生不利影响的错误	在并行程序、并行线程或进程中存在的同步问题	对内存对象的不正确处理造成的错误	程序无法在指定的计算硬件、操作系统或网页浏览器等环境中正常运行	与软件需求或开发人员意图不一致

表 2 以缺陷影响为分类依据的缺陷特性总结

类别	崩溃/异常	挂起	操作失败	功能错误	性能退化	警告式错误
说明	软件意外停止运行并输出错误信息	程序长时间运行中对输入没有响应	编译失败、未完成任务、拒绝服务、重复执行同一任务等意外行为	程序在运行或编译时没有出现任何报错信息, 但程序没有按照设计的功能运行	程序没有在预期的时间内输出结果时, 会发生性能下降	程序的运行不会受到干扰, 但仍需要消除该错误以消除风险或提高代码质量

各项实证研究工作的具体研究对象不同, 缺陷分类依据不同, 重点研究的缺陷特性也不完全相同, 在明确基本缺陷特性的基础上, 本文对目前一般类别缺陷特性实证研究工作进行总结, 具体信息如表 3 所示.

表 3 一般类别缺陷特性实证研究

文献	缺陷信息来源	具体研究对象	缺陷特性		
			根因	影响	修复模式
Chen等人[24]	GitHub	TensorFlow, PyTorch, MXNet, DeepLearning4J	算法实现错误, 类型混乱, 配置错误等	崩溃, 挂起, 功能错误等	—
Islam等人[35]	GitHub, Stack Overflow	TensorFlow, Theano, Torch, Caffe, Keras	模型参数错误, 性能不佳, 张量未对齐等	崩溃, 性能不佳, 功能错误等	—
Du等人[37]	GitHub	TensorFlow	环境配置, 并发, 内存等	—	—
Jia等人[36]	GitHub	TensorFlow	维度失配, 类型混乱, 预处理错误等	功能错误, 崩溃, 挂起等	修改参数, 函数替代, 值检查等
Yang等人[38]	GitHub	TensorFlow, Theano, PyTorch, Caffe, Keras, MXNet, CNTK, DeepLearning4J	数值精确度, 特殊类型数值, 算法实现错误等	崩溃, 效率低, 性能差等	—
Quan等人[39]	GitHub	TensorFlow.js	编程错误, 配置/依赖错误等	崩溃, 配置/初始化失败, 性能不佳等	调整API调用顺序, 修改参数, 更改版本
Du等人[32]	GitHub	TensorFlow, MXNet, PaddlePaddle	环境/配置错误, 并发, 内存等	崩溃, 挂起, 操作失败等	—

注: “—”表示该工作未涉及相关内容

Chen 等人[24]根据功能不同, 将 DL 框架的架构分为 5 个层级, 并在这 5 个层级中对 DL 框架的缺陷特性分别研究. 这项工作总结出 13 个缺陷根因. 数据类型混乱, 张量形状失配, 深度学习算法实现错误和运行环境不兼容

这 4 个缺陷根因是 DL 框架区别于其他软件系统特有的. 其中深度学习算法实现错误是最主要的缺陷根因. DL 框架的不同层级具有不同的缺陷分布特征, 其中算子执行这部分包含的缺陷数量最多. 和 Islam 等人[35]的研究结果类似, 这项工作同样发现崩溃 (crash) 是存在最普遍的缺陷症状, 占比达 48.25%. 此外, 作者为验证该研究结果的对框架测试研究的指导意义, 开发出原型 DL 框架测试工具 TenFuzz. TenFuzz 通过对 TensorFlow 中单元测试组件进行突变生成测试输入, 并交叉验证不同版本 TensorFlow 对应的输出从而实现差分测试, 成功检测到 6 个真实缺陷.

Islam 等人[35]对 5 个 DL 框架进行了研究, 包括 Caffe、Keras、TensorFlow、Theano 和 Torch. 在该研究中, 作者分析了来自 Stack Overflow 的 2 716 个帖子和来自 GitHub 的 500 个缺陷修复提交, 以探索缺陷产生的原因和造成的影响. 在该研究中, 作者总结出 6 种缺陷根因和 6 种缺陷影响. 其中, 崩溃在 5 个 DL 框架的缺陷影响中均占比最高. 此外, 作者对比发现, 除 API 误用以外, 其他缺陷根因在 Stack Overflow 和 GitHub 两个平台中的分布情况基本一致.

Jia 等人[36]对 TensorFlow 中的缺陷进行了研究. 他们分析了在 2017 年 12 月–2019 年 3 月之间的 202 个 TensorFlow 的缺陷修复提交, 其中 84 个有对应的缺陷报告. 在这项工作中, 作者对缺陷根因、缺陷影响以及不同组件中缺陷的比例进行了分析, 并通过 lift function 评估不同现象和根因之间的相关性强弱. 值得注意的是, 该研究发现在 TensorFlow 框架本身的缺陷中, 算法或接口实现上的错误最为常见, 占全部缺陷的 38.21%.

Yang 等人[38]在掌握 DL 框架层级特征的基础上, 构建出一套缺陷分类体系, 从源代码着手深入探究 DL 框架中的缺陷根因和缺陷现象, 并提出可行的解决方案. Yang 等人对来源于 8 个 DL 框架的 1 127 个缺陷报告进行分析, 发现框架中最主要的缺陷根因是算法实现错误和缺少条件检查两种. 从缺陷现象来看, 和基于 DL 的软件不同, DL 框架本身由于缺陷而出现的挂起和效率低现象极少, 更多的是输出错误和崩溃. 此外, 作者从 343 个缺陷补丁中总结出 15 种缺陷修复模式, 比如在代码库中增加张量形状和数据类型方面的检查等.

以往工作大多选择对基于 Python 的 DL 框架进行研究, 而忽略了对基于其他编程语言的框架进行研究的工作. Quan 等人[39]选择基于 JavaScript 的 DL 框架 TensorFlow.js[40]进行研究, 分析其中的缺陷特性. 采用 TensorFlow.js 开发的模型可以在浏览器上直接运行, 对平台和设备的依赖性低. 作者从现象、根因和修复模式这 3 方面对 GitHub 中与 TensorFlow.js 相关的缺陷进行分析. 研究发现: 崩溃是最普遍存在的缺陷现象, 其次是性能劣化; 程序编写错误是占比最高 (51%) 的缺陷根因; 更改框架或第三方库的版本是最普遍缺陷修复方式, 可以解决大部分缺陷.

上述工作都是从根本原因以及现象等方面对缺陷进行分析, Du 等人[32,37]首次基于缺陷触发和错误传播条件对深度学习系统中环境依赖缺陷特性进行了研究. 该研究参考传统软件缺陷分类体系[41], 将真实缺陷分为波尔缺陷 (Bohrbug)、老化缺陷 (aging-related bug) 和非老化曼德博缺陷 (non-aging-related Mandelbug) 这 3 个主要类别. 其中波尔缺陷是一类触发和错误传播条件都比较简单的缺陷, 因此这类缺陷比较容易被复现. 而曼德博缺陷的触发和失效的发生之间存在一定的时间滞后, 或者受到一些间接因素的影响, 导致其触发或错误的传播十分复杂. 该研究发现, 在 3 个 DL 框架中, 超过 2/3 的真实缺陷是波尔缺陷, 在所有缺陷类型中占比最高. 波尔缺陷最主要的根因是语义错误, 老化缺陷最主要的根因在于内存方面的问题. 在缺陷影响方面, 一半以上的波尔缺陷和曼德博缺陷会造成程序崩溃或者异常. 修复曼德博缺陷耗费的时间比修复波尔缺陷的时间更长.

4.2.2 特定类别缺陷特性

如表 4 所示, 除一般类别缺陷特性实证研究, 部分工作针对某些特殊子类的缺陷特性进行更加细致的研究.

Cao 等人[42]对基于 TensorFlow 和 Keras 开发的深度学习系统中的性能问题进行研究. 在 API 误用, 模型配置错误, DL 框架, 数据和计算硬件这 5 种性能缺陷的根因类型中, DL 框架中包含的缺陷占比 10.7%, 其中包括某个框架版本存在缺陷和某个框架版本与硬件不匹配两类. 比如, TensorFlow 1.X 版本在 CUDA 11.1 上得不到完全支持, 会导致启动模型训练过程的时间较长.

Makkouk 等人[43]针对 Tensorflow 和 PyTorch 中的性能缺陷 (performance bug) 进行实证研究, 具体从性能缺陷数量占比的变化趋势, 性能缺陷和非性能缺陷在缺陷复杂程度等方面的区别, 性能缺陷的根因这 3 方面进行分

析. 研究发现, 2016–2021 年, 性能缺陷在所有缺陷中的占比一直在增加, 说明性能缺陷表现出的问题日益严重. 相比非性能缺陷, 性能缺陷往往更加复杂, 修复难度更大, 需要多花费 94.46% 的时间, 多修改 35.71% 的代码量. 此外, 研究发现内存使用效率低是导致 DL 框架性能问题最主要的根因, 具体包括内存资源未及时释放和内存分配错误两方面. 作者进一步分析得知, DL 框架更容易发生内存相关的性能缺陷主要与底层编程语言的不合理使用和运行时框架内部组件和外部资源之间频繁的数据搬运这两方面有关.

表 4　特定类别缺陷特性实证研究

文献	缺陷信息来源	具体研究对象	缺陷类别
Cao等人[42]	GitHub、Stack Overflow	TensorFlow、Keras	性能缺陷
Makkouk等人[43]	GitHub	TensorFlow、PyTorch	性能缺陷
Long等人[44]	GitHub	TensorFlow、Keras、PyTorch	性能缺陷、精度缺陷
Kloberdanz等人[45]	GitHub	TensorFlow、PyTorch	数值不稳定缺陷
Tambon等人[46]	GitHub	TensorFlow、Keras	静默缺陷
Ren等人[47]	GitHub	TensorFlow、Torch7、Caffe2、PyTorch、OpenCV、Theano、Keras、Chainer、CNTK、MXNet	系统资源缺陷
Liu等人[48]	GitHub	TensorFlow、MXNet、PaddlePaddle、MindSpore	老化缺陷
Huang等人[49]	Stack Overflow	TensorFlow、Keras、PyTorch	依赖缺陷
Xiao等人[50]	CVE	TensorFlow、Torch7、Caffe	安全漏洞
Chen等人[51]	CVE	TensorFlow、Torch、Caffe	安全漏洞
Harzevili等人[52]	CWE	TensorFlow、PyTorch、Scikit-learn	安全漏洞
Filus等人[53]	CWE、GitHub	TensorFlow	安全漏洞

Long 等人[44]针对性能缺陷和精度缺陷 (accuracy bug) 进行实证研究. 该研究发现包含真实缺陷的缺陷报告实际上占比很低; 接近一半的缺陷是由开发人员在日常维护更新过程中检测到的, 而不是基于用户提交的缺陷报告发现的. 说明目前有效缺陷报告少, 而且框架的开发维护人员对缺陷报告利用率很低. 在缺陷报告中自动地确认是否存在真实的缺陷, 并可以识别哪些缺陷值得修正同样是很重要的工作.

在深度学习算法中存在大量数值计算, 确保框架的鲁棒性和可靠性最大的挑战就是数值稳定问题 (numerically stable problem). 数值不稳定缺陷很难直接表现出来, 往往会随着模型训练过程而逐渐积累最后影响模型的性能. 为此, Kloberdanz 等人[45]对 DL 框架中的数值不稳定缺陷特性作了实证研究, 通过深入分析 GitHub 缺陷库中 252 个与数值稳定性相关的提交, 总结其不稳定模式, 影响和解决方法等, 归纳出第 1 个深度学习领域数值不稳定缺陷和解决方案的数据库——DeepStability. 研究发现, 由数值不稳定导致的缺陷主要有数值上溢出 (占比 47%), 数值精度损失 (占比 34%) 和数值下溢出 (占比 16%) 这 3 种, 其中数值精度损失会导致其网络权重和偏置更新不准确, 从而导致模型训练过程劣化. 该研究指出可以采取修改数学计算公式, 增加数值精度或改变变量类型, 使用不同的算法, 限制输入范围等方式缓解数值不稳定问题.

Tambon 等人[46]针对静默缺陷 (silent bug) 进行实证研究, 这种缺陷通常会导致错误的计算结果或行为, 但不会表现出崩溃或挂起等明显的症状, 所以很难被发现. 作者根据缺陷影响将 TensorFlow 和 Keras 里的 77 个可复现的静默缺陷分成 7 类, 研究发现计算错误占比最高, DL 模型计算的某一步存在缺陷可能会导致接下来每一步都存在错误的结果, 并逐渐累积.

系统相关的缺陷往往会导致性能退化, 安全性缺乏保障和资源不合理利用等, 是决定 DL 框架可靠性和用户体验的关键. Ren 等人[47]专门对 DL 框架和传统软件系统中系统相关的缺陷 (system-related bug) 进行了对比实证研究. 他们发现内存分配不当, 内存泄漏, 多线程错误和性能退化是出现频率最高的 4 类问题, 而配置错误在传统软件系统中非常普遍, 在 DL 框架中则很少发生. 此外, 深度框架维护人员和用户在讨论 API 失配问题上消耗的时

间最多, 需要更多的信息为解决问题提供参考.

Liu 等人[48]针对 4 个 DL 框架 (TensorFlow、MXNet、PaddlePaddle 和 MindSpore) 中的老化缺陷进行了实证研究, 老化缺陷是一种随着运行时间增加, 误差逐渐累积, 进而表现出性能退化, 系统响应变慢等现象的曼德博缺陷 (Mandelbug). 相比于其他类型的缺陷, 老化缺陷的失效机理更加复杂, 因此检测和修复老化缺陷的难度也更大. 研究发现, 79% 的老化缺陷与内存问题有关, 其中内存泄漏问题是造成老化缺陷最主要的原因. 此外, 相比传统软件系统, 数值相关的老化缺陷在 DL 框架所有老化缺陷中占比更高. 这是由于深度学习是数据驱动的学习模式, 在模型运行过程会存在大量高维度数值计算过程, 容易将 DL 框架中的数值误差累积, 最后造成老化缺陷.

DL 应用易受到 DL 系统 (包括计算硬件、操作系统、驱动程序、运行环境和 DL 框架) 中依赖缺陷 (dependency bug) 的影响. Huang 等人[49]针对 DL 系统中依赖缺陷的根因、影响和修复模式进行研究, 作者发现 DL 框架是引入依赖缺陷占比最多 (占比 77.5%) 的层级, 其中 Keras (占比 15.6%)、TensorFlow (占比 51.2%) 和 PyTorch (占比 3.7%) 是最容易发生依赖缺陷的 DL 框架, 最主要的原因是 DL 框架与更底层的软件系统和硬件不兼容.

软件系统中的安全漏洞 (security vulnerability) 是指系统在设计、实施和维护过程中的一类特殊缺陷, 可能会被攻击者利用来破坏系统, 窃取数据或执行恶意代码[54]. 与其他类型的软件缺陷不同, 安全漏洞大多涉及软件系统的安全性和保护机制. 例如一个程序如果存在身份验证不足的问题, 可能导致黑客可以利用这个漏洞来攻击系统. 这个程序本身不存在功能实现上的缺陷, 但从系统安全的角度分析, 身份验证不足违反了系统安全准则, 因此是一个安全漏洞. 因此, 开发人员需要采取额外的措施来确保软件系统的安全性, 对安全问题进行深入的测试和审查. DL 框架简化了 DL 应用的设计和开发难度, 但也为用户掩盖了它所使用的组件依赖, 带来很多隐蔽的安全性问题. 每种 DL 框架都是实现在众多外部基础库和组件之上, 例如 TensorFlow 中包含多达 97 个 Python 模块, 任何在 DL 框架以及它所依赖的组件中的安全漏洞都会威胁到框架之上的应用.

360 安全研究院在 2017 年互联网安全大会人工智能技术应用安全论坛中展示出 6 个安全攻击案例, 包含框架依赖库中缺陷导致的堆溢出 (heap overflow), 整数溢出 (integer overflow), 崩溃和拒绝服务 (denial-of-service) 等安全漏洞. 例如由于 Numpy 科学计算中的 pad 函数存在代码逻辑错误导致基于 TensorFlow 的语音识别应用出现拒绝服务的安全漏洞[50].

Chen 等人[51]认为 DL 框架中出现的安全漏洞一般在于两方面: 攻击者利用 DNN 运行机制反过来生成对抗样本; DL 框架所依赖的外部库中存在缺陷. 这两种情况都有可能导致在 DL 框架基础的应用存在安全漏洞. 作者在调研分析 CVE 中 DL 框架漏洞的过程中发现, 这些漏洞往往会被攻击者挖掘并进行攻击. 为了更好认识框架的安全问题, 作者从攻击阶段、对抗知识、攻击频率、攻击目标和攻击范围 5 个方面将 DL 框架的攻击方法进行分类, 同时对具体的攻击和防御方法建立映射关系, 为相关领域人员提供参考. 例如相关人员可以采用大样本剔除异常值的方法[55]来防御数据投毒攻击 (data poisoning attack)[56].

Wang 等人[52]从类型、根因、现象、修复模式和修复工作量这 5 个方面对收集到的 596 个机器学习框架安全漏洞进行分析和研究. 作者在 596 个漏洞涉及的 19 个通用漏洞枚举 (common weakness enumeration, CWE)[57]类别基础上, 总结出 5 种安全漏洞类别. 其中, 数值和内存相关的漏洞最为普遍. 研究发现, 缺少对张量性质的检查是产生数据类型错误和内存错误一个普遍性原因, 开发人员可以通过添加张量性质检查器来修复这方面的漏洞. 同时, 作者将安全漏洞数据集公开并开发出 DeepMut 工具来印证实证研究中的发现和结论.

传统的静态代码分析工具在分析 DL 框架中更加深度的语义信息和数学方法方面能力有限, 难以检测出 DL 框架安全漏洞. Filus 等人[53]采用正交缺陷分类法 (orthogonal defect classification), 通过将语义信息转化为评估指标, 对缺陷进行分类. 作者对从 CWE 收集到的 104 例 TensorFlow 安全漏洞进行分析, 并将其分为越界读取数据、越界写入数据、空指针、不合适的输入校验和整数溢出 6 类, 分别从安全漏洞的严重程度和安全漏洞对 TensorFlow 机密性、完整性和可用性的影响进行重点分析. 研究发现大部分内存相关的安全漏洞是由于缺少或者不正确的检查语句导致的, 危害程度也最高. 此外, 这项工作总结出 TensorFlow 检查项列表方便 TensorFlow 用户和开发者进行代码审查, 消除可能存在的编程错误.

区别于以上直接对深度学习框架的缺陷特性进行研究, 还有部分研究通过对基于框架的上层应用中出现的问

题进行分析, 获得一些关于框架中缺陷特性相关的经验知识. Sun 等人[58]通过对 329 个来自于 GitHub 的缺陷报告进行分析, 将基于上述框架的应用程序缺陷分为 7 类, 分析出 12 种缺陷的修复模式, 并对修复时间等进行评估. 研究发现, 修复框架缺陷时, 采用 if 模式最为普遍, 主要包括添加 if 模块, 更改 if 判断条件两种. 在缺陷修复需要的时间方面, 大部分缺陷 (68.39%) 可以由开发人员在 1 个月之内解决, 40.73% 的缺陷可以在 1 周之内解决. Zhang 等人[59]通过构建问题自动分类的技术, 对 Stack Overflow 中涉及深度学习应用的问题和回答进行研究. 结果显示, 崩溃和模型迁移是 TensorFlow 出现最多的两方面问题, 根据收到回答的时间和答案质量来看, 性能相关的问题是最难回答的问题. Liu 等人[60]对 DL 框架中的技术债务进行分析, 技术债务是一种框架开发阶段为缩短开发时间而牺牲框架质量遗留的隐性问题. 研究发现, 在 TensorFlow 等 7 个 DL 框架中, 需求债 (占比 7.09%–31.48%) 会导致功能实现错误, 算法债 (占比 5.62%–20.67%) 会导致系统性能下降. 兼容性债是对其他项目的不成熟依赖, 在 Keras 和 Caffe 中, 兼容债占比超过 10%.

5 深度学习框架测试关键技术

在理解 DL 框架缺陷特性的基础上, 如何高效地生成测试输入, 如何判断 DL 框架的输出行为是否符合预期, 如何评估测试的质量和效果是 DL 框架测试流程中主要关注的重点问题, 也是 DL 框架测试面临的 3 个挑战. 针对这 3 个研究问题, 本文对调研到的测试关键技术进行分析和总结.

5.1 测试输入生成

测试输入生成 (test input generation) 是 DL 框架测试研究首先要面对的挑战. 测试输入生成, 即在程序测试过程中, 按照一定的算法或策略自动搜索输入空间, 不断生成测试输入, 以满足不同的测试目标. 面对规模庞大, 设计逻辑复杂的软件系统, 通过人工的方式生成测试输入存在效率较低, 成本高等问题. 尤其是在深度学习领域, 输入空间维度高且十分庞大, 几乎不可能通过人工方式来生成足够的测试输入.

经过调研, 目前具备程序测试输入自动生成能力的测试技术主要有模糊测试、随机测试和蜕变测试等, 其中模糊测试的思想在 DL 框架测试输入生成中得到广泛应用. 模糊测试 (fuzz testing) 是一种自动或半自动地生成非预期输入来测试软件系统, 通过监视程序异常 (崩溃或断言失败等) 来发现软件系统缺陷和安全漏洞的软件测试方法[61]. 因为模糊测试是不合逻辑的, 只是产生杂乱数据攻击程序, 更可能发现其他测试方法很难发现的安全漏洞. 模糊测试的核心在于测试输入生成策略, 即生成能触发程序异常的输入. 本文根据测试输入生成方式的不同将模糊测试分为基于生成的模糊测试 (generation-based fuzz testing) 和基于突变的模糊测试 (mutation-based fuzz testing)[62].

如何运用模糊测试, 针对 DL 框架的特点生成测试输入是本文在调研分析过程中重点关注的研究问题.

5.1.1 基于生成的模糊测试

基于生成的模糊测试是基于特定的规则直接生成满足一定约束条件的测试输入, 并在不断迭代的模糊过程中更新测试输入库的模糊测试技术. 基于生成的模糊测试优势在于生成的测试输入大多符合被测试程序的要求, 可以直接运用到测试执行中, 因此测试效率较高. 基于生成的模糊测试通常包括 4 个步骤: 定义或构造输入规则, 生成初始种子输入, 执行测试和监测输出行为. 其中第 1 步是基于生成的模糊测试方法核心步骤, 研究人员可以根据输入特性直接定义输入生成规则, 也可以是通过算法推理构造出输入生成规则.

在以 DNN 模型为输入的系统测试方法中, 要求模型内部结构满足一定的约束条件. 例如, 在构建卷积神经网络时, 需要确保卷积核通道数和输入图像通道数相匹配, 以确保卷积运算可以正确地进行. NeuRI[63]利用归纳程序综合的方法推理出可以构造有效模型的规则, 在这组规则下完成模型的生成. 在以特定参数为输入的组件测试中, 测试输入需要满足的约束主要包括张量维度、张量形状、张量类型、参数类型、参数数值和参数依赖关系等. 如图 10 所示, DocTer[64]利用自然语言处理 (natural language processing, NLP) 的方法, 对官方文档中关于 API 参数约束的自然语言描述进行依存关系分析 (dependency parsing) 生成一组语法规则, 从而提取出具体的 API 参数约束. 在一组约束条件下, DocTer 可以生成有效参数输入和边界值参数输入完成对 API 的缺陷检测.

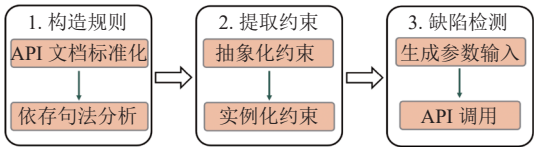

图 10 DocTer 测试输入生成技术

5.1.2 基于突变的模糊测试

基于突变的模糊测试对已有的种子输入实施突变来生成数量更多, 分布更加全面的测试输入. 基于突变的模糊测试不需要预定义复杂的输入约束条件, 而是在已知有效输入的基础上生成更接近实际情况的输入, 相比于基于生成的模糊测试更容易实现, 可以降低测试策略设计和实施的难度. 基于突变的模糊测试主要包括选择种子输入, 设计并实施突变规则, 执行测试, 优化更新种子输入库和突变规则和监测输出行为等步骤. 其中突变规则的设计和实施是基于突变的模糊测试方法核心步骤, 一般需要具体根据实施对象的特点进行分析.

在以 DNN 模型为输入的系统测试中, 为获得多样化的模型, 往往以 DNN 模型结构中的层和连接权重为突变规则的实施对象. LEMON[65]共设计 7 种层间的突变规则和 5 种层内突变规则, 例如移除 DNN 模型中的某一层; 对相邻层之间的权重添加噪声, 实施高斯模糊突变等. 在以特定参数为输入的组件测试中, 主要对 API 或算子中的参数设计并实施突变规则. 如图 11 所示, FreeFuzz[66]针对 API 中的参数类型和参数数值等性质实施突变来生成更多样性的参数输入, 以检测被测 API 对于不同类型、不同范围和不同长度参数的处理能力.

模糊测试不是基于确定逻辑设计的测试方法, 因此需要大量的时间和计算资源来生成和执行大量的测试用例, 并对测试结果进行分析和归纳. 部分 DL 框架测试研究工作采用灰盒模糊测试的思想, 在测试迭代的过程中, 利用某些中间信息 (覆盖度量指标或输出不一致程度) 指导测试输入生成和测试输入选择, 有效提高模糊测试的效率和发掘一般性缺陷和安全漏洞的能力.

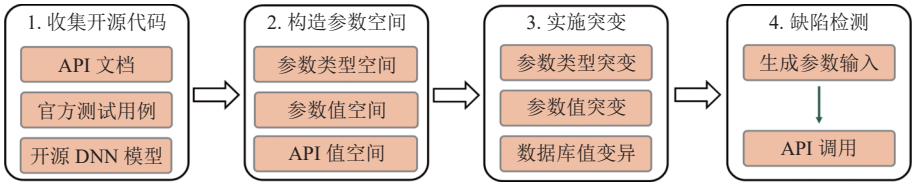

图 11 FreeFuzz 测试输入生成技术

5.2 测试预言

测试预言 (test oracle) 是一种判断待测程序在给定测试输入下的执行结果是否符合预期的机制[67]. 测试预言难以明确是软件系统测试中普遍存在的难题, 它导致测试人员只能选择一些可以预知结果的有限测试用例进行测试, 而不能完整有效地进行测试.

深度学习领域存在很多不确定的因素, 比如具体实现方式和运行环境的差异, 算法的随机性, 浮点数值计算误差等. 不确定因素将导致相同功能实现的两个 DL 框架测试用例即使在相同的输入下, 其输出也会不可避免地存在差异, 因此难以区分是框架内的缺陷还是不确定因素导致的问题. 研究人员测试 DL 框架时可以采用预言近似 (oracle approximation) 的方式缓解不确定因素带来的测试预言问题, 判断实际输出和预言之间的差异是否在容忍误差之内, 而不是要求待测代码的输出与预期完全相等. 预言近似问题定义如公式 (2) 所示, 其中 *tolerance* 为容忍误差:

$$|output - oracle| \leqslant tolerance \tag{2}$$

Nejadgholi 等人[68]对 DL 框架内出现的预言近似断言 (oracle approximation assertion) 进行实证研究, 根据 DL 框架中的源代码对测试预言和容许误差类型进行分析总结. 作者发现 TensorFlow 中 25% 的断言采用了预言近似

断言的方法对代码段进行测试, 其中 73% 的预言类型和 65% 的容许误差都是开发人员根据代码特征自定义的. 为适应 TensorFlow 版本演化, 开发人员还会不断地修改 API 及其对应的测试预言和容许误差阈值. 测试预言近似的方法只能对测试预言问题起到缓解作用, 普适性较差.

如何在 DL 框架测试中解决或缓解测试预言问题是本文在调研分析过程中重点关注的研究. 经过调研发现, 在 DL 框架测试研究中, 研究人员开始参考缓解传统软件测试预言难的技术, 比如蜕变测试和差分测试.

5.2.1 蜕变测试

蜕变测试 (metamorphic testing) 是一种主要缓解测试预言问题的黑盒测试方法, 主要依据被测软件系统的领域知识和实现方法建立蜕变关系 (metamorphic relation), 利用蜕变关系来生成新的测试用例, 通过验证蜕变关系是否被保持来决定测试是否通过[69]. 例如, 测试 TensorFlow 中深度学习算子 tf.math.sigmoid 时, 当测试输入为 1, 直接确定 *Sigmoid* 函数的输出是否正确是十分困难的, 但是可以利用其数学属性辅助测试该算子. 例如, 已知 *Sigmoid* 函数的性质: $Sigmoid(x)+Sigmoid(-x)=1$, 根据测试输入 1 生成另一个输入为−1, 通过测试 $Sigmoid(1)$ 与 $Sigmoid(-1)$ 相加是否等于 1 判断该算子实现是否存在缺陷. 如果 $Sigmoid(1)+Sigmoid(-1)=1$, 则这种蜕变关系得到保持, 一定程度上说明深度学习算子 tf.math.sigmoid 功能正常.

近年来, 蜕变测试也被应用到 DL 框架的测试中. Ding 等人[70]通过设计系统级、数据集级和数据元素级 3 种不同层级的蜕变关系来产生新的测试数据, 从而验证测试 DL 任务过程中 DL 框架的功能正确性. 具体来讲, 这项工作通过分析 DL 图像分类任务, 重组训练数据集和生成新图像这 3 种方式对分类准确率的影响来验证 3 种层级的蜕变关系是否得到保持. 例如, 在训练数据集中的每个类别中加入 10% 的新图像后, 其分类准确率不应该受到影响. 最后, 作者将蜕变关系实施到用于生物细胞图像自动分类的 DL 分类器中, 成功证明该方法的有效性. FreeFuzz[66] 在针对 DL 框架中的 API 进行测试时, 参考数值精度和运行速度之间的蜕变关系, 比如采用 float16 数值精度的 API, 其运行速度应该比采用 float32 数值精度的 API 快. 通过这种方式, FreeFuzz 成功检测到 DL 框架性能方面的缺陷.

5.2.2 差分测试

给定相同的测试输入, 相同功能实现的软件系统应输出相同的结果. 利用这一点, 差分测试 (differential testing) 方法通过观测两个相同功能实现的软件系统或程序在相同输入下的输出是否存在差异来检测可能存在的缺陷[71]. 由于 DL 框架的开源特性, 研究人员可以方便地实现功能等价的实例. 这种功能等价的实例可以是不同框架下相同 DNN 模型的实现, 同一框架下相同计算图的实现, 同一框架下相同 API 的实现和同一 API 在不同计算硬件上的实现等. 差分测试更适于检测代码逻辑错误和数值误差等语义缺陷, 是解决测试预言问题最有效的方法之一, 已经在 DNN 模型测试, DL 框架测试和 DL 编译器测试等领域得到广泛应用.

Pham 等人[72]开创性地从差分测试的角度对 DL 框架展开研究, 提出测试方法 CRADLE. CRADLE 采用不同 DL 框架实现相同结构的 DNN 模型, 通过检测模型输出的不一致程度, 逐步定位框架内存在的缺陷. 值得注意的是, 输出不一致程度评估指标在以 DNN 模型为输入的系统测试方法中, 为实现差分测试提供重要参考. 之后部分研究工作将 CRADLE 中提出的不一致程度指标作为参考, 有助于实现以 DNN 模型为输入的自动化测试, 发掘更多的缺陷. 以下是 CRADLE 中评估输出不一致程度的关键指标, 主要包括输出层间距、隐藏层间距和隐藏层间距变化率.

输出层间距 *D_CLASS* 可以评估用于分类任务的 DNN 模型在两个不同 DL 框架后端实现下的预测结果差异, 具体定义如公式 (3) 和公式 (4) 所示. 公式 (3) 中 C 为真值 (ground-truth) 标签, $Y=[y_1,y_2,y_3,\ldots,y_N]$ 为表示模型预测结果的 N 维向量, $rank_{C,Y}$ 为真值标签 C 出现在输出向量 Y 中的位次, k 默认为 5. 公式 (4) 中的 Y^1 和 Y^2 分别为相同 DNN 模型在以 DL 框架 1 和 DL 框架 2 为后端实现时的模型预测结果向量.

$$\sigma_{C,Y}=\begin{cases}2^{k-rank_{C,Y}}, & rank_{C,Y}<k\\ 0, & rank_{C,Y}\geqslant k\end{cases} \tag{3}$$

$$D\_CLASS_{C,Y^1,Y^2}=\left|\sigma_{C,Y^1}-\sigma_{C,Y^2}\right| \tag{4}$$

此外, CRADLE 基于平均绝对偏差 (mean absolute deviation) 距离设计同时适用于回归模型和分类模型的输出层间距指标 D_MAD. 如公式 (5) 和公式 (6) 所示, $G=[g_1,g_2,g_3,\ldots,g_N]$ 为真值标签组成的向量, $Y=[y_1,y_2,y_3,\ldots,y_N]$ 为表示模型预测结果的 N 维向量, Y^1 和 Y^2 分别为相同 DNN 模型在以 DL 框架 1 和 DL 框架 2 为后端实现时的模型预测结果向量.

$$\delta_{Y,G}=\frac{1}{N}\sum_{i=1}^{N}|y_i-g_i| \tag{5}$$

$$D\_MAD_{G,Y^1,Y^2}=\frac{\left|\delta_{Y^1,G}-\delta_{Y^2,G}\right|}{\delta_{Y^1,G}+\delta_{Y^2,G}} \tag{6}$$

如公式 (7)–公式 (9) 所示, CRADLE 提出隐藏层间距 δ 和隐藏层间距变化率 R, 在差分测试检测到不一致输出基础上, 对 DNN 模型隐藏层输出向量进行分析, 从而实现缺陷定位. 公式 (7) 中, $\delta_{S_l^1,S_l^2}$ 为相同模型在 DL 框架 1 和 DL 框架 2 为后端实现下, 在同一隐藏层 l 输出 N 维向量的平均绝对偏差. 公式 (8) 中, δ_{pre} 为 DNN 模型中位于隐藏层 L 之前且与 L 相连接层的 $\delta_{S_l^1,S_l^2}$ 最大值. 公式 (9) 中, 隐藏层间距变化率 R_L 定义为在某一隐藏层 L 处, $\delta_{S_l^1,S_l^2}$ 相较于 δ_{pre} 数值的变化率, τ 取 10^{-7}.

$$\delta_{S_l^1,S_l^2}=\frac{1}{N}\sum_{i=1}^{N}\left|S_l^1-S_l^2\right| \tag{7}$$

$$\delta_{\text{pre}}=\max_{l\in pre(L)}(\delta_{S_l^1,S_l^2}) \tag{8}$$

$$R_L=\frac{\delta_{S_l^1,S_l^2}-\delta_{\text{pre}}}{\delta_{\text{pre}}+\tau} \tag{9}$$

除构建相同模型作为差分测试的对象外, 也有研究针对 DL 框架中的计算图展开差分测试. 如图 12 所示, EAGLE[73]这项工作通过构建两个逻辑功能等价的计算图, 观测其输出是否一致来检测框架中可能存在的缺陷.

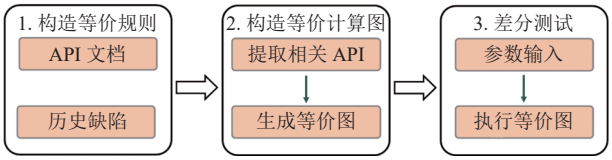

图 12 EAGLE 测试预言

差分测试的广泛应用有助于缓解测试预言问题, 但也面临新的挑战, 即 DL 框架的开发人员需要不断地了解第三方库在版本更新中的代码更改情况来评估其对框架内部差分测试用例带来的影响. Prochnow 等人[74]提出 DiffWatch, 这种方法可以自动检测 DL 框架中的差分测试用例, 并将其分为有断言的测试用例和有预言的测试用例两类; 然后监测外部库代码更新可能给差分测试带来的影响, 为开发人员管理差分测试用例和代码维护提供帮助.

差分测试有其局限性: 如果两个功能等价的实例中包含同样的缺陷, 则在输出上没有明显差异, 差分测试将无法检测到这些缺陷, 容易漏报. 此外, 差分测试输出不一致不一定是由实际的缺陷导致的, 需要研究人员进一步确认. 因此, 研究人员一般结合其他测试方法设计 DL 框架测试方法.

5.3 测试评估

在对软件系统进行测试之后, 还需进一步评估和衡量软件测试质量和效果. 在对框架测试的研究工作中, 主要包括对测试有效性和测试充分性两方面的评估. 除通过检测出的缺陷数量和类型来评估测试有效性以外, 研究者还采用变异测试评估测试用例的质量和有效性. 在测试充分性方面, 研究者主要通过设计测试覆盖度量进行评估. 需要说明的是, 测试充分性是测试有效性的先决条件, 但充分的测试并不总是意味着测试有效性较高. 在有些情况下, 即使进行了大量的测试, 也可能会出现一些未被发现的缺陷和问题. 如何更加全面地评估测试策略的质量和效果是本文在调研分析过程中重点关注的研究问题.

5.3.1 测试覆盖度量

1975 年, Goodenough 等人[75]首先在软件系统测试领域引入了测试充分性 (test adequacy) 这一概念. 测试充分性是指一组测试数据在软件系统中的表现是否能够充分反映该软件的总体表现. 测试覆盖度量是用来评估测试充分性的重要指标, 传统软件测试中比较常用的代码覆盖度量 (例如行覆盖、分支覆盖和条件覆盖等) 主要是针对软件代码设计的, 衡量的是测试输入对程序源代码执行的程度. 近年来, 研究人员针对 DNN 模型特点不断提出测试覆盖度量. 比如, Pei 等人[76]在白盒测试框架 DeepXplore 中首先提出用神经元覆盖 (neuron coverage) 作为度量, 并用神经元覆盖率指导 DNN 模型的测试输入生成. 但 DL 框架内部组件的特征与传统软件和 DNN 模型差异较大, 针对源代码设计的代码覆盖和针对神经网络结构设计的神经元覆盖等均不完全适用于 DL 框架测试. 如何针对 DL 框架的特点设计测试覆盖度量指标是本文在调研分析过程中重点关注的研究问题.

经过调研, 本文总结现有工作提出的基于框架内部组件特征设计的测试覆盖度量, 如表 5 所示. 目前 API 覆盖是评估 DL 框架测试充分性最常用的度量指标, 此外还有部分研究人员提出更有针对性的测试覆盖度量. 例如, COMET[77]重点关注 API 调用的 3 个性质 (层输入、层参数值和层序列) 来度量层 API 调用的多样性和充分性. 基于这 3 个性质, 作者提出层类型覆盖、层对覆盖和层参数覆盖这 3 种覆盖准则来驱动模型突变过程, 生成可以覆盖更多 API 的模型对框架进行测试, 提高测试充分性. GraphFuzz[78]在从计算图的角度对 DL 框架进行测试的研究中, 提出算子级覆盖这种更加细粒度的充分性准则, 主要包括算子类型覆盖、张量形状和参数覆盖, 基于计算图理论的算子入度覆盖、出度覆盖和边覆盖. 这项工作在进行模糊测试的过程中, 将最大算子级覆盖率作为目标来暴露模型推理阶段的输出错误, 提高了框架测试的充分性和有效性.

表 5 深度学习框架测试覆盖度量

文献	测试覆盖度量	说明	是否指导测试过程
COMET[77]	层类型覆盖 (layer type coverage)	被调用层API占所有预定义层API的百分比	是
	层对覆盖 (layer pair coverage)	覆盖到的两层API之间调用序列在预定义层API之间所有可能调用序列的所占比例	
	层参数覆盖 (layer parameter coverage)	调用层参数的百分比	
DeepCov[81]	层边覆盖 (layer edge coverage)	覆盖到的模型每个层和层之间连接的边的比例	是
Muffin[82]	功能API覆盖 (functionality API coverage)	覆盖到的API在所有与学习相关API所占比例	否
SkipFuzz[83] FreeFuzz[66] DeepREL[84] TitanFuzz[85] FuzzGPT[86]	API覆盖 (API coverage)	覆盖到的API数量	否
GraphFuzz[78]	算子类型覆盖 (operator type coverage)	覆盖到的算子类型所占比例	是
	入度覆盖 (input degree coverage)	覆盖到的算子入度所占比例	
	出度覆盖 (output degree coverage)	覆盖到的算子出度所占比例	
	边覆盖 (single edge coverage)	覆盖到的算子连接边所占比例	
	形状/参数覆盖 (shapes/parameters coverage)	覆盖到的算子形状参数组合数量所占比例	

5.3.2 变异测试

如图 13 所示, 变异测试 (mutation testing) 是一种基于故障注入的测试技术, 将按照不同规则变异生成的错误代码段插入到被测程序中, 以验证当前测试输入是否可以发现注入的错误[79]. 被注入错误的代码段称为变异体 (mutant), 在符合约束条件的前提下, 将原有代码段转变成变异体的操作符为变异算子 (mutation operator). 在注入

变异后, 测试输入能发现该错误, 即变异体被“杀死”, 则表明测试输入是有效的; 反之, 变异体“存活”则表明需要补充该变异的测试输入. 变异测试的重要作用在于评估测试输入的有效性, 即检测缺陷的能力.

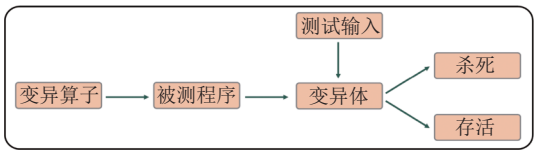

图 13 变异测试概况

Harzevili 等人[52]为了印证他们在 DL 框架安全漏洞实证研究上的发现和结论, 开发出自动变异测试工具 DeepMut. 根据已知的安全漏洞模式和攻击方式, DeepMut 可以实现在代码段中引入特定的安全感知变异算子, 以评估框架对外来攻击的抵抗能力. DeepMut 将 TensorFlow 作为实验对象, 最终发现在现有 DL 框架测试用例下, 3 000 个变异体中有超过 1 000 个变异体没有被现有“杀死”, 说明当前测试用例质量较低, 应该尽快提高测试用例的质量和覆盖范围. Jia 等人[80]设计 13 类变异算子将故障注入 DL 框架, 并通过观测程序运行的输出行为, 判断现有测试输入是否可以有效检测注入的错误. 研究发现, 超过 60% 植入的缺陷在相应测试输入下不会发生崩溃, 其输出行为与正常结果没有显著差异, 写在测试用例的断言并没有发现很多植入缺陷, 说明当前测试用例质量较低. 以上研究表明变异测试在评估测试有效性, 特别是评估测试用例质量方面的重要作用.

变异测试的局限性在于可能会导致源代码的语义变化, 生成无效的变异体, 浪费测试资源. 其次, 变异测试对框架测试评估的效果依赖于变异算子. 如果变异算子实现的操作不够全面或者不够准确, 可能会导致变异测试的效果不佳.

6 深度学习框架测试方法

本文在第 5 节中论述的测试关键技术是指解决 DL 框架测试中某一问题的具体技术, 而仅采用单一的测试技术并不能对 DL 框架进行全面的缺陷检测和质量评估. 本节所论述的测试方法是用来指导测试执行的整体性方案, 重点考虑如何针对具体被测对象和测试研究问题合理配置不同的测试技术, 以确保测试的有效性、充分性和效率.

测试输入生成是 DL 框架测试方法中的第 1 步, 也是最为关键的一步. 测试输入的质量决定整个测试方法的实际作用效果, 高质量的输入可以高效揭露 DL 框架核心代码中的缺陷. 本节对现有研究工作进行梳理和分析, 根据不同的测试输入形式, 将 DL 框架测试方法分为以 DNN 模型为输入的系统测试 (system testing), 以计算图为输入的系统测试和以特定参数为输入的组件测试 (component testing) 这 3 类. 除输入形式的不同, 这 3 类测试方法区别在于, 前两类系统测试方法通过 DNN 模型或计算图调用相关组件完成 DL 任务, 侧重于对 DL 框架整体功能实现方面是否存在缺陷进行系统性测试; 而后者主要针对 DL 框架中的基本组件 (API 和算子) 进行更加细粒度的测试.

6.1 以 DNN 模型为输入的系统测试

本节主要论述以 DNN 模型为输入的系统测试研究, 包括这一类测试方法的总体概述和每一项相关工作的具体分析.

6.1.1 总体概述

如图 14 所示, 以 DNN 模型为输入的系统测试一般结合基于模糊测试的测试输入生成技术, 基于差分测试的测试预言技术以及测试输入选择算法来实现. 这类研究工作的主要特征在于需要构建出完整的 DNN 模型, 通过对比相同输入时, 同一 DNN 模型在不同 DL 框架实现下的输出是否一致, 来检测框架内可能存在的缺陷.

本文通过进一步分析发现, 这类研究工作的难点在于如何尽可能多地生成有效且多样化的 DNN 模型作为测试输入, 有效即模型结构和参数配置本身不存在问题, 多样化即要求模型结构不尽相同, 尽可能覆盖多样性的 API 和算子. 为了实现这一目标, 研究人员在测试输入生成方面采用基于突变的模糊测试或基于生成的模糊测试两种

方法; 同时采用差分测试方法解决测试预言不明确的问题. 此外, 为了提高测试效率, 使缺陷检测更有针对性, 部分工作引入启发式搜索算法, 利用测试过程的中间信息 (测试覆盖率或输出不一致程度等) 指导模糊测试中的测试输入生成; 或者引入选择算法, 在输入选择上过程中, 给予缺陷发现能力更强的种子模型更大的优先级. 这一类测试方法的整体思路基本一致, 但是在具体算法选择和实施上不尽相同, 比如采用基于突变的模糊测试生成新模型的工作往往需要额外引入对突变算子的选择算法, 采用基于生成的模糊测试生成模型的工作重点在于研究如何构造一组合理的模型生成规则.

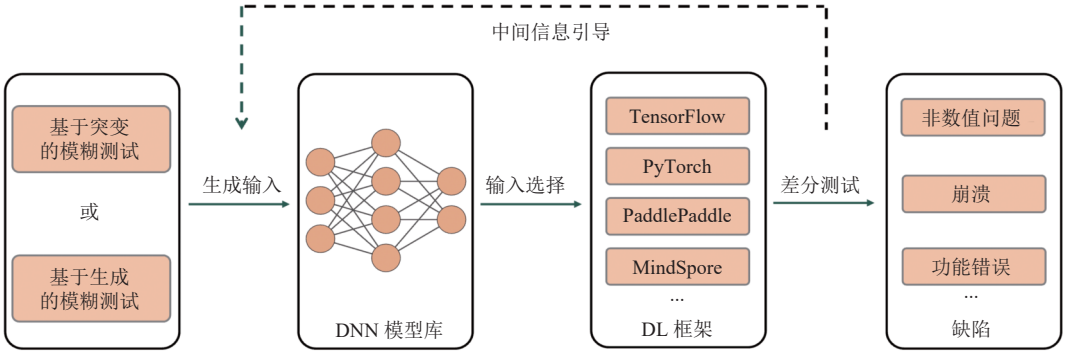

图 14 以 DNN 模型为输入的 DL 框架系统测试概况

以 DNN 模型为输入的系统测试普遍采用差分测试技术来缓解测试预言难的问题, 可以检测到除崩溃以外的多种类型缺陷. 但由于 DNN 模型训练或预测过程中需执行大量非线性操作, 因此需要耗费较多的计算资源和时间才能在差分测试中检测到输出之间的差异. 此外, 差分测试不一致的输出不一定是由实际缺陷导致的, 需要额外的工作进行缺陷确认和缺陷定位. 以 DNN 模型为输入的系统测试将 DL 框架视为一个抽象的整体系统, 其测试目标在于验证整个 DL 框架功能实现是否存在缺陷, 因此测试粒度更粗. 此外, 以 DNN 模型为输入的系统测试只能覆盖到使用频率较高的 API 组件, 难以覆盖用户较少调用的 API, 或者某种少见的 API 调用方式, 测试充分性较低.

6.1.2 相关工作

如表 6 所示, 本节将现有研究工作提出的以 DNN 模型为输入的系统测试方法进行梳理和对比, 重点介绍如何解决测试输入生成、测试预言和测试评估方面存在的难点问题, 并对比分析每项工作的优缺点和适用范围.

表 6 以 DNN 模型为输入的 DL 框架系统测试方法总结

文献	时间, 来源	测试输入生成	测试预言	测试覆盖度量
CRADLE[72]	2019, ICSE	—	差分测试	—
AUDEE[87]	2020, ASE	基于突变的模糊测试	差分测试	—
LEMON[65]	2020, ESEC/FSE	基于突变的模糊测试	差分测试	—
Ramos[88]	2023, JSS	基于突变的模糊测试	差分测试	—
FAME[89]	2021, DSC	基于突变的模糊测试	差分测试	—
DeepCov[81]	2022, DSC	基于突变的模糊测试	差分测试	层边覆盖
COMET[77]	2023, TOSEM	基于突变的模糊测试	差分测试	分支覆盖、层类型覆盖、层对覆盖、层参数覆盖
MMOS[90]	2022, GLOBECOM	基于突变的模糊测试	差分测试	—
NeuRI[63]	2023, arXiv	基于生成的模糊测试	差分测试	分支覆盖
ExAIS[91]	2022, ICSE	基于生成的模糊测试	差分测试	—

注: “—”表示该工作未涉及相关内容

CRADLE[72]最早开始从 DNN 模型的角度对 DL 框架测试展开研究, 这项工作结合差分测试的方法, 创新性地提出输出层间距 *D_CLASS* 和 *D_MAD* 以及隐藏层间距变化率 R_L 两类指标, 分别用于缺陷检测和缺陷定位. 输出

层间距用于评估由相同 DNN 模型在不同 DL 框架实现下的输出不一致程度, 在对比发现可能存在缺陷的框架后, 根据隐藏层间距变化率可以定位产生缺陷的某一层, 从而进一步确定缺陷实际位置. 这项工作成功地在 Keras 不同库后端中发现 12 个缺陷. CRADLE 提出的两个指标多次被后续工作采用, 为以 DNN 模型为输入的系统测试这一类工作奠定了基础, 但是该项工作只涉及对已有的开源模型进行差分测试, 无法生成新的模型以覆盖 DL 框架中更多的 API, 测试充分性较差. 此外, 误差累积等不确定因素造成的输出不一致程度可能和缺陷造成的输出不一致程度是相近的, 这一局限性为后续缺陷定位和确认造成困难.

AUDEE[87]在 CRADLE 的基础上创新性地引入新模型生成方面的工作, 一定程度上解决了测试充分性问题. 这项工作采用基于遗传算法的启发式搜索方法, 设计 3 种不同的突变策略来探索网络结构、模型权重和种子数据集的不同组合, 实现更加复杂的模型. 值得注意的是, AUDEE 将 CRADLE 的隐藏层间距作为遗传算法的适应度指标来指导测试输入生成过程, 在模型不断迭代突变的过程中, 适应度指标可以不断放大模型在不同 DL 框架上的输出不一致程度. 实验表明, AUDEE 成功地在 TensorFlow 等 4 个 DL 框架中检测到 26 个缺陷, 包括逻辑缺陷、崩溃和非数值错误这 3 种类型. AUDEE 虽然可以通过突变生成更多复杂的模型, 但其种子模型结构并没有发生本质改变.

LEMON[65]同样把重点放在如何生成数量更多, 多样性更强的模型中. 与 AUDEE 不同的是, LEMON 主要以网络模型的层为突变对象, 直接改变已有模型的结构. LEMON 共设计 7 种层间的突变规则和 5 种层内突变规则. 通过该对种子输入实施突变, 可以生成更多新的模型. 由于输入空间过大, 为提高效率并保证模型多样性, LEMON 采取轮盘赌算法 (roulette wheel selection)[92]作为种子模型选择策略, 使之前很少被选中的种子模型在下一轮迭代中有更大可能被选中. 此外, 基于 CRADLE 的输出层间距指标 *D_MAD* , LEMON 设计累计不一致程度指标作为突变规则选择策略的重要依据. LEMON 采用马尔可夫链蒙特卡罗 (Markov chain Monte Carlo, MCMC) 算法作为突变规则选择策略, 通过参考突变规则的历史行为, 并加入一定的随机性, 从而决定每个突变规则的排序优先级. 相比于 AUDEE, 这项工作的进步在于开始对 DNN 模型中的结构进行突变, DNN 模型多样性得到极大提高.

Zou 等人[88]认为 LEMON 的突变规则难以生成新的算子之间互相调用关系, 因此模型多样性较差. Zou 等人提出 Ramos, 通过设计分层和启发式方法生成模型, 其中每一层代表一组算子之间的相互调用关系. 通过对算子调用边进行突变, 来实现更多样化的算子调用组合. 为避免不同 DL 框架相似 API 的参数配置不同导致的缺陷误报, Zou 等人提出了一组 API 映射规则, 并将其应用于不同框架下的模型代码转换, 来实现差分测试. Zou 等人在 TensorFlow、PyTorch 和 MindSpore 上验证 Ramos 方法的性能, 实验结果显示, Ramos 对崩溃的误报率为 0, 同时可以触发更多的计算错误.

Shen 等人[89]指出了 LEMON[65]中部分突变规则的局限性, 例如增加层 (layer addition) 规则要求相邻的层类型和结构要严格匹配, 这种突变约束同样限制了生成模型的多样性, 不利于检测出崩溃一类的缺陷. 因此, 作者提出 FAME, 通过对 LEMON 中的突变规则进行优化改进, 并补充设计 API 突变规则来生成包含大量有效的模型.

Wu 等人[81]认为以往工作采用输出不一致作为主要指标驱动整个测试过程有一定的局限性. 他们提出 DeepCov, 在 LEMON 的基础上, 重点考虑模型层之间的连接关系, 设计增加关键层 (strategic layer addition) 的突变规则和层边覆盖 (edge layer coverage) 准则作为模型选择过程中的适应度指标.

Li 等人[77]提出覆盖引导的模型生成方法——COMET 对框架进行测试. 针对模型运行成本高, 测试效率低的问题, COMET 在不影响模型多样性的前提下, 首先对模型规模进行压缩. 针对模型多样性差的问题, COMET 针对层类型、层对和层参数这 3 个方面提出多种类型的突变规则. 与 LEMON 不同的是, COMET 将压缩后的种子模型和突变规则的组合作为 MCMC 选择策略的实施对象, 以确保突变规则对于特定种子模型的有效性. COMET 同样参考 CRADLE 中的输出层间距指标 *D_MAD* 来评估模型在不同 DL 框架上对输出不一致程度, 最终实现差分测试. 值得注意的是, 除分支覆盖外, 作者重点提出层类型覆盖、层对覆盖和层参数覆盖这 3 种评估测试充分性的覆盖度量. 实验结果显示, COMET 在测试充分性方面完全超过了 CRADLE 和 LEMON 的性能, 并且成功检测到之前未报道的缺陷 29 个.

Li 等人[90]发现之前研究工作一般认为不同突变规则选择策略发现缺陷的效率是一致的, 比如 LEMON 和

COMET 中采用的 MCMC 算法对所有突变规则设置了相同的选择权重, 一定程度上影响了缺陷检测效率. 因此, 作者提出 MMOS, 设计多阶段的突变算子选择策略和基于多臂赌博机 (multi-armed bandit) 模型[93]的突变算子能量调度策略, 成功提高了缺陷检测的效率.

Liu 等人[63]认为以往研究缺乏对模型算子约束的考量, 无法使用不同的算子直接构造有效且多样的模型. 为了解决这个问题, 他们提出一种通过综合分析 API 约束来生成模型的方法——NeuRI. 在 FreeFuzz[66]抓取到的开源代码基础上, NeuRI 追踪分析这些代码, 收集其中涉及的 API 调用信息; 然后利用这些信息实施归纳程序综合 (inductive program synthesis)[94]来推理出可以构造有效模型的规则; 在这些规则的指导下, 通过在已知有效的 DNN 程序不断中插入正确的算子代码段来完成模型的生成.

Schumi 等人[91]引入 Prolog 来生成有效的模型. Prolog 是一种逻辑编程语言, 它可以根据事实和规则自动分析其中的逻辑关系, 并且允许用户通过查询, 完成复杂的逻辑运算, 广泛应用在 NLP 等人工智能领域[95]. 基于 Prolog 对 TensorFlow 中所有与层构造相关的 API[96]分析出的可执行语义信息, 可以在模糊测试过程中对层相关 API 进行配置从而自动生成完整有效的模型. 同时, Prolog 分析出的可执行语义信息可作为明确的测试预言完成差分测试. 实验表明, 该研究提出的方法成功检测到 TensorFlow 中 14 个缺陷, 有助于提高 DL 框架质量.

6.2 以计算图为输入的系统测试

本节论述以计算图为 DL 框架测试输入的系统测试研究, 主要包括这一类测试方法的总体概述和每一项工作的具体分析.

6.2.1 总体概述

计算图 (也称数据流图) 是一种用来描述运算的有向无环图 (directed acycline graph) 数据结构. 计算图由节点和边组成, 每个节点都表示一种由算子执行的运算, 边表示张量的流动方向. 2009 年, 深度学习三巨头之一的 Bengio 为了理清数据输入和各种运算之间的关系, 首次将计算图引入人工智能领域[97].

如图 15 所示, DL 框架通过计算图来组织和执行模型的计算过程, 其中定义了数据流转方式, 数据计算方式以及各种计算节点之间的相互依赖关系[98]. 计算图作为 DNN 模型在后端执行时的一种中间表示, 可以为各类 DNN 模型的结构和运行过程提供统一的抽象描述, 承接前端编程语言和接口, 同时方便底层库的编译优化.

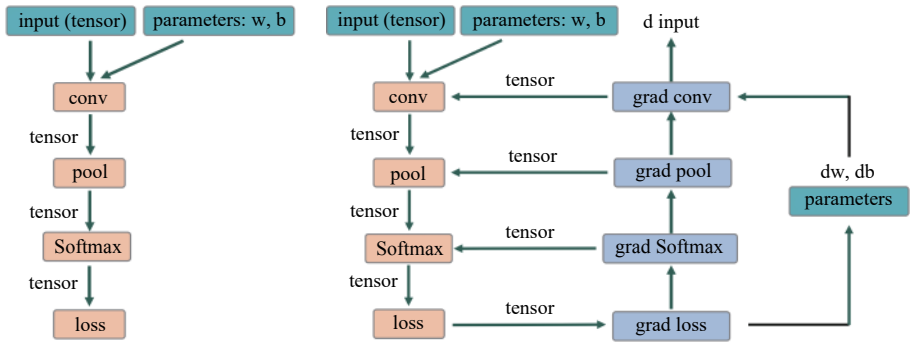

图 15 计算图执行过程示例

在实现深度学习任务时, DL 框架把 DNN 模型统一转换为计算机能够识别的计算图, 通过计算图调用相关算子实现反向传播算法中的自动梯度计算, 更新网络权重. 以计算图为输入的系统测试方法, 可以帮助检测 DL 框架在描述 DNN 模型结构, 节点操作或边数据流时是否存在缺陷, 确保整个 DL 框架实现的正确性.

6.2.2 相关工作

如表 7 所示, 本节将以计算图为输入的系统测试相关研究工作进行分析和总结.

为了使自动生成的模型更加多样化, 尽可能多地覆盖库中不同层类型函数, Muffin[82]参考神经网络架构搜索

方法 (neural architecture search)[99]使用计算图的结构模版 (例如链式结构和单元结构) 来生成模型结构信息, 利用计算图的节点入度生成层信息, 在此基础上逐层描述 DNN 模型. 在实施差分测试时, 不同于之前的研究工作多关注模型的推理阶段, Muffin 着重监测模型训练阶段的数据, 通过模型输出不一致检测可能存在的缺陷. 这项工作首次实现对 DL 框架训练相关功能实现进行测试, 成功实现 98.305% 的覆盖率, 并在 3 个框架中检测到 39 个之前未知的缺陷.

表 7　以计算图为输入的系统测试方法总结

文献	时间, 来源	测试输入生成	测试预言	测试覆盖度量
GraphFuzz[78]	2021, ICSE	基于突变的模糊测试	差分测试	算子级覆盖
EAGLE[73]	2022, ICSE	基于生成的模糊测试	差分测试	—
Muffin[82]	2022, ICSE	基于生成的模糊测试	差分测试	API覆盖, 代码行覆盖

注: “—”表示该工作未涉及相关内容

图论可以帮助研究和解释神经网络的拓扑结构[100], Luo 等人[78]从计算图的角度出发, 提出 GraphFuzz. 通过设计 4 种模型级突变策略和 2 种源级突变策略对 DNN 模型对应的计算图进行突变, 来探索模型结构、参数和数据输入的组合. 同时引入一系列基于图论的算子级覆盖度量指导种子库更新过程, 采用蒙特卡罗树搜索 (Monte Carlo tree search, MCTS) 算法[101]作为种子选择策略在种子库中选择子图. 实验结果显示, 相比随机搜索, 基于 MCTS 的搜索在提高算子级覆盖率和检测异常方面表现更好.

EAGLE[73]采取同一 DL 框架实现等价计算图来执行差分测试的思路. 首先根据文档中 API 的描述和 DL 框架中已知缺陷的特征挖掘信息, 总结出 16 条计算图等价规则; 用相关的 API 实例化这些规则, 从而生成两个功能等价的计算图作为 DL 框架的输入; 利用差分测试的思想, 通过对比两个计算图的输出是否一致, 从而测试框架中是否存在缺陷. 作者将 EAGLE 方法实施在 TensorFlow 和 PyTorch, 成功检测到 13 个之前尚未发现的缺陷, 9 个缺陷已得到开发人员的确认并进行修复.

6.3 以特定参数为输入的组件测试

本节论述以特定参数为输入的组件测试方法研究工作, 主要包括这一类测试方法的总体概述和每一项工作的具体分析.

6.3.1 总体概述

在传统软件系统中, 组件一般指系统中相对独立且具有特定功能的模块或单元, 开发人员可以通过组件测试对每个组件独立运行时的功能进行验证. 如本文第 2.2 节所述, API 和算子均是 DL 框架的内部组件, 只是其复杂程度和使用场景不同. 前者为用户提供更高抽象级别的接口, 而后者更接近基本功能函数的底层代码, 负责实现具体的计算操作. 在 DL 框架测试领域, 部分研究工作针对 DL 框架内部 API 或算子进行缺陷检测, 从而实现更加细粒度的测试.

在软件测试领域, 参数是指在调用 API 或算子时传递给内部代码的输入值. 不同于传统软件系统, DL 框架组件测试中输入的特定参数具有明显的领域特征, 其参数格式和使用方式与具体 DL 任务和 DL 框架内部程序规范相关. 例如, DL 框架通常采用张量来表示高维数据, 用于支持高效的数值计算和模型训练. 此外, 部分参数的数据维度和数据类型等性质不唯一, 需要根据不同深度学习场景下的数据处理和计算需求而确定. 如图 16 所示, 以 TensorFlow 中二维卷积 API[102](tf.nn.conv2d) 部分参数为例, 参数 input 的数据类型是张量, 其数据类型可以是 half 或 bfloat16 等, 参数 strides 的数据类型可以是整数型 (int) 或列表 (list).

以特定参数为输入的组件测试主要特征在于直接针对 DL 框架中的 API 或算子尽可能生成满足约束的参数输入来检测其中是否存在缺陷. 如图 17 所示, 研究人员在测试输入生成方面一般采用基于突变的模糊测试和基于生成的模糊测试两种方法, 难点在于如何满足特定参数的约束, 比如参数的维度、形状、数值、类型以及参数之间的依赖关系等. 部分研究工作在采用基于生成的模糊测试方法时, 以 DL 框架官方文档为信息来源, 来获取特定

参数的具体约束; 部分研究工作在采用基于突变的模糊测试方法时, 一般以开源的 API 调用代码段作为种子, 通过对种子中的参数实施多维度突变来生成更多的测试输入. 此外, 一部分工作采用差分测试方法解决测试预言不明确的问题, 或者直接监测是否出现崩溃等显式的缺陷. 为了提高测试效率, 少部分工作引入启发式搜索算法, 利用测试过程的中间信息指导模糊测试中的测试输入生成.

Args	
Input	A Tensor. Must be one of the following types: half, bfloat16, float32,float64. A Tensor of rank at least 4. The dimension order is interpreted according to the value of data_format; with the all-but-inner-3 dimensions acting as batch dimensions. See below for details.
Filters	A Tensor. Must have the same type as input.A 4-D tensor of shape [filter_height, filter_width, in_channels, out_channels]
Strides	An int or list of ints that has length 1, 2 or 4. The stride of the sliding window for each dimension of input. If a single value is given it is replicated in the H and W dimension. By default the N and c dimensions are set to 1. The dimension order is determined by the value of data_format, see below for details.

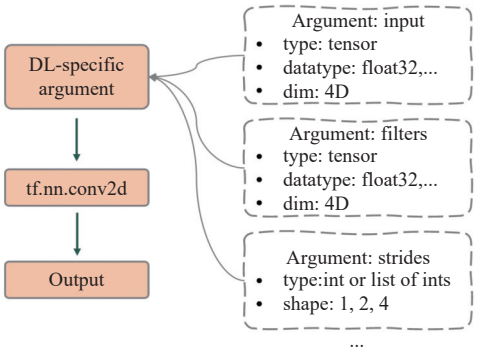

图 16　tf.nn.conv2d 部分参数

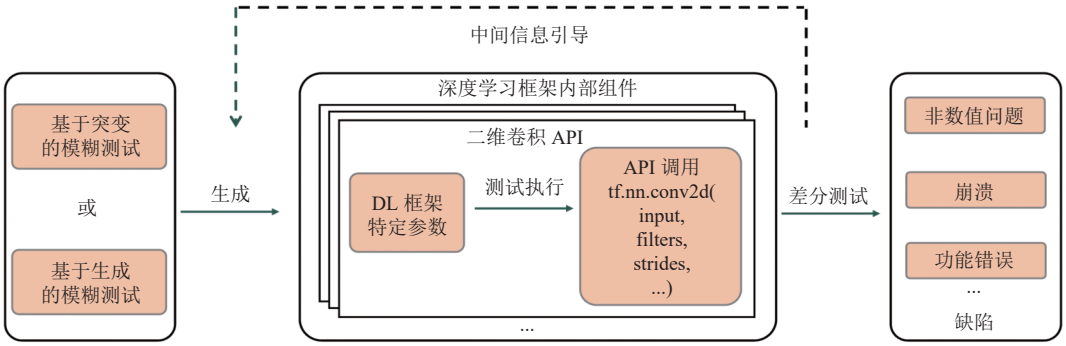

图 17　以特定参数为输入的 DL 框架组件测试方法概况

相比以 DNN 模型或计算图为输入的系统测试方法, 以特定参数为输入的组件测试方法更加细粒度, 可以直接覆盖到更多的深度学习功能性组件, 测试充分性较强, 测试成本较低. 但以特定参数为输入的组件测试方法通常难以对组件之间的交互以及系统的整体功能实现进行测试, 而往往某些缺陷只有在组件之间的相互调用时才会被检测出来. 以特定参数为输入的组件测试同样面临测试充分性不足的问题, 无论是采用基于生成的模糊测试或基于突变的模糊测试的测试输入生成方法, 都难以全面地覆盖到 DL 框架中的 API 库. 此外, 现有研究工作生成的参数输入不同程度地存在不符合约束的问题, 影响整个测试过程的效率.

6.3.2　相关工作

如表 8 所示, 本节对以特定参数为输入的组件测试相关工作进行总结和分析, 同时对每项工作对应的测试层级 (API 和算子) 加以区分. 其中, 针对算子的测试研究工作更多地将其底层 C/C++代码作为具体实施对象.

Christou 等人[103]利用 API 之间的层级关系提出自下而上的缺陷检测策略——IvySyn. DL 框架的低阶 API 一般是用 C/C++编写的, IvySyn 则利用这种低阶 API 的静态代码特性对其实施基于突变的模糊测试, 从而发现输出崩溃的“违规输入”. IvySyn 根据低阶 API 到高阶 API 的映射关系, 利用“违规”输入合成可以触发内存错误的代码段, 用这种方式来检测 DL 框架中的内存安全漏洞.

FreeFuzz[66]这项工作实现了 API 级的全自动测试流程. FreeFuzz 通过运行从开源信息中收集到的代码片段来

抓取 API 中间过程信息, 包括 API 中的参数类型、参数数值以及输入或输出张量的形状. 将这些中间过程信息作为种子输入, 通过对其实施不同突变策略的模糊测试来生成更多 API 的输入和参数. 相同 API 为适配不同硬件, 其具体代码实现方式也有所不同. FreeFuzz 采用差分测试的方法, 对比相同 API 在不同硬件设备 (CPU, GPU) 上输出的不一致来检测计算错误缺陷; 利用数值精度和运算速度之间的蜕变关系实施蜕变测试来检测性能缺陷; 通过监测 API 运行过程来检测崩溃和异常. FreeFuzz 这项工作的局限性在于, 难以对该开源代码片段未涉及的 API 进行测试, 测试覆盖率有限. 此外, DL 框架在不同的硬件平台上代码实现逻辑相似, 因此这种差分测试中采用的测试预言关系不明确, 最终导致可以检测到的缺陷类型有限.

表 8　以特定参数为输入的组件测试方法总结

文献	时间, 来源	测试层级	测试输入生成	测试预言	测试覆盖度量
IvySyn[103]	2023, USENIX Security	API	基于突变的模糊测试	—	—
FreeFuzz[66]	2022, ICSE	API	基于突变的模糊测试	差分测试、蜕变测试	API覆盖
DeepREL[84]	2022, ESEC/FSE	API	基于突变的模糊测试	差分测试	API覆盖
TitanFuzz[85]	2023, ISSTA	API	基于突变的模糊测试	差分测试	API覆盖、代码覆盖
FuzzGPT[86]	2023, arXiv	API	基于突变的模糊测试	—	API覆盖、代码覆盖
∇Fuzz[104]	2023, ICSE	API	基于突变的模糊测试	差分测试	代码覆盖
SkipFuzz[83]	2019, arXiv	API	基于生成的模糊测试	—	API覆盖、输入性质覆盖
DocTer[64]	2022, ISSTA	API	基于生成的模糊测试	差分测试	—
ACETest[105]	2023, ISSTA	算子	基于生成的模糊测试	—	分支覆盖
Predoo[106]	2021, ISSTA	算子	基于突变的模糊测试	差分测试	—
Duo[107]	2021, TR	算子	基于突变的模糊测试	差分测试	—
Gu等人[98]	2022, 计算机学报	算子	基于突变的模糊测试	差分测试	—

注: “—”表示该工作未涉及相关内容

DeepREL[84]受传统软件测试的启发, 通过定义两种不同粒度的 API 等价规则和用自然语言处理方法寻找等价 API 来两部分内容实现差分测试. 等价规则包括输出值等价和输出状态等价, 其中输出值等价是指在相同输入下, 功能相同的 API 其输出值应该是一致的; 而输出状态等价是指 API 输出状态一般有运行成功, 出现异常, 出现崩溃这 3 种可能的结果. 在相同参数下, 功能相似的 API 虽然输出值不尽相同, 但输出状态应该是相同的. DeepREL 采用 NLP 的方法从 API 文档中找到功能等价或相似的一对 API 后, 用 FreeFuzz 中突变生成的测试输入来调用成功配对的 API. 通过差分测试对比一对 API 的输出是否满足上述两类等价规则, 如果不满足, 则说明相应 API 可能存在缺陷. DeepREL 相比 FreeFuzz 的提高在于运用两种等价规则使差分测试中预言关系更加明确, 有利于充分测试出多种缺陷类型, 但输入多样性依然有限.

为了使测试覆盖到更多的 API 序列, Deng 等人[85]采取基于突变的模糊测试思想, 利用大语言模型[108]生成测试输入, 提出 TitanFuzz. 在规范的输入提示下, 生成式大语言模型 (generative LLM, 例如 Codex[109]) 可以自动生成高质量的种子代码段组成种子库; 和以往模糊测试研究类似, 为了不断扩充种子库, TitanFuzz 采取进化策略 (evolution strategy)[110], 利用适应度函数指导种子选择过程, 利用多臂赌博机算法指导突变规则优先级排序; 在选择某个突变规则之后, TitanFuzz 利用填充式大语言模型 (infilling LLM, 例如 Incoder[111]) 实现代码补全, 生成变异体代码, 获得更多样的参数输入. 因此, TitanFuzz 可以在不断模糊迭代的过程中生成更多复杂的代码段, 使其可以调用更多样的 API 序列. TitanFuzz 通过在不同的硬件设备上执行差分测试来检测可能存在的缺陷, 比如计算错误和崩溃. 实验结果显示, 这项工作在代码行覆盖, API 覆盖方面的实际作用效果已超出以往研究工作. 此外, 这项工作率先将 LLM 应用在 DL 框架测试研究中, 利用 LLM 生成测试输入, 具有一定的开创性.

Deng 等人[86]认为历史触发缺陷的代码段中往往包含对发现缺陷有用的要素, 比如某个不常规的 API 调用, 参数边界值等. Deng 等人提出 FuzzGPT, 利用 LLM 学习历史触发缺陷代码段的特征, 以生成新的测试用例.

FuzzGPT 通过少样本学习 (few-shot learning) 提示技术, 利用 LLM 分析出历史触发缺陷代码段中存在缺陷的 API, 进行缺陷数据标注, 构造缺陷数据集. FuzzGPT 将标注好的缺陷数据作为提示, 使 LLM 生成或编辑包含新参数输入的代码段. 此外, FuzzGPT 采用构造出的缺陷数据集对 LLM 进行模型参数微调, 使 LLM 更好地适配该任务. 通过采用差分测试方法, 分别在 CPU 和 GPU 上运行 LLM 生成的代码段, FuzzGPT 成功检测到 49 个之前未被发现的缺陷. 相比于 TitanFuzz, 这项工作在利用 LLM 学习历史触发缺陷代码的特征, 使生成测试输入方面更有针对性.

Yang 等人[104]认为过去研究大多关注模型推理阶段的测试, 他们提出∇Fuzz, 参考 FreeFuzz 中测试输入生成方法, 重点对模型训练阶段中自动微分 (automatic differentiation) 功能组件进行测试, 实现全自动测试过程. 自动微分是 DL 框架为实现反向梯度传播算法提供的核心功能组件集合, 为用户屏蔽了繁琐的求导细节和过程. ∇Fuzz 通过自动创建装饰器函数, 将 API 调用抽象成从输入到输出的函数映射, 然后对每个 API 映射出的函数及其低阶和高阶梯度进行差分测试来检测自动微分过程中是否存在缺陷.

有效的 API 测试输入都应该满足一定的约束. Kang 等人[83]提出 SkipFuzz, 在测试输入生成方面采取了以往研究不一样的路径. SkipFuzz 采用主动学习 (active learning)[112]的方法, 通过使用模糊测试过程中获得的 API 函数信息推断输入约束, 从而生成有效的参数输入. 通过这种方式, SkipFuzz 生成的参数输入更多样性, 有效参数输入所占的比例也高于之前的研究工作, 并有效缓解了模糊测试过程中测试输入冗余的问题.

Xie 等人[64]从 API 文档中的信息出发对 API 进行测试, 提出 DocTer. API 文档包含大量对参数配置的自然语言描述, DocTer 通过采用依存关系分析方法分析部分 API 文档中分析出频繁出现的句法模式[113], 从而构造出一系列输入生成规则. 在这些规则下, 所有 API 的依存关系分析树 (dependency parse tree) 都可以映射成具有深度学习领域特征的输入约束, 再基于这些约束来生成大量的模糊测试输入. DocTer 重点研究测试输入的边界值对检测 API 缺陷的有效性. 此外, DocTer 还通过生成无效输入来检测 API 输出是否会发上崩溃或者异常, 如果没有发生, 同样说明该 API 可能存在缺陷. 这项工作相比于 FreeFuzz 和 DeepREL 等相关工作的进步在于, 不需要受限于种子输入, 生成的 API 参数输入更加广泛. 但在构造 API 参数约束阶段, 需要人工介入对一部分 API 信息进行数据标注, 耗费人力较多.

算子是 DL 框架中基本功能函数的代码实现, 用来执行各种基本的数据计算与维度转换, 比如数据归一化 (data normalization), 激活函数 (activation function) 等. 从广义上讲, 任何一个函数的具体实现都可以认为是一个算子. 算子的计算正确性, 计算效率和计算精度等对 DL 框架的质量有决定性的影响.

为更准确地提取 DL 算子的复杂约束, 提高测试用例有效率, Shi 等人[105]直接对 DL 算子源代码进行分析, 提出 ACETest. 作者发现 DL 算子源代码中输入验证代码和功能性代码一般是解耦的两部分, 因此可以在程序控制流图 (control flow graph) 上从错误处理函数调用处进行反向追踪, 定位输入验证代码段. 从代码执行路径中分析提取输入约束, 构建有效且多样化的参数输入, 从而发现 DL 算子中功能性代码中的缺陷. 作者将 ACETest 实施在 TensorFlow 和 PyTorch 的算子上, 成功检测出 108 个新缺陷, 其中 5 个缺陷被收录在 CVE 中. ACETest 创新性地将动态符号执行的思想运用在 DL 框架算子测试领域, 但是在缓解测试预言方面能力不足, 只能检测到崩溃一类的缺陷.

深度学习算子在执行大量非线性复杂计算的过程中容易出现精度问题[114], 算子精度的误差难以评估. Zhang 等人[106]提出针对算子精度的模糊测试方法——Predoo. Predoo 比较分析对 TensorFlow 框架中的 7 个典型非线性算子 (conv2d、norm、pooling、ReLU、Sigmoid、Softmax 和 tanh) 在不同精度设定下的计算结果进行比较分析, 评估底层计算硬件的不同对算子精度的影响. 该方法通过将算子测试任务转变为一个搜索问题, 以最大化输出精度误差为目标, 寻找 DL 框架中算子可能存在的不一致性.

在 Predoo 的基础上, Zhang 等人[107]创新性地提出了一种基于覆盖信息分布引导的差分模糊测试框架——Duo. 与 Predoo 不同的是, Duo 这项工作不再局限于数值精度问题, 而是对 7 个典型非线性算子 (conv2d、norm、pooling、ReLU、Sigmoid、Softmax 和 tanh) 的质量问题进行综合性测试研究. 该工作对种子参数输入实施两类突变策略, 实现算子测试输入自动生成; 通过蒙特卡罗算法和能量调度算法优化模糊测试执行过程, 并结合输出分布特征等信息, 对模糊过程的种子库进行动态更新; 最后通过差分测试实现对算子的实现缺陷、运行耗时和精度误

差等多维度评估. 实验结果表明, Duo 可以检测出实现错误, 执行时间成本高和精度误差大这 3 种类型缺陷.

谷典典等人[98]针对 DL 框架内算子实现错误导致的一类缺陷, 将不同算子的共性计算逻辑抽象为“元算子”(meta-operators), 设计并实现了基于元算子的 DL 框架缺陷检测. 这项工作通过算子细粒度替换的方式, 对不同 DL 框架实现下模型中的每个算子进行逐一检测, 记录每一次算子替换之后模型的推断结果和模型参数梯度相关数据, 评估前后两次数据的变化差异, 进一步定位缺陷. 作者将基于“元算子”的测试方法应用在包含错误计算缺陷算子的深度学习模型中, 验证了该缺陷检测策略的有效性. 但这项工作提出的“元算子”计算机制, 其计算效率和计算时内存占用情况仍有进一步优化的空间.

7 未来研究方向

深度学习技术已经被广泛应用到各个领域, 作为其中的关键基础软件, 深度学习框架的性能和质量逐渐受到产业界和学术界的广泛关注. 近年来研究人员针对深度学习框架展开了大量研究, 也取得了一定的成果, 但该领域仍面临着许多挑战. 本节内容对尚未解决的难点问题进行分析并展望进一步的未来研究方向, 希望可以为相关研究人员提供参考.

(1) 研究更高效更可靠的测试输入生成技术

测试输入生成技术存在多样性和有效性两个难题, 主要表现在生成的测试输入多样性受限, 导致测试覆盖率较低; 生成的测试输入有效性较差, 部分测试输入不符合规范, 影响测试效率. 现有的研究工作为保证测试输入有效性, 采用各种方法不同程度地满足其约束和规范, 测试输入多样性和可拓展性因此受到限制. 如何提出更高效、更可靠的测试输入生成方法, 保证测试输入的有效性和多样性的同时, 减少测试输入的冗余, 提高测试输入的利用率和质量, 仍然是一个亟待解决的研究问题.

目前大语言模型已成功应用到软件工程领域的多个方向, 其训练数据集包含大量来自 GitHub 等平台的开源代码数据, 因此可以学习到软件工程领域更加复杂的知识、规则、模式和逻辑. 已有工作表明, LLM 在代码生成[115]和缺陷修复[116]等方面的能力十分优秀. 相比以往测试输入生成研究, LLM 可以学习到更加复杂的 DL 框架隐式约束, 因此 LLM 未来会更广泛地应用到 DL 框架测试中. 在用户的提示输入下, LLM 可以生成符合深度学习领域规范的测试输入. 值得注意的是, 大模型和具体的下游任务之间差距巨大, 往往简单的提示输入难以得到期望的输出响应. 如何运用提示工程 (prompt engineering)[117]技术, 对输入文本信息按照特定模板进行处理, 研究如何把具体任务重构成一个更能充分利用 LLM 处理的形式, 从而利用 LLM 为 DL 框架组件生成更高效更可靠的测试输入是一个非常值得探究的方向.

(2) 研究如何测试并提高 DL 框架安全性

DL 框架是一个高复杂度的软件系统, 在开发过程中难免存在安全漏洞, 例如对抗攻击、隐私泄露和拒绝服务等, 这些漏洞将严重威胁深度学习系统的安全. Le Quoc 等人[118]基于 TensorFlow 提出 secureTF 平台, 引入包括加密计算和访问控制等多项技术, 为数据集、模型和代码提供安全性保障. Rosetta[119]通过隐私计算算法将 DL 框架里的各类算子转化为“隐私算子”, 即算子的功能不发生改变, 同时支持在隐私保护前提下的使用, 为人工智能提供保护隐私的解决方案.

现有工作大多是基于 DL 框架开发安全保护机制, 并没有对 DL 框架本身的安全性进行测试评估, DL 框架内部的安全漏洞可能一直存在. 未来可以进一步研究如何通过静态代码审查, 模糊测试等技术挖掘 DL 框架的安全漏洞, 或者提出更有针对性的 DL 框架安全性测试的方法.

(3) 研究如何针对新一代 DL 框架组件特征进行测试

新一代 DL 框架为适应发展需求, 其内部组件呈现出更加复杂化、多样化的特征. 例如, 计图加入了元算子实现机制, 大幅提升了深度学习任务开发的灵活性. JAX 采用特定领域语言对科学计算相关组件进行优化. 如何针对 DL 框架中新的功能和结构特征设计测试方法也是未来需要面对的研究方向之一.

大模型为通用人工智能提供了解决方法, 但大模型的网络结构、数据参数和计算资源占用十分庞大, 需要分

布式训练提高效率. 分布式训练通过分布式用户接口, 实现模型的分布化, 在执行单节点训练的同时, 可以实现多节点之间的通信协调, 从而加速训练. 微软为支持超大规模模型训练, 基于 PyTorch 开发了新一代 DL 框架 DeepSpeed[120], 其分布式训练相关组件的质量将影响训练效率和成本. 当 DL 框架分布式训练相关 API 和算子优化存在缺陷时, 将导致数据通信效率低和节点失效. 如何针对 DL 框架分布式训练接口和组件进行测试, 也是非常有意义的研究方向.

(4) 研究面向 DL 框架的缺陷预测技术

随着 DL 框架内部代码规模的扩大和复杂度的不断提高, 仅考虑在 DL 框架发布前进行缺陷检测往往工作量大, 成本更高. 运用缺陷预测技术, 研究人员可以根据 DL 框架中提取的度量信息来尽早预测可能存在的缺陷, 基于预测结果可以合理地分配资源, 通过重新设计或修复 DL 框架中的缺陷模块, 提高 DL 框架质量的同时提高框架开发效率. 目前大部分研究仅关注测试方法, 极少涉及针对 DL 框架的缺陷预测研究. 如何将软件缺陷预测方法应用到 DL 框架, 在 DL 框架开发和维护阶段尽早挖掘出缺陷模块是一个非常有意义的研究方向.

传统缺陷预测是在软件开发过程的某个阶段, 对软件模块进行一次性的预测, 但是由于其粒度粗、即时性低和不可追溯等问题, 传统缺陷预测已经难以满足 DL 框架开发对软件质量的需求. 即时缺陷预测是在软件开发过程中, 对每次代码提交进行实时的预测. 目前已有工作针对 DL 框架作即时缺陷预测研究[121], 但仍然面临 DL 框架的相关数据集规模较小, 质量较差和对源代码的关键特征信息提取不足等问题, 导致即时缺陷预测效果较差以及普适性不足. 研究如何更好地将即时缺陷预测技术运用到 DL 框架上是一个非常意义的研究方向.

8 总 结

本文通过对 DL 框架测试研究相关的论文进行充分分析和总结, 主要从 DL 框架缺陷特性实证研究, DL 框架测试关键技术和基于不同测试输入形式的测试方法这 3 个方面对 DL 框架相关研究进行了详细全面的综述, 同时对目前存在的问题进行总结, 对未来研究方向进行展望. 本文主要结论和发现包括以下几个方面.

(1) 研究热度

本文发现, 仅在近 5 年开始有 DL 框架测试研究相关论文发表, 说明目前研究尚处于初期. 近 3 年论文数量有随时间上升的趋势, 并且相当一部分论文发表在软件工程等领域的顶会、顶刊上, 说明 DL 框架测试现在以及将来都是一个备受关注的研究方向.

(2) 研究内容

本文主要研究 DL 框架缺陷特性、测试关键技术和测试方法 3 部分内容. 在了解 DL 框架缺陷特性的基础上, 本文主要关注测试输入生成、测试预言和测试评估这 3 个研究问题, 从基于不同测试输入形式的测试方法角度进行探究, 总结如何运用模糊测试等测试关键技术来解决 DL 框架测试中的上述研究问题.

从实际的缺陷特性分析结果来看, DL 框架的缺陷特性复杂多样, 并且 DL 框架版本更新速度快, 需要更科学细致的实证研究来为 DL 框架测试研究做指导. 目前 DL 框架测试关键技术和测试方法存在一定局限性, 对 DL 框架缺陷的检测和挖掘能力还有待提高, 部分实证研究发现的 DL 框架缺陷类型并没有被现有测试方法充分检测到.

(3) 目前研究的不足和未来研究方向

在大量研究人员的探究下, DL 框架测试研究逐渐成熟, 部分测试技术和测试方法已经得到实际验证. 但值得注意的是, 当前仍然存在一些难点问题亟待解决, 比如目前研究存在测试输入生成能力不足, 针对 DL 框架安全性研究不足, 缺乏针对新一代 DL 框架特征的测试研究等问题.

未来可以研究如何运用大语言模型等新兴技术为 DL 框架测试生成更高质量的输入, 研究如何针对性地采用模糊测试等技术充分挖掘 DL 框架的安全漏洞并对 DL 框架安全性进行评估, 研究如何将即时缺陷预测技术运用到 DL 框架开发和维护过程中等. 期待未来会有更多的研究人员共同努力, 推进 DL 框架不断成熟.

References:

[1] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science, 2006, 313(5786): 504–507. [doi:

10.1126/science.1127647]

[2] Deng L. Artificial intelligence in the rising wave of deep learning: The historical path and future outlook [perspectives]. IEEE Signal Processing Magazine, 2018, 35(1): 180–177. [doi: 10.1109/MSP.2017.2762725]

[3] Abadi M, Agarwal A, Barham P, *et al*. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467, 2016.

[4] Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin ZM, Desmaison A, Antiga L, Lerer A. Automatic differentiation in PyTorch. In: Proc. of the 31st Conf. on Neural Information Processing Systems. Long Beach: NIPS, 2017. 1–4.

[5] Huawei Technologies Co. Ltd. Huawei mindspore AI development framework. In: Proc. of the 2023 Artificial Intelligence Technology. Singapore: Springer, 2023. 137–162. [doi: 10.1007/978-981-19-2879-6_5]

[6] Taigman Y, Yang M, Ranzato MA, Wolf L. DeepFace: Closing the gap to human-level performance in face verification. In: Proc. of the 2014 IEEE Conf. on Computer Vision and Pattern Recognition. Columbus: IEEE, 2014. 1701–1708. [doi: 10.1109/CVPR.2014.220]

[7] Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, Ng AY. Deep speech: Scaling up end-to-end speech recognition. arXiv:1412.5567, 2014.

[8] Fujiyoshi H, Hirakawa T, Yamashita T. Deep learning-based image recognition for autonomous driving. IATSS Research, 2019, 43(4): 244–252. [doi: 10.1016/j.iatssr.2019.11.008]

[9] Ren XZ, Zhou PY, Meng XF, Huang XJ, Wang YD, Wang WC, Li PF, Zhang XD, Podolskiy A, Arshinov G, Bout A, Piontkovskaya I, Wei JS, Jiang X, Su T, Liu Q, Yao J. PanGu-Σ: Towards trillion parameter language model with sparse heterogeneous computing. arXiv:2303.10845, 2023.

[10] The Verge. A Google self-driving car caused a crash for the first time. 2016. https://www.theverge.com/2016/2/29/11134344/google-self-driving-car-crash-report

[11] CVE-2021-43811: Sockeye is an open-source sequence-to-sequence framework for neural machine translation built on PyTorch. Sockeye uses YAML. 2021. https://www.cvedetails.com/cve/CVE-2021-43811/

[12] MathWorks. Deep learning toolbox. 2023. https://www.mathworks.com/products/deep-learning.html

[13] OpenNN: Open neural networks library. 2023. https://www.opennn.net/

[14] Torch: Scientific computing for LuaJIT. 2023. http://torch.ch/

[15] Al-Rfou R, Alain G, Almahairi A, *et al*. Theano: A Python framework for fast computation of mathematical expressions. arXiv:1605.02688, 2016.

[16] Jia YQ, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, Guadarrama S, Darrell T. Caffe: Convolutional architecture for fast feature embedding. In: Proc. of the 22nd ACM Int'l Conf. on Multimedia. Orlando: ACM, 2014. 675–678. [doi: 10.1145/2647868.2654889]

[17] Tokui S, Oono K, Hido S, Clayton J. Chainer: A next-generation open source framework for deep learning. arXiv:1908.00213, 2019.

[18] Caffe2. 2023. http://caffe2.ai/

[19] Seide F, Agarwal A. CNTK: Microsoft's open-source deep-learning toolkit. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 2135. [doi: 10.1145/2939672.2945397]

[20] Chen TQ, Li M, Li YT, Lin M, Wang NY, Wang MJ, Xiao TJ, Xu B, Zhang CY, Zhang Z. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv:1512.01274, 2015.

[21] Keras: The Python deep learning API. 2023. https://keras.io/

[22] Levental M, Orlova E. Comparing the costs of abstraction for DL frameworks. arXiv:2012.07163, 2020.

[23] GitHub. google/jax: Composable transformations of Python+numpy programs: Differentiate, vectorize, JIT to GPU/TPU, and more. 2023. https://github.com/google/jax

[24] Chen JJ, Liang YH, Shen QC, Jiang JJ, Li SC. Toward understanding deep learning framework bugs. ACM Trans. on Software Engineering and Methodology, 2023, 32(6): 135. [doi: 10.1145/3587155]

[25] Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of the 18th Int'l Conf. on Evaluation and Assessment in Software Engineering. London: ACM, 2014. 38. [doi: 10.1145/2601248.2601268]

[26] Hu X, Chen JM, Li HF. Researches on software security vulnerability pattern based on ontology. Journal of Beijing University of Aeronautics and Astronautics, 2022 (in Chinese with English abstract). [doi: 10.13700/j.bh.1001-5965.2022.0783]

[27] GitHub. Let's build from here. 2023. https://github.com/

[28] Stack Overflow. Empowering the world to develop technology through collective knowledge. 2023. https://stackoverflow.co/

[29] CVE. 2023. https://cve.mitre.org/

[30] Long GM, Chen T, Cosma G. Multifaceted hierarchical report identification for non-functional bugs in deep learning frameworks. In: Proc. of the 29th Asia-Pacific Software Engineering Conf. Japan: IEEE, 2022. 289–298. [doi: 10.1109/APSEC57359.2022.00041]

[31] Yang ZC, Yang DY, Dyer C, He XD, Smola A, Hovy E. Hierarchical attention networks for document classification. In: Proc. of the 2016 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. San Diego: Association for Computational Linguistics, 2016. 1480–1489. [doi: 10.18653/v1/N16-1174]

[32] Du XT, Sui YL, Liu ZH, Ai J. An empirical study of fault triggers in deep learning frameworks. IEEE Trans. on Dependable and Secure Computing, 2023, 20(4): 2696–2712. [doi: 10.1109/TDSC.2022.3152239]

[33] Han JW, Kamber M, Pei J. Data Mining: Concepts and Techniques. 3rd ed., Boston: Morgan Kaufmann, 2012. 243–278. [doi: 10.1016/C2009-0-61819-5]

[34] Zar JH. Spearman rank correlation. In: Armitage P, ed. Encyclopedia of Biostatistics. 2nd ed., Hoboken: John Wiley & Sons, Ltd., 2005. 7. [doi: 10.1002/0470011815.b2a15150]

[35] Islam MJ, Nguyen G, Pan R, Rajan H. A comprehensive study on deep learning bug characteristics. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 510–520. [doi: 10.1145/3338906.3338955]

[36] Jia L, Zhong H, Wang XY, Huang LP, Lu XS. The symptoms, causes, and repairs of bugs inside a deep learning library. Journal of Systems and Software, 2021, 177: 110935. [doi: 10.1016/j.jss.2021.110935]

[37] Du XT, Xiao GP, Sui YL. Fault triggers in the TensorFlow framework: An experience report. In: Proc. of the 31st IEEE Int'l Symp. on Software Reliability Engineering. Coimbra: IEEE, 2020. 1–12. [doi: 10.1109/ISSRE5003.2020.00010]

[38] Yang YL, He TX, Xia ZL, Feng Y. A comprehensive empirical study on bug characteristics of deep learning frameworks. Information and Software Technology, 2022, 151: 107004. [doi: 10.1016/j.infsof.2022.107004]

[39] Quan LL, Guo QY, Xie XF, Chen S, Li XH, Liu Y. Towards understanding the faults of JavaScript-based deep learning systems. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 105. [doi: 10.1145/3551349.3560427]

[40] TensorFlow.js. Machine learning for JavaScript developers. 2023. https://www.tensorflow.org/js

[41] Grottke M, Trivedi KS. Software faults, software aging and software rejuvenation(<special survey>new development of software reliability engineering). The Journal of Reliability Engineering Association of Japan, 2005, 27(7): 425–438.

[42] Cao JM, Chen BH, Sun C, Hu LJ, Wu SH, Peng X. Understanding performance problems in deep learning systems. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 357–369. [doi: 10.1145/3540250.3549123]

[43] Makkouk T, Kim DJ, Chen THP. An empirical study on performance bugs in deep learning frameworks. In: Proc. of the 2022 IEEE Int'l Conf. on Software Maintenance and Evolution. Limassol: IEEE, 2022. 35–46. [doi: 10.1109/ICSME55016.2022.00012]

[44] Long GM, Chen T. On reporting performance and accuracy bugs for deep learning frameworks: An exploratory study from github. In: Proc. of the 26th Int'l Conf. on Evaluation and Assessment in Software Engineering. Gothenburg: ACM, 2022. 90–99. [doi: 10.1145/3530019.3530029]

[45] Kloberdanz E, Kloberdanz KG, Le W. DeepStability: A study of unstable numerical methods and their solutions in deep learning. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 586–597. [doi: 10.1145/3510003.3510095]

[46] Tambon F, Nikanjam A, An L, Khomh F, Antoniol G. Silent bugs in deep learning frameworks: An empirical study of Keras and TensorFlow. arXiv:2112.13314, 2021.

[47] Ren Y, Gay G, Kästner C, Jamshidi P. Understanding the nature of system-related issues in machine learning frameworks: An exploratory study. arXiv:2005.06091, 2020.

[48] Liu ZH, Zheng Y, Du XT, Hu Z, Ding WJ, Miao YM, Zheng Z. Taxonomy of aging-related bugs in deep learning libraries. In: Proc. of the 33rd IEEE Int'l Symp. on Software Reliability Engineering. Charlotte: IEEE, 2022. 423–434. [doi: 10.1109/ISSRE55969.2022.00048]

[49] Huang KF, Chen BH, Wu SS, Cao JM, Ma L, Peng X. Demystifying dependency bugs in deep learning stack. arXiv:2207.10347, 2022.

[50] Xiao QX, Li K, Zhang DY, Xu WL. Security risks in deep learning implementations. In: Proc. of the 2018 IEEE Security and Privacy Workshops. San Francisco: IEEE, 2018. 123–128. [doi: 10.1109/SPW.2018.00027]

[51] Chen HS, Zhang YP, Cao YR, Xie J. Security issues and defensive approaches in deep learning frameworks. Tsinghua Science and Technology, 2021, 26(6): 894–905. [doi: 10.26599/TST.2020.9010050]

[52] Harzevili NS, Shin J, Wang JJ, Wang S, Nagappan N. Characterizing and understanding software security vulnerabilities in machine

learning libraries. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Mining Software Repositories. Melbourne: IEEE, 2023. 27–38. [doi: 10.1109/MSR59073.2023.00018]

[53] Filus K, Domańska J. Software vulnerabilities in TensorFlow-based deep learning applications. Computers & Security, 2023, 124: 102948. [doi: 10.1016/j.cose.2022.102948]

[54] Shirey R. RFC4949 Internet security glossary. IETF, 2007. [doi: 10.17487/RFC4949]

[55] Paudice A, Muñoz-González L, Gyorgy A, Lupu EC. Detection of adversarial training examples in poisoning attacks through anomaly detection. arXiv:1802.03041, 2018.

[56] Yerlikaya FA, Bahtiyar Ş. Data poisoning attacks against machine learning algorithms. Expert Systems with Applications, 2022, 208: 118101. [doi: 10.1016/j.eswa.2022.118101]

[57] CWE. Common weakness enumeration. 2023. https://cwe.mitre.org/

[58] Sun XB, Zhou TC, Li GJ, Hu JJ, Yang H, Li B. An empirical study on real bugs for machine learning programs. In: Proc. of the 24th Asia-Pacific Software Engineering Conf. Nanjing: IEEE, 2017. 348–357. [doi: 10.1109/APSEC.2017.41]

[59] Zhang TY, Gao CY, Ma L, Lyu M, Kim M. An empirical study of common challenges in developing deep learning applications. In: Proc. of the 30th IEEE Int'l Symp. on Software Reliability Engineering. Berlin: IEEE, 2019. 104–115. [doi: 10.1109/ISSRE.2019.00020]

[60] Liu JK, Huang Q, Xia X, Shihab E, Lo D, Li SP. Is using deep learning frameworks free?: Characterizing technical debt in deep learning frameworks. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering: Software Engineering in Society. Seoul: ACM, 2020. 1–10. [doi: 10.1145/3377815.3381377]

[61] Niu SJ, Li P, Zhang YJ. Survey on fuzzy testing technologies. Computer Engineering and Science, 2022, 44(12): 2173–2186 (in Chinese with English abstract). [doi: 10.3969/j.issn.1007-130X.2022.12.011]

[62] Boehme M, Cadar C, Roychoudhury A. Fuzzing: Challenges and reflections. IEEE Software, 2021, 38(3): 79–86. [doi: 10.1109/MS.2020.3016773]

[63] Liu JW, Peng JJ, Wang YY, Zhang LM. NeuRI: Diversifying DNN generation via inductive rule inference. arXiv:2302.02261, 2023.

[64] Xie DN, Li YT, Kim M, Pham HV, Tan L, Zhang XY, Godfrey MW. DocTer: Documentation-guided fuzzing for testing deep learning API functions. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM, 2022. 176–188. [doi: 10.1145/3533767.3534220]

[65] Wang Z, Yan M, Chen JJ, Liu S, Zhang DD. Deep learning library testing via effective model generation. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 788–799. [doi: 10.1145/3368089.3409761]

[66] Wei AJ, Deng YL, Yang CY, Zhang LM. Free lunch for testing: Fuzzing deep-learning libraries from open source. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 995–1007. [doi: 10.1145/3510003.3510041]

[67] Barr ET, Harman M, McMinn P, Shahbaz M, Yoo S. The oracle problem in software testing: A survey. IEEE Trans. on Software Engineering, 2015, 41(5): 507–525. [doi: 10.1109/TSE.2014.2372785]

[68] Nejadgholi M, Yang JQ. A study of oracle approximations in testing deep learning libraries. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 785–796. [doi: 10.1109/ASE.2019.00078]

[69] Chen TY, Kuo FC, Liu H, Poon PL, Towey D, Tse TH, Zhou ZQ. Metamorphic testing: A review of challenges and opportunities. ACM Computing Surveys, 2019, 51(1): 4. [doi: 10.1145/3143561]

[70] Ding JH, Kang XJ, Hu XH. Validating a deep learning framework by metamorphic testing. In: Proc. of the 2nd IEEE/ACM Int'l Workshop on Metamorphic Testing. Buenos Aires: IEEE, 2017. 28–34. [doi: 10.1109/MET.2017.2]

[71] McKeeman WM. Differential testing for software. Digital Technical Journal, 1998, 10(1): 100–107.

[72] Pham HV, Lutellier T, Qi WZ, Tan L. CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 1027–1038. [doi: 10.1109/ICSE.2019.00107]

[73] Wang JN, Lutellier T, Qian SS, Pham HV, Tan L. EAGLE: Creating equivalent graphs to test deep learning libraries. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 798–810. [doi: 10.1145/3510003.3510165]

[74] Prochnow A, Yang JQ. DiffWatch: Watch out for the evolving differential testing in deep learning libraries. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. Pittsburgh: ACM, 2022. 46–50. [doi: 10.1145/3510454.3516835]

[75] Goodenough JB, Gerhart SL. Toward a theory of test data selection. ACM SIGPLAN Notices, 1975, 10(6): 493–510. [doi: 10.1145/390016.808473]

[76] Pei KX, Cao YZ, Yang JF, Jana S. DeepXplore: Automated whitebox testing of deep learning systems. In: Proc. of the 26th Symp. on Operating Systems Principles. Shanghai: ACM, 2017. 1–18. [doi: 10.1145/3132747.3132785]

[77] Li MZN, Cao JL, Tian YQ, Li TO, Wen M, Cheung SC. COMET: Coverage-guided model generation for deep learning library testing. ACM Trans. on Software Engineering and Methodology, 2023, 32(5): 127. [doi: 10.1145/3583566]

[78] Luo WS, Chai D, Ruan XY, Wang J, Fang CR, Chen ZY. Graph-based fuzz testing for deep learning inference engines. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 288–299. [doi: 10.1109/ICSE43902.2021.00037]

[79] Jia Y, Harman M. An analysis and survey of the development of mutation testing. IEEE Trans. on Software Engineering, 2011, 37(5): 649–678. [doi: 10.1109/TSE.2010.62]

[80] Jia L, Zhong H, Huang LP. The unit test quality of deep learning libraries: A mutation analysis. In: Proc. of the 2021 IEEE Int'l Conf. on Software Maintenance and Evolution. Luxembourg: IEEE, 2021. 47–57. [doi: 10.1109/ICSME52107.2021.00011]

[81] Wu JW, Li SY, Li JQ, Luo L, Yu HF, Sun G. DeepCov: Coverage guided deep learning framework fuzzing. In: Proc. of the 7th IEEE Int'l Conf. on Data Science in Cyberspace. Guilin: IEEE, 2022. 399–404. [doi: 10.1109/DSC55868.2022.00061]

[82] Gu JZ, Luo XC, Zhou YF, Wang X. Muffin: Testing deep learning libraries via neural architecture fuzzing. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1418–1430. [doi: 10.1145/3510003.3510092]

[83] Kang HJ, Rattanukul P, Haryono SA, Nguyen TG, Ragkhitwetsagul C, Pasareanu C, Lo D. SkipFuzz: Active learning-based input selection for fuzzing deep learning libraries. arXiv:2212.04038, 2022.

[84] Deng YL, Yang CY, Wei AJ, Zhang LM. Fuzzing deep-learning libraries via automated relational API inference. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 44–56. [doi: 10.1145/3540250.3549085]

[85] Deng YL, Xia CS, Peng HR, Yang CY, Zhang LM. Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 423–435. [doi: 10.1145/3597926.3598067]

[86] Deng YL, Xia CS, Yang CY, Zhang SD, Yang SJ, Zhang LM. Large language models are edge-case fuzzers: Testing deep learning libraries via FuzzGPT. arXiv:2304.02014, 2023.

[87] Guo QY, Xie XF, Li Y, Zhang XY, Liu Y, Li XH, Shen C. Audee: Automated testing for deep learning frameworks. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: ACM, 2020. 486–498. [doi: 10.1145/3324884.3416571]

[88] Zou YL, Sun HF, Fang CR, Liu JW, Zhang ZP. Deep learning framework testing via hierarchical and heuristic model generation. Journal of Systems and Software, 2023, 201: 111681. [doi: 10.1016/j.jss.2023.111681]

[89] Shen XZ, Zhang JY, Wang XN, Yu HF, Sun G. Deep learning framework fuzzing based on model mutation. In: Proc. of the 6th IEEE Int'l Conf. on Data Science in Cyberspace. Shenzhen: IEEE, 2021. 375–380. [doi: 10.1109/DSC53577.2021.00059]

[90] Li JQ, Li SY, Wu JW, Luo L, Bai Y, Yu HF. MMOS: Multi-staged mutation operator scheduling for deep learning library testing. In: Proc. of the 2022 IEEE Global Communications Conf. Rio de Janeiro: IEEE, 2022. 6103–6108. [doi: 10.1109/GLOBECOM48099.2022.10001093]

[91] Schumi R, Sun J. ExAIS: Executable AI semantics. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 859–870. [doi: 10.1145/3510003.3510112]

[92] Lipowski A, Lipowska D. Roulette-wheel selection via stochastic acceptance. Physica A: Statistical Mechanics and its Applications, 2012, 391(6): 2193–2196. [doi: 10.1016/j.physa.2011.12.004]

[93] Thompson WR. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika, 1933, 25(3–4): 285–294. [doi: 10.1093/biomet/25.3-4.285]

[94] Lau TA, Weld DS. Programming by demonstration: An inductive learning formulation. In: Proc. of the 4th Int'l Conf. on Intelligent User Interfaces. Los Angeles: ACM, 1998. 145–152. [doi: 10.1145/291080.291104]

[95] Bramer M. Logic Programming with Prolog. 2nd ed., London: Springer, 2013. [doi: 10.1007/978-1-4471-5487-7]

[96] TensorFlow. Module: Tf. keras. layers. TensorFlow v2.11.0. 2023. https://www.tensorflow.org/api_docs/python/tf/keras/layers

[97] Bengio Y. Learning deep architectures for AI. Foundations and Trends® in Machine Learning, 2009, 2(1): 1–127. [doi: 10.1561/2200000006]

[98] Gu DD, Shi YN, Liu XZ, Wu G, Jiang HO, Zhao YS, Ma Y. Defect detection for deep learning frameworks based on meta operators. Chinese Journal of Computers, 2022, 45(2): 240–255 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2022.00240]

[99] Elsken T, Metzen JH, Hutter F. Neural architecture search: A survey. Journal of Machine Learning Research, 2019, 20(55): 1–21.

[100] Roberts F, Tesman B. Introduction to graph theory. In: Roberts F, Tesman B, eds. Applied Combinatorics. 2nd ed., New York: Chapman and Hall/CRC, 2009.

[101] Xu WW, Chen ST, Han GX, Yu N, Xu H. A Monte Carlo tree search-based method for decision making of generator serial restoration sequence. Frontiers in Energy Research, 2023, 10: 1007914. [doi: 10.3389/fenrg.2022.1007914]

[102] TensorFlow. Tf.nn.conv2d: TensorFlow v2.14.0. 2023. https://www.tensorflow.org/api_docs/python/tf/nn/conv2d

[103] Christou N, Jin D, Atlidakis V, Ray B, Kemerlis VP. IvySyn: Automated vulnerability discovery for deep learning frameworks. In: Proc. of the 32nd USENIX Conf. on Security Symp. Anaheim: USENIX Association, 2023. 2383–2400.

[104] Yang CY, Deng YL, Yao JY, Tu YX, Li HC, Zhang LM. Fuzzing automatic differentiation in deep-learning libraries. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 1174–1186. [doi: 10.1109/ICSE48619.2023.00105]

[105] Shi JY, Xiao Y, Li YK, Li YT, Yu DS, Yu CD, Su H, Chen YF, Huo W. ACETest: Automated constraint extraction for testing deep learning operators. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 690–702. [doi: 10.1145/3597926.3598088]

[106] Zhang XF, Sun N, Fang CR, Liu JW, Liu J, Chai D, Wang J, Chen ZY. Predoo: Precision testing of deep learning operators. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 400–412. [doi: 10.1145/3460319.3464843]

[107] Zhang XF, Liu JW, Sun N, Fang CR, Liu J, Wang J, Chai D, Chen ZY. Duo: Differential fuzzing for deep learning operators. IEEE Trans. on Reliability, 2021, 70(4): 1671–1685. [doi: 10.1109/TR.2021.3107165]

[108] Zhao WX, Zhou K, Li JY, *et al*. A survey of large language models. arXiv:2303.18223, 2023.

[109] Chen M, Tworek J, Jun H, *et al*. Evaluating large language models trained on code. arXiv:2107.03374, 2021.

[110] Beyer HG, Schwefel HP. Evolution strategies—A comprehensive introduction. Natural Computing, 2002, 1(1): 3–52. [doi: 10.1023/A:1015059928466]

[111] Fried D, Aghajanyan A, Lin J, Wang SD, Wallace E, Shi F, Zhong RQ, Yih W, Zettlemoyer L, Lewis M. InCoder: A generative model for code infilling and synthesis. In: Proc. of the 11th Int'l Conf. on Learning Representations. Kigali: OpenReview.net, 2023.

[112] Cambronero JP, Dang THY, Vasilakis N, Shen JS, Wu J, Rinard MC. Active learning for software engineering. In: Proc. of the 2019 ACM SIGPLAN Int'l Symp. on New Ideas, New Paradigms, and Reflections on Programming and Software. Athens: ACM, 2019. 62–78. [doi: 10.1145/3359591.3359732]

[113] Zaki MJ. TREEMINER: An efficient algorithm for mining embedded ordered frequent trees. In: Bandyopadhyay S, Maulik U, Holder LB, Cook DJ, eds. Advanced Methods for Knowledge Discovery from Complex Data. London: Springer, 2005. 123–151. [doi: 10.1007/1-84628-284-5]

[114] Chiang WF, Gopalakrishnan G, Rakamaric Z, Solovyev A. Efficient search for inputs causing high floating-point errors. In: Proc. of the 19th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. Orlando: ACM, 2014. 43–52. [doi: 10.1145/2555243.2555265]

[115] Yin PC, Li WD, Xiao KF, Rao A, Wen YM, Shi KS, Howland J, Bailey P, Catasta M, Michalewski H, Polozov O, Sutton C. Natural language to code generation in interactive data science notebooks. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics. Toronto: Association for Computational Linguistics, 2023. 126–173. [doi: 10.18653/v1/2023.acl-long.9]

[116] Fan ZY, Gao X, Mirchev M, Roychoudhury A, Tan SH. Automated repair of programs from large language models. In: Proc. of the 45th Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 1469–1481. [doi: 10.1109/ICSE48619.2023.00128]

[117] White J, Hays S, Fu QC, Spencer-Smith J, Schmidt DC. ChatGPT prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. arXiv:2303.07839, 2023.

[118] Le Quoc D, Gregor F, Arnautov S, Kunkel R, Bhatotia P, Fetzer C. SecureTF: A secure TensorFlow framework. In: Proc. of the 21st Int'l Middleware Conf. Delft: ACM, 2020. 44–59. [doi: 10.1145/3423211.3425687]

[119] LatticeX-foundation/rosetta: A privacy-preserving framework based on TensorFlow. 2023. https://github.com/LatticeX-Foundation/Rosetta

[120] Extreme speed and scale for DL training and inference. 2023. https://github.com/microsoft/DeepSpeed

[121] Ge J, Yu HQ, Fan GS, Tang JH, Huang ZJ. Just-in-time defect prediction for intelligent computing frameworks. Ruan Jian Xue Bao/Journal of Software, 2023, 34(9): 3966–3980 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6874.htm [doi: 10.13328/j.cnki.jos.006874]

附中文参考文献:

[26] 胡璇, 陈俊名, 李海峰. 基于本体的软件安全漏洞模式研究. 北京航空航天大学学报, 2022. [doi: 10.13700/j.bh.1001-5965.2022.0783]

[61] 牛胜杰, 李鹏, 张玉杰. 模糊测试技术研究综述. 计算机工程与科学, 2022, 44(12): 2173–2186. [doi: 10.3969/j.issn.1007-130X.2022.

12.011]

[98] 谷典典, 石屹宁, 刘譞哲, 吴格, 姜海鸥, 赵耀帅, 马郓. 基于元算子的深度学习框架缺陷检测方法. 计算机学报, 2022, 45(2): 240–255. [doi: 10.11897/SP.J.1016.2022.00240]

[121] 葛建, 虞慧群, 范贵生, 唐锏浩, 黄子杰. 面向智能计算框架的即时缺陷预测. 软件学报, 2023, 34(9): 3966–3980. http://www.jos.org.cn/1000-9825/6874.htm [doi: 10.13328/j.cnki.jos.006874]

马祥跃(1997－), 男, 博士生, 主要研究领域为智能软件测试.

郑阳(1991－), 男, 博士, 高级工程师, 主要研究领域为人工智能系统的测试, 监测和修复研究, 数据驱动测试.

杜晓婷(1990－), 女, 博士, 讲师, CCF 专业会员, 主要研究领域为智能软件测试, 软件仓库挖掘, 缺陷预测.

胡峭(1981－), 男, 博士, 高级工程师, CCF 专业会员, 主要研究领域为可信人工智能, 软件可靠性.

采青(2000－), 女, 硕士生, 主要研究领域为软件测试.

郑征(1980－), 男, 博士, 教授, CCF 专业会员, 主要研究领域为人工智能软件系统的可靠性及测试方法.