软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(7):3162−3179 [doi: 10.13328/j.cnki.jos.007106]

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

基于多模态表征的移动应用 GUI 模糊测试框架*

张少坤[1], 李元春[2], 雷瀚文[1], 蒋 鹏[1], 李 锭[1], 郭 耀[1], 陈向群[1]

[1](北京大学 计算机学院, 北京 100871)

[2](清华大学 智能产业研究院, 北京 100084)

通信作者: 郭耀, E-mail: yaoguo@pku.edu.cn

摘 要: GUI 模糊测试在提升移动应用可靠性和兼容性方面发挥着关键作用. 然而, 现有的 GUI 模糊测试方法大多效率较低, 主要原因是这些工作过于粗粒度, 仅基于单一模态的特征来整体理解 GUI 页面, 应用状态的过度抽象使得许多细节信息被忽略, 导致对 GUI 状态及小部件的理解不足. 为了解决上述问题, 提出了一种基于多模态表征的移动应用 GUI 模糊测试框架 GUIFuzzer. 该框架通过考虑多模态特征, 如视觉特征、布局上下特征和细粒度的元属性特征, 来联合推断 GUI 小部件的语义; 然后, 训练一个多层次奖励驱动的深度强化学习模型来优化 GUI 事件选择策略, 提高模糊测试的效率. 在大量的真实应用上对所提框架进行了评估. 实验结果表明: 与现有的竞争性基线相比, GUIFuzzer 显著地提升了模糊测试的覆盖率. 还对特定目标的定制化搜索即敏感 API 触发进行了案例研究, 进一步验证了 GUIFuzzer 框架的实用性.

关键词: GUI 模糊测试; 强化学习; 深度学习; 多模态表征; 定制化搜索

中图法分类号: TP311

中文引用格式: 张少坤, 李元春, 雷瀚文, 蒋鹏, 李锭, 郭耀, 陈向群. 基于多模态表征的移动应用 GUI 模糊测试框架. 软件学报, 2024, 35(7): 3162–3179. http://www.jos.org.cn/1000-9825/7106.htm

英文引用格式: Zhang SK, Li YC, Lei HW, Jiang P, Li D, Guo Y, Chen XQ. GUI Fuzzing Framework for Mobile Apps Based on Multi-modal Representation. Ruan Jian Xue Bao/Journal of Software, 2024, 35(7): 3162−3179 (in Chinese). http://www.jos.org.cn/1000-9825/7106.htm

GUI Fuzzing Framework for Mobile Apps Based on Multi-modal Representation

ZHANG Shao-Kun[1], LI Yuan-Chun[2], LEI Han-Wen[1], JIANG Peng[1], LI Ding[1], GUO Yao[1], CHEN Xiang-Qun[1]

[1](School of Computer Science, Peking University, Beijing 100871, China)

[2](Institute for AI Industry Research, Tsinghua University, Beijing 100084, China)

Abstract: GUI fuzzing plays a crucial role in enhancing the reliability and compatibility of mobile apps. However, most existing GUI fuzzing methods are inefficient, mainly because they are coarse-grained, relying solely on single-modal features to understand the GUI pages holistically. The excessive abstraction of app states leads to the neglect of many details, resulting in an insufficient understanding of GUI states and widgets. To address this issue, a GUI fuzzing framework called GUIFuzzer for mobile apps is proposed based on multi-modal representation. This framework leverages multi-modal features, such as visual features, layout context features, and fine-grained meta-attribute features, to jointly infer the semantics of GUI widgets. Then, it trains a multi-level reward-driven deep reinforcement learning model to optimize the GUI event selection strategy, thus improving the efficiency of fuzz testing. The proposed framework is evaluated on a large number of real apps. Experimental results show that GUIFuzzer significantly improves the coverage of fuzz testing compared with existing competitive baselines. A case study is also conducted on customized search for specific targets, namely sensitive API triggering, which further demonstrates the practicality of the GUIFuzzer framework.

* 基金项目: 国家自然科学基金(62141208)

本文由“面向复杂软件的缺陷检测与修复技术”专题特约编辑张路教授、刘辉教授、姜佳君副研究员、王博博士推荐.

收稿时间: 2023-09-09; 修改时间: 2023-10-30; 采用时间: 2023-12-14; jos 在线出版时间: 2024-01-05

CNKI 网络首发时间: 2024-03-09

Key words: GUI fuzzing; reinforcement learning; deep learning; multi-modal representation; customized search

移动设备如智能手机已经成为大多数人日常生活的必需品. 移动应用则提供了丰富的功能和服务, 涵盖了各个领域, 包括社交媒体、电子商务、金融服务、医疗保健等. 为了确保应用的质量和稳定性, 开发者需要对应用进行充分测试. 然而, 人工测试通常是费力、耗时且易错的, 移动应用需要健壮、可靠的自动化测试解决方案. GUI 模糊测试(GUI fuzzing)作为一种常用的自动化测试方法, 以用户的角度对移动应用进行测试, 模拟出各种可能的用户输入和操作. 这种测试方法不仅可以发现应用中潜在的错误和安全漏洞, 还能够评估应用在真实使用情况下的稳定性和可靠性. 此外, 由于移动设备的多样性和操作系统的碎片化, 不同设备以及不同版本操作系统可能导致应用在不同环境下的表现有所差异, 而 GUI 模糊测试可以帮助开发人员尽早发现问题并解决, 从而提高应用的兼容性和稳定性.

现有的 GUI 模糊测试工作主要分为基于随机策略、基于模型策略、基于精细化策略以及基于机器学习策略的方法. 对于随机 GUI 模糊测试方法[1], 其原理较为简单, 缺少对应用的深入分析, 这类策略存在测试事件大量冗余、测试时间分配不均的问题. 基于模型的 GUI 模糊测试方法[2−4]对模型要求高, 然而在模型构建过程中普遍存在模型与实际行为不统一的问题, 对测试结果产生影响. 基于精细化策略的方法[5,6]着重对较难触发的代码进行测试, 在整体代码覆盖度及缺陷发现方面表现一般. 此外, 符号执行等技术的使用带来较大的资源和时间开销, 在有限测试时间内难以取得较优效果. 近年来, 研究者们尝试采用机器学习方法[7−17]进行 GUI 模糊测试. 例如: Li 等人[7]提出了 Humanoid, 它利用深度学习模型从人类历史交互轨迹中学习经验知识, 然后利用学习到的知识指导测试输入生成; Zhang 等人[13]提出一个好奇心驱动的强化学习框架 UniRLTest, 用于指导探索不熟悉的功能. 然而, 它们大多是粗粒度的(见表 1), 仅基于单模态特征(如只使用 GUI 截图)来整体理解 GUI 页面, 而应用状态的过度抽象使许多细节信息被忽略, 导致对 GUI 状态与小部件理解不够充分, 从而影响了测试效率.

表 1　与代表性工作的比较

方法	会议/期刊	GUI 截图	布局	小部件属性
Monkey	–	×	×	×
Droidbot[2]	ICSE 2017	×	√	×
Humanoid[7]	ASE 2019	√	×	×
Q-testing[8]	ISSTA 2020	×	√	×
Xue, *et al*.[9]	ISSTA 2020	√	×	×
Deep GUI[10]	ASE 2021	√	×	×
WebExplor[11]	ICSE 2021	×	√	×
ARES[12]	TOSEM 2022	×	√	×
RELINE[13]	ICSE 2022	√	×	×
UniRLTest[14]	ISSTA 2022	√	×	△
GUIFuzzer (ours)	–	√	√	√

注: √表示考虑, ×表示未考虑, △表示部分考虑, –表示未发表

实现高效的模糊测试主要面临的挑战有:

(1) 难以理解 GUI 小部件的行为意图. 由于 GUI 图像中存在大量噪声和不确定性, 如小部件状态变化(例如按钮状态从禁用变为启用), 仅基于单模态信息难以推断出 GUI 小部件可能具有行为意图, 如选择、导航、输入等.

(2) GUI 小部件的测试偏好不清楚. 不同小部件对测试目标发现的贡献度不同.

(3) 缺乏训练数据. 在移动应用的复杂场景下, 提取有效的特征及建模复杂的关系通常需要大量样本.

针对上述问题, 本文提出了一种基于多模态表征的移动应用 GUI 模糊测试框架 GUIFuzzer. 该方法受启发于 GUIDER[18], 它利用多模态特征, 能够有效地修复 Android 应用程序的测试脚本. 为了解决第 1 个挑战, 我们首先提出了基于多模态的细粒度 GUI 表征来联合推断 GUI 小部件的语义. 为了解决第 2 个挑战, 我们提出利用深度学习模型从大量样本中自动地学习与测试目标相关的偏好. 为了解决第 3 个挑战, 我们提出利用强化学习代理自主生成训练数据, 再将其集成到学习算法中, 并且利用预训练模型来提取有效鲁棒的特征. 需要注意的是: 我们并不是直接从多模态表征中提取出小部件的行为意图, 而是通过一个深度强化学习模型

来学习如何利用这些特征来生成合适的测试动作.

GUIFuzzer 的核心思路是: 通过考虑多模态特征, 如视觉特征、布局上下特征以及细粒度的元属性特征, 来联合推断 GUI 小部件的语义, 然后训练一个多层次奖励驱动的深度强化学习模型来提高模糊测试的效率, 该多层次奖励主要用于驱动代理探索更多新的 GUI 状态以及发现更多测试目标. 我们通过大量实验评估所提框架 GUIFuzzer 的性能. 我们使用来自小米应用市场的 121 个真实应用来评估框架的性能. 实验结果表明: 与现有的竞争性基线相比, GUIFuzzer 显著地提升了模糊测试的覆盖率. 多模态特征对模型性能的有效性影响从低到高依次排序为: 视觉特征、布局上下文特征和细粒度元属性特征. 我们还对特定目标的定制化搜索即敏感 API 触发进行了案例研究, 进一步验证了 GUIFuzzer 框架的实用性.

我们的研究贡献如下:

(1) 我们提出利用多模态特征来联合推断 GUI 小部件的语义, 其中, 多模态特征包括视觉特征、布局上下文特征以及元属性特征.

(2) 我们提出一种基于多模态表征的移动应用 GUI 模糊测试框架 GUIFuzzer, 该框架通过一个多层次奖励驱动的深度强化学习模型来学习如何利用多模态特征生成合适有效的测试动作.

(3) 我们在大量真实应用上对 GUIFuzzer 进行评估. 实验结果表明: 与现有竞争性基线相比, GUIFuzzer 显著地提升了模糊测试的覆盖率.

(4) 我们还对 GUIFuzzer 在特定目标的定制化搜索方面, 即敏感 API 触发, 进行了案例研究. 结果表明: 我们的框架比竞争性基线发现更多的敏感 API 调用, 这也验证了我们框架的有用性.

本文第 1 节介绍 GUI 模糊测试的相关方法和研究现状. 第 2 节介绍本文所需的背景知识, 包括 Android GUI 编程以及深度强化学习. 第 3 节介绍研究动机. 第 4 节阐述本文提出的基于多模态表征的移动应用 GUI 模糊测试框架. 第 5 节通过实验验证框架的有效性. 第 6 节展示 GUIFuzzer 的相关讨论. 最后总结全文.

1 GUI 模糊测试相关工作

根据测试策略, 现有的移动应用 GUI 模糊测试工作主要可分为基于随机策略、基于模型策略、基于精细化策略以及基于机器学习策略的方法.

- 基于随机策略的 GUI 模糊测试

随机模糊测试方法采用伪随机的方式生成事件流对应用程序进行测试. 一个典型代表是 Monkey, 它是由 Google 开发的随机测试工具, 包含在 Android SDK 中. Monkey 支持广泛的用户和系统事件, 不需要任何系统的先验知识或理解. 它的目标是通过向被测应用或系统提供随机输入来发现潜在的安全漏洞或缺陷. 它在屏幕上的随机位置生成用户事件, 而不考虑给定位置是否存在小部件以及该小部件是否存在相应地事件处理程序. 其简单的设计使 Monkey 的事件生成速度非常快, 因为生成新事件时几乎没有任何决策. 然而, 这种策略的缺点也非常明显: 测试过程中可能会生成大量无效的事件(例如对屏幕中的空白处点击)或者冗余事件(例如点击两个不同坐标位置执行同一事件处理程序). Dynodroid[1]是基于 Monkey 改进的, 其主要思想是采用一种“观察-选择-执行”循环. 首先, 它观察当前应用程序状态下的可执行事件; 然后, 选择交互事件并执行以生成新状态; 最后, 重复以上过程. 对于 UI 事件, Dynodroid 通过程序的用户界面识别其相关动作. 对于系统事件, Dynodroid 通过插桩识别应用各状态下注册的事件监听器函数. 对于文本输入生成, Dynodroid 允许用户与目标应用程序交互, 用户可暂停测试过程, 手动生成所需文本输入. 随机方法具有策略简单、无偏性、实用性强等特点, 它可以在不需要明确测试方案的情况下快速地进行测试, 在有限的时间内尽可能多地测试系统. 此外, 一些研究[19]不考虑动态探索被测应用, 而是针对应用程序间的通信或构件间的通信进行测试. 例如, IntentFuzzer[19]利用模糊测试来生成不同的意图消息, 然后动态地将消息发送到公开接口来检测应用中可能存在的安全漏洞. 事实上, 这些漏洞主要是由开发者在实现功能时不安全处理意图通信消息引起的. 然而, 由于 IntentFuzzer 等工具并未采用模拟用户交互的方式测试应用, 因此测试中发现的问题在应用的实际使用过程中一般并不会发生. 然而, 随机生成的事件测试覆盖率低, 冗余程度高. 如果能提高其他工具测试事件的生成效

率, 那 Monkey 的相对优势就会有所减弱.

- 基于模型的 GUI 模糊测试

为了提高覆盖率, 一些研究[2−4]采用基于模型的策略动态探索被测应用. Stoat[3]是一种基于随机有限状态机的移动应用 GUI 模糊测试方法, 其思想是: 通过 GUI 模型充分测试应用程序, 并通过强制执行各种用户或系统交互验证应用程序的行为. Stoat 采用两阶段流程来测试应用程序.

(1) 模型构建. Stoat 首先从应用程序生成个随机有限状态机来描述应用的 GUI 交互, 然后利用动态分析技术探索应用行为. 在探索过程中, Stoat 记录所有 UI 事件的执行频率, 并根据它们生成初始的转移概率值.

(2) 模型突变、测试生成和执行. Stoat 迭代地改变随机模型, 并从模型变体中生成测试. Stoat 使用概率方法生成各种事件序列, 以充分测试 GUI 交互, 进而探索更多的未执行路径.

PUMA[4]是一个可编程的 Android 应用程序 UI 自动化框架. 利用 PUMA, 可编写研究应用程序行为的“分析”脚本, 它会自动运行每个应用程序并收集用于分析的信息. PUMA 首先解释 PUMAScript 识别脚本中的 Monkey 指令和应用程序指令部分: 前者指令提供了 Monkey 工具执行应用程序的必要输入; 后者规定应用程序哪些代码部分与分析相关, 并指定在执行这些代码片段时需采取的行动. ORBIT[20]是一种灰盒方法, 它能够自动地从目标应用程序中提取模型. 它首先利用静态分析技术识别应用程序 GUI 所支持的各种事件, 然后执行动态爬虫, 通过在目标应用上执行提取的事件系统地探索应用 GUI 的各个界面.

DroidBot[2]是一个轻量级的模糊测试工具, 它的优越性包含以下方面.

(1) 无须修改操作系统或对应用进行插桩. 为了获得动态生成测试用例所需的信息, 很多现有的动态测试工具需要修改系统源码或对应用进行插桩, 而 DroidBot 收集的信息无需对环境进行任何修改.

(2) 基于在线建立的 GUI 模型生成测试用例. 随机生成的测试覆盖率上升速度较不理想, 静态分析生成的测试用例中无效和重复比率较高, 且无法覆盖应用动态生成的内容, 例如动态生成的菜单和界面等. DroidBot 在运行时会根据发送的测试用例和已经到达过的界面动态生成一个 GUI 模型, 根据该模型, 可推断导向未探索界面的测试用例.

(3) 可收集测试过程中的 UI 状态和相关代码. DroidBot 在运行过程中会自动记录应用的 UI 状态, 包括截图和层次结构信息.

DroidDEV[21]通过动态构建多个有限状态流图来全面探索应用程序. DroidDEV 具有以下几个特性.

(1) 自终止. 它采用了一种自终止的探索策略, 可以有效避免在有限状态流图中出现重复探索的情况.

(2) 自动化. 它能够根据 UI 上下文生成合适的文本输入实现完全自动化的测试过程, 无须人工干预或预定义输入.

(3) 有效. 它通过对每个 UI 状态上的所有小部件进行操作, 系统地覆盖了应用程序的所有可能状态和转换.

(4) 高效. 它采用了一种贪婪的搜索算法即最佳优先搜索发现到达目标 GUI 状态的最短路径, 从而降低了探索步数和时间.

- 基于精细化策略的 GUI 模糊测试

精细化策略(systematic strategies)采用符号执行、遗传算法、约束求解等技术对部分较难覆盖的代码生成测试序列. Sapienz[5]使用基于多目标搜索的算法来自动测试 Android 应用程序. 它结合随机、精细化以及基于搜索的策略探索应用 GUI 状态, 在探索过程中采用随机字符串种子和多级插桩技术. 其核心思想是: 利用遗传算法最小化生成事件序列的长度来优化测试性能, 同时最大限度地提高代码覆盖率和故障检测率. 通过利用帕累托最优性(Paretol-optimal), Sapienz 可安全地用同等质量的较短事件序列替换长事件序列. 然而, 遗传算法的稳定性较差, 需要耗费大量的时间. A^3E-Targeted[6]是一种测试技术, 系统地探索 Android 应用程序, 且无须依赖源代码. 该技术的关键思想是, 使用静态的数据流分析应用程序字节码. 它实现了定向探索测试技术. 定向探索是一种利用静态污点分析来有效探索应用程序活动的技术, 能够直接调用特殊的应用程序活动,

其目的是进行快速的活动探索, 从而实现高活动覆盖率.为了实现定向探索, A^{3}E-Targeted 分析静态字节码提取静态活动转换图. 总体而言, 精细化的测试方法适合针对系统的特定代码部分进行深入测试, 而非对整个系统进行全面测试. 因此, 在安全漏洞发现以及整体代码覆盖率方面, 其他 GUI 模糊测试方法可能更具优势.

- 基于机器学习的 GUI 模糊测试

近来, 机器学习技术也被应用于移动应用 GUI 模糊测试[7−17]. QBE[15]是一种基于 Q 学习的探索方法, 它以最大化代码或 GUI 覆盖率为目标, 从历史的探索经验中学习如何测试应用程序. 然而, 由于 Q 学习的局限性, QBE 无法捕获细粒度的应用程序行为. SwiftHand[22]利用机器学习算法, 根据目标应用程序的行为, 学习一个近似的用户界面模型, 然后利用该模型系统地探索应用程序的状态空间, 并在运行过程中不断改进模型. SwiftHand 通过动态地学习和更新模型来减少应用程序重启的频率, 并在探索应用程序状态时尽快达到高代码覆盖率. Humanoid[7]是一种基于深度学习的自动化 Android 应用 GUI 模糊测试方法, 它能够学习人类如何与移动应用程序交互, 然后使用学习到的模型作为人类测试人员来指导测试生成. 利用从人类交互痕迹中学习到的知识, Humanoid 可以根据 GUI 页面上可能的交互的重要性和意义来对其进行优先级排序, 从而生成能够更快地生成重要状态的测试输入. 此外, 还有研究者[23−25]尝试先构建一个预训练行为模型, 然后针对被测应用进行微调. 它们之间最主要的区别是代理奖励函数的设计方式不同. 例如, 在 AutoBlackTest[26]中, 奖励函数由两部分组成: (1) 事件执行前后两个状态的差异; (2) 事件执行的频数. 然而, 该奖励函数过分强调前者, 容易导致在差异较大的两个状态间不断切换, 从而浪费测试时间. Huang 等人[8]提出采用好奇心驱动的状态探索策略, 通过一个记忆集合保存部分已探索到的状态, 以此来引导测试工具对不熟悉的状态进行探索. 该策略能够有效解决随机方法测试时间分配不均的问题, 同时避免了测试过程中对模型的过度依赖. 此外, 还有些研究将强化学习技术应用于其他测试工作中. Retecs[16]在回归测试中使用强化学习自适应地选择测试用例并确定其优先级. Wuji[17]将深度强化学习与进化算法相结合, 以自动地测试在线格斗游戏, 其中, 进化算法用于生成测试用例, 深度强化学习技术用于学习游戏策略.

2 背景知识

本节主要展示相关的背景知识, 其中包括 AndroidGUI 编程与深度强化学习.

2.1 Android GUI编程

在 Android 中, 活动(activity)是应用程序的基本构建块之一, 它代表了一个用户界面屏幕或窗口. 一个 Android 应用程序可以由多个活动组成, 每个活动可以显示一个 GUI 界面. 通常, 一个应用程序由一个主活动和其他辅助活动组成. GUI 小部件是构建用户界面的元素, 例如按钮、文本框、复选框、单选按钮等. Android 提供了丰富的 GUI 小部件库, 开发者可通过 XML 布局文件或 Java 代码来定义和操作它们.

GUI 窗口是活动的可视化表示, 用于呈现 GUI 界面并接受用户输入. 活动间的转换可通过意图(Intent)来实现. 意图是一种在不同活动间传递数据和触发操作的机制. 通过创建和启动意图, 开发者可在不同的活动间进行切换和通信. GUI 转换图是一种描述应用程序功能和行为的图形化表示方法. 正式地, 我们将其定义为一个有向图 G=(V,E), 其中, V 是节点集合, 表示应用程序内各活动或屏幕; E 是图中边的集合, 表示活动间可能的转换或动作. 它概述了应用程序的结构和流程, 能清晰地展示用户如何在不同活动间的操作流程以及哪些动作或事件可能触发活动转换. 该图在自动化测试中可帮助测试人员更好地理解和设计测试用例.

2.2 深度强化学习

深度强化学习是一种结合深度学习和强化学习的技术, 用于解决智能体在未知环境中学习并根据反馈做出决策的问题. 它的目标是学习最优策略, 使智能体在面对未知环境或任务时能够做出正确的决策, 最大化未来累积奖励. 深度 Q 网络是深度强化学习中的一个重要算法, 它使用深度神经网络来近似 Q 函数, 并解决了传统 Q 学习在高维状态空间下存储和更新 Q 值的问题. 神经网络接收环境的当前状态作为输入, 通过前向传播计算为每个可能动作的 Q 值. 代理通过在每一步选择具有最高 Q 值的动作来执行. 深度 Q 网络的训练过

程如下.

(1) 初始化深度 Q 网络的权重和偏置, 以及目标网络的参数. 两个网络的架构相同, 但目标网络的参数更新频率较低.

(2) 智能体与环境交互, 存储样本元组包括状态、动作、奖励以及下一个状态.

(3) 将元组放入经验回放缓冲池中. 经验回放缓冲池用于存储智能体的历史经验, 并从中随机选择样本进行训练. 其好处是打破样本间的相关性, 提高训练的稳定性和效率.

(4) 从经验回放缓冲池中随机选择一批样本用于更新深度 Q 网络的参数. 通过最小化当前 Q 值和目标 Q 值之间的差异来更新网络参数. 目标 Q 值的计算需要使用目标网络来估计下一个状态的最大 Q 值.

(5) 定期更新目标网络的参数, 将当前深度 Q 网络的参数复制给目标网络. 如此可减少训练过程中的波动和不稳定性.

(6) 重复步骤(2)–步骤(5), 不断训练深度 Q 网络, 直到达到预设的停止条件.

3 研究动机

GUI 模糊测试是一种常用的技术, 可以发现软件系统中的潜在漏洞和错误. 然而, 现有的 GUI 模糊测试方法存在效率低下的问题. 例如: 传统的 GUI 模糊测试方法只是随机地生成输入序列, 而不考虑 GUI 内小部件的含义及其关系, 这会造成大量无效或冗余的事件(例如对屏幕中没有响应的区域进行点击)以及测试时间分配不均等问题, 从而降低了测试覆盖率和错误检测能力. 因此, 迫切需要一种高效的方法, 能够根据 GUI 的特定和测试目标, 有针对性地生成有效的输入序列, 从而提高测试效率和发现潜在漏洞的能力. 我们的目标是提出一种 GUI 模糊测试框架, 该框架能够根据测试目标动态地调整 GUI 事件的策略, 从而优化测试过程.

- 动机示例

为了说明, 我们展示了一个动机示例, 其中包含两个来自不同应用的屏幕截图, 分别是 TED 和 AZLyrics. 如图 1 所示, TED 是一个提供各领域专家演讲的平台, 而 AZLyrics 是一个拥有海量歌词资源的应用. 图 1(a)展示了一个演讲视频的详情页, 而图 1(b)展示了一个歌词搜索的功能页面. 在图 1(a)中, 由橙色和浅蓝色框标识的 GUI 小部件所表示的含义令人困惑. 仅基于 GUI 截图的理解是粗粒度的, 难以推断出小部件的语义. 为了更细粒度地理解, 我们需要深入到小部件层面. 小部件有许多元属性, 它们揭示了小部件的意图、外观和功能. 在图 1(a)中, 我们用不同颜色框标记的 GUI 小部件, 它们的语义可通过从其“resource_id”属性中提取的文本来辅助理解, 例如“Queue”“Like”“Share”和“Download”. 尽管这些属性很有用, 但它们并不总是可用的. 根据以往研究报告[27,28], 由于程序员不规范的开发行为, 仍有许多 GUI 小部件缺少属性或其属性不正确. 如图 1(b)所示, 输入框的“text”属性值缺失, “resource_id”属性值为“eac-5300”, 没有任何实际意义. 这表明有必要根据上下文推断 GUI 小部件的语义, 例如小部件周围的描述性文本. 在本例中, 与输入框相关的文本描述是“Enter artist name or song title”.

- 基本思想

我们提出利用多模态特征细粒度地理解 GUI 小部件语义. 这些特征包括但不限于以下几个方面.

(1) 视觉特征: 通过对 GUI 界面截图, 并利用计算机视觉方法从图像中提取抽象语义特征.

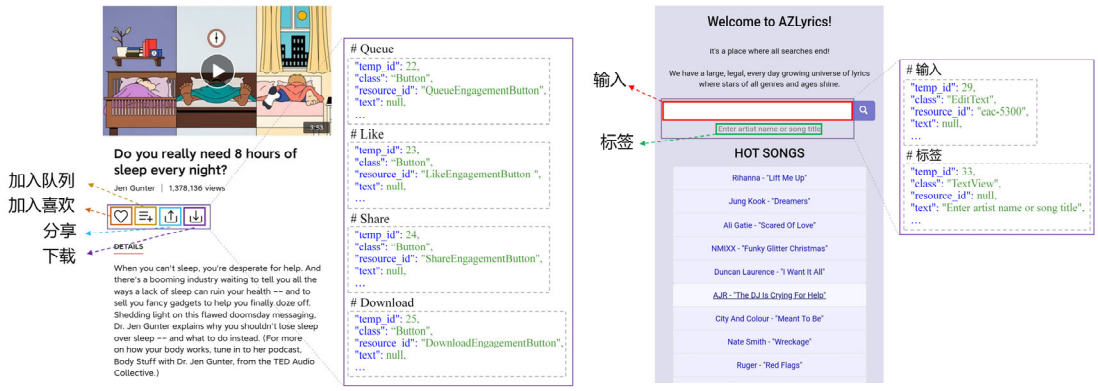
(2) 布局上下文特征: 通过从 GUI 页面的层次结构中提取布局特征, 以反映小部件的上下文.

(3) 小部件属性特征: 通过从小部件自身属性, 如文本标签、类型、坐标等中提取其元属性特征, 以描述小部件的功能和用途.

正式地, 假设我们有一个 GUI 页面 S, 其中包含多个小部件. 给定一个小部件 w_i, 我们可定义其属性集合为 $P(w_i)=\{p_i^1,p_i^2,...,p_i^n\}$, 其中, p_i^j 表示小部件 w_i 的第 j 个属性. 那么当前页面 S 的属性集合为 $P(S)=\{P(w_1), P(w_2),\ldots,P(w_m)\}$, 其中, m 表示页面 S 中小部件的数量. 为了融合多模态特征, 我们定义一个特征融合函数 $C(S)$, 其中, $C(S)$可表示为

$$C(S)=concat(F_{\text{img}}(S),F_{\text{layout}}(S),F_{\text{attr}}(S),F_{\text{oth}}(S)) \tag{1}$$

其中, $F_{\text{img}}(\cdot)$表示图像特征提取函数, 例如基于 CNN 的深度学习模型; $F_{\text{layout}}(\cdot)$表示布局特征提取函数, 例如自编码器; $F_{\text{attr}}(S)$表示从 GUI 界面 S 中提取的元属性特征; $F_{\text{oth}}(S)$表示可能考虑的其他多模态特征, $concat(\cdot)$表示特征拼接操作.



(a) TED (b) AZLyrics

图 1 基于 TED 和 AZLyrics 应用的动机示例

需要特别说明的是: 小部件的元属性特征, 因为不同的属性类型可能有不同的处理方式. 正式地, 我们定义小部件元属性特征如下.

$$F_{\text{attr}}(S)=\bigoplus_{i=1}^{m}\bigoplus_{j=1}^{n}g(w_i,p_i^j) \tag{2}$$

其中, $\oplus$表示连接操作. $g(w_i,p_i^j)$ 表示根据属性类型提取特征的函数, 定义如下.

$$g(w_i,p_i^j)=\begin{cases}h_{\text{text}}(p_i^j), & p_i^j\text{是文本}\\ h_{\text{type}}(p_i^j), & p_i^j\text{是类型}\\ h_{\text{num}}(p_i^j), & p_i^j\text{是数值}\end{cases} \tag{3}$$

其中, h_{text}、h_{type}、h_{num} 分别表示文本、类型以及数值属性的特征提取函数.

4 基于多模态表征的移动应用 GUI 模糊测试框架 GUIFuzzer

为了提高测试效率, 本文提出了一个基于多模态表征的移动应用 GUI 模糊测试框架 GUIFuzzer, 它能够利用深度强化学习模型根据不同测试目标自动确定小部件的优先级. 该框架的输入是一个 APK 文件, 输出是一个与测试目标相关的详细测试报告. 图 2 展示了 GUIFuzzer 框架. 该框架主要包括两个模块, 即多模态融合和多层次奖励. 多模态融合模块主要通过考虑融合多种模态特征如视觉特征、布局上下特征以及细粒度的元属性特征来联合推断 GUI 小部件的行为意图, 例如导航、输入、选择等. 多层次奖励模块用于指导强化学习代理从大量样本中自动地学习与测试目标相关的偏好. 基于这种偏好, 代理能够生成更有针对性的测试用例, 从而提高测试覆盖率和效率.

实现高效的 GUI 模糊测试主要面临以下挑战.

- 第 1 个挑战是如何理解 GUI 小部件的行为意图. 小部件在不同的上下文中可能有不同的功能和含义. 例如: 一个按钮可能用于分享内容, 也可能用于上传图片或文件. 仅通过观察 GUI 小部件的外观和位置, 很难判断它们的具体作用. 如果不能正确地识别其行为意图, 就无法进行有效的测试.
- 第 2 个挑战是 GUI 小部件测试偏好不清楚, 即: 在不同测试目标下, 难以确定应该如何优先选择和测试哪些小部件. 例如: 在测试隐私相关的功能时, 相对于其他小部件, 可能更需要测试与位置请求、通讯录获取等按钮相关的功能. 这种偏好关系是测试过程中需要考虑的因素.
- 第 3 个挑战是缺乏训练数据. 在移动应用的复杂场景下, 提取有效的特征以及建模复杂的关系通常需

要大量的样本. 然而在实际情况中, 很难在特定任务场景下获得足够多且高质量的训练数据, 导致 GUI 模糊测试难以进行自适应和优化.

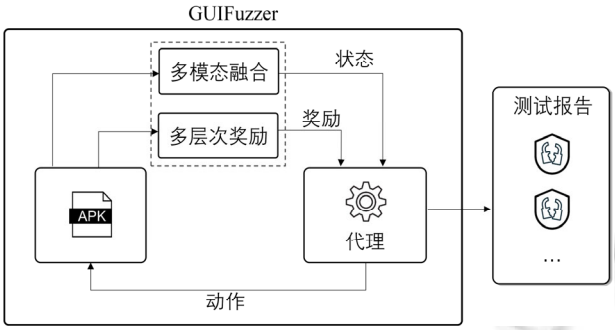


图 2 GUIFuzzer 架构

接下来, 我们将介绍如何应对上述挑战.

4.1 GUIFuzzer框架

GUI 模糊测试主要面临的挑战包括: (1)理解 GUI 小部件语义; (2) 捕获 GUI 小部件在不同测试目标下的偏好; (3) 缺乏训练数据. 为了解决后两个挑战, 我们分别采用以下策略: (1) 利用深度学习模型从大量样本中自动学习与测试目标相关的隐式偏好信息; (2) 利用强化学习代理在真实环境中模拟用户交互自主生成高质量的训练数据并将其有效地融合到学习算法中, 同时, 借助预训练模型来获取有效鲁棒的特征表示. 因此, 关键问题在于第 1 个挑战, 即如何理解小部件的语义. 然而, 这一任务极具挑战性, 因为存在外观相似的小部件(例如所有输入框看起来大致相同)以及小部件的不确定性(例如按钮状态从禁用变为启用). 事实上, 对于某些小部件, 即使人类也很难仅凭视觉理解它们的含义. 为了解决第 1 个挑战, 我们融合多模态特征来联合推断 GUI 小部件的语义.

GUIFuzzer 是一个基于深度 Q 网络的强化学习框架. 对于强化学习模型, 其核心模块主要包括状态空间以及奖励函数. 接下来将重点介绍这两个模块的设计.

4.1.1 多模态融合

GUIFuzzer 中的 Q 网络接受 3 种类型的输入特征, 即元属性特征、视觉语义特征以及布局上下文特征. 通过使用联结注意力层[29]对这些特征进行多模态融合, 得到强化学习代理的状态表示.

- 元属性特征. GUI 小部件包括许多元属性, 比如文本、占位符、可点击(clickable)、可滚动(scrollable)、可编辑(editable)等. 这些属性可容易地从转储布局文件中提取. 属性文本和占位符用于理解 GUI 小部件的语义, 帮助 GUIFuzzer 决定执行哪个小部件. 占位符属性指的是描述字段期望值的简短提示符(例如期望格式或示例值). 而可点击、可滚动和可编辑的属性可帮助 GUIFuzzer 理解如何与小部件交互. 我们利用预训练好的模型来提取 GUI 小部件的元属性特征. 首先, 我们根据主要数据类型将元属性分为两类: 字符串和布尔型. 如此设计是因为各元属性之间的功能存在显著差异. 字符串类型的属性通过使用预训练的自然语言处理模型[30]被编码为一个 768 维的嵌入向量表示. 对于布尔类型属性, 它们被表示独热(one-hot)嵌入向量. 然后, 拼接这两个嵌入向量得到最终的元属性特征.
- 视觉语义特征. GUI 状态是人类测试者从移动设备上最直接感知到的观察. 影响测试者选择执行哪个小部件的主要因素是当前的 GUI 状态. 例如: 在测试的早期阶段, 遇到的所有 GUI 状态特别是起始页都是未曾见过的. 没有任何测试经验的测试人员只能依靠当前状态进行决策. 为了提取视觉特征, GUIFuzzer 参考先前的研究[31], 采用一个无监督的视觉表示网络 AugNet, 将每个 GUI 状态嵌入到一个 768 维的特征向量. 需要注意的是: AugNet 模型需先在一个大规模公开 GUI 数据集 RICO[32]上预训练, 其中包含超过 66 000 个样本. 虽然该模型是在不同应用上训练的, 但它提取的视觉特征是鲁棒

且可区分的[33].

- 布局上下文特征. 如前所述, 布局上下文有助于推断 GUI 小部件的语义. 根据先前的研究[34], 我们使用自监督编码器 LayoutAutoEncoder[35]从活动层次结构中提取出一个 64 维的布局上下文特征向量. 该自编码器使用重构误差作为损失函数在 RICO 数据集上进行预训练. 重构误差的计算方法为输入和重构间的均方误差. 该过程分为两个步骤: 第 1 步是根据小部件的元属性将 GUI 截图的每个像素分别分配到不同的类别中, 其中类别包括文本、图像和背景, 然后分别用不同的颜色表示; 第 2 步将布局编码为位图, 然后使用自编码器提取它的结构化语义.

4.1.2 多层次奖励

多层次奖励机制的动机是为了解决 GUI 模糊测试中的两个挑战: 一是如何有效地探索 GUI 状态空间, 二是如何有效地发现测试目标. 我们认为: GUI 状态空间的复杂性和测试目标的多样性, 单一的奖励机制难以同时兼顾这两个目标. 因此, 我们设计了一个多层次奖励机制, 将 GUI 模糊测试的目标分解为 3 个层次: 活动级、小部件级和测试目标级. 每个层次都有一个对应的奖励函数, 指导代理在不同的粒度上进行 GUI 状态空间的探索和测试目标的发现. 测试目标级奖励旨在探索更多包含相关测试目标的活动. 活动级奖励和小部件级奖励旨在引导代理探索更多新活动. 具体而言, 活动级奖励旨在引导代理探索与当前活动差异较大的活动. 小部件级奖励旨在平衡两个偏好: 优先访问包含更多未访问小部件的活动和优先操作访问频率较低的小部件. 这 3 个层次的奖励是相互关联的. 例如: 如果代理选择了一个错误的小部件, 则会导致活动转换失败, 从而影响测试用例覆盖率. 基于该多层次奖励, 代理可以更有效地探索 GUI 状态空间, 并更有针对性地生成测试用例. 正式地, 我们将多层次奖励定义为

$$R_i = \sum_{k\in\{a,s,w\}} \gamma^k R_i^k \tag{4}$$

对于测试目标级奖励 R_a, 我们将其定义为活动中发现相关测试目标的数量. 比如: 如果测试目标是敏感 API 触发, 那么测试目标级奖励就是在操作小部件后跳转到的活动里触发的敏感 API 个数. 我们将活动级奖励定义为公式(5), 它使代理优先考虑有显著差异的活动.

$$R_s = \begin{cases} 1, & \text{if } dist(EB(X), EB(Y)) \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

其中, X 是当前活动的截图, Y 是后续活动的截图, τ为距离阈值, EB 为将截图转换为数值向量的嵌入模型, $dist(\cdot)$为欧几里得度量.

对于小部件级奖励, 我们将其定义为公式(6), 其中包含小部件的访问频率 VF_{ow}、后续活动中未访问的小部件数量 UN_{sa} 以及确保二者间具有可比性的缩放因子 M 和 N.

$$R_{\text{w}} = M \times \left(\frac{UN_{\text{sa}}}{N} + \frac{1}{VF_{\text{ow}}} \right) \tag{6}$$

小部件级奖励背后的基本想法是, 活动间的转换通常由操作小部件(如单击按钮)引起. 具有更多未访问小部件的活动更有可能转换到新活动. 因此, 我们结合小部件级奖励与活动级奖励以更有效地探索新的活动.

5 实验分析

为了评估 GUIFuzzer 的性能, 我们研究了以下 3 个问题.

- RQ1: GUIFuzzer 与其他竞争性基线相比测试覆盖率与故障检测能力如何?
- RQ2: 多模态特征和多层次奖励机制如何影响 GUIFuzzer 的测试性能?
- RQ3: GUIFuzzer 在敏感 API 触发的案例研究上效果如何?

5.1 实验设置

我们采用一种系统的方法即 ID 自增长策略来构建实验数据集. 这种方法可以保证数据集的代表性和公平性, 避免了人为选择或偏好的影响. 具体来说, 我们编写了一个爬虫程序从小米应用程序商店中下载应用

程序. 我们让爬虫程序按照 ID 访问网址(https://app.mi.com/download/{id}), 其中, 占位符 id 的取值范围为 1–500. 在剔除无效链接后, 数据集总共包含 121 个应用程序, 涵盖了多种类别和功能, 如社交、工具和新闻等. 这些应用程序的大小从 128 KB 到 424 MB 不等. 由于篇幅限制, 我们无法展示所有应用程序的详细信息, 但是我们提供了一个在线链接(https://disk.pku.edu.cn:443/link/73C914292CE7AE386E81CC630FCD66AE)供感兴趣的读者参考. GUIFuzzer 通过 Appium 动态地探索应用程序. 模块是基于深度 Q 网络架构, 在 Pytorch 框架下实现的. 根据以往的研究[36], 我们对每个移动应用设置最多 60 min 的探索时间, 并且探索过程重复 3 次以保证结果的可靠性, 因为模型在每次训练可能采取的探索策略不同[13]. 实验参数设置如下: 活动级奖励与小部件级奖励的加权因子分别设置为 2 和 1; 对于 RQ1 与 RQ2, 测试目标级奖励的加权因子设置为 0; 对于 RQ3, 由于是面向特定目标的, 我们将其加权因子设置为一个较大的数, 即 50; 我们设置小部件级奖励中的缩放因子 M=2 与 N=10; 距离阈值 τ 设置为 2. 此外, 对于这些商业应用, 由于我们无法获取它们的源代码, 所以我们参考了先前的研究[1], 使用活动覆盖率作为评估框架测试效率的指标.

5.2 总体结果

为了回答 RQ1, 我们首先选择了两个竞争性基线与所提框架比较覆盖率. 该研究问题的动机是, 证明我们框架能够更有效地探索移动应用的 GUI 状态空间. 我们选择基线基于以下两个标准: (1) 最新提出的方法, 即在最近的相关论文中提出的方法; (2) 常用有效的方法, 即在实际应用中广泛使用的方法, 它们通常具有较好的稳定性和可扩展性. 根据表 1, 我们选取了 UniRLTest[14]与 ARES[12]作为最新的方法. 需要注意的是: 我们没有选取 RELINE[13], 因为它是专门针对视频游戏的测试工具, 不适用于 Android 应用. 此外,根据先前的研究[8−14], 我们选择 Monkey 作为最常用的方法进行对比. 我们选择 Monkey 还因为许多研究者[7,37]证实了 Monkey 优于大多数测试工具, 因为它能在相同的时间内生成更多的输入.

图 3 展示了不同测试工具的活动覆盖率. 从图中可以看出, GUIFuzzer 显著优于基线 UniRLTest、ARES 与 Monkey. 在所有的测试对象中, GUIFuzzer 的平均活动覆盖率为 31.7%, 而 UniRLTest、ARES 和 Monkey 的平均活动覆盖率分为 26.5%、24.8%和 18.6%. 这意味着: 相比于基线 UniRLTest, GUIFuzzer 在覆盖率上平均提高了 19.6%; 相比于基线 ARES, GUIFuzzer 在覆盖率上平均提高了 27.8%; 相比于基线 Monkey, GUIFuzzer 在覆盖率上平均提高了 70.4%. 这些结果表明: 所提框架能够有效地生成高质量的测试用例, 从而提高测试的活动覆盖率.

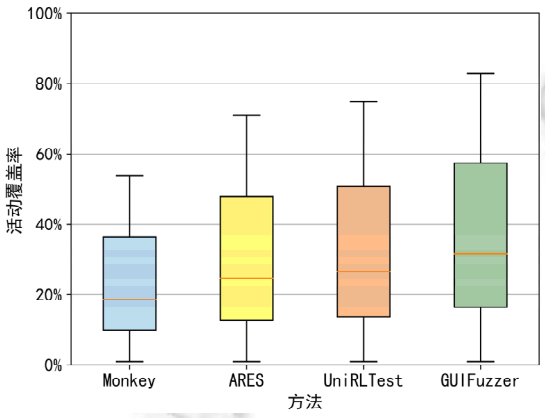


图 3 不同方法的平均活动覆盖率比较

图 4 展示了测试工具在不同输入事件数下的平均活动覆盖率变化情况. 从图中可以看出, 所有测试工具在初始阶段都有较快的活动覆盖率增长. 这主要是因为应用刚启动时, 所有 GUI 状态都未曾访问. 随着事件数的增加, GUIFuzzer 逐渐超过了其他测试工具, 这表明我们框架具有提高测试性能的巨大潜力. 此外, 我们观察到: 在第 600 个事件点附近, GUIFuzzer、UniRLTest 和 ARES 的活动覆盖率已基本趋于稳定, 而 Monkey

的覆盖率还有微小的上升. 这是因为 Monkey 只是随机地生成事件序列, 导致大量的冗余与无效事件, 在相同的事件数量下, 其活动覆盖率不如其他工具. 然而, Monkey 在相同时间下能够生成更多的事件, 因此在测试时间结束前, 其活动覆盖率还有轻微的提升, 最终达到了 18.6%.

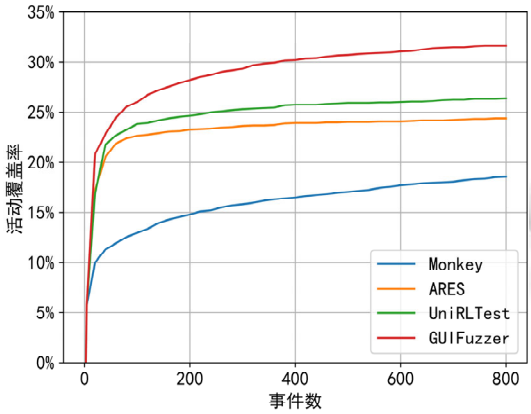


图 4 方法在不同输入事件数下的平均覆盖率

为了评估 GUIFuzzer 的故障检测能力, 我们在实验对象中额外增加了 10 个非商业的开源应用, 分别是 Amaze File Manager (版本 3.8.1)、Diary (版本 1.77)、OpenTasks (版本 1.2.3)、Simple Draw (版本 6.8.0)、Simple File Manager (版本 6.14.0)、WiFi Analyzer (版本 3.0.1)、Hibi (版本 1.3.1)、Suntimes (版本 0.14.12)、Nani (版本 0.3.0)、Currency (版本 1.31), 并报告 GUIFuzzer 和基线方法在这些应用上的故障检测效果. 结果显示: 在这 10 个应用中, 所有方法共发现了 8 个故障. 图 5 展示了一个由剪切粘贴操作引起的故障示例. 在这个操作中, 用户试图将一个文件夹剪切并粘贴到它自己的子文件夹中. 这样做会导致数组的索引超出范围, 从而触发了一个数组越界异常. 我们发现: GUIFuzzer 检测出更多的故障数量, 因为多级奖励能够使其探索更多的 GUI 状态增加发现故障的机会. 具体而言, GUIFuzzer 检测到的故障数量为 6 个, 而 Monkey、ARES、UniRLTest 检测到的故障数量分别为 1 个、3 个和 4 个. 在所有应用对象中, GUIFuzzer 分别发现了 6 个、3 个和 3 个未被 Monkey、ARES、UniRLTest 检测到的故障, 而 Monkey、ARES 和 UniRLTest 分别发现了 1 个、0 个和 1 个未被 GUIFuzzer 发现的故障. 上述结果表明了 GUIFuzzer 在故障发现上的有效性.

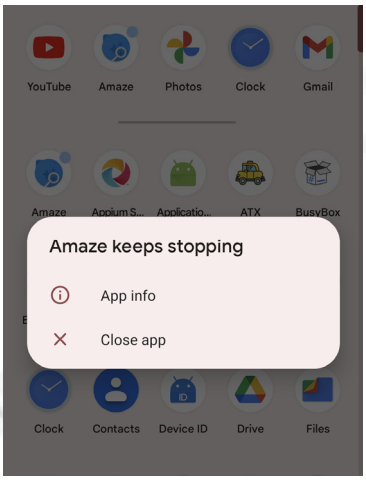


图 5 一个由剪切粘贴操作引起的故障示例

我们的回答: GUIFuzzer 在活动覆盖率和故障检测能力上显著优于竞争性基线方法. 相比于 UniRLTest、ARES 和 Monkey 基线方法, GUIFuzzer 的平均活动覆盖率分别提高了 19.6%、27.8%和 70.4%. 在故障检测方面, GUIFuzzer 发现的故障检测数量分别是 Monkey、ARES、UniRLTest 的 6 倍、2 倍、1.5 倍.

5.3 消融研究

为了回答 RQ2, 我们对不同模态特征和多层次奖励机制进行了消融研究. 该研究问题的动机是, 探究它们对框架性能的影响. 我们的评估方法是: 使用不同模态特征和多目标奖励机制配置在相同的测试对象上进行实验, 比较它们在活动覆盖率上的表现. 图 6 和图 7 展示了详细的数值结果. NoVF、NoLC 和 NoMA 分别表示我们消除了视觉特征、布局上下文特征和元属性特征, NoACT 和 NoWGT 分别表示我们消除了活动级奖励和小部件级奖励. 需要注意的是: 在 RQ1 中, GUIFuzzer 只关注覆盖率的提高, 而不是针对任何特定的测试目标, 因此我们没有使用测试目标级奖励. 测试目标级奖励的作用将在下一节中通过案例研究进行展示. 从图 6 中我们可明显地观察到, GUIFuzzer 比 NoMA 具有更好的性能. GUIFuzzer 的活动覆盖率高于 NoMA, 这表明细粒度元属性特征在提高覆盖率方面的有效性. 主要原因是元属性特征可直接反映 GUI 小部件的语义, 并且在不同上下文中具有泛化性.

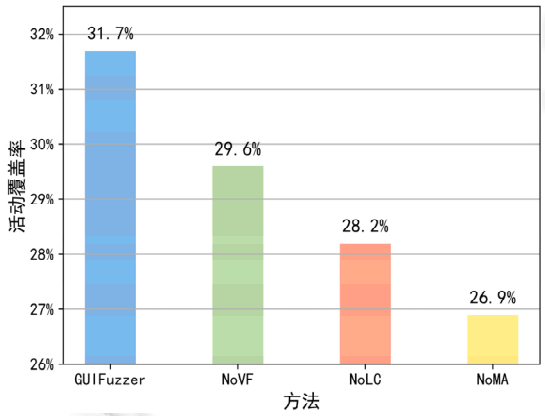


图 6 多模态特征消融分析

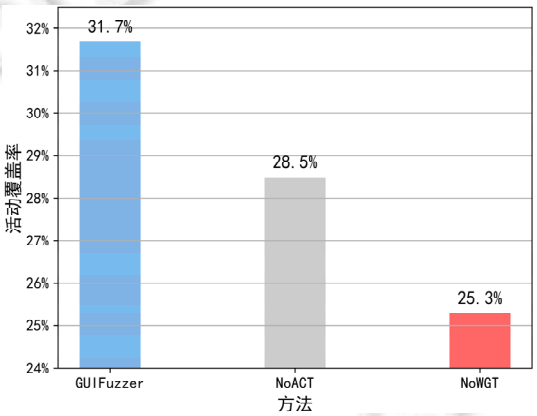


图 7 多层次奖励消融分析

GUIFuzzer 的性能也明显优于变体 NoLC. 与 NoLC 相比, GUIFuzzer 的覆盖率提升了 12.4%. 该结果表明布局上下文特征在提高覆盖率方面的有效性. 主要原因是: 当文本和内容描述等小部件元属性缺失或不准确时, 可根据运行时上下文推断 GUI 小部件的语义, 从而生成更合理的测试用例. 此外, 与其他两种特征相比, 视觉特征对框架性能的影响相对较小. 一种可能的解释是: 它仅基于视觉信息来整体理解一个复杂的 GUI 状态, 而这种粗粒度的理解难以捕捉到局部小部件的语义细节.

总的来说, 3 个变体的性能都有不同程度的下降. GUIFuzzer 的活动覆盖率为 31.7%, 而 NoVF、NoLC、NoMA 的活动覆盖率分别为 29.6%、28.2%、26.9%. 这表明, 集成这些模态特征有利于提高 GUIFuzzer 的整体性能. 不同变体间的结果差异表明了这些模态特征对模型性能的有效性影响从低到高依次排序为: 视觉特征、布局上下文特征和细粒度元属性特征.

如图 7 所示, 我们发现: 通过引入活动级奖励和小部件级奖励, GUIFuzzer 能够有效地提高其在测试应用程序中的活动覆盖率. NoACT 和 NoWGT 的平均活动覆盖率分别为 28.5%和 25.3%. 这意味着: 与 NoACT 相比, GUIFuzzer 的活动覆盖率提升了 11.2%; 与 NoWGT 相比, GUIFuzzer 的活动覆盖率提升了 25.3%. 该结果表明: 在多层次奖励函数中, 相比于活动级奖励, 小部件级奖励对 GUI 模糊测试的性能影响更大, 因为它能够引导代理更细粒度地探索不同的活动, 并发现更多的隐藏状态.

我们的回答: 相比于单一模态特征或单目标奖励机制, 多模态特征和多层次奖励机制能够有效提高

GUIFuzzer 框架的性能. 研究结果表明, 细粒度元属性特征和小部件级奖励对 GUIFuzzer 的性能影响最大.

5.4 案例研究

在本节中, 我们的动机是评估 GUIFuzzer 在定制化搜索即敏感 API 触发上的有用性, 以帮助开发者或测试人员发现移动应用中可能存在的安全或隐私问题. 我们的评估方法是使用一个敏感 API 列表作为输入, 让框架尝试触发这些敏感 API, 并记录在给定时间下发现敏感 API 的数量.

我们根据最近的研究收集并构建敏感 API 库[38,39]. 事实上, Android 恶意软件通常通过调用安全相关的 API 来传播恶意行为. 例如, *getLine*1*Number*(·)可获取用户的电话号码, *getDeviceID*(·)可获取设备的 IMEI. 为了有效地揭露恶意行为, 我们主要监测这些安全相关的 API 调用即敏感 API 调用. 基于 Gong 等人[38]和 Wu 等人[39]报告的结果, 再结合恶意软件分析经验, 最后共监测 393 个敏感 API, 见表 2. 它们按资源类型分主要包括电话、位置、蓝牙、麦克风与相机、连接状态、应用状态、账户和数据.

我们采用触发敏感 API 的速度即相同时间下发现敏感 API 的数量来衡量框架的性能. 我们将所提方法 GUIFuzzer 与 4 个基线方法比较, 其中两个是最新的通用测试工具 ARES[12]和 UniRLTest[14], 它们以覆盖率最大化为目标; 另外两个是 ARES 与 UniRLTest 的变体, 即 ARES-S 与 UniRLTest-S. 相比于 ARES/UniRLTest, ARES-S/UniRLTest-S 在目标函数中额外考虑了每个活动中敏感 API 调用的数量. 如此设计是为了表明对现有方法轻微修改也难以完成任务, 从而验证我们框架的有效性. 此外, 我们对敏感 API 调用的数量进行描述性统计包括均值、标准差和中位数, 并使用曼-惠特尼(Mann-Whitney)检验[40]来比较不同模型在识别敏感 API 调用时的性能特征. 我们还使用了一种非参数的序列数据分布度量方法(Cliff's delta)[41]来量化比较效应大小间的差异. 我们选择它是因为它无须预先对数据分布类型进行假设[42]. 我们遵循公认的分析指南[13]进行解释. 当$|d|<0.10$ 时, 认为是可忽略的(N); 当 $0.10\leqslant|d|<0.33$ 时, 认为是小的(S); 当 $0.33<|d|\leqslant0.474$ 时, 认为是中等(M); 当$|d|>0.474$ 时, 认为是大的(L).

表 2 监测敏感 API 列表

资源类型	API 类	监测 API 示例	描述
电话	TelephonyManager、SmsManager	*getLine*1*Number*(·)、*getDeviceID*(·)、*sendTextMessage*(·)	获取电话相关的敏感信息
位置	Location、Address、GoogleMap、LocationManagerService	*getLastKnownLocation*(·)、*requestLocationUpdates*(·)、*getLongitude*(·)	获取用户位置信息, 如 GPS 坐标
蓝牙	BluetoothDevice、BluetoothAdapter	*getName*(·), *getAddress*(·)、*getBondedDevices*(·)	潜在跟踪用户移动与识别附近设备
麦克风&相机	AudioRecord、MediaRecorder、Camera	*startRecording*(·)、*start*(·)、*takePicture*(·)	访问麦克风和摄像头来录制音频和视频
连接状态	WifiManager、WifiServiceImpl	*getScanResults*(·)、*getDhcpInfo*(·)、*getWifiState*(·)	获取用户的网络连接和使用模式信息
应用状态	PackageManager、UsageStatsManager	*getInstalledPackages*(·)、*getApplicationInfo*(·)、*queryUsageStats*(·)	检查防病毒和金融应用可访问性以检测网络钓鱼等攻击
账户	AccountManager	*getAccountsByType*(·)	访问和管理设备上的用户账户凭据
数据	ContentProvider	*query*(·)	管理应用数据存储

图 8 展示了 GUIFuzzer 与其他基线的定量结果. 我们发现, GUIFuzzer 的性能优于 UniRLTest、UniRLTest-S、ARES 和 ARES-S. 平均而言, GUIFuzzer 检测到 2 137 个敏感 API 调用, 而 UniRLTest-S、UniRLTest、ARES-S、ARES 分别为 1 495 个、1 442 个、1 362 个、1 224 个. 结果表明: 与竞争性基线相比, GUIFuzzer 识别出更多的敏感 API 调用. 这是符合预期的, 因为它有触发更多敏感 API 的明确目标. UniRLTest 与 ARES 检测较少敏感 API 调用的主要原因是: 它们采用了简单的奖励函数, 难以有效探索具有复杂业务逻辑的应用. 该奖励函数过分强调状态差异, 很容易导致代理在差异较大的状态间来回跳转. 我们还发现, UniRLTest-S/ARES-S 的性能优于 UniRLTest/ARES, 该结果表明了测试目标级奖励的作用. 然而, 尽管 UniRLTest-S 在奖励函数中考虑了敏感 API 调用数量, 但由于理解粗粒度, 仍无法有效学习到局部小部件和敏感 API 调用间的关系. 上述结果表明, 这些基线方法并不适用于发现敏感 API 调用.

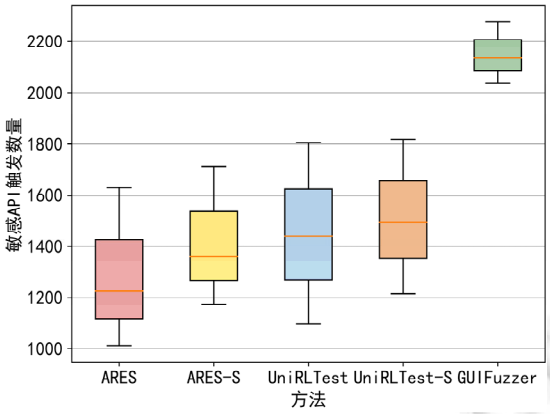


图 8　不同方法的敏感 API 触发数量比较

我们还比较了 GUIFuzzer 与基线方法在识别敏感 API 调用的性能特征. 为了清晰可见, 我们仅选择了性能较好的方法 UniRLTest 及其变体进行比较. 首先, 我们计算方法在每个应用程序上检测到的敏感 API 数量, 这将产生一个维度为应用数量大小的数值向量; 然后, 使用曼-惠特尼检验与序列数据分布度量方法来量化比较这些分布. 表 3 展示了 3 种方法成对统计比较的结果. 粗体表示每个测试中识别更多敏感 API 调用的方法. 与 UniRLTest 和 UniRLTest-S 相比, GUIFuzzer 调整后的 P 值小于 0.001 且效应量小, 表现出显著的统计学差异. 因此, 在大多数应用中, GUIFuzzer 识别出比 UniRLTest-S 和 UniRLTest 方法更多的敏感 API 调用, 表明了我们框架在检测敏感 API 调用方面的有效性.

表 3　显著性检验统计分析结果

测试	Cliff's delta	P 值
UniRLTest-S vs. UniRLTest	0.023 (可忽略)	<0.001
GUIFuzzer vs. UniRLTest-S	0.278 (小)	<0.001
GUIFuzzer vs. UniRLTest-S	0.301 (小)	<0.001

我们的回答: 与竞争性基线相比, GUIFuzzer 框架在相同的时间内能够识别更多的敏感 API 调用. 该结果表明了 GUIFuzzer 在定制化搜索方面的有用性.

6　讨　论

6.1　主要优势

GUIFuzzer 是一种基于多模态表征的 GUI 模糊测试框架, 它能够从图像、属性和布局等多个维度对 GUI 界面进行描述, 并通过一个深度强化学习模型利用这些信息来生成合适的测试用例. 框架的主要特点和优势如下.

(1) 它能够充分利用 GUI 界面中的视觉、语义和逻辑信息, 提高测试用例的质量和多样性. 例如: 框架能够根据按钮的图像、位置、文本等多个维度的特征推断它是登录按钮、返回按钮还是分享按钮, 然后根据不同的测试目标选择执行不同的操作. 这能有效地避免生成一些无效或者重复的测试用例, 提高测试效率和覆盖率.

(2) 它采用了深度强化学习模型来生成和变异测试用例, 实现了 GUI 测试过程的自动化和智能化, 降低了人工参与和成本. 同时, 深度学习模型还能够通过持续地学习 GUI 界面和测试用例间的映射关系, 提高测试用例的适应性和有效性.

6.2　局限性

GUIFuzzer 也存在一些局限性和不足之处.

- 它仅采用深度 Q 网络作为强化学习代理. 事实上, 除了深度 Q 网络, 还有一些其他强化学习架构, 如 A3C[43]、SAC[44]等, 它们在连续动作空间、多智能体协作等方面有着不同的优势. 我们计划在未来的工作中探索其他深度强化学习架构, 以提高 GUIFuzzer 的效率和覆盖率.
- 此外, 对于一些需要用户交互或者输入数据的 GUI 界面, 我们的框架可能无法模拟或者生成合理的用户行为或者数据等, 这会限制我们框架的适用性.

6.3 有效性威胁

内部有效性的主要威胁是超参数设置, 它会影响我们框架的性能. 由于时间和资源的限制, 我们无法对超参数进行全面的微调. 因此, 我们实验的当前参数可能不是最优的. 为了减轻这一威胁, 我们将所有参数设置为默认值或参考先前研究的推荐值. 对于没有已知参考的情况, 我们根据最佳实践进行小规模实验, 选择合适的参数.

外部有效性威胁是实验中采用的应用数量有限. 为了降低这一威胁, 我们采用了一种系统的方法, 即基于 ID 自增长策略, 从小米应用商店中选取应用构建数据集. 它是一种公平的方法, 避免了人为选择或偏好的影响. 实验数据集涵盖了不同类别和功能的应用程序, 它能够反映移动应用的多样性, 覆盖不同用户群体和使用场景. 因此, 我们采用 ID 自增长策略能够保证数据集的公平性和有效性.

此外, 虽然比较更多的方法可能会更有说服力, 但我们所选的方法都是最前沿的测试方法, 具有代表性和挑战性, 能够充分证明我们框架的创新性和有效性. 与这些具有代表性和竞争力的方法进行比较, 已经足够展示我们工作的优势和贡献.

6.4 性能分析

由于 GUIFuzzer 采用了多模态表征, 所以在测试时间和内存上并不比基于随机或单模态表征的方法有优势. 然而, 我们框架的推理时间和内存消耗是可接受的, 分别为 1.65 s 和 2.03 GB. 另外, 据我们所知, 现有工作都没有对动作生成效率进行比较. GUIFuzzer 的主要优势在于它能够利用多模态信息生成更多样化和有效的测试动作, 从而提高测试覆盖率. 我们相信, GUIFuzzer 在未来将作为随机方法的补充手段, 能够覆盖更多现有方法难以达到的页面.

6.5 未来研究工作

我们的实验数据集目前只涵盖了小米应用商店中的应用程序, 尚未包含其他平台或渠道的应用程序. 我们计划在未来的工作中进一步扩展和完善我们的数据集, 并采用类别、下载排行等应用选择标准来增加其可信度和可复现性. 此外, 我们并没有采用测试有效性等定性指标评估框架的性能. 我们认为, 这些指标难以量化和客观衡量. 测试有效性则需要人工参与判断测试结果是否符合预期, 这会增加评价方法的主观性和复杂度, 例如如何判断测试是否有效等. 我们需要未来进一步研究这些问题, 并设计合理的实验方案和分析方法.

7 总 结

本文通过对移动应用 GUI 模糊测试深入分析, 总结了现有工作的局限性和存在的问题, 即对 GUI 内小部件理解不足, 并基于总结的问题提出了基于多模态表征的移动应用 GUI 模糊测试框架 GUIFuzzer. 该框架通过考虑多模态特征, 如视觉特征、布局上下特征以及细粒度的元属性特征, 来更好地建模 GUI 小部件的语义, 然后利用一个多层次奖励驱动的深度强化学习模型从大量样本中自动地学习与测试目标相关的偏好. 基于该偏好, 代理能够生成更有针对性的测试用例, 从而提高测试覆盖率和效率. 我们在大量的真实应用上对所提框架进行评估. 实验结果表明: 与现有的竞争性基线相比, GUIFuzzer 显著地提升了模糊测试的覆盖率. 多模态特征对模型性能的有效性影响从低到高依次排序为: 视觉特征、布局上下文特征和细粒度元属性特征. 我们还对特定目标的定制化搜索, 即敏感 API 触发进行了案例研究. 结果表明: 我们的框架比竞争性基线发现更多地敏感 API 调用, 这也验证了 GUIFuzzer 框架的实用性.

References:

[1] Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for Android apps. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2013. 224−234.

[2] Li Y, Yang Z, Guo Y, Chen X. DroidBot: A lightweight UI-guided test input generator for Android. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering Companion (ICSE-C). IEEE, 2017. 23−26.

[3] Su T, Meng G, Chen Y, Wu K, Yang W, Yao Y, Pu G, Liu Y, Su Z. Guided, stochastic model-based GUI testing of Android apps. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017. 245−256.

[4] Hao S, Liu B, Nath S, Halfond WG, Govindan R. PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps. In: Proc. of the 12th Annual Int'l Conf. on Mobile Systems, Applications, and Services. New York: ACM, 2014. 204−217.

[5] Mao K, Harman M, Jia Y. Sapienz: Multi-objective automated testing for Android applications. In: Proc. of the 25th Int'l Symp. on Software Testing and Analysis. New York: ACM, 2016. 94−105.

[6] Azim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of Android apps. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object-oriented Programming Systems Languages & Applications. New York: ACM, 2013. 641−660.

[7] Li Y, Yang Z, Guo Y, Chen X. Humanoid: A deep learning-based approach to automated black-box Android app testing. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2019. 1070−1073.

[8] Pan M, Huang A, Wang G, Zhang T, Li X. Reinforcement learning based curiosity-driven testing of Android applications. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM, 2020. 153−164.

[9] Feng X. Automated mobile apps testing from visual perspective. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM, 2020. 577−581.

[10] Faraz Y, Malek S. Deep GUI: Black-box GUI input generation with deep learning. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2021. 905−916.

[11] Zheng Y, Liu Y, Xie X, Liu Y, Ma L, Hao J, Liu Y. Automatic Web testing using curiosity-driven reinforcement learning. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). IEEE, 2021. 423−435.

[12] Romdhana A, Merlo A, Ceccato M, Tonella P. Deep reinforcement learning for black-box testing of Android apps. ACM Trans. on Software Engineering and Methodology (TOSEM), 2022, 31(4): 1−29.

[13] Tufano R, Scalabrino S, Pascarella L, Aghajani E, Oliveto R, Bavota G. Using reinforcement learning for load testing of video games. In: Proc. of the 44th Int'l Conf. on Software Engineering. New York: ACM, 2022. 2303−2314.

[14] Zhang Z, Liu Y, Yu S, Li X, Yun Y, Fang C, Chen Z. UniRLTest: Universal platform-independent testing with reinforcement learning via image understanding. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM, 2022. 805−808.

[15] Koroglu Y, Sen A, Muslu O, Mete Y, Ulker C, Tanriverdi T, Donmez Y. QBE: Q-learning-based exploration of Android applications. In: Proc. of the 11th IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). IEEE, 2018. 105−115.

[16] Spieker H, Gotlieb A, Marijan D, Mossige M. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In: Proc. of the 26th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM, 2017. 12−22.

[17] Zheng Y, Xie X, Su T, Ma L, Hao J, Meng Z, Liu Y, Shen R, Cheen Y, Fan C. Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2019. 772−784.

[18] Xu T, Pan M, Pei Y, Li G, Zeng X, Zhang T, Deng Y, Li X. GUIDER: GUI structure and vision co-guided test script repair for Android apps. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM, 2021. 191−203.

[19] Sasnauskas R, Regehr J. Intent fuzzer: Crafting intents of death. In: Proc. of the 2014 Joint Int'l Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA). New York: ACM, 2014. 1−5.

[20] Yang W, Prasad MR, Xie T. A grey-box approach for automated GUI-model generation of mobile applications. In: Proc. of the Int'l Conf. on Fundamental Approaches to Software Engineering. Berlin: Springer, 2013. 250−265.

[21] Arnatovich YL, Ngo MN, Kuan THB, Soh C. Achieving high code coverage in Android UI testing via automated widget exercising. In: Proc. of the 23rd Asia-Pacific Software Engineering Conf. (APSEC). IEEE, 2016. 193−200.

[22] Choi W, Necula G, Sen K. Guided GUI testing of Android apps with minimal restart and approximate learning. ACM SIGPLAN Notices, 2013, 48(10): 623−640.

[23] Adamo D, Khan MK, Koppula S, Bryce R. Reinforcement learning for Android GUI testing. In: Proc. of the 9th ACM SIGSOFT Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. New York: ACM, 2018. 2−8.

[24] Vuong TAT, Takada S. A reinforcement learning based approach to automated testing of Android applications. In: Proc. of the 9th ACM SIGSOFT Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. New York: ACM, 2018. 31−37.

[25] Vuong TAT, Takada S. Semantic analysis for deep *Q*-network in Android GUI testing. In: Proc. of the 31st Int'l Conf. on Software Engineering and Knowledge Engineering, 2019. 123−170.

[26] Mariani L, Pezze M, Riganelli O, Santoro M. AutoBlackTest: Automatic black-box testing of interactive applications. In: Proc. of the IEEE 5th Int'l Conf. on Software Testing, Verification and Validation. IEEE, 2012. 81−90.

[27] Chen J, Chen C, Xing Z, Xu X, Zhut L, Li G, Wang J. Unblind your apps: Predicting natural-language labels for mobile GUI components by deep learning. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). New York: ACM, 2020. 322−334.

[28] Zhang S, Li Y, Yan W, Guo Y, Chen X. Dependency-aware form understanding. In: Proc. of the 32nd IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). IEEE, 2021. 139−149.

[29] Xu G, Cheng J, Guo P, Yang X. Attention concatenation volume for accurate and efficient stereo matching. In: Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition. IEEE, 2022. 12981−12990.

[30] Reimers N, Gurevych I. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In: Proc. of the 2019 Conf. on Empirical Methods in Natural Language Processing and the 9th Int'l Joint Conf. on Natural Language Processing (EMNLP-IJCNLP). ACL, 2019. 3982−3992.

[31] Zhang S, Wu L, Li Y, Zhang Z, Lei H, Li D, Guo Y, Chen X. ReSPlay: Improving cross-platform record-and-replay with GUI sequence matching. In: Proc. of the 34th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). IEEE, 2023. 439−450.

[32] Deka B, Huang Z, Franzen C, Hibschman J, Afergan D, Li Y, Nichols J, Kumar R. Rico: A mobile app dataset for building data-driven design applications. In: Proc. of the 30th Annual ACM Symp. on User Interface Software and Technology. New York: ACM, 2017. 845−854.

[33] Wang B, Li G, Zhou X, Chen Z, Grossman T, Li Y. Screen2Words: Automatic mobile UI summarization with multimodal learning. In: Proc. of the 34th Annual ACM Symp. on User Interface Software and Technology. New York: ACM, 2021. 498−510.

[34] Feiz S, Wu J, Zhang X, Swearngin A, Barik T, Nichols J. Understanding screen relationships from screenshots of smartphone applications. In: Proc. of the 27th Int'l Conf. on Intelligent User Interfaces. New York: ACM, 2022. 447−458.

[35] Li TJJ, Popowski L, Mitchell T, Myers BA. Screen2Vec: Semantic embedding of GUI screens and GUI components. In: Proc. of the 2021 CHI Conf. on Human Factors in Computing Systems. New York: ACM, 2021. 1−15.

[36] Pan X, Cao Y, Du X, He B, Fang G, Shao R, Chen Y. FlowCog: Context-aware semantics extraction and analysis of information flow leaks in android apps. In: Proc. of the 27th USENIX Security Symp. USENIX Association, 2018. 1669−1685.

[37] Choudhary SR, Gorla A, Orso A. Automated test input generation for Android: Are we there yet? In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2015. 429−440.

[38] Gong L, Li Z, Qian F, Zhang Z, Chen QA, Qian Z, Lin H, Liu Y. Experiences of landing machine learning onto market-scale mobile malware detection. In: Proc. of the 15th European Conf. on Computer Systems. New York: ACM, 2020. 1−14.

[39] Wu Y, Zou D, Yang W, Li X, Jin H. HomDroid: Detecting Android covert malware by social-network homophily analysis. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. New York: ACM, 2021. 216−229.

[40] Nahm FS. Nonparametric statistical tests for the continuous data: The basic concept and the practical use. Korean Journal of Anesthesiology, 2016, 69(1): 8−14.

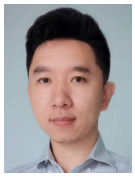
[41] Dunkler D, Haller M, Oberbauer R, Heinze G. To test or to estimate? P-values versus effect sizes. Transplant Int'l, 2020, 33(1): 50−55.

[42] Hegyi H. Connecting myelin-related and synaptic dysfunction in schizophrenia with SNP-rich gene expression hubs. Scientific Reports, 2017, 7(1): 1−14.

[43] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Proc. of the 33rd Int'l Conf. on Machine Learning. 2016. 1928−1937.

[44] Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proc. of the 35th Int'l Conf. on Machine Learning. 2018. 1861−1870.

张少坤(1994－), 男, 博士, CCF 专业会员, 主要研究领域为自动化测试, 隐私保护, 时空数据分析.



李元春(1993－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为边缘计算, 人工智能, 系统软件.

雷瀚文(1997－), 男, 博士生, 主要研究领域为基于能力的泛在操作系统构造方法与运行机制, 操作系统内存隔离机制, WebAssembly 内存安全.

蒋鹏(1994－), 男, 博士, 主要研究领域为系统安全, APT 攻击检测, 软硬协同防御.

李锭(1988－), 男, 博士, 助理教授, 博士生导师, CCF 专业会员, 主要研究领域为系统安全, 软件工程.

郭耀(1976－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为操作系统, 软件工程, 移动计算.

陈向群(1961－), 女, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为系统软件, 软件工程, 移动计算.