软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(6):2455−2476 [doi: 10.13328/j.cnki.jos.007468] [CSTR: 32375.14.jos.007468]

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

# 组件感知的安卓应用崩溃自动复现方法*

刘姝琪[1,2], 周 宇[1,2], 杨慧文[1,2]

[1](南京航空航天大学 计算机科学与技术学院, 江苏 南京 211106)

[2](高安全系统的软件开发与验证技术工信部重点实验室 (南京航空航天大学), 江苏 南京 211106)

通信作者: 周宇, E-mail: zhouyu@nuaa.edu.cn

**摘 要**: Android 应用开发人员需要快速、准确地复现错误报告以保障应用质量. 然而, 现有方法通常仅依赖堆栈跟踪中提供的崩溃信息生成事件序列, 难以准确定位崩溃页面, 无法为动态探索提供有效指导以触发崩溃. 为解决这一问题, 提出一种组件感知的安卓应用崩溃自动复现方法 CReDroid, 能够结合崩溃报告的标题信息和堆栈跟踪来有效地复现崩溃. 首先, CReDroid 通过动态探索被测应用构建组件转换图 (component transition graph, CTG), 结合堆栈跟踪的动态异常信息与 CTG 的静态组件交互信息, 精确定位目标崩溃组件; 其次, 基于崩溃报告标题中的关键操作与 CTG 中的可达路径, 设计自适应评分策略, 利用当前页面所属组件与崩溃组件的上下文关系为 GUI 控件分配选择优先级分数, 并通过强化学习全局优化动态探索过程, 有效减轻预测过程中的不准确性. 在 74 个崩溃报告上评估了 CReDroid 的性能, 并与当前先进的崩溃复现工具 CrashTranslator、ReCDroid、ReproBot 以及广泛使用的自动化测试工具 Monkey 和 APE 进行对比实验. 实验结果显示, CReDroid 成功复现了 57 个崩溃报告, 分别比 CrashTranslator、ReCDroid、ReproBot、Monkey 和 APE 多复现 13、25、27、30 和 17 个. 此外, 在成功复现相同崩溃的情况下, CReDroid 的平均用时较 CrashTranslator、ReCDroid、ReproBot、Monkey 和 APE 分别减少 26.71%、94.96%、71.65%、84.72% 和 88.56%.

**关键词**: 崩溃复现; 组件感知; 堆栈跟踪; 报告标题

**中图法分类号**: TP311

中文引用格式: 刘姝琪, 周宇, 杨慧文. 组件感知的安卓应用崩溃自动复现方法. 软件学报, 2026, 37(6): 2455–2476. http://www.jos.org.cn/1000-9825/7468.htm

英文引用格式: Liu SQ, Zhou Y, Yang HW. Component-aware Automatic Crash Reproduction Method for Android Applications. Ruan Jian Xue Bao/Journal of Software, 2026, 37(6): 2455–2476 (in Chinese). http://www.jos.org.cn/1000-9825/7468.htm

## Component-aware Automatic Crash Reproduction Method for Android Applications

LIU Shu-Qi[1,2], ZHOU Yu[1,2], YANG Hui-Wen[1,2]

[1](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

[2](Key Laboratory for Safety-critical Software Development and Verification (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, Nanjing 211106, China)

**Abstract**: Android application developers need to quickly and accurately reproduce error reports to ensure application quality. However, existing methods often rely solely on crash information provided in stack traces to generate event sequences, making it difficult to accurately locate the crash page and offer effective guidance for dynamic exploration to trigger the crash. To address this issue, this study proposes a component-aware automatic crash reproduction method for Android applications, called CReDroid, which effectively reproduces the crash by leveraging both the title and stack trace of the crash report. First, CReDroid dynamically explores the application under test to construct a component transition graph (CTG) and combines the dynamic exception information from the stack traces with the static

* 基金项目: 国家自然科学基金 (62372232); 中央高校基本科研业务费专项资金 (NG2023005)
收稿时间: 2025-01-07; 修改时间: 2025-02-17, 2025-04-16; 采用时间: 2025-05-09; jos 在线出版时间: 2025-09-10
CNKI 网络首发时间: 2025-09-11

component interaction data from the CTG to accurately locate the target crash component. Second, based on the critical operations in the crash report title and the reachable paths in the CTG, CReDroid designs an adaptive strategy that uses the contextual relationship between the current page's component and the crash component to assign priority scores to GUI widgets. The dynamic exploration process is globally optimized through reinforcement learning to effectively reduce inaccuracies in the prediction process. This study evaluates CReDroid using 74 crash reports and compares its performance with state-of-the-art crash reproduction tools, including CrashTranslator, ReCDroid, and ReproBot, as well as widely used automated testing tools, Monkey and APE. The experimental results show that CReDroid successfully reproduces 57 crash reports, which is 13, 25, 27, 30, and 17 more than CrashTranslator, ReCDroid, ReproBot, Monkey, and APE, respectively. Moreover, for the successfully reproduced crashes, CReDroid reduces the average reproduction time by 26.71%, 94.96%, 71.65%, 84.72%, and 88.56%, compared to CrashTranslator, ReCDroid, ReproBot, Monkey, and APE.

**Key words**: crash reproduction; component awareness; stack trace; report title

截至 2024 年 10 月, 安卓用户可以在 230 万个应用中进行选择[1], 其中 Google Play[2]的应用下载量已超过 256 亿次[3], 市场份额超过 70%[4], 成为全球应用数量最多的应用商店. 然而, 随着安卓应用 (Android application) 功能的不断丰富和交互环境的日益复杂, 应用崩溃问题也出现更加频繁, 这对用户体验和隐私安全带来极大影响[5]. 当用户遇到崩溃问题时, 他们通常会通过例如 GitHub[6]、Bugzilla[7]等问题跟踪系统提交一份错误报告, 描述应用故障情况, 并提供相应的复现步骤以协助开发人员解决问题. 为保障用户的高质量体验, 开发者需要根据报告中提供的信息尽快复现崩溃, 定位崩溃原因并快速修复崩溃问题.

用户通常通过文本描述、截图、录制视频等形式提供崩溃复现的步骤, 或直接提供堆栈跟踪信息来描述应用崩溃的情况. 现有研究主要利用崩溃报告的两类信息进行崩溃复现. 第 1 类方法侧重于基于复现步骤的崩溃复现, 通常依赖用户提供的文本描述或视觉记录等包含逐步指导信息的复现步骤[8−13]. 对于文本描述提供的复现步骤, 一些研究[8−12]通常设计用于解析复现步骤的预定义语法模式, 利用自然语言处理技术, 从中提取有用信息, 并采用动态探索策略, 在当前页面中匹配可能触发崩溃的相关图形用户界面 (graphical user interface, GUI/UI) 事件. 此外, 针对例如视频以视觉记录形式提供的复现步骤, GIFDroid[13]通过图像处理技术将视频的关键帧映射到 GUI 状态, 并基于 UI 转换图静态搜索视频中可能缺失的步骤. 第 2 类方法主要依赖堆栈跟踪信息实现崩溃复现. 例如, CrashTranslator[14]通过分析堆栈跟踪中的崩溃相关信息, 结合大型语言模型 (large language model, LLM) 预测可能的探索步骤. Mole[15]则对被测应用进行静态分析, 跟踪到崩溃点的可达路径中的相关控件 (widget), 从而实现从堆栈跟踪信息中自动复现崩溃.

尽管这些崩溃复现方法取得了一定的进展, 但在实际应用中仍存在以下局限性, 影响着崩溃复现的成功率和效率. 第一, 基于复现步骤的方法高度依赖用户提供完整的操作信息. 实际上, 用户提交的崩溃报告往往不完整, 缺乏详细的操作步骤, 甚至可能仅包含崩溃的堆栈跟踪信息, 导致复现困难. 报告标题中可能包含相关操作信息, 但这些信息通常较为有限, 难以提供完整的行为指导. 第二, 仅依赖堆栈跟踪难以准确定位崩溃组件 (component), 缺乏明确的方向指导动态探索过程. 例如, 在图 1 中 AnkiDroid 的问题 ID 为 4586[16]崩溃案例中, 由于堆栈跟踪中崩溃语句未包含显式的应用组件信息, CrashTranslator 无法定位到崩溃发生的具体活动 (activity), 从而无法有效引导探索. Mole 尝试通过分析 GUI 控件的属性状态来寻找通往崩溃点的可行路径, 在崩溃复现中关注可达路径中的相关控件. 然而, 由于底层调用图构造的复杂性, Mole 在依赖调用图定位崩溃点时可能会出现偏差. 此外, 现有研究忽略了触发事件对被测应用组件交互上下文的影响. 虽然 CrashTranslator 基于应用所有活动, 利用 LLM 预测下一个可达活动, 但其预测的活动转换可能并不真实存在, 进而影响崩溃复现的效率. 第三, 现有研究未能充分利用崩溃报告中的多维信息. 当缺乏复现步骤并且堆栈跟踪中的指导性信息不足时, 未能结合其他关键信息 (如标题中的自然语言描述), 导致搜索空间过大. 应用的 GUI 页面通常包含大量可交互控件, 缺乏关于崩溃场景信息的指导会使动态探索过程更具不确定性, 增加了触发崩溃事件序列的难度.

针对以上问题, 本文提出一种组件感知的安卓应用崩溃自动复现方法 CReDroid, 能够结合崩溃报告的标题信息和堆栈跟踪来有效地复现崩溃. 该方法融合了基于复现步骤与基于堆栈跟踪的两类方法优势, 不仅借助堆栈跟踪辅助崩溃组件定位, 还结合标题中的关键操作和堆栈信息引导崩溃路径的探索. 首先, 借助 LLM 从崩溃报告的标题中提取关键操作, 以此代表崩溃场景的核心信息. 其次, 使用自动应用探索工具对被测应用进行分析, 构建组

件转换图 (component transition graph, CTG), 并结合组件转换相关的意图 (Intent) 发送摘要和堆栈跟踪的动态异常信息来定位目标崩溃组件. 然后, CReDroid 通过以下 3 方面为当前 GUI 页面上的可操作控件分配分数: (1) 页面可达分数, 从 CTG 中检索从当前页面到达崩溃组件的所有可达路径并进行排序, 基于最佳路径预测控件触发的目标组件来计算匹配分数; (2) 语义相似度分数, 利用崩溃报告标题中的关键操作和堆栈跟踪中的类依赖, 匹配最可能导致崩溃的 GUI 控件; (3) 探索优化分数, 基于强化学习结合交互记录来分配分数, 调整页面可达分数和语义相似分数可能不准确的情况, 实现全局探索优化. 另外, CReDroid 结合当前页面所属组件与崩溃组件的上下文关系, 设计一种自适应评分策略, 以综合页面可达分数和语义相似度分数来选择最优控件触发操作, 以针对性地进行崩溃复现. 整个过程以迭代方式执行, 直至触发目标崩溃, 并最终生成自动重放脚本和附带人类可读图片解释的文本描述复现步骤.


图 1 AnkiDroid 中问题 ID 为 4586 崩溃报告对应的崩溃堆栈跟踪示例

为评估 CReDroid 的有效性和效率, 我们在 3 个数据集中收集了 74 个崩溃报告, 并与传统的自动化测试工具 Monkey[17]、APE[18]及最先进的崩溃复现工具 CrashTranslator[14]、ReCDroid[9]和 ReproBot[8]进行了对比实验. 实验结果表明, CReDroid 成功复现了 57 个崩溃报告, 分别比 CrashTranslator、ReCDroid、ReproBot、Monkey 和 APE 多复现 13、25、27、30 和 17 个. 在复现效率上, CReDroid 成功复现相同崩溃的平均用时分别比 CrashTranslator、ReCDroid、ReproBot、Monkey 和 APE 减少 26.71%、94.96%、71.65%、84.72% 和 88.56%.

本文的主要贡献总结如下.

● 提出了一种组件感知的安卓应用崩溃自动复现方法 CReDroid, 能够结合崩溃报告的标题信息和堆栈跟踪来有效地复现崩溃.

● 通过构建 CTG, 结合静态组件交互信息和堆栈跟踪的动态异常信息定位崩溃组件, 设计基于组件上下文关系的自适应评分策略为控件分配选择优先级, 并结合强化学习优化动态探索过程以减轻预测不准确的影响.

● 在 3 个数据集的 74 个崩溃报告上进行了实验评估, 结果表明 CReDroid 在崩溃复现的有效性和效率方面均优于现有最先进技术.

本文第 1 节通过引入一个示例介绍本文研究的动机. 第 2 节详细介绍提出的组件感知的安卓应用崩溃自动复现方法. 第 3 节描述实验设置. 第 4 节分析实验结果以验证所提方法在崩溃复现中的性能. 第 5 节讨论方法的局限性及潜在的有效性威胁. 第 6 节介绍崩溃复现领域的相关工作. 第 7 节总结全文并展望未来研究方向.

## 1 研究动机示例

AnkiDroid 是一款安卓开源学习软件, 旨在帮助用户高效记忆内容, 提供卡片自定义、同步功能以及多种学习模式. 图 2(a) 展示了与图 1 中堆栈跟踪对应的触发 AnkiDroid 的#4586 崩溃的事件序列: 用户打开应用后, 点击“Add”按钮进入添加笔记页面, 随后点击“Attach mutimedia content to the Front filed”按钮, 选择“CAMERA”添加图片时应用发生崩溃. 在对应用代码进行分析后发现, 在步骤 5 中点击“CAMERA”按钮后, 图 2(b) 所示的事件处理器 (onClick) 创建了隐式相机 Intent 对象以启动相机应用并捕获图片. 该 Intent 对象包含拍照后图片的保存位置 (第 3–8 行), 并通过 Android 系统提供的启动活动方法与相机应用通信 (第 9 行), 触发相机启动. 然而, 为防止隐私泄露和安全风险, 自安卓 7.0 (Nougat) 起, 系统限制了应用间直接通过 file://URI 直接共享文件. 当应用尝试将本地文件路径 (file://URI) 暴露给其他应用时, 会触发 android.os.FileUriExposed Exception 异常. 这一问题, 如图 1 中堆栈跟踪所示, 是由于违反安卓系统安全策略而引发的崩溃错误.

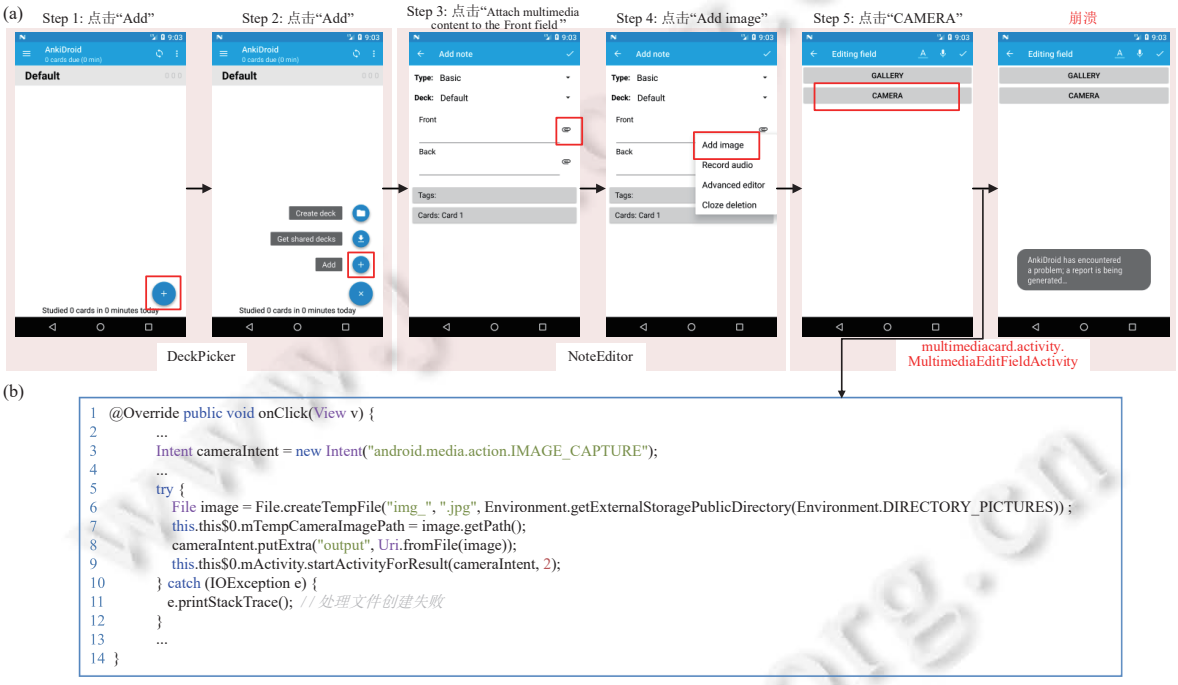



图 2 AnkiDroid 应用问题 ID 4586 崩溃对应的复现步骤和点击“CAMERA”按钮后创建的隐式 Intent 实例

以图 2 所示的 AnkiDroid 应用中的崩溃为例, 使用传统的自动化测试工具 Monkey[17]、APE[18]及代表性的崩溃复现工具 ReCDroid[9]、ReproBot[8]和 CrashTranslator[14]进行了 30 min 的测试, 其中 ReCDroid 和 ReproBot 以崩溃报告标题作为输入, CrashTranslator 以堆栈跟踪作为输入. 结果显示, 只有 ReCDroid 成功触发了崩溃, 复现时间接近 5 min. 分析表明, 由于 AnkiDroid 应用的 GUI 页面包含大量控件, 且触发崩溃需要严格的事件顺序, 上述工具难以高效触发该崩溃. 具体来说, Monkey 生成的事件完全随机, 无法按照特定的逻辑顺序执行 GUI 操作; APE 则在与崩溃无关的路径上消耗了大量测试资源, 导致效率低下; ReCDroid 基于 813 个缺陷报告语料库构建了 22 种语法模式, 包括动作、目标控件及输入内容等要素提取模板. 该策略依赖明确的操作描述和完整句法结构. 对于图 1 中所示崩溃报告标题“Crash on add picture from camera on Android 7”, 由于描述简短且未明确指示动作与控件关系 (如缺少“click camera”等指令), ReCDroid 难以通过预定义语法模式提取必要的“add”和“camera”关键操作信息.然而, 该方法通过动态事件序列推导补全缺失步骤, 从而成功触发崩溃; ReproBot 则不依赖于固定的语法模式, 而采用句法成分分析和连词语义推理提取操作元素, 并根据隐含的时间关系推断操作执行顺序. 在该示例中,

ReproBot 只从标题中识别出了动词关系, 即“add”动作作用于“picture”, 而上下文信息“from camera on Android 7”未包含可直接转化为具体操作的内容. 同时, 由于标题中缺乏时序连接词, 现有解析结果无法推导出具体的操作流程, 因而未能生成可复现的步骤. 尽管采用强化学习优化事件选择, 但复杂的 UI 事件组合仍使得其难以找到正确的触发路径; CrashTranslator 根据堆栈跟踪中提取崩溃位置, 但在该示例中仅能从应用级别的崩溃语句中检索到崩溃相关类 com.ichi2.anki.multimediacard.fields.BasicImageFieldontroller, 无法定位具体崩溃组件. 为引导崩溃路径探索, CrashTranslator 需向 LLM 提供当前页面和目标崩溃页面, 尝试预测应跳转至的下一个页面, 并生成可能触发该跳转的控件操作. 然而, 由于缺乏准确的目标指引, LLM 生成的跳转操作偏向不相关的页面, 导致动态探索路径偏离目标崩溃. 尽管 ReCDroid 成功复现了该崩溃, 但其高度依赖于复现步骤, 若标题不包含具体的操作信息, 则会大大增加触发崩溃的难度.

当崩溃报告缺乏逐步指导的复现步骤且堆栈跟踪信息不足时, 充分利用多源信息来触发崩溃尤为重要. 对 AnkiDroid 的#4586 崩溃的分析表明, 崩溃发生在通过隐式相机 Intent 启动相机应用捕获图片的过程中, 其中意图发送的信息可用于辅助定位崩溃组件. 结合意图发送信息与堆栈跟踪的动态异常信息, 可以有效定位崩溃实际发生的组件. 另外, 图 2(a) 中的事件触发了多个活动的交互, 而图 1 中崩溃报告标题“Crash on add picture from camera on Android 7”中包含的关键操作“add picture from camera”分别对应活动 NoteEditor 和崩溃活动 multimediacard.activity.MultimediaEditFieldActivity 中的步骤 4 和步骤 5 操作. 这表明, 应用组件的交互信息与崩溃报告标题中的关键操作对于指导崩溃探索具有重要意义.

因此, 准确定位被测应用的崩溃组件, 并结合目标组件的上下文信息调整探索方向, 是实现高效崩溃复现的关键. 在此过程中, 融合基于复现步骤与基于堆栈跟踪的两类方法以设计崩溃路径的探索策略显得尤为必要. 基于这一思路, 我们对被测应用构建 CTG, 结合组件转换中的意图发送信息和堆栈跟踪中的动态异常信息来定位崩溃组件. 然后, 通过分析应用当前状态与崩溃组件的上下文关系, 自适应地结合 CTG 中到崩溃组件的可达路径和崩溃报告标题中的关键操作信息计算当前 GUI 控件的优先级分数, 从而动态调整探索方向来触发崩溃.

## 2 组件感知的安卓应用崩溃自动复现方法

本文所提组件感知的安卓应用崩溃自动复现方法 CReDroid 框架如图 3 所示. 首先, 分析崩溃报告的标题和堆栈跟踪, 分别提取标题中的关键操作信息和崩溃相关 API; 其次, 对被测应用构建 CTG, 结合崩溃相关 API 定位崩溃组件; 最后, 基于标题中的关键操作与 CTG 中崩溃组件的可达路径, 设计自适应评分策略, 通过分析当前页面组件与崩溃组件的上下文关系, 为 GUI 控件分配选择优先级分数, 并引入强化学习设计探索优化分数来实现全局优化, 最终输出自动重放脚本及人类可读的复现步骤.

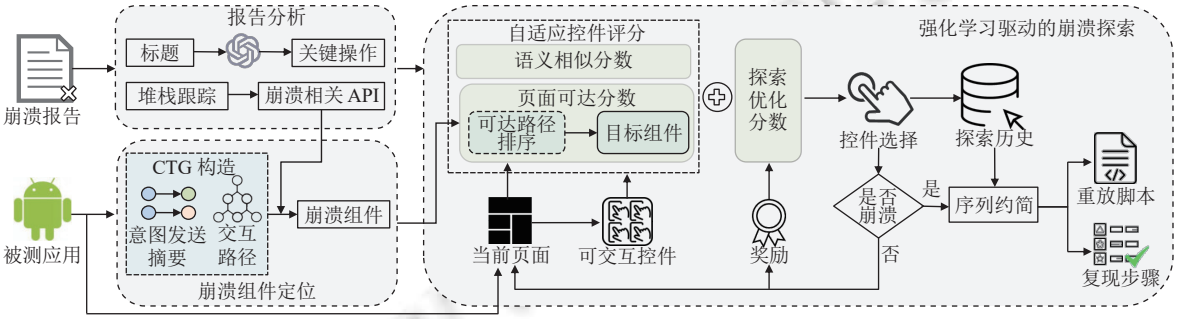



图 3 组件感知的安卓应用崩溃自动复现方法框架图

### 2.1 报告分析

CReDroid 首先对崩溃报告进行分析, 通过对标题和堆栈跟踪的预处理, 为崩溃复现准备必要的信息. 具体来说, 崩溃复现中将使用报告的两类关键信息.

● 标题关键操作. 与复现步骤类似, 崩溃报告的标题可能包含关键的 GUI 控件信息, 这些信息能够揭示触发崩溃的上下文信息并有效指导动态探索. 然而, 用户提交的标题格式多样, 给全面提取标题中的关键 GUI 控件信息带来了挑战. 考虑到生成式 AI 模型[19]的快速发展, 这些模型可以通过特定的指令或提示中回答, 为研究人员和开发人员提供指导. 我们利用预训练 LLM 来提取标题中包含的 GUI 控件信息. 具体而言, 我们向 LLM 提供报告的标题, 并要求其输出标题中与 GUI 控件相关的关键操作信息. 为此, 我们依照常规的提示模板[20,21], 经过 3 个作者的共同编写和讨论, 最终确定了适用于该任务的提示模式 (如表 1 所示). 以图 1 中的报告标题为例, 当输入标题“Crash on add picture from camera on Android 7”后, 向 LLM 提问“Which words in the title are likely to be related to the app UI elements?”, LLM 将预测并提取关键操作“add picture”和“camera”.

表 1 提示生成规则的示例

| 任务 | 提示 | 实例 | 预测 |
|---|---|---|---|
| 预测标题关键操作 | The crash report title is: <Title>. Which words in the title are likely to be related to the app UI elements? | The crash report title is: Crash on add picture from camera on Android 7. Which words in the title are likely to be related to the app UI elements? | add picture, camera |

为了进一步提高模型提取关键操作的准确性, 我们构建了一个涵盖多种交互场景的数据集用于微调 LLM. 具体来说, 我们从 GitHub 上收集了 50 个不同类型的应用程序, 以覆盖广泛的使用场景, 并提取真实用户报告中带有明显崩溃标识 (如包含“Crash”“can not”等词汇) 的标题. 通过人工标注这些标题中的关键操作信息, 帮助 LLM 学习不同的标题语法模式并推测所需关键操作. 需要注意的是, LLM 的微调过程仅需进行一次, 微调完成后的模型可直接应用于不同应用的标题关键操作预测任务.

● 崩溃相关 API. 堆栈跟踪提供了关于应用异常的信息, 我们提取堆栈跟踪中包含被测应用包名的行, 以定位与崩溃相关的 API. 例如, 在图 1 中的堆栈跟踪中, 提取到的崩溃相关 API 包括“com.ichi2.anki.Anki Activity.startActivityForResult”和“com.ichi2.anki.multimediacard.fields.BasicImageFieldController$2.onClick”.

### 2.2 崩溃组件定位

Android 应用由活动 (activity)、服务 (service)、广播接收器 (broadcast receiver) 和内容提供方 (content provider) 这 4 大基本组件构成. 这些组件通常通过 Intent 机制实现相互转换与通信. Intent 作为一种声明式对象, 不仅可以指明当前组件需要执行的具体动作, 还可以在不同组件之间传递数据, 从而协作以实现应用的复杂功能. 然而, 传统 Android 应用自动化测试工具通常模拟事件交互以覆盖尽可能多的可执行路径, 这种方式在与崩溃无关的路径上浪费了大量测试资源, 难以快速生成触发目标崩溃的有效事件序列. 为了解决这一问题, 我们引入组件转换图对被测应用进行建模, 捕获组件间的转换关系, 并通过 Intent 与组件的触发关系确定目标崩溃组件.

● CTG 构造. CTG 是应用探索的重要特征之一, 用于描述组件之间的转换和交互关系. 在一个 CTG 中, 节点集表示 Android 应用的基本组件 (activity、service、broadcast receiver 和 content provider) 及其特定上下文状态, 边集表示组件之间的交互关系, 并分为显式边 (explicit) 和隐式边 (implicit). 每条边携带详细的交互属性 (如 action、extras 和 flags 等), 用于描述 Intent 调用的细节. 我们采用 ICCBot[22]工具构建 CTG, 这是一种广泛使用的动态探索工具. 图 4 展示了 ICCBot 对 AnkiDroid 构建 CTG 的示例, 具体对应图 2 中“step 5”的操作, 即点击“CAMERA”按钮的 Intent 发送摘要. 示例中, 起点 source 为 com.ichi2.anki.multimediacard.activity.MultimediaEditFieldActivity, 表示触发事件的初始组件为一个 activity, 终点 destination 为目标 android.media.action.IMAGE_CAPTURE, 通过匹配 Intent 的 action 和参数触发, 类型为系统相机功能. 隐式边类型 Act2class 表示通过 action 匹配的间接调用关系.

● 定位目标崩溃组件. 为了高效定位崩溃组件, 我们结合堆栈跟踪的动态异常信息和静态分析的组件转换信息, 确定崩溃页面. 算法 1 描述了定位崩溃组件的过程, 输入为编译被测应用 APK 文件后可获得的应用信息 app_info、崩溃堆栈 TR、 崩溃相关 API 集合 $TR_{API}$ 和组件转换图 CTG, 输出为崩溃组件 $TR_{com}$. 首先, 初始化崩溃组件 $TR_{com}$ (第 1 行) . 其次, 结合崩溃相关 API 集合 $TR_{API}$, 基于应用包名从崩溃堆栈中筛选出与崩溃相关的类 $TR_{class}$, 由于最后一个类通常是直接导致崩溃的触发点, 提取最后一个崩溃相关类 $TR_{lastclass}$ 以缩小定位范围 (第 2 行). 然后,

利用 CTG 中的意图发送摘要信息和节点信息, 初步定位潜在的崩溃发生组件 (第 3–5 行). 若初步定位未能确定崩溃组件, 则进一步解析应用的 AndroidManifest 文件, 提取所有活动组件 A, 结合崩溃堆栈信息, 通过基于正则表达式的活动名称匹配扩展推断潜在的崩溃组件 (第 6–9 行) . 最终返回定位的崩溃组件 $TR_{com}$, 为后续崩溃复现提供动态探索的方向.

```
                                                          崩溃组件
<source name="com.ichi2.anki.multimediacard.activity.MultimediaEditFieldActivity"
        type="Activity">
  ...
  <destination ICCType="implicit"
               name="android.media.action.IMAGE_CAPTURE, "
               edgeType="Act2Class"                                    崩溃点
               method="&lt;com.ichi2.anki.multimediacard.fields.BasicImageFieldController$2:
                       void onClick(android. view. View)&gt;"
               instructionId="4"
               unit="virtualinvoke$r11.&lt;com.ichi2.anki. multimediacard.activity.Multimedia
                     EditFieldActivity:  void startActivityForResult(android.content. Intent,int)&gt;(r2, 2)"
               action="android.media. action.IMAGE_CAPTURE"
               extras="Parcelable-output"
               flags=""/>
</source>
```

图 4　一种隐式 Intent 发送摘要示例

以图 1 中的崩溃堆栈为例, 通过分析可确定 com.ichi2.anki.multimediacard.fields.BasicImageFieldController$2 为崩溃相关的关键类. 进一步解析 CTG 的意图发送摘要, 发现 BasicImageFieldController$2 在处理用户交互事件时调用了 startActivityForResult 方法, 并构造了指向 MultimediaEditFieldActivity 的隐式 Intent. 因此, 可以确定崩溃组件 $TR_{com}$ 为 com.ichi2.anki.multimediacard.activity.MultimediaEditFieldActivity. 通过综合分析组件交互场景, 包括隐式 Intent、内部类调用, 以及结合崩溃堆栈和应用配置等多种信息源, CReDroid 能够更准确地定位崩溃组件, 为后续动态探索提供确切的指导方向.

**算法 1**. 定位崩溃组件.

输入: 应用信息 app_info, 崩溃堆栈 TR, 崩溃相关 API 集合 $TR_{API}$, 组件转换图 CTG;
输出: 崩溃组件 $TR_{com}$.

1. $TR_{com} = \varnothing$ /*初始化崩溃发生组件*/
2. $TR_{lastclass}$, $TR_{class}$ = GetTraceClass(app_info.pkg, TR, $TR_{API}$) /*从堆栈中获取崩溃相关类和最后一个崩溃相关类*/
3. $C_{CTG\_s}$ = GetComOfSummary(CTG.Summary, $TR_{lastclass}$) /*根据意图发送摘要定位崩溃组件*/
4. $C_{CTG\_n}$ = GetComOfSummary(CTG.Nodes, $TR_{lastclass}$) /*根据 CTG 组件定位崩溃组件*/
5. $TR_{com} = TR_{com} \cup C_{CTG\_s} \cup C_{CTG\_n}$ /*将通过 CTG 定位的崩溃组件加入 $TR_{com}$*/
6. IF len($TR_{com}$) == 0 /*若尚未成功定位到崩溃发生组件, 则进一步扩展*/
7.  A = GetAllActivities(apk.manifestXml) /*分析 AndroidManifest 获取所有活动组件*/
8.  $C_{TR}$ = GetComOfTR(A, TR) /*根据应用活动和崩溃堆栈定位崩溃组件*/
9.  $TR_{com} = TR_{com} \cup C_{TR}$ /*将定位的崩溃组件加入 $TR_{com}$*/
10. RETURN $TR_{com}$

### 2.3　强化学习驱动的崩溃探索

强化学习驱动的崩溃探索旨在通过所定位的崩溃组件和崩溃报告的关键操作信息, 结合被测应用的 CTG 来全局搜索可能触发崩溃的事件序列. 为了触发崩溃, 可与崩溃报告标题中相似的 GUI 控件进行交互. 例如, 在步骤 4 中点击“Add image”, 在步骤 5 中点击“CAMERA”尝试启动相机, 如图 2(a) 所示. 然而, 到达崩溃页面可能还需要额

外的特定事件序列, 这些事件可能未在崩溃报告标题中记录, 例如在步骤 3 中的上传附件按钮操作与标题“Crash on add picture from camera on Android 7”无直接关联. 为解决此问题, 我们利用 CTG 构建探索路径的搜索空间, 从而到达崩溃发生的页面. 正如第 2.2 节所述, CTG 描述了组件间的交互关系. 基于此, 我们设计一种自适应评分策略, 为当前页面的 GUI 控件分配分数, 驱动探索过程. 首先, 通过 CTG 检索从当前页面到达崩溃组件的路径找到崩溃相关的 GUI 页面. 然后, 借助标题的关键操作选择页面中的 GUI 控件尝试触发崩溃. 此外, 为应对自适应控件评分可能产生的不精确匹配, 设计强化学习驱动崩溃探索过程以实现全局搜索, 进一步优化探索方向来触发崩溃.

2.3.1 自适应控件评分

被测应用的 GUI 页面通常包含大量可操作控件, 而选择最可能触发崩溃的控件是一项具有挑战性的任务. 尽管崩溃报告的标题中提供了关键操作信息, 其内容精简可能未涵盖所有相关操作. 同时, 尽管目标崩溃组件为探索提供了方向, 但被测应用中存在大量可达路径, 开发人员难以全面检查所有路径[23,24]. 因此, 我们基于当前页面与目标崩溃组件的上下文关系, 设计了一种自适应评分策略, 结合页面可达分数和语义相似度分数, 为 GUI 控件分配优先级分数, 以确定操作顺序.

算法 2 描述了自适应控件评分的过程, 输入为组件转换图 CTG、标题关键操作 title_actions、崩溃组件 $TR_{com}$ 和崩溃相关类 $TR_{class}$, 输出为 GUI 控件分数 $scores_w$. 首先, 初始化所有 GUI 控件的分数 $scores_w$ (第 1 行). 其次, 获取被测应用当前 GUI 页面的层次结构, 提取所有可操作候选控件信息, 并通过安卓调试桥 (Android debug bridge, ADB) 命令获取当前组件信息 (第 2, 3 行). 然后, 根据当前组件与目标崩溃组件的上下文关系, 结合自适应策略对候选控件进行评分 (第 4–13 行). 具体而言, 若当前组件即为崩溃组件 $TR_{com}$, 通过计算控件名称与标题关键操作的语义相似度作为控件分数 (第 4, 5 行); 若当前组件不为目标崩溃组件时, 则根据直接可达页面是否为崩溃组件, 采用不同策略对控件进行评分 (第 6–13 行). 在此过程中, CReDroid 从 CTG 中检索当前组件与崩溃组件之间的候选可达路径, 并基于路径中组件与标题关键操作的语义相关度及路径长度排序, 选取相关度高且距离较短的最佳路径 (第 7, 8 行). 基于最佳路径, 若下一个可达组件为崩溃组件, 结合堆栈类依赖信息和标题关键操作计算 GUI 控件的语义相似度分数 (第 9–11 行); 若下一个可达组件非崩溃组件, 结合控件与后两个可达组件及目标崩溃组件前驱组件的匹配度, 计算页面可达分数 (第 13 行).

**算法 2**. 自适应控件评分.

输入: 组件转换图 CTG; 标题关键操作 title_actions, 崩溃发生组件 $TR_{com}$, 崩溃相关类 $TR_{class}$;
输出: GUI 控件分数 $scores_w$.

1. $scores_w$ = {} /*初始化页面 GUI 控件分数*/
2. $w_{cans}$ = getCandidates() /*获取当前 GUI 页面的所有候选 GUI 控件*/
3. cur_com = getCurCom()/*获取当前 GUI 页面的组件信息*/
4. IF cur_com == $TR_{com}$ do /*如果当前组件为目标崩溃组件, 计算语义相似度分数*/
5. $scores_w$ = getSemanticScore($w_{cans}$, title_actions, $scores_w$)
6. ELSE /*否则推断可达页面*/
7. path = getReachPath(cur_com, $TR_{com}$, CTG) /*根据 CTG 检索从当前组件到崩溃组件的可达路径*/
8. path = rankPath(path, title) /*结合距离和语义信息对可达路径重新排序*/
9. next_com, after_com = getNextCom(path) /*从排序后的最佳路径中推断后两个可达组件*/
10. IF next_com == $TR_{com}$ do /*如果下一个可达组件为崩溃组件, 则根据堆栈类依赖信息和崩溃标题计算语义相似度分数*/
11. $scores_w$ = getSemanticScore($w_{cans}$, title_actions, $TR_{class}$, $scores_w$)
12. ELSE /*否则根据后两个可达组件计算页面可达分数*/

13. $scores_w$ = getPageReachScore($w_{cans}$, next_com, after_com, CTG, $scores_w$)
14. RETURN $TR_{com}$

以图 2(a) 为例, 当进入被测应用主页时, 当前组件为 DeckPicker, 目标崩溃组件为 MultimediaEditFieldActivity. 通过 CTG 检索, 从当前组件 DeckPicker 到崩溃组件 MultimediaEditFieldActivity 的多条可达路径包括“DeckPicker→NoteEditor→MultimediaEditFieldActivity”“DeckPicker→CardBrowser→NoteEditor→MultimediaEditFieldActivity”和“DeckPicker→Reviewer→NoteEditor→MultimediaEditFieldActivity”等. 基于标题关键操作和路径距离, 选取“DeckPicker→NoteEditor→MultimediaEditFieldActivity”为最佳路径, 如图 5 所示的可达路径. 根据此路径, 由于下一个可达页面 NoteEditor 不是崩溃组件, 计算 GUI 控件与 NoteEditor 及 MultimediaEditFieldActivity 组件的语义相似度, 并结合 GUI 控件与崩溃组件前驱组件 NoteEditor 的语义相似度, 得到页面可达分数作为最终控件分数. 根据分数, 选择控件“Add”进行操作. 以此类推, 当点击“Add image”后, 被测应用转换为目标组件 MultimediaEditFieldActivity, CReDroid 仅计算当前页面所有候选控件与标题关键操作的语义相似度, 以确定控件分数. 考虑在自然语言理解任务中表现出卓越的语义建模能力, 本文采用预训练的 BERT 模型[25]计算语义相似度分数.


图 5 自适应计算控件得分的示例

### 2.3.2 探索优化分数

自适应控件评分可能存在一定的不准确性, 影响崩溃复现的有效性. 为了解决这一问题, 我们将强化学习引入全局驱动的场景中, 将崩溃复现建模为一个马尔可夫决策过程 (Markov decision process, MDP). 通过引入 Q-learning 方法[26]优化探索得分, 从而提升整体探索规划的准确性, 弥补预测 GUI 控件触发崩溃的不足.

具体而言, 探索过程被定义为一个 MDP 实例, 形式化为一个四元组<$S$, $A$, $P$, $R$>, 其各部分定义如下.

$S$: 状态集 (states). 被测应用的每个 GUI 页面被定义为一个独立的状态, 其中布局着可供操作的 GUI 控件, 用于实现特定功能.

$A$: 动作集 (actions). 每个状态下可执行的事件构成动作集. CReDroid 在每个页面中提取布局层次并分析 GUI 控件属性获取所有可交互控件, 与控件的交互 (如单击、长按或输入文本) 构成可执行事件集. 每个事件被记录为 Q-table 中的状态-动作对 ($s$, $a$), 其中 $a$ 表示为状态 $s$ 的一个可执行事件. 为了在崩溃探索的过程中打破局部最优陷阱, CReDroid 采纳 $\epsilon$-贪婪策略[27]选择下一个动作, 这是 Q-learning 的标准方法. 具体而言, CReDroid 以概率 $1-\epsilon$ 选择 $Q$ 值最高的动作, 或以剩下的 $\epsilon$ 概率随机选择一个 $Q$ 值较低的动作进行探索. 根据先前的经验[8,28], 在初始阶段, 为了专注于当前最优 GUI 控件的探索, $\epsilon$ 被设置为一个接近 0 的数值. 然而, 在崩溃探索的过程中, 错误选择某个最优 GUI 控件可能导致探索陷入重复. 在这种情况下, CReDroid 将会动态调整 $\epsilon$ 接近 1 的数值, 从而提升较低得分动作的探索机会, 避免局部最优.

$P$: 转换函数 (transition function). 转换函数描述动作执行后被测应用状态的转换. 当执行一个动作 $a_t$ 后, 被测应用将从当前状态 $s_t$ 转换为新状态 $s_{t+1}$. 该转换记录为三元组<$s_t$, $a_t$, $s_{t+1}$>, 并存储在转换函数 $P$ 中.

$R$: 奖励函数 (reward function). 当执行一个动作 $a_t$ 之后, 奖励函数 $R$ 将生成一个数值用于评估所执行动作 $a_t$ 的质量. 例如, 所交互的 GUI 控件是否与崩溃或者崩溃报告标题相关. 为了有效地优化崩溃探索的过程, CReDroid 为每个动作设计了奖励和惩罚策略. 奖励函数被定义为以下几个方面奖励的总和.

● 触发崩溃奖励. 当某动作触发崩溃时, 通过崩溃日志验证该动作是否覆盖目标崩溃组件或涉及崩溃 API、类以及异常语句. 验证通过时, 该动作将获得极大正向奖励, 这表明该动作很可能与目标崩溃相关联. 当被测应用再次到达该状态时, CReDroid 会优先选择该动作执行, 从而进一步提高触发目标崩溃的可能性.

● 新状态和组件奖励. 若某动作成功探索到一个新的状态, 则会获得正向奖励, 以激励探索新状态的行为. 每个 GUI 页面所属的组件代表着崩溃探索的方向. 当新状态属于可达路径中的组件时, CReDroid 会赋予该动作正向奖励. 若新状态属于崩溃组件, 则获得更高奖励, 鼓励更深入的探索.

● 标题操作奖励. 若动作操作的 GUI 控件与崩溃报告标题中的关键操作相关联, 则获得正向奖励, 表明该动作可能是触发崩溃的关键操作, 有助于缩小探索范围.

● 重复或失败的状态处罚. 当触发一个动作使得应用转换到一个已经探索过的状态, 该动作会受到惩罚, 因为它触发崩溃的可能性较低. 另外, 如果某动作使得被测应用跳转到无关页面 (如打开浏览器) 或触发非目标崩溃, 则会受到大的惩罚. 这些惩罚旨在避免 CReDroid 在不可能触发目标崩溃的动作上浪费探索资源.

每个状态-动作对的 $Q$ 值存储在 Q-Table 中, 记录与崩溃相关的探索信息并帮助记忆有价值的动作. 当被测应用第 1 次探索新状态时, 该状态中的所有动作值会初始化为 0. 随着探索的进行, 根据奖励或惩罚通过 Bellman 函数更新 $Q$ 值:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha(r_t+\gamma Q^*(s_{t+1},a_{t+1})-Q(s_t,a_t)) \tag{1}$$

其中, $\alpha$ 为学习率, 控制每次更新的幅度, 设为 0.1; $r_t$ 为当前动作的奖励; $\gamma$ 为折扣因子, 权衡当前奖励与未来奖励, 设为 0.9; $Q^*(s_{t+1}, a_{t+1})$ 指的是转换后状态 $s_{t+1}$ 中所有动作的最大 $Q$ 值, 反映后续探索的潜在收益.

#### 2.3.3 序列约简

在崩溃复现过程中, 所有与 GUI 控件的交互操作都会被记录在探索历史中, 该历史在被测应用重新启动后会被清空. 在成功触发目标崩溃时, 探索历史中保存了从被测应用启动到崩溃发生的所有 GUI 操作. 然而, 这些记录往往包含重复或者循环的冗余交互[29,30], 无法直接反映复现崩溃的最直接步骤.

为了便于快速定位和触发崩溃, CReDroid 对探索历史中的交互序列进行自动化约简, 去除多余的重复和循环操作, 仅保留必要的关键步骤. 最终, 约简后的交互序列被转换为自动重放的脚本和人类可读的复现步骤. 如图 2 所示, 复现步骤以“图像+文本”的形式呈现. 具体而言, 每个复现步骤均通过在页面截图上用矩形框标注触发的目标 GUI 控件, 并以“事件类型+控件名称”的文本描述对应的操作.

## 3 实验设置

为了评估 CReDroid 在复现崩溃方面的整体性能, 本文设计了如下 4 个研究问题.

RQ1: CReDroid 在崩溃复现有效性与效率上分别表现如何?

RQ2: 这些最先进的技术在复现崩溃时是否互补?

RQ3: 自适应控件评分模块是否能够提高 CReDroid 崩溃复现的效果?

RQ4: CReDroid 生成的复现步骤是否能够正确触发崩溃?

### 3.1 实验数据

本文构建了一个包含 74 个崩溃报告的数据集, 用于评估所提出的 CReDroid. 该数据集的崩溃报告来源于以下 3 个相关工作: (1) ReCDroid[9]的评估数据集, 包含 33 个崩溃报告; (2) CrashTranslator[14]的评估数据集, 包含 20 个崩溃报告; (3) 从 AndroR2[31]数据集中筛选出 21 个崩溃报告, 这些报告与前两者不重复. 本研究仅关注崩溃报告, 不包含其他类型的非崩溃报告 (如界面显示问题的缺陷报告).

为确保数据集的完整性和准确性, 我们对每个崩溃报告对应版本的应用进行了实际交互, 以验证所选报告的可复现性, 并排除了重复的报告. 需要说明的是, 这 74 个崩溃报告 (33+20+21) 中并非全部包含堆栈跟踪信息. 对于缺乏堆栈跟踪的报告, 我们通过 GitHub 提供的复现步骤手动触发崩溃, 并从运行日志中提取相应的堆栈跟踪信息. 这些堆栈跟踪随后被用于在运行过程中验证崩溃是否能够成功复现. 在实验过程中, 本文仅使用崩溃报告中的标题和堆栈跟踪信息进行崩溃复现, 未参考报告中可能包含的复现步骤或截图. 此外, 崩溃报告在开发者确认和修复过程中可能会被修改, 以更准确地描述崩溃场景. 为避免被修改的标题无法真实反映用户的实际行为, 本文直接采用崩溃报告的初始标题构建标题数据集. 对于缺乏标题的崩溃报告, 则使用摘要作为替代.

### 3.2 实现细节及环境配置

本文使用 Python 3.9.21 实现了所提出的方法, 并基于多种现有工具进行了扩展, 以支持崩溃复现任务. 代码已发布于: https://github.com/liushuqi-2022/CReDroid. 在交互与数据预处理方面, 利用 Appium[32]与安卓应用进行交互, 提取当前页面的视图层次结构; 使用 Apktool[33]从 APK 文件中提取 AndroidManifest.xml 文件、资源文件和源代码; 结合数据流框架 Soot[34]和动态探索工具 ICCBot[22]构建 CTG, 支持页面间路径的分析. 在路径处理与语义分析方面, 采用 networkx[35]对获取的 CTG 进行操作, 检索页面间的可达路径; 通过 NLTK[36]对提取的文本数据进行预处理, 通过 Sentence-Transformers[37]模型生成控件描述的语义嵌入, 并结合 sklearn[38]计算嵌入之间的余弦相似度, 以衡量 GUI 控件的语义分数. 在崩溃日志记录与结果展示方面, 借助 LOGCAT[39]记录运行时异常日志, 提取触发崩溃的关键信息; 通过 OpenCV[40]在页面截图上标记交互的目标控件, 并以图文结合的方式直观展示复现步骤. 实验运行在 Windows 10 操作系统上, 设备配备 32 GB 内存和 Intel Core i7-12700 处理器. 测试环境使用 Android 4.4 至 8.1 版本的安卓模拟器, 用于验证 CReDroid 的性能.

此外, 随着训练数据规模的不断扩展, 基于大型语言模型的生成式 AI 在面对特定任务或提示时给出准确的回答, 并在多个领域展现卓越的性能. 本文使用在 OpenAI 官网发布的 ChatGPT 模型[41], 其基础模型是 GPT-3.5-turbo, 并通过自定义数据集结合官方 API 说明对模型进行微调. 微调后的模型被用于从崩溃报告的标题或者摘要中提取关键词, 为崩溃复现任务提供有效的辅助支持.

### 3.3 评估指标与评估设置

本文从有效性和效率两个维度对所提方法 CReDroid 的性能进行评估. 有效性通过计算在限定时间内成功复现崩溃报告的百分比 (即成功率) 进行验证, 效率则通过统计完成一次成功复现所需的时间 (即复现时间) 进行衡量.

针对 RQ1, 沿用 Huang 等人[14]的实验设置, 将崩溃复现的最长测试时间限定为 1 h. 为了减轻实验中因随机性导致的偏差, 每个崩溃报告在对应目标应用上分别运行 3 次, 并取平均值作为成功复现时间. 据我们所知, CReDroid 是第 1 个利用崩溃报告的标题和堆栈跟踪进行崩溃复现的方法. 为了全面评估 CReDroid 的性能, 我们选择了 5 种代表性工具作为基准方法进行对比, 即 CrashTranslator[14]、ReCDroid[9]、ReproBot[8]、Monkey[17]和 APE[18].

- CrashTranslator[14]. 该工具代表基于堆栈跟踪的崩溃复现, 与 CReDroid 最为接近. CrashTranslator 仅依赖堆栈跟踪信息, 通过 LLM 预测触发崩溃的探索步骤, 并结合强化学习缓解预测误差从而全面引导搜索. 在实验中, 我们仅提供崩溃堆栈跟踪作为其输入.
- ReCDroid[9]和 ReproBot[8]. 这两种工具依赖逐步指导的文本描述的复现步骤来实现崩溃复现. ReCDroid 采用预定义的语法模式解析报告中描述的事件, 并使用动态 GUI 探索合成事件序列以重现崩溃. ReproBot 通过时间归一化和重排序提取独立步骤, 并利用强化学习探索应用来匹配步骤和 UI 事件. 由于本研究未涉及详细复现步骤, 这些方法无法直接应用于我们的任务. 然而, 考虑到崩溃报告标题也是一种文本描述, 即使缺失详细步骤, 我们仍将其作为输入. 如果标题不可用, 则提供堆栈跟踪中的包含应用包的 API 信息, 不管方法是否能理解.
- Monkey[17]和 APE[18]. 这两种工具是典型的自动化 GUI 测试工具, 具备检测崩溃的能力. Monkey 通过随机事件 (如点击、滑动、输入等) 对应用进行无目标探索. APE 采用基于模型的测试方法, 利用运行时信息动态优化 GUI 抽象模型, 提高测试覆盖率. 这两种工具被广泛用于许多研究工作的基准对比方法.

针对 RQ2, 我们在 RQ1 的实验结果基础上, 进一步分析不同工具在崩溃复现任务中的互补性. 由于各工具在复现过程中依赖的崩溃报告信息不同, 我们将 74 个崩溃报告划分为两类: (1) 仅包含堆栈跟踪信息的崩溃报告 (9 个), 这些报告仅提供崩溃发生时的堆栈跟踪信息, 标题中通常仅包含关键的堆栈跟踪语句, 缺乏显式的用户操作描述; (2) 同时包含标题和堆栈跟踪信息的崩溃报告 (65 个), 这些报告除了提供完整的堆栈跟踪, 标题中还包含一定程度的操作描述. 基于此分类, 我们统计不同工具在这两类崩溃报告上的复现成功率, 并探讨其适用场景. 具体而言, 我们关注以下两个方面: 不同工具在两类崩溃报告上的复现能力, 分析是否存在方法在某类崩溃报告上的复现成功率更高; 每类崩溃复现工具的适用场景, 探讨是否存在某些崩溃仅能被特定工具复现, 并分析相同类别工

具之间的表现差异.

针对 RQ3, 我们重点分析自适应控件评分模块对成功崩溃复现的贡献. 该模块通过动态调整 GUI 控件的选择优先级, 影响方法的整体性能. 考虑到探索优化分数在避免局部最优解方面的重要性, 移除该分数可能导致探索陷入困境, 因此本研究仅评估其他两个分数的独立贡献. 具体而言, 自适应控件评分模块主要设计自适应评分策略根据当前页面与崩溃组件的上下文关系来为 GUI 控件计算分数, 包含以下两部分: 页面可达分数, 通过 CTG 预测下一个可达目标组件, 用于预测页面间的可达性; 语义相似度分数, 基于崩溃报告标题中的关键词和崩溃堆栈中的类依赖, 评估 GUI 控件与目标崩溃的相关性. 基于此, 设计了 CReDroid 的 3 个变体, 用以分析模块各部分的独立贡献, 即, 1) $CD_{wc}$, 移除了 CTG 预测页面可达分数的功能; 2) $CD_{wt}$, 不使用崩溃报告标题计算语义相似度分数; 3) $CD_{wa}$, 不设置自适应策略对 GUI 控件进行评分.

针对 RQ4, 为验证使用 CReDroid 生成的复现步骤的有效性, 我们设计了与手动复现崩溃的对比实验. 实验邀请了 12 名研究生参加评估, 他们均具有至少 6 个月的安卓测试开发经验. 我们要求这些参与者根据堆栈跟踪或由 CReDroid 生成的复现步骤, 以及崩溃标题信息, 尝试手动复现 CReDroid 成功复现的 57 个崩溃报告. 每个参与者被分配 15 个崩溃报告, 分别基于堆栈跟踪和标题信息, 以及 CReDroid 生成的复现步骤和标题信息尝试手动复现. 并且确保每份崩溃报告均由 3 名参与者复现, 以确保实验结果的可靠性. 所有崩溃报告对应的应用均已安装, 不同参与者复现相同崩溃报告时, 所使用的模拟器设备和系统版本保持一致. 与现有研究设立的规则一致, 实验过程中, 参与者记录理解和复现每份崩溃报告所需要的时间, 并限定每份报告的复现时间为 30 min. 如果超过限定的时间仍未复现成功, 则标记为复现失败. 基于复现崩溃成功率和成功复现所需的平均时间, 我们对两种复现方式的表现进行了比较, 包括基于堆栈跟踪和标题信息复现崩溃, 以及基于 CReDroid 的复现步骤和标题信息复现崩溃.

## 4 实验结果分析

RQ1: CReDroid 在崩溃复现有效性与效率上分别表现如何?

表 2 展示了 CReDroid 与基准方法在复现 3 个数据集崩溃报告方面的成功率, 其中, CD、CT、RD、RB、MK 和 APE 分别为方法 CReDroid、CrashTranslator、ReCDroid、ReproBot、Monkey 和 APE. 总体而言, 在限定的 1 h 测试时间内, CReDroid 成功复现了全部 74 个崩溃报告中的 57 个 (77.03%), 相较于 CrashTranslator、ReCDroid、ReproBot、Monkey 和 APE 分别多复现了 13 个 (17.57%)、25 个 (33.78%)、27 个 (36.49%)、30 个 (40.54%) 和 17 个 (22.97%) 崩溃报告. 这表明, CReDroid 能够有效地利用崩溃报告中的标题和堆栈跟踪进行崩溃复现. 具体来说, CReDroid 在 ReCDroid 数据集上成功复现了 29 个 (87.88%) 崩溃报告, 与 CrashTranslator 表现相当, 而其他基准方法的成功率仅在 45.45%–63.63% 之间. 在 AndroR2 和 CrashTranslator 数据集上, CReDroid 的崩溃复现成功率分别达到 61.9% 和 75%. 相比之下, 除了 APE 在 CrashTranslator 数据集上的表现与 CReDroid 持平外, 其他基准方法在这两个数据集上的复现成功率均不高于 50%, 其中最低甚至仅为 14.29%. 进一步分析表明, ReCDroid 数据集中的崩溃报告涉及的复现探索步骤相对较少, 因此所有方法在该数据集上的复现成功率均较高.

表 2 复现成功率 (有效性)

| #数据集 | CD | CT | RD | RB | MK | APE |
|---|---|---|---|---|---|---|
| ReCDroid数据集 (33) | 29 | 26 | 21 | 19 | 15 | 21 |
| AndroR2数据集 (21) | 13 | 8 | 4 | 3 | 4 | 5 |
| CrashTranslator数据集 (20) | 15 | 10 | 7 | 8 | 8 | 14 |
| 共计 (74) | 57 (77.03%) | 44 (59.46%) | 32 (43.24%) | 30 (40.54%) | 27 (36.49%) | 40 (54.05%) |

对于 CReDroid 未能成功复现的 17 个崩溃报告, 主要原因可总结为以下 3 个方面. 一是测试环境兼容性问题, 部分应用无法在我们的测试环境中正常运行, 例如 AndroR2 数据集中的 cgeo-7369 无法与模拟器兼容. 二是 CReDroid 在交互式 GUI 操作方面存在局限性, 虽然支持点击、长按、输入和旋转等操作, 但某些崩溃触发条件需

要向左或向上滑动, 或结合系统事件, 这超出了当前实现的能力. 三是本文方法存在一些技术上的局限性, CReDroid在某些复杂场景下的探索和分析能力存在不足, 这些将在后续讨论部分中详细说明.

图 6 展示了 CReDroid 与 5 种基准方法在成功复现崩溃报告时所需的时间对比. CReDroid 成功复现 57 个崩溃报告的平均时间为 75.37 s, 表明其从标题和崩溃追踪信息中自动复现崩溃的时间成本具有较高的效率. 从图 6 中可以看出, 无论是平均复现时间还是最大复现时间, CReDroid 均显著低于其他基准方法. 由于不同测试工具在不同崩溃报告上的复现性能存在差异, 我们进一步比较了在所有基准方法均成功复现的崩溃报告上的平均复现时间. 每种方法在 3 个数据集上的复现结果细节如表 3 所示. 上述方法成功复现崩溃, 将在表中记录复现时间 (以 s 为单位), 如果失败则将记录×. 为了节省空间, 省略了任何一种方法都无法复现的崩溃报告. 结果显示, CReDroid 的平均复现时间比最接近的 CrashTranslator 减少了 26.71% (68.05 s vs. 92.86 s). 与其他崩溃复现工具相比, CReDroid 的平均复现时间比 ReCDroid 减少了 94.96% (56.38 s vs. 1 117.81 s), 比 ReproBot 减少了 71.65% (59.33 s vs. 209.3 s). 同时, CReDroid 相较于 GUI 测试工具 Monkey 和 APE 的平均复现时间分别减少了 84.72% (57.8 s vs. 378.24 s) 和 88.56% (69.51 s vs. 607.51 s). 这些结果表明, CReDroid 在崩溃报告复现效率方面具有显著优势, 能够以更低的时间成本完成崩溃复现任务.

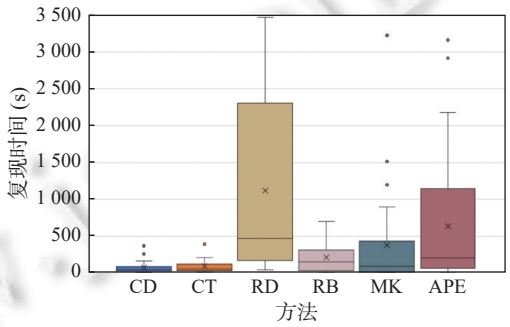



图 6 对成功复现的报告的复现时间 (效率)

表 3 在 3 个数据集上的复现结果细节

| ID | #Crash Reports | RQ1 | | | | | | RQ3 | | | RQ4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CD | CT | RD | RB | MK | APE | $CD_{wc}$ | $CD_{wt}$ | $CD_{wa}$ | $P_{t+t}$ | $P_{s+t}$ |
| R-1 | NewsBlur-1053 | 66 | 46 | 264 | × | 212 | 51 | 59 | 82 | 54 | 128 (3) | 59 (3) |
| R-2 | Markor-194 | 161 | × | × | × | × | 1 868 | 544 | 492 | 674 | 314 (2) | 62 (3) |
| R-3 | Birthdroid-13 | 116 | 126 | 233 | 631 | × | 1 472 | 124 | 117 | 173 | 121 (3) | 58 (3) |
| R-4 | Car Report-43 | 63 | 59 | 715 | 53 | 343 | × | 1 096 | 63 | 881 | 585 (2) | 97 (3) |
| R-5 | acv-11 | 18 | × | 3 324 | 552 | × | 1 693 | 18 | 202 | 25 | 78 (3) | 19 (3) |
| R-6 | AnyMemo-18 | 23 | 159 | 2 134 | 219 | 48 | 115 | 701 | 24 | 374 | 65 (3) | 17 (3) |
| R-7 | AnyMemo-440 | 61 | 192 | × | × | × | 842 | 59 | × | 72 | 83 (3) | 26 (3) |
| R-8 | Notepad-23 | 28 | 83 | 3 472 | 53 | × | 726 | 35 | 28 | 697 | × (0) | 45 (3) |
| R-9 | Olam-2 | 8 | 33 | 119 | × | 1 514 | 2 180 | 13 | 9 | 9 | 21 (3) | 21 (3) |
| R-10 | Olam-1 | 8 | 26 | 64 | 47 | 838 | 119 | 10 | 9 | 9 | 16 (3) | 16 (3) |
| R-11 | FastAdapter-394 | 13 | 12 | 714 | 140 | 9 | 15 | 15 | 13 | 15 | 49 (2) | 10 (3) |
| R-12 | LibreNews-22 | 103 | 133 | 2 276 | 170 | × | × | 103 | 125 | 284 | 152 (1) | 52 (3) |
| R-13 | LibreNews-23 | 147 | 109 | × | 437 | 93 | 35 | × | × | 122 | × (0) | 89 (3) |
| R-14 | LibreNews-27 | 43 | 157 | 1 501 | × | × | × | 36 | 102 | 73 | 35 (3) | 56 (3) |
| R-15 | SMSsync-464 | 23 | 145 | 2 579 | × | 1 196 | 84 | 68 | 48 | 124 | 39 (3) | 37 (3) |
| R-16 | Transistor-63 | 12 | 13 | 168 | 25 | × | 19 | 13 | 12 | 13 | 137 (3) | 39 (3) |
| R-17 | Zom-271 | 21 | 48 | × | 141 | 43 | 130 | 16 | 21 | 22 | 172 (1) | 46 (3) |
| R-18 | Pix-Art-125 | 69 | 61 | 3 134 | 220 | × | 426 | 49 | 58 | 110 | 209 (3) | 51 (2) |
| R-19 | Pix-Art-127 | 67 | 35 | 3 250 | 165 | 3 227 | 1 189 | 61 | 66 | 153 | 353 (1) | 36 (3) |

表 3 在 3 个数据集上的复现结果细节 (续)

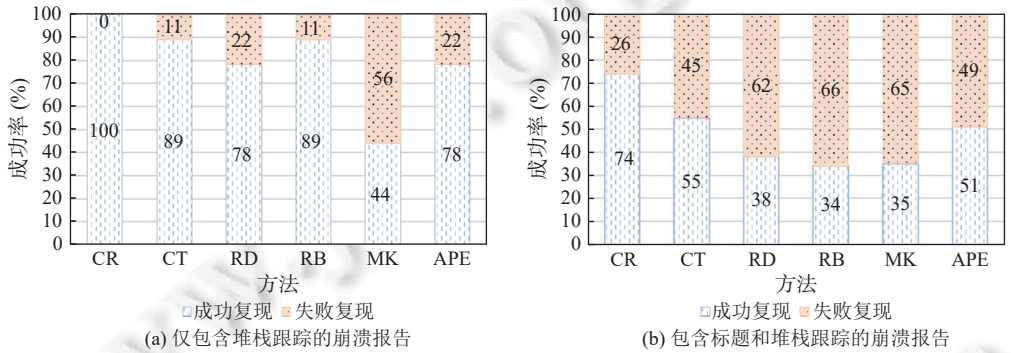
| ID | #Crash Reports | RQ1 | | | | | | RQ3 | | | RQ4 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | CD | CT | RD | RB | MK | APE | $CD_{wc}$ | $CD_{wt}$ | $CD_{wa}$ | $P_{t+t}$ | $P_{s+t}$ |
| R-20 | ScreenCam-25 | 34 | 112 | × | × | 123 | 79 | 65 | 12 | 40 | 38 (3) | 49 (3) |
| R-21 | ownCloud-487 | 368 | 78 | 202 | 290 | 26 | × | 287 | 385 | 299 | 97 (3) | 45 (3) |
| R-22 | OBDReader-22 | 281 | 396 | 2 799 | 295 | 181 | 198 | 232 | 85 | 298 | 171 (3) | 82 (3) |
| R-23 | Dagger-46 | 28 | 10 | 79 | 4 | 2 | 4 | 28 | 28 | 40 | 18 (3) | 18 (3) |
| R-24 | ODK-2086 | 259 | × | × | × | × | 382 | 213 | 109 | 450 | 72 (3) | 26 (3) |
| R-25 | K-9Mail-3255 | 24 | 24 | × | × | × | × | 24 | 24 | 191 | 86 (3) | 48 (3) |
| R-26 | K-9Mail-2612 | × | 119 | × | × | × | × | × | × | × | — | — |
| R-27 | K-9Mail-2019 | 20 | 22 | × | 26 | × | × | 20 | 20 | 21 | 60 (3) | 65 (3) |
| R-28 | Anki-4586 | 28 | × | 295 | × | × | × | 21 | × | 28 | 165 (3) | 42 (3) |
| R-29 | TagMo-12 | 12 | 24 | 56 | 13 | 12 | 12 | 12 | 12 | 13 | 27 (3) | 27 (3) |
| R-30 | FlashCards-13 | 29 | 97 | 318 | 33 | × | × | 32 | 54 | 34 | 41 (3) | 30 (3) |
| C-1 | NewPipe-7825 | 46 | 31 | 693 | 358 | × | × | 71 | 92 | 128 | 123 (2) | 68 (3) |
| C-2 | SDBViewer-10 | 11 | 25 | 51 | 11 | 38 | 21 | 11 | 11 | 14 | 25 (3) | 25 (3) |
| C-3 | Anki-9914 | 26 | × | × | 528 | × | 3 164 | 26 | 256 | 413 | 92 (2) | 39 (3) |
| C-4 | Anki-10584 | 43 | 56 | × | 65 | × | 171 | 43 | 46 | 77 | 213 (3) | 45 (3) |
| C-5 | Alarmio-47 | × | × | × | × | 506 | × | × | × | × | — | — |
| C-6 | GrowTracker-89 | × | × | × | × | 234 | 1 704 | × | × | × | — | — |
| C-7 | Anki-3370 | 68 | 60 | × | × | × | 125 | 355 | 679 | 595 | 159 (2) | 40 (3) |
| C-8 | WhereYouGo-368 | 376 | 207 | × | × | × | 386 | 64 | 245 | 765 | × (0) | 182 (3) |
| C-9 | privacy-friendly-food-tracker-55 | 46 | 23 | 216 | × | 113 | 322 | 38 | 43 | 84 | 105 (1) | 20 (3) |
| C-10 | GrowTracker-87 | 35 | 185 | × | × | × | 258 | 30 | 66 | × | 138 (3) | 47 (3) |
| C-11 | NordicSemiconductor-Android-nRF-Mesh-Library-495 | 39 | × | 2 315 | 702 | 895 | 203 | 24 | 28 | 841 | 119 (3) | 41 (3) |
| C-12 | sdbviewer-7 | 7 | × | 178 | 21 | 26 | 110 | 7 | 37 | 18 | 17 (3) | 17 (3) |
| C-13 | FakeStandby-30 | 28 | 43 | 658 | × | × | 1 497 | 30 | 25 | 46 | × (0) | 36 (3) |
| C-14 | privacy-friendly-pedometer-101 | 50 | × | × | 499 | × | 1 023 | 39 | 238 | 51 | 56 (3) | 37 (3) |
| C-15 | revolution-irc-183 | 20 | 47 | × | × | 435 | 1 182 | 443 | 63 | 28 | 81 (3) | 40 (3) |
| C-16 | Anki-3224 | 36 | × | × | × | × | × | 152 | 162 | 275 | 46 (3) | 30 (3) |
| C-17 | getodk-skunkworks-crow-219 | 11 | 20 | 129 | 30 | 11 | 61 | 10 | 16 | 13 | 25 (3) | 25 (3) |
| A-1 | HABPanel-25 | 26 | 35 | × | 218 | 14 | 46 | 26 | 26 | 26 | 15 (3) | 15 (3) |
| A-2 | NoadPlayer-1 | 36 | 25 | 39 | 38 | 12 | 6 | 37 | 37 | 36 | 17 (3) | 17 (3) |
| A-3 | citiususc-134 | 102 | × | × | × | × | 433 | 85 | × | 164 | 58 (3) | 63 (3) |
| A-4 | Weather-61 | 43 | 45 | 363 | × | × | × | 46 | 42 | 50 | 55 (3) | 36 (3) |
| A-5 | Berkeley-82 | 59 | 37 | 571 | 295 | 43 | 128 | 45 | 78 | 63 | 43 (3) | 28 (3) |
| A-6 | Osmdroid-1030 | 294 | × | × | × | × | × | 235 | × | 381 | × (0) | 52 (3) |
| A-7 | andOTP-500 | 110 | 401 | × | × | × | × | 278 | 214 | 332 | 90 (3) | 94 (3) |
| A-8 | K-9Mail-3971 | 129 | 391 | × | × | × | × | 108 | 158 | 275 | 158 (3) | 119 (3) |
| A-9 | privacy-friendly-pedometer-28 | 28 | × | × | × | 2 | × | 28 | 28 | 28 | 15 (3) | 15 (3) |
| A-10 | Transistor-149 | 51 | × | × | × | × | × | 80 | × | 169 | 104 (2) | 76 (2) |
| A-11 | gnucash-android-633 | 273 | × | × | × | × | 2 918 | 272 | 529 | × | 112 (3) | 68 (3) |
| A-12 | Firefox-3932 | 72 | 49 | 2861 | × | × | × | 504 | 57 | 119 | × (0) | 39 (3) |
| A-13 | Aegis-500 | 98 | 113 | × | × | × | × | 97 | 76 | 118 | 216 (3) | 128 (3) |

针对 RQ1 的结论: CReDroid 在崩溃复现的有效性和效率上表现优异. 相比于崩溃复现工具 CrashTranslator、ReCDroid、ReproBot 和 GUI 测试工具 Monkey、APE, CReDroid 分别多复现 13、25、27、30 和 17 个崩溃报告, 且在成功复现相同崩溃时, 平均复现时间分别减少 26.71%、94.96%、71.65%、84.72% 和 88.56%.

RQ2: 这些最先进的技术在复现崩溃时是否互补?

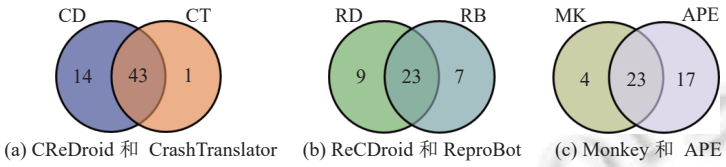
图 7 展示了 6 种最先进的技术在仅包含堆栈跟踪和同时包含标题和堆栈跟踪这两类崩溃报告上的复现情况.

在仅包含堆栈跟踪的 9 个崩溃报告上, CReDroid、CrashTranslator、ReCDroid、ReproBot、Monkey 和 APE 分别成功复现了 9、8、7、8、4 和 7 个崩溃报告, 对应复现成功率为 100%、89%、78%、89%、44% 和 78%. 在其余 65 个同时包含标题和堆栈跟踪的崩溃报告上, 这些工具分别成功复现了 48、36、25、22、23 和 33 个崩溃报告, 复现成功率分别为 74%、55%、38%、34%、35% 和 51%. 结果表明, CReDroid 和 CrashTranslator 在两类崩溃报告上的复现能力表现较强. 其中, 如图 7(b) 所示, 在同时包含标题和堆栈跟踪的崩溃报告上, CReDroid 通过额外考虑标题信息, 相较于仅依赖堆栈跟踪的 CrashTranslator, 复现成功率更高. 图 7(a) 显示, 在仅包含堆栈跟踪信息的报告上, 尽管 ReCDroid 和 ReproBot 主要依赖复现步骤中的具体操作信息, 但由于堆栈跟踪中包含的崩溃 API 可能涉及 GUI 控件信息, 它们仍能够一定程度上复现这些崩溃. 然而, 由于标题缺少完整的操作步骤, 它们的复现能力仍然受限, 成功率相对较低, 如图 7(b) 所示. 此外, 在没有崩溃报告信息作为指导的情况下, Monkey 复现成功率最低, 这表明随机测试策略难以有效触发特定崩溃; 而 APE 通过构建 GUI 模型, 能够更系统地覆盖应用状态, 提高测试覆盖率. 为了进一步分析不同工具的互补性, 我们比较了 6 种最先进的技术在这些崩溃报告上的具体表现, 并在图 8 中展示了相同类别工具复现崩溃的共同和独特部分. 结合表 3 的复现结果, 我们发现各工具在崩溃复现方面能补充其他工具, 并适用于不同的应用场景.




(a) 仅包含堆栈跟踪的崩溃报告 (b) 包含标题和堆栈跟踪的崩溃报告

图 7 在两类崩溃报告上的复现结果




(a) CReDroid 和 CrashTranslator (b) ReCDroid 和 ReproBot (c) Monkey 和 APE

图 8 同类工具成功复现崩溃的比较

● CReDroid 和 CrashTranslator. CReDroid 和 CrashTranslator 均能处理堆栈跟踪信息, 并推导出到达目标崩溃页面的路径, 两者复现了 4 个其他所有工具均无法复现的崩溃. 这些崩溃报告的标题中虽包含具体的操作信息, 但步骤不完整. CReDroid 额外结合标题中操作的指引, 提高了崩溃复现成功率, 相比 CrashTranslator 额外复现了 14 个独特的崩溃报告, 其中 13 个属于同时包含标题和堆栈跟踪的报告. CrashTranslator 在处理堆栈跟踪中带有明显活动信息的报告时表现较好, 尤其是用户经常操作的功能场景. 例如, 对于 CReDroid 失败崩溃复现的 ID 为 R-26 的 K-9Mail-2612 崩溃报告, 触发崩溃需要用户在删除邮件前先选中目标邮件, CrashTranslator 依赖 LLM 预测操作, 能正确推荐选中邮件这一关键操作, 从而成功复现崩溃. 然而, CrashTranslator 难以处理堆栈跟踪中应用级别崩溃信息不足的报告, 例如在 ID 为 R-28 的 Anki-4586 崩溃报告中, 由于缺乏显示的目标崩溃活动信息, CrashTranslator 无法有效推导触发崩溃的操作序列.

● ReCDroid 和 ReproBot. ReCDroid 和 ReproBot 能够处理崩溃报告的文本复现步骤、标题以及堆栈跟踪应用级崩溃语句. ReCDroid 采用预定义的语法模式提取具体操作, 适用性较强. 例如, 对于 ID 为 R-28 的 Anki-4586 崩

溃报告, 即使不借用 LLM, ReCDroid 也能准确提取标题中的具体操作使得崩溃复现成功. 然而, 当报告包含很少或缺失步骤时, 该方法的探索算法需要很长时间才能匹配到正确的 UI 事件. 例如, 在 ID 为 C-11 的 NordicSemiconductor-Android-nRF-Mesh-Library-495 崩溃报告中, 标题仅缺乏一个操作步骤, 而由于 UI 页面数量较多, ReCDroid 耗时近 40 min 来成功复现崩溃. ReproBot 在缺少步骤的报告中表现更优, 能够重现更多崩溃. 例如, 在 ID 为 C-4 的 Anki-10584 崩溃报告中, 即使 ReproBot 无法从标题中提取有效的指导信息, 但其结合强化学习来优化 UI 事件搜索, 使得在匹配 UI 事件时更具优势.

● Monkey 和 APE. 这两种工具不依赖于崩溃报告中提供的信息, 能够触发某些特定类型的崩溃. 例如, Monkey 和 APE 均能复现其他工具不能触发的 ID 为 C-6 的 GrowTracker-89 崩溃报告, 触发崩溃需在 settings 页面进行滑动选择目标控件. Monkey 将几乎所有的测试预算用于事件生成, 在短路径崩溃的触发上具有优势. 在其复现的 4 个 APE 无法复现的独特崩溃中, 有 3 个的复现步骤均不超过 5. 相比之下, APE 通过动态 GUI 模型对应用进行更系统的探索, 能够局部穷尽性地执行应用的某些部分, 并发现比其他工具更多的活动. 例如, 在 ID 为 A-3 的 citiususc-134 崩溃报告中, 触发崩溃需要深入创建药物页面, 拨动 Stock 控件并与新出现的控件进行交互. APE 通过更深层次地遍历 GUI 层级成功复现了该崩溃, 而大多数其他工具未能完成该操作.

针对 RQ2 的结论: 这些最先进的技术在崩溃复现方面具有互补性, 且适用于不同的场景. CReDroid 在结合堆栈跟踪和标题信息的情况下复现能力最强; CrashTranslator 在堆栈信息明确的情况下表现优越; 在利用文本复现步骤时, ReCDroid 在报告标题明确包含操作信息时复现效果较好, ReproBot 在处理缺少部分步骤的报告时具备更强的探索能力; Monkey 和 APE 则能够触发其他工具难以复现的特定类型崩溃.

RQ3: 自适应控件评分模块是否能够提高 CReDroid 崩溃复现的效果?

我们通过崩溃复现的成功率和复现时间来评估自适应控件评分模块的有效性, 具体包括融合 CTG 的页面可达分数、基于崩溃报告标题关键操作的语义相似度分数, 以及自适应评分策略对复现性能的影响. 表 3 中列出的 $CD_{wc}$、$CD_{wt}$ 和 $CD_{wa}$ 分别展示了 CReDroid 在未使用 CTG 构造、未考虑报告标题关键操作以及未设置自适应评分策略时的崩溃复现结果.

表 3 中显示, $CD_{wc}$ 成功复现了 56 个崩溃报告, 比 CReDroid 少复现 1 个, 具体为 ID 为 R-13 的 LibreNews-23 崩溃报告未能成功复现. 分析发现, 崩溃堆栈中涉及的关键 API 为 app.librenews.io.librenews.views.Settings ActivityFragment.onSharedPreferenceChanged, 其关联的崩溃组件为 app.librenews.io.librenews.views.Settings ActivityFragment, 这是一个 Fragment 粒度的组件. 然而, Librenews 应用的 APK 文件中 AndroidManifest.xml 主要声明 Activity 粒度的信息, 单纯依赖 XML 分析无法准确定位崩溃组件. 在未使用 CTG 构造的情况下, CReDroid 无法有效识别崩溃发生的具体组件, 缺乏组件间转换路径的指导, 动态探索难以准确导航至崩溃组件, 导致崩溃复现失败. 此外, $CD_{wc}$ 在复现相同崩溃报告时的平均复现时间比 CReDroid 慢 72.03% (127.46 s vs. 74.09 s). 我们发现, 尽管对于部分触发崩溃的探索序列较短的报告, 仅依赖语义相似度就能找到目标 GUI 控件, CTG 对复现时间影响较小. 然而, 考虑 CTG 的 CReDroid 能够通过当前页面到崩溃组件的可达路径, 帮助快速导航至崩溃发生的页面, 从而提升探索效率. 例如, 对于 ID 为 R-6 的 AnyMemo-18 崩溃报告, 尽管从应用主页仅需两个测试步骤即可触发崩溃, 但页面中存在大量候选 GUI 控件, 给动态探索带来难度. 在这种情况下, CTG 提供的路径信息能够逐步指导控件选择, 避免探索多余的路径, 从而节约测试资源. 由此可见, CTG 的构造对于准确定位崩溃组件并提高复现成功率至关重要.

当未考虑崩溃报告标题信息时, $CD_{wt}$ 成功复现了 51 个崩溃报告, 比 CReDroid 少复现 6 个. 崩溃报告的标题通常包含触发崩溃的关键操作信息. 例如, ID 为 A-3 的 citiususc-134 崩溃报告的标题为“Can’t open the add/remove medicine stock dialog”, 尽管定位到崩溃组件 MedicineCreateOrEditFragment 提供了指导信息, 但该 Fragment 页面包含多个创建药物的 GUI 控件. 复现崩溃需要拨动“Stock”开关控件以触发添加或移除药物库存的操作, 最终导致崩溃. 缺乏标题信息的指导时, 动态探索难以聚焦于特定操作, 导致复现失败. 在复现相同崩溃报告时, $CD_{wt}$ 的平均复现时间比 CReDroid 慢 50.9% (106.90 s vs. 70.84 s). 这是因为缺乏标题信息的辅助, 虽然可以通过遍历页面触

发目标 GUI 控件, 但无法利用标题中隐含的操作信息到达崩溃状态, 从而导致探索效率下降.

在为当前页面的 GUI 控件评分时, 如果不采用自适应策略, $CD_{wa}$ 成功复现了 55 个崩溃报告, 比 CReDroid 少复现 2 个. 标题通常包含最接近崩溃页面的操作信息, 因此自适应策略根据当前页面所在组件与崩溃页面的关系, 动态调整评分重点, 从启动应用时侧重 GUI 组件逐步过渡到关注标题信息. 在不考虑自适应策略的情况下, 当应用启动页面包含大量可操作 GUI 控件时, 评分机制的效果可能受到限制. 例如, 针对 ID 为 A-12 的 gnucash-android-633 崩溃报告, 由于标题未总结崩溃页面触发崩溃的具体操作, 同时从应用主页到崩溃组件的路径较多, 且不同组件间的转换较为灵活, 动态探索可能被引导至错误方向, 无法快速导航到崩溃组件并触发目标控件, 导致无法在有限的时间内成功复现崩溃. 另一种情况是, 当无法明确检索到从主页到达崩溃组件的路径, 同时标题中的关键操作在多个页面中均可匹配时, 复现效率会降低. 例如, ID 为 C-3 的 Anki-9914 崩溃报告, 其标题“v2.16alpha33 crashes when changing Decks in Card Browser and Statistics”中的关键词“decks”与多个页面的 GUI 控件相关联, 评分机制可能因此受到误导, 导致额外的探索时间. 统计结果表明, 在复现相同崩溃报告时, $CD_{wa}$ 的平均复现时间比 CReDroid 慢 160.96% (189.22 s vs. 72.51 s), 显著降低了复现效率.

针对 RQ3 的结论: 自适应控件评分模块的设计显著提升了 CReDroid 的崩溃复现性能, 不仅提高了复现成功率, 还显著优化了复现时间. CTG 的构造帮助准确定位崩溃组件, 其提供的路径信息逐步指导控件选择, 避免探索多余的路径; 崩溃报告标题信息中隐含的操作信息帮助动态探索聚焦于特定操作, 快速到达崩溃状态; GUI 控件自适应评分策略根据当前页面所在组件与崩溃页面的关系, 动态调整评分重点, 从启动应用时侧重 GUI 组件, 逐步过渡到关注标题信息, 显著提升复现效率.

RQ4: CReDroid 生成的复现步骤是否能够正确触发崩溃?

表 3 中的 $P_{t+t}$ 和 $P_{s+t}$ 两列分别展示了参与者基于堆栈跟踪和 CReDroid 生成的复现步骤, 以及参考崩溃标题进行手动复现崩溃的结果. 该结果被记录为平均时间 (成功复现崩溃的参与者数量), 其中 CReDroid 不能成功复现的崩溃报告未在实验评估范围内, 记录为“—”. 从表 3 可以看出, 基于 CReDroid 生成的复现步骤以及崩溃标题的指引, 100% 的崩溃报告能够至少被两名参与者成功复现. 相较之下, 基于堆栈跟踪和崩溃标题进行崩溃复现时, 在所有 57 个崩溃报告中, 有 10 个未能被至少两名参与者成功复现, 成功复现比例下降了 17.54%. 此外, 即使有崩溃标题提供的操作信息作为参考, 仍有 6 个崩溃报告未能被任何参与者成功复现. 在成功复现时间方面, 基于 CReDroid 生成的复现步骤复现相同崩溃的效率显著更高, 平均时间比堆栈跟踪复现减少了 58.13% (106.15 s vs. 44.45 s). 这表明, CReDroid 生成的复现步骤可以有效辅助崩溃标题进行探索指导, 能够显著提升复现效率. 在成功复现的相同崩溃中, 有 70.59% (36/51) 的案例表明, CReDroid 的复现步骤比基于堆栈跟踪的复现方式更有效, 进一步验证了 CReDroid 能够比手动复现更快地定位崩溃, 提升开发者的效率.

在实验结束后, 我们对参与者进行了非结构访谈, 了解他们在两种复现方式中遇到的挑战. 大多数参与者表示, 在崩溃复现过程中, 报告标题相较于崩溃堆栈更易于理解, 因此他们主要参考报告标题来指导崩溃复现. 然而, 当崩溃报告缺乏关键操作信息时, 由于堆栈跟踪中与崩溃相关的信息有限, 难以推断出具体的复现步骤, 从而无法成功复现崩溃. 即使报告标题中包含了操作信息, 若触发崩溃需要特定额外的事件序列, 也可能导致复现失败. 例如, 在 ID 为 C-8 的 WhereYouGo-368 崩溃报告中, 参与者根据标题信息找到对应的 GUI 控件并触发后, 应用仍正常运行, 重复操作也无法触发崩溃. 通过进一步查看崩溃报告的文本复现步骤发现, 在触发相应 GUI 控件后, 需要关闭并重启应用才能触发崩溃. 这种复杂的操作序列使得参与者在多次尝试后失去兴趣, 认为崩溃不可复现.

尽管在部分案例中, CReDroid 复现崩溃所需时间略长于手动方式, 但考虑到其通过动态探索自动完成额外的操作以发现崩溃触发路径, 避免了繁琐的人工干预, 整体表现优于手动复现.

针对 RQ4 的结论: CReDroid 生成的复现步骤能够有效触发崩溃, 并显著提升复现效率. 在参考崩溃标题的情况下, 与基于堆栈跟踪的复现方式相比, 基于 CReDroid 生成的复现步骤展现了更高的成功率, 且复现时间减少了 58.13%. 尽管在部分案例中复现时间略长, CReDroid 通过动态探索能够自动完成额外操作, 整体上优于手动复现, 提升了开发者的效率.

## 5 讨 论

### 5.1 局限性

除了实验结果中讨论的两个限制外, CReDroid 在有效复现所有崩溃报告时还面临以下 3 项技术局限, 这进一步揭示了基于标题和堆栈跟踪进行崩溃复现所需要解决的技术挑战.

首先, CReDroid 难以复现依赖复杂用户操作的崩溃. 一方面, 某些崩溃需要特定的多步操作和动态数据状态, 这显著增加了测试复现的难度. 例如, 在卡片浏览器的多选模式下更改卡片类型时触发的崩溃, 由于涉及复杂的 GUI 控件交互且操作顺序稍有偏差便可能无法触发崩溃, 导致复现失败. 另一方面, CReDroid 专注于应用内的 GUI 操作, 对于系统级操作支持不足. 例如, 下拉通知栏并点击系统级控件“删除浏览历史”的场景需要频繁切换应用上下文 (从前台到后台再回到通知栏), 此类操作流程的复杂性超出了当前 CReDroid 的覆盖范围, 难以实现自动化测试.

其次, CReDroid 在需要有效输入内容才能触发的崩溃场景中难以完全自动化. 例如, 在登录场景中需要提供正确的用户名和密码, 而这些输入内容通常涉及用户隐私, 未包含在堆栈跟踪或崩溃报告中, 因而必须依赖人工干预. 然而, 通过预先请求用户输入相关信息, CReDroid 仍然能够实现对这些场景的自动化复现.

最后, CReDroid 的语义匹配方法在某些语境下的准确性仍有待提高, 这可能成为复现崩溃的障碍. 目前的语义匹配方法在处理词义扩展和上下文关联时存在不足, 尤其在短文本或词义接近的场景下容易产生匹配偏差. 例如, 在被测应用 Shuttle Player 创建新播放列表时的崩溃复现中, 预训练 BERT 模型未能正确识别“play”和“playlist”之间的语义关系, 导致关键操作未被正确匹配, 从而复现失败. 为了提高 CReDroid 在复杂语境中的语义匹配能力, 未来的工作将尝试引入 LLM 以增强其对上下文的理解和处理能力.

### 5.2 有效性威胁

● 内部有效性威胁. 第 1 个内部有效性威胁来自 CReDroid 方法的实现. 为减少此类威胁, 我们在实现 CReDroid 时, 采用 Python 中现有成熟的第三方库以确保实现的可靠性. 在报告分析阶段, 选择了 ChatGPT 大规模语言模型从崩溃标题中提取关键操作信息. 由于 ChatGPT 具备强大的语言理解能力, 能够应对复杂的分析任务, 同时通过构建数据集对其进行微调, 使得其能学习崩溃报告标题的多样化描述方式. 此外, 为确保 CReDroid 能够准确定位崩溃组件, 本文采用广泛使用的 ICCBot 工具构造被测应用的 CTG, 其正确性已在大量实验中得到验证. 第 2 个内部有效性威胁涉及基准方法的执行. 我们从方法的作者提供的开源链接获取源代码, 并严格按照其指引步骤在我们的数据集上进行实验. 对于 CReDroid 在选择 GUI 控件被设计为以一定的随机性选择一个非最优的控件, 我们通过运行 3 次实验并记录平均值的方式, 减少随机性对结果的影响.

● 外部有效性威胁. 外部有效性威胁主要来源于评估中所选用数据的代表性. 本文严格遵循崩溃复现相关研究的程序, 选择了广泛使用的 ReCDroid、 AndroR2 和 CrashTranslator 数据集. 这些数据集中的崩溃报告均来自真实的应用问题跟踪系统, 能够较好地反映实际崩溃场景, 从而减少数据选择带来的偏差. 另一个外部有效性威胁来自 CReDroid 方法的通用性. 为提高 CReDroid 的通用性, CReDroid 使用 LLM 从崩溃报告中提取关键操作信息, 并通过微调使其能够适应不同描述方式的报告分析, 增强在实际应用中的适用性. 第 3 个威胁涉及参与者的相应偏差. 根据现有研究方法[9,14], 我们假设具有 Android 编程经验的学生能够代替测试人员. 参与者在实验中的成功率和复现时间被视为具有代表性的结果, 尽可能减少参与者背景对实验结论的影响.

## 6 相关工作

当前已有广泛的研究聚焦于安卓应用崩溃复现, 旨在提升软件维护的效率. 用户通常通过文本描述、截图、录制视频等形式的复现步骤、堆栈跟踪等信息来提交崩溃报告, 这些信息对开发者快速、准确地定位问题并加以解决至关重要. 崩溃报告的质量对崩溃复现具有决定性影响. 由于实际提交的崩溃报告内容往往不完整, 从有限信息中推断并自动复现崩溃是一个具有挑战性的研究问题.

一些研究[8−13]致力于通过崩溃报告中逐步指导的复现步骤来复现崩溃, 主要聚焦于基于文本描述和视觉记录的复现步骤分析. 文本描述的复现步骤通常记录着崩溃发生涉及的控件名称和事件类型. 这类研究通过从复现步骤中提取有用信息, 并匹配相应的 UI 操作, 探索触发崩溃的事件序列. 例如, ReCDroid[9]和 ReCDroid+[10]基于预定义的语法模式从复现步骤中提取操作控件及输入值, 并利用贪婪策略的动态探索确定触发崩溃的事件序列. ScopeDroid[11]则通过构建应用的状态转换图, 设计多模态神经匹配网络来推导候选事件与复现步骤之间的模糊匹配矩阵, 并基于此进行路径规划以指导探索. 然而, 复现步骤可能通过连接多个句子的长句形式来描述崩溃. 为此, ReproBot[8]利用时间标准规范化句子中的操作执行顺序来细化复现步骤, 从而能处理更广泛的崩溃报告. 同时, ReproBot 结合动态探索中的 Q-learning 技术, 通过触发 GUI 控件查找复现崩溃所需的缺失步骤. 然而, 当崩溃报告缺失部分描述步骤时, 这些方法在实际应用页面中往往难以找到正确的匹配. 针对这一问题, ROAM[12]通过对应用进行 UI 建模, 并基于动态规划技术在模型中执行全局搜索, 识别最优路径以找到最可能的匹配. 另一类崩溃复现方法专注于视觉记录的分析, 如屏幕截图或录制的视频. GIFDroid[13]试图复现视频中记录的崩溃, 其通过图像处理技术将视频的关键帧映射到 GUI 状态, 并基于 UI 转换图进行静态搜索以补全视频中可能缺失的步骤, 生成完整的复现轨迹.

另一类工作聚焦于分析堆栈跟踪中记录的应用异常信息, 以帮助开发者定位问题和修复崩溃. 一些研究, 例如, 通过定位错误函数生成测试用例[42−46], 或计算堆栈跟踪之间的相似性以识别重复的崩溃报告[47−50]. 近年来, 一些工作尝试直接从堆栈跟踪中进行崩溃复现. CrashTranslator[14]通过利用预训练 LLM 来预测触发崩溃的探索步骤. 然而, 堆栈跟踪中记录的类信息可能无法直接指向崩溃发生的具体页面, 缺乏明确的指导信息使得探索过程更加困难. Mole[15]通过对被测应用进行静态分析, 跟踪 GUI 控件的属性状态, 并在通往崩溃点的可达路径中关注相关 GUI 控件. 该方法依赖调用图定位崩溃点, 可能会出现偏差. CrPDroid[51]则结合堆栈跟踪与复现步骤, 利用应用项目文件构建控件层次图, 分析缺陷报告与项目文件以定位可能触发崩溃的可疑控件, 通过适应度函数来指导探索过程, 以提升复现效率. 另外, CrashDroid[52]基于应用启动到崩溃过程中的所有堆栈跟踪方法调用生成复现步骤.

目前针对安卓应用崩溃复现的研究大多依赖崩溃报告中的复现步骤和堆栈信息等进行分析. 那些基于堆栈信息展开分析的工作对标题或摘要等高层次文本信息的利用尚显不足. 事实上, 标题中往往包含对崩溃问题的高度概括, 这些语义信息能够帮助更精准地推断崩溃的上下文及潜在原因. 此外, 现有方法在动态探索过程中未充分考虑组件转换关系, 导致探索路径缺乏明确导向性. CReDroid 融合了基于复现步骤与基于堆栈跟踪的两类主流崩溃复现方法的策略. 一方面, 该方法结合堆栈跟踪中的动态异常信息与静态分析组件转换信息来定位崩溃组件; 另一方面, 借助崩溃报告标题中的关键操作信息与当前页面到崩溃组件的可达路径, 引导最有可能触发崩溃的 UI 操作, 从而提升崩溃复现的准确性.

## 7 总结与展望

本文提出了一种组件感知的安卓应用崩溃自动复现方法 CReDroid, 能够结合崩溃报告的标题信息和堆栈跟踪有效地复现崩溃. 该方法通过动态构建被测应用的 CTG, 结合组件转换的意图发送信息和堆栈跟踪的动态异常信息定位目标崩溃组件. 基于当前应用页面与崩溃组件的上下文关系, CReDroid 设计自适应评分策略, 根据到崩溃组件的可达路径和报告标题的关键信息为 GUI 控件分配优先级分数. 另外, CReDroid 通过强化学习技术全局优化动态探索过程, 减少预测过程中的不准确性. 实验结果表明, CReDroid 在复现崩溃的有效性和效率上均优于最先进的崩溃复现工具 CrashTranslator、ReCDroid、ReproBot 以及广泛使用的自动化测试工具 Monkey 和 APE.

目前, CReDroid 仅依赖 LLM 来提取崩溃报告标题的关键操作, 未来的研究将进一步将该技术融入崩溃复现过程. 另外, 我们计划使 CReDroid 支持滑动操作以及与系统操作的交互, 以增强其在更复杂崩溃场景中的适应能力.

**References**

[1] Ceci L. Number of APPs available in leading APP stores. 2024. https://www.statista.com/statistics/276623/number-of-apps-available-in-

leading-app-stores/
[2] Google Play Store. https://play.google.com/store/apps
[3] Ceci L. Number of iOS and Google Play APP downloads as of Q1 2024. 2024. https://www.statista.com/statistics/695094/quarterly-number-of-mobile-app-downloads-store/
[4] Sherif A. Market share of mobile operating systems worldwide 2009–2025, by quarter. 2025. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
[5] Li C, Jiang YY, Xu C. GUI event-based record and replay technologies for Android APPs: A survey. Ruan Jian Xue Bao/Journal of Software, 2022, 33(5): 1612–1634 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6551.htm [doi: 10.13328/j.cnki.jos.006551]
[6] GitHub. https://github.com
[7] Bugzilla. https://bugzilla.mozilla.org/home
[8] Zhang ZX, Winn R, Zhao Y, Yu TT, Halfond WG. Automatically reproducing Android bug reports using natural language processing and reinforcement learning. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 411–422. [doi: 10.1145/3597926.3598066]
[9] Zhao Y, Yu TT, Su T, Liu Y, Zheng W, Zhang JZ, Halfond WGJ. ReCDroid: Automatically reproducing Android application crashes from bug reports. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 128–139. [doi: 10.1109/ICSE.2019.00030]
[10] Zhao Y, Su T, Liu Y, Zheng W, Wu XX, Kavuluru R, Halfond WGJ, Yu TT. ReCDroid+: Automated end-to-end crash reproduction from bug reports for Android APPs. ACM Trans. on Software Engineering and Methodology, 2022, 31(3): 36. [doi: 10.1145/3488244]
[11] Huang YC, Wang JJ, Liu Z, Wang S, Chen CY, Li MY, Wang Q. Context-aware bug reproduction for mobile APPs. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 2336–2348. [doi: 10.1109/ICSE48619.2023.00196]
[12] Zhang ZX, Tawsif FM, Ryu K, Yu TT, Halfond WGJ. Mobile bug report reproduction via global search on the APP UI model. Proc. of the ACM on Software Engineering, 2024, 1: 2656–2676. [doi: 10.1145/3660824]
[13] Feng SD, Chen CY. GIFdroid: Automated replay of visual bug reports for Android APPs. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1045–1057. [doi: 10.1145/3510003.3510048]
[14] Huang YC, Wang JJ, Liu Z, Wang YW, Wang S, Chen CY, Hu YZ, Wang Q. CrashTranslator: Automatically reproducing mobile application crashes directly from stack trace. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 18. [doi: 10.1145/3597503.3623298]
[15] Masoudian M, Huang HQ, Amini M, Zhang C. Mole: Efficient crash reproduction in Android applications with enforcing necessary UI events. IEEE Trans. on Software Engineering, 2024, 50(8): 2200–2218. [doi: 10.1109/TSE.2024.3428543]
[16] AnkiDroid. 2024. https://github.com/ankidroid/Anki-Android/issues/4586
[17] Monkey. 2024. http://developer.android.com/tools/help/monkey.html
[18] Gu TX, Sun CN, Ma XX, Cao C, Xu C, Yao Y, Zhang QR, Lu J, Su ZD. Practical GUI testing of Android applications via model abstraction and refinement. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 269–280. [doi: 10.1109/ICSE.2019.00042]
[19] Zhang CN, Zhang CS, Zheng S, Qiao Y, Li CH, Zhang MC, Dam SK, Thwal CM, Tun YL, Huy LL, Kim D, Bae SH, Lee LH, Yang Y, Shen HT, Kweon IS, Hong CS. A complete survey on generative AI (AIGC): Is ChatGPT from GPT-4 to GPT-5 all you need? arXiv:2303.11717, 2023.
[20] Chen X, Zhang NY, Xie X, Deng SM, Yao YZ, Tan CQ, Huang F, Si L, Chen HJ. KnowPrompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In: Proc. of the 2022 ACM Web Conf. Lyon: ACM, 2022. 2778–2788. [doi: 10.1145/3485447.3511998]
[21] Gu YX, Han X, Liu ZY, Huang ML. PPT: Pre-trained prompt tuning for few-shot learning. arXiv:2109.04332, 2022.
[22] Yan JW, Zhang SX, Liu YP, Yan J, Zhang J. ICCBot: Fragment-aware and context-sensitive ICC resolution for Android applications. In: Proc. of the 44th ACM/IEEE Int'l Conf. on Software Engineering: Companion Proc. Pittsburgh: ACM, 2022. 105–109. [doi: 10.1145/3510454.3516864]
[23] Lin GY, Cui ZQ, Chen X, Zheng LW. State transition graph guided testing approach for detecting ARP bugs. Ruan Jian Xue Bao/Journal of Software, 2025, 36(2): 469–487 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7142.htm [doi: 10.13328/j.cnki.jos.007142]
[24] Liu SQ, Zhou Y, Han TT, Chen TL. Test reuse based on adaptive semantic matching across Android mobile applications. In: Proc. of the 22nd IEEE Int'l Conf. on Software Quality, Reliability and Security. Guangzhou: IEEE, 2022. 703–709. [doi: 10.1109/QRS57517.2022.

00076]

[25] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. arXiv:1810.04805, 2019.

[26] Watkins CJCH, Dayan P. Q-learning. Machine Learning, 1992, 8(3): 279–292. [doi: 10.1007/BF00992698]

[27] Sutton RS, Barto AG. Reinforcement Learning: An Introduction. Cambridge: MIT Press, 1998. 26–122.

[28] Pan MX, Huang A, Wang GX, Zhang T, Li XD. Reinforcement learning based curiosity-driven testing of Android applications. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 153–164. [doi: 10.1145/3395363.3397354]

[29] Hao R, Feng Y, Li YY, Chen ZY. Multi-granularity fusion Android test sequence reduction based on event labeling. Ruan Jian Xue Bao/Journal of Software, 2025, 36(5): 2006–2025 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7203.htm [doi: 10.13328/j.cnki.jos.007203]

[30] Liu SQ, Zhou Y, Ji LB, Han TT, Chen TL. Enhancing test reuse with GUI events deduplication and adaptive semantic matching. Science of Computer Programming, 2024, 232: 103052. [doi: 10.1016/j.scico.2023.103052]

[31] Wendland T, Sun JY, Mahmud J, Mansur SMH, Huang S, Moran K, Rubin J, Fazzini M. AndroR2: A dataset of manually-reproduced bug reports for Android APPs. In: Proc. of the 18th IEEE/ACM Int'l Conf. on Mining Software Repositories. Madrid: IEEE, 2021. 600–604. [doi: 10.1109/MSR52588.2021.00082]

[32] Appium. 2024. https://github.com/appium/appium

[33] Apktool. https://apktool.org

[34] Vallée-Rai R, Co P, Gagnon E, Hendren L, Lam P, Sundaresan V. Soot: A Java bytecode optimization framework. In: Proc. of the 2010 CASCON First Decade High Impact Papers. Toronto: IBM, 2010. 214–224. [doi: 10.1145/1925805.1925818]

[35] Networkx. 2024. https://github.com/networkx

[36] NLTK. https://www.nltk.org

[37] Sentence-Transformers. 2024. https://github.com/UKPLab/sentence-transformers

[38] sklearn. 2024. https://github.com/topics/sklearn

[39] Logcat. 2024. https://developer.android.com/tools/logcat

[40] OpenCV. https://opencv.org

[41] pre-trained ChatGPT model. 2024. https://platform.openai.com/docs/models

[42] Chen N, Kim S. STAR: Stack trace based automatic crash reproduction via symbolic execution. IEEE Trans. on Software Engineering, 2015, 41(2): 198–220. [doi: 10.1109/TSE.2014.2363469]

[43] Rößler J, Zeller A, Fraser G, Zamfir C, Candea G. Reconstructing core dumps. In: Proc. of the 6th IEEE Int'l Conf. on Software Testing, Verification and Validation. Luxembourg: IEEE, 2013. 114–123. [doi: 10.1109/ICST.2013.18]

[44] Soltani M, Derakhshanfar P, Devroey X, van Deursen A. A benchmark-based evaluation of search-based crash reproduction. Empirical Software Engineering, 2020, 25(1): 96–138. [doi: 10.1007/s10664-019-09762-1]

[45] Soltani M, Panichella A, van Deursen A. A guided genetic algorithm for automated crash reproduction. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering. Buenos Aires: IEEE, 2017. 209–220. [doi: 10.1109/ICSE.2017.27]

[46] Xuan JF, Xie XY, Monperrus M. Crash reproduction via test case mutation: Let existing test cases help. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 910–913. [doi: 10.1145/2786805.2803206]

[47] Dang YN, Wu RX, Zhang HY, Zhang DM, Nobel P. ReBucket: A method for clustering duplicate crash reports based on call stack similarity. In: Proc. of the 34th Int'l Conf. on Software Engineering. Zurich: IEEE, 2012. 1084–1093. [doi: 10.1109/ICSE.2012.6227111]

[48] Khvorov A, Vasiliev R, Chernishev G, Rodrigues IM, Koznov D, Povarov N. S3M: Siamese stack (trace) similarity measure. In: Proc. of the 18th IEEE/ACM Int'l Conf. on Mining Software Repositories. Madrid: IEEE, 2021. 266–270. [doi: 10.1109/MSR52588.2021.00038]

[49] Rodrigues IM, Aloise D, Fernandes ER. FaST: A linear time stack trace alignment heuristic for crash report deduplication. In: Proc. of the 19th Int'l Conf. on Mining Software Repositories. Pittsburgh: ACM, 2022. 549–560. [doi: 10.1145/3524842.3527951]

[50] Rodrigues IM, Khvorov A, Aloise D, Vasiliev R, Koznov D, Fernandes ER, Chernishev G, Luciv D, Povarov N. TraceSim: An alignment method for computing stack trace similarity. Empirical Software Engineering, 2022, 27(2): 53. [doi: 10.1007/s10664-021-10070-w]

[51] Cui ZQ, Lin GY, Zheng LW, Zhang ZH. A fast crash reproduction method for Android applications based on widget hierarchy graphs. IEEE Internet of Things Journal, 2024, 11(8): 13217–13230. [doi: 10.1109/JIOT.2024.3357209]

[52] White M, Linares-Vásquez M, Johnson P, Bernal-Cárdenas C, Poshyvanyk D. Generating reproducible and replayable bug reports from Android application crashes. In: Proc. of the 23rd IEEE Int'l Conf. on Program Comprehension. Florence: IEEE, 2015. 48–59. [doi: 10.1109/ICPC.2015.14]

## 附中文参考文献

[5] 李聪, 蒋炎岩, 许畅. 基于 GUI 事件的安卓应用录制重放关键技术综述. 软件学报, 2022, 33(5): 1612–1634. http://www.jos.org.cn/1000-9825/6551.htm [doi: 10.13328/j.cnki.jos.006551]

[23] 林高毅, 崔展齐, 陈翔, 郑丽伟. 状态转换图制导的 ARP 错误检测方法. 软件学报, 2025, 36(2): 469–487. http://www.jos.org.cn/1000-9825/7142.htm [doi: 10.13328/j.cnki.jos.007142]

[29] 郝蕊, 冯洋, 李玉莹, 陈振宇. 基于事件标记的多粒度结合安卓测试序列约减. 软件学报, 2025, 36(5): 2006–2025. http://www.jos.org.cn/1000-9825/7203.htm [doi: 10.13328/j.cnki.jos.007203]

## 作者简介

**刘姝琪**, 博士生, CCF 学生会员, 主要研究领域为移动应用测试.

**周宇**, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为智能软件工程, 软件演化分析, 软件验证.

**杨慧文**, 博士生, 主要研究领域为智能软件测试技术.