LLM-Explorer: Towards Efficient and Affordable LLM-based Exploration for Mobile Apps

LLM-Explorer: 面向移动应用的高效且经济的基于大型语言模型 (LLM) 探索

Shanhui Zhao [1,†] , Hao Wen [1,†] , Wenjie Du [1,2] , Cheng Liang [1,3] , Yunxin Liu [1,5] , Xiaozhou Ye[4] , Ye Ouyang [4] , Yuanchun Li [1,5,6,‡]

赵善辉 [1,†]，文浩 [1,†]，杜文杰 [1,2]，梁成 [1,3]，刘云鑫 [1,5]，小舟 Ye[4]，欧阳烨 [4]，李元春 [1,5,6,‡]

[1] Institute for AI Industry Research (AIR), Tsinghua University [2] Hong Kong University of Science and Technology [3] Beijing University of Posts and Telecommunications [4] AsiaInfo Technologies (China), Inc [5] Shanghai Artificial Intelligence Laboratory [6] Beijing Academy of Artificial Intelligence (BAAI)

[1] 清华大学人工智能产业研究院 (AIR) [2] 香港科技大学 [3] 北京邮电大学 [4] 亚信科技 (中国) 有限公司 [5] 上海人工智能实验室 [6] 北京人工智能研究院 (BAAI)

ABSTRACT

摘要

Large language models (LLMs) have opened new opportunities for automated mobile app exploration, an important and challenging problem that used to suffer from the difficulty of generating meaningful UI interactions. However, existing LLM-based exploration approaches rely heavily on LLMs to generate actions in almost every step, leading to a huge cost of token fees and computational resources. We argue that such extensive usage of LLMs is neither necessary nor effective, since many actions during exploration do not require, or may even be biased by the abilities of LLMs. Further, based on the insight that a precise and compact knowledge plays the central role for effective exploration, we introduce LLM-Explorer, a new exploration agent designed for efficiency and affordabil-ity. LLM-Explorer uses LLMs primarily for maintaining the knowledge instead of generating actions, and knowledge is used to guide action generation in a LLM-less manner. Based on a comparison with 5 strong baselines on 20 typical apps, LLM-Explorer was able to achieve the fastest and highest coverage among all automated app explorers, with over 148x lower cost than the state-of-the-art LLM-based approach.

大型语言模型 (LLMs) 为自动化移动应用探索开辟了新机遇，这是一项重要且具有挑战性的问题，过去因难以生成有意义的用户界面 (UI) 交互而受限。然而，现有基于 LLM 的探索方法几乎在每一步都依赖 LLM 生成动作，导致代币费用和计算资源消耗巨大。我们认为如此频繁使用 LLM 既非必要也非高效，因为探索过程中的许多动作并不需要，甚至可能因 LLM 能力而产生偏差。此外，基于精准且紧凑的知识在有效探索中起核心作用的洞见，我们提出了 LLM-Explorer，一种旨在提高效率和经济性的全新探索代理。LLM-Explorer 主要利用 LLM 维护知识，而非生成动作，知识则用于以无 LLM 方式指导动作生成。通过与 20 个典型应用上的 5 个强基线比较，LLM-Explorer 在所有自动化应用探索器中实现了最快且最高的覆盖率，成本比最先进的基于 LLM 方法低 148 倍以上。

CCS CONCEPTS

计算机分类系统 (CCS) 概念

- Human-centered computing → Ubiquitous and mobile computing; • Computing methodologies → Artificial intelligence.

- 以人为中心的计算 → 普适与移动计算；• 计算方法学 → 人工智能。

KEYWORDS

关键词

Mobile App Exploration, Large Language Models, Software Testing, LLM-based Agent

移动应用探索，大型语言模型，软件测试，基于 LLM 的代理

ACM Reference Format:

ACM 引用格式:

Shanhui Zhao [1,†] , Hao Wen [1,†] , Wenjie Du [1,2] , Cheng Liang [1,3] „ Yunxin Liu [1,5] , Xiaozhou Ye[4] , Ye Ouyang [4] , Yuanchun Li[1,5,6,‡] . 2025. LLM-Explorer: Towards Efficient and Affordable LLM-based Exploration for Mobile Apps. In The 31st Annual International Conference on Mobile Computing and Networking (ACM MobiCom '25), November 3-7, 2025, Hong Kong, China. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3680207.3723494

赵善辉 [1,†]，文浩 [1,†]，杜文杰 [1,2]，梁成 [1,3]，刘云鑫 Liu [1,5]，小舟 Ye[4]，欧阳烨 [4]，李元春 Li[1,5,6,‡] 。2025。LLM-Explorer: 面向移动应用的高效且经济的基于大型语言模型探索。载于第 31 届国际移动计算与网络会议 (ACM MobiCom '25)，2025 年 11 月 3-7 日，中国香港。ACM，纽约，美国，15 页。https://doi.org/10.1145/3680207.3723494

1 INTRODUCTION

1 引言

Automated mobile app exploration is a long-standing research problem, with lots of important applications including app testing [7, 9, 16 − 19, 22, 23] , malware detection [1, 3, 4, 32] , and in-app data crawling [10, 11, 14]. The performance of mobile app exploration is usually measured by coverage, i.e. the number of activities or lines of code reached in a limited period of time. Higher activity coverage correlates with better overall system performance. Several existing approaches, including Humanoid [17], GPTDroid [21], and DroidAgent [39], have adopted this metric to evaluate their effectiveness. Recently, emerging intelligent smartphone agents [13, 15, 34] also rely on exploration to collect necessary knowledge for task automation. Since the main interface of mobile apps is the graphical user interface (GUI or UI for short), the exploration of mobile apps is also usually grounded by GUI - The exploration agent navigates between different GUI states of an app by sending different GUI actions (touch, scroll, input text, etc.), just like how human users interact with the app.

自动化移动应用探索是一个长期研究问题，具有包括应用测试 [7, 9, 16 − 19, 22, 23] 、恶意软件检测 [1, 3, 4, 32] 和应用内数据爬取 [10, 11, 14] 等多种重要应用。移动应用探索的性能通常通过覆盖率衡量，即在有限时间内达到的活动或代码行数。更高的活动覆盖率通常意味着更好的整体系统性能。已有多种方法，包括 Humanoid [17]、GPTDroid [21] 和 DroidAgent [39]，均采用该指标评估其有效性。近期，涌现的智能手机代理 [13, 15, 34] 也依赖探索收集任务自动化所需的知识。由于移动应用的主要界面是图形用户界面 (GUI 或简称 UI)，移动应用的探索通常基于 GUI——探索代理通过发送不同的 GUI 动作 (触摸、滚动、输入文本等) 在应用的不同 GUI 状态间导航，类似于人类用户与应用的交互方式。

The key question in mobile app exploration is how to generate the GUI actions that can efficiently discover new functionalities in the app. The major policies include random [7, 24, 25] (randomly selecting which UI elements to interact with and how), model-based [2, 16, 31, 38] (creating a model or representation of the UI and derive test cases to systematically cover different parts of the app functions), and learning-based [12, 17, 37] (leveraging machine learning or

移动应用探索中的关键问题是如何生成能够高效发现应用新功能的 GUI 操作。主要策略包括随机策略 [7, 24, 25](随机选择交互的 UI 元素及方式)、基于模型的策略 [2, 16, 31, 38](构建 UI 的模型或表示，并推导测试用例以系统覆盖应用功能的不同部分) 以及基于学习的策略 [12, 17, 37](利用机器学习或

ACM MobiCom ’25, November 3-7, 2025, Hong Kong, China

ACM MobiCom ’25，2025 年 11 月 3-7 日，中国香港

ACM ISBN 979-8-4007-1129-9/25/11...$15.00

ACM ISBN 979-8-4007-1129-9/25/11...$15.00

https://doi.org/10.1145/3680207.3723494

---

† Co-primary authors.

† 共同第一作者。

‡ Corresponding author: Yuanchun Li (liyuanchun@air.tsinghua.edu.cn).

‡ 通讯作者: 李元春 (liyuanchun@air.tsinghua.edu.cn)。

reinforcement learning techniques to generate test actions). Many explorers adopt a mixture of different policies. Despite lots of existing attempts, there is still much room of improvement in app exploration. Mobile apps are dynamic, featuring a wide variety of user interfaces with diverse combinations of UI elements, and unique UI actions leading to different states. This complexity makes comprehensive exploration challenging, emphasizing the need for strategic testing methods [19, 28, 30, 40]. Therefore, we aim to efficiently cover as much app functionality as possible without needing to explore every single UI state, which might be infinite. Achieving higher and faster coverage requires a deep understanding of the apps' functionalities and historical traces, which is difficult for most existing exploration agents.

强化学习技术生成测试操作)。许多探索者采用多种策略的混合。尽管已有大量尝试，应用探索仍有很大提升空间。移动应用具有动态特性，拥有多样的用户界面和不同 UI 元素的组合，以及导致不同状态的独特 UI 操作。这种复杂性使得全面探索具有挑战性，强调了战略性测试方法的必要性 [19, 28, 30, 40]。因此，我们旨在高效覆盖尽可能多的应用功能，而无需探索每一个可能的 UI 状态 (可能是无限的)。实现更高更快的覆盖率需要对应用功能和历史轨迹有深入理解，而这对大多数现有探索代理来说是困难的。

Recently, pretrained foundation models, represented by large language models (LLMs), have demonstrated remarkable performance in language understanding, reasoning, and generation. Which makes it possible for LLM-based agents to understand and execute tasks in a human-like fashion. Since the app GUI can also be represented by natural language text and images, the LLM-based agents can potentially better understand the functionalities of apps and therefore improve the exploration performance. Such an idea has already been studied in prior work [30]. For example, GPTDroid [21] introduces a method to generate testing actions by directly passing the current GUI information and historic trace to LLMs. DroidAgent [39] uses multiple autonomous agents for generating unexplored tasks, observing the interface, generating actions, and judging and reflecting on task completion. These approaches have demonstrated the ability of LLM to generate more meaningful GUI actions.

近年来，以大型语言模型 (LLMs) 为代表的预训练基础模型在语言理解、推理和生成方面表现出色，使得基于 LLM 的代理能够以类人方式理解和执行任务。由于应用 GUI 也可以用自然语言文本和图像表示，基于 LLM 的代理有潜力更好地理解应用功能，从而提升探索性能。这一思路已在先前工作中得到研究 [30]。例如，GPTDroid[21] 提出通过直接传递当前 GUI 信息和历史轨迹给 LLM 来生成测试操作的方法。DroidAgent[39] 使用多个自主代理生成未探索任务、观察界面、生成操作并判断及反思任务完成情况。这些方法展示了 LLM 生成更有意义 GUI 操作的能力。

However, existing LLM-based exploration approaches have led to two important issues, limited efficiency and huge cost. The two issues originate from one cause - the excessive dependence on LLMs. Existing approaches extensively query the LLMs to generate or plan steps, which is slow and costly (e.g. generating a GUI action typically consumes around 500 tokens and 3 seconds, and exploring an app usually takes thousands of steps). Actually, most steps during exploration do not demand much LLM-based reasoning and planning, which is analogous to human exploration of unknown places that is aimless at most times. Moreover, asking LLMs to generate each action may bias the exploration process against unusual use cases, which are also important for exploration.

然而，现有基于 LLM 的探索方法存在两个重要问题: 效率有限和成本高昂。两者均源于对 LLM 的过度依赖。现有方法大量调用 LLM 生成或规划步骤，速度慢且成本高 (例如生成一次 GUI 操作通常消耗约 500 个 token 和 3 秒，探索一个应用通常需数千步)。实际上，大多数探索步骤并不需要复杂的 LLM 推理和规划，这类似于人类探索未知场所时大部分时间的无目的行为。此外，要求 LLM 生成每个操作可能导致探索过程偏向常规用例，而忽视同样重要的异常用例。

To address the above problems, we propose to combine the ability of LLMs and careful interaction knowledge modeling in mobile app exploration. Our key insight is that the cornerstone of efficient exploration is the knowledge maintenance and utilization, instead of the action generation and planning abilities. Specifically, similar to human or robots exploring a new environment, the key to efficient app exploration is reducing repetitive meaningless actions (i.e. being creative),

为解决上述问题，我们提出结合 LLM 能力与移动应用探索中的交互知识建模。我们的核心见解是，高效探索的基石是知识的维护与利用，而非操作生成与规划能力。具体而言，类似于人类或机器人探索新环境，高效应用探索的关键在于减少重复且无意义的操作 (即具备创造力),

rather than planning future actions at each step (i.e. being foresighted). Since LLMs are trained by learning patterns from large datasets, they are usually considered to have only weak forms of creativity [8]. On the contrary, an agent can be creative during exploration if it compares the candidate actions (and action combinations) against a high-quality knowledge. Maintaining the knowledge is mainly about information summarization, where LLMs can be more helpful.

而非每一步都规划未来操作 (即具备远见)。由于 LLM 通过大规模数据学习模式训练，通常被认为仅具备弱形式的创造力 [8]。相反，如果代理能将候选操作 (及其组合) 与高质量知识进行比较，则能在探索中展现创造力。知识维护主要涉及信息总结，LLM 在这方面能发挥更大作用。

Based on the insight, LLM-Explorer handles the automatic app exploration process with two main modules: LLM-assisted Knowledge Maintenance and Knowledge-guided Exploration. The LLM-assisted Knowledge Maintenance module is responsible for recording and categorizing reached UIs to prevent repetitive explorations or loops. This is crucial as variations in UI states or actions can make it challenging, even for LLMs. To address this, we introduce an app knowledge that contains abstract representations of UI states, elements and actions, as well as abstract interaction graphs to track transitions between UI states. The Knowledge-guided Exploration module then selects the next UI action to execute based on the maintained knowledge, with LLMs being occasionally invoked to tackle particularly complex UI actions (e.g. text input). This approach aims to enhance exploration efficiency by reducing unnecessary LLM queries and focusing on strategic knowledge management and interaction planning.

基于这一洞察，LLM-Explorer 通过两个主要模块处理自动化应用探索过程:LLM 辅助的知识维护和知识引导的探索。LLM 辅助的知识维护模块负责记录和分类已到达的 UI，以防止重复探索或循环。这一点至关重要，因为 UI 状态或操作的变化即使对 LLM 来说也具有挑战性。为此，我们引入了一种应用知识，包含 UI 状态、元素和操作的抽象表示，以及用于跟踪 UI 状态转换的抽象交互图。知识引导的探索模块则基于维护的知识选择下一步执行的 UI 操作，LLM 会在遇到特别复杂的 UI 操作 (如文本输入) 时偶尔被调用。该方法旨在通过减少不必要的 LLM 查询，专注于战略性的知识管理和交互规划，从而提升探索效率。

We evaluate the effectiveness of our LLM-Explorer approach on 20 apps, in comparison with strong baselines including DroidAgent, GPTDroid, Humanoid, Droidbot, and Monkey. We also included the human exploration performance for reference. The results have demonstrated that LLM-Explorer can achieve 4%-35% higher activity coverage than the baselines within a fixed time, matching human-level exploration efficiency on some apps. Compared with other LLM-based exploration approaches (DroidAgent and GPTDroid), LLM-Explorer can reduce the LLM cost by 9x-148x times.

我们在 20 个应用上评估了 LLM-Explorer 方法的有效性,并与包括 DroidAgent、GPTDroid、Humanoid、Droidbot 和 Monkey 在内的强基线进行了比较。我们还包含了人工探索性能作为参考。结果表明，LLM-Explorer 在固定时间内的活动覆盖率比基线高出 4%-35%，在某些应用上达到了人类级别的探索效率。与其他基于 LLM 的探索方法 (DroidAgent 和 GPTDroid) 相比，LLM-Explorer 能将 LLM 成本降低 9 倍至 148 倍。

Our work makes the following technical contributions:

我们的工作做出了以下技术贡献:

(1) We study the LLM-based mobile app exploration with specific consideration on affordability, which can potentially benefit cost-sensitive individual mobile app developers and market-scale app analyzers.

(1) 我们研究了基于 LLM 的移动应用探索，特别关注其可负担性，这有望惠及成本敏感的个人移动应用开发者和市场规模的应用分析者。

(2) We propose a new abstraction of an app's exploration knowledge, which is maintained by LLMs and can guide efficient future exploration. We believe this abstraction is also useful for a broader scope of app analysis.

(2) 我们提出了一种新的应用探索知识抽象，由 LLM 维护，可指导高效的未来探索。我们认为该抽象对更广泛的应用分析领域也有用处。

(3) Through a comprehensive evaluation, we demonstrate the effectiveness of our approach over strong baselines and the potential to advance the field of mobile app exploration.

(3) 通过全面评估，我们展示了该方法相较强基线的有效性及其推动移动应用探索领域发展的潜力。

Our code is open-sourced at https://github.com/MobileLLM/ LLM-Explorer.

我们的代码已开源，地址为 https://github.com/MobileLLM/LLM-Explorer。

2 BACKGROUND AND MOTIVATION

2 背景与动机

2.1 Mobile App Exploration

2.1 移动应用探索

App exploration is also called app traversal, crawling, or fuzzing based on different usage scenarios [14, 16, 40]. The goal of automated mobile app exploration is to traverse the functions of an app by automatically interacting with it. Since the main interface of mobile apps is GUI, the output of app exploration is usually a sequence of GUI actions, including touching, scrolling, typing text, etc. Based on different purposes of app exploration, the output of exploration may also include the comprehensive report detailing the discovered bugs or issues, the performance metrics of the app, the data crawled from the app, or the screenshots or videos recording the exploration process. More efficient and sufficient exploration usually leads to better performance in downstream applications, such as more bugs detected (for app testing), more precise malware classification (for malware detection), and more accurate task automation (for task automation).

应用探索也称为应用遍历、爬取或模糊测试，具体名称取决于不同的使用场景 [14, 16, 40]。自动化移动应用探索的目标是通过自动交互遍历应用的功能。由于移动应用的主要界面是图形用户界面 (GUI)，应用探索的输出通常是一系列 GUI 操作，包括点击、滚动、输入文本等。根据探索的不同目的，探索结果还可能包括详细的发现缺陷或问题的综合报告、应用的性能指标、从应用中爬取的数据，或记录探索过程的截图或视频。更高效且充分的探索通常能提升下游应用的表现，如检测更多缺陷 (用于应用测试)、更精准的恶意软件分类 (用于恶意软件检测) 以及更准确的任务自动化 (用于任务自动化)。

The major performance metric of app exploration is the coverage. Unlike software testing approaches whose performance is usually measured with code coverage (i.e. number of reached source code lines), mobile app explorers usually deal with apps without source code available. The fundamental building block of an Android application is called activity, which represents a function component that provides a UI screen for users to interact with. For instance, an email app has different activities for different functions including the inbox, reading email, editing email, and settings. Therefore, measuring the effectiveness of an automated app explorer is often based on the activity coverage, i.e. how many unique activities can be reached within a fixed time.

应用探索的主要性能指标是覆盖率。与通常以代码覆盖率 (即达到的源代码行数) 衡量的软件测试方法不同，移动应用探索通常面对无源代码的应用。Android 应用的基本构建单元称为 activity(活动)，它代表一个功能组件，为用户提供交互的 UI 界面。例如，邮件应用有不同的 activity 对应不同功能，包括收件箱、阅读邮件、编辑邮件和设置。因此，衡量自动化应用探索器的有效性通常基于 activity 覆盖率，即在固定时间内能达到多少唯一的 activity。

Besides, implementing a mobile app explorer involves several key concepts about the GUI: 1) UI state refers to the current state of the mobile app visible through the user interface, which can interact with a user or an automatic explorer. 2) UI element is a visual component that users can interact with in a UI screen, including buttons, checkboxes, input boxes, etc. 3) UI action encompasses the interactions (such as clicks, text inputs, swipes) performed by users or automatic explorer to on UI elements to navigate or manipulate the app's functionality. The job of an explorer is to send appropriate UI actions to navigate between different UI states.

此外，实现移动应用探索器涉及几个关于 GUI 的关键概念:1)UI 状态指通过用户界面可见的移动应用当前状态，用户或自动探索器可与之交互。2)UI 元素是用户可在 UI 界面中交互的视觉组件，包括按钮、复选框、输入框等。3)UI 操作涵盖用户或自动探索器对 UI 元素执行的交互 (如点击、文本输入、滑动)，以导航或操作应用功能。探索器的任务是发送适当的 UI 操作以在不同 UI 状态间导航。

2.2 Difficulties of App Exploration

2.2 应用探索的难点

Despite the numerous efforts in automatic app exploration, it faces fundamental challenges that hinder its further improvement. Existing app explorers can hardly match the efficiency and effectiveness of human users in traversing the app functions. The main difficulties include:

尽管自动应用探索已有大量努力，但仍面临阻碍其进一步提升的根本挑战。现有的应用探索器难以匹敌人类用户在遍历应用功能时的效率和效果。主要难点包括:

1) Dynamic UI states. Many apps can dynamically change content and layouts based on user data, interactions, or even

1) 动态 UI 状态。许多应用会基于用户数据、交互甚至

preferences. For example, a contact list's UI may constantly evolve due to additions or modifications to the contacts, creating potentially infinite states. This variability can trap automatic explorers in endless loops if they fail to recognize these changes as the same state.

偏好动态改变内容和布局。例如，联系人列表的 UI 可能因联系人添加或修改而不断变化，产生潜在的无限状态。如果自动探索器未能识别这些变化为同一状态，可能陷入无尽循环。

2) Large action space. Some apps include a large number of interactive UI elements, a lot of which are similar or repetitive, (e.g. time selectors, date pickers, checkbox list). This complexity makes comprehensive app exploration difficult, necessitating strategic testing plans to manage the extensive variety of options effectively.

2) 动作空间大。一些应用包含大量交互式 UI 元素，其中许多相似或重复 (例如时间选择器、日期选择器、复选框列表)。这种复杂性使得全面探索应用变得困难，需制定策略性测试计划以有效管理大量选项。

3) Non-deterministic behaviors. Mobile apps may exhibit unpredictable behaviors due to various factors, such as network latency, server-side processing, or interactions with other apps and services. These inconsistencies can make it challenging to navigate to specific UI states, hindering the explorer's ability to conduct thorough explorations.

3) 非确定性行为。移动应用可能因网络延迟、服务器端处理或与其他应用和服务的交互等多种因素表现出不可预测的行为。这些不一致性使得导航到特定 UI 状态变得困难，阻碍探索者进行全面探索。

2.3 AI and LLM for Mobile App Exploration

2.3 移动应用探索中的人工智能与大型语言模型 (LLM)

The aforementioned difficulties are mainly due to the lack of semantic understanding of GUI interactions, where AI techniques, especially LLMs can be helpful. Machine learning models have the potential to enhance UI exploration efficiency by detecting dynamic UI changes and reducing repetitive actions. They excel in understanding UI component dependencies and relationships, enabling them to generate context-based test cases [10, 29].

上述困难主要源于缺乏对 GUI 交互的语义理解，而人工智能技术，尤其是大型语言模型 (LLM)，在这方面具有帮助作用。机器学习模型通过检测动态 UI 变化和减少重复操作，有潜力提升 UI 探索效率。它们擅长理解 UI 组件的依赖关系和关联，从而能够生成基于上下文的测试用例 [10, 29]。

However, most existing AI-based exploration methods, including deep learning [17, 36] and reinforcement learning approaches [26, 27, 35], depend heavily on extensive training data and struggle with application generalization [6, 17]. Despite various innovations, these methods often fall short of the human ability to quickly identify interactive elements, sometimes causing inefficiency or endless loops. LLMs, with their extensive pretraining on large-scale data and alignment with human preferences, present remarkable semantic understanding and zero-shot generalization ability for unseen apps and pages. This makes using LLMs a promising solution for mimicking human users' capabilities in app exploration [21, 39].

然而，大多数现有基于 AI 的探索方法，包括深度学习 [17, 36] 和强化学习方法 [26, 27, 35]，高度依赖大量训练数据，且在应用泛化方面存在困难 [6, 17]。尽管有诸多创新，这些方法往往难以达到人类快速识别交互元素的能力，有时导致效率低下或陷入死循环。大型语言模型凭借其在大规模数据上的广泛预训练及与人类偏好的对齐，展现出卓越的语义理解能力和对未见应用及页面的零样本泛化能力，使其成为模拟人类用户应用探索能力的有前景的解决方案 [21, 39]。

Nevertheless, applying LLMs to mobile app exploration still presents several challenges. First, utilizing LLMs, whether deployed on the cloud or locally, incurs high inference costs. Accessing LLMs often involves considerable delays and expenses. Given that thoroughly exploring an app with high activity coverage typically needs numerous steps to explore an app (1,000+ steps at least), querying LLMs at each step leads to significant latency and costs. Second, LLMs could fall short in the creative thinking required for comprehensive app testing. They may tend to consistently select the main functions of the app based on its common sense while overlooking less apparent features [8]. This oversight can restrict

然而，将大型语言模型应用于移动应用探索仍面临若干挑战。首先，无论是在云端还是本地部署,使用大型语言模型都伴随高昂的推理成本。访问大型语言模型通常涉及显著的延迟和费用。鉴于全面探索一个应用以实现高活动覆盖通常需要大量步骤(至少1000步以上)，每步调用大型语言模型会导致显著的延迟和成本。其次，大型语言模型在全面应用测试所需的创造性思维方面可能存在不足。它们可能倾向于基于常识持续选择应用的主要功能，而忽视不那么明显的特性[8]。这种忽视可能限制

the exploration of all app activities and hinder the discovery of critical issues.

对所有应用活动的探索，阻碍关键问题的发现。

3 OUR APPROACH: LLM-EXPLORER

3 我们的方法:LLM-EXPLORER

3.1 Overview

3.1 概述

We propose LLM-Explorer, an automated mobile application exploration system powered by LLMs to address the aforementioned challenges. The main idea of LLM-Explorer is to maintain a high-quality app knowledge to guide the exploration process. As shown in Figure 1, LLM-Explorer comprises two main modules: the LLM-assisted Knowledge Maintenance module and the Knowledge-guided Exploration module.

我们提出了LLM-Explorer, 一种由大型语言模型驱动的自动化移动应用探索系统, 以应对上述挑战。LLM-Explorer的核心思想是维护高质量的应用知识以指导探索过程。如图1所示，LLM-Explorer包含两个主要模块: 大型语言模型辅助的知识维护模块和知识引导的探索模块。

The LLM-assisted Knowledge Maintenance module utilizes LLMs to simplify the raw data of the exploration process, forming abstract app knowledge. It merges states and actions with similar functions into abstract states and actions, creating an Abstract Interaction Graph for smooth and efficient app navigation. The Knowledge-guided Exploration module utilizes the insights stored in the app knowledge, strategically choosing UI actions and devising test plans with context awareness. If the chosen action is not executable in the current UI state, LLM-Explorer navigates to the corresponding UI states using the Abstract Interaction Graph for guidance.

大型语言模型辅助的知识维护模块利用大型语言模型简化探索过程的原始数据, 形成抽象的应用知识。它将功能相似的状态和动作合并为抽象状态和动作，构建抽象交互图，实现流畅高效的应用导航。知识引导的探索模块利用存储在应用知识中的洞见，策略性地选择UI动作并制定具备上下文感知的测试计划。如果所选动作在当前UI状态下不可执行，LLM-Explorer将利用抽象交互图引导导航至相应UI状态。

3.2 LLM-assisted Knowledge Maintenance

3.2 大型语言模型辅助的知识维护

The input of the LLM-assisted Knowledge Maintenance module is a raw UI exploration trace, including a list of UI test steps featuring UI states, elements, and actions. The module maintains a high-quality app knowledge composed of abstract UI states, elements, actions, and an Abstract Interaction Graph (AIG). Prior approaches [16, 17] also maintain a UI transition graph composed of raw states and actions, while we try to compress the knowledge components based on their semantic meanings. The idea is simple yet effective as it imitates how human users memorize concepts and reduce repetitive behaviors.

大型语言模型辅助的知识维护模块的输入是原始 UI 探索轨迹，包括包含 UI 状态、元素和动作的 UI 测试步骤列表。该模块维护由抽象 UI 状态、元素、动作及抽象交互图 (AIG) 组成的高质量应用知识。先前方法 [16, 17] 也维护由原始状态和动作组成的 UI 转换图，而我们尝试基于语义意义压缩知识组件。该思路简单而有效，模拟了人类用户记忆概念和减少重复行为的方式。

There are two problems to solve for maintaining the app knowledge: (i) Identifying UI states is crucial as seemingly identical UIs may represent different states, while dynamically changing UIs might belong to the same state. This distinction prevents excessive growth of app knowledge and aids in choosing efficient navigation paths. (ii) Reducing the UI action space is necessary because UIs can have many similar components, such as a calendar app's date elements for a month. Documenting every UI action increases redundancy and complicates the exploration module.

维护应用知识需解决两个问题:(i) 识别 UI 状态至关重要，因为表面相同的 UI 可能代表不同状态，而动态变化的 UI 可能属于同一状态。此区分防止应用知识过度膨胀，有助于选择高效的导航路径。(ii) 减少 UI 动作空间是必要的，因为 UI 可能包含许多相似组件，如日历应用中一个月的日期元素。记录每个 UI 动作会增加冗余，复杂化探索模块。

3.2.1 Knowledge Organization. LLM-Explorer builds up the exploration knowledge by summarizing the raw interaction trace. Each step of the raw trace includes the UI state,

3.2.1 知识组织。LLM-Explorer 通过总结原始交互轨迹构建探索知识。原始轨迹的每一步包括 UI 状态，

the UI element in the state, and the UI action type (touch, scroll, etc.). To properly compress the knowledge, we introduce abstract UI states, elements, actions, and interaction graph in LLM-Explorer.

状态中的 UI 元素及 UI 动作类型 (触摸、滚动等)。为有效压缩知识，LLM-Explorer 引入了抽象 UI 状态、元素、动作及交互图。

An abstract UI state, symbolized as s_i^{abs} , represents a group of UI states $s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(j)}$ encountered during exploration that serve the same set of functions within an app. The actual states of an abstract state may differ visually with different content (e.g. two Contacts screens with different contact names), but they are expected to share similar use cases.

抽象 UI 状态，表示为 s_i^{abs} ，代表在探索过程中遇到的一组 UI 状态 $s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(j)}$ ，它们在应用中执行相同的一组功能。抽象状态的实际状态在视觉上可能因内容不同而有所差异 (例如两个联系人屏幕显示不同的联系人姓名)，但它们预期具有相似的使用场景。

Abstract UI actions group together similar user interactions, often invoking the same function or API call but with different parameters. This method identifies repetitive components within an application (such as dates or contact names) that usually result in the same abstract UI state upon interaction. By categorizing these interactions under a single abstract action, it simplifies the action space and avoids potential infinite loops. An abstract UI action is defined by four key components: Action Type, Actual Element, Exploration Flag, and Function: 1) Action Type includes the four most common interactions: touch, long touch, scroll, and input. 2) Actual Element refers to specific UI elements involved in the action. These can either be a collection of elements within a single UI state that share similar functions or elements that are positioned identically across multiple actual states within an abstract state. 3) Exploration Flag indicates the exploration status of an action with three possible values: unexplored (not yet executed), explored (has been executed by the exploration module), and ineffective (found to be non-functional or leading to no UI changes). 4) Function denotes the functionality of the group of actual elements summarized by LLM.

抽象 UI 动作将相似的用户交互归为一组，通常调用相同的功能或 API 接口，但参数不同。该方法识别应用中重复的组件 (如日期或联系人姓名)，这些组件在交互后通常导致相同的抽象 UI 状态。通过将这些交互归类为单一的抽象动作，简化了动作空间，避免潜在的无限循环。抽象 UI 动作由四个关键组成部分定义: 动作类型、实际元素、探索标志和功能:1) 动作类型包括四种最常见的交互: 触摸、长按、滚动和输入。2) 实际元素指参与动作的具体 UI 元素，这些元素可以是单一 UI 状态中具有相似功能的一组元素，或在抽象状态内多个实际状态中位置相同的元素。3) 探索标志表示动作的探索状态，有三种可能值: 未探索 (尚未执行)、已探索 (已被探索模块执行) 和无效 (被发现无功能或不引起 UI 变化)。4) 功能表示由大语言模型 (LLM) 总结的实际元素组的功能。

The Abstract Interaction Graph (AIG) is a directed graph, whose nodes represent abstract UI states, and edges correspond to abstract UI actions. The source of the edge (abstract action) is the abstract UI state it is performed on, and the target is the resulting UI state after the action is performed. This model offers a more concise representation compared to the conventional UI Transition Graph (UTG) [10, 16]. By omitting redundant actions and consolidating dynamic UI states, the AIG enables a more efficient navigation of UI states.

抽象交互图 (AIG) 是一个有向图，其节点代表抽象 UI 状态，边对应抽象 UI 动作。边的起点 (抽象动作) 是执行该动作的抽象 UI 状态，终点是动作执行后产生的 UI 状态。与传统的 UI 转换图 (UTG)[10,16] 相比，该模型提供了更简洁的表示。通过省略冗余动作和合并动态 UI 状态，AIG 实现了对 UI 状态的更高效导航。

3.2.2 Knowledge Update. LLM-Explorer updates the app knowledge after each exploration step. Each exploration step produces a tuple, representing the starting UI state (s') , the UI action taken (a') , and the ensuing UI state (s) .

3.2.2 知识更新。LLM-Explorer 在每次探索步骤后更新应用知识。每个探索步骤产生一个元组，表示起始 UI 状态 (s') 、执行的 UI 动作 (a') 和随后的 UI 状态 (s) 。

Firstly, the abstract states are updated by matching the new UI state with existing states. We attempted two methods to match the abstract UI states: the rule-based method and the LLM-based method. The rule-based method capitalizes on the insight that UIs with similar functions, such as $s_i^{(m)}$ and $s_i^{(n)}$, often share comparable element structures. The differences lie in some dynamic UI element properties such as text content,

首先，通过将新的 UI 状态与现有状态匹配来更新抽象状态。我们尝试了两种方法来匹配抽象 UI 状态: 基于规则的方法和基于 LLM 的方法。基于规则的方法利用了这样一个见解: 具有相似功能的 UI, 如 $s_i^{(m)}$ 和 $s_i^{(n)}$ ，通常具有相似的元素结构。差异主要体现在一些动态 UI 元素属性上，如文本内容,

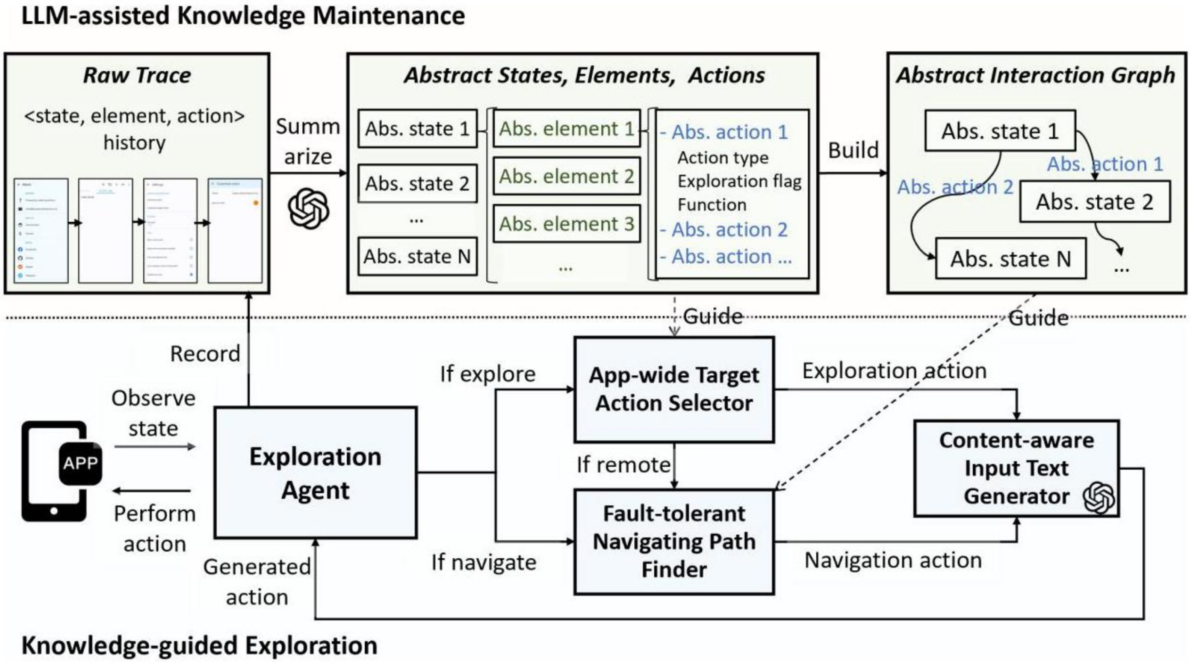


Figure 1: The workflow of LLM-Explorer.

图 1:LLM-Explorer 的工作流程。

selection status, scrolling positions, etc. Specifically, after excluding these dynamic properties, if two states $s_i^{(m)}$ and $s_i^{(n)}$ have the same set of elements, we combine them into a singular abstract UI state s_i^{abs} . On the other hand, the LLM-based method queries LLMs to determine if two UI states are functionally equivalent. It is based on the former researches that prove LLMs to be effective for summarizing UI functions and answering UI-related questions [29, 34]. We found that using the rule-based method is more effective for merging the abstract states. If the new UI state s does not belong to either of the abstract UI state in the app knowledge, LLM-Explorer recognizes it as a new abstract UI state s_i^{abs} and adds it to the app knowledge.

选择状态、滚动位置等。具体来说，排除这些动态属性后，如果两个状态 $s_i^{(m)}$ 和 $s_i^{(n)}$ 拥有相同的元素集合，我们将它们合并为单一的抽象 UI 状态 s_i^{abs} 。另一方面，基于 LLM 的方法通过查询大语言模型判断两个 UI 状态是否功能等效。该方法基于先前研究，证明 LLM 在总结 UI 功能和回答 UI 相关问题方面有效 [29,34]。我们发现，使用基于规则的方法在合并抽象状态时更为有效。如果新的 UI 状态 s 不属于应用知识中的任何抽象 UI 状态，LLM-Explorer 将其识别为新的抽象 UI 状态 s_i^{abs} 并将其添加到应用知识中。

Secondly, LLM-Explorer updates the set of abstract UI actions. When encountering a new abstract UI state, LLM-Explorer aggregates possible UI actions that could be performed upon it into abstract UI actions $\{a_1^{abs}, a_2^{abs} \ldots\}$, facilitated by querying LLMs. This is effective because LLMs excel at grouping repetitive UI components into generalized categories, thereby narrowing down the action space significantly. These newly incorporated abstract actions are labeled as unexplored. Meanwhile, for the action a that has just been taken, its status is updated based on the UI state it leads to. The abstract state resulting from this action is compared to the preceding state's abstract state. If the resulting abstract state matches the previous one, it indicates that action a didn't change the state, and the action is marked as ineffective. Otherwise, it is marked as explored.

其次，LLM-Explorer 更新抽象 UI 动作集。当遇到新的抽象 UI 状态时，LLM-Explorer 通过查询大型语言模型 (LLMs) 将可能在该状态下执行的 UI 动作聚合为抽象 UI 动作 $\{a_1^{abs}, a_2^{abs} \ldots\}$ 。这是有效的，因为 LLMs 擅长将重复的 UI 组件归类为通用类别，从而显著缩小动作空间。新加入的抽象动作被标记为未探索。同时，对于刚执行的动作 a ，其状态根据其导致的 UI 状态进行更新。该动作产生的抽象状态与前一状态的抽象状态进行比较。如果结果抽象状态与之前相同，说明动作 a 未改变状态，该动作被标记为无效。否则，标记为已探索。

Thirdly, LLM-Explorer updates the Abstract Interaction Graph G . If the abstract UI state s^{abs} of the current state is newly created, LLM-Explorer creates a new node for the current s^{abs} in G . Subsequently, LLM-Explorer checks whether there is a directed edge from the abstract state $s^{'abs}$ of the last state to s^{abs} in G with the attribute of the executed action. If such an edge does not exist, it is added to the graph.

第三，LLM-Explorer 更新抽象交互图 G 。如果当前状态的抽象 UI 状态 s^{abs} 是新创建的，LLM-Explorer 会在 G 中为当前 s^{abs} 创建一个新节点。随后，LLM-Explorer 检查在 G 中是否存在从上一个状态的抽象状态 $s^{'abs}$ 到 s^{abs} 的带有执行动作属性的有向边。如果不存在，则将该边添加到图中。

The whole process is depicted in Algorithm 1.

整个过程如算法 1 所示。

3.3 Knowledge-guided Exploration Policy

3.3 知识引导的探索策略

3.3.1 Policy Overview. Algorithm 2 illustrates the exploration process. At the beginning, LLM-Explorer analyzes the current UI state and initializes the app's knowledge with it. It then enters a loop where it continuously executes actions, and updates the knowledge. This process continues until all abstract actions in the app's knowledge have been explored, marking the end of the exploration.

3.3.1 策略概述。算法 2 展示了探索过程。开始时，LLM-Explorer 分析当前 UI 状态并以此初始化应用知识。然后进入循环，持续执行动作并更新知识。该过程持续进行，直到应用知识中的所有抽象动作均被探索完毕，探索结束。

At the start of each explore step, the exploration agent first chooses an abstract action from the app knowledge to be executed based on the app-wide action selection method. Next, the agent verifies if the target element of the chosen action is present in the current UI state of the app. If it is, the action is executed immediately. If not, LLM-Explorer navigates to the UI state where the target UI element is located based on the fault-tolerant navigation path finder. After executing the UI action, LLM-Explorer updates the app knowledge with the new state transition, represented as (s_i, a_i, s_{i+1}) .

在每次探索步骤开始时，探索代理首先根据全应用动作选择方法从应用知识中选择一个抽象动作执行。接着，代理验证所选动作的目标元素是否存在于应用当前 UI 状态中。如果存在，立即执行该动作；如果不存在，LLM-Explorer 基于容错导航路径查找器导航至目标 UI 元素所在的 UI 状态。执行 UI 动作后，LLM-Explorer 用新的状态转移更新应用知识，表示为 (s_i, a_i, s_{i+1}) 。

3.3.2 App-wide Action Selector. LLM-Explorer introduces a novel strategy that focuses on individual UI elements across the entire app. This differs significantly from the existing methods that focus on UI state granularity [16, 17, 21]. This strategy stems from the insight that exploring an app by its individual UI elements could be more effective than examining through its various UI states. This is because an application can potentially generate an infinite combination of UI states by mixing and matching different elements. However, the number of distinct UI elements, aside from some minor variations like content descriptions or colors, remains limited. As a result, the exploring algorithm could reach completion once it has examined all the UI elements. This is in contrast to UI state granularity methods, which may find themselves in continuous loops when faced with dynamic UI states. Consequently, at each exploration step, LLM-Explorer selects and executes an abstract UI action from the app knowledge, rather than selecting a UI state to investigate.

3.3.2 全应用动作选择器。LLM-Explorer 引入了一种新策略，聚焦于整个应用中的单个 UI 元素。这与现有聚焦于 UI 状态粒度的方法 [16, 17, 21] 有显著不同。该策略基于洞察: 通过单个 UI 元素探索应用可能比通过各种 UI 状态更有效。因为应用可能通过不同元素的组合生成无限多的 UI 状态，但除了一些细微差异如内容描述或颜色外，不同 UI 元素的数量是有限的。因此，探索算法在检查完所有 UI 元素后即可完成。相比之下，UI 状态粒度方法在面对动态 UI 状态时可能陷入循环。因此，在每个探索步骤，LLM-Explorer 选择并执行应用知识中的抽象 UI 动作，而非选择 UI 状态进行调查。

Algorithm 1 Knowledge Update Process of LLM-Explorer.

算法 1 LLM-Explorer 的知识更新过程。

Input: Existing knowledge K (including raw trace $K.T$, abstract states $K.S$,

输入: 现有知识 K (包括原始轨迹 $K.T$ ，抽象状态 $K.S$ ，

abstract actions $K.A$, abstract interaction graph $K.G$), last state s' , last

抽象动作 $K.A$ ，抽象交互图 $K.G$)，上一个状态 s' ，上一个

action a' , and new state s

动作 a' ，以及新状态 s

Output: Updated knowledge

```
输出: 更新后的知识
  function UPDATEKNOWLEDGE $(K, s', a', s)$ :
  函数 UPDATEKNOWLEDGE $(K, s', a', s)$ :
    Add $(s', a', s)$ to the raw trace $K.T$
    将 $(s', a', s)$ 添加到原始轨迹 $K.T$
    Try classify $s$ to existing abstract states $K.S$ then
    尝试将 $s$ 分类到现有抽象状态 $K.S$ 中
    if $s$ matches existing abstract states $K.S$ then
    如果 $s$ 匹配现有抽象状态 $K.S$ 则
      $s^{abs} \leftarrow$ matched abstract state of $s$ in $K.S$
      $s^{abs} \leftarrow$ 匹配了 $K.S$ 中的 $s$ 的抽象状态
    else
    否则
      $s^{abs} \leftarrow$ create new abstract state from $s$
      $s^{abs} \leftarrow$ 从 $s$ 创建新的抽象状态
      Add $s^{abs}$ into $K.S$
      将 $s^{abs}$ 添加到 $K.S$ 中
    end if
    结束条件判断
    for each new action $a$ in state $s$ do  $\triangleright$ LLM assisted
    对于状态 $s$ 中的每个新动作 $a$ 执行  $\triangleright$ LLM 辅助操作
      if $a$ doesn’t match existing actions $K.A$ then
      如果 $a$ 不匹配现有动作 $K.A$ 则
        $a^{abs} \leftarrow$ create new abstract action from $a$
      $a^{abs} \leftarrow$ 从 $a$ 创建新的抽象动作
      $a^{abs}$ . exploration_flag $\leftarrow$ unexplored
      $a^{abs}$ 的 exploration_flag $\leftarrow$ 为未探索
      Add $a^{abs}$ into $K.A$
      将 $a^{abs}$ 添加到 $K.A$ 中
    end if
    结束条件判断
  end for
  结束循环
  $s'$ abs $\leftarrow$ matched abstract state of $s'$ in $K.S$
  $s'$ abs $\leftarrow$ 匹配了 $K.S$ 中的 $s'$ 的抽象状态
  $a'$ abs $\leftarrow$ matched abstract action of $a'$ in $K.A$
  $a'$ abs $\leftarrow$ 匹配了 $K.A$ 中的 $a'$ 的抽象动作
  $a'^{abs}$ .exploration_flag $\leftarrow$ explored
  $a'^{abs}$ 的 exploration_flag $\leftarrow$ 已探索
  if $s'^{abs} = s^{abs}$ then
  如果 $s'^{abs} = s^{abs}$ 则
    $a'$ abs.exploration_flag.add(ineffective)
    $a'$ abs.exploration_flag.add(ineffective)
  end if
  end if
  Update graph $K.G$ with transition $\left(s'^{abs}, a'^{abs}, s^{abs}\right)$
  Update graph $K.G$ with transition $\left(s'^{abs}, a'^{abs}, s^{abs}\right)$
  return Updated knowledge $K$
```

return Updated knowledge K

end function

end function

Algorithm 2 Exploration Policy of LLM-Explorer.

算法 2 LLM-Explorer 的探索策略。

Input: App under test, the device for testing

输入: 待测应用，测试设备

Output: Exploration knowledge and log

输出: 探索知识和日志

function EXPLORE-MAIN(app):

function EXPLORE-MAIN(app):

$s_i \leftarrow$ Observe GUI state of the app $\triangleright$ Get initial state

$s_i \leftarrow$ 观察应用 $\triangleright$ 的 GUI 状态，获取初始状态

$K \leftarrow$ UpdateKnowledge $(\varnothing,$ null , null , $s_i)$ $\triangleright$ Initialize knowledge

$K \leftarrow$ UpdateKnowledge $(\varnothing,$ null , null , $s_i)$ $\triangleright$ 初始化知识

nav_steps $\leftarrow 0$ $\triangleright$ Initialize navigation steps

nav_steps $\leftarrow 0$ $\triangleright$ 初始化导航步骤

while K.unexplored_abstract_actions $\neq \varnothing$ do

while K.unexplored_abstract_actions $\neq \varnothing$ do

if nav_steps $\neq \varnothing$ then $\triangleright$ Try navigation

if nav_steps $\neq \varnothing$ then $\triangleright$ 尝试导航

$a_i \leftarrow$ pop next action from nav_steps

```
    $a_i \leftarrow$ 从 nav_steps 中弹出下一个动作
  else - Try exploration
  else - 尝试探索
    $a_i \leftarrow$ SelectExploreAction $(K, s_i)$
    $a_i \leftarrow$ 选择探索动作 $(K, s_i)$
    if $a_i \notin s_i$ .available_actions then ▷ Switch to navigation
    如果 $a_i \notin s_i$ .available_actions 则 ▷ 切换到导航
      nav_steps $\leftarrow$ FindNavigatePath $(K, s_i, a_i)$
      nav_steps $\leftarrow$ 查找导航路径 (FindNavigatePath) $(K, s_i, a_i)$
      $a_i \leftarrow$ pop next action from nav_steps
      $a_i \leftarrow$ 从 nav_steps 弹出下一个动作
    end if
    结束如果
  end if
  结束如果
  if $a_i$ is a text input action then
  如果 $a_i$ 是文本输入动作则
    $a_i \leftarrow$ GenerateInputText $(s_i, a_i)$ ▷ LLM assisted
    $a_i \leftarrow$ 生成输入文本 (GenerateInputText) $(s_i, a_i)$ ▷ 由大语言模型辅助
  end if
  结束如果
  Perform action $a_i$ on the device
  在设备上执行动作 $a_i$
  Observe new state $s_{i+1}$
```

```
            观察新状态 $s_{i+1}$
             $K \leftarrow$ UpdateKnowledge $(K, s_i, a_i, s_{i+1})$
            $K \leftarrow$ 更新知识 (UpdateKnowledge) $(K, s_i, a_i, s_{i+1})$
             if $a_i$ is navigation and $s_{i+1}$ doesn’t match nav_steps then
            如果 $a_i$ 是导航且 $s_{i+1}$ 与 nav_steps 不匹配则
               nav_steps $\leftarrow$ UpdateNavigatePath $(K,$ nav_steps $)$
              nav_steps $\leftarrow$ 更新导航路径 (UpdateNavigatePath) $(K,$ nav_steps $)$
             end if
            结束如果
        end while
       结束循环
      end function
     结束函数
```

Using its high-quality app knowledge that encompasses all UI actions, LLM-Explorer is able to choose from all the UI actions encountered rather than being restricted to the actions available in the current UI state. LLM-Explorer starts by going through the abstract actions in app knowledge, pulling out actions available in the current UI state that are labeled as unexplored. This creates a set of UI actions, among which it randomly picks one as the next action to explore. If no unexplored actions are found in the current state, LLM-Explorer widens its search to all UI states, gathering all the unexplored elements and action types to form an action pool, from which it randomly selects the next action for exploration. Compared to the LLM agents that select actions only from the current UI state [21, 39], this strategy saves LLM queries because it does not query LLM when navigating to a specific UI state.

利用其涵盖所有 UI 操作的高质量应用知识，LLM-Explorer 能够从遇到的所有 UI 操作中进行选择，而不局限于当前 UI 状态下可用的操作。LLM-Explorer 首先遍历应用知识中的抽象操作，提取当前 UI 状态下标记为未探索的可用操作。这会生成一组 UI 操作，从中随机选择一个作为下一步探索的操作。如果在当前状态下未发现未探索的操作，LLM-Explorer 会将搜索范围扩大到所有 UI 状态，收集所有未探索的元素和操作类型，形成一个操作池，从中随机选择下一步探索的操作。与仅从当前 UI 状态选择操作的 LLM 代理 [21, 39] 相比，该策略节省了 LLM 查询，因为在导航到特定 UI 状态时不需要查询 LLM。

3.3.3 Fault-tolerant Navigating Path Finder. The exploration agent of LLM-Explorer should possess the capability to navigate to a specific UI state, especially when the required UI action is not available in the current state. This can be achieved by utilizing the abstract UI interaction graph, where the exploration agent can select the shortest path from the current to the target UI state where the UI action is available. In this graph, edges represent UI actions, allowing for sequential navigation. Should the shortest path fails, the system will attempt alternative paths for a limited number of times.

> 3.3.3 容错导航路径查找器。LLM-Explorer 的探索代理应具备导航到特定 UI 状态的能力，尤其是在当前状态下所需 UI 操作不可用时。通过利用抽象 UI 交互图，探索代理可以选择从当前状态到目标 UI 状态 (该状态下操作可用) 的最短路径。在该图中，边表示 UI 操作，支持顺序导航。如果最短路径失败，系统将尝试有限次数的替代路径。

However, due to unpredictable app behavior, the target abstract UI state might sometimes be unreachable. In these cases, LLM-Explorer will restart the app and attempt to navigate from the initial state to the target UI state again. If this approach also fails, the navigation attempt will be deemed as failed, and the corresponding action will be removed from the interaction graph so that future navigations can avoid generating infeasible paths.

> 然而，由于应用行为不可预测，目标抽象 UI 状态有时可能无法到达。在这种情况下，LLM-Explorer 会重启应用并尝试从初始状态重新导航到目标 UI 状态。如果此方法仍失败，则导航尝试被视为失败，并将相应操作从交互图中移除，以避免未来导航生成不可行路径。

3.3.4 Content-aware Input Text Generator. Generating human-like input text for text boxes is complex, as it requires

> 3.3.4 内容感知输入文本生成器。为文本框生成类人输入文本较为复杂，因为这需要

understanding how to use the mobile apps. Thus, we rely on LLMs for this task. LLM-Explorer generates a structured prompt based on the name of the app under test and the current state information, and then queries LLM to obtain the input text. An example is shown in Figure 3.

> 理解如何使用移动应用。因此，我们依赖 LLM 来完成此任务。LLM-Explorer 基于被测应用名称和当前状态信息生成结构化提示，然后查询 LLM 以获取输入文本。示例见图 3。

3.4 Detailed Usage of LLM

3.4 LLM 的详细使用

In this section, we further clarify the usage of LLM in our approach and analyze the cost. LLM-Explorer involves LLMs in two parts, including knowledge organization and text input generation.

> 本节进一步阐明我们方法中 LLM 的使用方式并分析其成本。LLM-Explorer 在两个部分涉及 LLM，包括知识组织和文本输入生成。

Using LLM for Knowledge Organization. The LLM is firstly utilized to categorize a group of similar UI actions, grouping them into an abstract UI action for app knowledge management, as detailed in Section 3.2.2. The structure of the prompt is illustrated in Figure 2, which is composed of four modules. It begins with an introduction including the testing app name and an explanation of the purpose of the prompt. Next, the UI is represented in the HTML syntax in line with previous studies [29]. A chain-of-thought prompting module [33] is also included to encourage LLMs' logical reasoning. The output format is required to be a json dict for straightforward parsing. LLMs are expected to generate a merging instruction for UI elements, with the aim of consolidating elements with the same functions into a single UI element. This approach simplifies the action space by combining actions on these elements, which have similar functionality, into a single abstract UI action.

利用 LLM 进行知识组织。LLM 首先用于对一组相似的 UI 操作进行分类，将它们归为一个抽象 UI 操作以便应用知识管理，详见 3.2.2 节。提示结构如图 2 所示，由四个模块组成。首先是介绍部分，包括测试应用名称和提示目的说明。接着，UI 以符合先前研究 [29] 的 HTML 语法表示。还包含链式思维提示模块 [33]，以促进 LLM 的逻辑推理。输出格式要求为 json 字典，便于解析。LLM 需生成 UI 元素合并指令，旨在将功能相同的元素合并为单一 UI 元素。此方法通过将对这些功能相似元素的操作合并为单一抽象 UI 操作，简化了操作空间。

Using LLM for Content-Aware Input Generation. LLM-Explorer also employs the LLM to generate text inputs for UI elements, such as input boxes, based on the current UI context. Figure 3 illustrates the prompt structure. It provides the information about the current UI state and the specific input box to be filled and requests LLMs to generate the input text. We observe that LLMs could generate human-like input data, such as usernames, phone numbers, email addresses, etc.

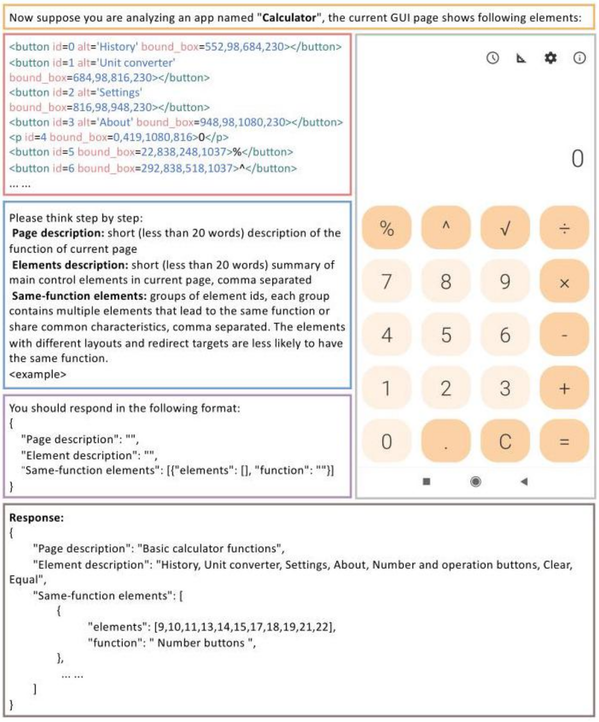
利用 LLM 进行内容感知输入生成。LLM-Explorer 还利用 LLM 根据当前 UI 上下文为 UI 元素 (如输入框) 生成文本输入。图 3 展示了提示结构。它提供当前 UI 状态和需填写的具体输入框信息，并请求 LLM 生成输入文本。我们观察到 LLM 能够生成类人输入数据，如用户名、电话号码、电子邮件地址等。

These two usages of LLM are uneasy for traditional AI techniques since it involves few-shot understanding of diverse UI content and free-form text generation.

这两种 LLM 的使用对传统 AI 技术来说较为困难，因为它涉及对多样 UI 内容的少量示例理解和自由形式文本生成。

According to the usages, the LLM queries of LLM-Explorer are determined by the number of unique states/actions (in knowledge organization) and the number of text input boxes (in input generation). These numbers are bounded for most of the apps, although there might be a few exceptions with very dynamic GUI. Meanwhile, these numbers are guaranteed to be smaller than the number of steps, since each unique state/action or input box maps to at least one step in the exploration. Therefore, LLM-Explorer is expected to have much slower token consumption than existing approaches that use LLM for action generation.

根据使用情况，LLM-Explorer 的 LLM 查询次数由唯一状态/操作数量 (知识组织中) 和文本输入框数量 (输入生成中) 决定。对于大多数应用，这些数量是有限的，尽管某些动态 GUI 应用可能有例外。同时，这些数量保证小于步骤数，因为每个唯一状态/操作或输入框至少对应探索中的一步。因此，LLM-Explorer 的令牌消耗预计远低于现有使用 LLM 进行操作生成的方法。



Now suppose you are analyzing an app named "**Calculator**", the current GUI page shows following elements:

```
<button id=0 alt='History' bound_box=552,98,684,230></button>
<button id=1 alt='Unit converter'
bound_box=684,98,816,230></button>
<button id=2 alt='Settings'
bound_box=816,98,948,230></button>
<button id=3 alt='About' bound_box=948,98,1080,230></button>
<p id=4 bound_box=0,419,1080,816>0</p>
<button id=5 bound_box=22,838,248,1037>%</button>
<button id=6 bound_box=292,838,518,1037>^</button>
... ...
```

Please think step by step:
Page description: short (less than 20 words) description of the function of current page
Elements description: short (less than 20 words) summary of main control elements in current page, comma separated
Same-function elements: groups of element ids, each group contains multiple elements that lead to the same function or share common characteristics, comma separated. The elements with different layouts and redirect targets are less likely to have the same function.
<example>

You should respond in the following format:

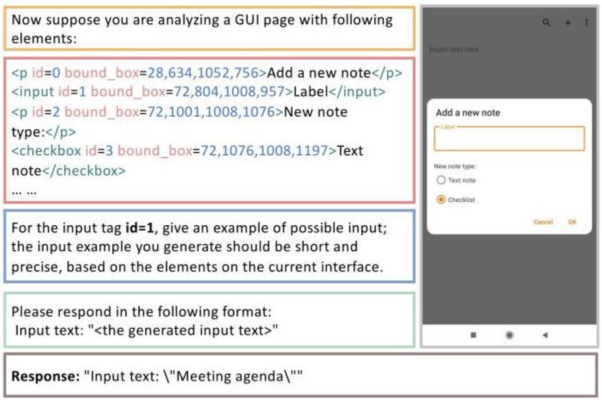
```
{
  "Page description": "",
  "Element description": "",
  "Same-function elements": [{"elements": [], "function": ""}]
}
```

Response:

```
{
     "Page description": "Basic calculator functions",
     "Element description": "History, Unit converter, Settings, About, Number and operation buttons, Clear,
Equal",
     "Same-function elements": [
          {
               "elements": [9,10,11,13,14,15,17,18,19,21,22],
               "function": " Number buttons ",
          },
           ... ...
     ]
}
```

Figure 2: The prompt for knowledge organization. From top to bottom: general instructions, UI representation, chain-of-thought module, output format, and response, respectively.

图 2: 知识组织的提示。从上到下依次为: 通用说明、UI 表示、链式思维模块、输出格式和响应。



Now suppose you are analyzing a GUI page with following elements:

```
<p id=0 bound_box=28,634,1052,756>Add a new note</p>
<input id=1 bound_box=72,804,1008,957>Label</input>
<p id=2 bound_box=72,1001,1008,1076>New note
type:</p>
<checkbox id=3 bound_box=72,1076,1008,1197>Text
note</checkbox>
... ...
```

For the input tag **id=1**, give an example of possible input; the input example you generate should be short and precise, based on the elements on the current interface.

Please respond in the following format:
Input text: "<the generated input text>"

Response: "Input text: \"Meeting agenda\""

Figure 3: The prompt for content-aware input generation.

图 3: 内容感知输入生成的提示。

From top to bottom, the prompt comprises: overall guidance, page representation, input request, response format, and response, respectively.

从上到下，提示依次包括: 整体指导、页面表示、输入请求、响应格式和响应。

4 EVALUATION

4 评估

We implemented LLM-Explorer with Python and Java atop DroidBot [16] and GPT-3.5 (specific version: gpt-3.5-turbo- 1106), and we conducted experiments on real app exploration tasks to evaluate its performance. In total, the experiments

我们基于 DroidBot [16] 和 GPT-3.5(具体版本:gpt-3.5-turbo-1106) 使用 Python 和 Java 实现了 LLM-Explorer，并在真实应用探索任务上进行了实验以评估其性能。总共，实验

cost about $1,000 for GPT services, including both GPT-3.5 and GPT-4 usage, covering LLM-Explorer and all baselines.

花费了约 1000 美元的 GPT 服务费用，涵盖了 LLM-Explorer 及所有基线方法中 GPT-3.5 和 GPT-4 的使用。

4.1 Experimental Setup

4.1 实验设置

Benchmark Apps. We mainly evaluated LLM-Explorer on 20 apps collected from F-Droid and Google Play. The apps used in our study were selected based on the following rules: 1. Apps were chosen from the most common categories in app stores, with 1-3 apps selected per category. 2. Open-source versions of apps were prioritized where available. The sources and complexity of the apps is shown in Table 1. We didn't include more apps for analysis due to the expensive cost of LLM services. However, we believe these apps are representative enough since they cover the most common app functionalities including contact management, messaging, photography, playing music, email management, and so on.

基准应用。我们主要在从 F-Droid 和 Google Play 收集的 20 个应用上评估 LLM-Explorer。研究中使用的应用选择遵循以下规则:1. 从应用商店中最常见的类别中选择，每个类别选取 1-3 个应用。2. 优先选择开源版本的应用 (如有)。应用的来源和复杂度见表 1。由于 LLM 服务费用较高，我们未包含更多应用进行分析。但我们认为这些应用具有代表性，因为它们涵盖了联系人管理、消息传递、摄影、音乐播放、邮件管理等最常见的应用功能。

Hardware. We evaluate the end-to-end performance of LLM-Explorer on real Android devices and emulators with Android 10. The exploration agents run on a desktop with 2 NVIDIA GeForce RTX 3090 GPUs and 48GB memory.

硬件。我们在搭载 Android 10 的真实 Android 设备和模拟器上评估 LLM-Explorer 的端到端性能。探索代理运行在配备 2 块 NVIDIA GeForce RTX 3090 GPU 和 48GB 内存的台式机上。

Baselines. We chose DroidAgent [39], GPTDroid [21], Humanoid [17], DroidBot [16], and Monkey [7] as our baselines. DroidAgent and GPTDroid are LLM-based automated exploration methods. DroidAgent uses LLMs (GPT-4 and GPT-3.5) to plan tasks, observe the page, generate actions, determine task completion, etc. GPTDroid uses an LLM to choose the next action based on the current UI state. Humanoid uses a deep neural network to generate human-like UI actions during exploration. DroidBot is a generic app testing framework, in which we used the default depth-first traversal policy to explore the apps. Monkey is a popular and official command-line tool for automated app fuzz testing. Note that since the source code of GPTDroid is not fully usable at the time of our experiments, we use DroidAgent's reimple-mentation of GPTDroid, which uses a function-call-based action selector instead of the matching network in GPTDroid. We also use GPT-3.5 instead of GPT-3 in GPTDroid. We also compared other variations of LLM-Explorer based on different LLMs (Vicuna-13B and GPT-4 (specific version: gpt-4-1106-preview) in Section 4.4.

> 基线方法。我们选择了 DroidAgent [39]、GPTDroid [21]、Humanoid [17]、DroidBot [16] 和 Monkey [7] 作为基线。DroidAgent 和 GPTDroid 是基于 LLM 的自动探索方法。DroidAgent 使用 LLM(GPT-4 和 GPT-3.5) 进行任务规划、页面观察、动作生成、任务完成判定等。GPTDroid 基于当前 UI 状态使用 LLM 选择下一步动作。Humanoid 使用深度神经网络在探索过程中生成类人 UI 动作。DroidBot 是通用的应用测试框架，我们使用其默认的深度优先遍历策略进行应用探索。Monkey 是流行且官方的命令行自动化应用模糊测试工具。注意，由于实验时 GPTDroid 的源码尚不可完全使用，我们采用了 DroidAgent 对 GPTDroid 的重新实现，使用基于函数调用的动作选择器替代 GPTDroid 中的匹配网络。我们还在 GPTDroid 中使用了 GPT-3.5 替代 GPT-3。我们还在第 4.4 节比较了基于不同 LLM(Vicuna-13B 和 GPT-4(具体版本:gpt-4-1106-preview)) 的 LLM-Explorer 变体。

Reference Human Performance. Since certain app activities lack predefined navigation logic set by developers, the upper limit of activity coverage during exploration may not reach 100%. Therefore, to better understand the performance of the automated explorers, we included human performance as a reference for the upper bound of explorable activities. The human performance is collected with a user study approved by our Institutional Review Board (IRB) with the research question "How many activities of apps can human users reach?". We invited 6 experienced smartphone users to explore the 20 apps in Table 2 using our lab device. Participants were selected based on their proficiency with computers and smart-phones, with priority given to those with a background in computer science. They were all from our campus, had at least five years of smartphone experience, and possessed foundational knowledge in computer science. The number of participants is determined by how many times each benchmark app can be explored. While we acknowledge the limited scale of the user study, we emphasize that six participants sufficed to explore each app twice. This provides adequate exploration coverage per app. Participants were asked to explore all the pages and elements with different functions within the apps. The exploration time for each app was required to be no less than 10 minutes and the exploration of an app ended when the participant felt there was no more to explore. During the process, our backend system automatically recorded the activities explored by the participants. In the end, each app was explored for more than twice, and we took the union of explored states as the final result. To the best of our knowledge, this is also the first time in the community to compare against reference human performance in app exploration.

参考人类表现。由于某些应用活动缺乏开发者预设的导航逻辑，探索过程中活动覆盖率的上限可能达不到 100%。因此，为了更好地理解自动探索器的性能，我们将人类表现作为可探索活动上限的参考。人类表现通过经机构审查委员会 (IRB) 批准的用户研究收集，研究问题为“人类用户能达到多少应用活动?”。我们邀请了 6 名经验丰富的智能手机用户使用实验室设备探索表 2 中的 20 个应用。参与者根据其计算机和智能手机使用熟练度选取，优先考虑计算机科学背景者。所有参与者均来自本校，拥有至少五年智能手机使用经验及计算机科学基础知识。参与者数量由每个基准应用可被探索的次数决定。尽管用户研究规模有限，我们强调六名参与者足以对每个应用进行两次探索，保证了每个应用的充分探索覆盖。参与者被要求探索应用内所有页面及不同功能元素。每个应用的探索时间不少于 10 分钟，且当参与者认为无更多内容可探索时结束。过程中，后台系统自动记录参与者探索的活动。最终，每个应用均被探索超过两次，我们取探索状态的并集作为最终结果。据我们所知，这是社区首次将自动应用探索与人类参考表现进行比较。

Metrics. Similar to most existing work on mobile app exploration, we test our method and the baselines by comparing the progressive coverage, i.e. the improvement of coverage over time and the achieved final coverage. Since measuring the source code coverage is difficult for compiled APKs, we opt for monitoring the activity coverage (i.e. the ratio of Activities reached by the agent among all Activities defined in the app), which is also a common practice.

指标。与大多数现有的移动应用探索工作类似，我们通过比较渐进覆盖率，即覆盖率随时间的提升以及最终达到的覆盖率，来测试我们的方法和基线方法。由于对已编译 APK 测量源代码覆盖率较为困难，我们选择监控活动覆盖率 (即代理触达的活动数与应用中定义的所有活动数的比率)，这也是一种常见做法。

4.2 Exploration Effectiveness and Efficiency

4.2 探索效果与效率

To evaluate the effectiveness and efficiency of LLM-Explorer, we test 20 popular apps with LLM-Explorer and the five baselines. To ensure fair comparisons, each method was given a fixed exploration time of 2 hours.

为了评估 LLM-Explorer 的效果与效率，我们对 20 款热门应用分别使用 LLM-Explorer 及五个基线方法进行测试。为确保公平比较，每种方法均被赋予固定的 2 小时探索时间。

Table 2 presents the final activity coverage achieved by each method. The results show that LLM-Explorer is able to achieve a higher activity coverage than all baseline methods. While there is still some gap to human performance, it can attain a similar or even higher level of coverage than human exploration on some simple apps.

表 2 展示了各方法最终达到的活动覆盖率。结果表明，LLM-Explorer 能够实现比所有基线方法更高的活动覆盖率。虽然与人工表现仍有一定差距，但在一些简单应用上，其覆盖率可达到甚至超过人工探索水平。

Figure 4 and Figure 5 illustrate the progressive activity coverage of LLM-Explorer and baselines over time and over the number of steps, respectively.

图 4 和图 5 分别展示了 LLM-Explorer 及基线方法随时间和步骤数的渐进活动覆盖率。

As shown in Figure 4, at the beginning of exploration, LLM-Explorer, Humanoid, and DroidBot achieved similar coverage growth rates, which were significantly higher than DroidAgent, GPTDroid, and Monkey. This is because they could send actions at a faster speed and the actions could easily reach new activities in the early stage. Gradually, the growth rates slowed down as discovering new activities became harder, while LLM-Explorer and DroidAgent began

如图 4 所示，探索初期，LLM-Explorer、Humanoid 和 DroidBot 的覆盖率增长速度相近，且显著高于 DroidAgent、GPTDroid 和 Monkey。这是因为前者能够更快地发送操作，且操作在早期更容易触达新活动。随着新活动发现难度增加，增长速度逐渐放缓，而 LLM-Explorer 和 DroidAgent 开始

Table 1: Information about the benchmark apps

表 1: 基准应用信息

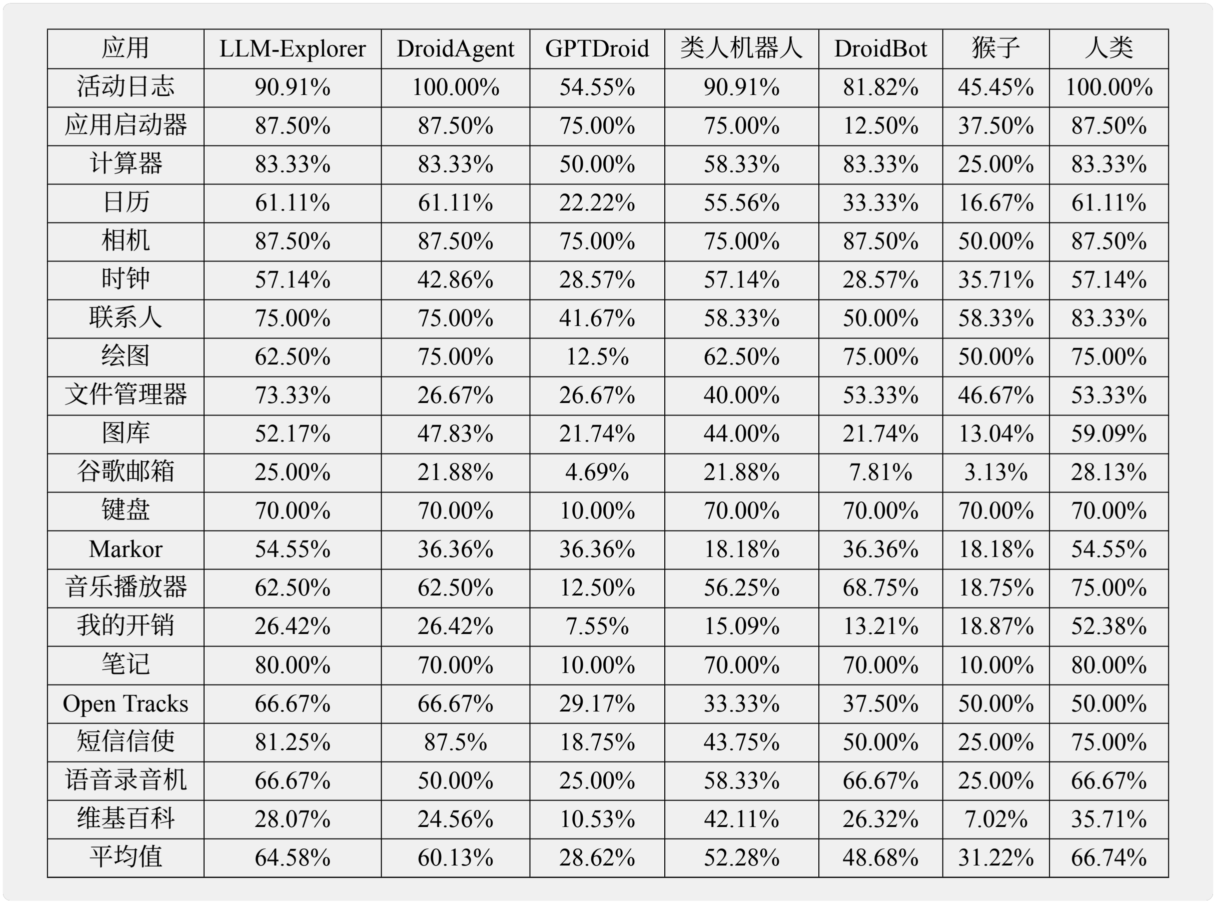
App Name	From	Activity Count	Complexity	App Name	From	Activity Count	Complexity
Activity Diary	F-Droid	11	Medium	App Launcher	F-Droid	8	Low
Calculator	F-Droid	12	Medium	Calendar	F-Droid	18	Medium
Camera	F-Droid	8	Low	Clock	F-Droid	14	Medium
Contacts	F-Droid	12	Medium	Draw	F-Droid	8	Low
File Manager	F-Droid	15	Medium	Gallery	F-Droid	23	High
Google Mail	Google Play	64	High	Keyboard	F-Droid	10	Low
Markor	F-Droid	11	Medium	Music Player	F-Droid	16	Medium
My Expenses	F-Droid	53	High	Notes	F-Droid	10	Low
Open Tracks	F-Droid	24	High	SMS Messenger	F-Droid	16	Medium
Voice Recorder	F-Droid	12	Medium	Wikipedia	F-Droid	57	High

应用名称	来自	活动计数	复杂度	应用名称	来自	活动计数	复杂度
活动日志	F-Droid	11	中等	应用启动器	F-Droid	8	低
计算器	F-Droid	12	中等	日历	F-Droid	18	中等
相机	F-Droid	8	低	时钟	F-Droid	14	中等
联系人	F-Droid	12	中等	绘图	F-Droid	8	低
文件管理器	F-Droid	15	中等	图库	F-Droid	23	高
谷歌邮箱	谷歌商店	64	高	键盘	F-Droid	10	低
Markor	F-Droid	11	中等	音乐播放器	F-Droid	16	中等
我的开销	F-Droid	53	高	笔记	F-Droid	10	低
Open Tracks	F-Droid	24	高	短信信使	F-Droid	16	中等
录音机	F-Droid	12	中等	维基百科	F-Droid	57	高

Table 2: The activity coverage achieved by LLM-Explorer and baselines on 20 typical apps.

表 2:LLM-Explorer 及基线方法在 20 个典型应用上的活动覆盖率。

App	LLM-Explorer	DroidAgent	GPTDroid	Humanoid	DroidBot	Monkey	Human
Activity Diary	90.91%	100.00%	54.55%	90.91%	81.82%	45.45%	100.00%
App Launcher	87.50%	87.50%	75.00%	75.00%	12.50%	37.50%	87.50%
Calculator	83.33%	83.33%	50.00%	58.33%	83.33%	25.00%	83.33%
Calendar	61.11%	61.11%	22.22%	55.56%	33.33%	16.67%	61.11%
Camera	87.50%	87.50%	75.00%	75.00%	87.50%	50.00%	87.50%
Clock	57.14%	42.86%	28.57%	57.14%	28.57%	35.71%	57.14%
Contacts	75.00%	75.00%	41.67%	58.33%	50.00%	58.33%	83.33%
Draw	62.50%	75.00%	12.5%	62.50%	75.00%	50.00%	75.00%
File Manager	73.33%	26.67%	26.67%	40.00%	53.33%	46.67%	53.33%
Gallery	52.17%	47.83%	21.74%	44.00%	21.74%	13.04%	59.09%
Google Mail	25.00%	21.88%	4.69%	21.88%	7.81%	3.13%	28.13%
Keyboard	70.00%	70.00%	10.00%	70.00%	70.00%	70.00%	70.00%
Markor	54.55%	36.36%	36.36%	18.18%	36.36%	18.18%	54.55%
Music Player	62.50%	62.50%	12.50%	56.25%	68.75%	18.75%	75.00%
My Expenses	26.42%	26.42%	7.55%	15.09%	13.21%	18.87%	52.38%
Notes	80.00%	70.00%	10.00%	70.00%	70.00%	10.00%	80.00%
Open Tracks	66.67%	66.67%	29.17%	33.33%	37.50%	50.00%	50.00%
SMS Messenger	81.25%	87.5%	18.75%	43.75%	50.00%	25.00%	75.00%
Voice Recorder	66.67%	50.00%	25.00%	58.33%	66.67%	25.00%	66.67%
Wikipedia	28.07%	24.56%	10.53%	42.11%	26.32%	7.02%	35.71%
Average	64.58%	60.13%	28.62%	52.28%	48.68%	31.22%	66.74%



应用	LLM-Explorer	DroidAgent	GPTDroid	类人机器人	DroidBot	猴子	人类
活动日志	90.91%	100.00%	54.55%	90.91%	81.82%	45.45%	100.00%
应用启动器	87.50%	87.50%	75.00%	75.00%	12.50%	37.50%	87.50%
计算器	83.33%	83.33%	50.00%	58.33%	83.33%	25.00%	83.33%
日历	61.11%	61.11%	22.22%	55.56%	33.33%	16.67%	61.11%
相机	87.50%	87.50%	75.00%	75.00%	87.50%	50.00%	87.50%
时钟	57.14%	42.86%	28.57%	57.14%	28.57%	35.71%	57.14%
联系人	75.00%	75.00%	41.67%	58.33%	50.00%	58.33%	83.33%
绘图	62.50%	75.00%	12.5%	62.50%	75.00%	50.00%	75.00%
文件管理器	73.33%	26.67%	26.67%	40.00%	53.33%	46.67%	53.33%
图库	52.17%	47.83%	21.74%	44.00%	21.74%	13.04%	59.09%
谷歌邮箱	25.00%	21.88%	4.69%	21.88%	7.81%	3.13%	28.13%
键盘	70.00%	70.00%	10.00%	70.00%	70.00%	70.00%	70.00%
Markor	54.55%	36.36%	36.36%	18.18%	36.36%	18.18%	54.55%
音乐播放器	62.50%	62.50%	12.50%	56.25%	68.75%	18.75%	75.00%
我的开销	26.42%	26.42%	7.55%	15.09%	13.21%	18.87%	52.38%
笔记	80.00%	70.00%	10.00%	70.00%	70.00%	10.00%	80.00%
Open Tracks	66.67%	66.67%	29.17%	33.33%	37.50%	50.00%	50.00%
短信信使	81.25%	87.5%	18.75%	43.75%	50.00%	25.00%	75.00%
语音录音机	66.67%	50.00%	25.00%	58.33%	66.67%	25.00%	66.67%
维基百科	28.07%	24.56%	10.53%	42.11%	26.32%	7.02%	35.71%
平均值	64.58%	60.13%	28.62%	52.28%	48.68%	31.22%	66.74%

to have higher growth rates than the others with the help of LLMs. Eventually, LLM-Explorer was able to achieve the highest activity coverage.

在大语言模型 (LLMs) 的帮助下，其增长率高于其他方法。最终，LLM-Explorer 实现了最高的活动覆盖率。

In Figure 5, the coverage of LLM-Explorer gradually converged at about 1500 steps, faster than most of the baselines. During exploration, although DroidAgent's per-step actions were more effective in contributing to activity coverage improvement, DroidAgent's each step was more time-consuming due to the fact that it requested LLM extensively for reasoning and planning, with the lowest number of actions generated in the 2-hour exploration. We also observed that the activity coverage growth rate of DroidAgent decreased

在图 5 中，LLM-Explorer 的覆盖率在约 1500 步时逐渐收敛，速度快于大多数基线方法。在探索过程中，尽管 DroidAgent 每步动作在提升活动覆盖率方面更有效，但由于其大量调用 LLM 进行推理和规划，每步耗时较长，导致在 2 小时探索中生成的动作数量最少。我们还观察到 DroidAgent 的活动覆盖率增长率在探索结束时随着步数增加而下降,

with the number of steps at the end of exploration, which we thought was because LLM-based action selection became less effective for discovering more detailed and unusual activities. In Table 3, we show the average single-step time of LLM-Explorer and baselines.

我们认为这是因为基于 LLM 的动作选择在发现更细节和异常活动时效果减弱。在表 3 中，我们展示了 LLM-Explorer 及基线方法的平均单步耗时。

The advantages of LLM-Explorer over the baselines could be attributed to the following reasons: (i) LLM-Explorer merges states with identical functionality into a single abstract state, thereby eliminating redundant exploration of states with the same functionality. This strategy allows the exploration process to focus on discovering more unknown features of the app under test. Our further analysis found that

LLM-Explorer 相较基线方法的优势可归因于以下原因:(i)LLM-Explorer 将功能相同的状态合并为单一抽象状态，从而消除了对功能相同状态的冗余探索。该策略使探索过程能够专注于发现被测应用的更多未知特性。我们的进一步分析发现,

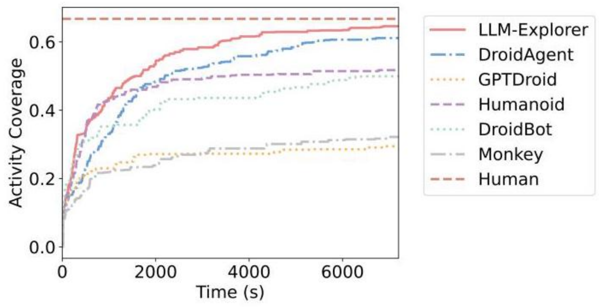


Figure 4: Progressive activity coverage of LLM-Explorer and baselines over time. The brown dotted line is the reference human performance.

图 4:LLM-Explorer 及基线方法随时间的活动覆盖率进展。棕色虚线为参考的人类表现。

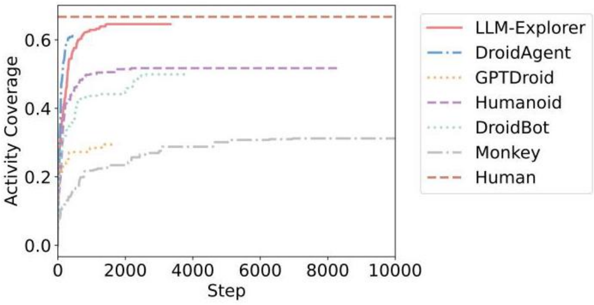


Figure 5: Progressive activity coverage of LLM-Explorer and baselines over steps within 2 hours. The brown dotted line is the reference human performance. The maximum step differs for each method due to the different per-step time. Note that Monkey produced over 20,000 steps in 2 hours of exploration, but the activity coverage nearly converged after 10,000 steps, so this figure only shows up to 10,000 steps.

图 5:LLM-Explorer 及基线方法在 2 小时内随步数的活动覆盖率进展。棕色虚线为参考的人类表现。由于每步耗时不同，各方法的最大步数不同。注意，Monkey 在 2 小时探索中产生了超过 20,000 步，但活动覆盖率在 10,000 步后几乎收敛，因此本图仅显示至 10,000 步。

Table 3: Average per-step time of LLM-Explorer and baselines. LE: LLM-Explorer, DA: DroidAgent, GD: GPTDroid, HU: Humanoid, DB: DroidBot, MO: Monkey.

表 3:LLM-Explorer 及基线方法的平均单步耗时。LE:LLM-Explorer，DA:DroidAgent，GD:GPTDroid，HU:Humanoid，DB:DroidBot，MO:Monkey。

Method	LE	DA	GD	HU	DB	MO
Time (seconds)	5.19	19.10	5.59	3.19	2.94	0.79

方法	LE	DA	GD	HU	DB	MO
时间 (秒)	5.19	19.10	5.59	3.19	2.94	0.79

each app has 128 abstract states on average. LLM-Explorer reduced the numbers of redundant state and action visits by 4.08 and 14.71 per abstract state, which effectively reduces the exploration space. (ii) LLM-Explorer performs exploration based on abstract actions, which allows a more comprehensive coverage of all functionalities within the current app, including those that are not primary features, such as viewing the list of contributors on the ”About” page of Notes app. (iii) LLM-Explorer’s LLM-assisted knowledge maintenance strategy of merging functionally identical elements avoids

每个应用平均有 128 个抽象状态。LLM-Explorer 将每个抽象状态的冗余状态访问次数和动作访问次数分别减少了 4.08 次和 14.71 次，有效缩小了探索空间。(ii) LLM-Explorer 基于抽象动作进行探索，这使得对当前应用内所有功能的覆盖更加全面，包括那些非主要功能，例如在 Notes 应用的“关于”页面查看贡献者列表。(iii) LLM-Explorer 利用大语言模型 (LLM) 辅助的知识维护策略，通过合并功能相同的元素避免了

Table 4: The average number of tokens and average cost consumed by LLM-Explorer, DroidAgent and GPTDroid when exploring an app. GPT-3.5, GPT-3.5-16K, and GPT-4 represent the tokens consumed when exploring an app using the corresponding model, respectively. Cost denotes the total cost of LLM when exploring an app.

表 4:LLM-Explorer、DroidAgent 和 GPTDroid 在探索应用时的平均令牌数和平均消耗成本。GPT-3.5、GPT-3.5-16K 和 GPT-4 分别表示使用对应模型探索应用时消耗的令牌数。成本指的是探索应用时大语言模型 (LLM) 的总消耗成本。

Method	GPT-3.5	GPT-3.5-16K	GPT-4	Cost($)
LLM-Explorer	96,884.2	0	0	0.11
DroidAgent	1,384,079.0	1,141,095.29	335,913.29	16.31
GPTDroid	993,939.35	0	0	1.07

方法	GPT-3.5	GPT-3.5-16K	GPT-4	费用 (美元)
LLM-Explorer	96,884.2	0	0	0.11
DroidAgent	1,384,079.0	1,141,095.29	335,913.29	16.31
GPTDroid	993,939.35	0	0	1.07

LLM-Explorer repeatedly exploring duplicate elements with the same functionality. For example, within the ”file import” page of the File Manager app, LLM-Explorer can combine the file name buttons on the page into the same abstract element. This makes it possible to explore the page by importing a file only once, thereby accomplishing the exploration of the file import functionality.

LLM-Explorer 反复探索具有相同功能的重复元素。例如，在文件管理器应用的“文件导入”页面中，LLM-Explorer 可以将页面上的文件名按钮合并为同一抽象元素。这使得只需导入一次文件即可探索该页面，从而完成文件导入功能的探索。

We further analyzed why LLM-Explorer failed in some cases: (i) LLM-Explorer merged different states into an abstract state based on rules, but if there was a unique element in the state, LLM-Explorer could not efficiently merge the state with the previous state. We also tried using LLM to merge states, but the result was not satisfactory, so we didn’t adopt this solution in the end. (ii) In LLM-assisted knowledge maintenance, LLM may incorrectly treat elements with different functions as having the same function, resulting in the merging of elements that could reach different activities into one abstract element, causing LLM-Explorer to miss some activities when exploring. Examples are given in section 4.5.

我们进一步分析了 LLM-Explorer 在某些情况下失败的原因:(i) LLM-Explorer 基于规则将不同状态合并为抽象状态，但如果状态中存在唯一元素，LLM-Explorer 无法高效地将该状态与之前的状态合并。我们也尝试使用 LLM 合并状态，但结果不理想，因此最终未采纳该方案。(ii) 在 LLM 辅助的知识维护中，LLM 可能错误地将功能不同的元素视为具有相同功能，导致能够到达不同活动的元素被合并为一个抽象元素，造成 LLM-Explorer 在探索时遗漏某些活动。相关示例见第 4.5 节。

4.3 LLM Cost of Exploration

4.3 LLM 探索成本

LLM-Explorer uses the LLM in two scenarios including knowledge maintenance and input text generation. GPTDroid and DroidAgent use LLMs mainly for action generation, and also for planning, observing, etc. We compare the number of tokens and API fees consumed by different methods during the exploration process. Table 4 shows the average cost of using LLM to explore an app. As compared to DroidA-gent (which achieved comparative coverage as ours), LLM-Explorer significantly reduced the average cost of exploring an app from $16.31 to $0.11, which was over 148 times less.

LLM-Explorer 在知识维护和输入文本生成两个场景中使用 LLM。GPTDroid 和 DroidAgent 主要使用 LLM 进行动作生成，同时也用于规划、观察等。我们比较了不同方法在探索过程中的令牌数量和 API 费用。表 4 展示了使用 LLM 探索应用的平均成本。与实现了相当覆盖率的 DroidAgent 相比，LLM-Explorer 显著将探索应用的平均成本从 16.31 美元降低到 0.11 美元，减少了超过 148 倍。

We also compared the number of query and the number of query tokens during exploration for different methods, as shown in Table 5. LLM-Explorer reduces the number of query by 18.48 and 9.40 times compared to DroidAgent and GPTDroid respectively. At the same time, LLM-Explorer used fewer input tokens while obtaining approximately twice the number of output tokens compared to other methods, and also had the lowest average token count per query.

我们还比较了不同方法在探索过程中的查询次数和查询令牌数，如表 5 所示。LLM-Explorer 相比 DroidAgent 和 GPTDroid 分别减少了 18.48 倍和 9.40 倍的查询次数。同时，LLM-Explorer 在使用更少输入令牌的情况下，获得了约两倍于其他方法的输出令牌数，并且每次查询的平均令牌数最低。

Table 5: Number of queries and average number of query tokens by LLM-Explorer, DroidAgent and GPT-Droid.

表 5:LLM-Explorer、DroidAgent 和 GPTDroid 的查询次数及平均查询令牌数。

Method	LLM-Explorer	DroidAgent	GPTDroid
Number of queries	157.12	2903.15	1476.51
Input tokens per query	507.15	933.71	623.00
Output tokens per query	109.47	51.80	50.17

方法	LLM-Explorer	DroidAgent	GPTDroid
查询次数	157.12	2903.15	1476.51
每次查询的输入标记数	507.15	933.71	623.00
每次查询的输出标记数	109.47	51.80	50.17

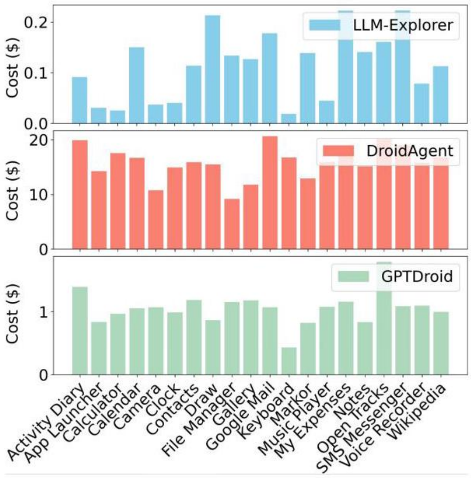


Figure 6: The cost of exploration in all apps under test by LLM-Explorer, DroidAgent, GPTDroid, respectively.

图 6:LLM-Explorer、DroidAgent、GPTDroid 在所有测试应用中的探索成本。

To understand the relation between the cost of the exploration process and the complexity of the apps, we analyzed the exploration cost of LLM-Explorer, DroidAgent and GPTDroid on different apps. As shown in Figure 6, the cost of LLM-Explorer is more adaptive for different apps, with lower cost on the apps with smaller number of activities (App Launcher, Keybord, etc.) while higher cost on more complex apps (My Expenses, SMS Messenger, etc.). However, DroidAgent and GPTDroid do not clearly show this adaptivity, and won't significantly reduce costs when exploring simpler apps. Meanwhile, the costs increased linearly with the step count in DroidAgent and GPTDroid, while the increase of LLM-Explorer was sublinear, as shown in Figure 7. Such adaptivity and sublinearity come from the way LLM-Explorer uses LLM, which has been discussed in Section 3.4.

为了理解探索过程的成本与应用复杂度之间的关系, 我们分析了 LLM-Explorer、DroidAgent 和 GPT-Droid 在不同应用上的探索成本。如图 6 所示，LLM-Explorer 的成本对不同应用表现出更强的适应性，在活动数量较少的应用 (如应用启动器、键盘等) 上成本较低，而在更复杂的应用 (如我的支出、短信信使等) 上成本较高。然而，DroidAgent 和 GPTDroid 未表现出明显的适应性，在探索较简单应用时成本不会显著降低。同时，如图 7 所示，DroidAgent 和 GPTDroid 的成本随着步骤数线性增加，而 LLM-Explorer 的成本增长呈亚线性。这种适应性和亚线性增长源于 LLM-Explorer 使用大型语言模型 (LLM) 的方式，相关内容已在第 3.4 节讨论。

4.4 Model Variation Test

4.4 模型变异测试

Given the pivotal role of the LLM (GPT-3.5) in the exploration policy of LLM-Explorer, we evaluate its influence by replacing it with a weaker and a stronger variant. The results are shown in Figure 8.

鉴于大型语言模型 (LLM)(GPT-3.5) 在 LLM-Explorer 探索策略中的关键作用，我们通过替换为较弱和较强的变体来评估其影响。结果如图 8 所示。

Using a Smaller Local LLM. We first tested the performance of LLM-Explorer with Vicuna-13B [5]. We quantized

使用更小的本地大型语言模型 (LLM)。我们首先测试了 LLM-Explorer 与 Vicuna-13B [5] 的性能。我们进行了量化

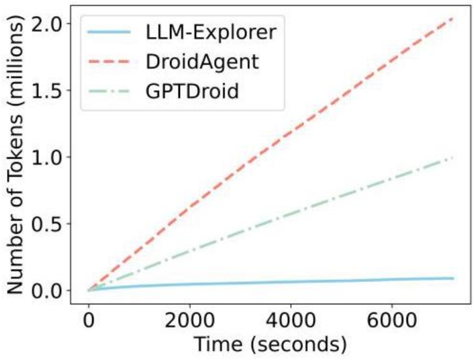


Figure 7: The progressive LLM token consumption of LLM-Explorer, DroidAgent, and GPTDroid.

图 7:LLM-Explorer、DroidAgent 和 GPTDroid 的逐步大语言模型 (LLM) 令牌消耗情况。

it with AWQ [20] to 4bit and deployed it on a desktop computer equipped with an NVIDIA GeForce RTX 3090 GPU. As shown in Figure 8, there were slight decreases in coverage for most of the apps when using LLM-Explorer with Vicuna- 13B instead of the default GPT-3.5. The reason was mainly due to the decreased quality of LLM responses. Specifically, the smaller Vicuna-13B model may return responses with incorrect formats, invalid element ids, or wrong classifications, leading to inaccurate summarization of abstract elements. Although we had queried the LLM for multiple times to increase the chances of good responses, the problem was not easily avoidable. However, the Vicuna-based LLM-Explorer still exhibited acceptable overall performance, suggesting that LLM-Explorer does not heavily rely on a larger LLM and may be further improved with local model training.

将其使用 AWQ [20] 量化至 4 位，并部署在配备 NVIDIA GeForce RTX 3090 GPU 的台式计算机上。如图 8 所示，使用基于 Vicuna-13B 的 LLM-Explorer 替代默认的 GPT-3.5 时，大多数应用的覆盖率略有下降。原因主要在于 LLM 响应质量的降低。具体而言，较小的 Vicuna-13B 模型可能返回格式错误、无效元素 ID 或错误分类的响应，导致抽象元素的总结不准确。尽管我们多次查询 LLM 以增加获得良好响应的机会，但该问题难以避免。然而，基于 Vicuna 的 LLM-Explorer 仍表现出可接受的整体性能，表明 LLM-Explorer 并不严重依赖更大的 LLM，且有望通过本地模型训练进一步改进。

Using GPT-4. We also evaluated the performance of LLM-Explorer with GPT-4. As depicted in Figure 8, there were slight increases in coverage for most apps, attributed to the higher quality responses from GPT-4. However, despite the superior reasoning capability of GPT-4 compared to GPT- 3.5, the enhancement in average coverage was not significant. Using GPT-3.5 in LLM-Explorer is good enough for most apps, except for some complicated apps (e.g. Wikipedia) that demand more advanced knowledge management. These findings suggest that using the most powerful LLMs may not be necessary and we choose GPT-3.5 as the default choice.

使用 GPT-4。我们还评估了 LLM-Explorer 在 GPT-4 下的表现。如图 8 所示，大多数应用的覆盖率略有提升，这归因于 GPT-4 提供了更高质量的响应。然而，尽管 GPT-4 在推理能力上优于 GPT-3.5，平均覆盖率的提升并不显著。对于大多数应用，使用 GPT-3.5 的 LLM-Explorer 已足够，只有一些复杂应用 (如维基百科) 需要更高级的知识管理。这些发现表明，使用最强大的大型语言模型 (LLM) 可能并非必要，我们选择 GPT-3.5 作为默认选项。

4.5 Case Studies

4.5 案例研究

We further dive deeper into the succeeded and failed cases of LLM-Explorer to understand its advantages and limitations.

我们进一步深入分析了 LLM-Explorer 成功与失败的案例，以了解其优势和局限性。

Cases outperforming human. LLM-Explorer achieved better coverage than human on three apps including File Manager, Open Tracks and SMS Messenger. We compared the activities reached by human and LLM-Explorer to understand the reasons. In Open Tracks, human participants missed the Marker Edit and Marker Detail activities. This was because these two activities require the creation of a marker first, but

案例表现优于人类。LLM-Explorer 在包括文件管理器 (File Manager)、Open Tracks 和短信信使 (SMS Messenger) 在内的三个应用上实现了比人类更好的覆盖率。我们比较了人类和 LLM-Explorer 所达到的活动，以了解原因。在 Open Tracks 中，人类参与者错过了标记编辑 (Marker Edit) 和标记详情 (Marker Detail) 活动。这是因为这两个活动首先需要创建一个标记，但

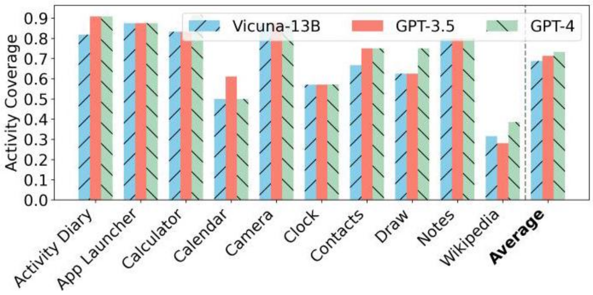


Figure 8: The activity coverage achieved by LLM-Explorer on 10 apps with different LLMs.

图 8:LLM-Explorer 在 10 个应用上使用不同大型语言模型 (LLM) 实现的活动覆盖率。

the creation of a marker fails when the GPS signal is weak. For other activities that LLM-Explorer explored but human users did not, it was mainly due to human participants overlooking detailed functions, such as the Recycle Bin Conversations activity in SMS Messenger, and the Save As activity in File Manager. The feasible path to reach the Save As activity is shown in the top half of Figure 9. To reach the activity, one needs to select a file, click the ”More Options” button, click the ”Share” button, and finally click the ”Save as” button on the screen. The ”Save as” was invisible when the human user selected a folder (instead of a file) to share, and the Save As activity was unreachable. Such a detailed function was uneasy to be noticed by human users, while automated explorers can solve this problem through extensive traversal.

当 GPS 信号较弱时，标记的创建会失败。对于 LLM-Explorer 探索但人类用户未涉及的其他活动，主要原因是人类参与者忽视了细节功能，例如短信应用中的回收站对话活动，以及文件管理器中的另存为活动。达到另存为活动的可行路径如图 9 上半部分所示。要进入该活动，需要先选择一个文件，点击“更多选项”按钮，再点击“共享”按钮，最后点击屏幕上的“另存为”按钮。当人类用户选择共享文件夹 (而非文件) 时，“另存为”按钮不可见，导致另存为活动无法访问。此类细节功能不易被人类用户察觉，而自动化探索器则能通过广泛遍历解决此问题。

Effects of LLM-assisted Knowledge Management. The knowledge of LLM-Explorer could help to reduce the number of redundant actions. Figure 10 shows the same-function element groups determined by the LLM in three different GUI pages. The elements in each box were considered to have the same function and their actions were stored as one abstract action. In the Calculator app, LLM-Explorer grouped the number entry buttons as a cluster, and all arithmetic symbol buttons as another cluster. In the About page, LLM-Explorer was able to group several communication platforms in the SOCIAL section as a cluster having the same function. In the Import Folder page, LLM-Explorer can group several folders together. Such groupings reduced a lot of meaningless actions and action combinations.

LLM 辅助知识管理的效果。LLM-Explorer 的知识能够帮助减少冗余操作的数量。图 10 展示了 LLM 在三个不同 GUI 页面中确定的同功能元素组。每个框中的元素被认为具有相同功能，其操作被存储为一个抽象操作。在计算器应用中，LLM-Explorer 将数字输入按钮归为一组，将所有算术符号按钮归为另一组。在关于页面，LLM-Explorer 能够将社交部分的多个通信平台归为同一功能的群组。在导入文件夹页面，LLM-Explorer 可以将多个文件夹归为一组。这种分组减少了大量无意义的操作及操作组合。

Effects of Content-aware Input Text Generation. LLM-Explorer also uses LLM to generate text input. Figure 11 shows LLM-Explorer’s input when creating a new contact in the Contacts app. LLM-Explorer was able to fill in the contact’s name, phone number, and address information based on the element description. After entering the user name ”John Doe”, it could generate email address information associated with the contact name. These meaningful inputs made LLM-Explorer easier to pass the input checks in the apps.

内容感知输入文本生成的效果。LLM-Explorer 还利用 LLM 生成文本输入。图 11 展示了 LLM-Explorer 在联系人应用中新建联系人时的输入。LLM-Explorer 能够根据元素描述填写联系人的姓名、电话号码和地址信息。在输入用户名“John Doe”后，它还能生成与联系人姓名相关的电子邮件地址信息。这些有意义的输入使得 LLM-Explorer 更容易通过应用中的输入校验。

We also analyzed the situations where LLM-Explorer performed worse. The main causes are as follows.

我们还分析了 LLM-Explorer 表现较差的情况。主要原因如下。

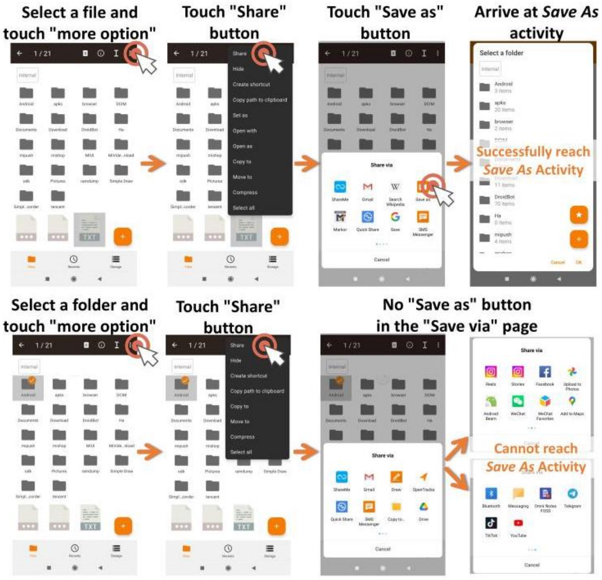

Select a file and touch "more option"
Touch "Share" button
Touch "Save as" button
Arrive at Save As activity
Successfully reach Save As Activity
Select a folder and touch "more option"
Touch "Share" button
No "Save as" button in the "Save via" page
Cannot reach Save As Activity

Figure 9: A case where LLM-Explorer outperformed the human performance. Top: LLM-Explorer successfully reached the Save As activity by selecting a file to save. Bottom: Human users only tried to select a folder, which couldn't reach the Save As activity.

图 9:LLM-Explorer 表现优于人工的案例。上图:LLM-Explorer 通过选择文件成功进入“另存为”操作。下图: 人工用户仅尝试选择文件夹，无法进入“另存为”操作。

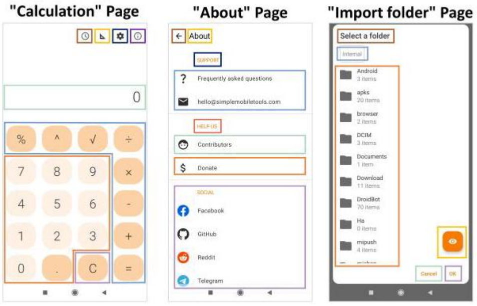


Figure 10: Illustrations of same-function elements determined by LLM-assisted knowledge maintenance module.

图 10: 由 LLM 辅助知识维护模块确定的同功能元素示意图。

(i) Failed to reach activities requiring cross-app jumps. LLM-Explorer performed poorly for activities that need to be navigated across apps. For example, as shown in Figure 12, in the Contacts app, the path to reach the Insert Or Edit Contact Activity was to click on the Dialpad button and then click the ”Add” button on the Dialpad page. However, since the Dialpad page used by Contacts belongs to the Dialer app (not the app under test), LLM-Explorer automatically returned to the Contacts app without further exploration. This kind of failure could potentially be fixed by allowing the agent to explore outside the target app for more steps.

(i) 未能到达需要跨应用跳转的活动。LLM-Explorer 在需要跨应用导航的活动中表现较差。例如，如图 12 所示，在联系人应用中，进入“插入或编辑联系人”活动的路径是点击拨号盘按钮，然后在拨号盘页面点击“添加”按钮。然而，由于联系人应用使用的拨号盘页面属于拨号器应用 (非测试应用)，LLM-Explorer 自动返回联系人应用，未进行进一步探索。这类失败可能通过允许代理在目标应用外探索更多步骤来解决。

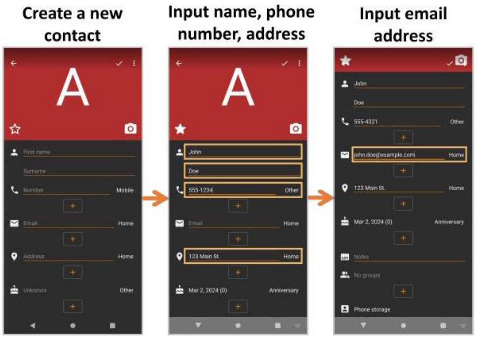


Figure 11: Examples of meaningful text inputs generated by LLM-Explorer in the Contacts app.

图 11:LLM-Explorer 在联系人应用中生成的有意义文本输入示例。

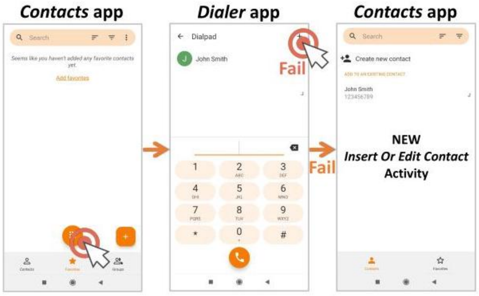


Figure 12: LLM-Explorer failed to reach the activity that required cross-app navigation.

图 12:LLM-Explorer 未能到达需要跨应用导航的活动。

(ii) Errors in maintaining the knowledge. In the LLM-assisted knowledge maintenance module, the LLM may falsely classify elements with different functions into the same group, resulting in some useful actions being grouped into the same abstract action in the knowledge and ignored in future explorations. For example, when LLM-Explorer explored the page shown in Figure 13, the knowledge maintenance module incorrectly judged the elements in the purple box as having the same function. As a result, in the app knowledge, actions of these elements were combined as one abstract action. However, in fact, they could navigate to different GUIs. Similarly, in the About page of the contacts app, the touch actions on the ”Privacy policy” button and the ”Third-party licenses” button were combined into the same abstract action. LLM-Explorer only touched the ”Privacy policy” button and completed the exploration of this part early, missing the chance to visit the License activity by touching the ”Third-party licenses” button. These errors could be mitigated by using better LLMs to generate more precise abstractions.

(ii) 知识维护中的错误。在 LLM 辅助知识维护模块中，LLM 可能错误地将功能不同的元素归为同一组，导致一些有用操作被合并为同一抽象操作并在后续探索中被忽略。例如，当 LLM-Explorer 探索图 13 所示页面时，知识维护模块错误地判断紫色框内的元素具有相同功能，结果在应用知识中将这些元素的操作合并为一个抽象操作。但实际上，它们可以导航到不同的 GUI。同样，在联系人应用的关于页面中，“隐私政策”按钮和“第三方许可”按钮的触摸操作被合并为同一抽象操作。LLM-Explorer 仅触摸了“隐私政策”按钮，提前完成了该部分的探索，错过了通过触摸“第三方许可”按钮访问许可活动的机会。这些错误可通过使用更优的 LLM 生成更精确的抽象来缓解。

5 DISCUSSION

5 讨论

Cross-app navigation. LLM-Explorer automatically navigates back when detecting a transition to another app. This design ensures that the exploration remains focused on the specified app but hinders the discovery of activities that require cross-app transitions. A potential solution is to merge the abstract interaction graphs of different apps into a unified graph, allowing LLM-Explorer to search for possible cross-app return paths when transitioning to another app and enabling more effective cross-app navigation and exploration.

跨应用导航。LLM-Explorer 在检测到跳转到其他应用时会自动返回。此设计确保探索保持在指定应用内，但阻碍了需要跨应用跳转的活动的发现。一个潜在的解决方案是将不同应用的抽象交互图合并为统一图，使 LLM-Explorer 在跳转到其他应用时能搜索可能的跨应用返回路径，从而实现更有效的跨应用导航和探索。

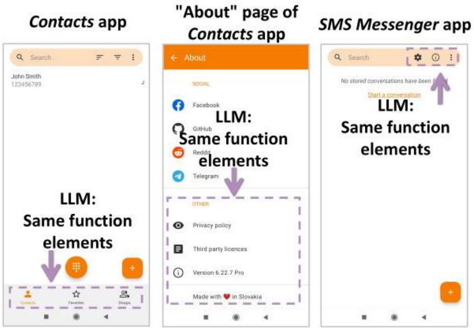


Figure 13: The LLM-assisted knowledge maintenance module of LLM-Explorer may mistakenly group actions with different functions into one abstract action.

图 13:LLM-Explorer 的 LLM 辅助知识维护模块可能错误地将功能不同的操作归为同一抽象操作。

指定应用，但阻碍了需要跨应用跳转的活动的发现。一个潜在的解决方案是将不同应用的抽象交互图合并为统一图，使 LLM-Explorer 在跳转到其他应用时能搜索可能的跨应用返回路径，从而实现更有效的跨应用导航和探索。

Utilization of VLMs in app exploration. Incorporating visual modalities, such as screenshots, into prompts using state-of-the-art multimodal models like GPT-4o is an effective approach to maintaining higher-quality app knowledge and has the potential to enhance system performance. Screen-shots provide crucial information that may not be accurately conveyed through text alone, such as descriptions of images and icons within the interface. However, adding visual modalities typically leads to increased token consumption, which consequently raises system overhead.

应用探索中视觉语言模型 (VLM) 的利用。将截图等视觉模态信息融入提示，使用如 GPT-4o 等先进多模态模型，是维护高质量应用知识的有效方法，且有望提升系统性能。截图提供了文本难以准确传达的重要信息，如界面中的图像和图标描述。然而，加入视觉模态通常会增加令牌消耗，进而提高系统开销。

6 CONCLUSION

6 结论

We present an LLM-based exploration agent for mobile apps. Experiment results show that our method can achieve effective and efficient exploration, outperforming strong baselines. By wisely choosing when and how to use LLM in the exploration process, we can significantly reduce the cost of LLMs while maintaining high exploration performance. We believe that the synergy between LLM and well-organized domain knowledge is the key to making intelligent agents and services more affordable.

我们提出了一种基于大型语言模型 (LLM) 的移动应用探索代理。实验结果表明，我们的方法能够实现高效且有效的探索，优于强基线方法。通过明智地选择在探索过程中何时以及如何使用 LLM，我们可以显著降低 LLM 的成本，同时保持高水平的探索性能。我们认为，LLM 与有序领域知识的协同作用是实现智能代理和服务更具性价比的关键。

ACKNOWLEDGEMENT

致谢

This work is supported by National Natural Science Foundation of China (Grant No.62272261), Tsinghua University (AIR)-AsiaInfo Technologies (China) Inc. Joint Research Center, Wuxi Research Institute of Applied Technologies, Tsinghua University (Grant No.20242001120) and Beijing Academy of Artificial Intelligence (BAAI). Wenjie Du and Cheng Liang contributed as interns at Tsinghua University.

本工作得到中国国家自然科学基金 (项目编号 62272261)、清华大学 (AIR)-亚信科技 (中国) 有限公司联合研究中心、无锡应用技术研究院、清华大学 (项目编号 20242001120) 及北京智源人工智能研究院 (BAAI) 的支持。杜文杰和梁成作为清华大学实习生参与了本研究。

REFERENCES

参考文献

[1] Anshul Arora, Sateesh K. Peddoju, Vikas Chouhan, and Ajay Chaud-hary. 2018. Hybrid Android Malware Detection by Combining Supervised and Unsupervised Learning. In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (Mobi-Com '18). Association for Computing Machinery, New York, NY, USA, 798-800. https://doi.org/10.1145/3241539.3267768

Anshul Arora, Sateesh K. Peddoju, Vikas Chouhan, 和 Ajay Chaudhary. 2018. 结合监督学习和无监督学习的混合安卓恶意软件检测。载于第 24 届年度国际移动计算与网络会议 (Mobi-Com '18) 论文集。计算机协会，纽约，美国，798-800 页。https://doi.org/10.1145/3241539.3267768

[2] Nataniel P. Borges, Maria Gómez, and Andreas Zeller. 2018. Guiding App Testing with Mined Interaction Models. In Proceedings of the 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft '18). Association for Computing Machinery, New York, NY, USA, 133-143. https://doi.org/10.1145/3197231.3197243

Nataniel P. Borges, Maria Gómez, 和 Andreas Zeller. 2018. 利用挖掘的交互模型指导应用测试。载于第五届国际移动软件工程与系统会议 (MOBILESoft '18) 论文集。计算机协会，纽约，美国，133-143 页。https://doi.org/10.1145/3197231.3197243

[3] Abhijit Bose, Xin Hu, Kang G. Shin, and Taejoon Park. 2008. Behavioral detection of malware on mobile handsets. In Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys '08). Association for Computing Machinery, New York, NY, USA, 225-238. https://doi.org/10.1145/1378600.1378626

Abhijit Bose, Xin Hu, Kang G. Shin, 和 Taejoon Park. 2008. 移动终端恶意软件的行为检测。载于第六届国际移动系统、应用与服务会议 (MobiSys '08) 论文集。计算机协会，纽约，美国，225-238 页。https://doi.org/10.1145/1378600.1378626

[4] Haipeng Cai. 2020. Assessing and improving malware detection sustain-ability through app evolution studies. ACM Transactions on Software Engineering and Methodology (TOSEM) 29, 2 (2020), 1-28.

Haipeng Cai. 2020. 通过应用演化研究评估和提升恶意软件检测的可持续性。ACM 软件工程与方法学汇刊 (TOSEM)29 卷，第 2 期 (2020)，1-28 页。

[5] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https: //msys.org/blog/2023-03-30-vicuna/

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, 和 Eric P. Xing. 2023. Vicuna: 一款开源聊天机器人，达到 90%* ChatGPT 质量，令 GPT-4 印象深刻。https://msys.org/blog/2023-03-30-vicuna/

[6] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17). Association

for Computing Machinery, New York, NY, USA, 845-854. https://doi.org/10.1145/ 3126594.3126651

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, 和 Ranjitha Kumar. 2017. Rico: 用于构建数据驱动设计应用的移动应用数据集。载于第 30 届 ACM 用户界面软件与技术研讨会 (UIST '17) 论文集。计算机协会，纽约，美国，845-854 页。https://doi.org/10.1145/3126594.3126651

[7] Android Developers. [n. d.]. UI/Application Exerciser Monkey. https: //developer.android.com/studio/test/other-testing-tools/monkey.

Android 开发者. [无日期]. UI/应用程序测试工具 Monkey。https://developer.android.com/studio/test/other-testing-tools/monkey。

[8] Giorgio Franceschelli and Mirco Musolesi. 2023. On the creativity of large language models. arXiv preprint arXiv:2304.00008 (2023).

Giorgio Franceschelli 和 Mirco Musolesi. 2023. 大型语言模型的创造力研究。arXiv 预印本 arXiv:2304.00008 (2023)。

[9] Yongxiang Hu, Xuan Wang, Yingchuan Wang, Yu Zhang, Shiyu Guo, Chaoyi Chen, Xin Wang, and Yangfan Zhou. 2024. AUITestAgent: Automatic Requirements Oriented GUI Function Testing. arXiv preprint arXiv:2407.09018 (2024).

Yongxiang Hu, Xuan Wang, Yingchuan Wang, Yu Zhang, Shiyu Guo, Chaoyi Chen, Xin Wang, 和 Yangfan Zhou. 2024. AUITestAgent: 面向需求的自动 GUI 功能测试。arXiv 预印本 arXiv:2407.09018 (2024)。

[10] Forrest Huang, Gang Li, Tao Li, and Yang Li. 2023. Automatic Macro Mining from Interaction Traces at Scale. arXiv preprint arXiv:2310.07023 (2023).

Forrest Huang, Gang Li, Tao Li, 和 Yang Li. 2023. 大规模交互轨迹的自动宏挖掘。arXiv 预印本 arXiv:2310.07023 (2023)。

[11] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. 2018. Why Are They Collecting My Data? Inferring the Purposes of Network Traffic in Mobile Apps. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2, 4, Article 173 (dec 2018), 27 pages. https://doi.org/10.1145/ 3287051

金浩健, 刘敏义, Kevan Dodhia, 李元春, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, 和 Jason I. Hong. 2018. 他们为什么要收集我的数据? 推断移动应用中网络流量的目的。ACM 交互式移动、可穿戴及普适技术学报 2 卷 4 期, 文章 173 (2018 年 12 月), 27 页. https://doi.org/10.1145/3287051

[12] Yuanhong Lan, Yifei Lu, Zhong Li, Minxue Pan, Wenhua Yang, Tian Zhang, and Xuandong Li. 2024. Deeply Reinforcing Android GUI Testing with Deep Reinforcement Learning. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE '24). Association for Computing Machinery, New York, NY, USA, Article 71, 13 pages. https://doi.org/10.1145/3597503.3623344

兰元宏, 陆一飞, 李中, 潘敏学, 杨文华, 张天, 和李轩东. 2024. 利用深度强化学习强化 Android 图形用户界面测试。发表于第 46 届 IEEE/ACM 国际软件工程大会 (ICSE ’24)。计算机协会, 纽约, 美国, 文章 71, 13 页. https://doi.org/10.1145/3597503.3623344

[13] Sunjae Lee, Junyoung Choi, Jungjae Lee, Hojun Choi, Steven Y Ko, Sangeun Oh, and Insik Shin. 2023. Explore, select, derive, and recall: Augmenting llm with human-like memory for mobile task automation. arXiv preprint arXiv:2312.03003 (2023).

李善宰, 崔俊英, 李正宰, 崔浩俊, Steven Y Ko, Oh Sangeun, 和申仁植. 2023. 探索、选择、推导与回忆: 通过类人记忆增强大型语言模型 (LLM) 以实现移动任务自动化。arXiv 预印本 arXiv:2312.03003 (2023).

[14] Tong Li, Yong Li, Mohammad Ashraful Hoque, Tong Xia, Sasu Tarkoma, and Pan Hui. 2022. To What Extent We Repeat Ourselves? Discovering Daily Activity Patterns Across Mobile App Usage.

李彤, 李勇, Mohammad Ashraful Hoque, 夏彤, Sasu Tarkoma, 和潘辉. 2022. 我们在多大程度上重复自己？发现移动应用使用中的日常活动模式。

IEEE Transactions on Mobile Computing 21, 4 (2022), 1492-1507. https://doi.org/10.1109/TMC.2020.3021987

IEEE 移动计算汇刊 21 卷 4 期 (2022), 1492-1507. https://doi.org/10.1109/TMC.2020.3021987

[15] Yuanchun Li, Hao Wen, Weijun Wang, et al. 2024. Personal LLM Agents: Insights and Survey about the Capability, Efficiency and Security. arXiv preprint arXiv:2401.05459 (2024).

李元春, 文浩, 王伟军, 等. 2024. 个人大型语言模型代理: 关于能力、效率与安全性的洞见与综述。arXiv 预印本 arXiv:2401.05459 (2024).

[16] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. Droid-Bot: A Lightweight UI-Guided Test Input Generator for Android. In Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C ’17). IEEE Press, 23-26. https: //doi.org/10.1109/ICSE-C.2017.8

李元春, 杨子越, 郭尧, 和陈向群. 2017. Droid-Bot: 一种轻量级的基于 UI 引导的 Android 测试输入生成器。发表于第 39 届国际软件工程大会伴随会议 (ICSE-C ’17)。IEEE 出版社, 23-26 页. https://doi.org/10.1109/ICSE-C.2017.8

[17] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2019. Humanoid: A deep learning-based approach to automated black-box android app testing. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 1070-1073.

李元春, 杨子越, 郭尧, 和陈向群. 2019. Humanoid: 一种基于深度学习的自动化黑盒 Android 应用测试方法。发表于 2019 年第 34 届 IEEE/ACM 国际自动化软件工程大会 (ASE)。IEEE, 1070-1073 页.

[18] Chieh-Jan Mike Liang, Nicholas D. Lane, Niels Brouwers, et al. 2014. Caiipa: Automated Large-Scale Mobile App Testing through Contextual Fuzzing. In Proceedings of the 20th Annual International Conference on

Mobile Computing and Networking (MobiCom '14). Association for Computing Machinery, New York, NY, USA, 519-530. https: //doi.org/10.1145/2639108.2639131

梁杰然, Nicholas D. Lane, Niels Brouwers, 等. 2014. Caiipa: 通过上下文模糊测试实现的大规模自动化移动应用测试。发表于第 20 届年度国际移动计算与网络会议 (MobiCom '14)。计算机协会, 纽约, 美国, 519-530 页. https://doi.org/10.1145/2639108.2639131

[19] Hao Lin, Jiaxing Qiu, Hongyi Wang, Zhenhua Li, Liangyi Gong, Di Gao, Yunhao Liu, Feng Qian, Zhao Zhang, Ping Yang, and Tianyin Xu. 2023. Virtual Device Farms for Mobile App Testing at Scale: A Pursuit for Fidelity, Efficiency, and Accessibility. In Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23). Association for Computing Machinery, New York, NY, USA, Article 45, 17 pages. https://doi.org/ 10.1145/3570361.3613259

林浩, 邱嘉兴, 王洪毅, 李振华, 龚良义, 高迪, 刘云浩, 钱峰, 张钊, 杨平, 和徐天寅. 2023. 大规模移动应用测试的虚拟设备农场: 追求真实性、效率与可访问性。发表于第 29 届年度国际移动计算与网络会议 (ACM MobiCom '23)。计算机协会, 纽约, 美国, 文章 45, 17 页. https://doi.org/10.1145/3570361.3613259

[20] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv (2023).

林吉, 唐佳明, 唐昊天, 杨尚, 邓星宇, 和韩松. 2023. AWQ: 面向大型语言模型压缩与加速的激活感知权重量化。arXiv (2023).

[21] Zhe Liu, Chunyang Chen, Junjie Wang, et al. 2023. Make LLM a Testing Expert: Bringing Human-like Interaction to Mobile GUI Testing via Functionality-aware Decisions. In Proceedings of the 45th International Conference on Software Engineering (ICSE). IEEE/ACM.

刘哲, 陈春阳, 王俊杰, 等. 2023. 让大型语言模型成为测试专家: 通过功能感知决策为移动 GUI 测试带来类人交互。发表于第 45 届国际软件工程大会 (ICSE)。IEEE/ACM.

[22] Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang. 2023. Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing. In 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). 1355-1367. https://doi.org/10.1109/ICSE48619.2023.00119

刘哲, 陈春阳, 王俊杰, 车星, 黄岳凯, 胡军, 王青. 2023. 填空: 面向移动 GUI 测试的上下文感知自动文本输入生成. 载于 2023 年 IEEE/ACM 第 45 届国际软件工程大会 (ICSE). 1355-1367. https://doi.org/10.1109/ICSE48619.2023.00119

[23] Zhe Liu, Cheng Li, Chunyang Chen, Junjie Wang, Boyu Wu, Yawen Wang, Jun Hu, and Qing Wang. 2024. Vision-driven Automated Mobile GUI Testing via Multimodal Large Language Model. arXiv preprint arXiv:2407.03037 (2024).

刘哲, 李成, 陈春阳, 王俊杰, 吴博宇, 王雅文, 胡军, 王青. 2024. 基于视觉驱动的多模态大语言模型自动移动 GUI 测试. arXiv 预印本 arXiv:2407.03037 (2024).

[24] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 2013. Dynodroid: An Input Generation System for

Android Apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013). Association for Computing Machinery, New York, NY, USA, 224-234. https://doi.org/10.1145/2491411.2491450

Aravind Machiry, Rohan Tahiliani, Mayur Naik. 2013. Dynodroid: 安卓应用的输入生成系统. 载于 2013 年第九届软件工程基础联合会议 (ESEC/FSE 2013) 论文集. 计算机协会, 纽约, 美国, 224-234. https://doi.org/10.1145/2491411.2491450

[25] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-Objective Automated Testing for Android Applications. In Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016). Association for Computing Machinery, New York, NY, USA, 94-105. https://doi.org/10.1145/2931037.2931054

毛柯, Mark Harman, 贾越. 2016. Sapienz: 安卓应用的多目标自动化测试. 载于第 25 届国际软件测试与分析研讨会 (ISSTA 2016) 论文集. 计算机协会, 纽约, 美国, 94-105. https://doi.org/10.1145/2931037.2931054

[26] Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, and Xuandong Li. 2020. Reinforcement learning based curiosity-driven testing of Android applications. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2020). Association for Computing Machinery, New York, NY, USA, 153-164. https: //doi.org/10.1145/3395363.3397354

潘敏学, 黄安, 王国新, 张天, 李轩东. 2020. 基于强化学习的好奇心驱动安卓应用测试. 载于第 29 届 ACM SIGSOFT 国际软件测试与分析研讨会 (ISSTA 2020) 论文集. 计算机协会, 纽约, 美国, 153-164. https://doi.org/10.1145/3395363.3397354

[27] Andrea Romdhana, Alessio Merlo, Mariano Ceccato, and Paolo Tonella. 2022. Deep reinforcement learning for black-box testing of android

Andrea Romdhana, Alessio Merlo, Mariano Ceccato, Paolo Tonella. 2022. 基于深度强化学习的安卓黑盒测试

apps. ACM Transactions on Software Engineering and Methodology (TOSEM) 31, 4 (2022), 1-29.

应用. ACM 软件工程与方法学汇刊 (TOSEM)31 卷 4 期 (2022), 1-29.

[28] Konstantin Rubinov and Luciano Baresi. 2018. What Are We Missing When Testing Our Android Apps? Computer 51, 4 (2018), 60-68. https://doi.org/10.1109/MC.2018.2141024

Konstantin Rubinov, Luciano Baresi. 2018. 测试安卓应用时我们遗漏了什么? 计算机杂志 51 卷 4 期 (2018), 60-68. https://doi.org/10.1109/MC.2018.2141024

[29] Bryan Wang, Gang Li, and Yang Li. 2023. Enabling Conversational Interaction with Mobile UI Using Large Language Models. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23). Association for Computing Machinery, New York, NY, USA, Article 432, 17 pages. https://doi.org/10.1145/3544548.3580

Bryan Wang, Gang Li, Yang Li. 2023. 利用大语言模型实现移动 UI 的对话式交互. 载于 2023 年 CHI 人机交互大会论文集 (CHI ’23). 计算机协会, 纽约, 美国, 文章 432, 17 页. https://doi.org/10.1145/3544548.3580895

[30] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2024. Software Testing with Large Language Models: Survey, Landscape, and Vision. IEEE Transactions on Software Engineering (2024), 1-27. https://doi.org/10.1109/TSE.2024.3368208

王俊杰, 黄宇超, 陈春阳, 刘哲, 王松, 王青. 2024. 利用大语言模型的软件测试: 综述、现状与展望. IEEE 软件工程汇刊 (2024), 1-27. https://doi.org/10.1109/TSE.2024.3368208

[31] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, and Tao Xie. 2018. An Empirical Study of Android Test Generation Tools in Industrial Cases. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE ’18). Association for Computing Machinery, New York, NY, USA, 738-748. https://doi.org/10.1145/3238147.3240465

王文宇, 李登峰, 杨威, 曹雨瑞, 张振文, 邓月堂, 谢涛. 2018. 工业案例中安卓测试生成工具的实证研究. 载于第 33 届 ACM/IEEE 自动化软件工程国际会议 (ASE ’18) 论文集. 计算机协会, 纽约, 美国, 738-748. https://doi.org/10.1145/3238147.3240465

[32] Yin Wang, Ming Fan, Xicheng Zhang, Jifei Shi, Zhaoyu Qiu, Haijun Wang, and Ting Liu. 2024. LIReDroid: LLM-Enhanced Test Case Generation for Static Sensitive Behavior Replication. In Proceedings of the 15th Asia-Pacific Symposium on Internetware (Internetware ’24). Association for Computing Machinery, New York, NY, USA, 81-84. https://doi.org/10.1145/3671016.3671404

王银, 范明, 张锡成, 史继飞, 邱兆宇, 王海军, 刘婷. 2024. LIReDroid: 基于大语言模型的静态敏感行为复现测试用例生成. 载于第 15 届亚太互联网软件研讨会 (Internetware ’24) 论文集. 计算机协会, 纽约, 美国, 81-84. https://doi.org/10.1145/3671016.3671404

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems 35 (2022), 24824-24837.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, 等人. 2022. 链式思维提示激发大型语言模型中的推理能力. 神经信息处理系统进展 35 (2022), 24824-24837.

[34] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. Autodroid: Llm-powered task automation in android. In Proceedings of the 30th Annual International Conference on Mobile Computing and Networking. 543-557.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, 和 Yunxin Liu. 2024. Autodroid: 基于大型语言模型的安卓任务自动化. 载于第 30 届国际移动计算与网络年会论文集. 543-557.

[35] Husam N Yasin, Siti Hafizah Ab Hamid, and Raja Jamilah Raja Yusof. 2021. Droidbotx: Test case generation tool for android applications using Q-learning. Symmetry 13, 2 (2021), 310.

Husam N Yasin, Siti Hafizah Ab Hamid, 和 Raja Jamilah Raja Yusof. 2021. Droidbotx: 利用 Q 学习的安卓应用测试用例生成工具. 对称性 13, 2 (2021), 310.

[36] Faraz YazdaniBanafsheDaragh and Sam Malek. 2021. Deep GUI: Black-box GUI Input Generation with Deep Learning. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). 905-916. https://doi.org/10.1109/ASE51524.2021.9678778

Faraz YazdaniBanafsheDaragh 和 Sam Malek. 2021. Deep GUI: 基于深度学习的黑盒 GUI 输入生成. 载于 2021 年第 36 届 IEEE/ACM 自动化软件工程国际会议 (ASE). 905-916. https://doi.org/10.1109/ASE51524.2021.9678778

[37] Faraz YazdaniBanafsheDaragh and Sam Malek. 2022. Deep GUI: black-box GUI input generation with deep learning. In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE ’21). IEEE Press, 905-916. https://doi.org/10.1109/ ASE51524.2021.9678778

Faraz YazdaniBanafsheDaragh 和 Sam Malek. 2022. Deep GUI: 基于深度学习的黑盒 GUI 输入生成. 载于第 36 届 IEEE/ACM 自动化软件工程国际会议 (ASE ’21) 论文集. IEEE 出版社, 905-916. https://doi.org/10.1109/ASE51524.2021.9678778

[38] Chao-Chun Yeh, Shih-Kun Huang, and Sung-Yen Chang. 2013. A black-box based android GUI testing system. In Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys ’13). Association for Computing Machinery, New York, NY, USA, 529-530. https://doi.org/10.1145/2462456.2465717

Chao-Chun Yeh, Shih-Kun Huang, 和 Sung-Yen Chang. 2013. 一种基于黑盒的安卓 GUI 测试系统. 载于第 11 届国际移动系统、应用与服务年会 (MobiSys ’13) 论文集. 计算机协会, 纽约, 美国, 529-530. https://doi.org/10.1145/2462456.2465717

[39] Juyeon Yoon, Robert Feldt, and Shin Yoo. 2023. Autonomous Large Language Model Agents Enabling Intent-Driven Mobile GUI Testing. arXiv preprint arXiv:2311.08649 (ICST 2024) (2023).

Juyeon Yoon, Robert Feldt, 和 Shin Yoo. 2023. 支持意图驱动移动 GUI 测试的自主大型语言模型代理. arXiv 预印本 arXiv:2311.08649 (ICST 2024) (2023).

[40] Li Lyna Zhang, Chieh-Jan Mike Liang, Wei Zhang, and Enhong Chen. 2017. Towards A Contextual and Scalable Automated-testing Service for Mobile Apps. In Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications (HotMobile ’17). Association for Computing Machinery, New York, NY, USA, 97-102. https://doi.org/10.1145/3032970.3032972

Li Lyna Zhang, Chieh-Jan Mike Liang, Wei Zhang, 和 Enhong Chen. 2017. 面向移动应用的上下文感知且可扩展的自动化测试服务. 载于第 18 届国际移动计算系统与应用研讨会 (HotMobile ’17) 论文集. 计算机协会, 纽约, 美国, 97-102. https://doi.org/10.1145/3032970.3032972