Automated Software Engineering (2026) 33:113
https://doi.org/10.1007/s10515-026-00660-7

Reproducing UI contexts described in app reviews with large language models

Ran Jin[1] · Tianming Liu[1,2] · Jun Chen[1] · Yanjie Zhao[1] · Xiao Chen[3] · Xiaoning Du[2] · Li Li[4] · Haoyu Wang[1]

Received: 7 June 2026 / Accepted: 9 August 2026

Abstract

User reviews play a crucial role in the development and maintenance of mobile apps, offering valuable insights into emerging issues, feature requests, and user experiences. However, the sheer volume of reviews poses significant obstacles for developers to handle effectively. This paper introduces LLMARE, an approach that leverages large language models to automatically reproduce UI contexts described in app reviews. LLMARE first pre-explores the app to build a repository of UI states, then analyzes reviews to identify target states that match the described UI elements and functionalities, and finally navigates to the target state to reproduce the described interactions under LLM guidance. We evaluate LLMARE on 160 user reviews from eight popular apps using GPT-4o, achieving initial success rates of 66.3% with a single reproduction attempt, and improving to 88.8% with multiple attempts, where reviews that fail on the first attempt and are eventually reproduced require an average of 1.6 additional attempts, highlighting its potential for app testing and maintenance workflows. Furthermore, evaluation with open-source LLM shows that LLMARE achieves 95% of GPT-4o's performance, indicating strong promise for industrial deployment where data privacy concerns may restrict the use of proprietary LLMs. By automating the reproduction of UI contexts in user reviews, LLMARE bridges the gap between large volumes of user feedback and actionable app improvements, representing a significant step towards more efficient and responsive mobile app development and maintenance processes.

Keywords Mobile app reviews · Issue reproduction · LLM-based mobile testing · Mobile app maintenance

Ran Jin and Tianming Liu are the co-first authors.

Extended author information available on the last page of the article

Published online: 21 August 2026

1 Introduction

The mobile app market has experienced explosive growth in recent years, with millions of apps available on major platforms like the Apple App Store and Google Play (iOS Apple App Store Statistics and Trends 2026; Android and Google Play Statistics 2026). This rapid expansion has intensified the need for effective app maintenance and quality assurance. Among the myriad tasks involved in app maintenance, enhancing apps based on user reviews stands out as crucial. User reviews serve as an indispensable source of insights, offering direct feedback on app performance and user experience (Dabrowski et al. 2022). They often contain valuable suggestions for new features or highlight critical bugs and issues that undermine user satisfaction. By carefully analyzing these reviews, developers can gain valuable insights into app flaws, allowing for prompt correction and improvement to ensure the app stays relevant and impactful in the ever-changing mobile landscape.

However, the sheer volume of app reviews, particularly for popular apps that receive thousands of reviews daily (Top 20 Play Store App Reviews 2026), poses a significant challenge for developers to process and act upon effectively. Despite the recognized importance of user reviews (Dabrowski et al. 2022), little effort has been put into automating the reproduction of reviews within apps. This task is inherently complex, demanding sophisticated natural language understanding to interpret often ambiguous review content, coupled with advanced reasoning capabilities to accurately reconstruct user-described situations during testing.

Recent advancements in large language models (LLMs), particularly in their semantic understanding and reasoning capabilities, offer promising opportunities to address this challenge. To leverage these capabilities, we propose LLMARE (**LLM**-based **A**pp **R**eview R**E**producer), an approach that automatically reproduces UI contexts described in app reviews using LLMs. In this work, we define the term UI context to refer to a specific user-described scenario within an app review, comprising both the UI state (i.e., the specific state of the app's interface) and the user interactions performed in that state (§2.3).

LLMARE 's design was informed by a preliminary study (§3) that explored the direct application of LLM for user review reproduction. Through this study, we identified key challenges in reproducing UI contexts from user reviews using LLM and developed targeted solutions in LLMARE. LLMARE operates in three main phases: first, it comprehensively pre-explores the app to construct a repository of UI states, and leverages LLM to summarize UI states to prepare for the reproduction; second, it analyzes user reviews to extract UI contexts and employs a two-step approach to precisely identify the target activity and UI state required for initiating reproduction; finally, LLMARE navigates to the target state and employs step-by-step LLM guidance to reproduce the described interactions, completing the UI context reproduction.

We evaluated LLMARE on 160 user reviews collected from eight popular apps across prominent app categories on Google Play. Our approach achieved an initial success rate of 66.3% with a single reproduction attempt. This success rate is significantly improved to 88.8% with multiple attempts, while requiring an average of only 1.6 reattempts per case, and an average cost of $0.12 per attempt using GPT-4o. The evaluation result demonstrates LLMARE 's effectiveness and underscores its

potential for app testing and maintenance workflows. In addition, evaluation with open-source LLM achieves 95% of GPT-4o's performance (§5.4), indicating strong promise for industrial deployment where data privacy concerns may limit the use of proprietary LLMs. We also conducted a detailed analysis of success rates across different reproduction stages, identifying root causes of failures and areas for potential enhancement.

In summary, this paper makes the following contributions:

- We make the first attempt at leveraging LLM for automatically reproducing user reviews within apps.
- We propose `LLMARE`, an approach that automatically reproduces UI contexts described in app reviews using LLMs, facilitating more efficient app maintenance processes. `LLMARE` is open-sourced at https://github.com/LLMARE-REPO/LLMARE.
- We present a preliminary study that identified key challenges in applying LLMs for user review reproduction, which informs `LLMARE` 's design that combines app pre-exploration, target state identification, and step-by-step guided reproduction.
- Evaluation shows `LLMARE` achieves a success rate of 66.3% with a single reproduction attempt and 88.8% with multiple attempts using GPT-4o, where initially failed reviews that are eventually reproduced require an average of 1.6 additional attempts, while performance remains comparable when an open source LLM is used, demonstrating its strong potential for app testing and maintenance workflows.

2 Background and related work

2.1 App review analysis

Over the past decade, app reviews have become widely recognized as a vital resource for understanding user needs and enhancing app quality. Numerous studies have focused on app review analysis for various purposes (Arambepola et al. 2025; Dabrowski et al. 2022; Assi et al. 2026), including categorizing and grouping similar reviews (Chaudhary et al. 2025; Maalej et al. 2016; Guzman et al. 2015; Wang et al. 2025), analyzing user sentiments (Motger et al. 2025; Eser and Sahin 2024; Luiz et al. 2018), and extracting specific information such as feature requests and issues (Tiessler and Motger 2025; Motger et al. 2025; Gao et al. 2018). More recently, the emergence of LLMs has significantly advanced app review analysis. Recent studies have leveraged LLMs for automated review classification, feature extraction, and feature recommendation generation, demonstrating superior semantic understanding of user feedback compared with conventional NLP techniques (Gunathilaka and Silva 2025; Motger et al. 2025; Zhao et al. 2025). Nevertheless, these studies mainly focus on extracting or organizing information from app reviews, rather than reproducing the UI contexts described by users. App reviews have also proven valuable for bug detection. Tang et al. (2024) demonstrated the value of app reviews in bug detection by matching user reviews with historical bug reports and showing that relevant issues

could be identified and reproduced. Haering et al. (2021) further demonstrated this value by matching bug reports with app reviews, and discovered that in 47 out of 200 problem-reporting reviews in their dataset, users had identified and described the problem before developers did. Recent work has also begun to leverage LLMs to enrich app reviews and automatically reproduce performance issues, further highlighting the growing potential of app reviews as a valuable source for automated software maintenance (Li et al. 2025).

The work most closely related to ours is RepRev (Li et al. 2020), which extracts information from app reviews to guide GUI exploration for crash reproduction. Unlike RepRev (Li et al. 2020), which aims to reproduce crash-triggering execution paths from bug-related reviews, LLMARE focuses on reproducing the UI contexts described in general app reviews. This requires not only navigating the app but also identifying the intended UI state from often incomplete or ambiguous user descriptions. To address this challenge, LLMARE introduces an LLM-driven pipeline that integrates app pre-exploration, review understanding, and UI context reproduction.

Despite extensive research in this area, the automated reproduction of app reviews, especially the UI contexts they describe, remains largely uncharted. This gap is likely due to the task's inherent complexity, which demands advanced natural language understanding, contextual interpretation, and the capability to simulate app interactions based on user descriptions. However, the emergence of LLMs offers promising opportunities to address these challenges.

2.2 LLM applications in app testing

With the rapid development of LLMs in recent years, significant effort has been directed towards leveraging LLMs to assist in mobile app testing. A majority of these efforts focus on using LLMs for automated task execution, primarily to facilitate mobile agents or helpers. Representative examples include the Mobile-Agent series (Wang et al. 2024; Ye et al. 2025; Wang et al. 2025; Xu et al. 2026), the AppAgent series (Zhang et al. 2025; Li et al. 2024), and the AutoDroid series (Wen et al. 2024, 2025). More recently, these efforts have continued to evolve with improved GUI grounding, planning, reflection, and long-horizon task execution capabilities, demonstrating the rapid progress of LLM-driven mobile GUI agents (Li et al. 2025; Wu et al. 2025; Yu et al. 2026; Chen et al. 2025). While these approaches have demonstrated impressive capabilities in executing user-specified tasks, their task inputs are typically explicit user instructions with clear objectives. Reproducing app contexts from user reviews presents distinct challenges, as reviews often contain informal and ambiguous descriptions, which can be difficult to interpret even for human testers.

Beyond task execution, LLMs have also been applied to various aspects of mobile app testing, including text input generation (Liu et al. 2023, 2024a, b), test migration (Beyzaei et al. 2025; Cao et al. 2025; Zhang et al. 2026), bug report reproduction (Feng and Chen 2024; Wang et al. 2024; Huang et al. 2025), non-crash bugs detection (Ju et al. 2024; Liu et al. 2025), exploratory testing (Su et al. 2024, 2025), and crowdsourced testing and report clustering (Yu et al. 2026; Ling et al. 2025; Wang et al. 2025). These innovative studies have greatly advanced app testing and provided valuable insights that have influenced our approach, as detailed in §4.

A common approach in these applications is **step-by-step LLM guidance** (Liu et al. 2024), which continuously describes the current app screen to the LLM and requests the next action. This iterative process leverages the LLM's language understanding and reasoning abilities to navigate through app interfaces, simulating human-like decision-making and interaction.

2.3 UI context and scope of LLMARE

As introduced in §1, we define a *UI context* as the combination of a specific UI state and the user interactions described in a review. The UI state refers to the app's interface at a particular moment, while the user interactions refer to the actions the user performs or attempts to perform in that state. For example, in the review *"...to select a song when I'm on an artists page..."*, "select a song" is the described user interaction, while "on an artists page" is the UI state where the interaction occurs.

LLMARE is primarily designed for app developers, with a focus on reproducing UI contexts described in user reviews, rather than the entire app review. This targeted approach is chosen because user reviews often describe issues tied to specific conditions or environments (Chen et al. 2024; Su et al. 2020), such as particular device models or configurations, with crucial technical details often omitted. This diversity makes comprehensive replication in a controlled lab environment challenging. By concentrating on reproducing UI contexts, LLMARE aims to streamline the process of improving mobile apps based on user feedback, bridging the gap between the vast volume of user feedback and actionable improvements in areas such as issue fixing, responsive design, and overall user experience optimization.

3 Preliminary study

To effectively identify the challenges in reproducing UI contexts described in app reviews using LLMs, we conducted an initial experiment, attempting to reproduce UI contexts employing the widely adopted step-by-step LLM guidance approach, as outlined in §2.2.

For this preliminary study, we selected four apps with billions of installations: TikTok, Pinterest, Google Photos, and CapCut. From each app's Google Play listing, we manually curated 30 real-world user reviews that we determined contained human-reproducible UI contexts, resulting in a total of 120 reviews for our experiment.

Our methodology employed the step-by-step LLM guidance approach for each review, utilizing GPT-4o (Gpt-4o | Openai 2026):

- Summarize current device screen into an HTML representation (detailed in §4.1.2) and list actions already executed.
- Present this information to the LLM and query it to determine: a) whether the reproduction of the described UI contexts has been completed, and b) if not, what action should be executed next.
- Execute the recommended action if the LLM determines the reproduction is not yet complete.

This process continues iteratively until either the LLM determines it has successfully reproduced the UI contexts described in the review, or the process reaches the predefined limit of 30 actions. For each review, we allowed up to three such reproduction attempts, manually validating the success after each attempt. A reproduction attempt is considered successful only when both the UI states and all described user interactions are correctly reproduced.

As a result, GPT-4o achieved an overall success rate of **54.2%** (65/120). To understand the limitations of this approach, we conducted a detailed manual analysis of all 55 failed reviews and identified three key challenges:

- **C1: Challenges in accurately reaching UI states described in user reviews.** Among all failed reproductions, 24 instances (43.6% of the total failures) were due to difficulties in interpreting and navigating to the described UI states. User reviews often lack detailed descriptions of the steps required to navigate from the app's initial page to the target page where the described interaction occurs. This omission forces the LLM to infer the necessary actions, which often lead to errors. For instance, consider this CapCut review: *"I have one major issue: next to the full-screen and play buttons is an invisible bar that, if you accidentally touch it, drags you to..."* The described UI state includes a "play button" and a "full-screen button", which are only available when editing a video. Reaching this state requires creating a new project or selecting an existing one from drafts, which are crucial steps not mentioned in the review. Without these details, the LLM may fail to navigate to the correct UI state.
- **C2: Challenges in completing consecutive interactions described in user reviews.** 23 of the 55 failed reproductions (41.8%) were attributed to this issue. This issue generally falls into two main categories. First, **user reviews often omit crucial intermediate steps for completing the interaction.** For instance, in Pinterest, "saving a pin" actually requires three distinct actions: selecting the pin thumbnail, clicking the "save" button, and choosing a destination board. The LLM often considers the task complete after clicking "save," missing the critical board selection step not mentioned in the review. **Second, user reviews may describe a sequence of consecutive interactions, where LLMs struggle to reproduce them all.** Consider this Pinterest review: *"...I deleted a board, and now when I try to create another one with the same name, it says 'you have another board with the same name.'"* The LLM might skip the initial board deletion step and proceed directly to board creation, failing to reproduce the complete scenario.
- **C3: Challenges in reproducing multiple non-consecutive UI contexts from a single review.** Users sometimes describe multiple issues within a single review, each corresponding to a separate UI context. The LLM may reproduce only one of these contexts while missing others. This limitation accounted for 13 cases (23.6%) of the 55 failed reproductions (note that some failed reproductions involve multiple challenges, so the total percentage may exceed 100%).

These identified key challenges have informed the design of `LLMARE`. To address **C1**, `LLMARE` begins with an App Pre-exploration phase (§4.1), comprehensively exploring and summarizing UI states within the app. This enables the LLM to select

from known UI states and efficiently navigate to the desired state. To address **C2**, we leverage few-shot learning (Wang et al. 2020) with chain-of-thought reasoning (Wei et al. 2022), guiding the model to fully reproduce described interactions (§4.3). To address **C3**, we instruct the LLM to first extract individual UI contexts from the review and then reproduce each separately.

4 Approach

This paper proposes LLMARE (**LLM**-based **A**pp **R**eview R**E**producer), an approach that automatically reproduces UI contexts described in app reviews using LLMs. LLMARE comprises three main phases: App Pre-exploration, Target Identification, and UI Context Reproduction. Figure 1 presents an overview of LLMARE.

- The **App Pre-exploration** phase is conducted before the reproduction, aiming to comprehensively explore UI states of the app, and employs LLM to summarize UI states, preparing for reproduction. This generally occurs once per app.
- The **Target Identification** phase takes one user review as input at a time. It first extracts all reproducible UI contexts from the user review, then employs a two-step approach to identify the target activity and UI state for initiating UI context reproduction, utilizing information obtained from App Pre-exploration.
- The **UI Context Reproduction** phase first attempts to reach the target UI state identified in the previous phase, then engages with LLM to guide the app step-by-step to complete the reproduction.

4.1 App pre-exploration

This phase explores and summarizes the UI states of the app, which occurs before the actual reproduction. Firstly, it **requires a thorough exploration of the app to cover as many UI states as possible**, as reproduction will rely on these UI states for selecting reproduction targets. This can be achieved using automated UI testing tools (e.g., Li et al. 2017, Lv et al. 2022), through manual exploration, or more com-

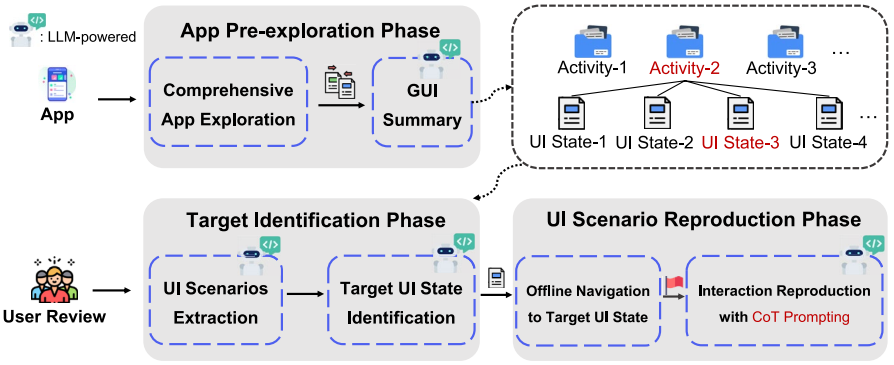

Fig. 1 Overview of LLMARE

monly, a combination of both, which is **not included in our approach**. As LLMARE is primarily designed for app developers, it is likely they have already extensively tested their app. Even if not, as each app generally only needs this process once, the additional workload is limited. We quantify this cost in §5.5, where automated exploration takes approximately 5 hours per application and manual supplementation takes approximately one hour per application, and this one time effort is amortized over all subsequent review reproductions. We further discuss LLMARE 's pre-exploration requirements in §6.

During this comprehensive app exploration, two types of information are collected and provided to LLMARE:

- Basic view hierarchy information for each explored UI state, named **viewtree**, which most automated UI testing tools rely on to understand the UI state and generate UI actions. This can be obtained using UI Automator (Write automated tests with UI Automator | Android Developers 2026) or the underlying Android AccessibilityService (Create your own accessibility service | Android Developers 2026) it relies on, easily accessible for both automated and manual exploration approaches.
- Transition relationships between UI states, referred to as **UTG** (UI Transition Graph Li et al. 2017). This is a directed graph where nodes are explored UI states, and edges are actions leading to transitions between states. The UTG is constructed by recording state transitions during exploration: each time a UI action leads to a new state, a directed edge is added from the source state to the target state, with the action as the edge label. An example of a UTG can be found in Droidbot (2026).

The UI state's viewtree is immediately used in this phase for the LLM to understand the UI state and generate a sub-task list, while the UTG serves as input to the UI Context Reproduction phase for reaching the target state (§4.3), which we elaborate on later.

After laying this groundwork, we now proceed to the design of our App Pre-exploration phase.

4.1.1 App activity and UI state abstraction

In Android, an activity serves as the fundamental component that facilitates interactions between users and the screen. Each activity can encompass multiple UI states, with even minor changes within the activity potentially leading to a new UI state. It is also common for these UI states within the same activity to differ significantly. To enable the subsequent module to select both the target activity and the corresponding UI state for reproduction, we must first establish an effective abstraction for app activities and their UI states. This abstraction should be fine-grained enough to capture the necessary details for reproducing, yet concise enough to ensure that the prompt for selecting the UI state remains manageable for a LLM. Additionally, shorter prompts help reduce LLM costs.

Table 1 Prompt Pattern for Summarizing the Sub-task List of a UI State

ID	Prompt Type	Prompt Instantiation
Parameter		
1	Page HTML Representation	**The widget information on the screen:** Sign up for TikTok Create a pro le, follow other accounts <button id=3 class= "id/c5a" alt= "Use phone or email"></button> ...
Query Pattern		
2	Query Sub-Task List	Please enumerate operations (sub-tasks) that can be performed on the screen. Each sub-task includes the following information: (1) sub-task name; (2) sub-task description. Output format: { name: "..." , description: "..." } Example: { name: "Open Settings", description: "Activate the settings option, accessible through the collapsing toolbar layout." } **<Page HTML Representation>**
Output Format		

3	Output	{
		name: "Open Settings",
		description: "Activate the settings option, accessible through the toolbar."
		};
		{
		name: "Mute Clip Audio",
		description: "Mute the audio of a selected clip using the mute button."
		};
		{
		name: "Search",
		description: "Use the search feature to locate speci c functions."
		} .....

LLMARE **describes a UI state as the set of sub-tasks** that the app can perform within that specific state. It then defines the activity as the union of all sub-tasks across the UI states contained within the activity. A sub-task is an operation a user can accomplish within this UI state, described in our approach with a name and a description generated by LLM as shown in Table 1, **Row 3**. This decision is driven by the fact that users often describe app UI contexts in the form of completing or attempting to complete a sub-task in their app reviews, and draws insight from a recent work (Lee et al. 2024).

4.1.2 Summarizing UI states

We now describe how we use LLM to summarize each UI state, i.e., generating a list of sub-tasks for the UI state. The initial input is the viewtree of the UI state. However, we cannot directly feed the viewtree to the LLM for summarizing sub-tasks, as it is very long and contains too much redundant information. In LLMARE, we first convert the viewtree into an HTML representation before feeding it to the LLM. This decision is inspired by Wang et al. (2023) showing that LLM can efficiently understand UI screens with HTML representation, possibly owing to the fact that LLM's training corpus contains a lot of HTML. The HTML-style representation of UI screens is also adopted by other LLM-based app testing works such as LLMDroid (Wang et al. 2025) and AutoDroid (Wen et al. 2024).

The viewtree to HTML conversion process first involves selecting UI components that contain linguistic information to feed to LLM, as we prompt the LLM based on text conversations (we discuss the use of multimodel LLM in §6.2), and UI compo-

nents without semantic information cannot be understood by LLM. By sufficiently exploring the UI attributes within viewtrees and referencing previous work, we found three primary attributes that contain semantic information: “resource-id", “content-desc", and “text". A UI component with at least one of these three attributes being non-empty is retained in the HTML representation. Following Wang et al.’s practice (Wang et al. 2023), we include these attributes in different components of the HTML syntax, as exampled in Table 1, **Row 1**. In this HTML representation, we establish a taxonomy of five HTML tags, each corresponding to distinct UI component types. Clickable components are mapped to the “button" tag. Editable components are represented by the “input" tag. “ImageView" components are translated to the “img" tag. “TextView" components are rendered as the “p" tag. All other component types are encapsulated within the “div" tag.

To avoid interacting with LLM too frequently for every UI state generated during previous exploration, which could be a very high number since every user action could result in one state, we first perform similarity checks against UI states that have already been summarized by LLM within the same activity. The similarity between two UI states is determined by the number of same views they share, calculated by the Dice-Sørensen Coefficient (2026), which is commonly used to measure similarity between two sets and is defined as $\frac{2|X \cap Y|}{|X|+|Y|}$, where $|X \cap Y|$ is the number of shared views between two UI states, and $|X|$ and $|Y|$ are the total number of views in each UI state respectively. We define views as the same when the class, resource-id, supported actions, width, and height attributes are all identical. In this work, we use a similarity threshold of 0.85 as a heuristic value based on our development experience. During pilot experiments, we found that UI states exceeding this threshold were typically near-duplicates and could reuse the same summary without noticeably affecting subsequent target-state identification. The threshold primarily serves to reduce redundant LLM summarization and to keep the candidate UI-state repository compact. This approach also helps reduce the number of UI states, making it easier for the LLM to select from them in the next phase.

When a UI state is not sufficiently similar to any previously summarized UI states, we prompt the LLM to summarize the new UI state for sub-tasks, as shown in Table 1, **Row 2**.

4.1.3 Summarizing app activities

`LLMARE` creates a comprehensive sub-task list for each activity by combining the sub-task lists of all its UI states. However, this combined list often becomes lengthy, containing similar or duplicate tasks with redundant information. To address this, `LLMARE` leverages the LLM to de-duplicate the sub-task list and to generate a concise 30-word description capturing the activity’s core functionality. An activity is then abstracted as the description and the de-duplicated sub-task list in `LLMARE` to facilitate the target activity identification process in §4.2.2.

4.2 Target identification

The phase is core to LLMARE, which takes an app review as input, extracts all reproducible UI contexts from the user review, and identifies the target UI states for reproduction from those summarized in the previous phase. As directly identifying a target UI state from all possible states is challenging for LLM, we adopt a two-step approach: first identifying the target activity, then pinpointing the specific target UI state within that activity.

4.2.1 Extracting UI contexts from user reviews

To address C3, i.e., the challenge of multiple UI contexts in reviews (§3), LLMARE first employs LLMs to identify and extract all individual UI contexts within a user review before reproducing them one by one. A consideration in this process is that an individual UI context may contain a sequence of consecutive actions, which the LLM might incorrectly split into separate UI contexts. In contrast, truly separate UI contexts typically arise when users include multiple distinct complaints within a single review. Therefore, we resolve this by applying few-shot learning (Wang et al. 2020), providing examples that clearly define one UI context with consecutive actions and non-consecutive UI contexts for the LLM. During this extraction, reviews that do not contain any reproducible UI contexts are naturally filtered out, as there is nothing for LLMARE to reproduce, such as reviews containing only general complaints like "this app is too bad".

4.2.2 Identifying the target activity and UI state

After extracting UI contexts from the user review, LLMARE leverages LLM to determine the target activity for reproducing each UI context. We input to the LLM a set of activities to choose from, each abstracted with both the concise description and the sub-task collection of this activity mentioned in §4.1.3. Both elements are necessary: without the detailed sub-task collection, the LLM might fail to match the described app UI context (associated with a specific sub-task) with the right activity; however, as the sub-task collection can be lengthy even after de-duplication, the concise description helps the LLM quickly grasp the activity's main purpose.

Since the sub-task collection can be extensive and an app may have multiple activities, inputting all activity information at once might exceed LLM's processing capacity, potentially yielding poor selection results. To address this, we employ a multi-step approach:

- We use the BERT text embedding model (Hugging Face 2026) to filter for the top n candidate activities most related to the UI context description extracted from the review, achieved by calculating the semantic similarity between the description and each activity's abstraction.

- We then prompt the LLM to choose one activity from these top n activities that best matches the key information from the app review. The LLM is instructed to either select one activity that could reproduce the UI context or return a “not found" if none of the top n activities match.
- If no match is found, we present the next top n activities for selection, repeating this process until the activity list is exhausted. If the LLM fails to choose an activity after examining all candidate activities, we deem the reproduction attempt unsuccessful.

This “sliding window" strategy ensures that even if the preliminary similarity-based candidate activity identification fails, it does not necessarily result in reproduction failure, while also maintaining a concise list of activities in each prompt, enabling superior LLM performance. Note that this embedding, similarity calculation, and sliding window strategy is also applied in the subsequent target state identification phase. For brevity, we omit the specific prompt pattern from the paper. Readers interested can visit our website for more details and code implementations.

After determining the target activity, `LLMARE` employs a similar approach to identify a specific UI state for reproducing the UI context among all UI states belonging to the target activity. The only difference is that each UI state is represented solely by its sub-task list, without an additional concise description. This decision is based on the observation that, unlike different activities which generally serve distinct main purposes, UI states within the same activity can sometimes be quite similar. This is because UI states within the same activity often share highly similar layouts and differ only in a small number of widgets or UI elements. Adding a concise description to each UI state could be futile or even potentially confuse the LLM.

In `LLMARE`, we use sliding-window sizes of 4 for target activity identification and 3 for target state identification. These values were chosen as heuristic configurations based on our development experience. During target activity identification, each candidate activity is represented by both its activity name and an LLM-generated functional description, providing relatively distinctive semantic cues that allow the LLM to effectively distinguish among a larger set of candidates. In contrast, presenting too many highly similar UI states simultaneously not only increases prompt length but also makes it more difficult for the LLM to identify subtle differences between candidates. Therefore, we use a slightly smaller sliding window for target state identification to balance contextual coverage and prompt complexity. These parameters can also be adjusted based on the capability and context length of the LLM being used.

4.3 UI context reproduction

After determining the target UI state, we leverage the UI Transition Graph (UTG) obtained in the App Pre-exploration phase (§4.1) to calculate the shortest path from the app’s first state to the target state using Dijkstra’s algorithm. We then attempt to reach the target UI state by replaying the actions along this path. In our experience, this target-reaching step yields a relatively high success rate, therefore we implement

it as an offline step without LLM assistance for efficiency. However, it is worth noting that this is essentially a replay problem, which is another line of research (Lam et al. 2017; Gomez et al. 2013; Hu et al. 2015) whose integration with LLM is not discussed in this work. Industrial users could even consider saving device snapshots of UI states to facilitate jump-starting to the target state, or "time-traveling testing" as referred to in Dong et al. (2020).

After reaching the target UI state, we employ LLM for step-by-step guidance to complete the UI context reproduction. This process primarily addresses the reproduction of specific interactions described in the review. To address C2, i.e., the challenge of completely reproducing consecutive interactions (§3), we enhance our prompt design by integrating chain-of-thought reasoning (Wei et al. 2022) and few-shot learning (Wang et al. 2020), two widely-adopted techniques for improving LLM performance, where we guide LLM using carefully curated examples (an example is shown in Table 2) that demonstrate detailed step-by-step reasoning for determining the complete action sequence required for reproduction. These examples specifically address the two situations identified in C2: breaking down seemingly simple user-described interactions into multiple necessary atomic actions, and handling multiple consecutive interactions described by ensuring each is performed in order and properly completed. Prior work also explores automated search over prompt configurations (Zhang et al. 2025), whereas LLMARE adopts fixed prompt templates so that reproduction behavior remains deterministic across applications.

To prevent the LLM from entering loops or executing excessive actions without recognizing reproduction completion, we implement a maximum limit of 15 actions for reproducing the interactions using LLM. This conservative limit helps prevent `LLMARE` from accidentally stumbling upon the correct UI context and potentially inflating its effectiveness.

Table 2 Example of Few-shot Chain-of-Thought Prompting

Reference these examples to understand the reasoning process for completely reproducing the described interactions in the UI context and their required steps.
##Example 1
UI Context: Delete a board and create another board with the same name.
The Complete Action Sequence required to reach the UI context:
1.click the "Delete" button to delete the board;
2.click the "YES" button to confirm deletion;
...
7.click "Done" button to complete creation.
Chain-of-Thought Reasoning Process:
Let's think step by step. The UI context involves two main tasks: deleting a board and creating a new one with the same name.
For the deletion task: First, based on the widget information of the current page, locate the component for deleting the board. The deletion process likely requires confirmation steps.
For the creation task... Therefore, the specific actions required are: click the delete board button, and confirm deletion (Actions 1–2). Next...

5 Experimental results

5.1 Dataset and experimental setup

Our dataset comprises eight highly popular apps of prominent categories from Google Play, four of which already included in the preliminary study (§3): TikTok (SOCIAL), Instagram (SOCIAL), CapCut (VIDEO_PLAYERS), Spotify (MUSIC_AND_AUDIO), Google Photos (PHOTOGRAPHY), Pinterest (LIFESTYLE), Google Translate (TOOLS) and Google Chrome (COMMUNICATION). We selected these highly popular apps as they represent the complexity of modern mobile apps and they have a large number of app reviews for testing. The dataset is limited to only eight apps due to the significant effort required for comprehensive app pre-exploration as non-developers. We discuss the implications of focusing on popular apps and the generalizability of our findings to less popular apps in §6.2.

For these eight apps, we collected approximately 6,000 user reviews per app from Google Play. To eliminate duplicates and clearly irrelevant reviews (e.g., single-word ratings, non-English reviews), we provided the complete initial dataset to the LLM for preliminary screening, reducing each app's dataset to around 1,000 filtered reviews. We then randomly selected reviews from these filtered datasets as input to `LLMARE`. During UI context extraction (§4.2.1), reviews from which no UI contexts could be extracted were not processed further. Additionally, we instruct the LLM to filter for reviews that are meaningful for app improvement, such as those describing issues or suggesting features, as this reflects the typical usage of `LLMARE` where developers focus on actionable feedback. We then manually excluded reviews beyond our experimental setup (e.g., cross-app or cross-device scenarios), reviews describing duplicate UI contexts, and reviews that, despite passing the automatic filtering, did not actually contain reproducible UI contexts upon inspection. After this process, we retained **20** valid, unique, and reproducible reviews per app for our evaluation. This number is constrained mainly by the manual effort required for validating successful reproduction, especially since reviews may undergo multiple reproduction attempts (§5.2).

We employed (Gpt-4o | Openai 2026) throughout our experiment. For the App Pre-exploration phase, we used a combination of automatic exploration (using Droidbot Li et al. 2017, a well-adopted testing tool in our community, to explore each app for 5 hours, discovering an average of 8 Activities and 37 UI states per app) and manual supplementation (approximately one hour, adding an average of 32 UI states per app) to ensure comprehensive app exploration and coverage of UI states associated with the app reviews. After automatically reproducing all 160 reviews, two authors independently assessed each reproduction result. Before the annotation process, the two annotators jointly established the success criterion: a reproduction was considered successful only when all UI contexts described in the review were fully reproduced, including both the specific UI state and the associated interactions for each UI context. For each review, the annotators were provided with the original user review, the target Activity, the target UI state selected by `LLMARE`, the execution action sequence log, and all corresponding screenshots generated during reproduction. Based on these materials, each annotator independently determined whether `LLMARE` successfully reproduced the UI contexts described in the review. If a reproduction was judged

unsuccessful, the annotator was also required to specify the primary reason for the failure. After the independent annotation, we calculated the inter-rater reliability using Krippendorff's α (Krippendorff 2004; Marzi et al. 2024), which reached 0.89, indicating a substantial level of agreement. Finally, any remaining disagreements were resolved through discussion to produce the final consensus labels.

5.2 Overall success rate of LLMARE

The experimental results presented in Table 3 show the success rates for each app. As shown in the yellow columns, LLMARE achieves an overall success rate of **66.3%** with a single reproduction attempt per review.

Motivated by LLM's known output variability even with identical inputs (Ouyang et al. 2023), we explored LLMARE 's performance with multiple reproduction attempts. For reviews that failed in their first attempt, we allowed up to five additional attempts, ceasing once successful reproduction was achieved. The results of this multiple-attempt approach are presented in the blue columns of Table 3.

The results with multiple attempts are remarkable: LLMARE achieves an average success rate of **88.8%**, while requiring only **1.6 retests** on average for previously failed reviews. Most notably, the vast majority of previously failed reviews were successfully reproduced within one (58.3%), two (22.2%), or three (16.7%) retests, suggesting that even a few additional attempts could significantly improve the success rate to 79.4%, 84.4% and 88.1%, respectively. Only one review required four retests for successful reproduction, and none succeeded after five reattempts. This pattern indicates that two to three retests are sufficient to achieve optimal results, with minimal gains in success rate beyond three reattempts.

Analysis of successfully retested reviews reveals two key patterns. First, when apps contain multiple UI states capable of achieving the same action, LLM may initially select a suboptimal state. For instance, in Pinterest, most operations can be performed on both picture and video pins, and LLM might overlook the UI context's specification of the intended object. Second, when user reviews omit detailed operational steps (as mentioned in C2, §3), LLM might not execute all necessary actions in its first attempt, but succeeds in subsequent tries.

These findings are particularly encouraging given that LLMARE is automated and that the cost per reproduction attempt is relatively low ($0.12 on average using GPT-4o). Developers could feasibly implement three or four attempts per review for optimal performance. **This underscores LLMARE 's potential for app testing and maintenance workflows.**

Table 3 Overall Success Rate of LLMARE

App	Single Attempt Success Rate	After Retest Success Rate	Avg. Refests
Google Translate	60.0%	85.0%	2.4
Google Photos	70.0%	95.0%	2.2
Google Chrome	70.0%	95.0%	1.4
TikTok	65.0%	85.0%	1.4
Instagram	85.0%	95.0%	1
CapCut	50.0%	70.0%	2
Spotify	65.0%	90.0%	1
Pinterest	65.0%	95.0%	1.3
Total	**66.3%**	**88.8%**	**1.6**

As shown in Table 3, Google Translate and Google Photos require higher average numbers of retest attempts. Further analysis revealed this is primarily due to target state selection challenges. Both apps contain numerous similar pages, causing the LLM to either struggle with decision-making or select incorrect target states.

CapCut exhibits the lowest success rates across both single attempts (50%) and multiple attempts (70%). This poor performance can be attributed to two factors: the presence of functionally similar activities that lead to LLM selection errors, and the inherent complexity of video editing operations that challenges the LLM's ability to complete described interactions. We provide a detailed stage-wise analysis of success rates to further investigate these failure patterns in the following subsections.

5.3 Success rate breakdown per stage

To gain deeper insights into `LLMARE` 's performance, we perform a breakdown of the success rates with multiple reproduction attempts, across four critical reproduction stages, as illustrated in Table 4. These stages encompass UI context extraction (§4.2.1), target activity selection, target UI state selection (§4.2.2), and UI context reproduction (§4.3). The results reveal that `LLMARE` demonstrates relatively consistent high success rates with each stage ranging from 93.4% to 98.8%. All success rates presented in this subsection are calculated conditionally, assuming 100% successful completion of the previous step, allowing for a more accurate assessment for each individual stage.

This breakdown also allows us to identify the root causes of reproduction failures. Among the four key stages, **completing the UI context reproduction is the most challenging**. Ten user reviews failed at this step, with an overall success rate of **93.4%**. Note that in calculating this success rate, we consider cases successful if the UI contexts are successfully reproduced, even with extra actions when the LLM was unable to determine that reproduction was completed. We made this decision because cases involving extra actions are relatively rare (7.7%), and most of them involve the LLM repeatedly performing actions mentioned by the user.

Table 4 Success Rate Breakdown After Reattempts

App	Stage Name			
	Extract UI Contexts	Identify Target Activity	Identify Target State	UI Context Reproduction
G.Translate	100.0%	100.0%	90.0%	94.4%
G.Photos	100.0%	100.0%	100.0%	95.0%
G.Chrome	100.0%	100.0%	100.0%	95.0%
TikTok	95.0%	100.0%	100.0%	89.5%
Instagram	95.0%	100.0%	100.0%	100.0%
CapCut	100.0%	90.0%	100.0%	93.3%
Spotify	100.0%	100.0%	95.0%	94.7%
Pinterest	100.0%	100.0%	95.0%	100.0%
Average	98.8%	98.7%	97.4%	93.4%

The relatively low success rate in the final stage can be attributed to the following reasons:

- Three cases failed due to incomplete or unclear descriptions of operations in user reviews. In these instances, the LLM struggled to deduce consecutive interactions or infer the next steps accurately. A representative example from TikTok illustrates this challenge: *"I have followed several pages, but when I go to see my followed pages, they aren't there. Then when I search for the page, it has the button for me to follow"*. While this scenario requires searching for users from the homepage's search bar, the LLM, failing to correctly infer the context, attempted the search within the "followed pages" section, which has its own search functionality designed to search for already-followed users.
- Two cases failed due to negatively framed user descriptions. For instance, in *"when I press do not turn on backup, it just spins forever and doesn't let me into my photos at all"*, the user describes a failed attempt to access photos after declining backup. While `LLMARE` should have reproduced both the backup decline and the subsequent photo access attempt, the negative framing of the description challenged our prompt's directive to reproduce the intended actions, causing LLM to miss reproducing the crucial "access photos" step.
- Two additional cases from CapCut involved complex video editing actions, such as video splitting, that exceeded `LLMARE` 's execution capabilities.
- Another two failures resulted from the absence of multimodal capabilities, causing the loss of critical visual information. For example, in Google Translate, while reproducing a Chinese-to-English translation scenario, the HTML representation captured the languages but missed the crucial translation direction indicated by a non-textual arrow figure.
- The final failure stemmed from the LLM's misinterpretation of a user description in CapCut: *"While recording, the video doesn't have any audio..."*. Although the user described an audio issue during video recording, the LLM incorrectly focused on audio-related components instead of reproducing the core action of video recording.

Recall that the UI Context reproduction phase starts with replaying to the target state based on the UI Transition Graph (UTG) (§4.3) before executing interaction reproductions utilizing LLM. While **no user reviews ultimately failed at reaching the target state after multiple attempts**, we observed intermittent failures during single attempts when replaying to the target state. These failures typically occurred due to dynamic changes in app pages. For example, TikTok's starting page, which displayed a video during app pre-exploration, would occasionally show a live stream during reproduction, causing replay to fail.

Among the three steps, the success rate for **extracting UI contexts** is the highest. Since this step only requires extracting descriptions related to the UI context from user reviews, it is relatively easy for LLM to achieve good results. There are only 2 failed cases, both due to users described the UI contexts too ambiguously, to a point even causing authors having conflicts on what it refers to. **Target activity selection** also achieved near-perfect results, with only 2 failed cases in the CapCut app, which

is due to the large number of activities in the app and the similarity between activity subtasks, challenging the LLM's ability to accurately identify and select the correct target activity.

The overall success rate for **selecting the target state** is relatively good with only four failed cases. Analysis of these failures reveals two main causes: LLM's misinterpretation by incorrectly emphasizing secondary aspects of user descriptions (similar to the final case in the UI context reproduction phase), and the presence of similar but distinct subtasks across different UI states. The latter reason is illustrated by the Pinterest app, where "save pins" and "download pins" represent different operations but share similar textual descriptions, where the LLM struggles to differentiate, leading to incorrect state selection when attempting to reproduce download-related scenarios.

This breakdown has led to several potential improvements For UI Context reproduction phase, while LLMARE employs common prompt engineering techniques (few-shot learning and chain-of-thought reasoning) to enhance LLM performance (§4.3), some interactions still fail to reproduce. We propose three possible solutions:

- For handling complex operations, we propose that practitioners could create "atomic" action scripts from snapshots of these operations. When such operations are needed, the LLM could either directly invoke these scripts or learn from them for reproduction.
- **Multimodal models could serve as an alternative solution for failed cases**. While these models are more costly than pure text models, our analysis shows that only 2 reviews failed due to lack of multimodal capabilities. Therefore, reserving multimodal models for specific failed cases could provide an efficient balance between cost and performance.
- Regarding the challenge of negatively framed actions, we have implemented additional examples to illustrate this specific problem in the prompt. This improvement has already shown promise: the two previously failed cases succeeded with one and two attempts respectively.

Regarding target activity and UI state selection, a key challenge emerges when similar abstractions (such as subtasks or activity descriptions) impede the LLM's ability to make accurate distinctions. To address this, we propose enhancing the app pre-exploration phase to proactively identify and reflect on such cases. When similar abstractions are detected, we can prompt LLM to either refine these abstractions to highlight their distinguishing features or merge them where appropriate.

5.4 Baseline comparison and performance with open-source LLM

We evaluated LLMARE against several baselines and assessed its viability with open-source models for industrial deployment. For baseline comparison, we first evaluated the step-by-step guidance approach using GPT-4o, as detailed in our preliminary study (§3). Additionally, while prior work has explored user review reproduction using traditional machine learning techniques, no existing study has investigated this problem using LLMs. Meanwhile, significant progress has been made in closely related LLM-

based areas, including automated task execution (Wen et al. 2024; Lee et al. 2024; Zhang et al. 2025) and bug report reproduction (Feng and Chen 2024; Wang et al. 2024; Huang et al. 2025). Therefore, we selected AppAgent (Zhang et al. 2025) and ReBL (Wang et al. 2024) as representative LLM-based methods from these related tasks. Although they address different problem formulations, both translate natural language descriptions into executable UI interaction sequences, making them the closest available LLM-based baselines for evaluating the effectiveness of LLMARE. For industrial adoption, data privacy and security concerns may limit the use of proprietary LLMs. To evaluate LLMARE 's performance with open-source alternatives, we assessed its capabilities using gpt-oss-120b (Introducing gpt-oss | OpenAI 2026), an open-source model released by OpenAI in August 2025. We selected this model as it shares the same model family with GPT-4o, allowing for a more controlled comparison, while maintaining reasonable computational requirements for industrial deployment. We did not evaluate its 20B variant as preliminary tests showed insufficient performance on the complex multi-step reasoning and instruction-following tasks required for our application.

For this comparison, we randomly selected 10 user reviews from each app in our dataset (§5.1), totaling 80 reviews. To ensure a fair comparison, we employed GPT-4o as the underlying LLM for AppAgent and ReBL, and allowed one initial attempt and up to five additional attempts per review for all approaches, consistent with the evaluation setting. For AppAgent and ReBL, we provided the reviews to them as tasks and bug reports respectively. We applied the same success criteria and validation process outlined in §5.1.

The results in Table 5 show that LLMARE reaches a higher success rate than the adapted baselines under our experimental setup, achieving 86.3% with GPT-4o and 82.5% with gpt-oss-120b after multiple attempts, while the adapted baselines remain below 60%. We note that these baselines were not originally designed for app-review-driven reproduction, so this comparison is intended to characterize the difficulty of transferring them to this task rather than to claim general superiority. This performance gap highlights that solutions designed for automated task execution or bug report reproduction may not directly transfer to user review reproduction. This distinction can be attributed to the unique challenges of user reviews: unlike bug reports which often contain detailed reproduction steps, or task inputs which are typically concise and focused, user reviews often present informal and ambiguous descriptions and complex action patterns, including both consecutive and non-consecutive actions, therefore necessitating dedicated solutions to resolve these challenges.

Table 5 Success Rates of Baselines, LLMARE with GPT-4o, and LLMARE with Open-source LLM

Tool	Single Attempt	After Retest
	Success Rate	Success Rate
Step-by-step (GPT-4o)	32.5%	50.0%
AppAgent	41.3%	51.3%
ReBL	41.3%	57.5%
LLMARE (GPT-4o)	65.0%	86.3%
LLMARE (gpt-oss-120b)	61.3%	82.5%

Compared with LLMARE using GPT-4o, LLMARE with gpt-oss-120b exhibits slightly lower yet highly competitive performance. The open-source model achieves a success rate of 61.3% for single attempts and 82.5% after retesting, corresponding to 94.6% (61.3%/65.0%) and 95.6% (82.5%/86.3%) of GPT-4o's performance, respectively. Notably, gpt-oss-120b requires an average of only 1.5 retests for previously failed cases, which is almost identical to GPT-4o, indicating similar improvement trends with multiple attempts. These results suggest that **large open-source LLMs can serve as practical alternatives** for automating user review reproduction, **making** LLMARE **suitable for industrial deployment** where data privacy concerns may restrict the use of proprietary LLMs.

5.5 Cost

Human cost The only stage requiring manual effort is the one-time UI repository construction during pre-exploration. Automated UI exploration identifies an average of 8 Activities and 37 UI states per application, accounting for approximately 54% of the final UI repository, within 5 hours without human intervention. To improve repository completeness, an average of 32 additional UI states are manually supplemented, requiring approximately one hour per application. After construction, the UI repository achieves over 97.5% coverage of target UI states and can be reused across all subsequent user review reproduction tasks. Consequently, no manual intervention is required during the online reproduction process.

Monetary cost The monetary cost of LLMARE primarily comes from LLM API calls. During the one-time repository construction, each retained UI state requires one LLM call for subtask extraction, and each activity also requires a subtask summary, with an average cost of approximately $0.70 per application. Since this cost is incurred only once and amortized across all experiments, it does not contribute significantly to the runtime overhead. For online reproduction, which includes target Activity identification, target UI state identification, and UI context reproduction, each reproduction attempt incurs an average cost of approximately $0.12 using GPT-4o.

Execution time The execution time of a reproduction attempt is mainly determined by the LLM inference latency and the number of UI interaction steps required to reproduce the target workflow. In our experiments, each reproduction attempt executed an average of 4.7 UI actions. Reproducing a single user review once typically takes 5–15 minutes, depending on the complexity of the target UI context.

Cumulative cost of repeated attempts The cost of reproducing a review scales with the number of attempts. Reviews reproduced on the first attempt incur the cost of a single attempt. Reviews that initially fail and are eventually reproduced require an average of 1.6 additional attempts, which corresponds to approximately 2.6 times the

per attempt cost and execution time. Reviews that are never reproduced consume the full budget of one initial attempt and five additional attempts. Over the full dataset of 160 reviews, LLMARE executes an average of about 1.9 attempts per review, which corresponds to about 2.2 attempts, approximately 0.26 US dollars, and roughly 11 to 33 minutes per successful reproduction. Developers can therefore trade the additional 22.5 percentage points of success rate obtained through retrying against roughly a twofold increase in average cost.

6 Discussion

6.1 Usage scenarios of LLMARE

Previous research has established the value of user reviews in improving app quality, demonstrating that review analysis can support requirements engineering, design, testing, and maintenance activities (Dabrowski et al. 2022). We further illustrate LLMARE 's capability to amplify this value through two potential application scenarios.

First, LLMARE significantly improves the **issue resolution** process by automatically reproducing UI contexts described in user reviews, enabling developers to directly observe and validate reported issues instead of manually interpreting and reproducing them. As mentioned in §2.1, several studies (Grano et al. 2018; Haering et al. 2021; Tang et al. 2024) have demonstrated the value of app reviews in issue identification. However, without automatic reproduction capabilities, leveraging these reviews remains labor-intensive, as developers must manually reproduce described scenarios across various environments to confirm bugs before investigating root causes. LLMARE transforms this workflow by allowing developers to directly observe reported scenarios within apps, significantly accelerating the resolution process. This is particularly valuable for issues that only manifest under specific conditions or device settings. Once LLMARE successfully reproduces the UI context that may trigger the issue, developers can replay this reproduction across device farms (Li et al. 2022) with different configurations, streamlining issue identification, validation, and resolution.

Second, LLMARE aids in **feature improvement** by providing direct visualization of user-reported interaction patterns. Consider this Pinterest review: *"...I am enjoying it. However, it would be appreciated if the process of moving pins from a board to its sub-categories were easier and faster..."*. Instead of relying on abstract textual descriptions, LLMARE provides product managers with concrete visualizations of the user's interaction sequences through UI context reproduction. As shown in Fig. 2, this enables product managers to directly observe the complexity of this seven-step operation, gaining tangible insights into user pain points and opportunities for interface optimization.

Fig. 2 (Simplified) Visualization of A Seven-step User-complained Operation Flow

6.2 Limitations and future work

This work represents a pioneering effort in automating the reproduction of user reviews for app improvement, a critical process for modern app developers. LLMARE demonstrates the potential of leveraging LLM to bridge the gap between user feedback and actionable development tasks. However, it is important to recognize that this is merely an initial step towards fully automating app improvement through user reviews. Future effort in this domain could explore various other aspects, including a more fine-grained classification and prioritization of app reviews, cross-device replay and reproduction for device-specific issues, the development of test oracles for automatically verifying whether reproduced UI contexts actually trigger bugs, and root cause localization by linking reproduced scenarios to relevant code segments. These areas, among others, present promising opportunities for LLM integration.

On the size and representativeness of the evaluation dataset Our evaluation includes 160 manually selected user reviews from eight popular Android applications. While this dataset provides sufficient evidence for demonstrating the feasibility of LLMARE, it is not intended to serve as a statistically representative benchmark for the Android ecosystem. The limited number of reviews per application may reduce statistical power and introduce sampling bias, as manually selected reviews cannot fully reflect the diversity of real-world user feedback. Therefore, our reported success rates should be interpreted as evidence of LLMARE ’s effectiveness on the evaluated dataset rather than definitive estimates of performance across all applications. Future work should construct larger-scale evaluation benchmarks covering more applica-

tions, broader review categories, and varying popularity levels, potentially through semi-automatic benchmark construction techniques that improve both scalability and representativeness.

On the generalizability of LLMARE beyond popular applications We acknowledge that our evaluation focuses on eight popular Android applications, and that LLMs may already be familiar with these apps because information about them is likely to appear in their training data. Consequently, the evaluation results may not directly generalize to less popular apps. However, evaluating on less popular apps is also challenging, as they typically lack sufficient reviews containing reproducible UI contexts. We also considered fabricating user reviews for these apps, either through human creation or LLM generation, but ultimately decided against this approach as synthetic reviews might not accurately represent real-world user feedback patterns. For less popular apps, developers can potentially mitigate LLM unfamiliarity through techniques such as Retrieval-Augmented Generation (RAG) (Gao et al. 2023), providing app-specific information before reproduction using knowledge graphs (Su et al. 2025).

On generalization to other platforms While `LLMARE` is currently engineered specifically for the Android OS, with all existing testing and validation procedures conducted within this environment, its architectural design remains platform-agnostic, holding significant potential for broader application. The core principles and methodologies could be readily extended to other operating systems, such as iOS, HarmonyOS, and even desktop operating systems. This expansion would require modifications to the testing framework, adapting it to each platform's unique characteristics and requirements.

On multimodal applications `LLMARE` currently employs a purely text-based interaction with LLM, which is more economically viable but results in the loss of UI widgets without semantic information in viewtrees. This limitation contributed to reproduction failures in 2 out of 160 reviews in our experiments, as discussed in §5.3. While this issue did not significantly impact our experiments with popular apps, where developers tend to include semantic information more frequently, it could become more pronounced when applied to a wider range of apps. Potential remedies include incorporating traditional image caption models like pix2struct (Lee et al. 2023), directly adopting multimodal LLMs, or using them as a fallback for failed cases to balance cost and performance, as discussed in §5.3.

On `LLMARE` 's pre-exploration requirements `LLMARE` requires comprehensive app pre-exploration to effectively reproduce UI contexts, as it cannot reproduce scenarios involving UI states not captured during this initial phase. However, we consider this a manageable constraint, given that `LLMARE` is designed primarily for app developers who would have already extensively tested their apps. In cases where such omissions occur, developers can systematically update their UI state dataset based on `LLMARE` 's target identification failures. Moreover, future extensions could reduce the pre-exploration burden by integrating techniques for testing app or software updates (e.g.,

Ngo et al. 2022, Ngo et al. 2024, Su et al. 2026), or activity-launching approaches (e.g., Hu et al. Hu et al. , Yan et al. 2020), which can automatically access deeper or newly introduced app states. Such integration would allow LLMARE to incrementally adapt to app updates and better handle edge cases described in user reviews.

On comparison with previous work A noteworthy predecessor in this field is RepRev (Li et al. 2020) in 2020, which aimed to reproduce crashes described in app reviews. RepRev utilized NLP techniques to extract keywords, subsequently guiding app exploration based on UI text similarities. This approach represented a significant step forward and, to our knowledge, was the first attempt at automatically reproducing app reviews. RepRev reported a 70% success rate in reproducing crashes from a manually curated set of 63 user reviews. This rate is superior to LLMARE 's initial success range of 66.3%. However, a key distinction lies in LLMARE 's capacity for improvement through multiple attempts, which elevated its success rate to 88.8%, as detailed in §5.2, while RepRev does not possess this iterative improvement feature. It is worth noting that direct comparative experiments were not feasible due to the unavailability of RepRev. While RepRev's approach was pioneering, we posit that LLM-based methods, as employed in LLMARE, represent a more promising direction for future work in app review reproduction. LLMs offer superior flexibility and potential for comprehending the complex, natural language descriptions typical in user reviews.

Parameter sensitivity The current implementation adopts several heuristic parameters, including the UI state similarity threshold and the sliding-window sizes used during target activity identification and target state identification. These values were selected based on development experience and pilot experimentation rather than a systematic parameter search. Although they worked well across the evaluated applications, we did not conduct a comprehensive sensitivity analysis to quantify how different parameter settings affect reproduction success rate, prompt length, computational cost, or common failure modes. Such an analysis would provide a better understanding of the robustness of LLMARE and may enable adaptive parameter selection for different applications and LLMs. We leave this investigation as future work.

7 Conclusion

This paper introduced LLMARE, an approach leveraging LLMs to automatically reproduce user-described UI contexts in app reviews. Through a preliminary study, we identified the key challenges of directly applying LLMs in this task and developed targeted solutions, leading to LLMARE 's three-phase reproduction pipeline. Evaluation on 160 reviews from eight popular apps showed promising results, with an initial success rate of 66.3% in a single attempt, and significantly improving to 88.8% with multiple attempts, while maintaining comparable performance when using open-source LLM. These findings highlight LLMARE 's promising potential for app testing and maintenance in practice. By automating UI context reproduction from user reviews, LLMARE represents a significant step towards more efficient and responsive app maintenance processes.

Author Contributions R.J., T.L., X.C., X.D., L.L., and H.W. contributed to the conceptualization of this study. R.J., T.L., and H.W. developed the methodology. R.J., T.L., and J.C. conducted the investigation and validation. R.J., T.L., and Y.Z. prepared the original draft of the manuscript. All authors reviewed and edited the manuscript. H.W. was responsible for project administration and funding acquisition. X.C., X.D., L.L., and H.W. supervised the research.

Data Availability Our artifact, including the implementation of LLMARE, the experimental datasets, and results, is available at the following URL: https://github.com/LLMARE-REPO/LLMARE.

Declarations

Competing Interests The authors declare no competing interests.

Ethical Approval Not applicable since there are no human and/or animal studies included in this paper.

References

Android and Google Play statistics, development resources and intelligence (2026). https://www.appbrain.com/stats

Arambepola, N., Munasinghe, L., Wimalasena, W.: From conventional methods to large language models: a systematic review of techniques in mobile app review analysis. Interdisciplinary Journal of Information, Knowledge & Management **20**, (2025)

Assi, M., Hassan, S., Zou, Y.: Llm-cure: Llm-based competitor user review analysis for feature enhancement. ACM Trans. Softw. Eng. Methodol. **35**(4) (2026). https://doi.org/10.1145/3744644

Beyzaei, B., Talebipour, S., Rafiei, G., Medvidović, N., Malek, S.: Automated test transfer across android apps using large language models. Proc. ACM Softw. Eng. **2**(ISSTA), 2227–2250 (2025)

Cao, S., Pan, M., Lan, Y., Li, X.: Intention-based gui test migration for mobile apps using large language models. Proc. ACM Softw. Eng. **2**(ISSTA), 2296–2318 (2025)

Chaudhary, M., Jain, C., Anish, P.R.: Exploring zero-shot app review classification with chatgpt: Challenges and potential. In: Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering, pp. 672–677. (2025)

Chen, J., Li, K., Chen, Y., Wei, L., Liu, Y.: Demystifying device-specific compatibility issues in android apps. In: 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 525–537 (2024). IEEE

Chen, M., Liu, Z., Chen, C., Wang, J., Xue, Y., Wu, B., Huang, Y., Wu, L., Wang, Q.: Beyond static gui agent: Evolving llm-based gui testing via dynamic memory. In: 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1603–1615 (2025). IEEE

Create your own accessibility service | Android Developers (2026). https://developer.android.com/guide/topics/ui/accessibility/service

Dabrowski, J., Letier, E., Perini, A., Susi, A.: Analysing app reviews for software engineering: a systematic literature review. Empir. Softw. Eng. **27**(2), 43 (2022)

Dice-Sørensen coefficient (2026). https://en.wikipedia.org/wiki/Dice-S%C3%B8rensen_coefficient

Dong, Z., Böhme, M., Cojocaru, L., Roychoudhury, A.: Time-travel testing of android apps. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 481–492. (2020)

DroidBot UTG (2026). https://honeynet.github.io/droidbot/report_com.yelp.android/

Eser, G., Sahin, C.: Sentiment analysis and rating prediction for app reviews using transformer-based models (2024)

Feng, S., Chen, C.: Prompting is all you need: Automated android bug replay with large language models. In: Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, pp. 1–13. (2024)

Gao, C., Zeng, J., Lyu, M.R., King, I.: Online app review analysis for identifying emerging issues. In: Proceedings of the 40th International Conference on Software Engineering, pp. 48–58. (2018)

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H., Wang, H., et al: Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997, **2**(1), 32 (2023)

Gomez, L., Neamtiu, I., Azim, T., Millstein, T.: Reran: Timing-and touch-sensitive record and replay for android. In: 35th International Conference on Software Engineering (ICSE), pp. 72–81. IEEE (2013)
google-bert/bert-base-uncased · Hugging Face (2026). https://huggingface.co/google-bert/bert-base-uncased
GPT-4o | OpenAI (2026). https://openai.com/index/hello-gpt-4o/
Grano, G., Ciurumelea, A., Panichella, S., Palomba, F., Gall, H.C.: Exploring the integration of user feedback in automated testing of android applications. In: IEEE 25Th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 72–83. IEEE (2018)
Gunathilaka, S., Silva, N.: Automatic analysis of app reviews using llms. In: ICAART (2), pp. 828–839 (2025)
Guzman, E., El-Haliby, M., Bruegge, B.: Ensemble methods for app review classification: An approach for software evolution (n). In: 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 771–776. IEEE (2015)
Haering, M., Stanik, C., Maalej, W.: Automatically matching bug reports with related app reviews. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 970–981. IEEE (2021)
Hu, Y., Azim, T., Neamtiu, I.: Versatile yet lightweight record-and-replay for android. In: Proceedings of the 2015 Acm Sigplan International Conference on Object-oriented Programming, Systems, Languages, and Applications, pp. 349–366. (2015)
Hu, H., Wang, H., Dong, R., Chen, X., Chen, C.: Enhancing gui exploration coverage of android apps with deep link-integrated monkey. ACM Transactions on Software Engineering and Methodology (2024)
Huang, Y., Wang, J., Liu, Z., Li, M., Wang, S., Chen, C., Hu, Y., Wang, Q.: One sentence can kill the bug: Auto-replay mobile app crashes from one-sentence overviews. IEEE Transactions on Software Engineering (2025)
Introducing gpt-oss | OpenAI (2026). https://openai.com/index/introducing-gpt-oss/
iOS Apple App Store Statistics and Trends 2026. https://42matters.com/ios-apple-app-store-statistics-and-trends (2026)
Ju, B., Yang, J., Yu, T., Abdullayev, T., Wu, Y., Wang, D., Zhao, Y.: A study of using multimodal llms for non-crash functional bug detection in android apps. In: 2024 31st Asia-Pacific Software Engineering Conference (APSEC), pp. 61–70. IEEE (2024)
Krippendorff, K.: Reliability in content analysis: Some common misconceptions and recommendations. Hum. Commun. Res. **30**(3), 411–433 (2004)
Lam, W., Wu, Z., Li, D., Wang, W., Zheng, H., Luo, H., Yan, P., Deng, Y., Xie, T.: Record and replay for android: Are we there yet in industrial cases? In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 854–859. (2017)
Lee, K., Joshi, M., Turc, I.R., Hu, H., Liu, F., Eisenschlos, J.M., Khandelwal, U., Shaw, P., Chang, M.-W., Toutanova, K.: Pix2struct: Screenshot parsing as pretraining for visual language understanding. In: International Conference on Machine Learning, pp. 18893–18912. (2023). PMLR
Lee, S., Choi, J., Lee, J., Wasi, M.H., Choi, H., Ko, S., Oh, S., Shin, I.: Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, pp. 1119–1133 (2024)
Li, Y., Yang, Z., Guo, Y., Chen, X.: Droidbot: a lightweight ui-guided test input generator for android. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 23–26. IEEE (2017)
Li, S., Guo, J., Fan, M., Lou, J.-G., Zheng, Q., Liu, T.: Automated bug reproduction from user reviews for android applications. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice, pp. 51–60 (2020)
Li, C., Jiang, Y., Xu, C.: Cross-device record and replay for android apps. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 395–407. (2022)
Li, Y., Zhang, C., Yang, W., Fu, B., Cheng, P., Chen, X., Chen, L., Wei, Y.: Appagent v2: Advanced agent for flexible mobile interactions. arXiv preprint arXiv:2408.11824 (2024)
Li, Z., Li, Z., Ding, Z.: From feedback to failure: Automated android performance issue reproduction. arXiv preprint arXiv:2508.11147 (2025)

Li, N., Qu, X., Zhou, J., Wen, M., Du, K., Lou, X., Peng, Q., Wang, J., Zhang, W.: Mobileuse: A hierarchical reflection-driven gui agent for autonomous mobile operation. Adv. Neural. Inf. Process. Syst. **38**, 15101–15128 (2025)

Ling, Y., Yu, S., Fang, C., Zhou, Q., Chen, Z.: Llm-based crowdsourced test report clustering. ACM Trans. Softw. Eng. Methodol. (2025). https://doi.org/10.1145/3765756

Liu, Z., Chen, C., Wang, J., Che, X., Huang, Y., Hu, J., Wang, Q.: 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), pp. 1355–1367 (2023). IEEE

Liu, Z., Chen, C., Wang, J., Chen, M., Wu, B., Tian, Z., Huang, Y., Hu, J., Wang, Q.: Testing the limits: Unusual text inputs generation for mobile app crash detection with large language model. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–12 (2024a)

Liu, Z., Chen, C., Wang, J., Chen, M., Wu, B., Huang, Y., Hu, J., Wang, Q.: Unblind text inputs: predicting hint-text of text input in mobile apps via llm. In: Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, pp. 1–20 (2024b)

Liu, Z., Chen, C., Wang, J., Chen, M., Wu, B., Che, X., Wang, D., Wang, Q.: Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–13 (2024)

Liu, Z., Li, C., Chen, C., Wang, J., Chen, M., Wu, B., Wang, Y., Hu, J., Wang, Q.: Seeing is believing: Vision-driven non-crash functional bug detection for mobile apps. IEEE Transactions on Software Engineering (2025)

Luiz, W., Viegas, F., Alencar, R., Mourão, F., Salles, T., Carvalho, D., Gonçalves, M.A., Rocha, L.: A feature-oriented sentiment rating for mobile app reviews. In: Proceedings of the 2018 World Wide Web Conference, pp. 1909–1918. (2018)

Lv, Z., Peng, C., Zhang, Z., Su, T., Liu, K., Yang, P.: Fastbot2: Reusable automated model-based gui testing for android enhanced by reinforcement learning. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, pp. 1–5. (2022)

Maalej, W., Kurtanović, Z., Nabil, H., Stanik, C.: On the automatic classification of app reviews. Requirements Eng. **21**, 311–331 (2016)

Marzi, G., Balzano, M., Marchiori, D.: K-alpha calculator-krippendorff's alpha calculator: A user-friendly tool for computing krippendorff's alpha inter-rater reliability coefficient. MethodsX **12**, 102545 (2024). https://doi.org/10.1016/j.mex.2023.102545

Motger, Q., Oriol, M., Tiessler, M., Franch, X., Marco, J.: What about emotions? guiding fine-grained emotion extraction from mobile app reviews. In: 2025 IEEE 33rd International Requirements Engineering Conference (RE), pp. 6–18. IEEE (2025)

Motger, Q., Miaschi, A., Dell'Orletta, F., Franch, X., Marco, J.: Leveraging encoder-only large language models for mobile app review feature extraction. Empir. Softw. Eng. **30**(3), 104 (2025)

Ngo, C.D., Pastore, F., Briand, L.: Automated, cost-effective, and update-driven app testing. ACM Trans. Softw. Eng. Method. (TOSEM) **31**(4), 1–51 (2022)

Ngo, C.D., Pastore, F., Briand, L.: Testing updated apps by adapting learned models. ACM Trans. Softw. Eng. Methodol. **33**(6), 1–40 (2024)

Ouyang, S., Zhang, J.M., Harman, M., Wang, M.: Llm is like a box of chocolates: the non-determinism of chatgpt in code generation, (2023). arXiv:2308.02828 arXiv preprint

Su, T., Fan, L., Chen, S., Liu, Y., Xu, L., Pu, G., Su, Z.: Why my app crashes? understanding and benchmarking framework-specific exceptions of android apps. IEEE Trans. Softw. Eng. **48**(4), 1115–1137 (2020)

Su, Y., Liao, D., Xing, Z., Huang, Q., Xie, M., Lu, Q., Xu, X.: Enhancing exploratory testing by large language model and knowledge graph. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, pp. 1–12. (2024)

Su, Y., Xing, Z., Wang, C., Chen, C., Xu, S., Lu, Q., Zhu, L.: Automated soap opera testing directed by llms and scenario knowledge: Feasibility, challenges, and road ahead. Proc. ACM Softw. Eng. **2**(FSE), 757–778 (2025)

Su, Y., Pradel, M., Chen, C.: Rippleguitester: Change-aware exploratory testing, (2026). arXiv:2603.03121 arXiv preprint

Tang, X., Tian, H., Kong, P., Ezzini, S., Liu, K., Xia, X., Klein, J., Bissyandé, T.F.: App review driven collaborative bug finding. Empir. Softw. Eng. **29**(5), 124 (2024)

Tiessler, M., Motger, Q.: Feclustre: Hierarchical clustering and semantic tagging of app features from user reviews, (2025). arXiv:2510.18799 arXiv preprint

Top 20 Play Store App Reviews (Daily Update) (2026). https://www.kaggle.com/datasets/odins0n/top-20-play-store-app-reviews-daily-update

Wang, Y., Yao, Q., Kwok, J.T., Ni, L.M.: Generalizing from a few examples: A survey on few-shot learning. ACM Comput. Surv. (csur) **53**(3), 1–34 (2020)

Wang, B., Li, G., Li, Y.: Enabling conversational interaction with mobile ui using large language models. In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–17 (2023)

Wang, J., Xu, H., Jia, H., Zhang, X., Yan, M., Shen, W., Zhang, J., Huang, F., Sang, J.: Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. Adv. Neural. Inf. Process. Syst. **37**, 2686–2710 (2024)

Wang, D., Zhao, Y., Feng, S., Zhang, Z., Halfond, W.G., Chen, C., Sun, X., Shi, J., Yu, T.: Feedback-driven automated whole bug report reproduction for android apps. In: Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 1048–1060. (2024)

Wang, Q., Erqsous, M., Khatiwada, P., Karwankar, A., Alhassan, F.M., Chandrasekaran, A., Abraham, B., Lovell, F., Ngo, A.A., Mauriello, M.L.: Leveraging large language models for review classification and rating estimation of mental health applications. In: Proceedings of the International AAAI Conference on Web and Social Media, vol. 19, pp. 2017–2029. (2025)

Wang, Z., Xu, H., Wang, J., Zhang, X., Yan, M., Zhang, J., Huang, F., Ji, H.: Mobile-agent-e: Self-evolving mobile assistant for complex tasks. arXiv preprint arXiv:2501.11733 (2025)

Wang, Y., Zhao, Y., Yu, S., Chen, Z.: Multi-dimensional assessment of crowdsourced testing reports via llms. In: 2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1794–1806. IEEE (2025)

Wang, C., Liu, T., Zhao, Y., Yang, M., Wang, H.: Llmdroid: Enhancing automated mobile app gui testing coverage with large language model guidance. Proc. ACM Softw. Eng. **2**(FSE), 1001–1022 (2025)

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. Adv. Neural. Inf. Process. Syst. **35**, 24824–24837 (2022)

Wen, H., Li, Y., Liu, G., Zhao, S., Yu, T., Li, T.J.-J., Jiang, S., Liu, Y., Zhang, Y., Liu, Y.: Autodroid: Llm-powered task automation in android. In: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, pp. 543–557 (2024)

Wen, H., Tian, S., Pavlov, B., Du, W., Li, Y., Chang, G., Zhao, S., Liu, J., Liu, Y., Zhang, Y.-Q., et al: Autodroid-v2: Boosting slm-based gui agents via code generation. In: Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services, pp. 223–235 (2025)

Write automated tests with UI Automator | Android Developers (2026). https://developer.android.com/training/testing/other-components/ui-automator

Wu, Q., Cheng, K., Yang, R., Zhang, C., Yang, J., Jiang, H., Mu, J., Peng, B., Qiao, B., Tan, R., et al.: Gui-actor: Coordinate-free visual grounding for gui agents. Adv. Neural. Inf. Process. Syst. **38**, 15101–15128 (2025)

Xu, H., Zhang, X., Liu, H., Wang, J., Zhu, Z., Zhou, S., Hu, X., Gao, F., Cao, J., Wang, Z., et al.: Mobile-agent-v3. 5: Multi-platform fundamental gui agents. arXiv preprint arXiv:2602.16855 (2026)

Yan, J., Liu, H., Pan, L., Yan, J., Zhang, J., Liang, B.: Multiple-entry testing of android applications by constructing activity launching contexts. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 457–468 (2020)

Ye, J., Zhang, X., Xu, H., Liu, H., Wang, J., Zhu, Z., Zheng, Z., Gao, F., Cao, J., Lu, Z.: Mobile-agent-v3: Fundamental agents for gui automation, (2025). arXiv:2508.15144 arXiv preprint

Yu, S., Ling, Y., Fang, C., Zhou, Q., Zhao, Y., Chen, C., Zhu, S., Chen, Z.: Scenario-guided llm-based mobile app gui testing. ACM Transactions on Software Engineering and Methodology (2026)

Yu, S., Ling, Y., Fang, C., Chen, Z., Chen, C.: Towards automated crowdsourced testing via personified-llm. arXiv preprint arXiv:2603.24160 (2026)

Zhang, C., Yang, Z., Liu, J., Li, Y., Han, Y., Chen, X., Huang, Z., Fu, B., Yu, G.: Appagent: Multimodal agents as smartphone users. In: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, pp. 1–20. (2025)

Zhang, T., Ma, L., Cheng, S., Liu, Y., Li, N., Wang, H.: Automatic prompt design via particle swarm optimization driven llm for efficient medical information extraction. Swarm Evol. Comput. **95**, 101922 (2025)

Zhang, Y., Liu, C., Xie, X., Lin, Y., Dong, J.S., Hao, D., Zhang, L.: Gui test migration via abstraction and concretization. ACM Trans. Softw. Eng. Methodol. **35**(2), 1–29 (2026)

Zhao, Y., Hou, X., Wang, S., Wang, H.: Llm app store analysis: A vision and roadmap. ACM Trans. Softw. Eng. Methodol. **34**(5), 1–25 (2025)

Authors and Affiliations

Ran Jin[1] · Tianming Liu[1,2] · Jun Chen[1] · Yanjie Zhao[1] · Xiao Chen[3] · Xiaoning Du[2] · Li Li[4] · Haoyu Wang[1]

✉ Haoyu Wang
haoyuwang@hust.edu.cn

Ran Jin
jran_@hust.edu.cn

Tianming Liu
tmliu@hust.edu.cn

Jun Chen
jun_c@hust.edu.cn

Yanjie Zhao
yanjie_zhao@hust.edu.cn

Xiao Chen
xiao.chen@newcastle.edu.au

Xiaoning Du
Xiaoning.Du@monash.edu

Li Li
lilicoding@ieee.org

[1] Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China

[2] Faculty of Information Technology, Monash University, Melbourne, Australia

[3] University of Newcastle, Newcastle, Australia

[4] School of Software, Beihang University, Beijing, China