Understanding and Detecting Accessibility Issues in Ageing-Fit Mobile Applications

Wenjie Li[†‡], Weiwei Jiang[§], Cong Li[¶], Yepang Liu[‖], Kaizhong Zuo[††], Chang Xu[‡]
[†]*School of Intelligent Information and Advanced Manufacturing*, *Anhui Normal University*, China
[‡]*State Key Laboratory of Novel Software Technology*, *Nanjing University*, China
[§]*School of Computer Science*, *Nanjing University of Information Science and Technology*, China
[¶]*Department of Computer Science*, *ETH Zurich*, Switzerland
[‖]*Department of Computer Science and Engineering*, *Southern University of Science and Technology*, China
[††]*School of Computer and Information*, *Anhui Normal University*, China
Email: wenjielinju@gmail.com, weiweijiangcn@gmail.com, cong.li@inf.ethz.ch, liuyp1@sustech.edu.cn, zuokz@ahnu.edu.cn, changxu@nju.edu.cn

***Abstract*—Ageing-fit mobile applications (or ageing-fit mobile apps) are increasingly adopted by ageing people, helping them use mobile apps more conveniently. However, Accessibility issues For agEing people (AFE issues for short) are prevalent in ageing-fit apps, significantly impeding their accessibility. Unfortunately, there is a lack of attention and in-depth study concerning AFE issues in ageing-fit apps. This paper presents a large-scale, comprehensive empirical study involving 691 real-world AFE issues gathered from 31 popular, commercial mobile ageing-fit mobile apps specifically designed for ageing users. Our study confirms the prevalence and severity of AFE issues in ageing-fit mobile apps. In addition, we found that certain types of AFE issues in aging-fit mobile apps stand out more prominently compared to previous studies conducted on general apps, and there are AFE issues in ageing-fit apps that were not previously recognized as accessibility issues in general apps. Furthermore, existing detectors either overlooked the AFE issues or failed to effectively detect them. Building on the findings, we developed AGEINGDROID, a tool specifically designed for effective AFE issue detection. We evaluated AGEINGDROID with 36 real-world, popular, and commercial ageing-fit apps. The experimental results are encouraging: AGEINGDROID detected 345 previously-unknown AFE issues, of which 333 have been confirmed by ageing people.**

***Index Terms*—Accessibility; User experience; Mobile application; Ageing people**

I. INTRODUCTION

The world is aging. The World Health Organization forecasts that individuals aged 60 years and older will rise from 1 billion in 2020 to 1.4 billion in 2030 [1]. Meanwhile, *mobile apps* play a significant role in the ageing people's daily life, serving various purposes such as entertainment, socializing, and shopping [2]. Thus, it is crucial to make mobile apps *accessible* to ageing people, considering the common age-related decline in various abilities, including vision, cognitive function, and memory, which can hinder their effective use of mobile apps [3]–[7].

Popular mobile operating systems like iOS and Android offer integrated assistive services, such as Text Scaling Assistive Service [8] and TalkBack [9], to help ageing people use general mobile apps for general audiences and perform tasks that may be difficult to complete. Nevertheless, there are numerous accessibility and compatibility issues for mobile apps with assistive services, which makes it still difficult for ageing people to use mobile apps when assistive services are enabled [10]–[15]. To alleviate this problem, there is an emerging trend where commercial apps are becoming *ageing-fit*, offering a specialized mode for the ageing people. This ageing-fit mode is customized to accommodate the physiological characteristics of ageing people and does not rely on assistive services integrated in mobile operating system, effectively eliminating accessibility and compatibility issues for mobile apps with assistive services. However, the lack of development experience for designing apps specifically for ageing people has resulted in significant accessibility challenges, including issues such as small fonts and icons. In this paper, we call such Accessibility issues For agEing people *AFE issue*s.

Unfortunately, there is an absence of enough attention and in-depth study concerning AFE issues in ageing-fit mobile apps. Different from general accessibility issues concentrate on the accessibility when assistance services enabled in general mobile app [10]–[14], [16], AFE issues are not related to assistance services, making empirical research results and detection tools for general accessibility issues unsuitable for AFE issues. Our study results show that there are some differences for AFE issues in the issue types and the frequency of issue occurrence compared with general accessibility issues. Although there still exist some tools for accessibility issues in general mobile apps can be applied to AFE issue detection [15], [17], [18], our study results (Section IV-C) show that these tools fail to identify more than 60% of the AFE issues in ageing-fit mobile apps for the reason that the rules defined behind these tools do not well-suited to address the unique characteristics of accessibility issues specifically for ageing-fit mobile apps.

This paper presents the first comprehensive and quantitative study towards characterizing AFE issues in commercial ageing-fit mobile apps. First, we employed three senior students who are not involved in this project to learn the international, standard accessibility principles [19]–[21] for ageing people carefully and identified potential AFE issues

from 31 real-world popular, commercial ageing-fit mobile apps. Next, we recruited 34 ageing people who are using mobile phones in their daily life with diverse ages and fair gender distributions to confirm whether each issue identified by students are true AFE issues. Finally, we aggregated all issues that are truly confirmed into a dataset which we call AFE-101 (including 691 AFE issues spanning 174 GUI pages of all 31 studied apps) and carefully analyzed each issue in detail. The study highlights several important findings:

1) AFE issues are very common in ageing-fit mobile apps: We have found AFE issues in all of the 31 studied ageing-fit mobile apps and finally 691 AFE issues were confirmed.
2) A few types of AFE issue can cover more than 96.0% AFE issues: *Small Text* (54.0%), *Small Icon* (17.4%), and *Missing Prompt* (24.3%). Among them, *Small Text* and *Small Icon* stand out more prominently compared to existing study results [18] in general apps, and *Missing Prompt* is not considered as accessibility issues in general apps before.
3) State-of-the-art tools for accessibility issue detection is ineffective when applied to AFE issues: 62.1% AFE issues are failed to detect.

Based on our empirical findings, we designed and implemented AGEINGDROID for effective detection of AFE issues in ageing-fit mobile apps. We experimentally validated the effectiveness and usefulness of AGEINGDROID by applying it to 36 real-world popular commercial ageing-fit mobile apps. AGEINGDROID reported surprisingly encouraging results that it achieved a precision of 0.963 and a recall of 0.739 with respect to AFE-101 and also discovered 333 confirmed, previously unknown AFE issues in 15 apps. Our study and evaluation, which involve a total of 1,024 valid AFE issues, not only demonstrate the effectiveness and usefulness of AGEINGDROID, but also highlight that *there is still a long way to go in meeting the needs of ageing users in practice*.

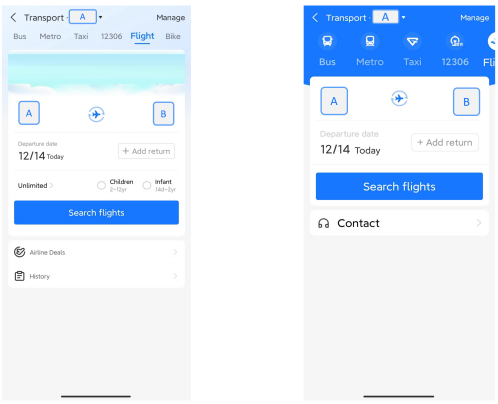
In summary, this paper makes the following major contributions:

- We conducted the first comprehensive empirical study of AFE issues in 31 real-world popular, commercial ageing-fit apps and created a dataset called AFE-101 involving 691 valid AFE issues.
- We devised AGEINGDROID to detect AFE issues in practice , and the encouraging experimental results confirmed AGEINGDROID's effectiveness and usefulness.
- We publicized AFE-101, AGEINGDROID, and all the evaluation data via the following link to facilitate follow-up studies: https://github.com/AFE-Data/afe-data

II. BACKGROUND

A. Characteristics of Ageing-Fit Mobile Apps

Ageing people typically experience a decline in a wide variety of their abilities such as hearing loss, vision and physical decline, and cognitive limitations [22]. However, ageing people are important customers to mobile apps [3], [4], [20]. To make mobile apps even accessible to ageing users, many projects and guidelines have been proposed. For example: European Commission funded the WAI-AGE project to explore the standard for addressing ageing user needs [23]. Web Content Accessibility Guideline (WCAG) was also extended to support ageing users [24]. Recently, China Consumers Association released a report to investigate the development of ageing-fit mobile apps [19] which are supposed to follow the guidelines proposed by the Chinese government [25]. In addition, a large number of mobile apps have been customized or wholly redeveloped (such as enlarging fonts and simplifying logic) to meet the needs of ageing users following certain design principles [20]. Figure 1 shows an example of ALIPAY's general-purpose and ageing-fit modes. Compared with the general-purpose mode, the ageing-fit mode eliminates unnecessary functions for ageing people, increases font size, and incorporates informative icons. These modifications are intended to enhance the accessibility of the app for ageing people, ensuring a more intuitive and user-friendly experience.



General-Purpose Mode Ageing-Fit Mode

Fig. 1: ALIPAY'S TICKET PURCHASE FUNCTION without (left) and with (right) the ageing-fit option enabled.

B. AFE Issues in Ageing-Fit Mobile Apps

Developing ageing-fit mobile apps is challenging as it requires the developers to have deep understanding of the physiological characteristics to meet the needs of ageing users [4]. Failing to satisfy these requirements roots an app with AFE issues and is likely to lead to serious consequences. For example, some apps contain in-app, native ads that blend seamlessly with the surrounding content [26]. Such ad spaces are often used by bad ad providers who may embed links to "rogue" or even malicious software that many people, especially ageing people with cognitive limitations, are "cheated", resulting in privacy data theft or even property damage [27]. Other apps are always choosing small icons/texts for advertising and misleading prompts to hide real intentions, making it difficult for ageing people to discern [28]. A recent study also reported that ageing people are expecting more secure ageing-fit mobile apps [19].

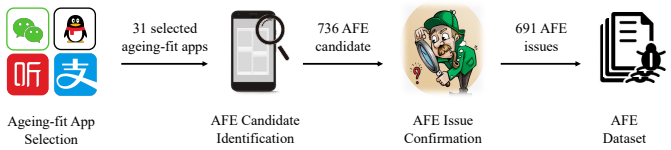


Fig. 2: The overall process of collecting AFE issues.

III. Empirical Study Methodology

A. Research Questions

The empirical study of AFE issues in ageing-fit mobile apps is organized around the following research questions:

RQ1 *How prevalent are AFE issues in ageing-fit mobile apps?* The study aims to provide valuable data to developers and policy makers, with the goal of shedding light on the current state of the AFE issues of ageing-fit mobile apps.

RQ2 *What are the most common types of AFE issues in ageing-fit mobile apps?* The common types can help reveal the main barriers that prevent ageing people from effectively using ageing-fit mobile apps. They can also offer guidance to app developers for avoidance.

RQ3 *Can existing techniques effectively detect AFE issues in ageing-fit mobile apps?* This evaluates the current status of the state-of-the-art techniques in accessibility issue detection.

To answer the above questions, we conducted the empirical study by first collecting a dataset of AFE issues from real-world, popular, commercial ageing-fit mobile apps. As the first (to the best of our knowledge) comprehensive study to AFE issues, our study targets Android, the most popular platform in the world [29]. Figure 2 illustrates the overall process of AFE issue collection.

B. Dataset Collection

App selection. We randomly selected 31 ageing-fit Android apps as subjects meeting the following criteria:

- *Popular*: Our study only considered real-world, commercial apps with at least 50 million downloads in Google Play and Xiaomi GetApps. We choose commercial apps because they are the primary targets of ageing people, compared with open-source apps. We did not select apps with less popularity because our experience tells us that such apps (1) are seldom used by ageing people and (2) often lack well ageing-fit designs of which the AFE issues are weakly representative.
- *Ageing-fit*: All apps selected should have a customized ageing-fit version. We excluded the apps without customized ageing-fit version for the reason that their general audience is not targeted towards the ageing people and the qualities of AFE issues in these apps are less valuable.
- *Diversified-categories*: We selected a wide range of app categories for a complete empirical study, including Social, Shopping, Media, etc. Diverse app categories can help avoid the lack of generality in research results due to the unique characteristics of some certain apps.

AFE candidate identification. To identify AFE candidates, we employed three senior students who are irrelevant to this project and are with rich experience in Android development to carefully read and understand the popular, international accessibility guidelines for ageing-fit mobile apps [19], [30]. After that, we asked the students to manually execute each app being studied for an hour and obtained screenshots of GUI pages that are ageing-fit designed[1]. We believe that one hour of manual execution is sufficient to reach the main GUI pages that ageing people can reach in practice, as prior research showed that ageing users hardly use the deep, complicated logical functionalities [31]. For each screenshot, we asked the students to check whether the design of the respective GUI page meets the guidelines. Specifically, if any widget in the screenshot is not compliant with the guidelines, they reported an AFE issue in the GUI page. We then conducted a cross validation for each reported issue and required confirmations from two out of the three students before regarding it as a candidate. Finally, we obtained a total of 736 AFE candidates.

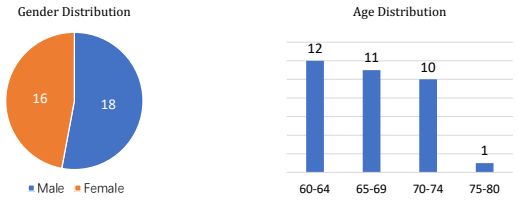


Fig. 3: Distributions of recruited ageing users in the study.

AFE issue confirmation. In order to confirm whether the identified AFE candidates are true AFE issues, we also offered these issues to ageing users for confirmation[2]. We did not consult app developers for confirmation because our study aims at collecting AFE issues that truly affect ageing users; we were also realized that a majority of app developers are unaware of accessibility design guidelines [32], [33], which may lead to inaccurate confirmation due to lack of awareness about accessibility [18]. Specifically, we recruited a set of 34 ageing users with a fair gender distribution and a broad range of ages (Figure 3). For each AFE candidate, we randomly distributed it to three different ageing people. We treated it as a true AFE issue if we received confirmation from two of them that they have difficulties in reading, understanding, and interactions, etc. In the end, out of the 736 AFE candidates, 691 were confirmed as true issues and we organized them into a dataset called AFE-101.

C. Dataset Analysis

Based on AFE-101, we tried to answer the three research questions raised in Section III-A.

[1]Although all apps that we selected are ageing-fit, we were realized that not all GUI pages of them are specifically ageing-fit customized. For such GUI pages, we excluded them from our study.

[2]The authors' affiliated institution approved the authors to conduct this experiment.

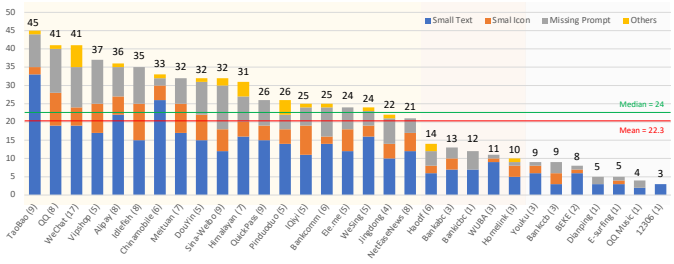


Fig. 4: Distribution of AFE issues in AFE-101 across studied ageing-fit mobile apps. Numbers above each box is the number of AFE issues for each app. Numbers inside parenthesis following the name of each app represent the number of involved GUI pages.

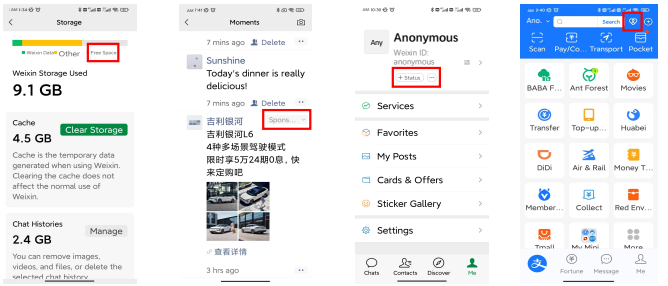
- To answer RQ1, we counted the number of confirmed AFE issues in each studied ageing-fit Android app and the number of GUI pages involving AFE issues.
- To answer RQ2, we classified each AFE issues based on the characteristics on the GUI page and counted the number of issues in each category.
- To answer RQ3, we chose the state-of-the-art accessibility analysis tool for Android, Google's Accessibility Scanner (GASCANNER), and applied it over AFE-101 to test its capability in detecting accessibility issues specifically for ageing people. We have excluded other tools for detecting accessibility issues, e.g., AccessiText [10], from our study because the issue types they concern do not overlap with ours.

IV. STUDY RESULTS

A. Answering RQ1: Prevalence of AFE Issues

Overall, the 691 AFE issues span 174 GUI pages of 31 ageing-fit Android apps, with ∼4.0 issues per GUI and ∼22.3 issues per app; the median value is 3 and 24 issues, respectively. Figure 4 displays the issue distribution over different apps. In particular, *all* the apps under study exhibit AFE issues that are not identified by the respective app developers; even the most widely used apps with billions of active users (such as TAOBAO, WECHAT and ALIPAY) encounter dozens of issues. It is also worth noting that over 60% (19/31) apps have more than 20 AFE issues; the number for ≥10 issues is 77% (24/31). Considering that (1) all the apps we have selected are popular with ageing-fit options/versions which are designed following some ageing-fit guidelines and (2) our efforts in exploring these apps are limited to the major functionalities (Section III) within one hour, we believe that these numbers evidently suggest the severity and high prevalence of AFE issues in real-world ageing-fit mobile apps.

Finding 1: AFE issues are prevalent in ageing-fit mobile apps with >20 issues per app under study, which considerably prevent ageing people from accessing their functionalities.



(a) WeChat: Small Text (b) WeChat: Small Text (c) WeChat: Small Icon (d) Alipay: Missing Prompt

Fig. 5: Example AFE issues in ageing-fit mobile apps

B. Answering RQ2: Types of AFE Issues

To try understanding AFE issues in ageing-fit mobile apps in depth, we manually inspected every AFE issue in AFE-101 and classified them into one of the following types.

Small text. More than half (375/691, 54.3%) AFE issues relate to a "small text" widget, i.e., the displayed text are excessively small. This hinders ageing people from obtaining the information behind the text fluently; and in a worst case, they might miss important information. Figure 5(a) shows a mild example where the widget with text "Free Space" in WECHAT's Storage screen is small; by contrast, the size of the adjacent widget with text "Other" is normal. Figure 5(b) presents a severe example where the "ad" widget is small. In the previous example, ageing users experiencing difficulties in reading "Free Space" due to vision impairments are merely unaware of the available free space. However, if ageing users overlook the "Spons..." in the latter example, they may inadvertently tap one of the four image widgets displaying cars and enter into a new GUI page where advertisers attempt to gather their personal information without proper advertising caution.

Small icon. Similar to *Small Text*, 17.4% (120/691) AFE issues is due to a widget with a "small icon", resulting in accidental taps and difficulties in navigating within the apps. Figure 5(c) shows two such examples within the same GUI page.Due to the small size and close proximity of these two icons, ageing users with motion decline face difficulty in accurately tapping their intended targets, in addition to their difficulty in obtaining the clear information fluently; there is a high likelihood that they tap the incorrect one. This impedes them to interact efficiently with the app. On the other hand, the overly small qrcode icon on the same page also impacts ageing users from showing their qrcode to their friends. This is a key functionality in WECHAT but the app developers missed it.

Missing prompt. Nearly a quarter (168/691, 24.3%) of the AFE issues are because of widgets with missed prompts, stopping ageing users from further exploring the app. Figure 5(d) illustrates an example that all three ageing users for issue confirmation were unable to determine the purpose of the icon. Unlike young users, ageing users, particularly those with cognitive limitations, face challenges in comprehending

the iconography in mobile apps, which hampers their ability to grasp the functionalities that the widgets represent. In the absence of clear navigation prompts near the widgets, widgets displayed as icons can cause confusion, misinterpretation, and unintended interactions within the app. This significantly affects ageing users who may spend excessive amount of time attempting to memorize these widgets through trial and error or seek external helps. Other prompts near the widgets displayed as icons, such as "Transport" and "Pocket", can effectively help ageing people understand the functional meanings behind these icons.

Others. The above three types of AFE issues have occupied a majority (663/691, 96.0%) of AFE issues in AFE-101. The rest include issues causing low color contrast between text and background (16/691, 2.3%) or issues with truncated text (4/691, 0.6%). We draw the type distribution of AFE issues for each app with different colors in Figure 4.

> **Finding 2:** Three issue types cover most (96.0%) AFE issues in AFE-101: *Small Text* (54.3%), *Small Icon* (17.4%), and *Missing Prompt* (24.3%). Among them, *Small Text* and *Small Icon* in aging-fit mobile apps stand out more prominently compared to existing study results [18] in general apps, and *Missing Prompt* is not considered as accessibility issues in general apps before.

C. Answering RQ3: Capability of GASCANNER

Existing techniques for detecting mobile accessibility issues can be broadly categorized into two categories: static analysis and dynamic detection. Static analysis technique requires access to app's source code [34], which limits their applicability to commercial closed-source ageing-fit mobile apps. While there are dynamic detection techniques available for mobile accessibility issues [12], [13], [15], [17], [18], [35], [36], only a few of them are specifically relevant to the AFE issues studied in our research [15], [17], [18]. Additionally, we observed that many of these techniques are based on Google's Accessibility Scanner (GASCANNER) or the underlying engine of GASCANNER, Google's Accessibility Testing Framework [37] . GASCANNER is the official, standard, state-of-the-art tool developed by the Android team for effective detection of general accessibility issues [38]. Therefor, we tried to apply GASCANNER for a test. [3] In particular, GASCANNER scans the GUI pages of an app and provides suggestions accordingly. Therefore, we offered every GUI page in AFE-101 to GASCANNER and checked whether it can detect the AFE issues within the GUI page. The results are shown in Table I. Over the 691 AFE issues, GASCANNER was only able to discover 262 (37.9%), far from a half. This includes 157/375 (41.9%) *Small Text* issues, 93/120 (77.5%) *Small Icon* issues, and 12/28 (42.9%) issues from other types. We hypothesize that this is probably because the rules defined by GASCANNER for general accessibility issues detection do not well-suited to address the unique characteristics of accessibility issues specifically for ageing users. It should also be noted that GASCANNER did not uncover any AFE issue in the *Missing Prompt* type because it does not consider the relationships of different widgets [39].

[3]We only conducted a comparatively easy-to-setup evaluation over GASCANNER in our empirical study, while we made the evaluation far more thorough in our final experiment (Section VI).

TABLE I: Capability of GASCANNER over AFE-101

#	Issue Type	#AFE	Detectable	Undetectable	Recall
1	*Small Text*	375	157	218	41.9%
2	*Small Icon*	120	93	27	77.5%
3	*Missing Prompt*	168	0	168	0.0%
4	*Others*	28	12	16	42.9%
	Total	691	262	429	37.9%

> **Finding 3:** GASCANNER (the state-of-the-art technique) is unable to detect AFE issues effectively in real-world ageing-fit mobile apps: (1) GASCANNER has a low recall on *Small Text* and *Small Icon* AFE issues; (2) AFE issues of *Missing Prompt* are even beyond the capability of GASCANNER.

V. AGEINGDROID: EFFECTIVE DETECTION OF AFE ISSUES

We present AGEINGDROID, an automated tool for effective detection of AFE issues. AGEINGDROID targets the Android platform and is designed only around *Small Text*, *Small Icon*, and *Missing Prompt* that are the most common, affecting the most number of ageing people; we leave others as our important future work.

A. Overview

Figure 6 shows an overview of AGEINGDROID which is composed of two components: (1) `UITraverser` gathers necessary execution information from the ageing-fit mobile app under test; (2) `IssueDetector` analyzes the obtained execution information and detects AFE issues accordingly.

B. UITraverser

`UITraverser` traverses the ageing-fit mobile app under test and obtains necessary execution information for `IssueDetector`. Specifically, `UITraverser` performs a depth-first search to scrape every ageing-fit GUI page of the ageing-fit app under test. During execution, `UITraverser` memorizes every single event that has been executed and records an event sequence for each GUI page. `UITraverser` also saves a screenshot and the associated GUI tree for each GUI page that appears at runtime after it runs into a stable state that all the widgets within no longer change [40]. It should be noted that, for apps with ageing-fit options (rather than a customized ageing-fit version), `UITraverser` pre-executes a predefined script to enable the ageing-fit mode before search.

In addition, `UITraverser` performs the following actions to ensure that the obtained execution information (i.e., the

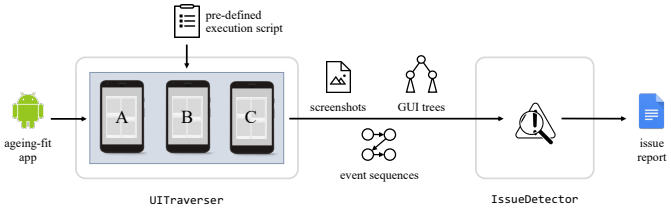


Fig. 6: Overview of AGEINGDROID

event sequence, screenshot, and GUI tree for each GUI page) is comprehensive and reasonable.

- For widgets located at the boundary of a GUI page, `UITraverser` checks whether the GUI page is scrollable (horizontally or vertically) and proactively scrolls the GUI page to avoid obtaining incomplete GUI page (of which at-boundary widgets are only displayed with a portion).
- `UITraverser` is embedded with a heuristic rule to check whether the GUI page is ageing-fit designed. For each traversed GUI page, `UITraverser` checks the GUI page against the rule and discards GUI pages violating the rule. This is because not all GUI pages, even for ageing-fit designed apps, are ageing-fit designed. Specifically, `UITraverser` calculates the average size of all widgets with text/icon on the GUI page and compare it with an experience threshold ($T_{\texttt{ageing}}$).

C. IssueDetector

`IssueDetector` finds different types of AFE issues based on the execution information (i.e., the event sequences, screenshot, and GUI tree for each GUI page) obtained by `UITraverser` and our findings (i.e., the issue characteristics of *Small Text*, *Small Icon*, and *Missing Prompt*) in Section IV.

Small text. `IssueDetector` discovers *Small Text* AFE issues based on the assumption that the number of widgets with overly small text is also very small in the GUI page. For each GUI page, `IssueDetector` scans and calculates the average height $\bar{h}_{\texttt{text}}$ of all the text widgets (i.e., widgets with non-empty text for example `TextView`) in the GUI tree. Then, `IssueDetector` reports an AFE issue if a text widget in the GUI tree whose height is less than $0.5 \cdot \bar{h}_{\texttt{text}}$. Meanwhile, `IssueDetector` marks the widget in the screenshot and emits the event sequence for the GUI page. In order to prevent corner-case situations where most text widgets on a GUI page is excessively small, `IssueDetector` is provided with an experiencing threshold ($T_{\texttt{text}}$) and directly reports an AFE issues for text widgets whose height is less than $T_{\texttt{text}}$, before calculating $\bar{h}_{\texttt{text}}$.

Small icon. `IssueDetector` finds *Small Icon* AFE issues following the same assumption and manner of *Small Text*'s. Specifically, `IssueDetector` compares the height/width of each clickable icon widget (e.g., `ImageView`) against the $0.5\times$ average height/width of all the clickable icon widgets in the GUI page. We also provide experiencing thresholds T_{iconh} and T_{iconw} to avoid corner-cases.

Missing prompt. Due to cognitive limitations, ageing users face difficulty in comprehending the semantics behind icon widgets. Therefore, it is necessary that every icon widgets should be accompanied with a text for prompting. Specifically, for each GUI page, `IssueDetector` scans all the clickable icon widgets in the GUI tree and check whether or not there exists any text close to the clickable widget. We carefully checked hundreds of clickable icons with prompts and found that all prompts are located near the bottom, right side, left side, and in the center of the icons. Therefore, for a clickable icon with boundaries l (left-boundary), t (top-boundary), r (right-boundary), and b (bottom-boundary), `IssueDetector` deems the text widget with boundaries l', t', r', and b' as the prompt, if any of the following are satisfied:

1) The text is located near the bottom of the icon:
$$t' - b > 0 \wedge r' - l > 0 \wedge r - l' > 0 \wedge t' - b < T_{\texttt{prompt}};$$
2) The text is located near the right side of the icon:
$$l' - r > 0 \wedge t' - b < 0 \wedge b' - t > 0 \wedge l' - r < T_{\texttt{prompt}};$$
3) The text is located near the left side of the icon:
$$r' - l < 0 \wedge t' - b < 0 \wedge b' - t > 0 \wedge r' - l > -T_{\texttt{prompt}};$$
4) The text is located in the center of the icon:
$$l - l' < 0 \wedge r - r' > 0 \wedge t - t' < 0 \wedge b - b' > 0.$$

Otherwise, `IssueDetector` reports an *Missing Prompt* AFE issue that the icon widget lacks a prompt, with the icon widget being marked on the screenshot and the event sequence being emitted. Although there are chances that the icon and the nearby text is irrelevant, leading to false positives, our experimental results in Section VI indicates that the number of false positives raised by `IssueDetector` can be handled by a single app developer with negligible effort.

It is worth mentioning that `IssueDetector` is decomposed from `UITraverser` so that `IssueDetector` can be combined with any test runners (e.g., DroidBot [41]) with the ability to obtain event sequences, screenshots, and GUI trees. `UITraverser` only provides a basic implementation. In addition, although `IssueDetector` adopts a simple, rule-based methodology, our evaluation clearly confirmed its effectiveness and efficacy in practical AFE issue detection.

D. Implementation

We implemented AGEINGDROID upon UIAutomator [42] (for taking screenshots and dumping GUI trees) and adb [43] (for executing events). AGEINGDROID does not require access to ageing-fit mobile apps' source code, so it can be applied to both open-source and closed-source Android apps. In addition, we set all thresholds used by AGEINGDROID based on our experience in conducting our empirical study (Section III and Section IV), and hundreds of practical tests: $T_{\texttt{ageing}} = 60$, $T_{\texttt{text}} = 75$, $T_{\texttt{iconh}} = 100$, $T_{\texttt{iconw}} = 100$, and $T_{\texttt{prompt}} = 100$. AGEINGDROID is publicly available on our website.

VI. Evaluation

We have evaluated AgeingDroid on real-world, popular ageing-fit Android apps to answer the following research questions:

RQ4 How effective is AgeingDroid in detecting AFE issues?

RQ5 For false positives and false negatives, what are the root causes of such failures, respectively?

RQ6 How efficient is AgeingDroid in detecting AFE issues?

RQ7 Can AgeingDroid detect new AFE issues outside AFE-101?

A. Experimental Setup

Answering RQ4–RQ6. To evaluate the efficacy of AgeingDroid with respect to AFE-101, we applied `IssueDetector` to each GUI page in AFE-101, involving 663 issue detection tasks for 164 GUIs of 31 popular, commercial ageing-fit Android apps. Note, our evaluation does not take AFE issues with types other than *Small Text*, *Small Icon*, and *Missing Prompt* into consideration because they are out of the scope of AgeingDroid.

To answer RQ4, we recorded every issue that AgeingDroid has reported and manually classified them as one of the following categories:

1) *True positives*, namely issues that are included in AFE-101.
2) *New issues*, namely issues that are not included in AFE-101 but were confirmed as real AFE issues
3) *False positives*, all the rest reported issues.

After that, we calculate the precision and recall of AgeingDroid. A high precision implies that AgeingDroid is able to report as less false alarms to developers as possible while a high recall indicates that AgeingDroid seldom misses any AFE issues.

To answer RQ5, we further collected false negatives, i.e., issues in AFE-101 that are missed by AgeingDroid and manually inspected every false positive/negative to categorize their root causes.

Finally, we traced the time that `IssueDetector` spent analyzing each GUI page to answer RQ6.

Answering RQ7. The second part concerns whether or not AgeingDroid is useful in practical AFE issue detection. Specifically, we are interested in if AgeingDroid can find new AFE issues in other GUI pages of the apps in AFE-101 or even apps outside AFE-101. Hence, we applied AgeingDroid (i.e., `UITraverser` and `IssueDetector`) to 15 popular, commercial ageing-fit Android apps, where 10 apps were randomly selected from AFE-101 and the rest five are new apps (outside AFE-101) from Xiaomi GetApps app market [44]. We ensured that all the selected apps are with ageing-fit options and with $\geq$50 million downloads.

For each app, we manually prepared a predefined script for enabling the ageing-fit mode and started `UITraverser` to traverse the app for an hour. During this process, we ran `IssueDetector` to analyze each obtained GUI page and report suspicious AFE issues.

To answer RQ7, we recorded every reported issue and confirmed one as a new issue if it affects at least one ageing person that we have employed in our empirical study.

Other settings. All experiments were performed on a commodity laptop with Intel i5 processor and 16 GiB RAM running Windows 11 and a physical Redmi Note 11 Pro+ mobile device with 12 GiB RAM running Android 11. For confirming new issues reported by AgeingDroid and/or GAScanner, we employed the same set of ageing users as in our empirical study (Section III).

B. RQ4: AgeingDroid's Effectiveness

Our first experiment aims to measure the effectiveness of AgeingDroid with respect to AFE-101. Hence, we applied `IssueDetector` and GAScanner to analyze each GUI page in AFE-101. Table II and Table III presents the experimental result from per app and per issue-type perspectives, respectively.

Overall, AgeingDroid demonstrated superior performance compared with GAScanner, achieving a precision of 0.963 and a recall of 0.739 for all detection tasks in AFE-101. In contrast, the precision and recall of GAScanner are 0.661 and 0.377, respectively.

For 90.3% (28/31) apps, AgeingDroid's precision is 0.9 or higher. Moreover, for 67.7% (21/31) apps, AgeingDroid achieved a perfect precision of 1.0. Conversely, GAScanner achieved a precision of at least 0.8 for only 29.0% (9/31) apps. Despite the inspiring precision and recall, we noticed lower precision for specific apps such as MeiTuan (22/26, 84.6%), NetEase News (15/18, 83.3%), and E-surfing (5/6, 83.3%). Except for E-surfing, which only has 5 AFE issues within AFE-101, we conducted a careful analysis of the other apps and identified two major reasons for false positives. Firstly, non-standard, customized widgets causes AgeingDroid to fail in obtaining correct coordinates. Secondly, transparent widgets that act as separators among functional widgets also misled AgeingDroid. We will discuss further details of these issues in Section VI-C. Regarding the specific types of AFE issues that we focus on, AgeingDroid achieved a precision of 0.990 for *Small Text*, 0.950 for *Small Icon*, and 0.899 for *Missing Prompt*.

In addition, out of 663 AFE issues, AgeingDroid can recall 490 (73.9%), nearly double of GAScanner (37.7%). Among them, AgeingDroid missed the most number of issues in BankComm (17/24, 70.8%). This is because AgeingDroid was not able to obtain the coordinates of some customized text widgets, failing to determine the prompt of some icon widgets. We discuss them further in Section VI-C. On the other hand, AgeingDroid recalled the most number (302/375, 80.5%) of AFE issues in the *Small Text* type and the least (96/168, 57.1%) in the *Missing Prompt* type.

Compared with AgeingDroid, GAScanner is inferior both in precision and recall. Unfortunately, due to the closed-source nature, we were not able to identify the exact rea-

TABLE II: RQ4: Per-app effectiveness of AGEINGDROID for AFE-101. Columns "#TP", "#FP", "#NI", and "#FN" represent the number of true positives, false positives, new issues, and false negatives, respectively.

#	App Name (Ageing-Fit Mode)	#AFE	AGEINGDROID #TP	#FP	#NI	#FN	Time/ms	GASCANNER #TP	#FP	#NI	#FN
1	NETEASENEWS	21	14	3	1	7	20.9	6	4	0	15
2	HIMALAYAN	27	20	2	0	7	45.6	6	10	0	21
3	IQIYI	24	23	3	5	1	66.5	16	5	2	8
4	QQ MUSIC	4	4	0	0	0	33.0	0	0	0	4
5	DOUYIN	31	29	2	0	2	63.5	15	4	0	16
6	HAODF	12	10	0	0	2	14.9	7	1	0	5
7	SINA-WEIBO	35	27	1	0	3	42.8	13	2	0	17
8	IDLEFISH	35	19	0	0	16	31.1	8	6	0	27
9	ELE.ME	24	15	0	0	9	29.8	10	3	0	14
10	WECHAT	35	21	0	0	14	19.2	10	5	0	25
11	MEITUAN	32	22	4	0	10	22.7	13	2	0	19
12	JINGDONG	21	17	0	0	4	33.5	8	3	0	13
13	TAOBAO	44	33	0	0	11	14.8	11	5	0	33
14	QQ	40	36	1	0	4	49.5	17	5	0	23
15	YOUKU	9	7	0	0	2	19.0	3	1	0	6
16	12306	3	3	0	0	0	48.0	3	1	0	0
17	WESING	23	18	0	0	5	20.4	6	2	0	17
18	DIANPING	5	3	0	0	2	66.0	2	16	0	3
19	E-SURFING	5	5	1	0	0	33.0	0	0	0	5
20	PINDUODUO	22	14	0	0	8	23.2	7	35	0	15
21	HOMELINK	9	6	0	0	3	21.0	6	2	0	3
22	VIPSHOP	37	26	0	0	11	37.3	21	2	0	15
23	BANKCOMM	24	7	1	0	17	71.0	7	0	0	17
24	CHINAMOBILE	32	29	0	0	3	33.1	16	3	0	16
25	ALIPAY	35	22	0	0	13	9.3	11	4	0	24
26	BANKBOC	9	7	0	0	2	32.0	5	4	0	4
27	QUICKPASS	26	19	1	0	7	23.4	3	1	0	23
28	BANKICBC	12	9	0	0	3	22.0	3	3	0	9
29	BANKABC	13	10	0	0	3	15.3	8	0	0	5
30	BEKE	8	5	0	0	3	16.5	4	0	0	4
31	WUBA	11	10	0	3	1	56.7	5	1	2	6
	Total	663	490	19	9	173	-	250	130	4	413
	Statistics	-	pre=0.963		rec=0.739		ave=32.3	pre=0.661		rec=0.377	

TABLE III: RQ4: Per-category effectiveness of AGEINGDROID for AFE-101. Columns "#TP", "#FP", "#NI", and "#FN" represent the number of true positives, false positives, new issues, and false negatives, respectively.

Category	#AFE	AGEINGDROID #TP	#FP	#NI	#FN	Time/ms	GASCANNER #TP	#FP	#NI	#FN
Small Text	375	302	3	3	63	509	157	72	1	218
Small Icon	120	92	5	4	28	730	93	58	3	27
Missing Prompt	168	96	11	2	72	1,769	0	0	0	168
Total	663	490	19	9	173	3,008	250	130	4	413
Statisties	-	pre=0.963		rec=0.739		ave= 1,002,7	pre=0.661		rec=0.377	

sons behind the low precision and recall. Nevertheless, after carefully analyzing the issues reported by GASCANNER, we hypothesize that the deficiencies may be attributed to the following factors: (1) The rules defined by GASCANNER for detecting general accessibility issues may not effectively address ageing-fit issues. (2) GASCANNER lack specific rules for AFE issues with *Missing Prompt*.

Finally, AGEINGDROID was able to find nine additional, new issues that are missed in our empirical study (Section III), while GASCANNER found only four of them. Specifically, these issues include 5 *Small Icon* issues and 4 *Small Text* issues of two apps.

C. RQ5: Failure Reasons

Despite the impressive precision/recall of AGEINGDROID, `IssueDetector` still incorrectly reported 19 issues and missed 173 issues in AFE-101.

Failure reasons for false positives.

- *Customized widgets* (12/19, 63.2%). Popular apps typically develop customized widgets for better graphics while the developers may not adhere to the Accessibility Test Framework [37]. This results in the absence or inaccuracy of the retrieved information (e.g., coordinates, shapes) for these customized widgets in the GUI tree when using UIAutomator [42]. As AGEINGDROID heavily relies on precise measurements of width, height,

and positioning for AFE issue detection, such incorrect information poses considerable challenges in issue detection, inducing the most number of false positives in our evaluation.

- *Transparent widgets* (7/19, 36.8%). In addition to customized widgets, transparent widgets are also commonly used. As the name implies, such widgets remain hidden to users and act as separators to clearly divide functional widgets in the GUI page. However, the current AGEINGDROID is unable to differentiate transparent widgets from functional widgets and consequently flagged small transparent widgets as widgets with AFE issues.

Failure reasons for false negatives.

- *Customized widgets* (79/173, 45.7%). This is also a major source of false negatives. On one hand, customized widgets providing incorrect information makes it difficult for AGEINGDROID to find the prompt widget for some icon widgets. On the other hand, customized widgets of which the `importantForAccessibility` attribute is `no` or `noHideDescendants` may not show in the GUI tree and thus the widget escapes the analysis of AGEINGDROID.
- *Semantics unawareness* (30/173, 17.3%). When it comes to the semantics of the widgets, AGEINGDROID is not sensitive enough to determine whether a nearby text widget is actually the prompt for the analyzed icon widget. Consequently, there is a possibility that AGEINGDROID may identify an unrelated adjacent text widget as the prompt, resulting in false negatives.
- *Implementation limitations* (64/173,37.0%). Finally, the simple rule-based implementation of AGEINGDROID has its own limitations: (1) The current AGEINGDROID mainly focus on operational (i.e., clickable) icon widgets and cannot analyze icon widgets that are used merely for displaying app-specific information (51/64). (2) Limited by UIAutomator, the current AGEINGDROID cannot analyze GUI pages empowered by `WebView` (12/64). (3) Even though AGEINGDROID sets thresholds set based on our experience in our empirical study and hundreds of attempts, they do not fit for all situations (1/64).

D. RQ6: AGEINGDROID's Efficiency

Our evaluation also concerns the efficiency of AGEINGDROID. Since `IssueDetector` is decomposed from `UITraverser` and can be combined with any test runners equipped with GUI traversal capabilities (as discussed in Section V-C), we specifically examine the effectiveness of `IssueDetector` in this experiment. The experimental results are displayed in the "Time/ms" column of Table II and Table III.

Notably, `IssueDetector` is capable of analyzing a GUI page and identifying suspicious AFE issues in an average time of ∼32.3ms. For more than 20% (7/31) apps, `IssueDetector` completes its analysis within 20ms. We also observed that analyzing BANKCOMM costs ∼70ms. After manual inspection, we found that the GUI pages of BANKCOMM is more complicated than other apps, with a higher number of clickable icon widgets per GUI page, resulting in more analysis time.

As for the issue types that the current `IssueDetector` focuses on. Type *Missing Prompt* requires the most time, while *Small Text* and *Small Icon* exhibit comparable overhead. This is reasonable as the process of identifying prompt widgets for clickable icon widgets involves searching in the vicinity, in addition to GUI traversal. On the other hand, `IssueDetector` only needs to examine the widget itself to identify *Small Text* and *Smal Icon* issues.

E. RQ7: Applying AGEINGDROID in Practice

The combination of impressive precision, recall, and negligible time cost suggests that AGEINGDROID has promising potential to provide accurate AFE issue reports while minimizing false positives and negatives. Thus, we examine the practical usefulness of AGEINGDROID by applying it to 15 apps. 10 apps (#1–#10) were randomly selected from AFE-101 and the rest five (#11–#15) are new apps outside AFE-101 from Xiaomi GetApps app market [44]. Table IV presents the information of the 15 apps and experimental results.

Particularly, AGEINGDROID has detected a total of 336 new AFE issues spanning 87 GUI pages of the 15 apps under evaluation within an average time of ∼44.9ms per GUI page. Out of the issues, 324 (96.4%) were confirmed by at least one ageing person that we have recruited in our empirical study. For evaluation, we deliberately adopt this "at-least-one" criterion instead of the stricter majority-vote rule (e.g., requiring two out of three confirmations as in our empirical study), because the latter could wrongly exclude issues that are perceived by only a single ageing person but still represent genuine accessibility barriers. It is worth noting that *all* the reported AFE issues and the affected GUI pages in this experiment are not included in AFE-101. We believe that this clearly demonstrates the usefulness of AGEINGDROID in practical AFE issue detection.

On the other hand, per app is affected by at least 20 AFE issues, ranging from 2 to 48. Among all the confirmed issues, 102/324 (31.5%) are found in the five apps outside AFE-101, where AGEINGDROID disclosed the most number of AFE issues in TENCENT-NEWS and TOUTIAO, two news apps that are extensively popular among ageing people. For apps within AFE-101, ALIPAY–one of the most widely used financial app in the world–is affected the most seriously. Given that `UITraverser` was only executed for one hour where a quantities of GUI pages were not covered, this evidently implies the severity of AFE issues and that the needs of ageing users are far from met in practice.

F. Discussions

***Limitations of* AGEINGDROID**. We designed AGEINGDROID following a rule-based methodology closely around *Small Text*, *Small Icon*, and *Missing Prompt*. Thus, AGEINGDROID is unable to detect AFE issues beyond these types. Furthermore, the semantic unawareness nature of AGEINGDROID makes it

TABLE IV: RQ7: Usefulness of AGEINGDROID in practice. Columns "ST", "SI", and "MP" represent *Small Text*, *Small Icon*, and *Missing Prompt*. Column "#FP" and numbers within parenthesis is the number of false positives. Column "#GUI" displays the number of GUI pages that have been traversed.

#	**App Name** (Ageing-Fit Mode)	**Cate.**	**Down.**	**#AFE**	**ST**	**SI**	**MP**	**#FP**	**#GUI**	**Time/ms**
1	WECHAT	Comm.	7.3B	30	13	8	9	0	11	30.0
2	ALIPAY	Fina.	5.7B	48	34	14	0	0	11	20.1
3	NETEASE NEWS	Media	580M	6	5	0	1	0	5	44.3
4	HIMALAYAN	Ente.	330M	38	30 (1)	6	2	1	5	63.2
5	HAODF	Heal.	68M	2	2	0	0	0	1	71.7
6	TAOBAO	Shop.	7.2B	26	16	9	1 (1)	1	10	24.9
7	BANKBOC	Fina.	9.0B	34	15	10	9	0	7	36.7
8	QUICKPASS	Fina.	520M	7	5 (2)	1	1	2	4	28.1
9	WESING	Ente.	120M	12	10	2	0	0	5	36.7
10	WUBA	Serv.	220M	28	23 (2)	2	3	2	6	31.6
11	*Learningpower*	Educ.	430M	25	12	10	3	0	5	49.6
12	ALLINWAN	Offi.	84M	1	1	0	0	0	1	72.0
13	TOUTIAO	Media	7.4B	34	23 (2)	4	7	2	6	36.2
14	TENCENT-NEWS	Media	2.8B	38	26 (1)	5 (1)	7 (2)	4	8	48.7
15	CAINIAO	Tool	540M	10	4	3	3	0	2	80.0
-	**Total**	-	-	336	117 (8)	74 (1)	45 (3)	12	87	`ave`=44.9

incapable of determining the semantics/types of different widgets. We are also working on equipping `IssueDetector` with the ability of automatic widget detection and comprehension by deep learning models.

Threats to validity. This paper is based on 691 AFE issues from 31 popular commercial Android apps, which, although encompasses a significant number of issues, may not fully represent all possible AFE issues encountered in practice. We mitigated this threat by collecting a range of popular commercial Android apps across various categories. Another potential concern is the validity and bias of the collected AFE issues. To mitigate the threat, we employed three senior students and 34 ageing users who are irrelevant to this project to respectively identify AFE candidates and confirm AFE issues. To be included in AFE-101, we required that each issue has to be examined by three ageing users, and at least two of them confirm that it is indeed an AFE issue.

VII. RELATED WORK

Understanding accessibility issues in mobile apps. Gaining a comprehensive understanding of accessibility issues in mobile apps is crucial before tackling them. Some research work concerned about the compatibility and privacy issues in mobile apps when accessibility assistance services are enabled. Alshayban et al. [10] studied five types of text accessibility issues when the Text Scaling Assistive Service is enabled. Mehralian et al. [32] investigated the issue of overly accessible elements in Android apps. Ross et al. [45] examined label-based accessibility barriers in three classes of image-based buttons. In addition, some research work also conducted empirical studies on the accessibility issues in mobile app design. Alshayban et al. [18] studied the prevalence, root causes, and user feedback regarding accessibility issues in Android apps. Díaz-Bossini et al. [3] evaluated the accessibility of two mobile apps involving ageing users.Some research work also analyzes accessibility issues in mobile apps from the perspectives of developers and users. Bi et al. [46] analyzed practitioners' perspectives on accessibility development and design.Reyes et al. [47] studied how accessibility issues are expressed in user reviews.Yan et al. [48] evaluated the status of accessibility in mobile apps by investigating the GUI structures and conformance to accessibility guidelines.

The above research work understand accessibility issues in mobile apps from various aspects. However, none of the above-mentioned work systematically investigate AFE issues in ageing-fit mobile apps, a new mobile app mode designed for ageing people.

Detecting accessibility issues in mobile apps. Several tools and techniques have been developed to detect accessibility issues in mobile apps. MATE [12] checks whether user interface components have labels that can be used by screen readers. Latte [13] repurposes existing tests for functional correctness to assess the accessibility of mobile apps. AccessiText [10] detects five types of text accessibility issues resulting from incompatibility between apps and Text Scaling Assistive Service. OverSight [32] identifies overly accessible elements using a set of conditions that can result in over-access problem and dynamically verify their accessibility using an assistive technology. Groundhog [14] assesses the functionality of UI elements with and without assistive service TalkBack. ScaleFix [49] repairs UI scaling accessibility issues in mobile apps. dVermin [50] detects the inconsistency of a view under the default and a larger display scale. Mehralian et al. [51] presented TIMESTUMP, an automated framework that leverages our findings in the formative study to detect accessibility issues regarding dynamic changes. Gu et al. [52] proposed an automated approach for repairing color-related accessibility issues in Android apps. Zhang et al. [53] proposed VisualDroid, a method based on a multi-modal large language model for testing and classifying GUI visual changes using a tailored three-hop reasoning prompting framework.

For the work on detecting AFE issues in ageing-fit mobile

apps, GASCANNER [38] is the most widely used tool and can detect two main types of AFE issues *Small Text* and *Small Icon* with low rates in precision and recall. What's more, AFE issues of Missing Prompt are beyond the capability of GASCANNER. Compared with GASCANNER, the detection tool AGEINGDROID we proposed can detect all three main types of AFE issues with high rates both in precision and recall.

VIII. CONCLUSION

This paper presents the comprehensive empirical study on accessibility issues in ageing-fit mobile apps, involving 691 real-world AFE issues collected from 31 popular commercial mobile apps. Our study provides empirical evidence confirming the severity of these AFE issues and identifies three common types. Based on these findings, we have developed AGEINGDROID, an effective tool for detecting AFE issues in practice, and evaluated it using 36 popular commercial mobile apps. The experimental evaluation demonstrates that AGEINGDROID achieved high precision and recall in detecting AFE issues and successfully identified previously unknown issues.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for their constructive feedback on this paper. This work is supported in part by the National Natural Science Foundation of China (Grant Nos. 92582201, 62372219, and 62402229). Cong Li is the corresponding author.

REFERENCES

[1] WHO, "Ageing and health." https://www.who.int/news-room/fact-sheets/detail/ageing-and-health, 2023.

[2] Statista, "Share of adults in the united states who owned a smartphone from 2015 to 2021, by age group." https://www.cnet.com/tech/mobile/older-people-are-using-more-smartphones-pew-finds/, 2023.

[3] J.-M. Díaz-Bossini, L. Moreno, and P. Martínez, "Towards mobile accessibility for older people: A user centered evaluation," in *Proceedings of the 8th International Conference on Universal Access in Human-Computer Interaction. Aging and Assistive Environments - Volume 8515*. Berlin, Heidelberg: Springer-Verlag, 2014, p. 58–68.

[4] K. Kalimullah and D. Sushmitha, "Influence of design elements in mobile applications on user experience of elderly people," *Procedia computer science*, vol. 113, pp. 352–359, 2017.

[5] A. D. A. Oliveira and M. M. Eler, "Exploring accessibility of mobile applications through user feedback: insights from app reviews in a systematic literature review," in *Proceedings of the XXIII Brazilian symposium on human factors in computing systems*, 2024, pp. 1–15.

[6] S. Jain, S. F. Huq, Z. He, and S. Malek, "Automated detection of web application navigation barriers for screen reader users," in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2025, pp. 1906–1918.

[7] S. Wickramathilaka, J. Grundy, K. Madampe, and O. Haggag, "Adaptive and accessible user interfaces for seniors through model-driven engineering," *Automated Software Engineering*, vol. 32, no. 2, 2025.

[8] Android, "Text scaling." https://support.google.com/accessibility/android/answer/12159181?, 2023.

[9] Google, "Get started on android with talkback." https://support.google.com/accessibility/android/answer/6283677?, 2023.

[10] A. Alshayban and S. Malek, "Accessitext: Automated detection of text accessibility issues in android apps," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 984–995.

[11] C. Vendome, D. Solano, S. Liñán, and M. Linares-Vásquez, "Can everyone use my app? an empirical study on accessibility in android apps," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 41–52.

[12] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser, "Automated accessibility testing of mobile apps," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 2018, pp. 116–126.

[13] N. Salehnamadi, A. Alshayban, J.-W. Lin, I. Ahmed, S. Branham, and S. Malek, "Latte: Use-case and assistive-service driven automated accessibility testing framework for android," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI'21. New York, NY, USA: Association for Computing Machinery, 2021.

[14] N. Salehnamadi, F. Mehralian, and S. Malek, "Groundhog: An automated accessibility crawler for mobile apps," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'22. New York, NY, USA: Association for Computing Machinery, 2023.

[15] A. S. Alotaibi, P. T. Chiou, and W. G. Halfond, "Automated repair of size-based inaccessibility issues in mobile applications," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 730–742.

[16] J. Huang, M. Backes, and S. Bugiel, "A11y and privacy don't have to be mutually exclusive: Constraining accessibility service misuse on android," in *USENIX Security Symposium*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235222461

[17] A. S. Ross, X. Zhang, J. Fogarty, and J. O. Wobbrock, "Epidemiology as a framework for large-scale mobile application accessibility assessment," in *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 2–11.

[18] A. Alshayban, I. Ahmed, and S. Malek, "Accessibility issues in android apps: State of affairs, sentiments, and ways forward," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1323–1334.

[19] C. C. Association, "Research report on the consumption supervision and evaluation of the ageing-fit apps." https://www.cca.cn/jmxf/detail/30541.html, 2023.

[20] W3C, "Older users and accessibility." https://www.w3.org/WAI/older-users/, 2023.

[21] B. of Apps, "Things to be taken care of when designing apps for the elderly." https://www.w3.org/WAI/older-users/, 2023.

[22] L. Paez and C. Zapata, *Elderly Users and Their Main Challenges Usability with Mobile Applications: A Systematic Review*, 07 2019, pp. 423–438.

[23] E. C. Project, "Wai-age project (ist 035015)," https://www.w3.org/WAI/WAI-AGE/, 2010.

[24] W3C, "Developing websites for older people: How web content accessibility guidelines (wcag) 2.0 applies," https://www.w3.org/WAI/older-users/developing/, 2010.

[25] M. of Industry and I. T. of PRC, "Notice on further grasping the implementation of the special action of internet applications with ageing-fit and accessibility support," https://www.gov.cn/zhengce/zhengceku/2021-04/13/content_5599225.htm, 2021.

[26] AppsFlyer, "In-app advertising done right – the complete guide," https://appsflyer.com/resources/guides/in-app-advertising/, 2023.

[27] TINA.org, "Six deceptive ads targeting seniors," https://truthinadvertising.org/articles/six-deceptive-ads-targeting-seniors/, 2014.

[28] F. Dong, H. Wang, L. Li, Y. Guo, G. Xu, and S. Zhang, "How do mobile apps violate the behavioral policy of advertisement libraries?" in *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications*, ser. HotMobile '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 75–80.

[29] StatCounter, "Mobile operating system market share worldwide." https://gs.statcounter.com/os-market-share/mobile/worldwide, 2023.

[30] Clutch, "Designing apps for elderly smartphone users." https://clutch.co/app-development/resources/designing-apps-for-elderly-smartphone-users, 2023.

[31] K. Schaie and S. Willis, *Handbook of the Psychology of Aging (8th ed.)*, 2016.

[32] F. Mehralian, N. Salehnamadi, S. F. Huq, and S. Malek, "Too much accessibility is harmful! automated detection and analysis of overly accessible elements in mobile apps," 01 2023, pp. 1–13.

[33] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, G. Li, and J. Wang, "Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 322–334.

[34] Android, "Android lint." http://tools.android.com/tips/lint, 2023.

[35] A. S. Alotaibi, P. T. Chiou, and W. G. J. Halfond, "Automated detection of talkback interactive accessibility failures in android applications," *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 232–243, 2022.

[36] R. Fok, M. Zhong, A. Ross, J. Fogarty, and J. Wobbrock, "A large-scale longitudinal analysis of missing label accessibility failures in android apps," 04 2022, pp. 1–16.

[37] Google, "Accessibility test framework for android." https://github.com/google/Accessibility-Test-Framework-for-Android, 2023.

[38] Android, "Accessibility scanner." https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility, 2023.

[39] Google, "Android accessibility help." https://support.google.com/accessibility/android/answer/6376570?hl=en, 2023.

[40] S. Feng, M. Xie, and C. Chen, "Efficiency matters: Speeding up automated testing with gui rendering inference," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 906–918.

[41] Y. Li, Z. Yang, Y. Guo, and X. Chen, "Droidbot: a lightweight ui-guided test input generator for android," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 23–26.

[42] Android, "UIAutomator," https://developer.android.com/training/testing/ui-automator, 2023.

[43] ——, "Android Debug Bridge (adb)," https://developer.android.com/studio/command-line/adb, 2023.

[44] Xiaomi, "Getapps." https://global.app.mi.com, 2023.

[45] A. S. Ross, X. Zhang, J. Fogarty, and J. O. Wobbrock, "Examining image-based button labeling for accessibility in android apps through large-scale analysis," in *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 119–130.

[46] T. Bi, X. Xia, D. Lo, J. Grundy, T. Zimmermann, and D. Ford, "Accessibility in software practice: A practitioner's perspective," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 4, pp. 1–26, 2022.

[47] J. E. Reyes Arias, K. Kurtzhall, D. Pham, M. W. Mkaouer, and Y. N. Elglaly, "Accessibility feedback in mobile application reviews: A dataset of reviews and accessibility guidelines," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7.

[48] S. Yan and P. Ramachandran, "The current status of accessibility in mobile apps," *ACM Transactions on Accessible Computing (TACCESS)*, vol. 12, no. 1, pp. 1–31, 2019.

[49] A. S. Alotaibi, P. T. Chiou, F. M. Tawsif, and W. J. Halfond, "Scalefix: An automated repair of ui scaling accessibility issues in android applications," in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2023, pp. 147–159.

[50] Y. Su, C. Chen, J. Wang, Z. Liu, D. Wang, S. Li, and Q. Wang, "The metamorphosis: Automatic detection of scaling issues for mobile apps," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023.

[51] F. Mehralian, Z. He, and S. Malek, "Automated accessibility analysis of dynamic content changes on mobile apps," 2025.

[52] J. Gu and H. Huang, "Characterizing and repairing color-related accessibility issues in android apps," in *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2025, pp. 2033–2045.

[53] M. Zhang, J. Yu, C. Xu, J. Li, X. Yin, and h. liu, "Towards testing the accessibility of dynamic visual changes in android mobile gui with multi-modal llms," *ACM Transactions on Computer-Human Interaction*, 2026.