Proceedings of the 2nd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/73/20240413

# Empirical study on functional defects in multi-window Android systems

**Sisi Chen**

Nanjing University of Information Science and Technology, 219, Ningliu Road, Nanjing, Jiangsu, China

Abdeparture@gmail.com

**Abstract.** This study focuses on the performance, user experience, and security issues of mobile applications in the multi-window mode of the Android operating system. Using automated testing tools, the research evaluates the functionality of different types of applications in multi-window mode. The results indicate that most applications can run in the floating window and split-screen modes, while the picture-in-picture mode is primarily used in video and conferencing applications. However, the floating window mode has multiple window limitations, the picture-in-picture mode has functional restrictions, and the split-screen mode encounters interface adaptation issues. Security-related applications employ a black screen protection mechanism, but face limitations in screenshot operations. The researchers advocate for further research and improvements in the multi-window mode to enhance user experience and security.

**Keywords:** Android, Multi-window, GUI, Functional Testing

## 1. Introduction

In recent years, with the widespread use of multifunctional large-screen devices and the continuous pursuit of diversified experiences by users, operating systems have played an increasingly vital role in providing multi-window collaborative modes. Android, as a crucial component of a comprehensive intelligent ecosystem, thoroughly considers the role of mobile applications in the Internet of Things. Its uniqueness lies in its compatibility with various multi-window modes, including picture-in-picture (PIP), freeform (FF), and screensplit (SS) modes, which not only provide users with a more convenient usage experience but also create more diverse application scenarios for developers.

However, while the introduction of multi-window modes greatly enhances user operational experiences and application diversity, it also brings about certain anomalies and defects. Specifically, we focus on two categories of issues: firstly, front-end design anomalies, including misalignment of application interfaces and adaptation issues under multi-window mode; secondly, runtime anomalies, particularly those that may result in black screen situations. The existence of these issues not only impacts user experience but may also pose serious security risks for specific domains such as finance and security.

Building upon existing work, this paper aims to fill these research gaps by conducting a series of empirical studies to delve into the abnormal issues of mobile applications in the multi-window mode of the Android system. Specifically, we collect a series of typical Android system application software to construct our research samples. Subsequently, we use event traversal scripts to simulate three different

DOI: 10.54254/2755-2721/73/20240413

multi-window modes in the Android system, namely PIP, FF, and SS modes. Finally, we detect front-end design anomalies for ordinary applications and evaluate the black screen protection mechanism for special categories of applications, such as banking applications that are security-sensitive.

Through these research steps, our objective is to provide in-depth understanding and analysis of the abnormal issues of mobile applications under the multi-window mode of the Android system, offering valuable information to developers and researchers on how to improve application design and ensure security. The results of this study not only contribute to optimizing the application experience of the Android system in multi-window scenarios but also provide robust support for the further development of intelligent ecosystems.

## 2. Multi-window in Android

Android is an open-source system designed for mobile devices. To facilitate the development of the Android system and its application ecosystem, the Android Open-Source Project (AOSP) was established by the Open Handset Alliance led by Google. AOSP incorporates three different multi-view modes. In the picture-in-picture (PIP) mode, users can simultaneously display a small window of an application on the screen, allowing them to perform other operations without interrupting the primary task. In the freeform (FF) mode, application windows can be freely resized and repositioned, providing users with greater flexibility to adapt to different work scenarios. Additionally, the split-screen (SS) mode divides the screen into multiple areas, each of which can accommodate different applications simultaneously, enabling users to process multiple tasks or content in parallel.

However, the introduction of multi-window modes presents challenges on both user experience and security. The simultaneous operation of multiple applications on one screen may result in interface layout conflicts, resource competition, and performance issues. Addressing these challenges requires a comprehensive consideration of design, technology, and user expectations to ensure effective implementation of multi-window interaction in the Android system.

## 3. Approach

### *3.1. Study Design*

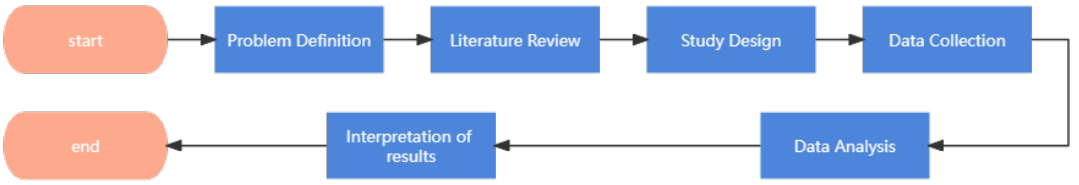


**Figure 1.** Workflow of the Experimental Study

**Experimental Tools**

UIAutomator2 and Weditor are two crucial tools for the Android platform, utilized for automated testing and UI element positioning. UIAutomator2, provided by Google, is an automated testing framework enabling developers to script in Python for the automated execution of Android application tests. Weditor is a UI element analysis tool typically used with UIAutomator2. It provides a visual interface that aids developers in identifying and analyzing UI elements of applications, simplifying the scripting of test scripts.

**Experimental Objectives**

The primary objective of this experiment is to assess the performance and user experience of different applications (Apps) in FF mode, SS mode, and PIP mode within the Android operating system. Additionally, the study evaluates the impact of the black screen protection mechanism in multi-window mode on security-related applications, such as banking apps, and other applications.

DOI: 10.54254/2755-2721/73/20240413

### *3.2. Implementation*

**Experimental Devices and Environment**

The testing was conducted on an oppo reno10 Android13 smartphone, with the test scripts running on a computer equipped with the following specifications: CPU: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (12 CPUs), ~2.6GHz; Memory: 32768MB RAM. Furthermore, the experiment involved the installation of the Python environment with the UIAutomator2 library and Weditor, effective ATX-Agent connection testing, and the installation of the target applications.

**Experimental Subjects**

In the selection of experimental subjects, we refer to the software store rankings, which is a common standard for verifying application popularity. We try to choose apps with a high-user-base to facilitate the identification of potential front-end design anomalies in commonly used apps.

Furthermore, we consider different types of applications, such as social media, video playback, instant conferencing, online payments, etc. This help to ensure diversity by the inclusion of applications from different domains Additionally, attention was paid to the source of the applications, we choose apps from different countries

**Table 1.** Performance of Tested Apps in Various App Stores

| App | App Store (Downloads) | Apple Store (Rating Count) | Google Play (Rating Count) |
|---|---|---|---|
| Tencent Video | 159.3 billion | 21,539,438 | / |
| Bilibili | 32.6 billion | 4,637,067 | / |
| YouTube | / | 961,131 | 140,806,191 |
| PayPal | / | / | 2,971,113 |
| Alipay | 122.8 billion | 1,039,992 | 73,368 |
| Zoom | / | 137,436 | 3,871,383 |
| Tencent Meeting | 7.1 billion | 216,317 | / |
| QQ | 132.3 billion | 2,127,362 | 211,228 |
| WeChat | 119.6 billion | 6,963,399 | / |
| X | / | 422,499 | 20,625,008 |

## 4. Evaluation

### *4.1. Experimental Procedure*

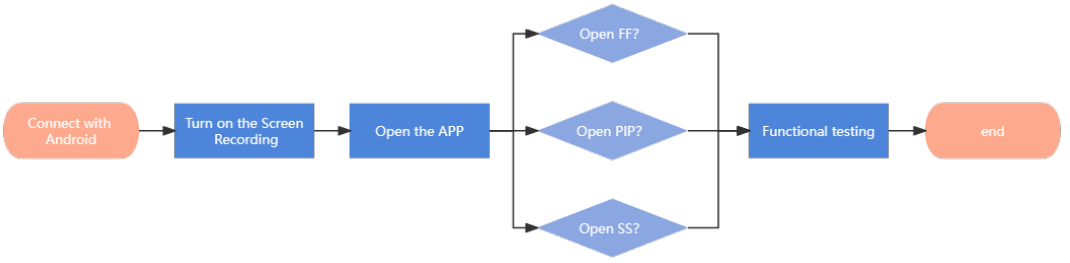


**Figure 2.** Design of the Experimental Procedure.

Step 1: Installation of the UIAutomator2 library using Python and deployment onto the test smartphone. Activating the screen recording function. Additionally, ensuring that the ATX-Agent is correctly installed and testing the connection status.

DOI: 10.54254/2755-2721/73/20240413

```
# Importing the UIAutomator2 library
import uiautomator2 as u2

def connect(self, device_name="(device serial number)"):
    self.device_name = device_name
    self.device = u2.connect(self.device_name)
    print("Connection successful")
```

```
def start_syst_screenRecord(self, video_name="test"):
    try:
        self.device.swipe(1065, 290, 730, 290)
        self.device(description='Screen recording').click()
self.device(resourceId="com.oplus.screenrecorder:id/iv_status_view").click()
    except Exception:
        print("Screen recording button not found")
```

Step 2: Opening the test App.

Step 3: Testing whether the three multi-window functions can be initiated successfully.

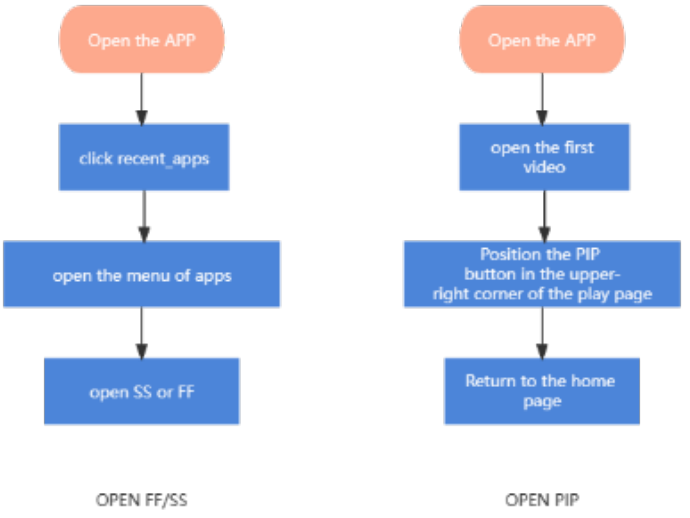


**Figure 3.** Launch Process of Multi-Window Mode

**Table 2.** Testing of Apps in Three Multi-window Modes

| Tested Apps | | FF | PIP | SS |
|---|---|---|---|---|
| Video Apps | Tencent Video | Y | Y | Y |
| | Bilibili | Y | Y | Y |
| | YouTube | N | N | Y |
| Payment Apps | PayPal | Y | N | N |
| | Alipay | Y | N | Y |
| Meeting Apps | Zoom | Y | Y | Y |
| | Tencent Meeting | Y | Y | Y |
| Social Apps | QQ | Y | N | Y |
| | WeChat | Y | N | Y |
| | X | Y | N | Y |

DOI: 10.54254/2755-2721/73/20240413

Step 4: Comparative functional testing of applications in multi-window mode.

1. Interface Adaptability: Does the application's interface layout differ in multi-window mode compared to the regular mode? Can it effectively adapt to different screen sizes and resolutions?

2. Functionality Accessibility: Are certain functions disabled or restricted in multi-window mode? For instance, certain applications might limit the use of specific functions in the floating window or split-screen mode.

3. User Experience: Do users have different experiences in regular and multi-window modes? Are there any additional conveniences or limitations?

**Table 3.** Interface and Functionality Accessibility of Tested Apps

| Test Cases | | FF | PIP | SS |
|---|---|---|---|---|
| Video Apps | Interface Layout | Y | Y | N |
| | Basic Player Functions | Y | Y | Y |
| | Advanced Player Settings | Y | N | Y |
| Payment Apps | Interface Layout | Y | / | N |
| | Payment Function | Y | / | Y |
| | Balance Inquiry | Y | / | Y |
| Meeting Apps | Interface Layout | Y | Y | N |
| | Video/Audio Switch | Y | Y | Y |
| | Advanced Meeting Functions | Y | N | Y |
| Social Apps | Interface Layout | Y | / | N |
| | Send/Receive Messages | Y | / | Y |

### *4.2. Result and Analysis*

1. Applicability of Multi-Window Interfaces, Functionality, and User Experience

From Table 2, it is evident that the majority of applications can successfully initiate the floating window mode and split-screen mode in multi-window mode. Comparatively, the picture-in-picture mode is primarily utilized in video and meeting applications. This is due to the necessity of displaying video content or meeting screens simultaneously on the screen, with the picture-in-picture mode facilitating the visibility of video or meeting windows while performing other tasks.

Table 3 reveals that the interface issues in the FF mode mainly arise from the overlap between the FF small window and the main interface, causing obstructions to certain functions and inadvertent touches. Additionally, conflicts occur between common operations, such as the back button, among multiple windows in the FF mode. The issues in the picture-in-picture mode primarily stem from functional restrictions. In this mode, most applications limit the operational functions, such as bilibili and Tencent Video players, typically retaining only basic functions like play and pause while omitting advanced functions such as speed adjustment, bullet screen display, volume adjustment, and quality selection. This may restrict user interaction capabilities and functional choices. The issues in the split-screen mode primarily pertain to interface adaptation. Test results indicate that the majority of applications do not undergo page adaptation in the split-screen mode, leading to misalignment of interface elements and abnormal displays. Furthermore, the split-screen mode currently does not support screen rotation, potentially causing inconvenience for users when switching screen orientations.

In the multi-window mode, we observed functional conflicts in meeting apps. When users engage in multiple meetings, discussions, or use other functions simultaneously, the sound of the second application may be preempted, rendering the user unable to hear sound, thereby impacting the normal user experience. Conflicts also arise from a range of external devices. For instance, when an app in multi-window mode activates the camera function, opening another application may lead to the camera image being preempted.

DOI: 10.54254/2755-2721/73/20240413

2. Black Screen Protection Issue in Multi-Window

During screen recording operations, security-related applications demonstrate proactive black screen protection mechanisms. This implies that when users conduct screen recording or screenshots in multi-window mode, these applications can identify potential sensitive information and promptly initiate black screen protection. This measure plays a crucial role in maintaining the security and privacy of user data, without hindering the screen recording operation. This proactive security feature helps ensure the confidentiality of user data.

However, in the multi-window environment, this black screen protection may either be insufficient or excessive, consequently affecting various aspects such as multi-window interfaces, functionality, and user experience. Some applications, such as Alipay, expand the scope of protection during black screen protection, obscuring main screen functions and affecting user experience. On the other hand, some security-sensitive applications, such as banking apps, do not provide sufficient black screen protection, resulting in a failure to black out in a timely manner.

In this regard, we recommend further research and improvement of the screenshot mechanism to better balance security and user experience. One possible approach is to transform the screenshot restriction strategy into a black screen protection mechanism similar to that of screen recording operations. This implies that in multi-window mode, applications can identify potential sensitive information and apply black screen protection during screenshots, instead of completely blocking the screenshot operation. This would help ensure the security of user data while allowing users the flexibility to take screenshots.

### *4.3. Limitations*

This study was conducted using only one smartphone, limiting our comprehensive understanding of the multi-window performance of different device models, manufacturers, and hardware specifications. [9] Different smartphones may exhibit varied behaviors and performance in multi-window mode. We tested a limited number of applications, including video, payment, meeting, and social applications. This cannot represent the performance of all types of applications in multi-window mode. Other types of applications may have different adaptations and functional support. Our testing primarily focused on specific versions of the Android operating system. Different versions of Android may exhibit variations in support and implementation of multi-window mode. Therefore, future research could consider a wider range of Android versions. The continuous updates and evolution of applications may result in the resolution or improvement of problems or limitations identified during testing in subsequent versions. Our testing did not account for variations in application versions, which may impact the continuity of test results.

## 5. Related Work

Automated testing plays a crucial role in Android application development. Many studies have been dedicated to developing automated testing tools for effectively detecting errors and defects in applications. Tools such as UIAutomator, Espresso, and Appium have been widely used for automated testing. To conduct tests on various apps, an automated testing method was designed [1] to replace purely manual testing. Considering the dynamic and unstable behavior of real-world Android applications, a multi-level comparison standard was referenced [2] to optimize the testing method under multi-window mode. Ultimately, tools were selected to assist in completing the automated testing. Through extensive GUI testing, it was observed that traditional model-driven GUI testing focused on code coverage and model coverage [3]. Optimization efforts for GUI testing primarily revolved around improving its precision [4] and performance [5], lacking further validation regarding the effects on real user experience. In terms of test cases, [6, 7] focused on preference testing in Android applications, primarily exploring the influence of user preferences on application functionality. These studies aimed to analyze how user preference settings affect application functionality and user experience. Meanwhile, [8] focused on testing content input generation, aiming to study how test cases effectively cover various aspects of the application. However, none of these studies work on the multi-window environment.In

DOI: 10.54254/2755-2721/73/20240413

contrast, this study focuses on the empirical investigation of functional defects in Android applications in a multi-window environment. By conducting an in-depth analysis of issues in this specific domain, it addresses the shortcomings of past research, making the methods for identifying and resolving functional defects in practical applications more targeted.

## 6. Conclusion

With the continuous evolution of screen design and the increase in mobile phone functionality, the use of multi-window mode is becoming increasingly common. However, this also brings more attention to usability and design defects in multi-window mode. In this context, this paper conducted functional and security testing of three types of applications under three multi-window modes, documenting some of the current issues in multi-window mode and proposing possible solutions. Through in-depth research and improvement, a better understanding and resolution of application challenges in multi-window mode can be achieved, thereby providing a better user experience and higher security.

## References

[1] Linares-Vásquez M, Bernal-Cárdenas C, Moran K, et al. [C]//2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017: 613-622.

[2] Baek Y M, Bae D H. Automated model-based android gui testing using multi-level gui comparison criteria[C]//Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016: 238-249.

[3] Su T, Meng G, Chen Y, et al. Guided, stochastic model-based GUI testing of Android apps[C]//Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 245-256.

[4] Gu T, Sun C, Ma X, et al. Practical GUI testing of Android applications via model abstraction and refinement[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 269-280.

[5] Gu T, Cao C, Liu T, et al. Aimdroid: Activity-insulated multi-level automated testing for android applications[C]//2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017: 103-114.

[6] Pan M, Lu Y, Pei Y, et al. Preference-wise Testing of Android Apps via Test Amplification[J]. ACM Transactions on Software Engineering and Methodology, 2023, 32(1): 1-37.

[7] Lu Y, Pan M, Zhai J, et al. Preference-wise testing for android applications[C]//Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019: 268-278.

[8] Zheng H, Li D, Liang B, et al. Automated test input generation for android: Towards getting there in an industrial case[C]//2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). IEEE, 2017: 253-262.

[9] Kowalczyk E, Cohen M B, Memon A M. Configurations in Android testing: they matter[C]//Proceedings of the 1st International Workshop on Advances in Mobile App Analysis. 2018: 1-6.