CiD4HMOS: A Solution to HarmonyOS Compatibility Issues

Tianzhi Ma[1], Yanjie Zhao[2], Li Li[3], Liang Liu[1,*]
[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
tianzhima@nuaa.edu.cn, liangliu@nuaa.edu.cn
[2]Faculty of Information Technology, Monash University, Melbourne, Australia
Yanjie.Zhao@monash.edu
[3]School of Software, Beihang University, Beijing, China
lilicoding@ieee.org
[*]Corresponding author: Liang Liu (email: liangliu@nuaa.edu.cn)

***Abstract*—HarmonyOS is an operating system boasting a substantial global user base and provides multiple versions of its SDK. Various open-source applications continue to utilize older versions, leading to compatibility issues arising from system constraints. These prevalent issues can substantially affect the user experience. Although numerous solutions have been suggested for addressing compatibility issues in Android, the subject remains largely unexplored within the context of HarmonyOS. To bridge this gap, we investigate the evolution of APIs in HarmonyOS to pinpoint those potentially causing compatibility issues. Based on these insights, we implement CiD4HMOS, a tool designed to detect and categorize compatibility issues in HarmonyOS. We evaluate the feasibility of CiD4HMOS with open-source apps and subsequently apply it to commercially released apps, highlighting its effectiveness in accurately identifying HarmonyOS compatibility issues. The experimental results uncover that CiD4HMOS is effective in detecting compatibility issues in HarmonyOS apps, achieving an accuracy rate of 86.8% in open-source apps. And, developers of commercially released apps have significantly endorsed our reports. Our research emphasizes the necessity of continuous exploration into compatibility issues within HarmonyOS, underlining the significant role tools like CiD4HMOS play in enhancing the overall user experience.**

***Index Terms*—HarmonyOS, Framework Base, Compatibility Issue**

I. Introduction

The Software Development Kit (SDK) serves as the fundamental component of mobile applications, enabling them to interact with the mobile phone's operating system (OS) and hardware [1]. It is comprised of APIs, development documentation, and simulated runtime environments provided by the respective mobile phone operating system developers. As the performance of mobile phone hardware improves, so does the number of features that can be utilized on a mobile phone [2]. This means that a limited number of OS SDK versions are insufficient for developers to fully harness the performance and capabilities of mobile phones. Consequently, mobile phone system providers periodically release new SDK versions to cater to developers seeking to utilize the full potential of a device's features and performance [3]. Each SDK update adjusts or removes certain APIs that have become obsolete while introducing new APIs to support emerging functionalities. Invoking these APIs on devices with incompatible SDK versions could cause compatibility issues [4]. Specifically, mobile phone users may face issues where some apps cannot be installed on their devices; or even if installed, apps may crash when specific functions are accessed, leading to an unsatisfactory user experience.

In the case of Android OS, a mechanism exists for developers to specify the range of SDK versions that apps support to tackle this issue [5]. Nonetheless, given the extensive number of APIs called by each app, it could be a daunting task for developers to determine exactly which SDK version range is appropriate for their app, so runtime crashes of Android apps due to compatibility issues are still common [6]. Therefore, for mobile systems with multiple SDK versions, detecting such incompatibilities to enhance user experience is essential. Fortunately, different solutions have been investigated to automatically identify compatibility issues in Android apps [7]–[17]. For example, Wei et al. [18] manually extracted and modeled a set of API invocations potentially leading to compatibility issues. They further introduced FicFinder, an automated approach for detecting compatibility issues arising from Android fragmentation; however, it still necessitated characterization with human endeavor. Huang et al. [19] proposed CIDER to detect callback API compatibility issues by constructing protocol inconsistency graphs (PI-Graphs) to capture control flow inconsistencies from API evolution. Still, the manual construction of PI-Graphs required human collaboration for each SDK version update, and CIDER's focus was limited to callback APIs. Li et al. [20] introduced a generic approach known as CiD, which identifies API-related compatibility issues by systematically modeling the lifecycle of Android APIs gathered from the official historical evolution of the Android framework.

It is not only Android that is suffering. HarmonyOS [21] is an operating system developed by Huawei, a Chinese multinational technology company. Introduced in August 2019, HarmonyOS was initially designed for Internet of Things (IoT) devices. HarmonyOS 2.0 was initially introduced as a Beta version at the Huawei Developer Conference in September 2020, and later officially launched in June 2021. With the

release of HarmonyOS 2.0, the operating system expanded its scope to include smartphones and other smart devices [22]. Huawei has devoted a lot of effort to the development of HarmonyOS, resulting in the release of 8 major SDK versions, where changes to some APIs are unavoidable during SDK version updates. Mirroring the widespread compatibility issues in Android apps [23], HarmonyOS apps face similar obstacles. Similar to Android, HarmonyOS allows developers to specify the system versions their apps can normally run on to avoid compatibility issues. For each app, developers can set the *Compatible* (equivalent to minSdkVersion in Android) and *Compile* (equivalent to targetSdkVersion in Android) parameters to inform users or app markets about the supported SDK versions. However, previous research into Android compatibility issues has shown that many developers lack precise control over their apps' version range and may not react to SDK version updates [24], [25]. If the *Compile* set by the developer is higher than the actual *Compile* SDK version, it could cause *backward compatibility issues*; if the *Compatible* is lower than the actual minimal SDK version, it can result in *forward compatibility issues*. A poor user experience could ensue if an app cannot run on an operating system with an adapted SDK version. Therefore, an effective approach to detect the appropriate SDK scope is also essential for HarmonyOS app developers. Unfortunately, while the Android community has some experience addressing compatibility issues [20], [26], HarmonyOS, being a brand-new system, everything is unexplored territory.

To fill this gap, in this work, we propose an approach called CiD4HMOS (**C**ompatibility **i**ssues **D**etector **For HarmonyOS**, which for the first time automates the detection of API-related compatibility issues in HarmonyOS apps. CiD4HMOS automatically extracts APIs from each SDK version, and then models and characterizes the corresponding lifecycle for each API. This allows for the detection of potential compatibility issues arising from the APIs invoked in individual apps. The design and implementation of CiD4HMOS draw inspiration from CiD [20], as CiD is a well-known and well-performing tool in the field of Android compatibility issue detection [27]. Experimental results, derived from a dataset consisting of 524 HarmonyOS apps gathered from Gitee [28] and Huawei AppGallery [29], demonstrate CiD4HMOS's effectiveness in identifying compatibility issues within HarmonyOS apps, with a total of 325 backward and 15 forward compatibility issues detected from 58 apps. Going a step further, we perform an in-depth analysis of the detected compatibility issues based on our experimental findings, categorizing them and subsequently offering corresponding repair suggestions. We have communicated these reports with 30 developers responsible for the 58 apps with compatibility issues, receiving 8 positive feedback responses. While 20 developers did not respond or informed us that the project was no longer maintained, and 2 acknowledged the detected issues but chose to disregard them, the feedback indicates that CiD4HMOS can effectively identify compatibility issues in HarmonyOS apps. In summary, our work makes the following key contributions:

TABLE I: The version information of major HarmonyOS versions.

No.	HarmonyOS Version	API Version	Release Date
1	2.0.1.95	4	16/12/2020
2	2.1.1.21	5	02/06/2021
3	2.2.0.3	6	15/09/2021
4	3.0.0.5	7	09/06/2022
5	3.1.1.2	8	09/06/2022

- We provide an overview of HarmonyOS API evolution to measure the scope of situations where compatibility issues might emerge in the HarmonyOS ecosystem. Additionally, we display real-world examples of API-related compatibility issues, thereby validating the necessity for compatibility issue detection in HarmonyOS apps.
- We are the first to propose an approach for detecting compatibility issues in HarmonyOS apps, i.e., CiD4HMOS [1]. Experimental evidence, a total of 325 backward and 15 forward compatibility issues, demonstrates that CiD4HMOS can be effectively utilized for HarmonyOS compatibility issue detection.
- We conduct a thorough analysis and summary of the compatibility issues identified in the collected HarmonyOS apps, proposing relevant solutions and sharing the reports with app developers. We have received positive feedback in response to our findings and suggestions.

II. Background & Motivation

A. HarmonyOS API Versions and Evolution

HarmonyOS is an operating system that was launched by Huawei in August 2019. Initially, the system was only used for IoT, but later in September 2020, Huawei introduced HarmonyOS 2.0 alongside the mobile system at the Huawei Developer Conference [22]. To date, HarmonyOS has introduced 8 release versions of the SDK. As the HarmonyOS 1.0 SDK is no longer available, we concentrate our research on HarmonyOS 2.0 and subsequent SDK versions. There are three types of official release definitions [30], i.e., Canary, Beta, and Release, representing preliminary versions for specific developers, beta versions for all developers (both without API stability guarantees), and official releases ensuring stable APIs for all developers, respectively. Given the minimal differences between the Canary, Beta, and Release versions of each major release and their relatively short usage durations, we opted to use the Release version of each major release as a representative sample. Table I displays the correspondence between each major HarmonyOS version, API level, and their respective release dates [30].

We extracted all public APIs and deprecated APIs from four major versions of the HarmonyOS framework, ranging from API Version 4 to API Version 7 [2]. We then established a public relation for each accessible API, allowing for the connection of the same APIs across different versions through this relation. This process enabled us to obtain the number of APIs per version and assess the modifications of each update.

[1]https://github.com/CiD4HMOS/CiD4HMOS

[2]The reasons for not discussing API Version 8 are stated in Section VI.

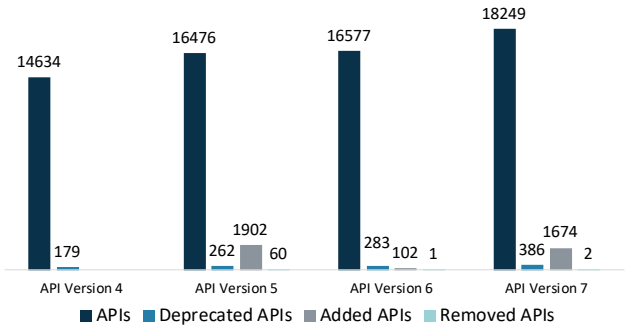

Fig. 1: The statistics of API numbers of HarmonyOS versions.

Figure 1 presents our statistical analysis of HarmonyOS API versions. Initially, we tallied the total number of API methods in each version, noting a growth of approximately 24.7% across four versions. In our analysis, we counted the number of APIs marked as @Deprecated (such as the case shown in Figure 2) in each HarmonyOS SDK version, which accounted for approximately 2% of all API methods at API Version 7. In addition, we use API Lifecycle, a model that allows easy querying of API additions or removals, to calculate the increase or decrease in the number of APIs between consecutive versions. Notably, a larger number of API methods were added to the HarmonyOS framework during the transitions. Since the SDK of the API Version 3 is unavailable now, the *Added APIs* and *Removed APIs* for API Version 4 are not displayed. We then tallied API deletions by version. Currently, this feature is not prominent in large numbers in HarmonyOS, with a total of 60 API methods removed in API Version 5 and just 2 removed in API Version 7. Overall, as evidenced by the statistics, a substantial portion of APIs are only available in a limited number of SDK versions.

Studies [18], [20], [27], [31] have shown that accessing unavailable APIs without proper safeguards can lead to app crashes and negatively affect users' experience. Thus, using these APIs could result in compatibility issues. Some APIs are unavailable since they are erased from the current SDK version. Other APIs could have been recognized as vulnerable and their usage could compromise the security issue, hence, it is not advisable to employ them. As a result, they are marked as "@Deprecated". For example, Figure 2 shows an API, *sendMessage*, excerpted from the HarmonyOS developer documentation [32], which reminds developers to pay attention to the API noted with "@Deprecated" in API Version 4. Deprecation during SDK version updates is one of the most common causes of compatibility issues [23]. If a program invokes this deprecated API and is executed under API Version 5 (where the API has been removed) or higher SDK versions, a *java.lang.NoSuchMethodError* will be thrown, leading to an app crash.

To avoid this problem, HarmonyOS has designed an API-based protection scheme that encompasses its operating system, app market, and apps, where developers can set parameters for their apps to notify the markets and users about the app's version range. Typically, the *config.json* file in the *.hap* folder of the app should include the following two API-related items:

sendMessage
@Deprecated
public void sendMessage(String destinationHost,String serviceCenter,String content)
Deprecated. Replaced by sendMessage(int, java.lang.String, java.lang.String, java.lang.String, ohos.telephony.ISendShortMessageCallback, ohos.telephony.IDeliveryShortMessageCallback).
Sends an SMS message.

Applications must have the ohos.permission.SEND_MESSAGES permission to call this method.

Since:
4

See Also:
splitMessage(java.lang.String)

Fig. 2: Documentation and deprecation message of *sendMessage*.

- *Compatible*: The minimum level of API supported by the app.
- *Compile/Target*: The level of API used by the developers for app development.

For each app, developers can set the *Compatible* - equal to minSdkVersion in Android - and *Compile* - equal to targetSdkVersion in Android - parameters in *build.gradle* to tell the user or the app market which SDK versions are supported by the system.

B. Compatibility Issues

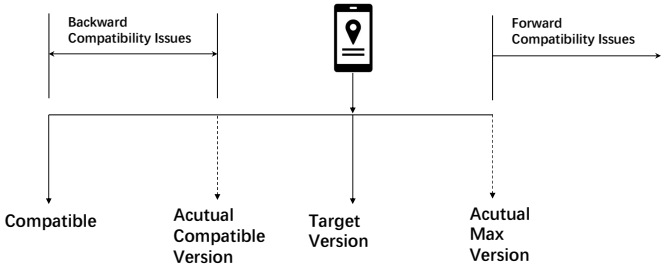

Fig. 3: Examples for Backward and Forward Compatibility Issues. The Actual Compatible Version and Actual Max Version refer to the range of app versions where all APIs can be called and executed without any issues.

Compatibility issues are prevalent in most SDKs. Between different SDK versions, compatibility issues may arise from the inaccessibility or absence of the called API due to API additions, modifications, deprecations, etc. Figure 3 illustrates this phenomenon. An app is expected to function flawlessly when the device operating it has a HarmonyOS SDK version equal to or higher than the version specified by *Compile*. However, if the developer sets *Compatible* to an SDK version lower than the Actual Compatible Version, a *backward compatibility issue* could arise when the app runs on a version between *Compatible* and the Actual Compatible Version. In such cases, triggering an inappropriate API will cause the app to crash due to the inability to find the corresponding API. Likewise, *forward compatibility issues* may emerge if an app runs on a version higher than its Actual Max Version and the relevant API is invoked. Failure of an app to function on a system equipped with a suitable SDK version can significantly impair the user experience. Unfortunately, it is clear from research into Android API compatibility issues that many mobile app

developers do not have precise control over the range of versions their apps use, and will not even react to SDK version updates [24], [25]. Therefore, it is essential for HarmonyOS mobile app developers to utilize a tool capable of detecting the correct SDK range for their apps.

C. Motivation Example

Listing 1 presents a code snippet excerpted from an open-source app, PaletteImageView [33]. The *Compatible* and *Compile* settings for this app are set at 4 and 5, respectively. This suggests that the developer intends for the app to function optimally on an API Version 5 operating system while maintaining minimum compatibility with API Version 4. Unfortunately, within this project, the developer invokes API *setEstimatedSize(int, int)*, as illustrated in Line 3, which was only incorporated into *ohos.agp.components* starting from API Version 5. Thus, if the app runs on a mobile device with an API Version 4 environment and an invocation of this API is triggered, it will result in the app crashing. This is a typical case of a *backward compatibility issue*, where the developer inadvertently calls an API that is not compatible with the specified version and fails to implement versioning protection when the potentially problematic API is invoked. Such instances are not coincidental during our preliminary exploration. Therefore, it is crucial to employ a detection tool specifically designed to address API-induced compatibility issues arising from inappropriate HarmonyOS SDK version settings.

```
1  public boolean onEstimateSize(int
       widthMeasureSpec, int heightMeasureSpec) {
2      ...
3      setEstimatedSize(
4        EstimateSpec.getChildSizeWithMode(
5          mWidth, mWidth, EstimateSpec.NOT_EXCEED),
6        EstimateSpec.getChildSizeWithMode(
7          mHeight, mHeight
8          , EstimateSpec.NOT_EXCEED));
9      ...
10 }
```

Listing 1: An example of Backward Compatibility Issue.

III. Our Approach: CiD4HMOS

A. Overview

In this section, we introduce CiD4HMOS, an innovative approach developed to tackle app compatibility issues within HarmonyOS apps. CiD4HMOS determines compatibility issues by detecting whether the version of an app aligns with the lifecycle of the APIs utilized within the app. Once provided with the existing SDK version and the app under examination, CiD4HMOS automatically detects compatibility issues, subsequently generating a comprehensive report detailing the origin and underlying cause of each identified issue.

B. Framework

As displayed in Figure 4, CiD4HMOS comprises three primary modules: HarmonyOS API Lifecycle Modeling (HALM), HarmonyOS API Usage Extraction (HAUE), and HarmonyOS API Compatibility Analysis (HACA). We now delve into the functions and structures of these individual modules.

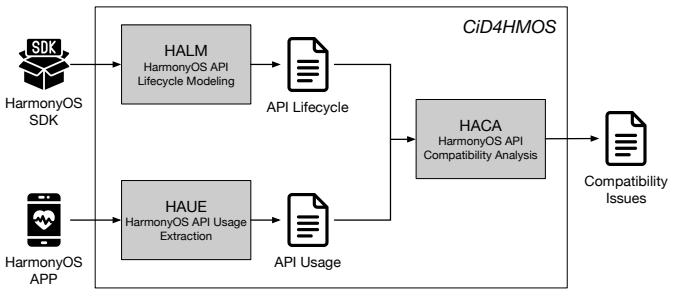

Fig. 4: The working process of CiD4HMOS.

1) HarmonyOS API Lifecycle Modeling (HALM): HALM's primary function is to construct a lifecycle model for HarmonyOS's APIs. This lifecycle can be employed to determine the range of any API within HarmonyOS, from its initial addition to the SDK to its final version in effect. While a comprehensive lifecycle model is highly practical, several challenges arise in constructing a complete and reliable lifecycle.

API Extraction. Our extraction targets comprise all official versions of the HarmonyOS SDK since its public release. This phase involves extracting APIs from the JAR file, focusing on the return values, names, and parameters. These are then saved in a specific format within the respective text files of each version.

API Lifecycle Modeling. After the extraction of APIs, HALM reads successive versions of the APIs to determine whether the current API has made a previous appearance. This process allows us to identify the introduction and deletion versions of each API. Upon completing the analysis across all versions, we can effectively model the lifecycle of the APIs. Interacting with this model facilitates the determination of the lifecycle of any specific API. For instance, the lifecycle of the *getColor(int)* API from the *ohos.app.Context* class was introduced in API Version 5 and persists until API Version 7.

Since HALM's entire extraction process is fully automated, the lifecycle extracted by HALM can be easily updated even if a new version of the API is introduced.

2) HarmonyOS API Usage Extraction(HAUE): In the HAUE module, our objective is to recognize all APIs invoked within the app under detection, extracting those APIs that meet the detection criteria as the output. Notably, HAUE does not require access to the app's source code; the compiled HAP file [3] suffices. To extract these APIs, we divide HAUE into the following three processing steps:

Load additional code. While constructing apps in the format of HAP, developers typically segregate functionalities of different modules, which, after compilation, become different DEX files. We term these codes as *extra codes*. During runtime, the corresponding DEX file is loaded into the system when a section defined in the module is invoked. Since the

[3]The HAP file in HarmonyOS is an installation package format used for deploying Harmony apps onto compatible devices.

extra code also accesses the HarmonyOS APIs, it should be considered when extracting API calls to avoid overlooking potential compatibility issues. We employ Li et al.'s [34] heuristic-based approach to locate dynamically loaded code. In particular, we decompress the provided HAP app, recursively check the files within, and extract all DEX files for potential compatibility issue detection.

Establish a conditional call graph. While it's straightforward to identify all APIs causing compatibility issues by simply checking if detected APIs fall within the app developer's set version range, this method can lead to false positives. Some developers adopt the *API Protection* strategy to avoid errors across varying device versions and to maintain consistent functionality by invoking different APIs under different versions. By using a system call, i.e., *ohos.system.version.SystemVersion: int getApiVersion()*, before calling the APIs to get the current SDK version, and then dynamically selecting appropriate APIs during runtime, the approach effectively avoids compatibility issues and enables specific requirements to be met. Listing 2, excerpted from an open-source app called *TimeTableView* [35], presents a challenge for compatibility issue detection. The API *setForwardTouchListener* at Line 3 was introduced starting from API Version 7. Invoking this API on a device with an API Version of 6 would inevitably lead to a crash. To circumvent this, developers ensure functionality by invoking a different API, i.e., API *setTouchEventListener* at Line 7, on devices with different API versions. This situation poses a challenge for the detection of compatibility issues, as it first necessitates determining whether the API is protected, followed by verifying the accuracy of the protection scope.

```
1  public ElasticScrollView(Context context,
       AttrSet attrSet, String styleName) {
2      if (SystemVersion.getApiVersion() >= 7) {
3          setForwardTouchListener(new
       ForwardTouchListener() {
4              // if block
5          });
6      } else {
7          setTouchEventListener(new
       TouchEventListener() {
8              // else block
9          });
10     }
11     // outside block
12 }
```

Listing 2: An example of API Protection.

To tackle this issue, we perform a backward data flow analysis in the HAUE module to ensure that APIs are invoked within suitable system environments. Recognizing that conditional judgments may not always execute within the current method, we designed our backward data flow analysis to possess cross-program judgment capabilities, thereby preventing inaccurate detections. In order to simplify the analysis process by HAUE, we make use of a specialized call graph known as a conditional call graph (CCG). This graph is designed to facilitate an inter-procedural, path-sensitive, and constructor-aware backward data-flow analysis. A CCG is characterized by a tuple (V, E, C, f), where V denotes a set of methods that form the vertices of the graph, and E signifies a set of directional edges that link two methods. For instance, in the edge $v_1 \rightarrow v_2$, v_1 acts as the caller and v_2 is the callee. C represents a set of conditions tied to the API version. $f : E \rightarrow 2^C$ is a function that assigns a subset of conditions to each edge. For a given edge $e_1 : v_1 \rightarrow v_2$, if $f(e_1) = c_1, c_2$, it implies that there exist at least two distinct call paths from method v_1 to method v_2, each associated with a unique condition (c_1 and c_2).

Resolve API Usage. After completing the first two steps, we introduced all the extra code and determined if the app requires protection when invoking the API. At this point, HAUE can resolve API usage as long as it can ascertain whether the method called by the app is a HarmonyOS API. At present, our focus is centered on the publicly accessible APIs as released by HarmonyOS. Once we detect a HarmonyOS API, we produce an output that provides details about the API, its call pathway, and any other relevant information.

3) HarmonyOS API Compatibility Analysis (HACA): With the HarmonyOS API lifecycle model obtained from HALM and all API calls in the app under detection derived from HAUE, HACA's role is to correlate these two sets of information. Each API discerned by HAUE can be referred to in the lifecycle model. Upon retrieving the respective lifecycle, HACA compares it with the *Compile* and *Compatible* set by the developers. For APIs that developers have protected, the comparison is made while considering conditional protections to filter out potential false positives. APIs that do not comply with the lifecycle are reported as compatibility issues. This approach allows us to generate dependable compatibility issue reports and effectively pinpoint potential compatibility issue APIs based on their paths. More specifically, the produced reports enumerate all the APIs invoked by the app with compatibility issues, detailing the lifecycle of these APIs as well as every method that has called upon these problematic APIs.

C. Implementation

The implementation of CiD4HMOS draws inspiration from CiD [20]. Primarily developed in Java, CiD4HMOS has been tested on a machine running Windows 11. HAP files are similar to APK apps, both of which can be treated as compressed packages. CiD uses DexHunter to obtain all DEX files in APK, but this util cannot be directly applied to HAP apps, necessitating modification of the input entry. We created a bash file (tested only on Ubuntu 20.04 but can be manually decompressed on Windows) to automatically decompress the HAP app, extract the DEX file, and use it as input.

We implemented the backward data flow analyzer on top of Soot [36], a Java framework designed for analyzing, instrumenting, optimizing, and visualizing Java applications. Soot effectively extracts both intra-program and inter-program fragments for CiD. In the HAUE stage, we require a collection file of HarmonyOS APIs to ascertain whether the current API belongs to HarmonyOS. Therefore, we generated a TXT file containing HarmonyOS API data and used the Soot runtime to judge API affiliation by referencing this file. When it comes

to determining running conditions, we need to invoke the *ohos.system.version.SystemVersion: int getApiVersion()* function of HarmonyOS. This step helps us to understand the scope of API protection as defined by the developer.

Furthermore, we have compensated for the limitations in CiD's handling of conditional call graphs, i.e., CiD does not sufficiently consider the protection scenarios for APIs. CiD assumes that anything within the *if* block is safeguarded, while anything outside this block acts as a counterbalance to the *if* condition. While this approach works under certain conditions, it can lead to incorrect detections if a developer uses an *if-else* statement or invokes other APIs outside of the *if* block. Listing 2 serves as an example. If an API, designed to run exclusively under a specific SDK version, such as API Version 6, is invoked outside of the *if-else* blocks - e.g., at Line 11 - it could potentially create problems. The CiD mode might mistakenly assume that this invoked API is protected and within the appropriate scope. However, this can indeed lead to compatibility issues and cause crashes when running on devices operating under other SDK versions, e.g., API Version 7. To address this, we have meticulously dissected the *if-else* protection structure, separating it into *if*, *else*, and *outside* blocks. The *if* block contains the positive aspect of API protection, the *else* block holds the negative aspect, and the outside part is considered unprotected. This strategy allows CiD4HMOS to eliminate a large number of false detections, addressing a key shortcoming of CiD [27].

IV. Study Design

A. Research Questions

The goal of this work is to evaluate whether CiD4HMOS can effectively detect compatibility issues in HarmonyOS apps. To evaluate the accomplishment of our objective, we aim to answer the following three key research questions:

- **RQ1:** Can CiD4HMOS effectively detect API compatibility issues within HarmonyOS apps?
- **RQ2:** What types of compatibility issues exist within the current HarmonyOS apps? Are there generic fixes for different types of software compatibility issues?
- **RQ3:** Can CiD4HMOS perform practically on commercially released HarmonyOS apps? What has been the feedback from developers on our detection reports?

B. Dataset

To address RQ1 and RQ2, we first gathered source code from open-source HarmonyOS apps available on Gitee [28], adhering to specified criteria which limited the main programming language to Java and restricted the version requirements to fall between API Versions 4 and 7. After manually removing projects that failed to compile and run using DevEco Studio, we assembled a collection of 501 open-source projects. For RQ3, we crawled commercially released HarmonyOS apps from Huawei AppGallery [29], with further restrictions that the apps align with the latest API version and have over 10,000 downloads. After filtering based on these constraints, we identified 23 eligible apps that may help uncover compatibility issues that are harder to detect.

V. Study Results

A. RQ1: assessing CiD4HMOS' effectiveness.

To evaluate the effectiveness of CiD4HMOS in detecting API compatibility issues, we executed it on the dataset comprised of 501 open-source projects gathered from Gitee. Table II presents projects with detected forward compatibility issues and those with more than three backward compatibility issues. The third column refers to the commit IDs of the projects obtained from Gitee and employed in our experiments. The fourth and fifth columns indicate the developers' version range settings for the projects. The sixth and seventh columns show the compatibility issues identified by CiD4HMOS. The majority (62.86%) of backward compatibility issues were found in projects with a *Compatible* of 4 and a *Compile* higher than or equal to 5. Out of 501 projects, 35 have backward compatibility issues, accounting for 7% of the total number of projects. In addition, 14 projects were detected with forward compatibility issues, with two projects detected with both backward and forward compatibility issues. As mentioned in Section II, this is attributable to numerous major updates between API Versions 4 and 5. Beyond that, during the scanning process, we identified one potential compatibility issue, which, fortunately, was safeguarded. As shown in Listing 2, the app's developer strategically used *setForwardTouchListener* or *setTouchEventListener* based on the current API Version, thereby preventing incompatibilities on devices with HarmonyOS at these API versions.

TABLE II: Results of open source projects detected by CiD4HMOS.

No.	Project Name	Commit	Developers' Settings		#.Compatibility Issues	
			Compatible	Compile	Backward	Forward
1	colorpicker	bb4c21b	4	5	19	0
2	cv4j	7aaee10	4	5	17	1
3	percentagechartview	ce1b5c5	4	5	14	0
4	custom-flow-layout	c9c6d98	4	7	10	0
5	ohos-imagecropview	96c02d8	4	5	6	0
6	flow-layout-ohos	71131dc	4	5	6	0
7	palette-image-view	2ba37e3	4	5	6	0
8	S.H.Home-Project-HarmonyOS	b66f6d5	4	5	5	0
9	material-icon-lib	402a1a7	4	5	4	0
10	PatternLockView	c01b52c	4	5	4	0
11	ZRefreshView	43562b5	4	5	4	0
12	Ohos-Iconics	e8603d9	4	6	4	0
13	zxing-embedded	d34dc4d	4	5	3	0
14	commonui	62c1c28	4	7	3	0
15	XUI	d77362e	5	6	3	2
16	Ohos-ActionItemBadge	19517e6	4	5	3	0
17	MyLittleCanvas	a4c86a5	5	5	0	1
18	zoomage	787a9de	5	6	0	1
19	ohos-otpview-pinview	794a4a8	5	6	0	1
20	PinView	db2f720	5	6	0	1
21	TimetableView	d5defe1	5	7	0	1
22	BottomBar	e92074a	5	5	0	1
23	FlycoRoundView	17e7876	5	5	0	1
24	MaterialBadgeTextView	8d0357e	5	5	0	1
25	bubble-popup-window	39c3524	5	5	0	1
26	XPopup	4be8eaa	6	7	0	1
27	EmailIntentBuilder	cbf0aca	5	6	0	1
28	SwitchButton	3e02a21	4	5	0	1

Table III showcases the APIs most frequently detected with backward compatibility issues, primarily belonging to the *ohos.agp.com-ponents.Component* class, a core provider of basic UI components. Given that the majority of HarmonyOS apps extensively utilize APIs for UI rendering, and that

TABLE III: High number of occurrences of compatibility issues API.

No.	API	Life	Counts
Backward Compatibility Issues			
1	setEstimateSizeListener(EstimateSizeListener)	[5,6,7]	8
2	setEstimatedSize(int,int)	[5,6,7]	8
3	setVisibility(int)	[7]	6
4	getSize(int)	[5,6,7]	6
5	setDuration(int)	[5,6,7]	5
Forward Compatibility Issues			
1	setLayoutRefreshedListener(LayoutRefreshedListener)	[5,6]	14
2	setOrientation(int)	[4]	1

HarmonyOS itself frequently introduces changes to these UI-related APIs, the most common compatibility issues tend to occur within these UI-related components. Conversely, despite significant changes in API Version 5, the Web Engine module has been found to have relatively few compatibility issues.

Overall, our analysis uncovered 325 compatibility issues, including 15 forward compatibility issues, distributed across 47 out of the 501 open-source projects. Excluding recurring API compatibility issues, we observed 62 backward compatibility issues and 2 forward compatibility issues. We randomly selected ten compatibility issues, each caused by a distinct API, and carried out verifications by (1) manually pinpointing the problematic modules in each project and scrutinizing the relevant code snippets, and (2) actually running the apps on a HarmonyOS device, triggering the suspect APIs to confirm they indeed cause compatibility issues that could lead to app crashes. All ten sampled compatibility issues have been manually verified and validated as correct, with every issue able to be reproduced on the HarmonyOS device.

> *RQ1: CiD4HMOS has proven to be efficient at identifying API compatibility issues in HarmonyOS open-source projects. With the aid of CiD4HMOS, we can quickly and accurately pinpoint the APIs with compatibility issues.*

B. RQ2: examining compatibility issue variations

To investigate HarmonyOS compatibility issues and suggest solutions, we manually analyzed all identified issues in RQ1's dataset and categorized them based on their source code. We compared the APIs' function designs and version update documentation to classify these issues. Subsequently, we manually tested the problematic APIs within selected apps. Based on the issues and crashes we witnessed, we devised repair solutions and applied them to the affected apps. we utilized CiD4HMOS to re-evaluate the modified apps, confirming that our repair solutions effectively resolved the compatibility issues. Specifically, through our experimental results, we can classify API with compatibility issues into six categories, with four of them associated with backward compatibility issues and two with forward compatibility issues. We now provide an in-depth examination of the various compatibility issues identified:

• ***API Added*** constituted the most pervasive API compatibility issue detected within our investigation, representing 77.4% of all backward compatibility issues and 75% of all occurrences. Such compatibility discrepancies originate when developers implement an API without duly accounting for its existence or nonexistence within the context of the project's *Compatible* and *Compile* versions. This oversight often culminates in runtime errors and other complications that can impede the software's performance and overall functionality.

Case Study: *ohos.agp.components.Component:void setEstimatedSize*. The particular compatibility issue related to this API was brought to light during the CiD4HMOS runtime analysis. The problematic API was first introduced to the HarmonyOS SDK in API Version 5 and remained through API Version 7. Apps employing this API designate *Compatible* to 4 and set *Compile* to an elevated version, thereby engendering compatibility issues when users execute the app on HarmonyOS with API Version 4. Taking *com.ramijemli.percentagechartview* as a case in point [37], CiD4HMOS run results indicate that this API is invoked in method *onMeasure* of class *PercentageChartView*. In response to these findings, we executed the app utilizing this API on a HarmonyOS device with API Version 4, which resulted in an app crash. Subsequently, we extracted the error message via the *hdc hilog* command, substantiating that the crash originated from an API compatibility issue.

Fix Recommendations. In light of the extant apps wherein *API added* transpires, we advocate for two remedial approaches. The first entails adjusting the app's *Compatible* from a lower version to one commensurate with the API under consideration. The second involves selecting an alternate API predicated on the SDK version of users' devices. After fixing the API compatibility issues we mentioned in the original projects according to our suggestions, we subjected the apps to a re-evaluation using CiD4HMOS. With the modifications, the original problem no longer exists.

```
1    public boolean onMeasure(int
       widthMeasureSpec, int heightMeasureSpec) {
2        ...
3  +     if(getApiVersion >= 5){
4            setEstimatedSize(widthMeasureSpec,
       heightMeasureSpec);
5  +     } else {
6  +         ...
7  +     }
8        return true;
9    }
```

• ***API Parameter Modification*** represents another instance of API addition. In this case, the API's name remains the same, but the type or number of its parameters may change. If a developer is unaware of this update, or mistakenly believes that the API was released in an earlier version without modifications, they may use an older *Compatible* version, leading to compatibility issues. In our detection of backward compatibility issues, we found that this category accounted for 12.9% of all problematic APIs, with a total of 8 distinct APIs. When considering the number of occurrences, these 8 APIs appeared a total of 17 times, which constitutes 13.7% of all instances.

Case Study: *ohos.agp.window.dialog.ToastDialog: ToastDialog setDuration.* This API serves as a case study of a compatibility issue identified during the runtime analysis by

CiD4HMOS. In API Version 4, the API returned a value type of *BaseDialog*, but in API Version 5, the API was updated to return the *ToastDialog* class, a child class of *BaseDialog*. Importantly, in Java, references to a child class cannot be assigned to a parent class. As a result, when users execute this API on a device operating on API Version 4, the child class *ToastDialog* tries to reference the parent class *BaseDialog*, which could potentially lead to an app crash. An illustrative example of this situation can be seen in the open-source project *com.jaredrummler.ohos.colorpicker* [38]. In this instance, the developer uses a *ToastDialog* type variable to retrieve the return value of the API *showHint* from *utils.colorPanelView*. When deployed on HarmonyOS operating under API Version 4, this execution led to a crash.

Fix Recommendations. Considering the present open-source projects with *API Parameter Modification* compatibility issues, we propose specific recommendations for different scenarios. When an API update modifies the return value type, and the return value is not employed in the original function, an adjustment may not be needed. However, to ensure safety, we advocate for the implementation of device version detection to protect the API. If the API update changes the type or quantity of parameters, we suggest altering the project compatibility or securing the API with *getApiVersion*.

```
public void showHint() {
    ...
-   ToastDialog cheatSheet = new
    ToastDialog(context).setDuration(2000);
+   BaseDialog cheatSheet = new
    ToastDialog(context).setDuration(2000);
  or
+   if(getApiVerson >=5){
        ToastDialog cheatSheet = new
    ToastDialog(context).setDuration(2000);
+    }
    ...
}
```

• ***API Deprecated*** issue occurs when an older API version is marked as deprecated in a newer version, but not immediately removed. These APIs, while still available, are not officially recommended, and developers are urged to shift to newly introduced APIs. We identified a single API causing this type of compatibility issue, comprising 1.6% of all APIs with backward compatibility issues. Regarding frequency, this API surfaced twice, making up 1.6% of all occurrences.

Case Study: *ohos.agp.window.dialog.ToastDialog: setComponent(Component).* In API Version 5, this API was introduced into the *ToastDialog* class, replacing the older *setComponent(DirectionalLay-out)* API. The replaced API was also marked as @deprecated in API Version 5.

Fix Recommendations. Although these compatibility issues don't cause crashes, it's crucial for developers to address them, as deprecated APIs might be removed in future SDK updates. In these situations, using *getApiVersion* to protect the API is generally more practical than altering *Compatible* directly, given that the code continues to operate correctly under previous SDK versions. After implementing protection for the only detected compatibility issue using *getApiVersion*, we could no longer identify similar compatibility issues.

```
    ...
+  if(getApiVersion >= 5) {
        dialog.setComponent(layout);
+  }
    ...
```

• ***API Moved*** refers to the relocation of an API from a parent class to a child class within the SDK. This situation constitutes a unique instance of API addition, as the API is duplicated in the child class while the parent class maintains the original API. Even though these APIs do not directly lead to crashes, they may still cause issues. Much like in the case of *API Deprecated*, the functions of the two APIs within the SDK may not align, leading developers to unintentionally use the wrong API. In our study, we found that five APIs, constituting 8.1% of all APIs, were responsible for backward compatibility issues, and these were encountered 12 times, accounting for 9.7% of all such instances.

Case Study. *ohos.agp.window.dialog.ToastDialog: hide().* In API Version 7, this API was introduced to the *ToastDialog* class, while in API Version 6, the API originates from its parent class, the *BaseDialog* class.

Fix Recommendations. While APIs with this type of compatibility issue have not led to crashes during our testing, we still recommend remediation measures in anticipation of possible changes in future API versions. For these potential compatibility risks, we suggest differentiating the introduction of APIs within the child classes using *getApiVersion*.

```
public void showToast(String toastText) {
    ...
+    if(getApiVersion < 7){
        ((BaseDialog) dialog).hide();
+    }
+    else {
+        dialog.hide();
+    }
    ...
}
```

• ***API Inherited*** occurs when an API is removed from a child class and an identical one is added to its parent class. In our experiments, all 15 detected forward compatibility issues fell under this category. Similar to *API Moved*, this type of forward compatibility issue doesn't directly cause crashes, yet it may pose potential challenges.

```
protected void onCreate() {
    ...
+    if(getApiVersion < 7){
        et_input.setLayoutRefreshedListener(...)
+    }
+    else {
+        ...
+    }
    ...
}
```

Case Study: *ohos.agp.components.Text:setLayoutRefreshedListener(LayoutRefreshedListener).* In API Version 5, this API was added to the *ohos.agp.components.Text* class and subsequently removed in API Version 7, requiring inheritance from the *Component* class for usage. For example, in the *InputConfirmPopupView* class of the app named *com.lxj. xpop-*

updemo [39], the *onCreate* method calls this API. The app has set *Compatible* and *Compile* to 6 and 7, respectively, indicating that in the context of API Version 7, *setLayoutRefreshedListener* is no longer present in *Text*, necessitating a call to the API in its parent class, *Component*.

Fix Recommendations. While this API does not currently lead to direct crashes, the potential differences in the returned object types might yield unpredictable results during usage. We recommend using *getApiVersion* to protect the API and applying forced type conversion to guarantee the method type is used with certainty.

• ***API Removed***, one type of forward compatibility issue, arises when an API from an older version is removed in a newly launched SDK, causing the older version of the app to be incompatible with systems running the newer SDKs. To date, no compatibility issues of this type have emerged in HarmonyOS, so we cannot provide a specific example. However, we can still offer recommendations for addressing such issues.

Fix Recommendations. Compatibility issues, in this case, are likely to result in *No such method error* problems, causing the app to crash. Adjusting the project *Compile* would restrict the SDK versions applicable to the project, which is not the optimal solution. Instead, we suggest utilizing *getApiVersion* to protect older versions of the API and employing custom functions or newer APIs for more recent versions. After testing, these issues can no longer be detected in applications modified in accordance with the above recommendations.

> *RQ2: We present a classification of existing compatibility issues with HarmonyOS apps and outline corresponding modifications based on our analysis and examples.*

C. RQ3: practical usefulness of CiD4HMOS

CiD4HMOS operates as a tool for detecting API compatibility issues in mobile applications, providing developers with crucial information needed to quickly pinpoint problematic modules and gain a comprehensive understanding of the issue's root cause. This includes details such as the API call stack and its actual lifecycle. To further substantiate the practicality of CiD4HMOS, we selected a variety of actively used HarmonyOS apps from the Huawei AppGallery, beyond the dataset used for RQ1, and employed CiD4HMOS on these apps to identify potential API compatibility issues. These commercially released apps, which are more widespread than those sourced from Gitee, are updated more frequently, involve professional developers, and can thus offer valuable insights about CiD4HMOS's effectiveness. As per Huawei AppGallery's submission guidelines [40], apps need to be compressed into a *.app* format using the Build APP function in DevEco Studio before they can be submitted. This *.app* file encapsulates all the project's feature and entry modules, along with a *pack.info* file that houses information about each project module. Moreover, during the build process, the app's *Compatible* and *Compile* settings must be consistent across all modules. Otherwise, the app will not operate correctly. Notably, most open-source HarmonyOS projects, such as those in the RQ1 dataset, feature only one entry module, with a few exceptions containing multiple modules. In light of this, for this research question, we also tested apps comprising several feature modules.

Table IV presents the experimental results from the apps we selected from Huawei AppGallery. The *Backward* and *Forward* columns represent the number of backward and forward compatibility issues detected, respectively. *Backward Protection* indicates the count of correctly protected APIs through API checking. It should be noted that the data for *Backward Protection* is computed independently from *Backward*. The detection reports uncovered common compatibility issues found in open-source apps, as discussed in RQ1, as well as compatibility issues in commercially released apps arising from the APIs in the packages *ohos.agp.components.webengine* and *ohos.distributedhardware*. These packages, which were introduced to HarmonyOS in API Version 5, are commonly used for web communication, explaining their scarcity in open-source software and their prevalence in commercially released apps. Additionally, the successful detection of backward compatibility issue protection in these apps provided empirical data to verify this feature. Developers effectively utilized the *getApiVersion* API to safeguard against these issues, thereby preventing backward compatibility issues in the APIs provided in API Version 7.

In order to validate our detection results and confirm that developers can identify the faulty API location based on the information we provide, we organized the collected reports by type and submitted them to the *Issues* sections of open-source projects on Gitee or to the email addresses of developers of the commercially-released apps. We communicated these reports with 30 developers responsible for the 60 apps with compatibility issues, from which we received 8 positive responses. 20 developers either did not respond or informed us that the project was no longer maintained, and 2 acknowledged the detected issues but chose not to address them. Importantly, the 8 issue reports were directly confirmed by developers who proceeded to rectify their programs based on our insights, with the revisions subsequently committed to their code repositories via Git. Several developers indicated that our work was practically valuable, assisting them in swiftly and effectively identifying perplexing compatibility issues. This feedback underscores the effectiveness of our efforts and our ability to alert development teams to compatibility issues within their apps, thereby enabling necessary rectifications. Moreover, while some developers initially disputed our proposed *API Moved* issue as unnecessary — since it did not directly induce software crashes — our subsequent dialogues successfully persuaded several developers to understand and agree with our perspective, leading them to fix the corresponding parts of their programs.

TABLE IV: Detected results of commercially released apps.

No.	App Name	Downloads	Update dates	APIs	Backward	Backward Protection	Forward
1	bankcomm	190,000	2021-11-02	1967	0	0	1
2	baidu	190,000	2021-10-03	444	1	0	1
3	cctv	2,110,000	2022-01-28	5277	0	0	0
4	citiccard	330,000	2022-03-11	3689	1	0	0
5	cto51	10,000	2021-12-13	52	0	0	0
6	dzh	90,000	2022-03-17	901	1	0	1
7	dzwww	10,000	2022-03-30	503	34	0	0
8	f1schedule	90,000	2022-03-17	124	0	0	0
9	flynormal	540,000	2022-04-07	1103	73	3	0
10	foundao	100,000	2021-08-30	67	0	0	0
11	ifeng	40,000	2021-12-10	138	0	0	0
12	ithome	890,000	2022-04-01	1005	0	1	1
13	moji	680,000	2021-11-01	491	0	0	0
14	pconline	20,000	2022-03-15	769	0	0	0
15	reader	10,000	2022-02-14	1046	80	0	1
16	sina.news	2,660,000	2022-04-01	4728	1	6	0
17	sjweather	60,000	2022-03-04	244	0	0	0
18	tzt	10,000	2022-03-12	799	3	0	1
19	uupt	10,000	2022-04-02	1025	0	0	0
20	wakeup	400,000	2022-03-23	110	0	0	0
21	weibo	440,000	2022-03-30	2136	100	0	0
22	zjs	20,000	2021-11-26	259	42	0	0
23	zyxuexishidai	10,000	2021-09-17	990	50	0	1

RQ3: We demonstrated that CiD4HMOS can provide accurate and pertinent information about compatibility issues to developers, enabling them to locate and address these issues effectively.

VI. Discussion

Software Coding Language. HarmonyOS app development permits a blend of Java, JavaScript, and ArkTS languages. As of now, CiD4HMOS only supports compatibility checks for Java modules within HarmonyOS apps. Fortunately, after randomly decompiling 80 apps from Gitee and Huawei AppGallery, we observed that 63 were primarily written in Java, with a few leveraging JavaScript for interfaces or components. Even though ArkTs was introduced alongside HarmonyOS 3.0 (corresponding to API Version 8) as a coding language for HarmonyOS apps, its stability remains questionable. Moreover, its recent Beta version (API Version 9) presented significant changes from the previous release. Given its current instability and the availability of only two versions, including the newly released Beta API Version 9, our focus is primarily on the Java SDK, i.e., API Version 7 and earlier versions. Hence, expanding our tool's capabilities to cover other coding languages will be a primary objective for future work.

"Expired" Experimental Data. The data collected in RQ1 may be outdated compared to the latest version, potentially affecting the study. At the time of data collection, most apps on Gitee didn't use the latest API Version 8, reducing forward compatibility issues. However, numerous cases between API Versions 4 and 6 helped detect backward compatibility issues for API Version 7. The threat was minimized by creating benchmark apps using forward compatibility issues from API Version 7 to test detectability.

Duplicate Function Naming. Our data analysis suggested that due to the early development stage of the HarmonyOS framework, some experienced developers might evolve existing APIs, leading to user-defined methods with identical names or parameters as in future HarmonyOS versions. This may cause CiD4HMOS to mistakenly classify such functions as compatibility issues when using strings as the search key. However, users can quickly identify and dismiss these false positives, and this issue can be mitigated through version detection.

VII. Related Work

As HarmonyOS is an emerging system with limited API evolution thus far, we look to related work on Android for insights. The issue of Android fragmentation has been a long-standing challenge for users and developers alike, simultaneously emerging as a popular topic of research [23], [26], [31], [41].

API Evolution. Numerous studies have examined the evolution of APIs in the Android framework [42]. Li et al. [23], [43] proposed a research-based prototype tool called CDA, which they applied to various Android framework code versions to characterize enabled APIs. This tool allowed them to investigate deprecated API usage and developer reactions to API deprecation. In addition to deprecated APIs, Li et al. [2] explored the evolution of inaccessible APIs during Android platform updates. Xia et al. [44] introduced an automated tool, RAPID, that combined static analysis and machine learning techniques. Using RAPID, they conducted a large-scale empirical study to determine if Android apps had resolved compatibility issues.

Outside the Android community, numerous studies have focused on API Evolution. Wu et al. [45] examined API changes and usage in 22 framework versions of the Apache and Eclipse ecosystems, identifying and categorizing situations that affect programs. Aué et al. [46] investigated how and why APIs evolve, classifying API changes according to their design intent.

Android Compatibility Issues. In recent years, Android compatibility issues have gained substantial attention, with many studies focusing on API evolution and fragmentation [47]. Scalabrino et al. [24], [48] developed ACRYL, an automated tool for detecting API compatibility issues in Android apps. ACRYL infers rules by analyzing conditional

API usage and assigns a confidence level to each rule, enabling the identification of suspicious APIs. Wei et al. [31] developed the PIVOT tool through an empirical study examining device-related compatibility issues. This method identifies correlations by defining API-Device identifiers, constructs inter-procedural control flow graphs, and identifies API functions related to devices. It then calculates the degree of API-Device correlation and outputs results with higher correlation degrees. Li et al. [20] introduced an automatic detection technique, CiD, to detect potential API compatibility issues in applications. They achieve this by modeling the API lifecycle and comparing API calls to determine whether an API causes compatibility issues. Mahmud et al. [41], [49] proposed a faster and more accurate tool for detecting API compatibility issues caused by API evolution, ACID, which can detect compatibility issues resulting from both Android method invocations and callbacks. However, this tool can only detect compatibility issues in a specific SDK, not those caused by a particular device. Liu et al. [27] used the original dataset of these tools and newly collected real-world apps as input, conducting replication experiments on the detection capability and detection range of these tools. They found that the tools were unable to detect all compatibility issues fully, with only a small overlap in the detection results of each tool.

In non-Android communities, some research on compatibility issues is noteworthy. Brito et al. [50] investigated the use of deprecated APIs in Java systems and designed a tool to detect deprecated APIs used in these systems. Zhou et al. [17] proposed a prototype tool, Deprecation Watcher, with high accuracy and recall for detecting deprecated API usage in Java source code.

VIII. Conclusion

In this paper, we present a concise overview of HarmonyOS API evolution to gauge potential compatibility issues in the ecosystem. With practical examples of such issues, we underscore the need for compatibility detection in HarmonyOS apps. We then introduce CiD4HMOS, the first approach proposed to automate the detection of compatibility issues within HarmonyOS apps. Despite the limited number of SDK versions, we found that compatibility issues can arise while developing apps using current HarmonyOS APIs. Based on these insights, we designed CiD4HMOS and applied it to apps from both the open-source community and the HarmonyOS app market. This enabled us to identify, categorize, and propose solutions for the APIs most commonly causing compatibility issues. The evidence from our experimental results and the positive feedback from app developers affirm CiD4HMOS's effectiveness in detecting HarmonyOS compatibility issues. The replication package of CiD4HMOS and the experimental data are publicly available at: https://github.com/CiD4HMOS/CiD4HMOS.

References

[1] "Software development kit," https://en.wikipedia.org/wiki/Software_development_kit, 2023.

[2] L. Li, T. F. Bissyandé, Y. Le Traon, and J. Klein, "Accessing inaccessible android apis: An empirical study," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 411–422.

[3] I. Krajci and D. Cummings, *History and Evolution of the Android OS*. Berkeley, CA: Apress, 2013, pp. 1–8. [Online]. Available: https://doi.org/10.1007/978-1-4302-6131-5_1

[4] T. McDonnell, B. Ray, and M. Kim, "An empirical study of api stability and adoption in the android ecosystem," in *2013 IEEE International Conference on Software Maintenance*, 2013, pp. 70–79.

[5] "¡uses-sdk¿," https://developer.android.com/guide/topics/manifest/uses-sdk-element, 2023.

[6] H. Wang, Z. Liu, Y. Guo, X. Chen, M. Zhang, G. Xu, and J. Hong, "An explorative study of the mobile app ecosystem from app developers' perspective," in *Proceedings of the 26th international conference on World Wide Web*, 2017, pp. 163–172.

[7] G. Brito, A. Hora, M. T. Valente, and R. Robbes, "Do developers deprecate apis with replacement messages? a large-scale analysis on java systems," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, 2016, pp. 360–369.

[8] M. Fazzini, Q. Xin, and A. Orso, "Automated api-usage update for android apps," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 204–215. [Online]. Available: https://doi.org/10.1145/3293882.3330571

[9] S. A. Haryono, F. Thung, H. J. Kang, L. Serrano, G. Muller, J. Lawall, D. Lo, and L. Jiang, "Automatic android deprecated-api usage update by learning from single updated example," in *Proceedings of the 28th International Conference on Program Comprehension*, ser. ICPC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 401–405. [Online]. Available: https://doi.org/10.1145/3387904.3389285

[10] S. A. Haryono, F. Thung, D. Lo, L. Jiang, J. Lawall, H. J. Kang, L. Serrano, and G. Muller, "Androevolve: automated android api update with data flow analysis and variable denormalization," *Empirical Software Engineering*, vol. 27, no. 3, p. 73, Mar 2022. [Online]. Available: https://doi.org/10.1007/s10664-021-10096-0

[11] A. Hora, R. Robbes, N. Anquetil, A. Etien, S. Ducasse, and M. Tulio Valente, "How do developers react to api evolution? the pharo ecosystem case," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 251–260.

[12] P. Kong, L. Li, J. Gao, K. Liu, T. Bissyande, and J. Klein, "Automated testing of android apps: a systematic literature review," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 45–66, Mar. 2019.

[13] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of android apps: A systematic literature review," *Information and Software Technology*, vol. 88, pp. 67–95, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584917302987

[14] R. Robbes, M. Lungu, and D. Röthlisberger, "How do developers react to api deprecation? the case of a smalltalk ecosystem," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: Association for Computing Machinery, 2012. [Online]. Available: https://doi.org/10.1145/2393596.2393662

[15] A. A. Sawant, R. Robbes, and A. Bacchelli, "On the reaction to deprecation of clients of 4 + 1 popular java apis and the jdk," *Empirical Software Engineering*, vol. 23, no. 4, pp. 2158–2197, Aug 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9554-9

[16] G. Yang, J. Jones, A. Moninger, and M. Che, "How do android operating system updates impact apps?" in *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, 2018, pp. 156–160.

[17] J. Zhou and R. J. Walker, "Api deprecation: A retrospective analysis and detection method for code examples on the web," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 266–277. [Online]. Available: https://doi.org/10.1145/2950290.2950298

[18] L. Wei, Y. Liu, and S.-C. Cheung, "Taming android fragmentation: Characterizing and detecting compatibility issues for android apps," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016. New York, NY,

USA: Association for Computing Machinery, 2016, p. 226–237. [Online]. Available: https://doi.org/10.1145/2970276.2970312

[19] H. Huang, L. Wei, Y. Liu, and S.-C. Cheung, "Understanding and detecting callback compatibility issues for android applications," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 532–542. [Online]. Available: https://doi.org/10.1145/3238147.3238181

[20] L. Li, T. F. Bissyandé, H. Wang, and J. Klein, "Cid: Automating the detection of api-related compatibility issues in android apps," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 153–163. [Online]. Available: https://doi.org/10.1145/3213846.3213857

[21] "Harmonyos," https://www.harmonyos.com/en/, 2023.

[22] "Hmos applied on new products," https://www.huawei.com/en/news/2021/6/huawei-launches-products-powered-by-harmonyos-2, 2023.

[23] L. Li, J. Gao, T. F. Bissyandé, L. Ma, X. Xia, and J. Klein, "Characterising deprecated android apis," in *Proceedings of the 15th International Conference on Mining Software Repositories*, 2018, pp. 254–264.

[24] S. Scalabrino, G. Bavota, M. Linares-Vásquez, M. Lanza, and R. Oliveto, "Data-driven solutions to detect api compatibility issues in android: An empirical study," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 288–298.

[25] A. Hora, R. Robbes, M. T. Valente, N. Anquetil, A. Etien, and S. Ducasse, "How do developers react to api evolution? a large-scale empirical study," vol. 26, no. 1, p. 161–191, mar 2018. [Online]. Available: https://doi.org/10.1007/s11219-016-9344-4

[26] Y. Zhao, L. Li, K. Liu, and J. Grundy, "Towards automatically repairing compatibility issues in published android apps," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2142–2153.

[27] P. Liu, Y. Zhao, H. Cai, M. Fazzini, J. Grundy, and L. Li, "Automatically detecting api-induced compatibility issues in android apps: a comparative analysis (replicability study)," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 617–628.

[28] "Gitee," https://www.gitee.com, 2023.

[29] "Appgallery," https://appgallery.huawei.com/Featured, 2023.

[30] "Api released time," https://developer.harmonyos.com/en/docs/documentation/doc-releases/harmonyos_release_definitions-0000001092857972, 2023.

[31] L. Wei, Y. Liu, and S.-C. Cheung, "Pivot: learning api-device correlations to facilitate android compatibility issue detection," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 878–888.

[32] "Harmony java api document," https://developer.harmonyos.com/cn/docs/documentation/doc-references/toastdialog-0000001054440045, 2023.

[33] "Paletteimageview," https://gitee.com/archermind-ti/palette-image-view/blob/master/paletteimageview/src/main/java/com/dingmouren/paletteimageview/PaletteImageView.java, 2023.

[34] L. Li, T. F. Bissyandé, D. Octeau, and J. Klein, "Droidra: Taming reflection to support whole-program analysis of android apps," ser. ISSTA 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 318–329. [Online]. Available: https://doi.org/10.1145/2931037.2931044

[35] "Timetableview," https://gitee.com/HarmonyOS-tpc/TimetableView/blob/master/entry/src/main/java/com/zhuangfei/hos_timetableview/view/ElasticScrollView.java, 2023.

[36] R. Vallée-Rai, P. Co, E. Gagnon, L. Hendren, P. Lam, and V. Sundaresan, "Soot: A java bytecode optimization framework," in *CASCON First Decade High Impact Papers*, ser. CASCON '10. USA: IBM Corp., 2010, p. 214–224. [Online]. Available: https://doi.org/10.1145/1925805.1925818

[37] "Percentagechartview," https://gitee.com/archermind-ti/percentagechartview/blob/master/percentagechartview/src/main/java/com/ramijemli/percentagechartview/PercentageChartView.java, 2023.

[38] "Colorpicker," https://gitee.com/archermind-ti/colorpicker/blob/master/colorpicker/src/main/java/com/jaredrummler/ohos/colorpicker/ColorPanelView.java, 2023.

[39] "Xpopup," https://gitee.com/HarmonyOS-tpc/XPopup/blob/master/library/src/main/java/com/lxj/xpopup/impl/InputConfirmPopupView.java, 2023.

[40] "Huawei appgallery submission page," https://developer.huawei.com/consumer/en/doc/distribution/app/agc-help-harmonyos-releaseapp-0000001126380068, 2023.

[41] T. Mahmud, M. Che, and G. Yang, "Acid: an api compatibility issue detector for android apps," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, 2022, pp. 1–5.

[42] L. Li, T. F. Bissyandé, and J. Klein, "Moonlightbox: Mining android api histories for uncovering release-time inconsistencies," in *The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE 2018)*, 2018.

[43] L. Li, J. Gao, T. F. Bissyandé, L. Ma, X. Xia, and J. Klein, "Cda: Characterising deprecated android apis," *Empirical Software Engineering*, vol. 25, pp. 2058–2098, 2020.

[44] H. Xia, Y. Zhang, Y. Zhou, X. Chen, Y. Wang, X. Zhang, S. Cui, G. Hong, X. Zhang, M. Yang *et al.*, "How android developers handle evolution-induced api compatibility issues: a large-scale study," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 886–898.

[45] W. Wu, F. Khomh, B. Adams, Y.-G. Guéhéneuc, and G. Antoniol, "An exploratory study of api changes and usages based on apache and eclipse ecosystems," *Empirical Software Engineering*, vol. 21, pp. 2366–2412, 2016.

[46] J. Aué, M. Aniche, M. Lobbezoo, and A. van Deursen, "An exploratory study on faults in web api integration in a large-scale payment company," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 13–22.

[47] H. Cai, Z. Zhang, L. Li, and X. Fu, "A large-scale study of application incompatibilities in android," in *The 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)*, 2019.

[48] S. Scalabrino, G. Bavota, M. Linares-Vásquez, V. Piantadosi, M. Lanza, and R. Oliveto, "Api compatibility issues in android: Causes and effectiveness of data-driven detection techniques," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5006–5046, 2020.

[49] T. Mahmud, M. Che, and G. Yang, "Android compatibility issue detection using api differences," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 480–490.

[50] G. Brito, A. Hora, M. T. Valente, and R. Robbes, "Do developers deprecate apis with replacement messages? a large-scale analysis on java systems," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1. IEEE, 2016, pp. 360–369.