Studying Ad Library Integration Strategies of Top Free-to-Download Apps

Md Ahasanuzzaman, Member, IEEE, Safwat Hassan, Member, IEEE, and Ahmed E. Hassan, Fellow, IEEE

Abstract—In-app advertisements have become a major revenue source for app developers in the mobile app ecosystem. Ad libraries play an integral part in this ecosystem as app developers integrate these libraries into their apps to display ads. In this paper, we study ad library integration strategies by analyzing 35,459 updates of 1,837 top free-to-download apps of the Google Play Store. We observe that ad libraries (e.g., Google AdMob) are not always used for serving ads – 22.5% of the apps that integrate Google AdMob do not display ads. They instead depend on Google AdMob for analytical purposes. Among the apps that display ads, we observe that 57.9% of them integrate multiple ad libraries. We observe that such integration of multiple ad libraries occurs commonly in apps with a large number of downloads and ones in app categories with a high proportion of ad-displaying apps. We manually analyze a sample of apps and derive a set of rules to automatically identify four common strategies for integrating multiple ad libraries. Our analysis of the apps across the identified strategies shows that app developers prefer to manage their own integrations instead of using off-the-shelf features of ad libraries for integrating multiple ad libraries. Our findings are valuable for ad library developers who wish to learn first hand about the challenges of integrating ad libraries.

Index Terms—Ad libraries, Integration strategies, Mining Android mobile apps, Google Play Store

1 Introduction

The mobile app market is continuously evolving at a tremendous rate with billions of mobile app downloads every year [50]. The majority of the apps in app stores are free-to-download [2]. To earn revenue from these free-to-download apps, app developers primarily use an *in-app advertising* model. In this model, app developers display advertisements (*ads*) to app users and earn revenue based on the number of displayed ads and the interactions of users with these ads. The in-app advertising model is a growing market with a forecasted revenue of over $200 billion by 2021 [26]. Figure 1 presents an overview of the in-app advertising model. The in-app advertising model consists of four main components: (1) *advertising companies* that pay for the display of ads for promoting their products, (2) *ad-displaying apps* that display ads and earn revenue from the displayed ads, (3) *mobile ad networks* which act as a bridge between advertising companies and ad-displaying apps, and (4) *users* who use the ad-displaying apps and interact with the displayed ads.

To display ads, app developers need to register with an ad network (e.g., Facebook Audience Network) and integrate into their app an *ad library* that is offered by the ad network. The objectives of an *ad library* is to manage the communication with an ad network, to display ads on a user's device, and to track user interactions with the displayed ads.

Since in-app advertising is a growing market, many ad networks are emerging in this market with their own ad libraries. In this competitive market, app developers select an ad network that maximizes their revenue (e.g., offering a high fill rate[1]) [8]. To earn more app revenue, app developers integrate multiple ad libraries with their apps to increase the fill rate [47].

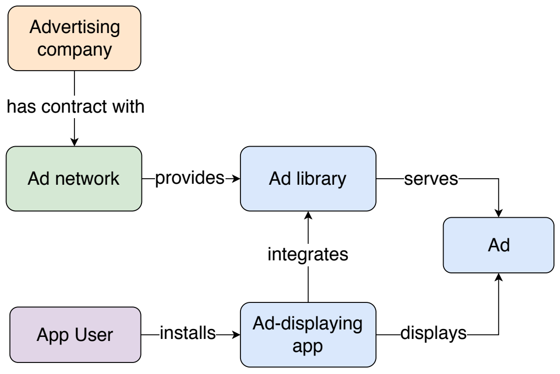

Fig. 1: An overview of the in-app advertising model.

Despite the integral role of ad libraries in the mobile app ecosystem, prior studies have not examined how these libraries are integrated into mobile apps and how app developers handle multiple ad libraries. In this paper, we perform an in-depth study of the common strategies of integrating such ad libraries in the top free-to-download apps in the Google Play Store. Our study can help ad library developers understand the common challenges of integrating multiple ad libraries into mobile apps. Hence, ad library developers can improve the design and the offered features of their ad libraries to ease the ad library integration process.

To study the integration strategies for ad libraries, we

• *Md Ahasanuzzaman, Safwat Hassan, and Ahmed E. Hassan are with the Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Canada.*
E-mail: md.ahasanuzzaman@queensu.ca,
{shassan, ahmed}@cs.queensu.ca

[1]Fill rate is the ratio of the number of displayed ads over the number of requested ads.

analyzed 35,459 updates of 1,837 top free-to-download apps across all the categories of the Google Play Store. Our scope for this research is to study how popular apps integrate ad libraries using standard practices. In particular, we studied such strategies along with the following research questions (RQs):

RQ1: ***What are the characteristics of apps which integrate multiple ad libraries?***
The integration of multiple ad libraries occurs commonly in the apps with a large number of downloads and ones in app categories where a high proportion of apps integrate ad libraries.

RQ2: ***How do app developers integrate multiple ad libraries with their apps?***
We manually examined a statistically representative random sample of ad-displaying apps (62) that integrate multiple ad libraries and derived a set of rules to automatically identify (four) strategies that app developers employ for integrating multiple ad libraries: *(1) external-mediation strategy* (app developers use an external-ad-mediator package that is provided by an ad library and do not write their own code to integrate other ad libraries), *(2) self-mediation strategy* (app developers write their own centralized code (self-mediator) to integrate ad libraries), *(3) scattered strategy* (app developers scatter their code across the different app screens), and *(4) mixed strategy* (app developers use both the external-mediation strategy and the scattered strategy).

We document the definition, example app, the benefits, and drawbacks of each identified strategy for integrating multiple ad libraries. Developers of ad libraries can leverage our strategies to ensure that their ad libraries can support the varying needs of ad-displaying apps.

Paper organization. Section 2 describes our data collection process. Section 3 discusses the characteristics of our dataset. Section 4 presents the results of our study. Section 5 discusses how app developers maintain their integrated ad libraries over time. Section 6 describes the implications of our work. Section 7 describes threats to the validity of our observations. Section 8 discusses related work, and Section 9 concludes the paper.

2 Data Collection

This section describes our process for collecting ad library data. Figure 2 represents an overview of our data collection process. As shown in Figure 2, we first collected the updates of top free-to-download apps in the Google Play Store. Then, we identified ad libraries that are integrated by the apps in these updates. Finally, we identified the updates that display ads. We briefly highlight each step below.

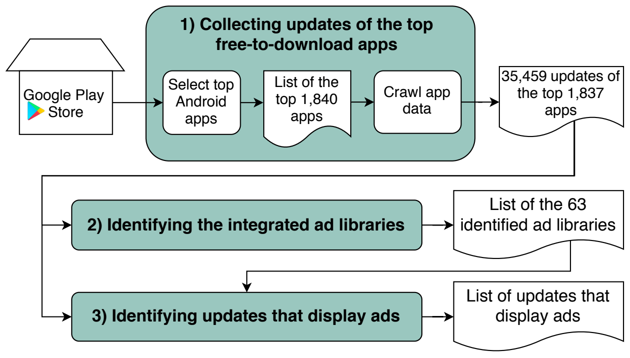
1) Collecting updates of the top free-to-download apps
Google Play Store
Select top Android apps
List of the top 1,840 apps
Crawl app data
35,459 updates of the top 1,837 apps
2) Identifying the integrated ad libraries
List of the 63 identified ad libraries
3) Identifying updates that display ads
List of updates that display ads

Fig. 2: An overview of our data collection process.

2.1 Collecting updates of the top free-to-download apps

Step 1: Select top Android apps. In our study, we focus on the top free-to-download apps as these apps have a large user-base. Hence, these apps are likely to follow the in-app advertising model to earn revenue. Moreover, these apps are likely to carefully maintain their ad integration code in order to ensure that they do not lose any ad revenue. To obtain the list of popular apps, we used the App Annie's report [25] that lists the popular apps across the 28 categories (e.g., Games) in the Google Play Store in 2016. Then, we selected the top 100 apps in each app category so that our study does not have any bias due to variances across the different app categories. During the app selection process, we found that 746 apps were already removed from the Google Play Store at the start of our study period and 214 apps were repeated across the app categories. In total, we selected 1,840 apps and downloaded all their deployed updates for our study.

Step 2: Crawl app data. We ran a custom crawler (based on the Akdeniz [23] Google Play crawler) for 18 months from April 20^{th} 2016 to September 20^{th} 2017 to collect the deployed updates of our studied apps. To study any changes (e.g., code changes) of an app, we need at least two updates of the app. We observed that three apps have only one update during our study period. Therefore, we removed these three apps from our study. Finally, we collected 35,459 updates of the 1,837 top free-to-download apps.

2.2 Identifying the integrated ad libraries

App developers integrate many third-party libraries and identifying an ad library package from these third-party libraries is a non-trivial task. To identify an ad library package, we followed a similar approach to the exhaustive one that is employed by Ruiz et al. [48]. We detail our process below.

First, we converted the APKs of the collected updates to JARs using the dex2jar tool [6]. Then, we used the BCEL tool [24] to extract the fully qualified class names (i.e., the class name and the package name) of all classes in the generated JARs. Since prior studies show that an ad library's packages or class names contain the term "ad" or "Ad" [44], we filtered the fully qualified class names using the regular expression "[aA][dD]". However, this exhaustive regular expression matches many class names that are not related to ad libraries (e.g., `com.fbox.load.ImageLoad`). Hence, to identify ad libraries, we followed Ruiz et al.'s [48] approach by manually verifying online the package name of each of the matched classes. We manually verified 303 packages on the web. In total, we identified 63 ad libraries.

2.3 Identifying updates that display ads

In the previous step, we identified the list of the integrated ad libraries. However, integrating an ad library in an update does not necessarily imply that the update displays ads (e.g., ad libraries can be used for analytical purposes as we discovered in our study). To identify the updates that display ads, first, we identified the app screens (as ads need to be displayed through the app screens). Then, we identified the screens that display ads. The details of our approach are as follows.

Step 1: Identify app screens. To create a single app screen, app developers write the required functionality of the screen in a java class which is known as an *Activity*. Then, app developers define the app screens (i.e, activities) in the *AndroidManifest.xml* file using the "<*activity*>" tag [1]. Hence, to identify the app screens, we parsed the *AndroidManifest.xml* file and listed all the defined activities (using the "<*activity*>" tag) and their corresponding classes.

Step 2: Identify the screens that display ads. First, we identified the integrated libraries in every screen using the BCEL tool [24]. Then, we identified screens that display ads if the screen code invokes the display method in the integrated ad library (e.g., calling the showAd() method). To identify the display ad methods, we read the documentation of the studied ad libraries and summarized the list of methods that are used for displaying ads. In our replication package[2], we added a list of such methods for each studied ad library. Finally, we flagged an update as an ad-displaying update if the update contains at least one screen that displays an ad.

At the end of this step, we identified all updates that display ads.

3 Data Characteristics

In this section, we describe the characteristics of our dataset in terms of (1) ad-displaying functionality, (2) app category, and (3) integrated ad libraries.

Ad libraries are not only used for serving ads but also for analytical purposes. The studied apps can be classified into two main categories: (1) *ad-displaying* apps (i.e., apps that integrate ad libraries to display ads) and (2) *non-ad-displaying* apps (i.e., apps that do not display ads). Table 1 describes our dataset.

As shown in Table 1, non-ad-displaying apps can be of two types: (1) apps that integrate ad libraries but do not display ads and (2) apps that do not integrate ad libraries. We observe that 22.5% of the non-ad-displaying apps belong to type 1 (i.e., integrate ad libraries but do not display ads), and all of these apps integrate the Google AdMob ad library. We also identified 77 apps (4.2% of the studied apps), where we observe that the apps contain ad library packages, but the static analysis tool could not find any method call to ad library packages. Of these 77 apps, we observe that 69 apps use native code. Studying native apps using static analysis tools is difficult and could introduce false positive cases in our analysis. Therefore, in this paper, we focus on studying the apps (1,076 apps) that our static analysis approach identifies a call to show-ad methods of ad libraries from app code.

In our further analysis of the non-ad-displaying apps that integrate ad libraries, we observed that all these apps integrate the Google AdMob ad library for analytical purposes. We observed that analytical libraries (e.g., Google Analytics and AppsFlyer analytics) were dependent on the Google AdMob ad library for uniquely identifying a users device. Table 2 shows the top ten used third-party libraries that depend on the Google AdMob ad library (for the studied 154 apps) to identify a users device. For example, the Google Analytics library depends on the package "com.google.android.gms.ads.identifier" [5] of the Google AdMob ad library which provides the functionality to generate an Android Advertising ID (AAID) to identify a users device instead of using a users personal information (e.g., IMEI number or device MAC address – a practice that is not recommended by Google) [19], [52].

Given our abovementioned observation that ad libraries are not used only for serving ads, further studies of ad libraries need to be careful that the analyzed apps are ad-displaying apps (i.e., the integrated ad libraries are used for serving ads). Otherwise, researchers on mobile ad libraries could falsely identify the ad-displaying apps.

Although the *Google AdMob* and *Facebook Audience Net-*

[2]https://github.com/SAILResearch/suppmaterial-18-ahsan-ads_consumer_apps

TABLE 1: Statistics of the studied apps.

App category	Category definition	# of apps	% of apps
Ad-displaying	Apps that integrate ad libraries and **display ads** (i.e., apps that call show-ad methods).	1,076	58.6%
Non-ad-displaying	Apps that **do not contain** any of the identified ad library packages.	530	28.9%
	Apps that integrate **Google AdMob for analytical purposes** instead of displaying ads.	154	8.4%
Others	Apps that contain ad library packages that **are not used (called)** by any other packages in the app: 69 apps with native code and 8 apps that do not contain native code.	77	4.2%

TABLE 2: Top ten third party libraries that depend on the Google AdMob ad library.

Package name	Library name	# of apps using the package	% of apps using the package
com.google.android.gms.analytics [13]	Google Analytics	151	98.1%
com.appsflyer [3]	AppsFlyer Analytics	23	14.9%
com.flurry.sdk [7]	Flurry Analytics	14	9.1%
com.kochava.android.tracker [9]	Kochava Analytics	13	8.4%
com.localytics.android [11]	Android Location Tracker	10	6.5%
com.life360.android.location [10]	Life 350 Location Tracker	4	2.6%
com.mologiq.analytics [14]	MoLogiq Analytic	4	2.6%
com.quantcast.measurement.service [15]	Quantcast Measure	4	2.6%
com.urbanairship.analytics [18]	Urban Airship Analytics	3	1.9%
com.moat.analytics.mobile.ovi [12]	Moat Analytics	2	1.3%

TABLE 3: Statistics for the top ten integrated ad libraries.

Ad library	# of ad-displaying apps	% of ad-displaying apps
Google AdMob	1,043	96.9%
Facebook Audience Network	478	44.4%
MoPub	287	26.7%
Amazon Mobile Ad	122	11.3%
Flurry	105	9.7%
InMobi	105	9.7%
Millennialmedia	104	9.6%
AdColony	91	8.5%
Applovin	84	7.8%
Unity Ads	65	6.1%

***work* are the most integrated ad libraries throughout the studied ad-displaying apps, some ad libraries are popular within certain app categories.** Table 3 presents the top ten integrated ad libraries of the studied ad-displaying apps. The Google AdMob is the most widely integrated ad library (96.4% of the ad-displaying apps integrate the Google AdMob ad library).

To understand the popularity of an ad library in every app category, we measured the percentage of apps that integrate every ad library in each app category. Table 4 shows the top five integrated ad libraries in each app category. We observe that the Google AdMob and Facebook Audience Network are the most integrated ad libraries in each app category. However, other ad libraries are popular within certain app categories. For example, we observe that the Unity Ads ad library is the second most integrated ad library in the Game category (52% of the ad-displaying apps in the Game category integrate the Unity Ads ad library). One possible reason for the popularity of the Unity Ads ad library in the Game category is that the library provides easy integration to the apps that are built on the Unity framework (a popular framework for building games). In addition, the Unity Ads ad library offers features for displaying rewarded video ads (e.g., users earn an extra life or coins if they watch a video ad), which have become popular among video gaming apps as these ads improve user engagement with the app [16], [20].

TABLE 4: Top five ranked ad libraries in each app category.

App category	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5
Music and audio	GA(95%)	FAN(34%)	MP(23%)	MM(13%)	IM(11%)
Weather	GA(100%)	**MP(53%)**	FAN(45%)	AMA(38%)	MM(30%)
Personalization	GA(96%)	FAN(85%)	MP(37%)	FL(9%)	UA(8%)
Entertainment	GA(85%)	FAN(43%)	MP(27%)	AV(23%)	UA(21%)
Photography	GA(94%)	FAN(59%)	MP(23%)	MV(19%)	AV(10%)
Game	GA(90%)	**UA(52%)**	AC(50%)	VL(40%)	AV(41%)
News and magazines	GA(95%)	FAN(31%)	MP(28%)	FH(15%)	IA(11%)
Tools	GA(98%)	FAN(69%)	MP(45%)	FL(18%)	DAP(18%)
Video players	GA(95%)	FAN(28%)	MP(15%)	IM(8%)	AV(6%)
Auto and vehicles	GA(100%)	FAN(14%)	MP(14%)	–	–
Sports	GA(90%)	FAN(20%)	FH(15%)	MP(13%)	MM(11%)
Social	GA(92%)	FAN(65%)	MP(37%)	FL(27%)	IM(22%)
Comics	GA(88%)	FAN(30%)	AMA(22%)	AC(19%)	IM(13%)
Books and reference	GA(88%)	FAN(24%)	MP(13%)	AMA(13%)	AB(11%)
Health and fitness	GA(100%)	FAN(35%)	MP(20%)	AMA(17%)	MV(10%)
Productivity	GA(95%)	FAN(64%)	MP(26%)	FL(16%)	DAP(9%)
Lifestyle	GA(100%)	FAN(45%)	MP(31%)	AMA(18%)	FL(13%)
Communication	GA(89%)	FAN(54%)	MP(35%)	FL(21%)	IM(18%)
Medical	GA(100%)	**MP(30%)**	FAN(23%)	AM(15%)	AC(11%)
Shopping	GA(90%)	FAN(18%)	TJ(4%)	MP(4%)	VL(4%)
Finance	GA(100%)	FAN(13%)	MP(6%)	FY(6%)	MM(6%)
Maps and navigation	GA(92%)	FAN(7%)	MP(7%)	AS(3%)	–
Travel and local	GA(78%)	**MP(21%)**	MM(14%)	FL(7%)	AOL(7%)
Education	GA(96%)	FAN(33%)	MP(22%)	FL(7%)	–
Libraries and demo	GA(65%)	**MP(11%)**	IM(11%)	FL(11%)	MP(11%)
Business	GA(91%)	FAN(35%)	MP(13%)	AMA(8%)	DAP(4%)

The abbreviations for ad libraries are as follows: AdColony *(AC)*, AdMarvel *(AM)*, AerServ *(AS)*, Amazon Mobile Ad *(AMA)*, AppBrain *(AB)*, Du Ad Platform *(DAP)*, Facebook Audience Network *(FAN)*, Flurry *(FL)*, FreeWheel *(FH)*, Google AdMob *(GA)*, InMobi *(IM)*, MillennialMedia *(MM)*, MobVista *(MV)*, MoPub *(MP)*, TapJoy *(TJ)*, Unity Ads *(UA)*, and Vungle *(VL)*.
* The bold text highlights ad libraries (in Rank 2) other than Facebook Audience Network *(FAN)* ad library.

We also observe that the MoPub (MP) ad library is the second most popular ad library in four app categories (i.e., the Weather, Medical, Travel and local, and Libraries and demo app categories). One possible reason for the MoPub's popularity in these categories is that the MoPub ad library offers an external-ad-mediator. The external-ad-mediator is an ad library feature that facilitates the integration of multiple other ad libraries. In particular, we observe that 76% of the ad-displaying apps in the Weather category integrate multiple ad libraries with the external-ad-mediator of the MoPub ad library. We also observe that the external-ad-mediator of the MoPub ad library is the most used external-ad-mediator in other app categories (i.e., Medical, Travel and local, and Libraries and demo app categories).

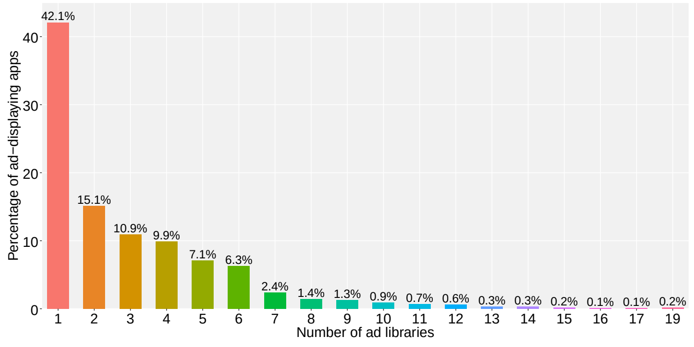

Fig. 3: The percentage of ad-displaying apps that integrate a specific number of ad libraries.

Summary

While ad libraries are commonly integrated for serving ads, they are often integrated for analytical purposes. The mobile ad market is heavily dominated by the Google AdMob and Facebook Audience Network ad libraries, yet other ad libraries still play a leading role in some particular app categories.

4 A Study of the Integration Strategies of Ad Libraries

We now present our study of the integration strategies of ad libraries. For each research question, we discuss the motivation, approach and results.

4.1 RQ1: What are the characteristics of apps which integrate multiple ad libraries?

Motivation: Ad networks decide whether to serve ads for a requesting app based on different factors (e.g., the characteristics of the user-base of that app). Hence, apps most often integrate more than one ad library. A good understanding of the apps that integrate multiple ad libraries would help the developers of ad libraries better understand how apps use their ad libraries and how their libraries co-exist with other competing ad libraries.

Approach: For this study, we calculated the percentage of ad-displaying apps that integrate a specific number of ad libraries. Then, we calculated the *multiple-ads* ratio (as the ratio of apps that integrate *multiple ad libraries* to apps that integrate *a single ad library*) across every download range. A multiple-ads ratio that is higher than one indicates that the number of apps that integrate multiple ad libraries is higher than the number of apps that integrate a single ad library (for a certain download range).

Findings: **57.9% of the ad-displaying apps integrate multiple ad libraries.** Figure 3 shows the percentage of ad-displaying apps along with the number of integrated ad

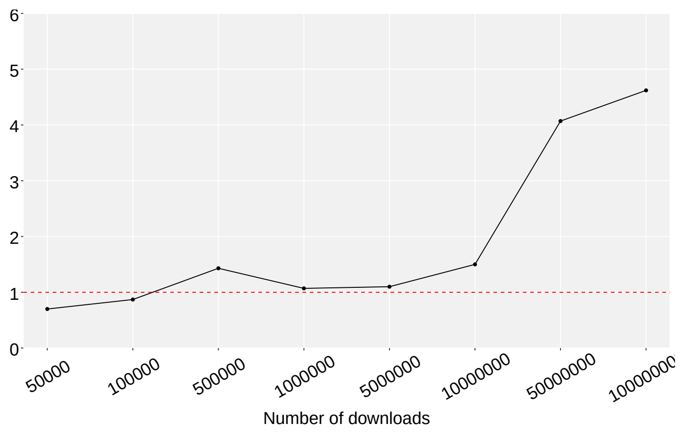

Fig. 4: A line plot showing the ratio of the number of apps that integrate more than one ad library over the number of apps that integrate a single ad library across the number of app downloads. The **red** dotted line in the figure shows the ratio value 1.

libraries. The number of integrated ad libraries could reach up to 19 ad libraries. For instance, the "FreeTone Free Calls & Texting"[3] app integrates 19 ad libraries; these ad libraries represent 32% of the binary size of the app. App developers integrate multiple ad libraries to increase the ad fill rate (i.e., to ensure that their apps can always display an ad) [47].

Apps with a large number of downloads are more likely to integrate multiple ad libraries. Figure 4 presents a line plot of the multiple-ads ratio along with the number of downloads. The ratio value increases as the number of downloads increases. The increase in the ratio value from one to five indicates that apps having a high number of downloads tend to integrate multiple libraries.

The Spearman's rank-order correlation between the number of downloads and the multiple-ads ratio is $\rho = 0.92$ (with a $p - value$ less than 0.05), indicating that the probability of integrating multiple ad libraries increases with the growth in the number of downloads.

Ad-displaying apps are distributed across app categories – with apps in categories having a high proportion of ad-displaying apps integrating multiple ad libraries. Table 5 presents the distribution of ad-displaying apps, the median number of integrated ad libraries, and the maximum number of integrated ad libraries in each app category. Ad-displaying apps are distributed across app categories, with some categories having a higher penetration of ads. For example, more than 90% of the studied apps in the *Music and audio*, the *Weather*, and the *Personalization* app categories integrate ad libraries.

The median number of integrated ad libraries is greater than one for 57.6% of the app categories. The Spearman's rank-order correlation between the percentage of ad-displaying apps and the number of integrated ad libraries (median) for every app category is $\rho = 0.7$ ($p - value$ is less than 0.05), indicating that the number of integrated ad libraries increases with the growth in the proportion of ad-displaying apps within a category.

[3]https://play.google.com/store/apps/details?id=com.textmeinc.freetone

TABLE 5: Distribution of apps that use ad libraries in each app category.

App category	# of studied apps	# of ad-displaying apps	% of ad-displaying apps	Median # of integrated ad libraries	Maximum # of integrated ad libraries
Music and audio	67	64	95%	2	11
Weather	76	71	93%	5	13
Personalization	89	82	92%	3	10
Entertainment	52	42	81%	3	15
Photography	94	77	81%	2	10
Game	59	47	79%	6	17
News and magazines	78	59	76%	2	7
Tools	93	69	74%	3	9
Video players	59	44	71%	1	7
Auto and vehicles	10	7	70%	1	2
Sports	74	52	70%	2	11
Social	81	54	67%	3	19
Comics	56	36	64%	2	7
Books and reference	77	45	58%	1	9
Health and fitness	70	38	54%	2	6
Productivity	80	42	52%	2	10
Lifestyle	42	20	49%	2	7
Communication	76	36	48%	3	12
Medical	55	26	47%	1	6
Shopping	53	22	42%	1	4
Finance	38	15	39%	1	4
Travel and local	69	27	39%	1	8
Maps and navigation	69	26	37%	1	3
Education	70	24	34%	1	6
Libraries and demo	27	9	33%	1	5
Business	81	23	28%	1	5

One of the possible explanations for integrating multiple ad libraries is that as the number of downloads of an app increases or the proportion of ad-displaying apps increases in a category, the competition for ads to display from ad libraries increases, which in turn leads to a lower fill rate. Hence, integrating multiple ad libraries increases the chance of having an ad to display and the potential ad revenue for ad-displaying apps [47].

Summary of RQ 1

The probability of integrating multiple ad libraries increases as the number of downloads of an app increases. Apps in categories with a high proportion of ad-displaying apps are more likely to integrate multiple ad libraries. We hypothesize that the integration of multiple ad libraries is a mechanism to cope with the high demand for ads in an attempt to improve the ad fill rate.

4.2 RQ2: How do app developers integrate multiple ad libraries?

Motivation: Integrating multiple ad libraries is a common practice in ad-displaying apps. A good understanding of how app developers integrate multiple ad libraries can help ad library developers identify the challenges and the possible improvements for their ad libraries.

Approach: To identify the strategies for integrating multiple ad libraries, the first and the second author of this paper manually analyze several ad-displaying apps where app developers integrate multiple ad libraries as follows.

Step 1: Selecting a statistically representative sample of ad-displaying apps. Our data set has 623 ad-displaying apps that integrate multiple ad libraries. Analyzing all these ad-displaying apps manually is both difficult and time consuming. Therefore, for our manual study, we selected a statistically representative random sample of 62 apps (out of the 623 ad-displaying apps) providing us with a confidence level of 90% and a confidence interval of 10%.

Step 2: Generating a static call graph for each selected app. To understand how app developers integrate multiple ad libraries, we need to analyze the ***call-site*** source code (i.e., the packages, classes, and methods that are needed to communicate with an ad library). Hence, we decompiled the generated JARs (in Section 2) into Java source code files using the Class File Reader (CFR) tool [4]. Then, we used the Understand tool [17] to generate and visualize the dependency call graph of each studied ad-displaying app.

Step 3: Identifying the strategies of integrating multiple ad libraries. We start our manual analysis with an open-ended question "How does an app integrate multiple ad libraries?". We observe that apps differ in the way by which they integrate ad libraries with respect to two practices: (1) whether the app code uses a centralized component (i.e., an ad **mediator component**) that handles the access to the multiple ad libraries and (2) whether the centralized component is written by the app developer or by the library designer.

Hence, we manually investigate every selected ad-displaying app based on the following two questions: "Does the app code call a centralized component that handles the access to the multiple ad libraries?" and "Is that centralized component written by the app developer or by the library developer?". Then, we grouped apps with similar integration behavior (in the context of the aforementioned investigated questions) as an **integration strategy**. Finally, we derived a set of rules to automatically identify the integration strategy for any unseen app.

To generate the static call graph and analyze the app code (i.e., classes, methods, and packages) for identifying integration strategies, we took 40 minutes on average for each of the studied apps.

Step 4: Analyzing the characteristics of the identified integration strategies. We ran the derived rules on the studied 623 ad-displaying apps and identified apps that belong to every integration strategy. Then, we studied the characteristics (i.e., the number of call-cite classes) of the apps that belong to every integration strategy. Finally, based on our analysis of the apps, we provide a description, an example, the benefits, and the drawbacks of each identified strategy for integrating multiple ad libraries.

To better understand the integration strategies for ad libraries, we analyze qualitative data sources. In particular, we manually examine two artifacts: (1) 500 Stack Overflow (SO) posts that are related to mobile ads, and (2) 35 ad-related articles from the developer forums and blogs of the top ten ad libraries (e.g., the InMobi blog) as follows:

Step 1: Collecting qualitative data. To identify the Android ad-network-related posts in Stack Overflow we employ SO's search option using the following keywords: "multiple ad network", "fill rate android", "ad mediation", and "admob banner facebook native". The SO's search option returned 255 posts for "multiple ad networks", 118 posts for "file rate android", 113 posts for "mediation ads android", and 14 posts for "admob banner facebook native". In total, we collect 500 ad-network-related posts. The collected SO posts are answered within one (median) day with a median of 667 views – highlighting that our topic and addressed challenges are of great relevance to the development community. To identify ad-network-related articles, we search on Google using a combination of the aforementioned keywords and the names of the different ad libraries. We select only those articles that are related to the forum discussions (e.g., Google Mobile Ads SDK Technical Forum) and blogs of ad libraries. We collected 35 ad-related articles from the developer forums and blogs of the top ten ad libraries.

TABLE 6: Distribution of apps across the different integration strategies.

Ad library integration strategy	# of apps	% of apps
Mixed strategy	317	50.9%
Self-mediation strategy	166	26.6%
External-mediation strategy	63	10.2%
Scattered strategy	77	12.3%

Step 2: Investigating qualitative data. In this step, the first two authors of this paper manually investigate each of the collected SO posts. Our objective of the analysis is to understand the strategies that developers follow to integrate multiple ad libraries into their apps and the associated issues with these strategies. To achieve this objective, we carefully read the title and body of each SO post. In addition, we examine the answers and the comments of every post to understand the integration strategies for ad libraries. Although each of the selected SO posts adds knowledge about integrating multiple ad libraries, we identified 25 SO posts that specifically mention drawbacks or benefits of the integration strategies for ad libraries.

During our manual analysis, if there was a disagreement in the meaning of a post, we (the authors) carefully reread the answers and the comments related to the post and further discussed it until consensus was reached. Since both authors analyzed together all SO posts and agreements (on all analysis of the examined posts) were reached in the end, the authors did not compute the inter-rater agreement. To facilitate the replicability of our work, the studied SO posts are available in our replication package. We follow the same approach to manually study the 35 ad-related articles to understand the process of integrating multiple ad libraries.

It took around 7 minutes to read each SO post and 10 minutes to read each ad-related article (from each author). In total, it took around 12 hours from each author to finish analyzing all the selected SO posts and ad-related articles.

Findings: We identified four strategies for integrating multiple ad libraries. Table 6 shows the distribution of apps across the identified strategies. In the next section, we explain the identified integration strategies.

(1) External-mediation strategy

Description of the external-mediation strategy:
In this strategy, app developers write code to integrate only one ad library that offers a mediator package. This mediator package is responsible for serving ads from other ad networks, which are supported by the ad library. Since

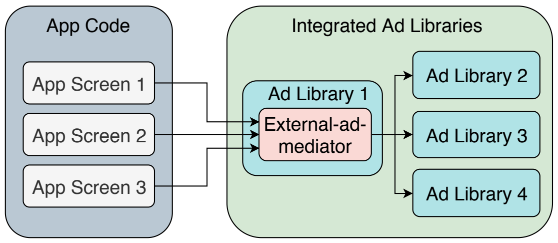

Fig. 5: An overview of the external-mediation strategy.

the mediator is not written by the app developers, we call this an external-mediation strategy.

Figure 5 shows an overview of how app developers integrate multiple ad libraries using an external-mediation strategy. App developers integrate an ad library (Ad Library 1) that has an external-ad-mediator. Every app screen that displays ads communicates only with the external-ad-mediator of the Ad Library 1. The external-ad-mediator communicates with the integrated ad libraries and serves ads from these libraries.

Rules for automatically identifying apps that use the external-mediation strategy:

We determine that an ad-displaying app is using the external-mediation strategy if the following two rules are met:

1) The number of accessed ad libraries by the app code is one, and the number of integrated ad libraries in the app is more than one.
2) The package of the accessed ad library contains an external-ad-mediator package that is accessed by the app code.

An example app that uses the external-mediation strategy:

The *"Ringtones & Wallpapers for Me"*[4] app, a popular app in the Personalization category, displays ads from ten ad libraries. The app code (i.e., the code that is written by the app developer) of this app contains the code for integrating only one ad library (Google AdMob). This app uses the external-ad-mediator of the Google AdMob ad library (com.google.andorid.gms.ads.mediation) which communicates with the other nine ad libraries as well as the Google AdMob ad library for displaying ads.

Benefits of using the external-mediation strategy:

- The ease of integrating multiple ad libraries is one of the main benefits of this strategy as the external-ad-mediator implements the required logic for serving ads from multiple ad libraries [51]. For example, to integrate a new ad library, app developers include the new ad library (along with the external-ad-mediator) in their apps. This process does not require any changes to the ad-call-cite code (compared to the self-mediator strategy that needs changes to the self-mediation support code). The ease of integrating multiple ad libraries maybe one of the reasons for the high ratio of adding or deleting ad libraries for the external-ad-mediator strategy compared to the other integration strategies.

Nevertheless, based on our qualitative analysis of Stack Overflow questions, we noticed that the process of using the external-mediation strategy is not that intuitive as developers may not know that they need to include the external-ad-mediator in their apps (in addition to adding the required ad libraries)[5,6,7].

- The external-ad-mediator selects an ad library for serving ads from the supported ad libraries based on dynamically estimated measures such as the eCPM which captures the ad monetization performance of an ad library at run-time; leading to much more dynamic and accurate estimates of the revenue for a served ad [21].

Drawbacks of using the external-mediation strategy:

- The external-ad-mediator of an ad library may not support all the existing ad libraries that are available in the app market, and integrating unsupported ad libraries could crash mobile apps [28]. Therefore, app developers cannot serve ads from the unsupported ad libraries.

For example, we observe that only 5 out of the identified 63 ad libraries offer an external-ad-mediator. The external-ad-mediators of these five ad libraries (Google AdMob, MoPub, AerServ, Fyber and HeyZap) offer support for serving ads from only 13 ad libraries (20% of the identified ad libraries). Hence, apps that use the external-mediation strategy cannot serve ads from other ad libraries unless they are supported by the external-ad-mediators.

- The entire process of serving an ad is not transparent as app developers have less control over the exact ad library from which an ad is to be served. For example, in one of the discussions in the Google Mobile Ads SDK Technical Forum about how external-ad-mediator works, a developer of the Google Mobile Ads SDK Team states: "Be noted that AdMob will be the one that would decide which mediated ad would display on an ad unit at any given time, depending on various factors (mostly on network priority and eCPM floors)" [31]. This answer indicates that app developers have no control over the selection algorithm. In addition, the external-ad-mediator might provide a preferential serving of ads from its network over other ad networks. This lack of transparency might cause a mistrust issue leading app developers to avoid the use of the external-ad-mediators of some ad libraries [33].

(2) Self-mediation strategy

Description of the self-mediation strategy:

In this strategy, app developers write their centralized package (i.e., self-mediator), which communicates with the integrated ad libraries and manages the process of serving ads.

Figure 6 presents an overview of the self-mediation strategy. All three app screens communicate with the self-mediator, and the self-mediator communicates with the integrated ad libraries to serve ads from them.

Rules for automatically identifying apps that use the self-mediation strategy:

[4]https://play.google.com/store/apps/details?id=com.apalon.ringtones

[5]https://stackoverflow.com/questions/14481380/how-implement-the-mediation-ad-in-android

[6]https://stackoverflow.com/questions/48363760/admob-mediation-with-facebook-audience-network-in-xamarin-forms

[7]https://stackoverflow.com/questions/25008446/android-mediation

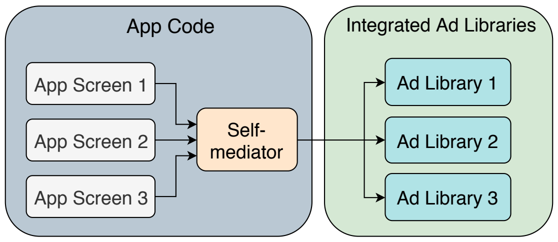

Fig. 6: An overview of the self-mediation strategy.

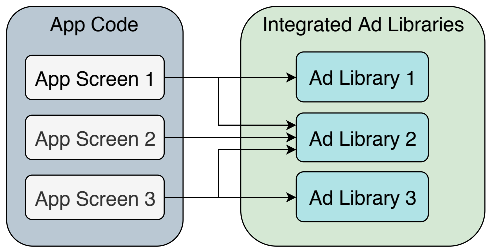

Fig. 7: An overview of the scattered strategy.

We determine that an ad-displaying app is using the self-mediation strategy if the following rules are met:

1) The number of integrated ad libraries is more than one.
2) The number of accessed ad libraries by the app code and the number of integrated ad libraries are equal.
3) The app code contains a centralized package that communicates with the integrated ad libraries.

An example app that uses the self-mediation strategy: The "*Calculator Plus Free*"[8] app, a popular app in the Tool category, integrates seven ad libraries using the self mediation strategy. The developer of the app wrote their self-mediator (com.digitalchemy.foundation.advertising), which communicates with the integrated ad libraries to serve ads.

Benefits of using the self-mediation strategy:

• A self mediation strategy provides a good encapsulation of the code code [34] because app developers write a centralized package that manages the selection and serving of ads from multiple ad libraries.

• App developers are free to integrate any ad library instead of being limited to a handful of supported ad libraries like in the case of the external-mediation strategy. For example, we observe that apps that use the self-mediation strategy integrate 41 ad libraries with 67.6% of these libraries not supporting external ad-mediators.

• App developers have more control over selecting the ad library from which to serve an ad. We observe that app developers mainly use the following three approaches:

1) **A round-robin approach without a preferred list of ad libraries.** In this approach, a random list of the integrated ad libraries is generated. If the first ad library in the list fails to serve an ad, the self-mediator requests an ad from the next ad library. This process continues in a circular order until an ad is served from an ad library.
2) **A round-robin approach with a preferred list of ad libraries.** In this approach, app developers set a preferred list of the integrated ad libraries based on some measures (e.g., the popularity of the ad library in a country or the offered feature of the ad library). The self-mediator selects the best preferred ad library for requesting an ad. If that library cannot serve an ad, then the mediator selects another ad library in a circular fashion from the preferred list until an ad is served from an ad library.
3) **A custom event-based approach.** In this approach, the self-mediator of an app selects an ad library based on custom events (e.g., when a user clicks a particular menu item or when a user earns a reward in the app). This custom event-based approach allows app developers to select the most suitable ad library for increasing user-engagement for that particular event.

Drawbacks of using the self-mediation strategy:

• App developers must write and maintain the code for the self-mediator. The self-mediator represents 8% (median) of the total number of classes of an app (in our studied apps). For example, the "*High-Powered Flashlight*"[9] app, a popular app in the Tool category, serves ads off 10 ad libraries which are integrated using the self-mediation strategy. The self-mediator (the "com.ihandysoft.ad" package) represents 22% of all the classes of this app. We observe that app developers modify the self-mediation support code in 64.5% (median) of their updates – supporting our intuition about the maintenance challenges of the self-mediation strategy.

• The ordering of ad libraries is static in nature. In contrast, the ordering of the external-ad-mediator is much more dynamic as it can order ad libraries based on the dynamically estimated eCPM value which is calculated at run-time based on the buying and selling of ads as conducted through real-time auctions that are facilitated by digital marketplaces (i.e., ad exchanges) [42]. For example, in a Stack Overflow question about the best practice for implementing a self-mediator for displaying ads[10], the accepted answer recommends iterating among the integrated ad libraries (i.e., selecting the first ad library then the next one without considering the current ad network data such as ad fill rate, ad response time, and eCPM). Hence, app developers may not select the best ad library based on the current market conditions [42].

(3) Scattered strategy

Description of the scattered strategy:
In this strategy, app developers neither write their own mediator nor use the external-ad-mediator of an ad library to serve ads from the integrated ad libraries. Rather, developers write code individually for each app screen to integrate each ad library for that particular screen.

Figure 7 shows an overview of the scattered strategy. Each app screen communicates with the integrated ad libraries directly. The integration code for the Ad Library 2 is written by the app developer for every app screen that displays an ad.

[8]https://play.google.com/store/apps/details?id=com.digitalchemy.calculator.freedecimal

[9]https://play.google.com/store/apps/details?id=com.ihandysoft.ledflashlight.mini

[10]https://stackoverflow.com/questions/26685425/best-coding-practice-for-implementing-not-using-mediation-multiple-ad-networks

Rules for automatically identifying apps that use the scattered strategy:
We determine that an ad-displaying app is using the scattered strategy if the following rules are met:
1) The number of integrated ad libraries is more than one.
2) The number of accessed ad library by the app code and the number of integrated ad libraries are equal.
3) The app code does not contain any centralized package that communicates with the integrated ad libraries.

An example app that uses the scattered strategy:
The "*Audiomack – Download New Music*"[11] app, a popular app in Music & audio category, integrates four ad libraries using the scattered strategy. The app displays ads on two screens. The developer of this app wrote code in two app screens for displaying ads from ad libraries individually.

Benefits of using the scattered strategy:

• App developers can quickly integrate several ad libraries as developers do not need to write a centralized package (e.g., the self-mediation support code). Designing a flexible and reusable self-mediator is challenging. For example, in a Stack Overflow question about the design of a self-mediator using the Factory pattern[12]. The answerer notes the complexity of developing a self-mediator: *"In case some of ads controller need additional action (for example update its state or something) you have to add a new method to interface, and this will be unused with other 100500 implementations"*.

• App developers can select ads of different ad formats (e.g., banner or native ad format) from different ad libraries based on custom events in their apps. For example, the "*The Coupons App*"[13] app, a popular app in the Shopping category, integrates Google AdMob and Facebook Audience Network ad libraries in the same app screen. The app selects the Google ad library to serve banner ads and the Facebook Audience Network to serve native ads based on custom events in the app (e.g., the clicking of different buttons in that particular screen).

Drawbacks of using the scattered strategy:

• App developers need more effort to maintain their code because they need to write the same integration code (i.e., copy and paste) for an ad library if the ad library is integrated for displaying ads in different app screens. We observe that the probability of modifying an ad-call-site code (i.e., the app code that invokes the methods for integrating an ad library) is 20% (median) across all ad integration strategies, with that probability increasing considerably to 30% (an increase of 50%) for the scattered strategy. We also observe that the median probability of modifying the ad-call-site code for apps that use the mixed strategy is twice that of the median probability of modifying ad-call-site code for apps that use the external-mediation strategy. Hence, app developers need to modify the ad-call-site code at a much larger rate. We further discuss the maintenance effort of each integration strategy in Section 5.

• The scattered code fetches ads from a single ad library for displaying ads on an app screen. For example, the "*The Coupons App*" app, a popular app in the Shopping category, uses Google AdMob to serve banner ads and uses the Facebook Audience Network to serve native ads in the same app screen. If any of the integrated ad library fails to fetch an ad (e.g., due to the network not filling the ad request), the app screen will fail to display some ads. Alternatively, the app could have used an external-ad-mediator, which provides the needed logic to display ads from different ad libraries if an ad library fails to fill an ad request [51]. Hence, the scattered strategy is not able to deal with low fill rate issues that might arise.

[11]https://play.google.com/store/apps/details?id=com.audiomack
[12]https://stackoverflow.com/questions/35146989/admodule-architecture-using-abstract-factory-pattern
[13]https://play.google.com/store/apps/details?id=thecouponsapp.coupon

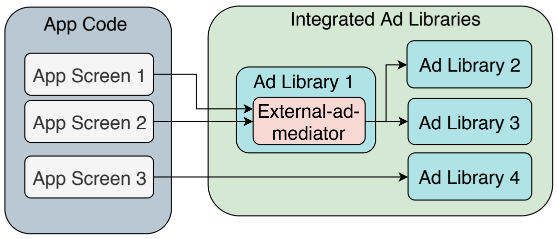

Fig. 8: An overview of the mixed strategy.

(4) Mixed strategy

Description of the mixed strategy:
In this strategy, app developers combine both the external-mediation strategy and the scattered strategy to serve ads.

Figure 8 shows an overview of the mixed strategy. App Screen 1 and App Screen 2 communicate with ad libraries using an external-mediation strategy, whereas App Screen 3 communicates with the Ad Library 4 using a scattered strategy.

Rules for automatically identifying apps that use the mixed strategy:
We determine that an ad-displaying app is using a scattered strategy if the following rules are met:
1) The number of integrated ad libraries is more than one.
2) The number of accessed ad libraries by the app is less than the number of integrated ad libraries.
3) The app contains an external-ad-mediator package that communicates to many of the integrated ad libraries.

An example app that uses the mixed strategy:
The "*Real Guitar Free - Chords, Tabs & Simulator Games*"[14] app, a popular app in the Music category, integrates 14 ad libraries to serve ads. The developers of the app use the external-ad-mediator of the Google AdMob (com.google.android.gms.ads.mediation) ad library to integrate 11 ad libraries. To integrate the remaining three ad libraries, the app developers write code in the activities of some specific screens using the scattered strategy.

Benefits of using the mixed strategy:

• In the mixed strategy, app developers can leverage the external-ad-mediator of an ad library to serve ads from different ad libraries and can also write their integration code for other ad libraries that are not supported by the external-ad-mediator. We observe that 73.9% of ad-displaying apps with the mixed integration strategy call at least one ad

[14]https://play.google.com/store/apps/details?id=com.gismart.guitar

TABLE 7: Mean and five-number summary of each strategy for integrating multiple ad libraries.

Ad library integration strategy	Mean	Min.	1st Qu.	Median	3rd Qu.	Max.
External-mediation strategy	4	2	2	4	5	10
Mixed strategy	5	2	4	5	6	19
Self-mediation strategy	3	2	2	3	4	12
Scattered strategy	2	2	2	2	4	5

library that is not supported by the currently available external-ad-mediators.

- Apps that use the mixed strategy integrate more ad libraries than the apps that use other strategies. Table 7 shows the mean and five-number summary of the integrated ad libraries for each of the four identified strategies. App developers integrate a maximum of 19 ad libraries using the mixed integration strategy. We find two apps from the *"TextMe, Inc"* company that integrate 19 ad libraries. We observe that they use the external-ad-mediator of the Google AdMob ad library which supports 13 ad libraries, the rest of the ad libraries are integrated using a scattered strategy since they are not supported by the external-ad-mediator of the Google AdMob ad library.
- One possible reason for using the mixed strategy is that app developers can display ad formats (e.g., native ads) that are not supported by the external-ad-mediator. For example, a Stack Overflow post notes that a developer used an external-ad-mediator to successfully display banner and interstitial ads from both FAN and Google AdMob ad libraries. Later, the developer wanted to display native ads from the Facebook Audience Network (FAN) using the external-ad-mediator of the Google AdMob. The accepted answer indicates that displaying native ads is currently not supported by the external-ad-mediator (*"Mediation through FAN for Native Express Ads is currently not possible. Only Banner ads and Interstitials have been enabled for mediation for FAN."*)[15]. Therefore, app developers need to use the mixed strategy to display banner, native, and interstitial ads from multiple ad libraries.

Drawbacks of using the mixed strategy: Since the mixed strategy is the combination of the external-mediation strategy and the scattered strategy, some of the drawbacks of these two strategies exist in the mixed strategy. For example, developers need to spend considerable effort on maintaining their ad library code as we observe that the mixed strategy has the highest probability of modifying ad-call-site code (37%) compared to other integration strategies. We further discuss the maintenance effort of each integration strategy in Section 5.

[15]https://stackoverflow.com/questions/37648710/facebook-audience-network-native-ads-via-admob-mediation-adapter

Summary of RQ 2

App developers dominantly use the mixed and the self-mediation strategy to integrate multiple ad libraries. This might be due to the currently available external-ad-mediators not satisfying their needs. To have more control over selecting ad libraries for displaying ads, app developers write their own centralized packages (self-mediator) based on preferred metrics (e.g., location information) or custom app events in the self-mediation strategy.

5 Discussion of the Maintenance Overhead of the Integrated Ad Libraries for Each Integration Strategy

In this section, we discuss how app developers maintain their integrated ad libraries over time across the different ad library integration strategies. In particular, we discuss the modifiability of the ad-call-site code (i.e., the app code that invokes the methods for integrating an ad library) and the flexibility of integrating ad libraries for each integration strategy.

5.1 The modifiability of ad-call-site code

In this section, we discuss the modifiability of the ad-call-site code along two aspects: (1) how frequently (in terms of the proportion of the updates of an app) do app developers modify the ad-call-site code, and (2) what is the proportion of the ad-call-site code that is modified across the integration strategies.

To determine if an ad-call-site code is modified, we follow the same approach of Ruiz et al. [48]. In this approach, for each update of an ad-displaying app that integrates multiple ad libraries, we generate the class signatures for all ad-call-site code (we consider only the statements that invoke ad library methods) of the integrated ad libraries. The ad-call-site code is modified in the app update (U_{i+1}) if the signature of the app update (U_{i+1}) is different than the signature of the app update (U_i).

The probability of modifying the ad-call-site code is 20% (median) across all integration strategies, with that probability increasing considerably to 37% (an increase of 60%) for ad-displaying apps which use the mixed strategy. Figure 9 shows the probability of modifying ad-call-site code for every integration strategy. The probability of modifying the ad-call-site code for ad-displaying apps that use the mixed or the scattered strategies is well above the median. We also observe that the median probability of modifying the ad-call-site code for apps that use the mixed strategy is twice that of the median probability of modifying ad-call-site code for apps that use the external-mediation strategy. This result indicates one of the drawbacks of using the scattered strategy since app developers would need to modify the ad-call-site code in a much larger proportions of the deployed updates of their apps.

To study whether the modifiability of ad-call-site code is significantly different across the identified four ad library integration strategies, we use the "Kruskal Wallis test" [54]

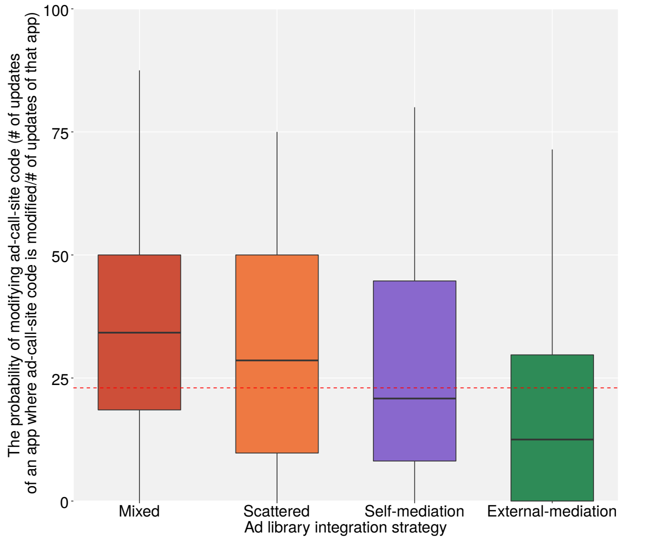

Fig. 9: The probability of modifying the ad-call-site code in a update for each ad library integration strategy. The red dotted line shows the median probability of modifying the ad-call-site code.

TABLE 8: The proportion of modifying ad call-site-code when an ad library is updated and when an ad library is not updated.

Ad library integration strategy	% of the modified ad-call-site code (median) when the integrated ad library is updated	when the integrated ad library is not updated
Mixed strategy	12.5	0.0
Scattered strategy	8.0	0.0
Self-mediation strategy	3.3	0.0
External-mediation strategy	0.0	0.0

for the four categorical variables (i.e., the integration strategies) and one metric (the probability of modifying ad-call-site code). We observe that the generated p-value of the test is less than 0.05 indicating that the the modifiability of ad-call-site code is significantly different across the identified four strategies.

The proportion of the modified ad-call-site code (# of ad-call-site code that is modified / # of total ad-call-site code) is highest in the apps that use the mixed strategy. We observe that app developers mostly modify ad-call-site code when they update their integrated ad libraries. Table 8 shows the proportion of the modified ad call-site-code in two cases: when the integrated ad library is updated and when the integrated ad library is not updated. The proportion (median) of the modified ad-call-site code is zero for each integration strategy (when the ad library is not updated) indicating that app developers usually do not optimize or change their ad library integration code. We also observe that in the case when an ad library is updated, the proportion of the modified ad-call-site code for the apps that use the mixed strategy is the highest whereas the proportion is almost zero for the apps that use the external-mediation strategy. This result is another indication that the mixed strategy may require more effort for maintaining the ad-call-site code over time.

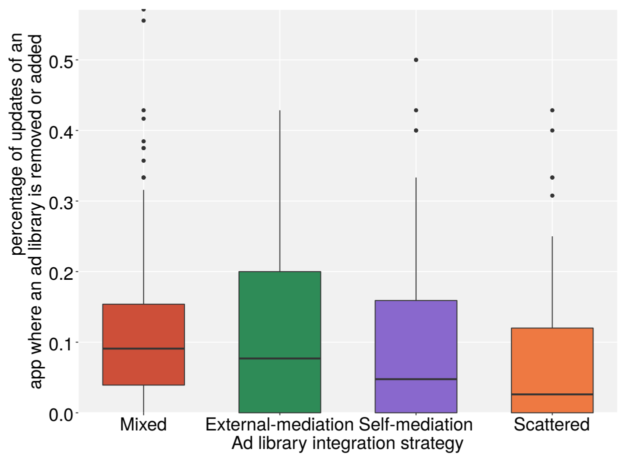

Fig. 10: The ratio of adding/removing ad libraries for each of the integration strategies.

5.2 The ratio of adding or removing ad libraries

To understand which ad integration strategy is more flexible for modifying (adding or removing) ad libraries, we calculate *the ratio of adding/removing ad libraries* for each integration strategy. *The ratio of adding/removing ad libraries* is the ratio of the number of updates of an app in which the app developer adds or removes an ad library to the total number updates of the app.

The mixed strategy provides app developers with the highest flexibility. Figure 10 shows the *the ratio of adding/removing ad libraries* for each identified strategy. The mixed strategy has the highest ratio value. We identify ad libraries that are added or removed in the cases of mixed strategy and observe that all these ad libraries are supported by the currently available external-ad-mediators that are currently in use by these apps. One explanation of this result is that developers do not need to write or update any code to add or remove ad libraries. This hypothesis explains as well the high *ratio of adding or removing ad libraries* for the external-mediation strategy.

6 Implications

In this section, we describe the implications of our study of ad library integration practices for ad library developers.

The developers of the Google AdMob should spin out their functionality for uniquely identifying a user's device out of their ad library. As described in Section 4.1, analytics libraries have a dependency on the Google AdMob ad library. These analytics libraries depend on one of the packages of the Google AdMob ad library named "com.google.android.gms.identifier" for the unique identification of a user's device. These analytics libraries need this functionality to track a user's in-app behavior. However, the main purpose of an ad library is to serve ads. This unusual dependency on the Google AdMob increases the size of many apps that use these analytical libraries. Hence, developers of the Google AdMob ad library should rethink their design and offer a separate library for uniquely identifying a user's device.

Ad library developers should improve their external-ad-mediators by (1) enabling the integration of new ad

libraries at run-time and (2) increasing the supported ad libraries. In Section 5, we observed that app developers use the mixed strategy to achieve the highest flexibility (i.e., continuously adding or removing an add library). To improve the flexibility of the external-ad-mediators, ad library developers need to provide some standardized interfaces to enable the integration of new ad libraries for app developers at run-time instead of only at design time. For example, the Google AdMob ad library has started to offer an SDK-less mediation feature that enables app developers to add or delete any new ad library by re-configuring their Google ads account (without a need for deploying an update that adds/removes the required ad libraries) [40].

Another dimension for improving the external-ad-mediators is to add support for a large number of ad libraries. For example, in Section 4.2, we observed that 73.9% of the ad-displaying apps that use the mixed integration strategy call at least one ad library that is not supported by any external-ad-mediators. Hence, we recommend ad library developers who offer external-ad-mediators to support more ad libraries so that app developers can choose different ad libraries and maximize ad revenue.

Ad library developers should offer more feature control over the selection of ad libraries. In Section 4.2, we observed that the external-ad-mediator selects an ad library from the integrated ad libraries based on dynamically calculated eCPM value. Although this selection process is useful for accurately estimating the revenue for a served ad, it might prevent an app from achieving an improved user-engagement as the process is not customizable. In addition, this process is not transparent to app developers (as eCPM is calculated dynamically) and does not allow app developers to control the selection of ad libraries. We observe that app developers write code (representing a median of 8% of the total number of classes of the app code) for their self-mediator which offers them custom control when selecting ad libraries based on a preferred list of ad libraries (e.g., a list of ad libraries based on the popularity of ad libraries in a country) or custom app events. Therefore, we recommend ad library developers to offer a more configurable interface for their external-ad-mediators so that app developers can have more control in selecting ad libraries when they desire.

7 Threats to Validity

Construct validity: App developers can obfuscate their code using obfuscation tools (e.g., Proguard) before releasing their apps. Although Google Admob requires app developers to keep the names of the ad-show methods non-obfuscated (so apps do not get issues while displaying ads) [39], we cannot assure that all the studied ad libraries mandate the same requirement. Hence, code obfuscation can impact our results as follows. First, if developers obfuscated their package names, our approach will not be able to identify whether an app contains an ad library (as our approach depends on the library package name to identify apps that integrate ad libraries). Moreover, if developers obfuscated the method name of their code, our approach cannot identify whether an app displays ads (as we depend on the usage of the ad-show methods). To assess the impact of code obfuscation on our analysis, we measured the percentage of classes, methods and packages that are obfuscated in our studied apps (i.e., 1,837). We followed a similar approach to identify the code obfuscation that is employed by Li et. al. [44]. We find at least one obfuscated package and method in 18% and 39% of the studied apps, respectively. However, in our further investigation on how much code is obfuscated in an app, we find that only 0.5% (median) of the methods in an app are obfuscated. This result shows that there can be an impact (albeit a small percentage) that this 0.5% of the obfuscated methods include show-ad methods of ad libraries.

To identify the apps that display ads, our approach depends on identifying the calls to the show-ad methods of ad libraries. Hence, if an app is obfuscated, our approach would miss identifying such apps as ad-displaying apps. As shown in Table 1, 530 apps do not contain any of the identified ad library packages. Out of these 530 apps, 225 apps (12% of our overall studied 1,837 apps) contain obfuscated methods in their app code. Since we are not the owner of such 225 apps, we cannot assure whether they integrate ad libraries for displaying ads. Hence, our approach may miss identifying ad-displaying apps for these 225 apps. To overcome the issue of missing the identification of ad-displaying apps, we extensively studied a large number of apps that display ads to understand how app developers integrate ad libraries into their apps on a large scale. It should be noted that our objective of this study is to understand the approaches that app developers use to integrate multiple ad libraries instead of providing an approach on how to reverse engineer obfuscated methods. Future studies could extend our work by including more apps from different app stores.

Identifying the ad libraries among the many integrated third-party libraries is a non-trivial task. First, we identify packages using the regular expression [aA][dD] (following a similar approach as proposed by Ruiz et al. [48]). Then, we manually search online each identified package to determine whether it corresponds to an ad library. However, there is a chance, albeit an extremely rare one, that an ad library exists for which we cannot find any web reference. To measure the comprehensiveness of our identified integration strategies for ad libraries, we randomly examined a statistical representative sample of 65 apps out of the 1,076 studied ad-displaying apps. Such a sample would provide us with a confidence level of 90% and a confidence interval of 10%. To eliminate any bias in our evaluation results, we do not include the 62 apps that we initially used for identifying the integration strategies for ad libraries. For each of the examined apps, we manually examine the app code (using the Understand tool [17]) and validate the integration strategies for ad libraries. We observe that our proposed rules correctly identify the integration strategies for ad libraries in each of the studied 65 apps.

Our approach for detecting ad libraries could suffer from both false positives and false negatives. In the case of false positives, an app can have dead code with the word ad in its class name or a popular third-party library might include ads mischievously. However, we studied top apps where we feel the chances of these concerns to occur are extremely low. The chances of an app to not remove dead code (and associated ad libraries) is quite low since mobile apps are very mindful of the size of their binaries. The chances of an app being malicious are very low again – recall we are looking at the top apps in the market. In the case of false negatives, there might be many ad networks that do not have the term "ad" in its APIs. The chances of an ad library not having the word "ad" is in it is quite low. We also retrieved a list of ad libraries that is curated by AppBrain [27] and find that the concern is not valid for that curated list. The list contains 120 ad libraries. Of these 120 ad libraries, we identified 63 ad libraries that are integrated by the studied apps. For the remaining 57 ad libraries, we read their documentation and GitHub packages and validate that their API contains "ad" keyword.

To identify the display ad methods, we read the documentation of the studied ad libraries and summarized the list of methods that are currently used for displaying ads. However, APIs may evolve, and developers of ad libraries might rename such methods (display ad methods). To determine whether such show-ad methods were changed during the evolution of their libraries, we investigated all the historical versions that are released during our study period of the top ten popular ad libraries as follows. First, we identified the methods that are currently used for displaying ads from the documentation of the selected ad libraries. Then, for each version of the selected ad library, we examined whether the identified methods exist in the prior versions. We observed that all identified methods exist in the prior versions. We did observe that a few ad libraries (e.g., Unity Ads) added new parameters to these methods. However, they did not change the name of these methods. For example, developers of Unity Ads did not change the name of the show method in version 2.0.0 but changed the parameter of the show method (i.e., developers changed *"public void show (Map <String, Object> options)"* to *"public void show (activity, placementId)"* in version 2.0.0).

One possible threat in our analysis on the modifiability of ad-call-site code is code obfuscation of the studied apps as we cannot track the changes in code statements of obfuscated code. Hence, in this analysis, we filtered the obfuscated classes by following the approach of Li et al. [44] and studied the classes of the app code that are non-obfuscated. As illustrated in our discussion about code obfuscation, the percentage of obfuscated code is only 0.5% (median). Consequently, there might be cases (albeit a small percentage) that the obfuscated code has modifications to the ads-call cite statements.

Since native code in Android apps is deployed in the app as executable and linkable format (ELF) files [56], using static analysis tools cannot generate all the dependency links accurately [22]. We observe that only 6.5% of the studied apps (121 apps out of 1,837) use native code in our dataset. We identified 69 of them integrate ad libraries (i.e., 6% of the studied ad-displaying apps). Studying native apps using static analysis tools is difficult and could introduce false-positive cases in our analysis of the integration strategies for ad libraries. Hence, we removed these 69 apps from our analysis of the integration strategies for ad libraries. Since these apps are only 3.8% of the studied apps (69 apps out of 1,837), we believe that they will not drastically impact our overall study of the integration strategies for ad libraries.

External validity: In this study, we only focused on the top free-to-download Android apps from the Google Play Store as these apps have a large user-base. Hence, these apps are likely to follow the in-app advertising model to earn revenue. Future studies should broaden the scope of our study and investigate how our findings apply to ad libraries that are integrated into other types of apps, such as non-free apps, Windows apps, or iOS apps.

According to the documentation of the top ten most used ad libraries in our dataset, the standard way of displaying ads (e.g., full-screen ads) is to call display methods (e.g., showAd()) of the integrated ad libraries [38], [46] from an activity. However, we do not claim that this is the only way to display ads. For example, app developers can display ads using intents or background services[16], which is not a recommended practice for displaying ads [37]. In addition, malicious apps and adware apps can also display ads using background threads, which can be difficult to detect using static analysis mechanisms. However, we analyzed popular apps that are actively maintained and present in the Google Play Store, and they are less likely to be malicious or adware apps. Additionally, our scope for this research is to study how popular apps integrate ad libraries using standard practices.

In Section 6, we point out three implications for ad library developers based on our study so that ad library developers could improve their ad libraries. We cannot deny that there might be some business constraints for improving ad libraries. For example, supporting the integration of all ad libraries in the ad market is not a simple task and might be influenced by competitive business logic instead of technical challenges. However, our findings and suggestions show the current needs and the potential directions for improving the design of third-party ad libraries. Hence, our study could be helpful for ad library developers who wish to improve their ad libraries.

Internal validity: In our analysis for identifying strategies for integrating multiple ad libraries, we manually investigated apps that integrate more than one ad library. In this analysis, we cannot deny the possibility of misinterpreting the identified strategies for integrating multiple ad libraries since we are not the original developers of the studied apps. To mitigate this threat, the first and the second author leveraged the Understand tool to analyze the call graph of the apps, carefully investigated each of the sampled apps, and consolidated their results.

App users are prone to repackaged or piggybacked apps. Hence, app stores (e.g., the Google Play Store) continuously remove piggybacked apps or malware apps [55], [57]. In our study, we selected 1,837 top free-to-download apps in the Google Play Store. These apps have a large user-base, and they are popular in the Google Play Store for multiple years. Therefore, our studied apps are less likely to be malicious apps.

There are different possibilities that can drive app developers to group their code in a single component. For example, poorly designed apps can be one of the reasons for this behaviour (e.g., app developers just group all their code in a single package). However, based on our definition of the self-meditation strategy, an app contains a self-

[16]https://stackoverflow.com/questions/14313641/show-admob-ads-from-service-context-android

mediator component whenever: (1) there is a centralized component that handles all the calls to ad libraries and (2) this component is separate from the main activity code components. Our definition means that the design of the self-meditation strategy encapsulates ad handling mechanism in a single package. However, the identified self-mediation components can be poorly designed if they contain non-ad related features. Assessing the quality of mobile apps architecture is out of the scope of this work. Further work can analyze the architectural quality of mobile apps.

To validate our approach for identifying the "self-mediation" strategy, we manually investigate a randomly selected sample of 20 apps that use the "self-mediation" strategy. We observe that all of these 20 apps have a centralized package that is mainly responsible for managing the integration of multiple ad libraries. For example, the "Agar.IO" app (a popular app in the Game category) has the package "com.miniclip.ads", which manages the integration of 12 ad libraries.

8 Related work

Prior research mainly studies the updates of ad libraries, the cost of ad libraries and the security and privacy issues surrounding ad libraries. Our study is the first to investigate the integration practices of ad libraries. We briefly highlight the related works as follows:

8.1 The updates of ad libraries

Ruiz et al. [48] analyzed 120,981 free-to-download apps in the Google Play Store and conducted an empirical analysis on the rational for updating ad libraries. The authors observed that ad libraries are updated frequently in 48% of the studied apps. They also observed that updating the interaction between ads and app users, integrating new types of ads, bugs related to memory management, and the improvement of the privacy of collected personal information are the main reasons for updating ad libraries.

Salza et al. [49] conducted an empirical study on the evolution history of 291 apps from the F-Droid repository to study how mobile app developers perform updates of third-party libraries including ad libraries. The authors observed that developers usually upgrade towards a newer version.

Derr [30] et al. studied the updatability of third-party libraries (including ad libraries). The authors analyzed the updatability of the integrated libraries of 1,264,118 apps from the Google Play Store. They observed that in 85.6% of the cases the integrated ad libraries can be updated to at least one version without any code changes.

8.2 The cost of ad libraries

Ruiz et al. [47] analyzed 236,245 apps of the Google Play Store to study the effect of ad libraries on the rating. The authors observed that there is no relation between the number of integrated ad libraries and the rating of an app. However, they observed that integrating certain ad libraries could negatively impact the rating of an app.

Gui et al. [41] analyzed 21 apps from the Google Play Store to study five types of costs due to the integration of ad libraries: performance, energy consumption, network, maintenance of ad-related code, and user reviews. The authors observed that the cost of ads in terms of performance, energy and bandwidth are the most concerning. They also observed that complaints related to ads had a negative impact on the rating of an app.

Gao et al. [35] designed a tool named IntelliAd to automatically measure the ads-related consumption (e.g., memory) on apps. In another study, Gao et al. [36] used the IntelliAd tool and analyzed 12 ad schemes that are used in 104 Android apps to measure and compare the performance cost of different ad schemes. They observed that ad schemes are significantly different and recommend app developers to choose appropriate ad providers and ad sizes.

To study the ad network traffic, Vallina-Rodriguez et al. [53] analysed the dataset of a European mobile career with more than 3 million subscribers. The author observed that ads account for 1% of all mobile traffic in the data and, static images and text files are likely to be re-downloaded. To limit the energy and network signalling overhead, the authors built a prototype implementation using the caching mechanism which shows an improvement of 50% in energy consumption for offline ad-sponsored apps.

Mohan et al. [45] studied the communication costs for serving ads by analyzing 15 Windows phone. The authors observed that ad modules consume a significant part of an app's energy and the overhead of ads is bigger in apps with no or small network activity. To reduce the energy of an app that displays ads, the authors proposed an solution of prefetching ads. The authors analyzed the logs of 1,693 Windows phone users over one month and 25 iPhone users over one year to predict app usage from historical data and built time-based models to predict available ad slots in future. The entropy-based evaluation result of their approach shows that the approach is capable to reduce energy consumption of client devices by 50%.

Li et al. [44] analyzed 1.5 million apps that use 1,113 third-party libraries and 240 ad libraries to investigate the use of commonly integrated libraries. Their study showed that the most used library is the Google ad library (AdMob). Li et al. also found that a significant portion of apps that used ad libraries are apps that are flagged by virus scanners. In our study, we focus on top rated apps to avoid dealing with malicious and spam apps.

8.3 The security of ad libraries

Book et al. [29] studied the evolution of the requested permission of ad libraries. The authors analyzed the integrated ad libraries of 114,000 apps for this study. They observed that the use of permission has increased over time and most of the requested permissions of ad libraries are risky in terms of the privacy and security of app users.

Kim et al. [43] analyzed the protective security measures of the four ad libraries (Google AdMob, MoPub, AirPush, and AdMarvel) against malicious advertising. The authors observed that malicious ads can collect sensitive information about a user with the help of permissions such as *WRITE_EXTERNAL_STORAGE* and *READ_EXTERNAL_STORAGE* which give access to external storage.

Dong et al. [32] studied ad fraud (e.g., cheating advertisers with fake ad clicks) in mobile apps and proposed an automated approach for detecting ad fraud. Their approach achieves 92% recall and 93% precision on a manually validated data set of 100 apps. A further study on 12,000 ad-supported apps that use 20 unique ad libraries showed that no ad library was exempt from fraudulent behaviours and AppBrain ad library is the most targeted ad library for ad fraud.

9 Conclusions

In the mobile app economy, ad libraries play an integral role. App developers integrate one or many ad libraries to display ads and gain revenue based on user interactions with the displayed ads. Even though ad libraries play an essential role in the app ecosystem, prior studies have not explored how these libraries are integrated by app developers. In this paper, we analyze 35,459 updates of the 1,837 top free-to-download apps to study the ad library integration practices. Our findings highlight how app developers leverage several ad libraries to display ads. Moreover, we noted several limitations of current ad libraries and offered suggestions for ad library developers to better serve the needs of app developers.

References

[1] App Manifest Overview. https://developer.android.com/guide/topics/manifest/manifest-intro. (Last accessed: March 2020).

[2] AppBrain Intelligence. https://www.appbrain.com/stats/. (Last accessed: March 2020).

[3] AppsFlyer. https://www.flurry.com/. (Last accessed: March 2020).

[4] CFR. http://www.benf.org/other/cfr/. (Last accessed March 2020).

[5] com.google.android.gms.ads.identifier. https://developers.google.com/android/reference/com/google/android/gms/ads/identifier/package-summary. (Last accessed March 2020).

[6] dex2jar. http://sourceforge.net/projects/dex2jar/. (Last accessed March 2020).

[7] Flurry Analytics. https://developer.yahoo.com/flurry/docs/analytics/. (Last accessed: March 2020).

[8] How do you decide which Ad Network is best for your Mobile App (eCPM, retention)? https://www.quora.com/How-do-you-decide-which-Ad-Network-is-best-for-your-Mobile-App-eCPM-retention. (Last accessed: March 2020).

[9] Kochava. https://support.kochava.com/. (Last accessed: March 2020).

[10] Life360. https://www.life360.com/. (Last accessed: March 2020).

[11] Localitics. https://docs.localytics.com/dev/android.html#android. (Last accessed: March 2020).

[12] Moat Analytics. https://moat.com/analytics. (Last accessed: March 2020).

[13] Mobile App Reporting in Google Analytics - Android. https://developers.google.com/analytics/devguides/collection/android/v4/. (Last accessed: March 2020).

[14] NinthDecimal. https://www.ninthdecimal.com/. (Last accessed: March 2020).

[15] Quantcast. https://www.quantcast.com/. (Last accessed: March 2020).

[16] Rewarded video ads will continue to dominate through 2018. https://mobilemarketingmagazine.com/rewarded-video-ads-will-continue-to-dominate-through-2018. (Last accessed March 2020).

[17] Understand Tool. https://scitools.com/. (Last accessed: March 2020).

[18] Urban Airship Analytics. https://www.airship.com/platform/analytics-data/performance-analytics/. (Last accessed: March 2020).

[19] Usage of Android Advertising ID. https://play.google.com/about/monetization-ads/ads/ad-id/. (Last accessed March 2020).

[20] Video ads bring in 31% of app revenue, while rewarded video ads top user experience approval charts. http://www.businessofapps.com/news/video-ads-bring-in-31-of-app-revenue-while-rewarded-video-ads-top-user-experience-approval-charts/. (Last accessed March 2020).

[21] What is eCPM and How Can You Increase It? https://www.ironsrc.com/blog/what-is-ecpm/. (Last accessed: March 2020).

[22] V. M. Afonso, P. L. de Geus, A. Bianchi, Y. Fratantonio, C. Krügel, G. Vigna, A. Doupé, and M. Polino. Going native: Using a large-scale analysis of android apps to create a practical native-code sandboxing policy. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium*, NDSS '16, 2016.

[23] Akdeniz. Google Play Crawler. https://github.com/Akdeniz/google-play-crawler. (Last accessed: March 2020).

[24] Apache Software Foundation. Download Apache Commons BCEL. https://archive.apache.org/dist/commons/bcel/, 2018. (Last accessed March 2020).

[25] AppAnnie. App Annie. https://www.appannie.com/, 2016. (Last accessed March 2020).

[26] AppAnnie. In-App Advertising Spend to Triple, Reach $201 Billion by 2021. https://www.appannie.com/en/insights/market-data/app-advertising-spend-2021/, 2017. (Last accessed: March 2020).

[27] AppBrain. Android Ad Network statistics and market share. https://www.appbrain.com/stats/libraries/ad-networks. (Last accessed: March 2020).

[28] G. Bavota, M. L. Vásquez, C. E. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk. The impact of API change- and fault-proneness on the user ratings of android apps. *IEEE Transactions on Software Engineering*, 41(4):384–407, 2015.

[29] T. Book, A. Pridgen, and D. S. Wallach. Longitudinal Analysis of Android Ad Library Permissions. *Computing Research Repository*, abs/1303.0857, 2013.

[30] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes. Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android. In *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2187–2200, 2017.

[31] G. M. A. S. Developers. Googlemobileads in unity – mediation sample? how do other "mediation adapter" show? https://groups.google.com/forum/#!topic/google-admob-ads-sdk/qqcssGAYEk4. (Last accessed: March 2020).

[32] F. Dong, H. Wang, L. Li, Y. Guo, T. F. Bissyandé, T. Liu, G. Xu, and J. Klein. FraudDroid: Automated Ad Fraud Detection for Android Apps. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE'18, pages 257–268, 2018.

[33] Eric Benjamin Seufert. Should I create a mobile ad mediation for internal use or use an existing ad mediation solution (Fyber, Appodeal, Heyzap)? https://www.quora.com/Should-I-create-a-mobile-ad-mediation-for-internal-use-or-use-an-existing-ad-mediation-solution-Fyber-Appodeal-Heyzap/answer/Eric-Benjamin-Seufert, 2019. (Last accessed January 2019).

[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[35] C. Gao, Y. Man, H. Xu, J. Zhu, Y. Zhou, and M. R. Lyu. IntelliAd: Assisting Mobile App Developers in Measuring Ad Costs Automatically. In *Proceedings of the 39th International Conference on Software Engineering Companion*, ICSE-C '17, pages 253–255, 2017.

[36] C. Gao, J. Zeng, F. Sarro, M. R. Lyu, and I. King. Exploring the effects of ad schemes on the performance cost of mobile phones. In *Proceedings of the 1st International Workshop on Advances in Mobile App Analysis*, A-Mobile '18, pages 13–18, 2018.

[37] Google. Admob SDK out of main thread. https://groups.google.com/forum/#!searchin/google-admob-ads-sdk/background|sort:date/google-admob-ads-sdk/XkUAZ9RYfP0/XWqfefqcBwAJ. (Last accessed: March 2020).

[38] Google. Google AdMob - mobile ads SDK (Android) - get started. https://developers.google.com/admob/android/quick-start. (Last accessed: March 2020).

[39] Google. Google Ads Developer Blog. https://ads-developers.googleblog.com/2015/10/proguard-and-admob-mediation.html. (Last accessed: March 2020).

[40] Google. SDK-less Mediation: An easier way to mediate. https://www.blog.google/products/admob/sdk-less-mediation/, 2016. (Last accessed: March 2020).

[41] J. Gui, S. Mcilroy, M. Nagappan, and W. G. J. Halfond. Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE '15, pages 100–110, 2015.

[42] InMobi team. Seven reasons why you shouldn't ignore mobile ad mediation. https://www.inmobi.com/blog/2018/05/17/seven-reasons-why-you-shouldnt-ignore-mobile-ad-mediation. (Last accessed: March 2020).

[43] D. Kim, S. Son, and V. Shmatikov. What Mobile Ads Know About Mobile Users. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium*, NDSS '16, pages 1–14, 2016.

[44] L. Li, T. F. Bissyandé, J. Klein, and Y. L. Traon. An Investigation into the Use of Common Libraries in Android Apps. In *Proceedings of the 23rd Software Analysis, Evolution, and Reengineering*, SANER '16, pages 403–414, 2016.

[45] P. Mohan, S. Nath, and O. Riva. Prefetching mobile ads: Can advertising systems afford it? In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 267–280, 2013.

[46] MoPub. Initialize the MoPub SDK for Android. https://developers.mopub.com/publishers/android/initialize/. (Last accessed: March 2020).

[47] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan. Impact of Ad Libraries on Ratings of Android Mobile Apps. *IEEE Software*, 31(6):86–92, 2014.

[48] I. J. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. E. Hassan. Analyzing Ad Library Updates in Android Apps. *IEEE Software*, 33(2):74–80, 2016.

[49] P. Salza, F. Palomba, D. Di Nucci, C. D'Uva, A. De Lucia, and F. Ferrucci. Do Developers Update Third-party Libraries in Mobile Apps? In *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, pages 255–265, 2018.

[50] Statista. Number of mobile app downloads worldwide in 2017, 2018 and 2022 (in billions). https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. (Last accessed: March 2020).

[51] Tapjoy. Mobile ad mediation – what developers need to know. https://www.tapjoy.com/resources/mobile-ad-mediation/. (Last accessed: March 2020).

[52] E. Terkki, A. Rao, and S. Tarkoma. Spying on Android users through targeted ads. In *Proceedings of the 9th International Conference on Communication Systems and Networks*, COMSNETS'17, pages 87–94, 2017.

[53] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: Characterizing mobile advertising. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, pages 343–356, 2012.

[54] A. Vargha and H. D. Delaney. The kruskal-wallis test and stochastic homogeneity. *Journal of Educational and behavioral Statistics*, 23(2):170–192, 1998.

[55] H. Wang, H. Li, L. Li, Y. Guo, and G. Xu. Why are android apps removed from google play? a large-scale empirical study. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR 18, page 231242, 2018.

[56] Wikipedia. Executable and Linkable Format. https://en.m.wikipedia.org/wiki/Executable_and_Linkable_Format. (Last accessed: March 2020).

[57] D. Wilde. Google removes 24 more malware-filled apps that amassed 500,000 downloads. https://9to5google.com/2019/09/10/google-removes-malware-apps/. (Last accessed: March 2020).

Md Ahasanuzzaman is a graduate student in the Software Analysis and Intelligence Lab (SAIL) at Queen's University, Canada. His research interests include mining software repositories, analyzing mobile apps and app stores, analyzing community question answering sties, natural language processing, and machine learning. His works got published in top venues of Software Engineering (e.g., MSR, SANER, and EMSE). He obtained his BSc from the University of Dhaka (Department of Computer Science and Engineering), Bangladesh. He has been awarded prestigious awards, such as Deans scholarship award and Prime Minister Gold Medal for his outstanding achievements in the B.Sc program. More about Md Ahasanuzzaman can be read on his website:https://ahasanuzzaman.com/research/

Safwat Hassan currently works as a Post-doctoral Fellow in the Software Analysis and Intelligence Lab (SAIL) at Queen's University, Canada. Hassan worked as a software engineer for ten years in different corporations, including the Egyptian Space Agency (ESA), HP, EDS, VF Germany (outsourced by HP), and Etisalat. During his ten years in the software industry, he worked on a variety of software systems, such as web-based systems and embedded systems. He also participated in diverse project roles (e.g., design service, customer support, and R&D) across multiple business domains (e.g., telecommunication, supply-chain, and aerospace). His research interests include data mining for software engineering, mobile app store analytics, software architecture, system anomaly prediction, continuous integration, and software performance analytics. More about Safwat Hassan can be read on his website: https://safwathassan.com

Ahmed E. Hassan is an IEEE Fellow, an ACM SIGSOFT Influential Educator, an NSERC Steacie Fellow, the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He received a PhD in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. He also serves/d on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, and PeerJ Computer Science. More information at: http://sail.cs.queensu.ca/