Empirical Software Engineering (2026) 31:137
https://doi.org/10.1007/s10664-026-10872-w

Simply the best – A systematic evaluation approach for third-party libraries based on mobile app quality attributes

Rubén Saborido[1] · Rémy Raes[2] · Rodrigo Morales[4] · Romain Rouvoy[3] · Foutse Khomh[5] · Yann-Gaël Guéhéneuc[4]

Received: 21 July 2025 / Accepted: 24 April 2026

Abstract
Mobile device applications (apps) are complex because they rely on integrating multiple third-party libraries (TPLs). Yet, TPLs ease app development by offering implementations of specific functionality. For example, app developers often use advertising libraries to generate revenue, integrate social networking libraries to simplify login, or include crash reporting libraries to monitor/report crashes in their apps. However, there are multiple TPLs with similar functionalities from which to choose, and developers often cannot foresee all the consequences of using these libraries in their apps. The sizes of apps grow with the addition and usage of TPLs, and so does the number of required permissions and resource consumption. Thus, TPLs may degrade the quality of apps and developers need help measuring and comparing them. We propose EQuAT, an approach for Evaluating Quality Attributes of TPLs that eases the comparison of TPLs. EQuAT takes as input minimal apps that integrate TPLs and playable scenarios to simulate user interaction while exercising a particular functionality of the included TPL. By collecting quality metrics and comparing them using plots, we provide app developers with a systematic approach to rank TPLs based on their preferences. We show how EQuAT helps developers make informed decisions about which libraries to integrate into their apps by validating them against nine TPLs across three categories.

Keywords Android · TPL · Performance · Quality

Communicated by: Vincenzo Riccio.

Extended author information available on the last page of the article

Published online: 08 May 2026

1 Introduction

The pace of smartphone growth around the world is unprecedented, helping billions of new app users come online for the first time in what Google calls *Build for Billions*[1]. Thus, the use of the Android operating system continues to grow. There are now over three billion monthly active Android devices[2]. More than two million apps are available on the Google Play Store[3], and they were downloaded 113 billion times by people from 190 countries, generating $47 billion in revenue[4]. However, in many countries, users face constraints, such as (1) slow, intermittent, or expensive connectivity, (2) devices with low memory and poor processor performance, and (3) limited opportunities to recharge batteries during the day.[5] To address these users' needs, resources demanded by apps must fulfill user and–or device constraints.

Technology companies care about users' constraints, especially in developing countries, releasing lite versions of their apps. Facebook released Facebook Lite, an app designed to work on all networks, even slow or unstable connections. Facebook Lite is less than 2MB so it downloads quickly and uses less storage space. It has been downloaded more than 1 billion times. Similarly, TikTok Lite runs on slow networks, reduces data usage, and occupies only 9 MB of storage space. Designed for less than 2 GB of RAM, limited data, and–or 2G or 3G networks, this app has been downloaded more than 500 million times. Google released Android Go Edition[6], a "light" version of the Android operating system, designed for low-end and low-budget smartphones with 2GB or less of RAM and lower resource and bandwidth consumption. Google Play Services was also modularized to reduce its memory footprint. However, not all app developers have the resources to produce lite versions of their apps. On the contrary, they often use third-party libraries (TPLs) to speed up development without a clear understanding of the impact these TPLs have on the sizes, permissions, and performances of their apps. They also lack comparisons of the different TPLs to make informed decisions. Yet, the sizes, permissions, and performances of apps correlate with the addition and usage of TPLs. For example, Minelli and Lanza (2013) observed that external calls to TPLs represent more than 75% of the total number of method invocations in apps.

Users enjoy mobile apps that meet their quality expectations, but excessive battery, memory, and network usage, slow render times, crashes, large sizes, and excessive permissions are sources of frustration for them[7]: (1) larger apps take longer to download/install and take undue storage, (2) the numbers and types of requested user permissions affect download and retention[8], (3) energy consumption is critical because mobile devices depend on their batteries to work, (4-5) CPU and memory limit the number of apps users can run, and (6) network access may be expensive. In an internal analysis of app reviews on Google Play, Google reported that half of the one-star reviews (the lowest rating) mentioned app

[1] https://developer.android.com/distribute/best-practices/develop/build-for-the-next-billion.html, last accessed April 14th, 2026

[2] https://backlinko.com/iphone-vs-android-statistics, last accessed April 14th, 2026

[3] https://42matters.com/stats, last accessed April 14th, 2026

[4] https://www.businessofapps.com/data/android-statistics/, last accessed April 14th, 2026

[5] https://developer.android.com/docs/quality-guidelines/build-for-billions, last accessed April 14th, 2026

[6] https://developer.android.com/guide/topics/androidgo, last accessed April 14th, 2026

[7] https://android-developers.googleblog.com/2017/08/how-were-helping-people-find-quality.html, last accessed April 14th, 2026

[8] https://developer.android.com/training/articles/user-data-permissions.html, last accessed April 14th, 2026

quality[7]. Therefore, developers who focus on app quality can see improvements in their app ratings and, thus, their usages and monetization.

Developers need information on the impact of TPLs on the quality of their apps: app size, number and type of permissions, and performance.

Previous studies by Wang et al. (2015), on more than 100,000 Android apps from five different Android markets, showed that more than 60% of the sub-packages in Android apps are from TPLs. On average, TPLs account for more than 60% of the code in Android apps. Others studies showed (1) the inconsistencies between apps' descriptions and requested permissions due to the usage of TPLs (Pandita et al. 2013; Gorla et al. 2014), (2) the relationship between apps' ratings and TPLs (Ruiz et al. 2014), (3) the impact of TPLs on app clone detection (Wang et al. 2015), (4) privacy issues due to third-party tracking (Paci et al. 2023), and (5) the inconsistencies between withdrawal decisions and apps' third-party data collection (Du et al. 2024). Yet, a systematic literature review by Zhan et al. (2022) shows that no previous work proposed an approach to measure the impact of TPLs on the quality of apps: developers do not have the comparisons needed to make informed decisions about the TPLs to integrate (or not) in their apps.

Therefore, when choosing a TPL (e.g., for advertising, crash-reporting, or monitoring), developers typically proceed by making decisions without a systematic evaluation (Vargas et al. 2020):

- Relying on general reputation or popularity. The developer may assume that the most popular library (e.g., Google AdMob for advertising, Firebase for analytics) is the best choice. They may search online ("best advertising SDK for Android") and pick one from a blog post, Reddit, or Stack Overflow discussion. Without detailed performance testing, they risk choosing a bloated or inefficient library.
- Checking the library's stated performance claims. Developers might look at official documentation or the SDK's website, which often highlights performance benefits but downplays drawbacks (e.g., resource usage, battery drain). If the SDK claims to be "lightweight" or "optimized", they may take it at face value without further testing.
- Looking at APK or AAR file size only. While size is a factor, developers may ignore runtime performance, such as CPU and memory usage, if they focus only on the impact of TPLs on the app's size.
- Running basic profiling. An uninformed developer might not run profiling tools such as the Android Profiler or Battery Historian. If they do test, they may only check one metric (e.g., CPU usage in a short test session) rather than analyzing multiple performance aspects over time.
- Testing on a High-End Device Only If the developer tests performance, they might do so only on a flagship device (e.g., Pixel 8, Galaxy S24), where performance issues are less noticeable. They may not test on mid-range or low-end devices, which experience greater performance impacts from heavy TPLs.
- Ignoring background performance impact. Some TPLs consume resources even when not actively used (e.g., advertising TPLs making background network requests, analytics running periodic syncs, etc.). An uninformed developer might only test the library when actively used in the foreground, missing long-term performance degradation.

While previous studies have identified quality-related factors as strong determinants in TPL selection from a practitioner's perspective, there remains a lack of empirical frameworks that provide quantitative comparisons of quality attributes (Vargas et al. 2020), specifically for the mobile context. Thus, TPL selection is usually characterized by a reliance on proxy metrics and qualitative perceptions rather than empirical evidence. This reliance on subjective, ad hoc criteria creates a void. Therefore, a significant gap persists: the absence of empirical frameworks for the quantitative evaluation of TPL behavior in mobile environments. Our research fills this void by transitioning from subjective quality perceptions to a systematic assessment of TPLs on mobile devices.

We propose Evaluating Quality Attributes of TPLs (EQuAT), an approach to compare TPLs in terms of several quality attributes.

To clarify the roles involved in our approach, we distinguish between app developers (who integrate TPLs into their projects) and end users (who interact with the final application). Throughout this paper, 'developer' refers to the TPL consumer, while 'user' (or 'app user') refers to the individual experiencing the runtime impact.

The contributions presented in this article are the following:

- We present an approach to measure and compare the impact of TPLs on six quality attributes of apps: (1) sizes, (2) numbers and types of requested permissions, (3) energy consumptions, (4) CPU usages, (5) memory usages, and (6) network usages.
- We validate the proposed approach using nine Android TPLs across three categories: advertising, crash reporting, and monitoring.
- We describe the techniques and tools required to measure and compare the impact of different TPLs on these six metrics, particularly energy consumption.
- We invite replications and extensions by making publicly available the source code, scenarios, and instruments used in this article[9].

In the following, Section 2 describes the approach for measuring and comparing the impact of TPLs on apps' quality. Section 3 validates the proposed approach through validation with nine well-known Android TPLs in three popular TPLs categories. Section 4 discusses the validation. Section 5 discusses threats to the validity of our results. Section 6 summarizes related work, and Section 7 concludes and outlines some future work.

2 Measuring and Comparing the Impact of TPLs on App Quality

Previous studies comparing Android data structures (Saborido et al. 2018) and the impact of anti-patterns on mobile apps' energy consumption (Morales et al. 2018) have identified CPU, memory, and energy as relevant metrics for assessing the efficiency of Android implementations. These metrics are critical because they directly correlate with the overall smoothness of the user interface and the device's battery life. Furthermore, network

[9] https://github.com/Android-TPL-energy-consumption/in_situ_measurements (this repository will be updated after acceptation)

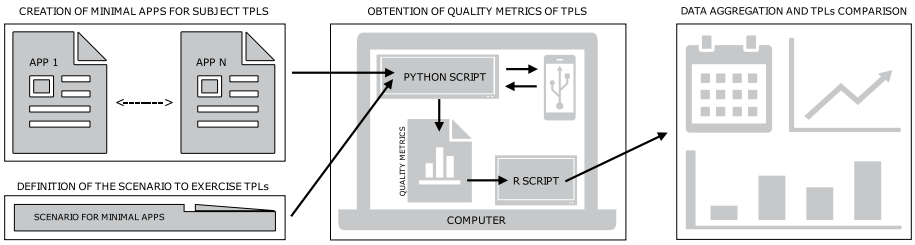

Fig. 1 Evaluating Quality Attributes of TPLs for comparing their impact on app quality

usage represents a high hidden cost in mobile apps, particularly when TPLs (such as ad networks or analytics) are involved. As demonstrated by Saborido et al. (2017), network activity is a decisive factor in distinguishing the resource overhead of different app versions and is often the primary driver of unexpected energy drain and data plan consumption. Thus, while CPU, memory, and network usages and energy consumption represent the operational cost, app size and permissions represent the architectural and privacy costs of TPLs. We consider these metrics as relevant quality attributes of mobile apps.

To measure and compare the impact of different TPLs on the quality attributes of apps we propose EQuAT, a four-step approach that automates running apps and collecting quality metrics on them. First, developers create a minimal app for each TPL under test (TPLUT). A minimal app is an app with the simplest graphical user interface (GUI) that includes the required components to exercise all the features of interest of the TPLUT. Second, they define as many scenarios as necessary to simulate a user's interaction with the TPLUT's features. Then, developers apply EQuAT on the resulting file of each minimal app and the defined scenario to automatically obtain the size of each app, the number and type of permissions required for each app, and the performance (CPU, memory, and network usages and energy consumption) of each app over several runs. Finally, it aggregates the collected data, compares the TPLs based on the quality metrics, and ranks them accordingly. Figure 1 summarizes the proposed approach.

EQuAT takes as input a path to folders containing the app files for minimal apps and the scenarios to execute, each corresponding to a single app. It automatically runs the apps, plays the scenarios, and collects the quality metrics. It generates a comma-separated values (CSV) file, containing all quality metrics for the TPLUT. Using this data, it aggregates quality metrics and generates plots to facilitate comparison of the TPLs. Each minimal app integrates one and only one TPLUT, and all the minimal apps share the same configuration and GUI. Therefore, our approach can attribute any difference in metric values between any pair of minimal apps to the TPLs used by the apps.

In the following, we provide further details of the whole process for measuring and comparing the impact of TPLs on app quality. We focus on Android apps without loss of generality for the proposed approach.

2.1 Creating Minimal Apps for TPLUT

EQuaT requires a minimal application for each TPLUT to ensure a fair comparison between different TPLs. Consequently, to spare developers the need to create minimal apps from scratch, we propose using of category-specific app templates. Each template is derived from a blank Android project to exclude unnecessary permissions or GUI overhead. Particularly,

a template consists of (i) a configuration layer defining the minimum SDK version and the essential permissions required for that category (e.g., INTERNET for Ads or Analytics), (ii) a GUI baseline, and (iii) the initialization of the TPL to trigger its primary functionality. Thus, these templates serve as a controlled baseline, ensuring that any measured differences in performance or resource usage are strictly attributable to the TPL rather than to variations in the host app's structure.

Developers can then derive minimal apps from each app template and integrate TPLUT into newly created apps: one TPLUT per minimal app. This guarantees all apps within a given category share the same configuration (e.g., the same manifest and minimum SDK) and GUI. This systematic approach minimizes variability and ensures that the differences measured (e.g., performance, APK size, and permissions) are attributable only to the integrated TPL, not to differences in the base app structure or GUI. To make the build process as similar as possible to the one developers use to release their apps to the marketplace, we recommend generating and signing the APK for the minimal app and building its release version. These building and signing are easy tasks for developers because minimal apps are just TPLUT integrations.

2.2 Defining the Scenario To Exercise TPLs Functionality

Developers define one playable scenario per TPL to run and exercise the TPLUT. We assume that developers want to compare TPLs in a category and that these TPLs offer similar features. Thus, developers define a unique scenario common to all the TPLs under study.

We recommend specifying usage scenarios as shell scripts, which are transferred to and executed on a test smartphone. Once deployed, these scripts simulate user interaction by running the app under test through a sequence of actions (such as `input tap ...` or `input type ...`) and sleep commands. Note that these actions ease the interaction with the phone in the same way app users would do: swipes, text inputs, etc.[10] This allows developers to programmatically interact with their apps. For example, Listing 1 shows a simple scenario that runs an app, simulates the user tapping at coordinates (X, Y), and wait for N seconds.

Listing 1 Example of scenario.

```
PACKAGE="myPackage"
ACTIVITY="myPackage.MainActivity"

# Launch app
am start -n $PACKAGE/$ACTIVITY

# Tap on coordinate (X, Y)
input tap X Y

# Wait few seconds to simulate user waiting
sleep N
```

[10] https://android.googlesource.com/platform/frameworks/base/+/android-4.4.2_r1/cmds/input/src/com/android/commands/input/Input.java#271, last accessed April 14th, 2026

Developers can define scenarios on Android phones by enabling developer mode and selecting the input option "Pointer location". This option shows current touch data on the screen, allowing developers to get coordinates of components of the GUI. To execute a scenario, EQuAT uploads it to the phone (`adb push /path/to/scenario.sh`) and executes it (`adb shell nohup sh /path/to/scenario.sh`), using the Android Debug Bridge (`adb`).

2.3 Measuring Quality Metrics

Given the app files of minimal apps and a scenario of usage, EQuAT uses a Python script that obtains for each minimal app its size (in MB), the number and type of permissions, the energy consumption (in J), and the CPU, memory, and network usages (in %, MB, and MB, respectively). To collect performance metrics, EQuAT runs each minimal app and plays the defined scenario N times, recording performance measurements. EQuAT automatizes the metrics measurement process without introducing variations in execution time due to user fatigue or skill level. This ensures that the results provide a meaningful representation of performance while remaining practical for automated pipelines.

Next, we describe the quality metrics of apps EQuAT collects:

- APK size (in MB) of each app file. It is the total size of a minimal application after a TPL has been successfully integrated and the app has been built.
- Number and type of permissions, using the command `aapt d permissions ...` available in the Android Software Development Kit (SDK). This command retrieves all declared Android application permissions from the APK's manifest. This includes standard permissions, as well as special, custom, or library-defined permissions.
- Energy consumption, using a hardware-based approach utilizing the Monsoon Low Voltage Power Monitor (LVPM)[11]. Monsoon offers a Python library that allows us to control the power monitor. We power a mobile phone via the LVPM, which is directly connected to the phone's motherboard, bypassing its battery to avoid any nterference. The power monitor is configured to supply the phone with the constant voltage its battery would provide in a normal setting, $4.4V$ in our case. The electric current drawn by the phone is then registered by the power monitor with a sampling frequency of $\approx 5kHz$. To ensure accurate energy measurements, the phone is not directly connected to the computer via USB, as this would power the phone and bypass the LVPM, leading to corrupted data. Instead, the USB connection passes through a power monitor that turns it off during experiments to prevent this issue. The total energy consumption is the sum of the energy associated with each sample. Note that, although there exist software-based approaches to measure the energy consumption of apps, as PETRA (Nucci et al. 2017), they rely on different sources of information to estimate the measurements, which are not guaranteed to be of high absolute accuracy. Thus, we decided to use a hardware-based approach.
- CPU usage, running in the background the `top` command periodically (at one-second intervals) on the phone, filtering by the name of the process associated with an app. It measures CPU usage as a percentage (%), indicating how much of the processor's capacity the system is currently using.

[11] https://msoon.github.io/powermonitor/, last accessed April 14th, 2026

- Memory usage, running in background the Android command `dumpsys meminfo` on the phone. It uses the Proportional Set Size (PSS), as the Android documentation suggests, to estimate the app's RAM use including shared pages across processes.[12] All RAM pages that are unique to a process directly contribute to its PSS value, while pages that are shared with other processes contribute to the PSS value only in proportion to the amount of sharing.
- Network usage, using the tool `tcpdump`[13] on the phone via the Android command `adb` to capture the number of bytes transmitted over the network connection. Network usage refers to the amount of data moving across a network (Wi-Fi, 3G, 4G, ...).

2.4 Comparing the Quality Metrics

Once EQuAT collects quality metrics for N runs, it uses an R script that aggregates all the values of each TPL using the median function. The aggregation uses the median because it is a robust measure of central tendency, while the mean is not. Then it applies statistical tests to determine whether the quality metrics differences between each pair of TPLs are significant. Following the evolution of statistical practices in empirical software engineering (de Oliveira Neto et al. 2019), EQuAT adopts a non-parametric by default strategy. While parametric tests are more powerful when normality and homoscedasticity assumptions are met, software performance data is frequently skewed and contains outliers that violate these assumptions. Thus, rather than performing automated normality tests, EQuAT performs the pairwise comparisons between different TPLs using the Wilcoxon rank-sum test (Hollander and Wolfe 1973), with a confidence level of 95% ($\alpha = 0.05$). The null hypothesis is that there is no systematic difference between the distributions of the two TPLs being compared. When a comparison is statistically significant (p-value < 0.05), EQuAT quantifies the magnitude of the difference using Cliff's δ (Cliff 1996). This non-parametric strategy is more conservative than parametric alternatives for the proposed approach, significantly reducing the risk of Type I errors (false positives) that occur when normality assumptions are violated.

As output, EQuAT reports quality metrics for each TPL into two different plots. One plot shows APK sizes, the number of permissions, and the distributions of performance metrics (energy consumption and CPU, memory, and network usages) for each TPL. A second plot shows the pairwise comparison of the TPLs under consideration for each quality metric. It includes the metric value for each TPL, the difference in metric values, whether the differences are significant, and, if so, the magnitude of the effect size. EQuAT also computes the TPLs ranking for each quality metric. First, it aggregates the data for each TPL and metric, computing the median over all the independent runs. Second, it obtains the ranking of each TPL for each quality metric using the aggregated data. With this information, developers can know which TPL is better than others for each quality metric.

[12] https://developer.android.com/studio/profile/investigate-ram.html, last accessed April 14th, 2026

[13] http://www.androidtcpdump.com/, last accessed April 14th, 2026

3 Validation

In this section, we validate EQuAT through a case study. Our hypothesis is that TPLs offering similar functionalities have a different impact on apps' quality: (1) sizes, (2) numbers and types of requested permissions, (3) energy consumption, (4) CPU usage, (5) memory usage, and (6) network usage. To illustrate how developers can leverage EQuAT in practice, we consider the following scenario. A development team is preparing a new mobile app and needs to integrate TPLs for crash-reporting and monitoring. In addition, the development team wants to get some revenue by including a TPL for advertising. There are multiple TPL candidates for crash-reporting, monitoring, and advertising. However, the team wants to select options that provide the required functionality while maintaining high performance and minimizing risks to privacy, security, and regulatory compliance. The team proceeds as follows:

- Identify candidate TPLs for each functional category. This is done manually by searching online, reviewing references, or consulting specialized developer forums.
- Generate the app template for each TPL category and derive a minimal app and its corresponding scenario for each TPL under evaluation. These scenarios provide representative usage patterns needed for accurate measurement.
- Run EQuAT to extract permissions, APK size, and runtime energy consumption and CPU, memory, and network usages.
- Compare overall quality and performance implications to understand the broader impact of each TPL. EQuAT provides a consistent basis for comparing all these dimensions across TPLs.
- Assess permission types and protection levels. EQuAT reports the list of permissions requested by each TPL. The developer then evaluates their protection level –*Normal*, *Dangerous*, *Signature*, or custom OS-level permissions– using internal policies or the Android documentation.
- Make an informed selection balancing functionality, performance, and risk. Based on the metrics, the team may decide to: prefer an advertising TPL with low energy consumption if energy or even CPU overhead is critical; select a monitoring library that minimizes network usage; adopt a crash-reporting tool that adds minimal app size.
- Document decisions for compliance and maintainability. Since EQuAT outputs structured, reproducible results, the team can record the metrics used for comparison, the justification for the decisions made, and any identified privacy or performance risks. This is particularly valuable for audits and compliance with regulations such as GDPR.

Next, we describe the objects of study (popular TPLs for advertising, crash reporting, and monitoring), the procedure for running EQuAT, and the results obtained.

3.1 Objects

AppBrain, a statistics Web platform, automatically analyzes Android apps on Google Play and retrieves all used libraries by analyzing the APK files. Allegedly updated daily, their website reports on the number of published applications, trending apps, download and rating statistics, and the popularity of Android libraries. Based on the Android library stats page[14], we select three of the most popular categories of TPLs and, for each of these cat-

[14] https://www.appbrain.com/stats/libraries, last accessed April 14th, 2026

Table 1 Popular Android TPLs for popular TPL categories

Name	Version	Provider	Category
Admob	20.5.0	Google	Advertising
Chartboost	9.1.0	Chartboost	Advertising
Max	11.5.3	AppLovin	Advertising
ACRA	5.9.7	ACRA	Crash-reporting
Crashlytics	29.0.4	Google	Crash-reporting
NewRelic	6.4.1	New Relic	Crash-reporting
Amplitude	2.23.2	Amplitude	Monitoring
Firebase	29.0.4	Google	Monitoring
NewRelic	6.4.1	New Relic	Monitoring

egories, three TPLs used by Android developers. We summarize this information in Table 1, where we also show the version and provider of each TPL.

Advertising TPLs allow developers to keep their content free and available while still making revenue. Analytic TPLs provide insights that helpdevelopers understand how their users are behaving, where they might be losing them, and more. Crash-reporting TPLs generate detailed error reports for apps, which are sent to developers.

3.2 Procedure

We created an app template tailored for each TPL category under study. These app templates are built to the minimum requirements necessary to integrate and execute the TPLUT. For advertising TPLs, the app template has a banner at the top of the device screen to load and show ads. For analytic TPLs, the app template defines a fragment pager adapter in the main activity with five fragments, each containing a different image and a string identifier (Image1, Image2, Image3, Image4, and Image5). Every time a user swipes left or right, the previous or next fragment is loaded, respectively, and an event is sent to the corresponding TPLUT server. Finally, for crash-reporting TPLs, the app template includes a button that throws a runtime exception to simulate a crash when clicked. From app templates, we derive a standalone minimal application for each TPLUT. Thus, minimal apps for TPLs in a category share the same GUI and functionality.

We then define one playable scenario per TPLUT, which specifically exercises the TPL core functionalities intended for integration into a final application. For advertising TPLs, we give libraries 40 seconds in the main activity to load an ad. For analytic TPLs, we navigate through the different fragments, then leave the app and wait for 12 seconds, for libraries to update the use reports to the respective backends. Finally, for crash-reporting TPLs, we click the button to throw a runtime exception, then wait 12 seconds for crash reports to be sent to servers.

Once the minimal apps have been developed and the scenarios defined, the corresponding APK file is then built for each minimal app. We then apply EQuAT, introduced in Section 2. It automatically runs minimal apps and their corresponding scenarios while collecting quality metrics to compare TPLs. To collect performance metrics, the approach runs each minimal app and plays the defined scenario N times independently. Finally, EQuAT aggregates all the data and generates multiple plots to facilitate the comparison of TPLs. We set the default value of N to 30, which is a conventionally accepted threshold in computer science performance evaluation (Georges et al. 2007). We clarify that this value does not inherently

guarantee statistical significance. However, it is a rationale-based default intended to provide sufficient observations for non-parametric tests to achieve acceptable statistical power. It provides a balance between achieving statistically significant results and minimizing the overhead of the data collection process, which can be time-consuming. Despite the previous, this value is not a fixed constant of the approach, but rather a configurable parameter that developers can adjust.

For the case study, we use a Samsung Galaxy A01 Core (also known as Galaxy M01 Core in India and as Galaxy A3 Core in Africa). It is a budget Android smartphone manufactured by Samsung Electronics as part of its A series. It was announced in July 2020 and launched in August 2020 as a low-cost phone. It features a 5.3-inch (14.6 mm) 720p display and runs One UI Core 2.0 on top of Android 10 Go Edition. It is a smaller and lighter version of the Samsung Galaxy A01.

3.3 Android Permission Types and Protection Levels

In addition to collecting the number of permissions of TPLs, we also analyze permission protection levels. Android[15] employs a permission system to control access to sensitive resources and user data. Permissions are categorized into different protection levels, including *Normal* permissions (which pose minimal risk to user privacy) and *Dangerous* permissions (which grant access to sensitive information, e.g., camera, location, contacts, etc., and require the user's explicit consent). We study the protection level of permissions introduced by TPLs on apps. We focus on dangerous permissions because they have the most significant impact on user privacy.

3.4 Results

Once EQuAT collects quality metrics of TPLs, it shows the distribution of each quality metric for each TPL. Figure 2 shows the distribution of each quality metric and TPL, grouped by TPL category. As expected, different TPLs have different quality metric values. For advertising TPLs, Admob increases both the APK size and the number of required permissions. Chartboost requires fewer permissions and, on average, uses less CPU, memory, and energy than Admob. Even though it requires more permissions than Chartboost, Max seems better on average across all the previously mentioned system metrics. All three libraries have comparable network usage. Concerning the crash-reporting category, ACRA increases APK size and network usage compared to the other TPLs in that category; however, it requires fewer permissions. Finally, for the monitoring category, energy consumption and CPU, memory, and network usages are very similar among Amplitude, Firebase, and NewRelic. Nonetheless, Firebase stands out for its larger APK size and more permissions. We discuss permissions qualitatively at the end of this section.

In addition to distributing metric values, EQuAT also computes the ranking of each TPL category and each quality metric. Table 2 shows the ranking of subject TPLs. With this information, developers can determine which TPL is better than others for different quality metrics. For the advertising TPL category, Admob ranks third for all quality metrics. Max TPL ranks first for APK size, energy consumption, and CPU and memory usages, but it requires more Android permissions and uses more data than Chartboost, which ranks best in

[15] https://developer.android.com/reference/android/Manifest.permission, last accessed April 14th, 2026

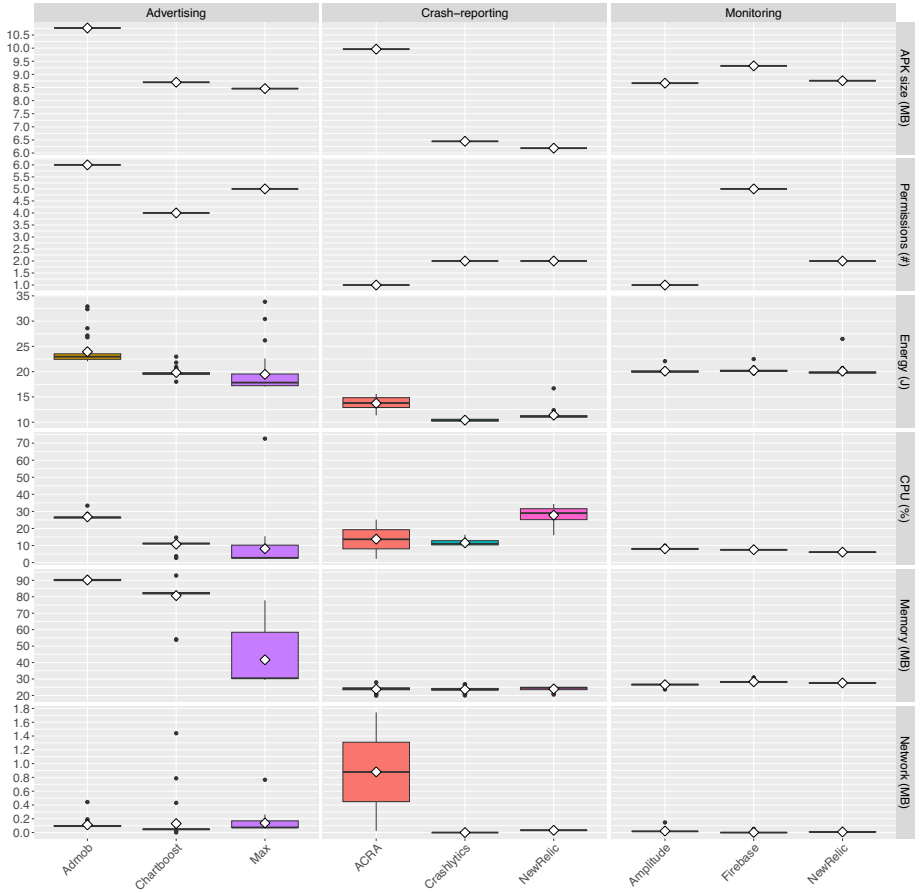

Fig. 2 Distribution of quality metrics for each TPL under study, grouped by TPL category. The lines in the boxes indicate the minimum value, lower quartile, median, upper quartile, and maximum values. The "◇" symbol represents the average value

Table 2 Rankings provided by EQuAT for each quality metric and TPL

TPL	APK size	#Permissions	Energy	CPU	Memory	Network	Category
Admob	3	3	3	3	3	3	Advertising
Chartboost	2	1	2	2	2	1	Advertising
Max	1	2	1	1	1	2	Advertising
ACRA	3	1	3	2	2	3	Crash-reporting
Crashlytics	2	2	1	1	1	1	Crash-reporting
NewRelic	1	2	2	3	3	2	Crash-reporting
Amplitude	1	1	2	3	1	3	Monitoring
Firebase	3	3	3	2	3	1	Monitoring
NewRelic	2	2	1	1	2	2	Monitoring

terms of the number of permissions and network usage. Regarding the crash-reporting TPL category, compared to ACRA and NewRelic, Crashlytics ranks first in performance metrics (energy consumption, and CPU, memory, and network usages). However, Crashlytics ranks second for APK size and number of permissions. Finally, concerning the monitoring TPL category, a trade-off occurs when comparing quality metrics. NewRelic ranks first for energy consumption and CPU usage, but second for APK size, number of permissions, and memory and network usages. Although Amplitude ranks first for APK size, number of permissions, and memory usage, it ranks third for CPU and network usage.

Although rankings are useful for intuitive comparisons, they do not allow informed decisions because differences between TPLs can be small or even not statistically significant. Figure 3 shows the pairwise comparison of the TPLs under study for each metric. Each cell contains the difference of the medians in %, the magnitude of the difference (in MB, number of permissions, Joules, %, MB, and MB, for APK size, permissions, energy

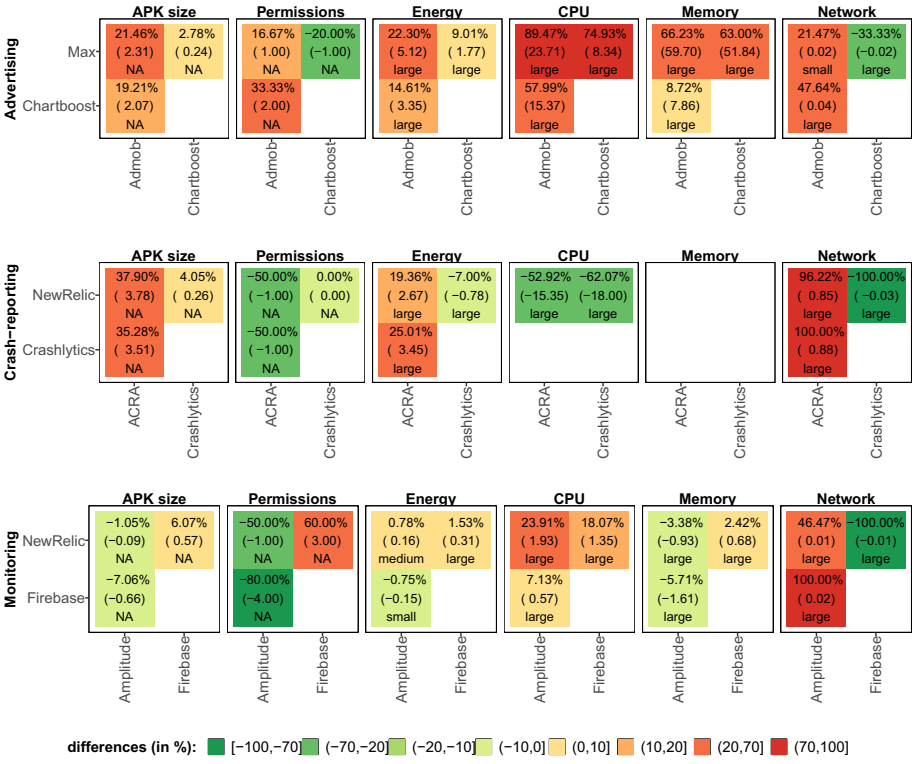

Fig. 3 Pairwise comparison of TPLs and quality metrics. Each cell contains the difference of the medians in %, the magnitude of the difference (in MB, number of permissions, Joules, %, MB, and MB, for APK size, permissions, energy consumption, CPU, memory, and network, respectively), and the magnitude of the effect size (small, medium, or large). The effect size is "NA" when it does not apply (*i.e.,* for APK size and permissions metrics where we only have a measurement for each TPL). Absent values indicate cases where there is not a statistically significant difference. For each cell in the matrix, negative differences (green colors) mean that the TPL at the bottom is better than the TPL at the left. On the contrary, positive differences (red colors) mean that the TPL at the bottom is worse than the TPL at the left. Therefore, greener colors have higher differences in favor of the TPL at the bottom and redder colors have higher differences in favor of the TPL at the left

consumption, CPU, memory, and network, respectively), and the magnitude of the effect size (small, medium, or large). The effect size is "NA" when it does not apply (*i.e.,* for APK size and permissions metrics where we only have a measurement for each TPL). Absent values indicate cases where there is not a statistically significant difference. Thus, for each cell in the matrix, negative differences (green colors) mean that the TPL at the bottom is better than the TPL at the left. On the contrary, positive differences (red colors) mean that the TPL at the bottom is worse than the TPL at the left. Therefore, greener colors show greater differences in favor of the TPL at the bottom, and redder colors show greater differences in favor of the TPL at the left. Only cases with a statistically significant difference are shown.

For the advertising TPL category, we observe that Admob increases APK size by 21.46% (2.31MB) and 19.21% (2.07MB) with respect to Max and Chartboost, respectively. Regarding the number of permissions, Chartboost requires 20% (1) afewer permissions than Max and 33.33% (2) fewer than Admob, respectively. We also observe that Max is the most energy-efficient TPL. Admob and Chartboost consumes 22.30% and 9.01% more energy than Max, respectively. Max also performs better than others in terms of CPU and memory usages, with large differences. Admob uses 89.47% and 66.23% more CPU and memory than Max, respectively. Chartboost uses 74.93% and 63% more CPU and memory than Max, respectively. However, Chartboost performs better than others in terms of network usage: it uses 33.33% and 52.36% (100%-47.64%) fewer data over the network than Max and Admob, respectively. Regarding the crash-reporting TPL category, ACRA and Crashlytics increase the APK size by 37.90% (3.78MB) and 4.05% (0.26MB), respectively, compared to NewRelic. Regarding the number of permissions, ACRA requires half as many as others. However, we observe that ACRA is the leasst efficient in terms of energy consumption and network usage. It consumes more energy (19.36% and 25.01%) and transmits more data over the network (96.22% and 100%) than NewRelic and Crashlytics, respectively. Regarding CPU usage, ACRA and Crashlytics use 52.92% and 62.07% less CPU than NewRelic, respectively. For the monitoring TPL category, Amplitude reduces APK size by 1.05% (0.09MB) and 7.06% (0.66MB) with respect to NewRelic and Firebase, respectively. Regarding the number of permissions, Amplitude requires 50% (1) fewer permissions than NewRelic and 80% (4) fewer than Firebase. In terms of energy consumption and CPU usage, NewRelic performs better than others. Amplitude and Firebase consume slightly more energy (0.78% and 1.53%) and use more CPU (23.91% and 18.07%) than NewRelic, respectively. Amplitude uses 3.38% and 5.71% less memory than NewRelic and Firebase, respectively. However, Amplitude transmits more data over the network (46.47% and 100%) than NewRelic and Firebase, respectively.

Next, we study the protection levels of permissions granted by TPLs to apps. Table 3 presents the Android permissions and their protection levels for the TPLs under study, illustrating the varying levels of risk associated with different permission types. The results demonstrate that integrating TPLs generally increases the risk associated with the requested permission types. Notably, libraries like Max and Firebase introduce two *Dangerous* permissions, whereas Admob introduces only one. The overall trend indicates that incorporating TPLs can lead to applications requiring more permissions, potentially raising concerns about user privacy and security.

Table 3 Android permissions and their protection level for the TPLs under study

TPL	Category	Permission	Protection level
Admob	Advertising	android.permission.INTERNET	Normal
Admob	Advertising	android.permission.ACCESS_NETWORK_STATE	Normal
Admob	Advertising	com.google.android.gms.permission.AD_ID	Dangerous
Admob	Advertising	android.permission.WAKE_LOCK	Normal
Admob	Advertising	android.permission.RECEIVE_BOOT_COMPLETED	Normal
Admob	Advertising	android.permission.FOREGROUND_SERVICE	Normal
Chartboost	Advertising	android.permission.INTERNET	Normal
Chartboost	Advertising	android.permission.ACCESS_NETWORK_STATE	Normal
Chartboost	Advertising	android.permission.READ_PHONE_STATE	Dangerous
Chartboost	Advertising	com.google.android.gms.permission.AD_ID	Dangerous
Max	Advertising	android.permission.INTERNET	Normal
Max	Advertising	android.permission.ACCESS_NETWORK_STATE	Normal
Max	Advertising	com.google.android.gms.permission.AD_ID	Dangerous
Max	Advertising	com.applovin.array.apphub.permission. BIND_APPHUB_SERVICE	Dangerous
Max	Advertising	android.permission.ACCESS_WIFI_STATE	Normal
ACRA	Crash-reporting	android.permission.INTERNET	Normal
Crashlytics	Crash-reporting	android.permission.INTERNET	Normal
Crashlytics	Crash-reporting	android.permission.ACCESS_NETWORK_STATE	Normal
NewRelic	Crash-reporting	android.permission.INTERNET	Normal
NewRelic	Crash-reporting	android.permission.ACCESS_NETWORK_STATE	Normal
Amplitude	Monitoring	android.permission.INTERNET	Normal
Firebase	Monitoring	android.permission.INTERNET	Normal
Firebase	Monitoring	android.permission.ACCESS_NETWORK_STATE	Normal
Firebase	Monitoring	android.permission.WAKE_LOCK	Normal
Firebase	Monitoring	com.google.android.finsky.permission. BIND_GET_INSTALL_REFERRER_SERVICE	Dangerous
Firebase	Monitoring	com.google.android.gms.permission.AD_ID	Dangerous
NewRelic	Monitoring	android.permission.INTERNET	Normal
NewRelic	Monitoring	android.permission.ACCESS_NETWORK_STATE	Normal

4 Discussion

Android developers often integrate TPLs into their apps to implement specific functionality. From the case study, we observed that TPLs impact the quality of Android apps. Thus, EQuAT could potentially provide developers with a mechanism to know the impact of TPLs on their apps. However, there is a trade-off between different TPLs and their impact on apps' quality metrics.

We suggest that developers select a TPL according to the specific context of use experienced by their users.

Let us focus, for instance, on emerging markets. Users in emerging markets own low-end devices with very limited CPU and memory. In addition, a majority of users in emerging markets face limited and expensive access to data connections.[5] Thus, it is important to prioritize app APK size and CPU, memory, and network usages in this context. Based on our

validation, Max is the most advisable advertising TPL to integrate in apps targeting emerging markets. This is the TPL that minimizes the APK size and is the most efficient in terms of energy consumption and CPU and memory usages. Using Max to show ads instead of the other advertising TPLs, developers can reduce app size by up to 21% and improve energy consumption and CPU and memory usages by up to 22%, 89%, and 66%, respectively. However, Chartboost transmits 33% less data over the network than Max. Regarding the crash-reporting TPL category, NewRelic reduces APK size by up to 38% while using more CPU than other solutions. Although Crashlytics increases APK size by 4% compared to NewRelic, it consumes less energy and uses less CPU and network than the others (differences in memory usage are not significant). Thus, Crashlytics could be advised for crash-reporting when targeting emerging markets. Concerning the monitoring TPL category, Amplitude minimizes APK size and memory usage, at the cost of increasing CPU and network usage by up to 24% and 100%, respectively, compared to other TPLs in the same category.

While prior studies often focus on the number of permissions requested by an app or its TPL, it is equally important to analyze the type of permission required. Not all permissions pose the same level of risk. *Normal* permissions allow access to low-impact features and do not require explicit user approval, whereas *Dangerous* permissions grant access to sensitive user data, such as device identifiers or location information, and must be explicitly granted by the user. Consequently, a single *Dangerous* permission can introduce significantly more risk than several *Normal* permissions. Moreover, certain permissions may have privacy and security implications that extend beyond their immediate functionality. For example, `READ_PHONE_STATE` is a dangerous permission that provides access to sensitive phone state information, including cellular network details, ongoing call status, and registered phone accounts. This permission is used by Chartboost. Additionally, several TPLs, such as Admob, Chartboost, Firebase, and Max, require non-standard Android OS-Level permissions such as `com.google.android.gms.permission.AD_ID`. This permission, defined within Google Play Services, is specific to Google's ecosystem, and since Android 12, Google requires apps that use the Advertising ID to explicitly declare this permission in their `AndroidManifest.xml`. Google also enforces policies restricting its use, particularly in response to privacy regulations like GDPR and CCPA[16]. In the context of this work, EQuAT directly supports this type-based permission analysis. EQuAT relies on the Android SDK tools to extract the complete list of permissions requested by each TPL. Android SDK tools provide the raw permission names exactly as declared in the TPL's manifest. Once these permissions are collected, the responsibility of determining their protection level lies with the developer or analyst using EQuAT. This design ensures flexibility, allowing the app developer to interpret risks in accordance with the latest Android documentation, regulatory requirements, or project-specific privacy considerations. By exposing the full set of requested permissions and enabling developers to cross-reference their protection levels, EQuAT facilitates a clearer understanding of the associated security and privacy risks introduced by TPLs. Thus, understanding the type of permissions requested by TPLs is critical for several reasons:

– Risk assessment. Different permission types introduce varying levels of security and privacy risks. Analyzing the type helps quantify potential threats more effectively than merely counting permissions.

[16] https://support.google.com/googleplay/android-developer/answer/6048248?hl=en, last accessed April 14th, 2026

- Compliance considerations. With data protection laws such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA), permissions that grant access to user data may raise regulatory concerns.
- User awareness and trust. App users often grant permissions without fully understanding their implications. Highlighting the type of permissions requested can improve transparency and user trust.
- Third-party influence on app security. Developers integrating TPLs may unintentionally introduce permissions that could compromise user privacy. Analyzing the type of permissions can help developers make informed decisions about which TPLs to include in their apps.

All the previous confirms that making decisions without a systematic evaluation of TPLs likely has a negative impact on app quality and performance. For instance, relying solely on TPLs' general reputation or popularity is not a better proxy for quality. Admob is a very popular TPL for advertising, but, based on our results, it performs worst across all metrics (size, permissions, energy consumption, and CPU, memory, and network usages) and introduces one *Dangerous* permission. Something similar happens with Firebase, which claims that "*Run your app with confidence and deliver the best experience for your users. Launch, monitor, and iterate with AI-assistive tools that help you optimize your app's quality and experience.*"[17]. However, based on our results, Amplitude and NewRelic are more energy-efficient for monitoring TPLs. Looking at APK size alone is neither recommended, as it ignores runtime performance. Based on our results, although NewRelic minimizes the APK size of apps in the crash-reporting category, it is the worst in terms of CPU and memory usages. Something similar applies to the monitoring category: Amplitude minimizes the APK size but uses more CPU and network than Firebase and NewRelic.

5 Threats to Validity

Threats to internal validity concern factors that could have influenced our results. We computed quality metrics (performance metrics, APK size, and number of permissions) using well-known approaches. Additionally, we repeated our measurements multiple times to confirm their statistical reliability. All minimal apps shared identical configurations and GUI, ensuring that observed differences can be attributed primarily to the integrated TPL rather than to variation in app structure. Threats to construct validity concern the relationship between theory and observation and the extent to which the measures represent real values. We used a Samsung Galaxy A01 Core phone. We measured power usage (energy consumption) at a sampling frequency one order of magnitude higher than in prior studies. However, with the physical measurement of energy consumption come the limitations of measurement and experimentation that exist in the natural sciences and engineering (Hindle 2016). CPU, memory, and network usages were collected using the commands `top` and `dumpsys`, and the tool `tcpdump` on the phone, respectively, which may have introduced extra energy consumption. To avoid any impact from other metrics' measurements, we collected power usage separately. The number

[17] https://firebase.google.com, last accessed April 14th, 2026

of permissions was collected using the command `aapt`, which is available in the Android SDK. Furthermore, our experimental design relies on minimal applications and controlled scenarios to isolate the impact of individual TPLs. While this design enables fair comparison across libraries, it may not capture all interactions present in complex real-world apps. Consequently, the measured metrics represent controlled approximations of real usage rather than exhaustive representations of all deployment contexts.

Threats to external validity concern the generalization of our findings. First, our experiments were conducted on a single device (Samsung Galaxy A01) running Android 10 Go Edition. Therefore, absolute metric values may differ on other devices, hardware configurations, or Android versions. Although we have no reason to believe that relative trends among TPLs will change fundamentally, replication across additional devices would strengthen generalizability. Second, the selection of TPL categories and libraries followed a purposive, non-random sampling strategy rather than a statistical sampling approach. We selected three widely used TPL categories and, within each, three popular libraries based on AppBrain usage statistics. This choice aimed to ensure practical representativeness by focusing on libraries commonly encountered by Android developers. This sampling decision reflects the primary goal of our study: not to produce an exhaustive catalog or ranking of all existing TPLs, but rather to validate and demonstrate a systematic methodology for comparing alternative libraries under controlled conditions. Consequently, our results should be interpreted as evidence of the applicability and usefulness of the proposed evaluation approach rather than as definitive quality rankings for the entire TPL ecosystem.

Threats to conclusion validity concern the relationship between the experiment and the outcome. We conducted our research using popular TPLs and applied appropriate statistical procedures. We paid attention not to violate assumptions of the constructed statistical models. In particular, we used non-parametric tests that do not assume the underlying data distribution.

6 Related Work

While the body of knowledge surrounding mobile app quality has grown significantly, the research has largely focused on optimizing code within the developer's direct control or detecting the presence of third-party components for security and licensing. Despite the fact that TPLs can represent more than 60% of an app's code (Wang et al. 2015) and account for the majority of external method invocations (Minelli and Lanza 2013), there remains a gap regarding their runtime behavior. In particular, there is an absence of empirical frameworks that enable the automated, dynamic evaluation of these libraries as independent units. In this section, we situate EQuAT within the broader landscape of performance optimization and TPL analysis, highlighting how our work addresses the current lack of quantitative benchmarks for TPLs comparison.

6.1 Analysis of Implementation and Design Decisions

Considerable research has investigated how developer decisions during the implementation phase dictate the eventual resource footprint of an app. This includes evaluating the impact of code obfuscation techniques (Sahin et al. 2014), identifying energy-greedy Android APIs (Linares-Vásquez et al. 2014), and measuring the overhead of specific system-level primitives such as storage subsystems (Mohan et al. 2017) or database transactions (Lyu et al.

2017). More recently, Saborido et al. (2018) provided a fine-grained comparison of primitive data structures (e.g., `ArrayMap` vs. `HashMap`), demonstrating that internal implementation choices can yield significant CPU and memory savings. Other studies shifted the focus toward automated optimization and refactoring. For instance, EARMO (Morales et al. 2018) was proposed to detect and automatically correct energy-intensive anti-patterns within an app's source code to extend battery life. These studies primarily focus on reactive optimization-refactoring or modifying internal source code after integration. In contrast, EQuAT enables proactive evaluation by treating TPLs as standalone units. By benchmarking TPLs in an isolated, minimal environment, our approach enable developers to compare TPLs across several quality attributes.

6.2 TPL Ecosystem: Identification, Selection, and Quality

As TPLs dominate a large portion of modern app packages (Wang et al. 2015; Minelli and Lanza 2013), research has intensified on their identification and their broader impact on app quality. Tools such as LibRadar (Ma et al. 2016), LibD (Li et al. 2017), and Libloom (Huang et al. 2023) address the technical challenge of accurately detecting TPLs in obfuscated or cloned binaries. Other studies have evaluated the external consequences of TPL usage, such as permission inconsistencies (Gorla et al. 2014) or the correlation (or lack thereof) between the number of ad libraries and user ratings (Ruiz et al. 2014). While identification tools are essential for security and compliance, they do not provide a quantitative assessment of a library's runtime behavior. Furthermore, studies on the performance impact of TPLs have historically been category-specific, focusing almost exclusively on ad networks (Gui et al. 2015; Saborido et al. 2017). EQuAT diverges from this by offering a general-purpose and automated benchmarking framework. We transition from the subjective quality perceptions reported by practitioners (Vargas et al. 2020) to an objective, evidence-based measurement of TPL quality.

7 Conclusion

Android is one of the most popular operating systems for mobile devices. Previous experiments showed that more than 60% of the sub-packages in Android apps are from TPLs. Thus, TPLs account for more than 60% of the code in Android apps. Other studies also found inconsistencies between Android app descriptions and requested permissions due to the use of TPLs.

Although there are multiple TPLs to choose from, developers need information on the impact of TPLs on appquality: app size, number of types and permissions, and performance. In this article, we proposed an approach for Evaluating Quality Attributes of TPLs (EQuAT) and analyzed the impact of TPLs on app size, permissions, and performance. EQuAT allows developers to make informed decisions about the TPLs to integrate into their apps, and it can be applied by developers at any time during the app development process. To validate EQuAT, we compared the quality metrics of popular TPLs across three categories: advertising, analytics, and crash reporting. The validation of our approach showed a trade-off between TPLs and their impact on different quality metrics. For the advertising category, Max has the least impact on APK size, energy consumption, and CPU and memory usages. However, Max performs worse than Chartboost in terms of network usage and requires

more permissions. For the crash-reporting category, Crashlytics consumes less energy and uses less CPU, memory, and bandwidth than ACRA and NewRelic. However, Crashlytics increases the APK size and requires more permissions than NewRelic and ACRA, respectively. For the monitoring category, Amplitude and NewRelic have different strengths and weaknesses, while Firebase is the most impactful for all considered metrics. Despite the above, Firebase introduces two *Dangerous* permissions, whereas Amplitude and NewRelic do not. Thus, we suggest that developers make their decision about integrating the TPL into their apps based on the context of use of their end users.

Despite the technical feasibility demonstrated by EQuAT, several limitations remain. Creating minimal apps for TPL comparison still requires manual effort, which we aim to mitigate through future automation. Furthermore, our current scope is restricted to app user-centric metrics, excluding developer-facing attributes such as maintainability or testability. We also acknowledge that our assumption of TPL overhead being additive in complex apps requires further empirical validation. Finally, since actual developers were not involved in this study, the practical utility and ease of integration of EQuAT from a practitioner's perspective remain to be confirmed. Moving forward, we leverage these limitations to define concrete research directions. We plan to develop IDE plugins or scriptable generators (potentially leveraging Artificial Intelligence) to automatically generate minimal app templates. We also aim to expand our quality model to incorporate ISO 25010 standards, such as testability and portability, through static analysis. To enhance the paper's practical relevance, future work will involve a study with professional developers to validate the framework's utility in their daily workflows. Lastly, we intend to integrate our approach with tools like LibRadar or LibD to provide automated suggestions for more efficient TPL alternatives.

Acknowledgments This work has been partly supported by the NSERC Discovery Grant program and the Canada Research Chair on IoT Holistic Softwarisation and by PID2022-142964OA-I00 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe".

Author Contributions All authors contributed to the conception and design of the study, data collection, analysis, and interpretation of results. All authors reviewed and approved the final manuscript.

Funding Funding for open access publishing: Universidad de Málaga/CBUA. Funding for open access charge: Universidad de Málaga / CBUA.

Data Availability All data related to this study are available on GitHub.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

Ethical Approval Not applicable.

Informed Consent Not applicable.

Clinical Trial Number Not applicable.

References

Cliff N (1996) Ordinal Methods for Behavioral Data Analysis. Erlbaum. https://books.google.ca/books?id=bIJFvgAACAAJ

de Oliveira Neto FG, Torkar R, Feldt R, Gren L, Furia CA, Huang Z (2019) Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. J Syst Softw 156:246–267. https://doi.org/10.1016/j.jss.2019.07.002, https://www.sciencedirect.com/science/article/pii/S0164121219301451

Di Nucci D, Palomba F, Prota A, Panichella A, Zaidman A, De Lucia A (2017) Software-Based Energy Profiling of Android Apps: Simple, Efficient and Reliable? In: Proceedings of the 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), Klagenfurt, Austria, pp 103–114

Du X, Yang Z, Lin J, Cao Y, Yang M (2024) Withdrawing is believing? detecting inconsistencies between withdrawal choices and third-party data collections in mobile apps. In: 2024 IEEE Symposium on Security and Privacy (SP), IEEE Computer Society, Los Alamitos, CA, USA, pp 14–14. https://doi.org/10.1109/SP54263.2024.00014, https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00014

Georges A, Buytaert D, Eeckhout L (2007) Statistically rigorous java performance evaluation. SIGPLAN Not 42(10):57–76. https://doi.org/10.1145/1297105.1297033

Gorla A, Tavecchia I, Gross F, Zeller A (2014) Checking App Behavior Against App Descriptions. In: Proceedings of the 36th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE 2014, pp 1025–1035. https://doi.org/10.1145/2568225.2568276

Gui J, Mcilroy S, Nagappan M, Halfond WGJ (2015) Truth in Advertising: The Hidden Cost of Mobile Ads for Software Developers. In: Proceedings of the 37th International Conference on Software Engineering (ICSE)

Hindle A (2016) Green software engineering: The curse of methodology. In: Leaders of Tomorrow Symposium: Future of Software Engineering, FOSE@SANER 2016, Osaka, Japan, March 14, 2016, pp 46–55. https://doi.org/10.1109/SANER.2016.60

Hollander M, Wolfe DA (1973) Nonparametric Statistical Methods. John Wiley & Sons, New York

Huang J, Xue B, Jiang J, You W, Liang B, Wu J, Wu Y (2023) Scalably detecting third-party android libraries with two-stage bloom filtering. IEEE Trans Software Eng 49(4):2272–2284. https://doi.org/10.1109/TSE.2022.3215628

Larios Vargas E, Aniche M, Treude C, Bruntink M, Gousios G (2020) Selecting third-party libraries: the practitioners' perspective. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA, ESEC/FSE 2020, p 245–256. https://doi.org/10.1145/3368089.3409711

Li M, Wang W, Wang P, Wang S, Wu D, Liu J, Xue R, Huo W (2017) LibD: scalable and precise third-party library detection in android markets. In: Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017, pp 335–346. https://doi.org/10.1109/ICSE.2017.38, http://doi.ieeecomputersociety.org/10.1109/ICSE.2017.38

Linares-Vásquez M, Bavota G, Bernal-Cárdenas C, Oliveto R, Di Penta M, Poshyvanyk D (2014) Mining Energy-greedy API Usage Patterns in Android Apps: An Empirical Study. In: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, New York, NY, USA, MSR 2014, pp 2–11. https://doi.org/10.1145/2597073.2597085, http://doi.acm.org/10.1145/2597073.2597085

Lyu Y, Gui J, Wan M, Halfond WGJ (2017) An Empirical Study of Local Database Usage in Android Applications. In: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)

Ma Z, Wang H, Guo Y, Chen X (2016) LibRadar: Fast and Accurate Detection of Third-party Libraries in Android Apps. In: Proceedings of the 38th International Conference on Software Engineering Companion, ACM, New York, NY, USA, ICSE '16, pp 653–656. https://doi.org/10.1145/2889160.2889178, http://doi.acm.org/10.1145/2889160.2889178

Minelli R, Lanza M (2013) Software Analytics for Mobile Applications-Insights & Lessons Learned. In: 17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013, pp 144–153. https://doi.org/10.1109/CSMR.2013.24

Mohan J, Purohith D, Halpern M, Chidambaram V, Reddi VJ (2017) Storage on Your SmartPhone Uses More Energy Than You Think. In: 9th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2017, Santa Clara, CA, USA, July 10-11, 2017. https://www.usenix.org/conference/hotstorage17/program/presentation/mohan

Morales R, Saborido R, Khomh F, Chicano F, Antoniol G (2018) Earmo: An energy-aware refactoring approach for mobile apps. IEEE Trans Software Eng 44(12):1176–120. https://doi.org/10.1109/TSE.2017.2757486

Paci F, Pizzoli J, Zannone N (2023) A comprehensive study on third-party user tracking in mobile applications. In: Proceedings of the 18th International Conference on Availability, Reliability and Security, Association for Computing Machinery, New York, NY, USA, ARES '23. https://doi.org/10.1145/3600160.3605079

Pandita R, Xiao X, Yang W, Enck W, Xie T (2013) Whyper: Towards automating risk assessment of mobile applications. In: Proceedings of the 22Nd USENIX Conference on Security, USENIX Association, Berkeley, CA, USA, SEC'13, pp 527–542. http://dl.acm.org/citation.cfm?id=2534766.2534812

Ruiz IJM, Nagappan M, Adams B, Berger T, Dienst S, Hassan AE (2014) Impact of Ad Libraries on Ratings of Android Mobile Apps. IEEE Softw 31(6):86–92. https://doi.org/10.1109/MS.2014.79

Saborido R, Khomh F, Antoniol G, Guéhéneuc YG (2017) Comprehension of Ads-supported and Paid Android Applications: Are They Different? In: Proceedings of the 25th International Conference on Program Comprehension (ICPC), IEEE, Buenos Aires, Argentina, pp 143–153. https://doi.org/10.1109/ICPC.2017.25

Saborido R, Morales R, Khomh F, Guéhéneuc YG, Antoniol G (2018) Getting the most from map data structures in android. Empir Softw Eng 23(5):2829–2864. https://doi.org/10.1007/s10664-018-9607-8

Sahin C, Tornquist P, Mckenna R, Pearson Z, Clause J (2014) How Does Code Obfuscation Impact Energy Usage? In: ICSME'14, pp 131–140

Wang H, Guo Y, Ma Z, Chen X (2015) WuKong: A Scalable and Accurate Two-phase Approach to Android App Clone Detection. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ACM, New York, NY, USA, ISSTA 2015, pp 71–82. https://doi.org/10.1145/2771783.2771795, http://doi.acm.org/10.1145/2771783.2771795

Zhan X, Liu T, Fan L, Li L, Chen S, Luo X, Liu Y (2022) Research on third-party libraries in android apps: A taxonomy and systematic literature review. IEEE Trans Software Eng 48(10):4181–4213. https://doi.org/10.1109/TSE.2021.3114381

Rubén Saborido is an Assistant Professor at the Institute for Software Engineering and Software Technology (ITIS) of the University of Málaga. He holds a PhD in Computer Engineering from Polytechnique Montréal (Canada) and a Degree in Computer Engineering, an M.Sc. in Software Engineering and AI, and a B.Sc. in Computer Systems from the University of Málaga (Spain). His doctoral thesis on search-based software engineering was nominated for the best thesis award in 2017. Following his PhD, he held a postdoctoral fellowship at Concordia University and was later awarded prestigious research grants in Spain, including the Juan de la Cierva (ranking in the top 5 candidates nationally). His research focuses on the intersection of artificial intelligence and software engineering, specifically on the use of evolutionary algorithms and preference-based multiobjective optimization to solve complex industrial problems. His research has been published in leading journals such as IEEE TSE, EMSE, and IEEE TEC, as well as in top conferences such as FSE and SANER. An active member of the research community, he has served on the program committees for GECCO and CEC since 2016 and is a recurring organizing committee member for the SERP4IoT workshop at ICSE. He has served as a guest editor for the IEEE Internet of Things Journal and acts as a scientific reviewer for leading journals. He has led a research project as Principal Investigator and maintains active collaborations with top-tier international researchers and industry partners.

Rémy Raes is a Ph.D. candidate in the Spirals research group (joint team with Inria and CRIStAL research laboratories). He received his Technological University Degree in Computer Science (2016), his BS. (2017), and MS. in Software Engineering (2019) from the University of Lille (France), and joined Inria in 2019 where he worked for four years as a research engineer on indoor location systems; in 2023, he started a Ph.D. thesis on distributed machine learning in ubiquitous environments using location-dependent models. His research interests include online and offline time series compression, local mobile computing, and communication challenges.

Rodrigo Morales is a Senior Lecturer in the Department of Computer Science and Software Engineering (CSSE) at Concordia University, where he has been teaching and conducting research since 2019. He received his B.Sc. in Computer Science (2005) and M.Sc. in Computer Technology (2008) from the National Polytechnic Institute of Mexico, where he subsequently served as a faculty member for five years. Prior to pursuing his doctoral studies, he also gained over three years of industry experience as a software developer in the banking sector. He earned his Ph.D. in Computer Engineering from Polytechnique Montréal in 2017, receiving the Best Thesis Award for his doctoral work. Dr. Morales Alvarado's research has been published in leading software engineering venues, including IEEE Transactions on Software Engineering, Empirical Software Engineering, and Journal of Systems and Software, as well as premier international conferences such as International Conference on Software Engineering and IEEE International Conference on Software Analysis, Evolution and Reengineering. Since 2019, he has co-organized the International Workshop on Software Engineering Research & Practices for the Internet of Things, co-located with ICSE, and actively contributes to the software engineering community through service on the program committees of COMPSAC, IEEE International Conference on Software Maintenance and Evolution, and IEEE/ACM International Conference on Program Comprehension. His research interests include the Internet of Things (IoT), software design quality, software energy efficiency, and mobile applications.

Romain Rouvoy is a full professor at the Department of Computer Science of the University of Lille (FR), which he has headed since 2024; he is the scientific leader of the Spirals team (a joint team with Inria and CRIStAL) on distributed systems and software sciences. He received a Ph.D. in Software Engineering from the University of Lille 1 in 2006. In 2016, he was nominated as Junior Fellow of the Institut Universitaire de France. From 2016 to 2025, he was chair of ACM Sigops France (ASF), and since 2023, he has been a member of the jury of the External Agrégation of Computer Science. His research sits at the intersection of software engineering and distributed systems, focusing on sustainable software services and user privacy. He has published papers in international conferences and journals, including IEEE/ACM ASE, IEEE/ACM CCGRID, IEEE SANER, and ACM WWW.

Foutse Khomh is an associate professor at Polytechnique Montréal, where he heads the SWAT Lab on software analytics and cloud engineering research (http://swat.polymtl.ca/). He received a Ph.D. in Software Engineering from the University of Montréal in 2010. His research interests include software maintenance and evolution, cloud engineering, service-centric software engineering, empirical software engineering, and software analytic. He has published over 125 conferences and journals papers. His work has received three ten-year Most Influential Paper (MIP) Awards, and four Best/Distinguished Paper Awards. He has served on the program committees of several international conferences including ICSM(E), SANER, MSR, ICPC, SCAM, ESEM and has reviewed for top international journals such as SQJ, JSS, EMSE, TSE, and TOSEM. He is program chair for Satellite Events at SANER 2015, program co-chair of SCAM 2015, ICSME 2018, and ICPC 2019, and general chair of ICPC 2018. He is one of the organizers of the RELENG workshop series (http://releng.polymtl.ca) and Associate Editor for IEEE Software.

Yann-Gaël Guéhéneuc is full professor at the Department of Computer Science and Software Engineering of Concordia University since 2017, where he leads the Ptidej team on evaluating and enhancing the quality of the software systems, focusing on the Internet of Things and researching new theories, methods, and tools to understand, evaluate, and improve the development, release, testing, and security of such systems. Prior, he was a faculty member at Polytechnique Montréal and Université de Montréal, where he started as an assistant professor in 2003. In 2025, he renewed my NSERC Research Chair Tier I on IoT Holistic Softwarisation, which had been first awarded in 2018 on Empirical Software Engineering for the IoT. In 2013-2014, for a sabbatical year, he visited KAIST, Yonsei University, and Seoul National University in Korea, as well as the National Institute of Informatics in Japan. In 2014, he received the NSERC Research Chair Tier II on Patterns in Mixed-language Systems. In 2010, he became an IEEE Senior Member. In 2009, he obtained the NSERC Research Chair Tier II on Software Patterns and Patterns of Software. In 2003, he received a Ph.D. in Software Engineering from the University of Nantes, France, under Professor Pierre Cointe's supervision. His Ph.D. thesis was funded by Object Technology International, Inc. (now IBM Ottawa Labs.), where he worked in 1999 and 2000. In 1998, he graduated as an engineer from École des Mines of Nantes (now École nationale supérieure Mines-Télécom Atlantique Bretagne–Pays de la Loire, IMT Atlantique). His research interests are program understanding and program quality, in particular through the use and identification of recurring patterns. He was the first to use explanation-based constraint programming in the context of software engineering to identify occurrences of patterns. He is also interested in empirical software engineering; he used eye-trackers to understand and to develop theories about program comprehension. He published papers in international conferences and journals, including IEEE TSE, Springer EMSE, ACM/IEEE ICSE, IEEE ICSME, and IEEE SANER. He was the program co-chair and general chair of several events, including IEEE ICPC'20 and '19, SANER'15, APSEC'14, and IEEE ICSM'13. He is the co-creator and co-organiser of the IEEE/ACM International Workshop on Software Engineering Research & Practices for the IoT (SERP4IoT) since 2019, the IEEE/ACM International Workshop on the Foundations of Applied Software Engineering for Games (FaSE4Games) since 2021, and the ReAnimate Summer School on Retro Gaming History, Critique, and Development, since 2024.

Authors and Affiliations

Rubén Saborido[1] · Rémy Raes[2] · Rodrigo Morales[4] · Romain Rouvoy[3] · Foutse Khomh[5] · Yann-Gaël Guéhéneuc[4]

✉ Rubén Saborido
rsain@uma.es

Rémy Raes
remy.raes@inria.fr

Rodrigo Morales
rodrigo.moralesalvarado@concordia.ca

Romain Rouvoy
romain.rouvoy@univ-lille.fr

Foutse Khomh
foutse.khomh@polymtl.ca

Yann-Gaël Guéhéneuc
yann-gael.gueheneuc@concordia.ca

[1] ITIS Software, Universidad de Málaga, Málaga, Spain

[2] Inria, Univ. Lille, CNRS, UMR 9189 CRIStAL, 59 000 Lille, France

[3] Univ. Lille, CNRS, Inria, UMR 9189 CRIStAL, 59000 Lille, France

[4] Dept. of Computer Science & Software Engineering, Concordia University, Montréal, Québec, Canada

[5] SWAT Lab., Mila - Quebec AI Institute, Montréal, Quebec, Canada