Quantifying the adoption of Kotlin on Android stores: Insight from the bytecode

Geoffrey Hecht
ISCLab, Department of Computer Science (DCC),
University of Chile, Chile

Alexandre Bergel
ISCLab, Department of Computer Science (DCC),
University of Chile, Chile

***Abstract*—Android apps have been traditionally built using Java since the inception of Android. However, Google announced Kotlin as an official supported language for the Android platform in May 2017. Since then, the popularity of Kotlin for Android projects has steadily increased, to the point that Google announced in 2019 that "Android development will be Kotlin-first" with nearly 60% of the top 1,000 Android apps containing Kotlin code. Yet, the transition from Java to Kotlin seems gradual and most applications still partially use Java. Outside open-source apps, little is known about the real proportion of code written in Kotlin inside apps. This paper supports a better understanding of the adoption of Kotlin in the Android ecosystem. We propose an approach to identify the language, Java or Kotlin, in which a class bytecode of an Android Package Kit (APK) originate from. We applied our model on more than 200k closed-source APKs from app stores and found that (i) most of the apps classes are still written in Java, indicating a mitigated adoption of Kotlin in less popular apps, (ii) the penetration of Kotlin is steadily increasing since 2017. We believe our insights are valuable to assess the adoption of Kotlin at large.**

I. Introduction

Kotlin is described as a modern, expressive and safer programming language than Java [1]. Some of the differences with Java, in addition to the more concise syntax, are default non-nullable reference types, data classes, and type inferences. Kotlin was designed with Java interoperability in mind so calling Java code from Kotlin (or Kotlin code from Java) is straightforward. On Android, Kotlin compiles to the same bytecode as Java, allowing a full compatibility.

Kotlin has become increasingly popular since it was made an officially supported Android programming language. Kotlin was the fastest growing language in 2018 on GitHub and was still ranked number four in 2019 [2]. Google claims that nearly 60% of the top 1,000 Android apps contain Kotlin code [3] whereas AppBrain states a market share of 75.95% for the top-500 US apps and 15.03% overall with over 125,000 apps using Kotlin [4]. It should be noted that the AppBrain dataset is also mostly composed of popular apps. Therefore, little is known about the adoption of Kotlin for less popular apps, although AppBrain data suggests that it is not as high. Moreover, AppBrain data does not tell us the proportion of code that is written in Kotlin. Indeed, detecting if an app features Kotlin code is trivial since the APK (package file) of an app will then have a *kotlin* folder at the root [5]. This folder contains the bytecode of the Kotlin Standard Library, hence, it is present as long as a class of the app (or one of its libraries) contains only a Kotlin class, but it does not give more information on the amount of Kotlin code. Knowing the easy interoperability with Java and that 86% of Kotlin users are still programming in Java [6], one might wonder if Kotlin's success is as great as these figures on popular apps suggest.

Nevertheless these numbers are still impressive for such a young language, and yet Kotlin is under-represented from publications on Android in the software engineering community. To illustrate this, we searched if Kotlin or Java were mentioned at least once in publications dealing mainly with Android of some reputed conferences (namely ICSE, MSR, SANER and MOBILESoft) between 2018 and 2020. The results are presented in Table I. Kotlin is mentioned only once in six publications [7]–[12] and one study focuses on its adoption [13], whereas Java is mentioned in about half of the publications. Of course, that does not invalidate the publications results since the conclusions of the publications are not necessarily language-dependent. But it does show that Kotlin is largely overlooked even when it could be relevant. For example, when providing prefetching technique to optimize app latency [14] or analyzing Android code smells from the source code of apps [15]. Some classes of the app might be overlooked while a Kotlin app is optimized in a different way than a Java app, and many Android code smells are language dependent.

	Mention	ICSE	MSR	SANER	MOBILESoft	Total
2018	Android	15	8	5	25	53
	Java	9	5	5	9	28
	Kotlin	0	0	0	2	**2**
2019	Android	11	9	8	19	47
	Java	4	6	3	10	23
	Kotlin	0	0	1	0	**1**
2020	Android	11	3	8	18	40
	Java	4	1	7	4	16
	Kotlin	1	1	1	1	**4**

TABLE I: Mentions of Kotlin and Java in publications focused on Android in ICSE, MSR, SANER and MOBILESoft

In this paper, we would therefore like to draw attention on the growing importance of Kotlin in the Android ecosystem and hope to pave the way for future studies that will consider Kotlin. First of all, in order to allow studies that are not limited to open-source applications, we propose the following research question:

RQ1: *Is it possible to differentiate Android bytecode that comes from Kotlin or Java classes?*

Subsequently, we did a preliminary study by applying our model on more than 200k apps, answering the following research question:

RQ2: *What is the proportion of Kotlin code over the years in our dataset?*

II. Related work

Kotlin being a novelty, publications concerning it are currently few and far between. Three publications are closely related to our work.

Oliveira *et al.* [13] performed a triangulation study on seven Android developers via interviews, to understand the perceptions of developers whom adopted Kotlin. They found that developers consider that Kotlin brings many advantages over Java, especially for code quality, readability, and productivity. However, they encounter new problems with the functional paradigm of Kotlin and the interoperation with Java.

Coppola *et al.* [16] analyzed a dataset of 1,232 open-source apps and evaluated their transition to Kotlin. They found that 19% of the apps featured Kotlin and that the transition from Java to Kotlin was usually fast and unidirectional. They also observed correlation between the presence of Kotlin code and the number of GitHub stars obtained.

Mateaus and Martinez [5] created a dataset of 2,167 open source apps and evaluated the quality of Android apps by analyzing the presence of code smells. They found 11.26% of apps featuring Kotlin and that for 63.9% of them the proportion of Kotlin increases along the app evolution. They also observed that the introduction of Kotlin in an app produced an increase of the quality in half of the apps.

These publications provide useful insights about the adoption of Kotlin and its potential impact on open-source apps. Our work is complementary, allowing for the analysis of the bytecode of millions of closed-source apps.

III. Differentiate bytecode from Kotlin and Java

In an Android APK, the classes' bytecode is stored inside classes.dex files, regardless of whether the original language is Java or Kotlin.

At first glance, the generated bytecode is similar between the two languages: they use the same keywords and structures. However, while reviewing this bytecode, a careful person may notice some recurring differences for a class written in Kotlin. For example, method calls to Kotlin standard lib functions can be observed. Also Kotlin bytecode will usually include metadata annotations, used by the reflection API, which are not usually present in bytecode produced by a Java compiler.

Unfortunately, these observations only hold if the app is not obfuscated. As soon as the classes, packages, methods are renamed and metadata annotations removed (default behavior of Proguard [17]) there no longer seems to be an easy and obvious way to differentiate bytecodes produced by the Kotlin compiler from the ones produced by the Java compiler.

We could, however, expect that the difference between Kotlin and Java will be reflected in the usage of the different keywords. That is why we decided to use the numerical statistic TFIDF (term frequency–inverse document frequency). Also, not knowing exactly which keywords will be affected, we decided to use a machine learning approach on top of TFIDF to determine which features are important and answer **RQ1**.

A. Dataset

To train our model, we collected all the latest versions of apps available in the open source app repository F-Droid [18] in October 2019. The repository contained 2010 open source apps from which we identified 299 apps featuring Kotlin.

For each app, F-droid provides us an APK and a corresponding source tarball. Our objective is to map the source classes to the resulting bytecode, and so identify if the bytecode originates from Java or Kotlin. However, when an app uses obfuscation we need the mapping files generated by Proguard to be able to perform this mapping since the name of classes are not kept. This file is not provided by F-Droid. We therefore needed to build these apps. 172 of the 299 apps were using Proguard, from which we were able to build 158 apps using a semi-automated approach. For all others apps (non-obfuscated and unable to build), we used the F-droid source tarball.

To obtain the features from the bytecode contained in the APK, we decompile the bytecode to the smali format using Apktool [19]. The smali format can be seen as equivalent of an assembler language for the Android bytecode. There is one smali files per class, including internal classes. These files are processed as text files and labeled as Kotlin or Java.

Within the 299 analyzed apps, we obtained a dataset of 51,120 Java classes and 44,198 Kotlin classes, which is then randomly balanced to 44,198 for both languages.

B. Features

To create the features, we first generate a vectors of words using TFIDF on the classes dataset. At first, we did not use a dictionary but then we realized that some app specific information, such as package name, were provoking overfitting when used with machine learning models.

Therefore we built a dictionary of 311 keywords[1]. The dictionary was generated using the documentation of Dalvik bytecode [20] using the syntax which is generated when the bytecode is transformed to smali. Therefore this dictionary contains words such as "move", "public", "goto/16", "method", etc. The dictionary also includes some recurrent hexadecimal values which are usually associated with specific accessFlags. The accessFlags are used to determine which are used to indicate the accessibility and overall properties of classes and class members. For example, accessFlags with the value *0x19* indicate a public (*0x01*), static (*0x08*), and final (*0x10*) class. We considered these possible values as important information, knowing that Kotlin considers each class as final, per default, and a class needs to be explicitly marked as "open" to allow inheritance, contrary to Java. Others keywords may reflect Kotlin specificities, for example, Kotlin does not offer a static keyword, developers have to create a companion objects to simulate Java static classes. Also *void* is replaced by *Unit* type in Kotlin.

[1]List of keywords : https://pastebin.com/UL13YgVm

We also added some keywords related to package and source code and are not always obfuscated such as “lkotlin”, “ljava”,“kt”, “jetbrains”, “jvm”. We expected these keywords to be a strong indicator (especially when specific to Kotlin) of the original language. Indeed in some case there will be inheritance or annotations specific to Kotlin, when there is no obfuscation, the name of the source file can also be present.

C. Results

Our problem may be expressed as a binary classification: a class is labelled as either Java or Kotlin. We compared the performance of four different machine learning classifiers: Random Forest, Linear Classifier, Naives Bayes and XGBoost.

To evaluate the performance of each classifier, we performed a 10-fold cross validation and calculated the mean precision, recall and F1-score, the results are presented in Table II.

	Precision	Recall	F1-score
Random Forest	0.97	0.96	0.96
Linear Classifier	0.95	0.93	0.94
Naives Bayes	0.94	0.76	0.84
XGBoost	0.96	0.93	0.95

TABLE II: Mean Precision, Recall and F1-score of classifiers in 10-Fold cross validation

All classifiers perform very well, especially for Random Forest with an F1-score of 0.96. We did not observe any difference of F1-score when the bytecode is obfuscated. After investigation, we found that mislabeled classes are often short, such as enumerations. They do not contains elements which are helpful to distinguish Java from Kotlin.

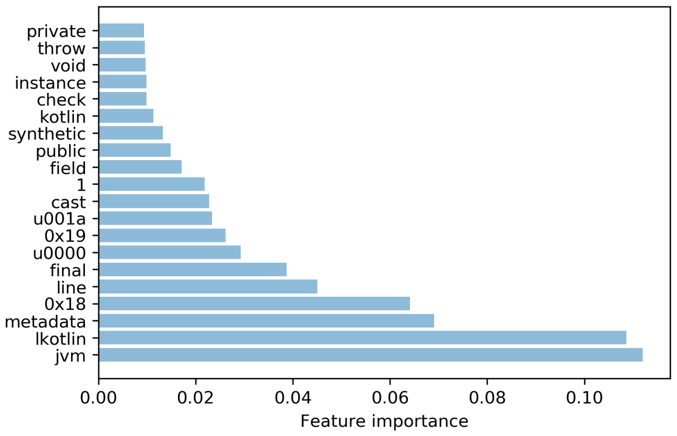

Fig. 1: Top 15 Feature importance of keywords with Random Forest Classifier

Figure 1 present the 15 most important features used by Random Forest. It provides a score that indicates how useful each feature was in the construction of the decision trees within the model. As mentioned in the previous section, we expected to observe such differences because of the peculiarities of Kotlin compared to Java, the Random Forest allows us to quantify their importance. We observe that the two most important keywords are related to Java and Kotlin packages used to perform calls. Kotlin metadata annotations are also well represented with the *metadata* keywords and common values for these metadata (*u0006*, *u001a*, *u0000*). We also observe keywords related to properties of class and methods, such as *final* or the *0x18* value of *accessFlags* presented in the previous subsection. Finally, there are some instructions such as *check*, *instance* or *cast* that appear at different frequencies for the two languages, especially when Java code is called from Kotlin code.

(RQ1) In summary, it is possible to differentiate bytecode that comes from Java or Kotlin classes with high precision and recall. Our best results were obtained, using a Random Forest classifier on a set of features generated using TFIDF on a set of bytecode keywords.

IV. Preliminary Study

Using our Random Forest classifier, we performed a preliminary study on a dataset of more than 201,000 randomly selected apps. The goal of this study is to further validate our model and to provide insights about the proportion of Kotlin code in Android apps and answer **RQ2**.

A. Dataset

We collected the APKs from the Androzoo dataset [21]. Androzoo is a growing collection of Android Apps collected from several apps stores, including the official Google Play Store, which currently contains more than 14 millions of mostly closed-source APKs.

We randomly selected APKs which were built between January 2017 and December 2020. Within a year, an APK is an unique app (there is no duplicate versions of it), however different versions of an app can be present in different years.

Our dataset is currently composed of 201,721 APKs[2].

The numbers of classes between APKs varies greatly as illustrated in Figure 2 (1552 APKs of more than 25,000 classes were excluded of this figure for visibility), the median number of classes is 4,637. We observe that apps tend to have more and more classes as the years go by.

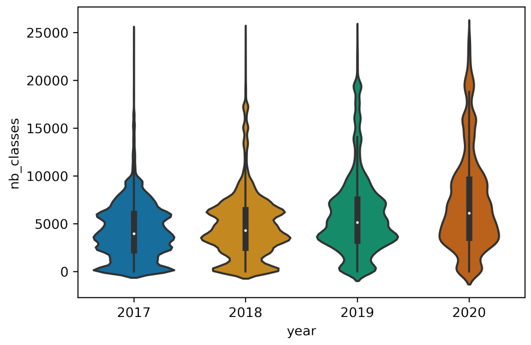

Fig. 2: Number of classes of APKs in the dataset

All these APKs were analysed using our Random Forest model. It should be noted that there is no difference between the bytecode of an app libraries and the app source code. Therefore, we also consider third-party libraries in this study.

[2]APKs list and raw results : https://zenodo.org/record/4660602

B. False positive validation

As mentioned in the introduction, the APK of an app featuring Kotlin will automatically contains a *kotlin* folder containing the Kotlin Standard Library bytecode at the root. Therefore, we know that if our classifier is detecting a Kotlin class in an APK without this folder, then it is a false positive.

Less than 5% of classes were classified as false positives in this situation. It is slightly worse than the 3% we expected considering the precision of our Random Forest model using the dataset of open-source apps, however it is in the same order of magnitude. We believe that this slight difference can be explained by the fact that non-Kotlin apps are overrepresented in this dataset (95% of APKs).

In the reminder of this paper our results are presented with these false positives corrected. Therefore, increasing the precision for non-Kotlin apps.

C. Results

Table III presents the results we obtained, and it clearly shows that the adoption of Kotlin is growing over the years.

The share of apps featuring Kotlin went from 0.24% in 2017 to 17.00% in 2020. Figures concerning the total proportion of Kotlin classes, seem less impressive at first glance, growing from 0.03% to 5.14%. But we should not forget that these results also include the embedded code of libraries, which could still be written in Java.

	2017	2018	2019	2020
number of apps	60793	66220	46127	28581
apps featuring Kotlin	145 (0.24%)	1600 (2.42%)	1222 (7.58%)	3738 (17.00%)
% of Kotlin classes (All apps)	0.03%	0.49%	1.76%	5.14%
% of Kotlin classes (Apps w/ Kotlin)	12.05%	8.62%	10.11%	15.10%

TABLE III: Results of the preliminary study, the last line only concern apps featuring Kotlin

If we focus on apps featuring Kotlin, we can see that a significant proportion of classes are written in Kotlin (around 15% in 2020). Interestingly, a high proportion of Kotlin classes can be observed in 2017 for such APKs. However, we can see in Figure 3 that the trend is increasing along the years. Since there is very few APKs featuring Kotlin in 2017, the overall percentage is heavily influenced by the few projects with a high proportion of Kotlin classes.

The Appbrain statistics made us suspecting that the adoption of Kotlin was slower in less popular apps. To observe this phenomenon, we wanted to find out if our dataset contained any popular apps. We downloaded the list of the top 100 most popular apps in each of the 58 categories of the Google Play Store in 2019. We found 561 of such apps in our dataset for 2019. The adoption of Kotlin is more important for these populars apps, culminating at 11.94% of apps featuring Kotlin in 2019 with a proportion of 12.68% of Kotlin classes. This limited dataset does not allow us to make any strong claims, however there seems to be a tendency for popular apps to adopt Kotlin faster as Appbrain's data suggested.

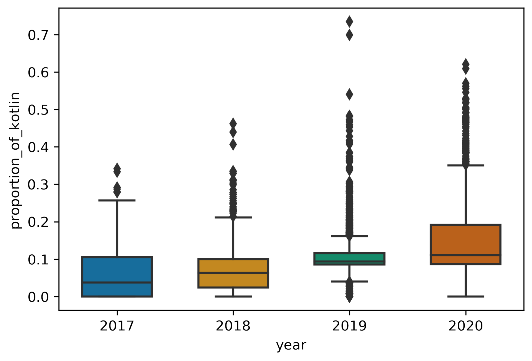

Fig. 3: Proportion of Kotlin classes in Apps featuring Kotlin

(RQ2) In summary, this preliminary study allowed us to confirm the good precision of our model. In our dataset, the penetration of Kotlin is increasing steadily but the proportion of Kotlin remains lower compared to Java. The adoption of Kotlin appears to be faster for popular apps.

V. Threats To Validity

Our model building relies on open-source apps, which are not representative of all apps. However, we could observe a good precision for non-Kotlin apps available on stores.

The only obfuscator used in our open-source dataset was Proguard, therefore we cannot guarantee that our results are equally valid when an alternative obfuscator is used. However, by separately testing obfuscated and non-obfuscated apps, we observed that the important features of our model vary little between the two. Moreover, previous studies indicate that Proguard is the most widely used obfuscator [22], [23].

Concerning our preliminary study, we do not claim that our dataset is representative of Android apps. Therefore the conclusion are not generalizable. Our goal, was to show a possible use of our model and to provide an insight of the adoption of Kotlin beyond the scope of open-source apps.

VI. Conclusion and Future Work

This paper presented a novel approach to differentiate which classes of an APK were written in Kotlin or Java with high precision and recall. We then performed a preliminary study on more than 200,000 apps and found that in our dataset, most of the bytecode comes from Java classes. However the adoption of Kotlin is steadily rising, especially in popular apps where the proportion of Kotlin code is already significant.

We believe our results can be key to answer a wide range of questions, including: How developers migrate from Java to Kotlin? Does Kotlin have an impact on apps quality? Does Kotlin affect developers' productivity? Is Kotlin also being adopted in libraries? How does Kotlin affect apps performance?

Before answering these questions, for future works, we would like to see how the apps integrate Kotlin over time and how the quality of apps is affected, similarly to what was done for open-source apps [5], [16].

Acknowledgements: This work is supported by Proyecto ANID/-FONDECYT Postdoctorado N°3180561, ANID/FONDECYT Regular project 1200067, and Lam Research.

References

[1] Google, "Develop android apps with kotlin," https://developer.android.com/kotlin, 2020, [Online; accessed 15 Jan-2020].

[2] GitHub, "The state of the octoverse : Top languages," https://octoverse.github.com/\#top-languages, 2019, [Online; accessed 15 Jan-2020].

[3] D. Winer, "Android's commitment to kotlin," https://android-developers.googleblog.com/2019/12/androids-commitment-to-kotlin.html, 2019, [Online; accessed 15 Jan-2020].

[4] AppBrain, "Android statistics / android libraries / kotlin," https://www.appbrain.com/stats/libraries/details/kotlin/kotlin, 2020, [Online; accessed 15 Jan-2020].

[5] B. G. Mateus and M. Martinez, "An empirical study on quality of android applications written in kotlin language," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3356–3393, 2019.

[6] Jetbrains, "Kotlin," https://www.jetbrains.com/lp/devecosystem-2019/kotlin/, 2019, [Online; accessed 15 Jan-2020].

[7] J. Businge, M. Openja, D. Kavaler, E. Bainomugisha, F. Khomh, and V. Filkov, "Studying android app popularity by cross-linking github and google play store," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 287–297.

[8] J. Wang, Y. Jiang, C. Xu, C. Cao, X. Ma, and J. Lu, "Combodroid: generating high-quality test inputs for android apps via use case combinations," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 469–480.

[9] P. Liu, L. Li, Y. Zhao, X. Sun, and J. Grundy, "Androzooopen: Collecting large-scale open source android apps for the research community," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 548–552.

[10] G. Hecht, C. Neverov, and A. Bergel, "Vision: alleviating android developer burden on obfuscation," in *Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, 2020, pp. 137–141.

[11] N. P. Borges, M. Gómez, and A. Zeller, "Guiding app testing with mined interaction models," in *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2018, pp. 133–143.

[12] R. Nunkesser, "Beyond web/native/hybrid: a new taxonomy for mobile app development," in *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2018, pp. 214–218.

[13] V. Oliveira, L. Teixeira, and F. Ebert, "On the adoption of kotlin on android development: A triangulation study," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 206–216.

[14] Y. Zhao, M. S. Laser, Y. Lyu, and N. Medvidovic, "Leveraging program analysis to reduce user-perceived latency in mobile applications," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 176–186.

[15] S. Habchi, N. Moha, and R. Rouvoy, "The rise of android code smells: who is to blame?" in *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press, 2019, pp. 445–456.

[16] R. Coppola, L. Ardito, and M. Torchiano, "Characterizing the transition to kotlin of android apps: a study on f-droid, play store, and github," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, 2019, pp. 8–14.

[17] Proguard, "Kotlin beta," https://www.guardsquare.com/en/products/proguard/proguard-manual-kotlin-beta, 2019, [Online; accessed 15 Jan-2020].

[18] "F-Droid," https://f-droid.org, [Online; accessed 15 Jan-2020].

[19] C. Tumbleson and R. Wiśniewski, "Apktool," https://ibotpeaches.github.io/Apktool/, [Online; accessed 15 Jan-2020].

[20] Google, "Dalvik bytecode," https://source.android.com/devices/tech/dalvik/dalvik-bytecode, 2019, [Online; accessed 15 Jan-2020].

[21] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 468–471.

[22] Y. Wang and A. Rountev, "Who changed you?: obfuscator identification for android," in *Proceedings of the 4th International Conference on Mobile Software Engineering and Systems*. IEEE Press, 2017, pp. 154–164.

[23] D. Wermke, N. Huaman, Y. Acar, B. Reaves, P. Traynor, and S. Fahl, "A large scale investigation of obfuscation use in google play," in *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 2018, pp. 222–235.