Frontmatter: Mining Android User Interfaces at Scale

Konstantin Kuznetsov
konstantin.kuznetsov@cispa.de
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany

Chen Fu*
fchen@ios.ac.cn
University of Chinese Academy of Sciences
Beijing, China

Song Gao*
gaos@ios.ac.cn
University of Chinese Academy of Sciences
Beijing, China

David N. Jansen*
dnjansen@ios.ac.cn
Institute of Intelligent Software
Guangzhou, China

Lijun Zhang*†
zhanglj@ios.ac.cn
University of Chinese Academy of Sciences
Beijing, China

Andreas Zeller
zeller@cispa.de
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany

ABSTRACT

We introduce Frontmatter: the largest open-access dataset containing user interface models of about 160,000 Android apps. Frontmatter opens the door for *comprehensive mining of mobile user interfaces*, jumpstarting empirical research at a large scale, addressing questions such as "How many travel apps require registration?", "Which apps do not follow accessibility guidelines?", "Does the user interface correspond to the description?", and many more. The Frontmatter UI analysis tool and the Frontmatter dataset are available under an open-source license.

CCS CONCEPTS

• **Software and its engineering** → *Automated static analysis*; • **Information systems** → **Data mining**; • **Human-centered computing** → *User interface design.*

KEYWORDS

App mining, user interfaces, Android, app stores, static analysis

ACM Reference Format:
Konstantin Kuznetsov, Chen Fu, Song Gao, David N. Jansen, Lijun Zhang, and Andreas Zeller. 2021. Frontmatter: Mining Android User Interfaces at Scale. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468264.3473125

1 INTRODUCTION

What are current trends in modern user interfaces? And what do these buttons all do? With Web pages and mobile apps being ubiquitous, it should be easy to analyze thousands or millions of user interfaces to answer such questions. However, only a *static code analysis* can reveal all aspects of a user interface. This already rules out Web applications from large-scale analysis, as the code is typically not available. And even for mobile apps, the analysis must provide an *accurate* and *comprehensive* mapping of user interface elements and functionality, which is not easy to obtain.

In this paper, we present Frontmatter: the largest and most precise repository of mobile graphical user interface (GUI) models to date. Applying a precise and scalable specialized static analysis on about 160,000 Android apps, we obtain visual, textual, structural, and interactive properties of their user interfaces, including their interplay with system services as well as with each other. Frontmatter thus opens the door for *comprehensive mining of mobile user interfaces*, jumpstarting empirical research at a large scale.

2 DATA COLLECTION

We now describe the source of the applications that was used to build the Frontmatter dataset. Next, we provide some details on the analysis infrastructure we developed. Then, we discuss the challenges and limitations that we faced when analyzing applications.

2.1 Data Source

Nowadays, there are dozens of Android app markets, of which the best known is the Google Play Store—the official market of Android—with almost 3 million available apps. However, the Google Play Store has features which complicate large-scale crawling, such as a limited number of downloads per day, account, and device. To build the Frontmatter data set, we therefore used AndroZoo [2], the largest publicly available dataset of Android applications. This repository provides unrestricted access to over 6 million apps crawled from different marketplaces, allowing straightforward reproducibility of research. We crawled AndroZoo for all latest versions of applications downloaded from the Google Play store in 2018, 2019, and the first half of 2020. In total, we gathered 423,583 APK files.

2.2 Mining Infrastructure

For our analysis, we developed the Frontmatter tool—a static analysis framework to automatically mine both user interface models and behavior of Android apps at a large scale with high precision.

*Also with State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China.
†Also with Institute of Intelligent Software, Guangzhou, China.

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473125

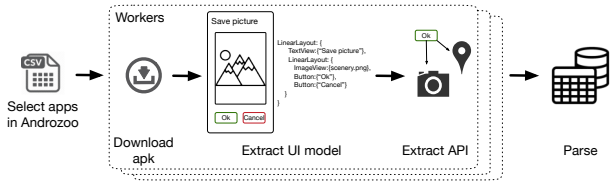


Figure 1: The workflow of FRONTMATTER

At the heart of the tool we used the SOOT [11] program analysis framework and BOOMERANG [10] approach, whose context-sensitive inter-procedural points-to analysis contributes to the high level of precision. All implementation details, as well as the comparison with competitors can be found in [5]. The code is available at https://github.com/uds-se/frontmatter.

The mining architecture is implemented on top of Luigi[1], which is a Python library allowing to build pipelines of batch jobs and boost parallel processing. Our mining workflow comprised the following steps (see Figure 1):

- First, we selected apps from the list of APKs provided by ANDROZOO[2] taking the latest version of apps uploaded to Google Play in 2018, 2019, and the first half of 2020;
- Next, we downloaded each apk file from ANDROZOO;
- Then, we ran FRONTMATTER and extracted UI hierarchies, followed by API analysis;
- Finally, for easier processing, we parsed the mined UI hierarchies and APIs into a table representation.

We limited the analysis time of each app to 25 minutes. Additionally, we set a timeout for each points-to analysis query to 20 seconds.

2.3 Mining Challenges

We encountered a couple of challenges preventing us from extracting UI hierarchies of some apps.

First, a lot of applications are games whose user interfaces are represented by drawings on a canvas. They do not contain standard Android UI widgets and, thus, cannot be analyzed with our tool.

Second, developers of contemporary apps tend to leverage various frameworks. Many of them allow to develop cross-platform apps. The code can be written in various languages, like C# or JavaScript, and then automatically transformed into Java classes. Usually, this transformation relies on reflection and internal APIs. Therefore, the final Java code uses custom loaders and does not operate with standard Android UI creation mechanisms, which prevents our analysis from reconstructing such UIs. The same hindrance occurs with web-based mobile app builder platforms (such as Kodular and AppyBuilder).

In order to identify platforms, we used several heuristics, especially involving the class names of activities:

- The `com.unity3d` prefix points to the Unity platform widely used in game development;
- Classes in the .Net framework MONO are prefixed with `mono.android`; while XAMARIN generates classes whose names start with a `md5` keyword followed by 32 characters;

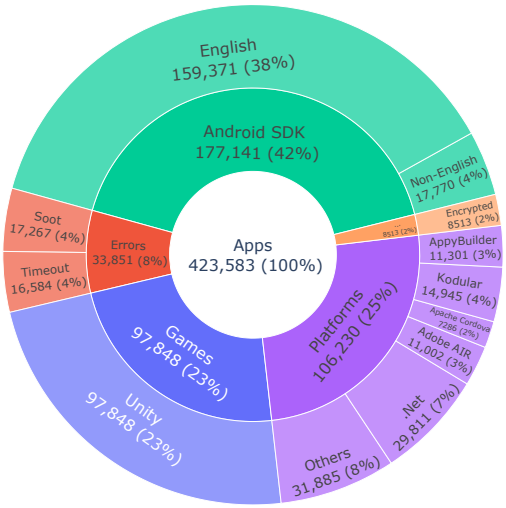


Figure 2: Distribution of analyzed APKs

- Apache Cordova–based apps tend to use listeners, whose names are prefixed by `org.apache.cordova`;
- The `air.` prefix indicates Adobe Air platform; `io.kodular` and `com.appybuilder` refer to Kodular and AppyBuilder.

If the activity classes declared in the manifest cannot be found by FRONTMATTER in the code, they are most frequently encrypted and stored in a separate file, which is dynamically loaded by the app at runtime.[3] When we could not recognize a hierarchy for every declared activity of an app, we considered it as implemented with a third-party framework. Moreover, some applications could not be analyzed due to errors raised by the SOOT static analysis framework or because of a timeout.

Finally, we focused our efforts on applications whose user interface is English. Although developers are encouraged to provide multilingual support, our preliminary analysis has shown that English resources are by far most complete. Therefore, we additionally identified the main language of the app and skipped the analysis of non-English applications. To this end, we extracted string resources of each app, combined them into one text, and applied the language detector by N. Shuyo[4].

Figure 2 shows the final distribution of apps in the FRONTMATTER dataset. A big portion of applications are games (which typically implement their own non-standard user interface) and cross-platform apps (around a quarter each). Still, we were able to extract hierarchies and behavior of about 160,000 English apps (38%) created with the standard Android SDK for user interfaces.

2.4 Static vs. Dynamic UI Analysis

As a static analysis tool, FRONTMATTER is subject to some inherent limitations. It is unable to extract those UI elements and their properties (like text labels), which are not available at static analysis time. This includes both the data transferred from a server at run time and that retrieved from databases. Furthermore, static analysis can be prone to over-approximation, reporting GUI features that

[1] https://github.com/spotify/luigi
[2] https://androzoo.uni.lu/static/lists/latest.csv.gz
[3] https://blog.zimperium.com/dissecting-mobile-native-code-packers-case-study/
[4] https://github.com/shuyo/language-detection/

are infeasible in actual executions. In contrast, dynamic analysis suffers from under-approximation, potentially missing UI elements and functionalities that are not exercised during testing.

In order to estimate how well our approach can capture UI features, we compared it with the Rico [3] dataset, which is *dynamically* obtained. To this end, we first downloaded the set of applications from AndroZoo that were analyzed in the Rico evaluation.[5] Next, Frontmatter produced results for 8,012 apps (the rest was based on non-analyzable platforms or failed with an error).

The main shortcoming of any dynamic analysis is the lack of coverage: Not all application screens and states can be explored. On average, Rico could cover only 25% of activities declared in the app manifest, of which Frontmatter identified 100%. For the activities covered by Rico, we used Frontmatter to extract GUI models and compared them against models reported by Rico. To this end, we collected UI elements from UI hierarchies and used *class name*, *resource id*, and *text* properties to identify matches. Although static analysis does not allow to extract dynamic content, on average, Frontmatter could retrieve only 35% fewer UI elements and their contents than were reported by dynamic analysis.

Sometimes, Frontmatter even identified more widgets than Rico, for instance, when a screen contained several fragments.

3 FRONTMATTER DATASET

Our dataset currently contains UI models for about 160,000 Android apps, which makes it, to the best of our knowledge, the largest open-access dataset containing not only precise GUI hierarchies, but also Android APIs invoked in response to a user interaction.

3.1 Data Organization

Each app is represented as two JSON files containing UI and API data. The UI file exposes mining metadata, a list of the app's declared activities with layouts and their GUI hierarchy, and a list of screen transitions. The API file contains information on which Android APIs can be triggered by an interaction with a particular UI element.

The mining metadata includes the default UI language of the app, its type—whether it is based on the standard Android SDK or a third party framework—, and the version of the mining tool.

Each activity may be composed of multiple layouts, since an activity can be assigned different layouts depending on the internal program state (e.g., for vertical and horizontal orientation). Sometimes Frontmatter recognizes fragments that are attached to a particular activity, but cannot identify a comprising container view. These fragments are listed in the *orphanedFragments* list.

Each layout contains a view hierarchy which captures all UI elements comprising a GUI, their properties, and attached listeners. For each UI element Frontmatter exposes its program properties such as class name, element id, and resource id; visual properties such as its relative position, dimensionality, displayed text, and icon; and structural properties such as a list of its children in the hierarchy. Among various ways to assign a label we recognize 'text', 'contentDescription', 'textOn', 'textOff', 'title', 'label', and 'hint' properties, and their respective setters. For icons we consider 'src' and 'background'. As storing icon files for such a big dataset would require a lot of space, currently we keep only icon file names. Still, usually they well describe the purpose of a UI element (e.g., *icon_delete.png* is used for the 'Delete' button), and the actual icon can be easily extracted from an apk on demand. The list of *listeners* describes interactive properties of an element, i.e., the ways a user can interact with it.

Along with the raw data, we provide *parsers* to transform hierarchies into the flattened table representation, collecting data on particular widgets or entire screens. The resulting tables can be conveniently analyzed with the *pandas* data manipulation tool or further converted into a SQL database.

Table 1: Breakdown of the Frontmatter Dataset

Apps	Screens	UI elements				APIs
		Total	w. Labels	w. Icons	Interactive	
159,371	1,462,782	26,255,086	9,434,176	3,738,814	2,990,532	125,177,717

3.2 Example Statistics

Table 1 shows the composition of the Frontmatter dataset. It is comprised of UI models of 159,371 applications. On average, each screen comprises around 18 UI elements, with 2 of them being interactive. Some widgets may trigger just a few Android APIs, like when starting a new activity, while others can call hundreds of methods performing database accesses or network communication. On average each listener invokes 42 APIs. Figure 3 shows the distribution of the UI sizes in the Frontmatter dataset.

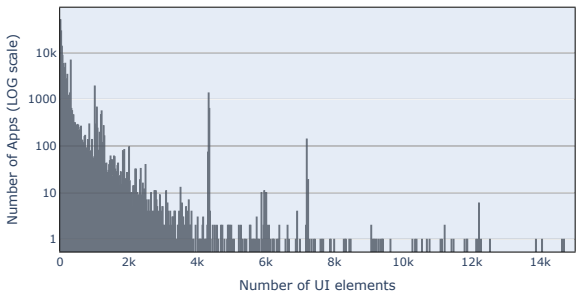


Figure 3: Distribution of UI size

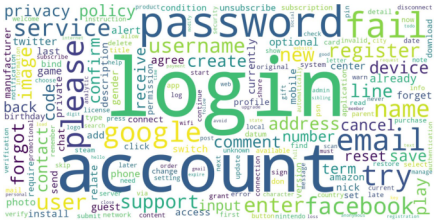
4 USE CASES

In this section, we present three possible applications of the Frontmatter dataset.

4.1 UI Elements Occurring Together

The first class of applications is straightforward, namely determining which UI elements occur in an app. Since the Frontmatter tool precisely determines the activities that UI elements are part of, one can use our dataset to find out *which UI labels occur together on a screen.* Such analyses could be helpful for assessing the

[5]The Rico paper [3] contains just the package name of an application and the date when it was downloaded from the Google Play Store, but not its version metadata. So it is impossible to identify which exact app was analyzed. Therefore, we applied a simple heuristic: we selected apps with the same package name and DEX file date as reported by Rico. In cases where no version of the app with an exact date match was found, we took the latest version published before the reported date. Out of 9,384 applications contained in Rico we were able to download 8,608 apps.



(a) UI labels occurring together with a "*login*" UI element

(b) APIs (Android classes) triggered by UI elements whose label is "*record*"

(c) Labels of UI elements interacting with the `FingerprintManager.authenticate()` API

Figure 4: Word clouds of APIs and UI elements extracted with FRONTMATTER

quality of a UI, for instance by determining how many different semantic concepts a screen is concerned with, or whether there is a reasonable mapping of concepts to screens. To allow for semantic search we associated each label with a numerical vector (a multi-dimensional meaning representation of a phrase) using *Word2Vec* embeddings [8] implemented in *SpaCy*[4]; Thus, for instance, querying "login" related elements would return not only exact matches but also semantically similar UIs like "signin".

As a simple example, consider Figure 4a, depicting a word cloud over all UI labels for all screens in AndroZoo that contain a "login" button. Not very surprisingly, "login" is the most common UI label, but we also see that it commonly occurs together with "password" and "account" labels. Though, only a small fraction of "login" screens refers to information about "privacy" or "policy".

4.2 APIs Invoked by Specific UI Elements

FRONTMATTER also contains information on which APIs are invoked by which UI elements. This allows to easily determine all UI elements associated with a specific set of APIs. Figure 4b shows a cloud of Android API classes whose methods are invoked by UI elements whose label is "record". We see that the Android `MediaRecorder` class is most frequently invoked; however, we also see that "record" buttons launch another activity via the `Activity` and `Intent` APIs.

Beyond empirical studies as discussed in Section 4.1, such associations can be used to detect *outliers*—that is, UI elements that access resources that users would not expect given the UI wording.

4.3 UI Elements Invoked By Specific APIs

Rather than looking up the APIs invoked by particular UI elements, we can also go the other way round and determine UI elements that interact with a particular API call. Figure 4c shows the labels of UI elements that all interact with the Android `FingerprintManager` API, which provides fingerprint authentication. The labels we see indeed all relate to authentication domains such as payment, cards, activations, currency, or wallets.

Typical applications of such a mapping are the same as in Section 4.2, namely empirical studies and anomaly detection. Here, an anomaly detection would detect mislabeled or uncommon UI elements. If, for instance, the term "amount" in a foreign language would have been translated to "quantity" instead of "amount", a distribution of words as in Figure 4c would show that in the given context (authentication with fingerprints), "amount" would be the more common choice of translation.

5 OTHER DATASETS

A number of researchers have conducted empirical research on Android. Shirazi et al. [9] collected 400 Android apps from the Google Play Store and analyzed the common design patterns of these apps. They estimated the complexity of each app design by counting the number of activities, layout files, and images, and computed descriptive statistics such as the most frequent interface elements. Alharbi et al. [1] applied a differential analyses to study design pattern changes over time in 24,436 Android apps. Deka et al. [3] created the RICO dataset by dynamically mining apps and captured view hierarchies, screenshots, and user interactions. The RICO dataset contains design data from more than 9.7k Android apps. It exposes visual, textual, structural, and interactive design properties of more than 72k unique UI screens. Based on the RICO dataset, Micallef et al. [7] investigated whether smartphone apps use login features, and what relationships exist between login features and app popularity. In contrast to RICO, FRONTMATTER relies on *static* analysis, which allows to determine and assess *all* user interface elements without having to explore them at runtime. Liu et al. [6] proposed an automatic approach for annotating mobile UI elements with both structural semantics such as buttons or toolbars and functional semantics such as add or search, and computed semantic annotations for the 72k unique UIs in the RICO dataset. This technique can be used to augment FRONTMATTER with UX concepts as well.

6 CONCLUSION

In this work, we presented the FRONTMATTER dataset containing around 160,000 UI models of Android applications which include apps' accessible screens, user interface elements, their textual and graphical features, as well as Android APIs invoked in response to user interaction. This dataset opens the door for *comprehensive mining of mobile user interfaces*, jumpstarting empirical research at a large scale. The FRONTMATTER dataset is available under an open-source license at

https://zenodo.org/record/5084655

ACKNOWLEDGMENTS

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant No. ZE 509/9-1, by NSFC of China (Grant No. 61761136011), and by the Guangdong Science and Technology Department (Grant No. 2018B010107004).

REFERENCES

[1] Khalid Alharbi and Tom Yeh. 2015. Collect, Decompile, Extract, Stats, and Diff: Mining Design Pattern Changes in Android Apps. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Copenhagen, Denmark) *(MobileHCI '15)*. Association for Computing Machinery, New York, NY, USA, 515–524. https://doi.org/10.1145/2785830.2785892

[2] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 468–471. https://doi.org/10.1145/2901739.2903508

[3] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 845–854. https://doi.org/10.1145/3126594.3126651

[4] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python.* https://doi.org/10.5281/zenodo.1212303

[5] Konstantin Kuznetsov, Chen Fu, Song Gao, David N. Jansen, Lijun Zhang, and Andreas Zeller. 2021. What do all these Buttons do? Statically Mining Android User Interfaces at Scale. arXiv:2105.03144 [cs.SE]

[6] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, and Ranjitha Kumar. 2018. Learning Design Semantics for Mobile Apps. In *The 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18)*. ACM, New York, NY, USA, 569–579. https://doi.org/10.1145/3242587.3242650

[7] Nicholas Micallef, Erwin Adi, and Gaurav Misra. 2018. Investigating Login Features in Smartphone Apps. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers* (Singapore, Singapore) *(UbiComp '18)*. Association for Computing Machinery, New York, NY, USA, 842–851. https://doi.org/10.1145/3267305.3274172

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2* (Lake Tahoe, Nevada) *(NIPS'13)*. Curran Associates Inc., Red Hook, NY, USA, 3111–3119. https://doi.org/10.5555/2999792.2999959

[9] Alireza Sahami Shirazi, Niels Henze, Albrecht Schmidt, Robin Goldberg, Benjamin Schmidt, and Hansjörg Schmauder. 2013. Insights into Layout Patterns of Mobile User Interfaces by an Automatic Analysis of Android Apps. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (London, United Kingdom) *(EICS '13)*. Association for Computing Machinery, New York, NY, USA, 275–284. https://doi.org/10.1145/2494603.2480308

[10] Johannes Späth, Karim Ali, and Eric Bodden. 2019. Context-, flow-, and field-sensitive data-flow analysis using synchronized Pushdown systems. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29. https://doi.org/10.1145/3290361

[11] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. 1999. Soot – a Java Bytecode Optimization Framework. In *Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collaborative Research* (Mississauga, Ontario, Canada) *(CASCON '99)*. IBM Press, 13–. http://dl.acm.org/citation.cfm?id=781995.782008