An Early Rico Retrospective: Three Years of Uses for a Mobile App Dataset

Biplab Deka, Bardia Doosti, Forrest Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Ranjitha Kumar, Tao Dong, and Jeffrey Nichols

Abstract The Rico dataset, containing design data from more than 9.7 k Android apps spanning 27 categories, was released in 2017. It exposes visual, textual, structural, and interactive design properties of more than 72 k unique UI screens. Over the years since its release, the original paper has been cited nearly 100 times according to Google Scholar and the dataset has been used as the basis for numerous research

B. Deka
McKinsey, Chicago, IL, USA
e-mail: biplab.uiuc@gmail.com

B. Doosti · C. Franzen · D. Afergan · Y. Li · T. Dong
Google, Inc., Mountain View, CA, USA
e-mail: bardiad@google.com

C. Franzen
e-mail: cfranzen@google.com

D. Afergan
e-mail: afergan@google.com

Y. Li
e-mail: liyang@google.com

T. Dong
e-mail: taodong@google.com

F. Huang
University of California, Berkeley, CA, USA
e-mail: forrest_huang@berkeley.edu

J. Hibschman
Northwestern University, Evanston, IL, USA
e-mail: jh@u.northwestern.edu

R. Kumar
University of Illinois at Urbana-Champaign, Champaign, IL, USA
e-mail: ranjitha@illinois.edu

J. Nichols (✉)
Apple, Inc., Seattle, WA, USA
e-mail: jeff@jeffreynichols.com

Y. Li and O. Hilliges (eds.), *Artificial Intelligence for Human Computer Interaction: A Modern Approach*, Human–Computer Interaction Series,
https://doi.org/10.1007/978-3-030-82681-9_8

projects. In this chapter, we describe the creation of Rico using a system that combined crowdsourcing and automation to scalably mine design and interaction data from Android apps at runtime. We then describe two projects that we conducted using the dataset: the training of an autoencoder to identify similarity between UI designs, and an exploration of the use of Google's Material Design within the dataset using machine learned models. We conclude with an overview of other work that has used Rico to understand our mobile UI world and build data-driven models that assist users, designers, and developers.

1 Introduction

We created the Rico[1] dataset and released it publicly in 2017 [21]. We believe it is still the largest repository of Android mobile app designs, comprising visual, textual, structural, and interactive property data for 72,219 UIs from 9,772 apps, spanning 27 Google Play categories. For each app, Rico presents a collection of individual user interaction traces, as well as a collection of unique UIs determined by a novel *content-agnostic similarity heuristic*. In total, the dataset contains more than 72 k unique UI screens.

To download the Rico dataset and learn more about the project, please visit http://interactionmining.org/rico.

To understand what makes Rico unique, it is helpful to consider it in the context of other Android app datasets. Existing datasets expose different kinds of information: Google Play Store metadata (e.g., reviews, ratings) [2, 25], software engineering and security related information [24, 52], and design data [7, 22, 45]. Rico captures both design data and Google Play Store metadata.

Mobile app designs comprise several different components, including user interaction flows (e.g., search, login), UI layouts, visual styles, and motion details. These components can be computed by mining and combining different types of app data. For example, combining the structural representation of UIs—Android *view hierarchies* [3]—with the visual realization of those UIs—screenshots—can help explicate app layouts and their visual stylings. Similarly, combining *user interaction* details with view hierarchies and screenshots can help identify the user flows that apps are designed to support.

Figure 1 compares Rico with other popular datasets that expose app design information. Design datasets created by *statically* mining app packages contain view hierarchies, but cannot capture data created at runtime such as screenshots or interaction details [7, 45]. ERICA's dataset, on the other hand, is created by *dynamically* mining apps and captures view hierarchies, screenshots, and user interactions [22].

Like the ERICA dataset, Rico is created by mining design and interaction data from apps at runtime. Rico's data was collected via a combination of human-powered and programmatic exploration, as shown in Fig. 2. Also like ERICA, Rico's app

[1] Rico—a Spanish word meaning "rich".

	Year	# Apps	# UIs	Mining	View Hierarchies	Screenshots	User Interactions
Shirazi et al.	2013	400	29K	Static	●	○	○
Alharbi et al.	2015	24K	-	Static	●	○	○
ERICA	2016	2.4K	18.6K	Dynamic	●	●	●
Rico	2017	9.7K	72.2K	Dynamic	●	●	●

Fig. 1 A comparison of Rico with other popular app datasets

mining infrastructure requires no access to—or modification of—an app's source code. Apps are downloaded from the Google Play Store and served to crowd workers through a web interface. When crowd workers use an app, the system records a *user interaction trace* that captures the UIs visited and the interactions performed on them. Then, an automated agent replays the trace to "warm up" a new copy of the app and continues the exploration programmatically. By combining crowdsourcing and automation, Rico can achieve higher coverage over an app's UI states than either crawling strategy alone.

Rico is four times larger than the ERICA dataset and presents a superset of its design information. Rico also exposes an additional view of each app's design data: while ERICA provides a collection of individual user interaction traces for an app, Rico additionally provides a list of the unique UIs discovered by aggregating over user interaction traces and merging UIs based on a similarity measure. This representation is useful for training machine learning models over UIs that do not depend on the sequence in which they were seen. Lastly, Rico annotates each UI with a low-dimensional vector representation that encodes layout based on the distribution of text and images, which can be used to cluster and retrieve similar UIs from different apps.

We chose to release the Rico dataset publicly as we believed others in the research community might benefit from it, and we were especially optimistic that the dataset would find use because it is large enough to support deep learning applications. We envisioned that the dataset would be used for applications related to user interface design, such as UI layout generation and UI code generation, similar to the directions that we were pursuing when we created it. To our surprise, the community has found much broader use cases for the dataset than we could have imagined. Over the years since Rico's release, the original paper [21] has been cited nearly 100 times according to Google Scholar and the dataset has been used as the basis for numerous research projects, including explorations of the usage of specific features within the app ecosystem [39, 44], the creation of intelligent assistants [47], and mapping natural language to UI actions [34]. The dataset has also been extended by a number of research groups, who have added semantic metadata [37], language mappings [34], UI embeddings [38], and more. We will examine these use cases and develop a preliminary taxonomy later in the chapter.

In this chapter, we begin by describing the creation of Rico using a system that combined crowdsourcing and automation to scalably mine design and interaction data from Android apps at runtime. We then describe two projects that we conducted using the dataset: the training of an autoencoder to identify similarity between UI

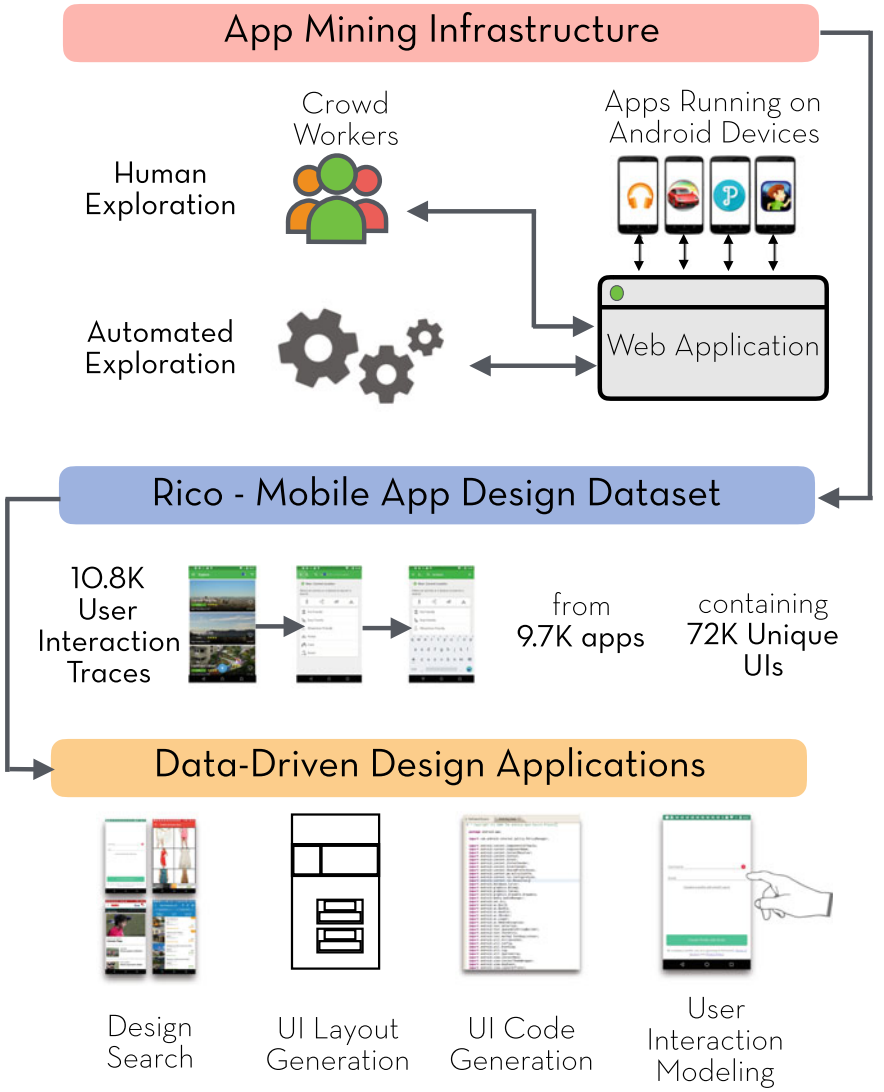

Fig. 2 Rico is a design dataset with 72 k UIs mined from 9.7 k free Android apps using a combination of human and automated exploration. The dataset can power a number of design applications, including ones that require training state-of-the-art machine learning models

designs, and an exploration of the use of Google's Material Design within the dataset using machine learned models. We conclude with an overview of other work that has used Rico to understand our mobile UI world and build data-driven models that assist users, designers, and developers.

2 Collecting Rico

To create Rico, we developed a platform that mines design data from Android apps at runtime by combining human-powered and programmatic exploration. Humans rely on prior knowledge and contextual information to effortlessly interact with a diverse array of apps. Apps, however, can have hundreds of UI states, and human exploration clusters around common use cases, achieving low coverage over UI states for many apps [10, 22]. Automated agents, on the other hand, can be used to exhaustively process the interactive elements on a UI screen [13, 50]; however, they can be stymied by UIs that require complex interaction sequences or human inputs (Fig. 3) [8].

We developed a hybrid approach for design mining mobile apps that combine the strengths of human-powered and programmatic exploration: leveraging humans to unlock app states that are hidden behind complex UIs and using automated agents to exhaustively process the interactive elements on the uncovered screens to discover new states. The automated agents leverage a novel *content-agnostic similarity heuristic* to efficiently explore the UI state space. Together, these approaches achieve higher coverage over an app's UI states than either technique alone.

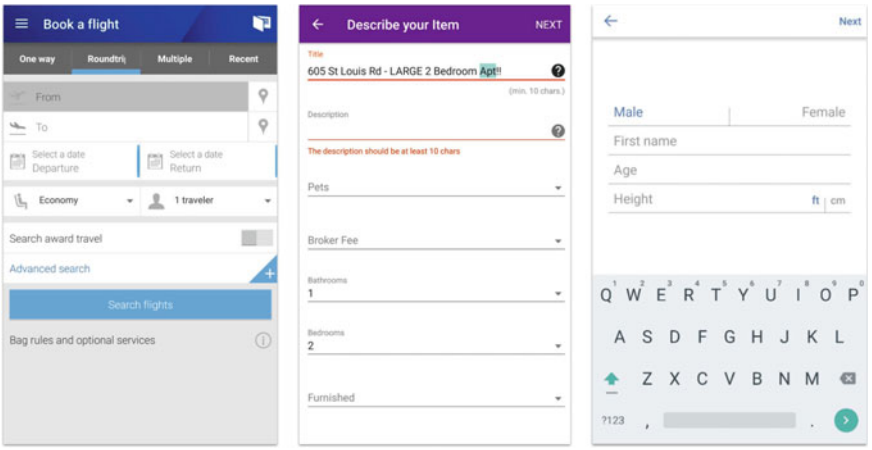

Fig. 3 Automated crawlers are often stymied by UIs that require complex interaction sequences, such as the three shown here

2.1 Crowdsourced Exploration

The crowdsourced mining system uses a web-based architecture similar to ERICA [22]. A crowd worker connects to the design mining platform through a web application, which establishes a dedicated connection between the worker and a phone in our mobile device farm. The system loads an app on the phone and starts continuously streaming images of the phone's screen to the worker's browser. As the worker interacts with the screen on his browser, these interactions are sent back to the phone, which performs the interactions on the app.

We extended the ERICA architecture to enable large-scale crowdsourcing over the Internet. We added an authorization system that supports both short- and long-term engagement models. For micro-task style crowdsourcing on platforms like Amazon Mechanical Turk, we generate URLs with tokens. When a worker clicks on a URL with a valid token, the system installs an app on a device and hands over control to the user for a limited time. To facilitate longer term engagements on platforms such as UpWork, we provide a separate interface through which workers can repeatedly request apps and use them. This interface is protected by a login wall, and each worker is provided separate login credentials.

We show the web interface in Fig. 4. To ensure that no personally identifiable information is captured, the web interface provides a name, email address, location, and phone number for crowd workers to use in the app. It also displays emails or text messages sent to the specified email addresses and phone numbers, letting crowd workers complete app verification steps with minimal effort.

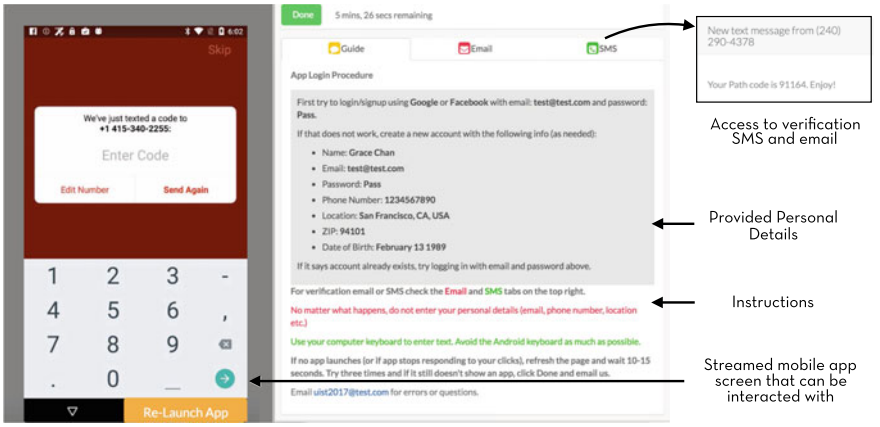

Fig. 4 Our crowd worker web interface. On the left, crowd workers can interact with the app screen using their keyboard and mouse. On the right, there are provided instructions and details such as the name, location, phone number, and email address to use in the app. The interface also allows workers to access SMS and email messages sent to the provided phone number and email to complete app verification processes

2.2 Automated Exploration

To move beyond the set of UI states uncovered by humans, Rico employs an automated mining system. Existing automated crawlers hard-code inputs for each app to unlock states hidden behind complex UIs [8, 33]. We achieve a similar result by leveraging the interaction data contained within the collected user traces: when the crawler encounters an interface requiring human input, it replays the interactions that a crowd worker performed on that screen to advance to the next UI state.

Similar to prior work [8, 33], the automated mining system uses a depth-first search strategy to crawl the state space of UIs in the app. For each unique UI, the crawler requests the view hierarchy to identify the set of interactive elements. The system programmatically interacts with these elements, creating an *interaction graph* that captures the unique UIs that have been visited as nodes and the connections between interactive elements and their resultant screens as edges. This data structure also maintains a queue of unexplored interactions for each visited UI state. The system programmatically crawls an app until it hits a specified time budget or has exhaustively explored all interactions contained within the discovered UI states.

2.3 Content-Agnostic Similarity Heuristic

After Rico's crawler interacts with a UI element, it must determine whether the interaction led to a new UI state or one that is already captured in the interaction graph. Database-backed applications can have thousands of views that represent the same semantic concept and differ only in their content (Fig. 5). Therefore, we employ a *content-agnostic similarity heuristic* to compare UIs.

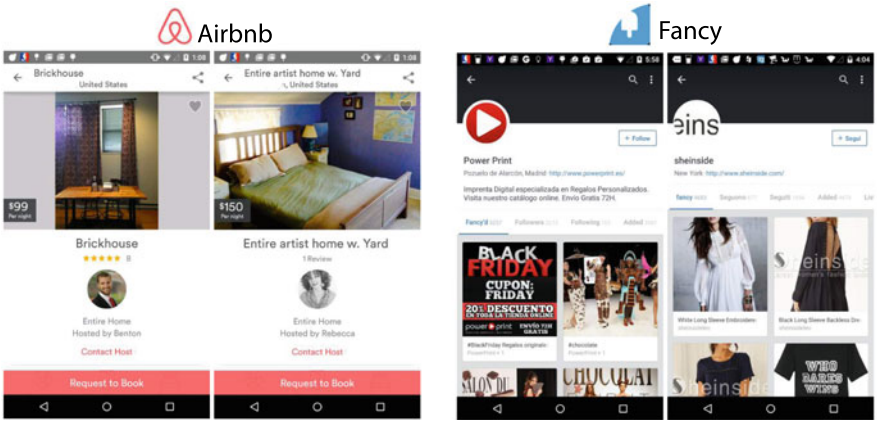

Fig. 5 Pairs of UI screens from apps that are visually distinct but have the same design. Our content-agnostic similarity heuristic uses structural properties to identify these sorts of design collisions

This similarity heuristic compares two UIs based on their visual and structural composition. If the screenshots of two given UIs differ by fewer than α pixels, they are treated as equivalent states. Otherwise, the crawler compares the set of element `resource-ids` present on each screen. If these sets differ by more than β elements, the two screens are treated as different states.

We evaluated the heuristic with different values of α and β on 1,044 pairs of UIs from 12 apps. We found that $\alpha = 99.8\%$ and $\beta = 1$ produce a false positive rate of 6% and a false negative rate of 3%. We use these parameter values for automated crawling and computing the set of unique UIs for a given app.

2.4 Coverage Benefits of Hybrid Exploration

To measure the coverage benefits of our hybrid exploration approach, we compare Rico's crawling strategy to human and automated exploration alone. We selected 10 apps (Fig. 6) from the top 200 on the Google Play Store. Each app had an average rating higher than 4 stars (out of 5) and had been downloaded more than a million times. We recruited 5 participants for each app and instructed them to use the app until they believed they had discovered all its features. We then ran the automated explorer on each app for three hours, after warming it up with the collected human traces.

Prior work [1, 10, 22] measured coverage using Android activities, a way of organizing an Android app's codebase that can comprise multiple UI screens. While activities are a useful way of statically analyzing an Android app, developers do

Name	Description
Polyvore	Fashion social-network and marketplace
Fabulous	Goal-setting app
Issuu	Magazine browsing and collection
Foursquare	City guide and reviews
Yelp	Guide for local businesses
Newsrepublic	World news digest
Etsy	Homemade and Vintage goods marketplace
Todoist	To-do list and reminder
WeHeartIt	Photo-sharing social network
Weather Channel	Weather tracker
Evernote	Note-taking app for collaboration

Fig. 6 The Android apps used in our evaluation. Each had a rating higher than 4 stars (out of 5) and more than 1 M downloads on the Google Play store

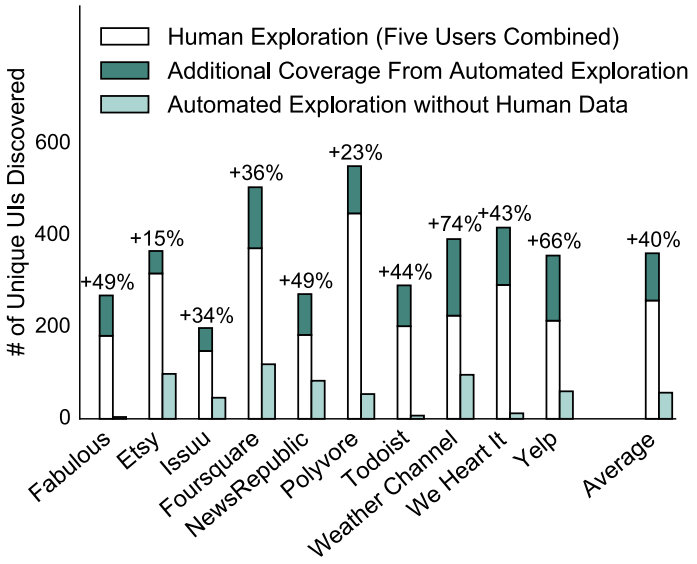

Fig. 7 The performance of our hybrid exploration system compared to human and automated exploration alone, measured across ten diverse Android apps

not use them consistently: in practice, complex apps can have the same number of activities as simple apps. In contrast, we use a coverage measure that correlates with app complexity: computing coverage as the number of unique UIs discovered under the similarity heuristic.

Figure 7 presents the coverage benefits of a hybrid system: combining human and automated exploration increases UI coverage by an average of 40% over human exploration alone and discovered several new Android activities for each app. For example, on the Etsy app, our hybrid system uncovered screens from 7 additional Activities beyond the 18 discovered by human exploration.

We also evaluated the coverage of the automated system in isolation, without bootstrapping it with a human trace. The automated system achieved 26% lower coverage across the tested apps than Rico's hybrid approach. This poor performance is largely attributable to experiences that are gated beyond a login screen or paywall that our pure, automated approach cannot handle. For instance, Todoist and WeHeartIt hide most of their features behind a login wall.

3 The Rico Dataset

The Rico dataset comprises 10,811 user interaction traces and 72,219 unique UIs from 9,772 Android apps spanning 27 categories (Fig. 8). We excluded from our crawl categories that primarily involve multimedia (such as video players and photo

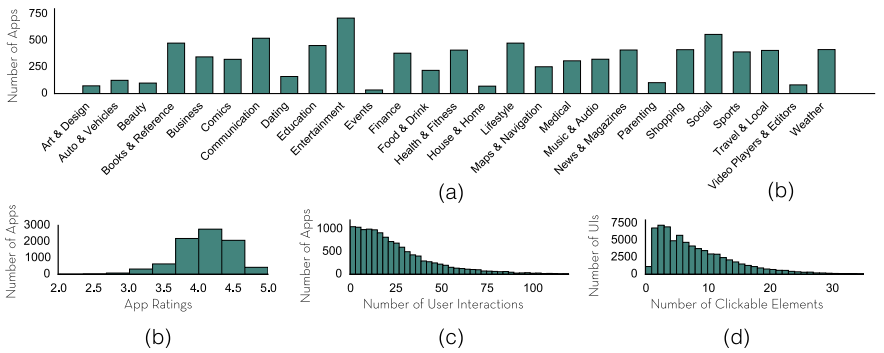

Fig. 8 Summary statistics of the Rico Dataset: app distribution by **a** category, **b** average rating, and **c** number of mined interactions. **d** The distribution of mined UIs by number of interactive elements

editors) as well as productivity and personalization apps. Apps in the Rico dataset have an average rating of 4.1 stars and data pertaining to 26 user interactions.

3.1 Data Collection

To create Rico, we downloaded 9,772 free apps from the Google Play Store and crowdsourced user traces for each app by recruiting 13 workers (10 from the US, 3 from the Philippines) on UpWork. We chose UpWork over other crowdsourcing platforms because it allows managers to directly communicate with workers: a capability that we used to resolve any technical issues that arose during crawling. We instructed workers to use each app as it was intended based on its Play Store description for no longer than 10 min.

In total, workers spent 2,450 h using apps on the platform over five months, producing 10,811 user interaction traces. We paid US $19, 200 in compensation or approximately two dollars to crowdsource usage data for each app. To ensure high-quality traces, we visually inspected a subset of each user's submissions. After collecting each user trace for an app, we ran the automated crawler on it for one hour.

3.2 Design Data Organization

For each app, Rico exposes Google Play Store metadata, a set of user interaction traces, and a list of all the unique discovered UIs through crowdsourced and automated exploration. The Play Store metadata includes an app's category, average rating, number of ratings, and number of downloads. Each user trace is composed of a sequence of UIs and user interactions that connect them. Each UI comprises a

screenshot, an augmented view hierarchy, a set of explored user interactions, a set of animations capturing transition effects in response to user interactions, and a learned vector representation of the UI's layout.

View hierarchies capture all of the elements comprising a UI, their properties, and relationships between them. For each element, Rico exposes its *visual* properties such as screen position, dimensionality, and visibility; *textual* properties such as class name, id, and displayed text; *structural* properties such as a list of its children in the hierarchy; and *interactive* properties such as the ways a user can interact with it. Additionally, we annotate elements with any Android superclasses that they are derived from (e.g., `TextView`), which can help third-party applications reason about element types. Rico contains more than 3 M elements, of which approximately 500 k are interactive. On average, each UI comprises eight interactive elements.

4 Our Uses of Rico

Having created the Rico dataset, we then made use of it in several different projects. Two of these projects are described here: the training of a UI layout embedding using an autoencoder, and an investigation of the usage of Material Design. We also built a system called Swire [27], which allowed users to investigate the Rico dataset by sketching a full or partial user interface and using that sketch as a search query over the dataset. Swire is described in more detail in Chapter 12 of this book.

4.1 Training a UI Layout Embedding

The large size of the Rico dataset makes it difficult to browse comprehensively, so in this work, we set out to create a method that would allow users to search the dataset. As a starting point, we allow users to use a screenshot of a mobile UI as their query.

Since the Rico dataset is large and comprehensive enough to support deep learning applications, we trained a deep autoencoder to learn an embedding for UI layouts and used it to annotate each UI with a 64-dimensional vector representation encoding visual layout. This vector representation can be used to compute structurally—and often semantically—similar UIs, supporting example-based search over the dataset (see figures in the original Rico paper [21]).

An autoencoder is a neural network that involves two models—an encoder and a decoder—to support the *unsupervised* learning of lower-dimensional representations [12]. The encoder maps its input to a lower-dimensional vector, while the decoder maps this lower-dimensional vector back to the input's dimensions. Both models are trained together with a loss function based on the differences between inputs and their reconstructions. Once an autoencoder is trained, the encoder portion is used to produce lower-dimensional representations of the input vectors.

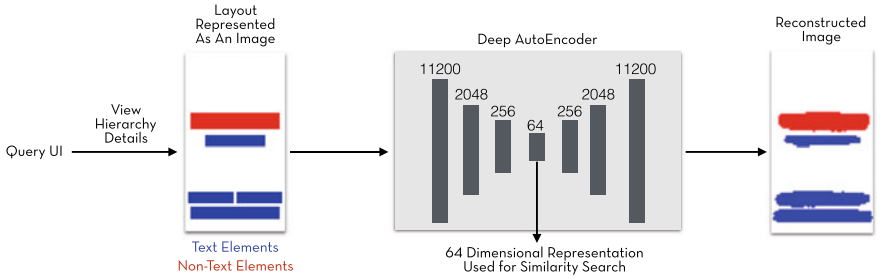

Fig. 9 We train an autoencoder to learn a 64-dimensional representation for each UI in the repository, encoding structural information about its layout. This is accomplished by creating training images that encode the positions and sizes of elements in each UI, differentiating between text and non-text elements

To create training inputs for the autoencoder that embed layout information, we constructed a new image for each UI encoding the bounding box regions of all leaf elements in its view hierarchy, differentiating between text and non-text elements (Fig. 9). Rico's view hierarchies obviate the need for noisy image processing or OCR techniques to create these inputs. In the future, we could incorporate more recent work on predicting functional semantic labels [37] for elements such as *search icon* or *login button* to train embeddings with even richer semantics.

The encoder has an input dimension of 11,200 and an output dimension of 64 and uses two hidden layers of dimension 2,048 and 256 with ReLU non-linearities [41]. The decoder has the reverse architecture. We trained the autoencoder with 90% of our data and used the rest as a validation set, and we found that the validation loss stabilized after 900 epochs or approximately 5 h on an Nvidia GTX 1060 GPU. Once the autoencoder was trained, we used the encoder to compute a 64-dimensional representation for each UI, which we expose as part of the Rico dataset.

Figure 10 shows several example query UIs and their nearest neighbors in the learned 64-dimensional space. The results demonstrate that the learned model is able to capture common mobile and Android UI patterns such as lists, login screens, dialog screens, and image grids. Moreover, the diversity of the dataset allows the model to distinguish between layout nuances, like lists composed of smaller and larger image thumbnails.

4.2 Understanding Material Design Usage in the Wild

Material Design[2] is a UI design pattern language introduced by Google in 2014, which can be applied to user interfaces on many types of computing devices. In this

[2] Material Design https://material.io/guidelines/.

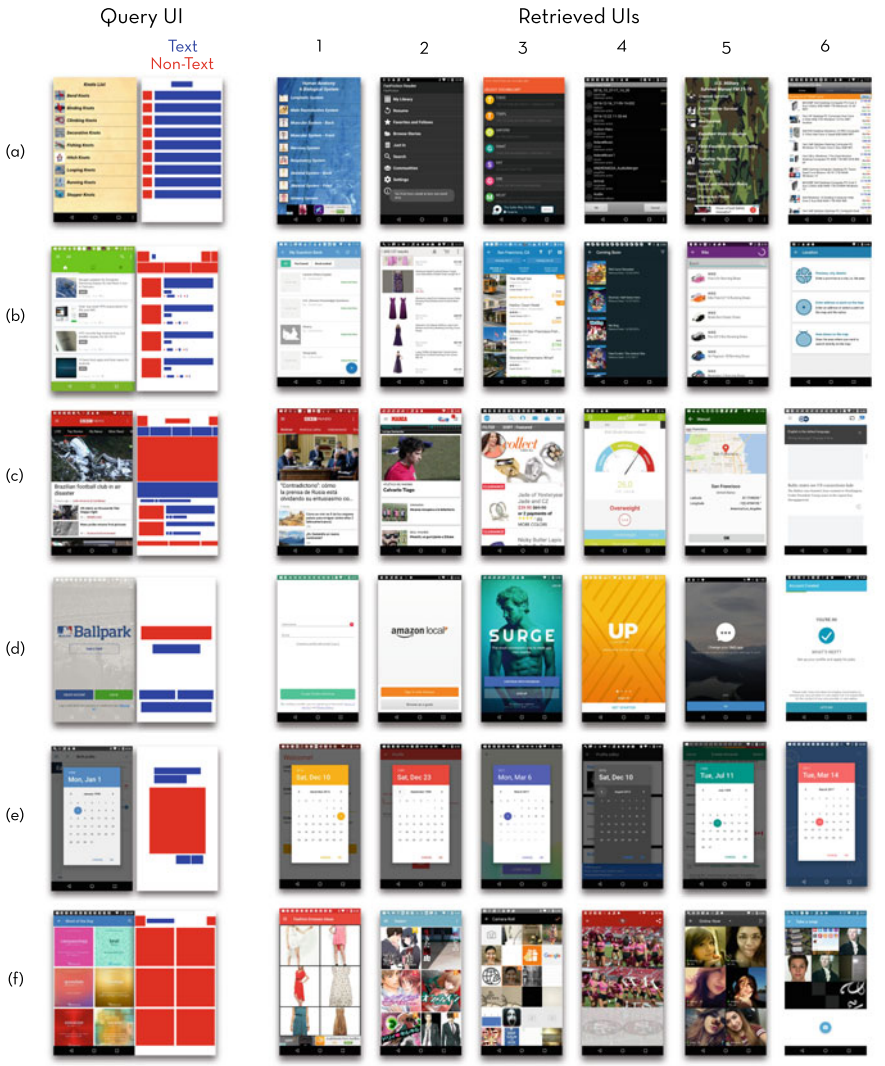

Fig. 10 The top six results obtained from querying the repository for UIs with similar layouts to those shown on the left, via a nearest-neighbor search in the learned 64-dimensional autoencoder space. The returned results share a common layout and even distinguish between layout nuances such as lists composed of smaller and larger image thumbnails (**a**, **b**)

work, we leverage the Rico dataset to understand how Material Design has been used on mobile devices.

Pattern languages have been long used in Human–Computer Interaction (HCI) for distilling and communicating design knowledge [14, 20]. According to Christopher Alexander [6], who introduced pattern-based design to architecture, "each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

HCI researchers and design practitioners have documented and introduced pattern languages for general UI design (e.g., [42, 51]) as well as a wide variety of application domains, such as learning management systems [9], ubiquitous computing [30], information retrieval [53], and many more. Nonetheless, as Dearden and Finlay point out, there have been relatively few evaluations of how useful pattern languages are in user interface design [20]. Since Dearden and Finlay published their critical review, more evaluations have been done on pattern languages in HCI (e.g., [18, 46, 53]). But these evaluations are usually limited in at least one of several ways. First, the pattern languages in those evaluations were often developed in an academic research setting. Few have been applied to real-world applications. Second, the evaluations were usually done in lab settings and hence lacked ecological validity. Last, those evaluations were done at a very small scale (i.e., applying a pattern language to either one or no more than a handful of systems). As a result of the limitations of how the field has evaluated pattern languages, we know little about whether pattern languages in HCI are fulfilling the promise of Alexander—providing design solutions that can be reused "a million times over".

The recent success of commercial UI design pattern languages offers a rare opportunity for us to evaluate the usefulness of pattern languages in HCI at scale and in the wild. In particular, Material Design seems to have been widely adopted by developers who build applications for Google's Android operating system. *How can we understand the impact of a pattern language in one of the largest computing ecosystems in the world?* This is the first research question we seek to answer in this project.

In addition to developing a method for measuring a pattern language's overall impact, we also want to address questions about how and where certain patterns should be used when they get applied to new use cases. For Material Design, few patterns have been more controversial, yet at the same time iconic, than the Floating Action Button (aka, FAB) and the Navigation Drawer (i.e., the hamburger menu). Tens of thousands of words have been written about the merits and more often the downsides of these two patterns (e.g., [11, 28, 48] for FAB and [4, 19, 43] for Hamburger Menu). Sometimes, the conclusions are daunting. For example, one online critic said, "...in actual practice, widespread adoption of FABs might be detrimental to the overall UX of the app" [48]. Even when the criticisms are moderate and well-reasoned, they are based on the writer's examination of a limited number of examples. It is hard to know whether these criticisms reflect the full picture, since these patterns are likely to be used in a huge number of different apps. Thus, the second research question driving this work is: *How can we examine real-world use of design patterns to inform debates about UI design?*

We took a data-driven approach to shed light on these two questions, leveraging the app screenshots and view hierarchies in the Rico dataset and the app ratings and downloads data from Google Play. Using text mining and computer vision techniques, we built a computational model to detect six widely used UI components specified in Material Design, including the FAB, the Navigation Drawer, and four other components. We then used the metadata of the apps in the first dataset to measure the relationship between the use of certain patterns and the popularity of apps, as indicated by app ratings and the number of installs. Furthermore, we used app category data to examine in what domain a certain pattern might be more useful.

Our results show that the use of Material Design is positively correlated with both the app's average rating and the number of installs, which we believe is the first quantitative evidence for the value of a pattern language applied to a large ecosystem in the wild. Our data analysis further shows that, despite the criticisms the FAB and the Navigation Drawer have received from vocal writers in the design community, they are more popular among apps with higher ratings and higher number of installs than their less popular peers. Furthermore, we found that the use of UI components varies by app category, suggesting a more nuanced view needed in ongoing debates about UI design patterns.

4.2.1 Detecting Material Design Elements

There were two general stages in our data analysis. First, we detected Material Design elements in a large number of mobile apps. We focused on six elements in Material Design: App Bar, Floating Action Button, Bottom Navigation, Navigation Drawer, Snack Bar, and Tab Layout. Second, we looked for relationships between usage of Material Design elements and app popularity as well as app category.

The main challenge in our data analysis was to reliably detect Material Design elements, which lacked standardized names in the app view hierarchies. Therefore, to detect as many Material Design elements as possible from an app (either implemented with official or unofficial Material Design libraries), we leveraged the pixel data from the apps' screenshots in the Rico dataset.

Specifically, we used computer vision techniques such as deep Convolutional Neural Networks (CNNs) to detect Material Design elements from screenshots. To train these models, we needed positive and negative cropped snapshots of Material Design elements as training data. To collect the ground truth data, we turned to those apps that implemented their UIs using the official Material Design library. These elements were easy to find by class name in the view hierarchy `JSON` files. In order to train a good classifier, negative examples of each element also need to be collected from a relevant location and should not be cropped from a random part of the screen. To this end, we created a heatmap for each type of element based on the apps using the official library. With these heatmaps, we cropped the UI regions which did not use Material Design elements. Therefore, for the screenshots which did not include a Material Design element, we cropped the screenshot based on the most probable part

Fig. 11 The heatmap of the frequency divided by maximum value of each Material Design element in *Rico* dataset

of the screen (heatmap) for that element to generate negative examples. Figure 11 shows the heatmaps of each Material Design element in *Rico* dataset.

After collecting a set of images for each Material Design element, we used deep Convolutional Neural Networks to detect Material Design elements in the apps. We trained a separate classifier for each Material Design element to detect that element in an app's screenshot. We selected the AlexNet [29] architecture for our Convolutional Neural Networks. We trained all the networks from scratch with a learning rate of 0.001 and 50,000 iterations. We split our data into 80% training, 10% validation, and 10% for testing the trained network and got at least 95% of accuracy on each component. We used Google's open-source machine learning platform TensorFlow [5] to implement all our machine learning models. For detailed information about our methodology, please refer to [23].

4.2.2 Results of Data Analysis

Our data analysis led to a number of interesting findings about Material Design's usage in the wild. We first report the usage of specific elements such as the Floating Action Button and the Navigation Drawer, two popular but somewhat controversial patterns in Material Design, and how the usage of these elements relate to app ratings, installs, and categories. We then report the usage of Material Design in general and examine its impact on app popularity.

4.2.3 Usage of Floating Action Buttons

If the drawbacks of FABs generally outweigh their benefits, as some design critics argued, one would assume that higher-rated apps would be less likely to use FABs than those lower-rated apps. To test this hypothesis, we split apps into two groups: a high-rating group and a low-rating group by the median average rating of all apps in the Play Store dataset, which was 4.16 at the time of this analysis. The two groups of apps were balanced and each group had 4673 number of apps.

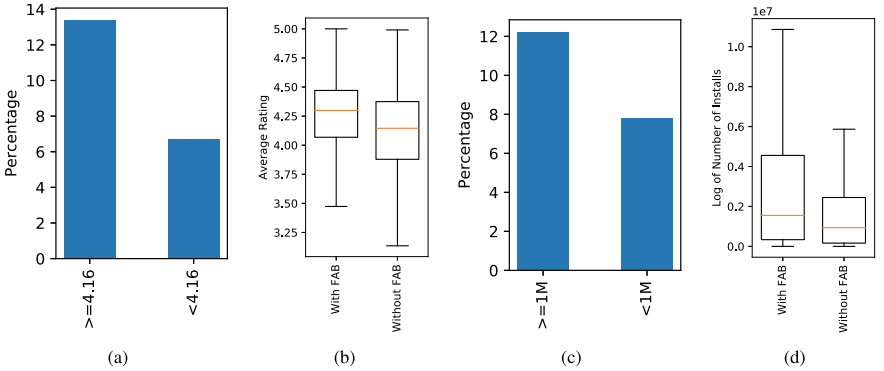

Fig. 12 **a** The percentage of apps using the Floating Action Button (FAB) in the high-rating group versus the low-rating group. **b** Box plots of the average ratings of apps using the FAB versus those not using the FAB. **c** The percentage of apps using the FAB in the more-installed group versus the less-installed group. **d** Box plots of the number of installs of apps using the FAB versus those not using the FAB

As Fig. 12a shows, there was actually a higher percentage of apps using FABs in the high-rating group than those in the low-rating group (13.4% vs. 6.6%). The box plots in Fig. 12b further shows that apps using the FAB were rated higher than those that did not use it. In fact, 66.6% of apps that used the FAB belonged to the higher-rating group.

We also used the number of installs as another measure of app popularity. Thus, we decided to split our apps into two groups: (1) apps with greater than or equal to 1 million installs, and (2) apps with less than 1 million installs. The two groups were nearly balanced after the split, with 4723 in the more-installed group and 4623 in the less-installed group. Similar to what we saw in the analysis of FAB usage and app ratings, apps in the more-installed group appeared to be more likely to feature FABs than those in the less-installed group (see Fig. 12c). Also, apps using the FAB had a larger number of installs in comparison to apps without the FAB (see Fig. 12d).

The results above suggest that many developers of popular apps consider the FAB to be a valuable design pattern. Nonetheless, it's still possible that the FAB is a more useful pattern in some situations than others.

To understand where the FAB might be more useful, we examined the usage of the FAB by app category. Figure 13 shows the top 11 app categories by the percentage of apps featuring the FAB, excluding categories for which there were too few apps in the *Rico* dataset (less than 0.05% of the apps of that category in Google Play). As it is obvious to see, FAB usage varied considerably among these 11 categories of apps. The *Food and Drink* category had the highest percentage of FAB usage among all the qualified categories. Figure 14 shows some of the FABs in the *Food and Drink* category. Each thumbnail belongs to a different app but there are FABs with similar icons in this category, suggesting common usage of the FAB such as suggesting recipes (the "folk" FAB), locating nearby restaurants (the "location" FAB). Note

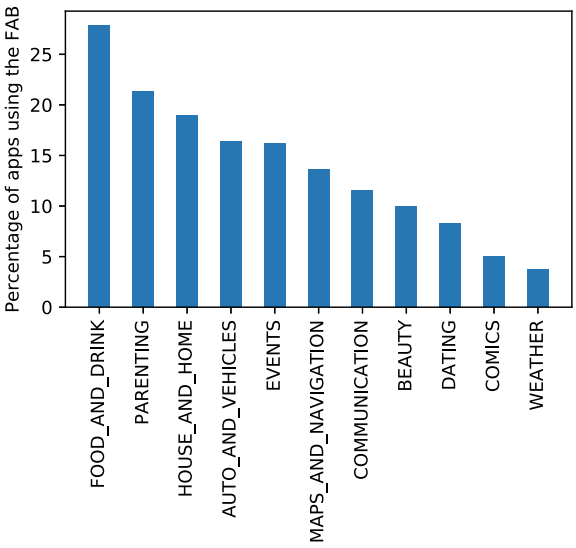

Fig. 13 The top category of apps which used the most FAB by percentage in their category

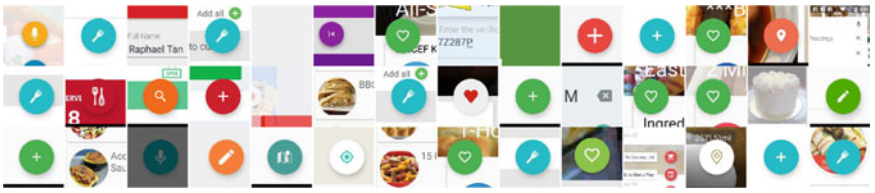

Fig. 14 Thumbnails of FABs in the Food and Drink category apps

that some of the thumbnails in this picture do not appear to include a FAB, because they are occluded by another UI component.

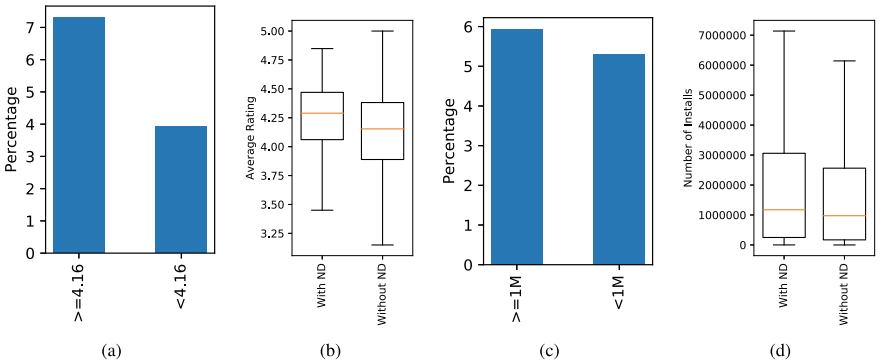

Fig. 15 **a** The percentage of apps using the Navigation Drawer in the high-rating group versus the low-rating group. **b** Box plots of the average ratings of apps using the Navigation Drawer versus those not using the Navigation Drawer. **c** The percentage of apps using the Navigation Drawer in the more-installed group versus the less-installed groups. **d** Box plots of the number of installs of apps using the Navigation Drawer versus those not using the Navigation Drawer

4.2.4 Usage of Navigation Drawers

We applied the same analysis to examine the usage of the Navigation Drawer in Material Design. As we can see in Fig. 15a, there were more apps in the high-rating group which had a Navigation Drawer than those in the low-rating group (7.3% vs. 3.9%). The box plots in Fig. 15b show that the average rating for apps using the Navigation Drawer was higher than those that did not use it. Among all the apps that used the Navigation Drawer, 65% of them belonged to the high-rating group.

Similar to our analysis of the FAB usage, we examined the usage of the Navigation Drawer and the number of app installs. As it is shown in Fig. 15b, apps in the high-rating group were slightly more likely to feature a Navigation Drawer than those in the low-rating group. Also, the box plots in Fig. 15d show that apps using Navigation Drawer had a slightly higher number of installs.

4.2.5 Material Design and App Popularity

We conducted an analysis to understand the usage of Material Design in general and its relationship to apps' average ratings and the number of installs. The first step was to determine if an app used Material Design. We adopted a relatively relaxed criterion: if an app used one of the six Material Design components our model could detect, we considered Material Design was used in that app.

First, we examined the relationship between the usage of Material Design and apps' average ratings. To this end, we sorted all the apps in the *Rico* dataset by their average ratings, split them into one hundred buckets, and calculated the percentage of apps that used Material Design for each percentile. We then plotted the percentage of

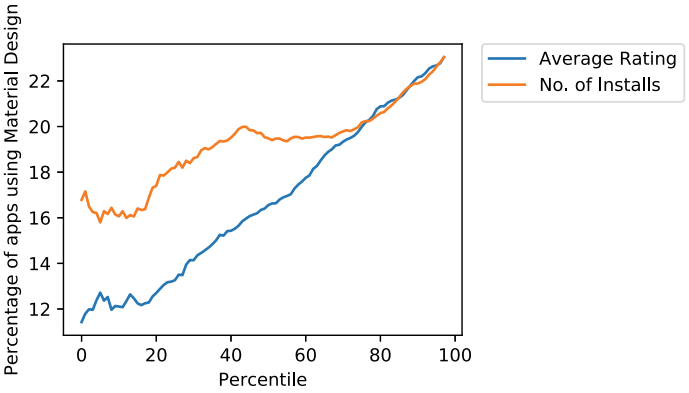

Fig. 16 Distribution of the percentage of apps using at least one of the six common Material Design elements over percentiles in average rating (blue) and number of installs (orange)

Material Design usage over average rating percentiles. As we can see in Fig. 16, the usage of Material Design was highly correlated with the average rating percentile (with the Pearson correlation coefficient $\rho = 0.99$ and p-value = 3.1×10^{-91}). In other words, as the average rating increased, the percentage of apps using Material Design also increased.

Next, we examined the relationship of the usage of Material Design and the number of installs, an alternate measure of app popularity. As in the previous step, we sorted apps by their number of installs and split them into one hundred equal-sized buckets by percentile. As shown in Fig. 16, the percent of the apps using Material Design is also highly correlated to number of installs $\rho = 0.94$ and p-value = 2.3×10^{-47}).

4.2.6 Summary

To sum up, we review the two research questions we set out to answer. Our first research question was *How can we understand the impact of a pattern language in one of the largest computing ecosystems in the world?* We developed a computational method to measure the relationship between Material Design, the pattern language in question, and app popularity in the Android ecosystem. We used Convolutional Neural Networks as a data mining tool to analyze big UI data. We trained multiple models to detect Material Design elements in apps' screenshots.

Our second research question was *How can we examine real-world use of design patterns to inform debates about UI design?* To answer this question, we examined the usage of the Floating Action Button and the Navigation Drawer, two frequently criticized patterns in online design discussions. While our results do not directly rebut specific arguments against these two patterns, they clearly show that many developers and designers found these two patterns valuable and use both patterns frequently in their highly rated and highly popular apps. Moreover, our results have

suggested that evaluating the merit of a design pattern should consider the context it is applied to. For example, developers used the FAB more frequently in certain app categories such as *Food and Drink* and *Parenting* than others.

5 Rico in the World

Over the last three years since its release, Rico has been used by many research teams as the basis for their own projects. In this section, we attempt to categorize the uses that Rico has seen to date and highlight a few projects of interest.

We found in our exploration of these use cases that they could be broadly categorized into five areas: (a) mobile ecosystem explorations, (b) UI automation, (c) design assistance, (d) understanding UI semantics, and (e) automated design. In addition, efforts in many of these areas have additionally enhanced the Rico dataset itself, which we will discuss separately. We describe the research in these areas below.

5.1 Mobile Ecosystem Explorations

The research summarized in this section attempts to understand the Android app ecosystem by using the Rico dataset as a sample. Our own work described in Sect. 4.2 falls under this category, where we used Rico to explore the usage of the Material Design pattern language in the Android app ecosystem.

Micallef et al. used Rico to study the use of login functionality in Android apps [39]. They found that 32% of apps use at least one kind of login functionality and 9% provided at least one social login (e.g., Facebook, Google). They found no correlation between the usage of login features and the number of app downloads or app ratings.

Ross et al. investigated the state of mobile app accessibility using a subset of the Rico dataset [44]. They specifically focused on the accessibility of image-based buttons and studied the prevalence of accessibility issues due to missing labels, duplicate labels, or uninformative labels. They discovered a bi-modal distribution of missing labels in apps with 46% of apps having less than 10% (46%) of their image-based buttons labeled and 36% of apps having more than 90% labeled. The correlation between accessibility, as measured by missing labels, and app ratings was found to be weak.

5.2 UI Automation

Several works have used Rico to develop mobile UI automation tools and techniques. One popular use of UI automation is for helping end users. Li et al. look at task automation in which natural language instructions must be interpreted as a sequence

of actions on a mobile touch-screen UI [34]. Toward that end, they used a subset of the Rico dataset, enhanced it with action commands, and created the RicoSCA dataset. This dataset, along with two other datasets, allowed them to develop models to map multi-step instructions into automatically executable actions given the screen information. Sereskeh et al. developed programming by demonstration system for smartphone task automation called VASTA[47]. They used the Rico dataset to train a detector for UI elements that powered the vision capabilities of their system.

Another popular use of UI automation is for testing. Li et al. developed Humanoid, a test input generator for Android apps [36]. They use the interactions traces from the Rico dataset to train a deep learning-based model of how humans interact with app screens. This model is then used to guide Humanoid's test input generation. Zhang et al. used Rico to train a deep learning-based model for identifying isomorphic GUIs, which are GUI's that may have different contents but represent the same screen semantically [55]. Although they intend to use such identification to enable robotic testing for mobile GUIs, this feature could also be useful for crawling mobile apps. Section 2.3 describes how we handled the identification of isomorphic GUIs during data collection for the Rico dataset.

5.3 Design Assistance

Another popular use of the Rico dataset is to develop data-driven tools to assist with mobile UI design. An example of such a tool is a search engine for finding example UIs of interest. Such search engines would enable designers to use relevant examples early on in the design process for inspiration and to guide their design process. Section 4.1 describes how we used Rico to train an autoencoder to learn an embedding for UI layouts and used it to demonstrate an example-based search for UIs which gives the user the ability to search for UIs similar to a UI of interest. Chen et al. collected an Android UI dataset similar to Rico and used it to train a CNN to enable searching UIs based on wireframes [15]. Huang et al. developed Swire, a UI retrieval system that can be queried by using hand-drawn sketches of UIs [27]. This was enabled by training a deep neural network to learn a sketch-screenshot embedding space for UIs in the Rico dataset and performing a nearest-neighbor search in that space. Swire is also described in Chapter XX in this book.

Another set of data-driven tools attempt to provide feedback and guide designers, especially novice designers, during the design process. Lee et al. developed GUIComp, a tool that provides real-time feedback to designers including showing other relevant examples, predicting user attention characteristics of the design, and showing design complexity metrics [31]. They used a subset of the Rico dataset as a basis for their tool and trained an autoencoder to find similar UIs following an approach similar to that described in Sect. 4.1. Wu et al. developed a tool to predict user engagement based on the types of animations used within the app [54]. Their approach was enabled by training a deep learning model on the animations released as part of the Rico dataset. Finally, Swearngin et al. built upon the Rico dataset to

create a new dataset for mobile interface tappability using crowdsourcing and then computationally investigated a variety of signals that are used by typical users to distinguish tappable versus not-tappable elements [49].

5.4 Understanding UI Semantics

Several recent works have also used the Rico dataset to develop approaches for developing a taxonomy of UI elements and then building detectors for different UI element types found in mobile UIs. Liu et al. identified 25 semantic concepts that are commonly implemented by UI elements, such as next, create, and delete. They then trained a Convolutional Neural Network (CNN) to detect the corresponding UI elements in app screens and used it to annotate the elements in Rico dataset. These semantic annotations are now available for download as part of the Rico dataset.

Moran et al. mined a dataset of app screens similar to Rico and used the resulting dataset to develop an automated approach for converting GUI mockups to implemented code [40]. To do that, they too developed techniques to detect and classify different UI elements found in UIs. Chen et al. used the Rico dataset to perform a large-scale empirical study of seven representative GUI element detection methods on over 50 k GUI images [16].

Finally, Li et al. collected natural language descriptions, called *captions*, for elements in the Rico dataset and used it to train models that generate captions for UI elements (useful for accessibility and language-based interactions) [35]. In this work, they also augmented Rico with 12 K newly crawled UI screens.

5.5 Automated Design

Another area where a UI dataset is essential is for the development of methods for the automated generation of UIs. Lee et al. developed the Neural Design Network, an approach to generate a graphic design layout given a set of components with user-specified attributes and constraints [32]. Gupta et al. developed the Layout-Transformer, a technique that leverages a self-attention-based approach to learn contextual relationships between layout elements and generate new layouts [26]. Both these works use the Rico dataset to test their approaches for mobile UIs.

5.6 Enhancements to the Rico Approach and Dataset

Several of the research projects discussed above have also enhanced the Rico dataset with new annotations or additional screens. Liu et al. added semantic annotations for UI elements (e.g, delete, save, search, etc.) to the Rico dataset [37]. This was

accomplished by (a) iterative open coding of 73 k UI elements and 720 screens, (b) training a convolutional neural network that distinguishes between icon classes, and (c) using that network to compute semantic annotations for the 72 k unique UIs in the Rico dataset, assigning labels for 78% of the total visible, non-redundant elements.

Leiva et al. released Enrico, a curated dataset of 1460 mobile UIs drawn from Rico that are tagged with one of 20 different semantic topics (e.g., gallery, chat, etc.) [38]. This was accomplished by using human annotators to (a) systematically identify *popular* UI screens that have consistent screenshots and view hierarchies in 10k randomly drawn screens from the Rico dataset, (b) create a taxonomy with 19 topics that accurately represented these popular UI screens, and (c) assign each UI to a topic.

For crowd crawling of apps, new tools are available to offer more support to crowd workers. For example, Chen et al. developed tools that offer guidance to crowd explorers that can reduce redundant exploration and increase coverage [17].

6 Discussion

To our knowledge, the Rico dataset remains the largest repository of Android mobile app designs, and it has been used by many research teams worldwide to facilitate our understanding of the Android mobile app ecosystem and to create tools and technologies that advance our use of mobile user interfaces. We are happy that we chose to release the dataset publicly and are impressed with the follow-on work that has been done as a result. We hope that others who augment Rico or create their own new datasets will likewise make them publicly available, as this helps the entire research community. One challenge for the future is how to aggregate these new additions and new datasets into a single accessible place, as today those that have been released are shared in a variety of locations with little standardization.

While Rico continues to be useful, it has weaknesses that we hope to address. The initial idea to create the dataset was born out of a need to train machine learned models that incorporated an understanding of the user interface. At the time, we were interested in creating generative models that could produce full or partial user interfaces designs. While we ended up not pursuing this direction, this initial use case is reflected in the type of data collected in the Rico dataset. Our goal was to collect a nearly complete picture of every app that we explored, including each of its screens, dialog boxes, etc. In our collection process, we intentionally did not try to capture data about how humans used these interfaces, and we disregarded the tasks for which the user interfaces might be used, common user behaviors with the interfaces, and other semantic information and metadata related to the user interfaces. Another omission in Rico is that it contains no task-related information nor any ecologically valid traces of human interaction on its UIs. Collecting such data will require new crowdsourcing techniques, especially at the scale needed for the data to be useful for deep learning, but would open up the possibility of many new applications that are not possible with the current dataset.

Another weakness in Rico is its lack of temporal data; Rico contains a snapshot of Android UIs collected at just one period of time in early 2017. Presumably many of the apps in the dataset have changed since they were originally collected, and certainly, new apps have been created that are not in the dataset. Although we see little evidence of it so far, models trained from the dataset could suffer from concept drift if present or future UIs change sufficiently from what was recorded in the dataset. To this end, we hope to capture an updated version of the dataset and release that publicly at some point in the future.

Finally, Rico contains just data from Android phone UIs in US English. Collecting data from other device types (e.g., tablets), other operating systems (e.g., iOS), and other internationalization and localization settings beyond US English would also open up new applications for the dataset. For example, being able to train design mappings between UIs that serve the same function but use different languages might create an opportunity to build automated or semi-automated tools for internationalizing UIs. Training mappings between phone and tablet interfaces could enable the creation of tools and techniques for improved responsive design.

7 Conclusion

In this chapter, we have:

- Presented the Rico dataset, the largest publicly released repository of Android mobile app designs, containing data for 72,219 user interfaces from 9,772 apps spanning 27 Google Play store categories.
- Described the hybrid human plus automation crawling process that was used to collect the dataset, which increased UI coverage by an average of 40% over human crawling alone.
- Shown a method of training a UI layout embedding with a deep autoencoder neural network architecture, which we demonstrated to be effective for searching for related screens within the Rico dataset.
- Explored the usage of the Material Design pattern language within the Rico dataset through machine learned techniques.
- Summarized the large body of work that others in the research community have undertaken using the Rico dataset.

While we are humbled by the many systems that have been built to date using Rico, applications of machine learning to the use and design of user interfaces are still very early. Further work in this area will continue to be enabled by the creation of datasets like Rico. We look forward to see what new datasets and applications that researchers create to make our user interfaces truly intelligent.

Acknowledgements We thank the reviewers for their helpful comments and suggestions and the crowd workers who helped build the Rico dataset. This work was supported in part by a Google Faculty Research Award.

References

1. Android Activities (2016). https://developer.android.com/guide/components/activities.html
2. Database of Android Apps on Kaggle (2016). https://www.kaggle.com/orgesleka/android-apps
3. UI Overview (2016). https://developer.android.com/guide/topics/ui/overview.html
4. A L (2014) Why and how to avoid hamburger menus
5. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org
6. Alexander C (1977) A pattern language: towns, buildings, construction. Oxford University Press, Oxford
7. Alharbi K, Yeh T (2015) Collect, decompile, extract, stats, and diff: mining design pattern changes in Android apps. In: Proceedings of MobileHCI
8. Amini S (2014) Analyzing mobile app privacy using computation and crowdsourcing. PhD thesis, Carnegie Mellon University
9. Avgeriou P, Papasalouros A, Retalis S, Skordalakis M (2003) Towards a pattern language for learning management systems. Educ Technol & Soc 6(2):11–24
10. Azim T, Neamtiu I (2013) Targeted and depth-first exploration for systematic testing of android apps. In: ACM SIGPLAN Notices
11. Babich N (2017) Floating action button in ux design
12. Bengio Y (2009) Learning deep architectures for ai. Found Trends Mach Learn 2:1
13. Bhoraskar R, Han S, Jeon J, Azim T, Chen S, Jung J, Nath S, Wang R, Wetherall D (2014) Brahmastra: driving apps to test the security of third-party components. In: Proceeding of the SEC
14. Borchers JO (2000) A pattern approach to interaction design. In: Proceedings of the 3rd conference on designing interactive systems: processes, practices, methods, and techniques, DIS '00, ACM, New York, NY, USA, pp 369–378
15. Chen J, Chen C, Xing Z, Xia X, Zhu L, Grundy J, Wang J (2020) Wireframe-based ui design search through image autoencoder. ACM Trans Softw Eng Methodol 29:3
16. Chen J, Xie M, Xing Z, Chen C, Xu X, Zhu L, Li G (2020) Object detection for graphical user interface: old fashioned or deep learning or a combination? New York, NY, USA, Association for Computing Machinery, pp 1202–1214
17. Chen Y, Pandey M, Song JY, Lasecki WS, Oney S (2020) Improving crowd-supported gui testing with structural guidance. In: Proceedings of the 2020 CHI conference on human factors in computing systems, CHI '20, Association for Computing Machinery, New York, NY, USA, pp 1–13
18. Chung ES, Hong JI, Lin J, Prabaker MK, Landay JA, Liu AL (2004) Development and evaluation of emerging design patterns for ubiquitous computing. In: Proceedings of the 5th conference on designing interactive systems: processes, practices, methods, and techniques, ACM, pp 233–242
19. Constine J (2014) Kill the hamburger button
20. Dearden A, Finlay J (2006) Pattern languages in hci: a critical review. Human-Comput Inter 21(1):49–102
21. Deka B, Huang Z, Franzen C, Hibschman J, Afergan D, Li Y, Nichols J, Kumar R (2017) Rico: a mobile app dataset for building data-driven design applications. In: 30th annual symposium on user interface software and technology, UIST '17, ACM, New York, NY, USA
22. Deka B, Huang Z, Kumar R (2016) ERICA: Interaction mining mobile apps. In: Proceedings of the UIST
23. Doosti B, Dong T, Deka B, Nichols J (2018) A computational method for evaluating UI patterns. ArXiv e-prints

24. Frank M, Dong B, Felt AP, Song D (2012) Mining permission request patterns from android and facebook applications. In: Proceeding of the ICDM
25. Fu B, Lin J, Li L, Faloutsos C, Hong J, Sadeh N (2013) Why people hate your app: making sense of user feedback in a mobile app store. In: Proceedings of the KDD
26. Gupta K, Achille A, Lazarow J, Davis L, Mahadevan V, Shrivastava A (2020) Layout generation and completion with self-attention
27. Huang F, Canny JF, Nichols J (2019) Swire: sketch-based user interface retrieval. In: Proceedings of the 2019 CHI conference on human factors in computing systems, CHI '19, Association for Computing Machinery, New York, NY, USA, pp 1–10
28. Jager T (2017) Is the floating action button bad ux design?
29. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds) Advances in neural information processing systems 25. Curran Associates Inc, Stateline, NV, USA, pp 1097–1105
30. Landay JA, Borriello G (2003) Design patterns for ubiquitous computing. Computer 36(8):93–95
31. Lee C, Kim S, Han D, Yang H, Park Y-W, Kwon BC, Ko S (2020) Guicomp: a gui design assistant with real-time, multi-faceted feedback. In: Proceedings of the 2020 CHI conference on human factors in computing systems, CHI '20, Association for Computing Machinery, New York, NY, USA, pp 1–13
32. Lee H-Y, Yang W, Jiang L, Le M, Essa I, Gong H, Yang M-H (2020) Neural design network: graphic layout generation with constraints. In: Proceedings of European conference on computer vision (ECCV)
33. Lee K, Flinn J, Giuli T, Noble B, Peplin C (2013) Amc: verifying user interface properties for vehicular applications. In: Proceeding of the Mobisys
34. Li Y, He J, Zhou X, Zhang Y, Baldridge J (2020) Mapping natural language instructions to mobile UI action sequences. In: Proceedings of the 58th annual meeting of the association for computational linguistics, Association for Computational Linguistics, pp 8198–8210
35. Li Y, Li G, He L, Zheng J, Li H, Guan Z (2020) Widget captioning: generating natural language description for mobile user interface elements. In: Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP), Association for Computational Linguistics, pp 5495–5510
36. Li Y, Yang Z, Guo Y, Chen X (2019) Humanoid: a deep learning-based approach to automated black-box android app testing. In: 2019 34th IEEE/ACM international conference on automated software engineering (ASE), pp 1070–1073
37. Liu TF, Craft M, Situ J, Yumer E, Mech R, Kumar R (2018) Learning design semantics for mobile apps. In: Proceedings of the 31st annual ACM symposium on user interface software and technology, UIST '18, Association for Computing Machinery, New York, NY, USA, pp 569–579
38. Luis AL, Asutosh Hota AO (2020) Enrico: a high-quality dataset for topic modeling of mobile ui designs. In: Proceedings of MobileHCI extended abstracts
39. Micallef N, Adi E, Misra G (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, UbiComp '18, Association for Computing Machinery, New York, NY, USA, pp 842–851
40. Moran K, Bernal-Cárdenas C, Curcio M, Bonett R, Poshyvanyk D (2020) Machine learning-based prototyping of graphical user interfaces for mobile apps. IEEE Trans Softw Eng 46(2):196–221
41. Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceeding of the ICML, pp 807–814
42. Neil T (2014) Mobile design pattern gallery: UI patterns for smartphone apps. O'Reilly Media, Inc., Sebastopol
43. Pernice K, Budiu R (2016) Hamburger menus and hidden navigation hurt ux metrics
44. Ross AS, Zhang X, Fogarty J, Wobbrock JO (2018) Examining image-based button labeling for accessibility in android apps through large-scale analysis. In: Proceedings of the 20th

international ACM SIGACCESS conference on computers and accessibility, ASSETS '18, Association for Computing Machinery, New York, NY, USA, pp 119–130
45. Sahami Shirazi A, Henze N, Schmidt A, Goldberg R, Schmidt B, Schmauder H (2013) Insights into layout patterns of mobile user interfaces by an automatic analysis of Android apps. In: Proceeding of the EICS
46. Saponas TS, Prabaker MK, Abowd GD, Landay JA () The impact of pre-patterns on the design of digital home applications. In: Proceedings of the 6th conference on designing interactive systems, ACM (2006), pp 189–198
47. Sereshkeh AR, Leung G, Perumal K, Phillips C, Zhang M, Fazly A, Mohomed I (2020) Vasta: a vision and language-assisted smartphone task automation system. In: Proceedings of the 25th international conference on intelligent user interfaces, IUI '20, Association for Computing Machinery, New York, NY, USA, pp 22–32
48. Siang TY (2015) Material design: why the floating action button is bad ux design. https://medium.com/tech-in-asia/material-design-why-the-floating-action-button-is-bad-ux-design-acd5b32c5ef, https://medium.com/tech-in-asia/material-design-why-, https://medium.com/tech-in-asia/material-design-why-the-floating-action-button-is-bad-ux-design-acd5b32c5ef-the-floating-action-button-is-bad-ux-design-acd5b32c5ef
49. Swearngin A, Li Y (2019) Modeling mobile interface tappability using crowdsourcing and deep learning. In: Proceedings of the 2019 CHI conference on human factors in computing systems, CHI '19, Association for Computing Machinery, New York, NY, USA, pp 1–11
50. Szydlowski M, Egele M, Kruegel C, Vigna G (2012) Challenges for dynamic analysis of iOS applications. In: Open problems in network security. Springer, pp 65–77
51. Tidwell J (2010) Designing interfaces: patterns for effective interaction design. O'Reilly Media, Inc., Sebastopol
52. Viennot N, Garcia E, Nieh J (2014) A measurement study of google play. In: ACM SIGMETRICS performance evaluation review, vol. 42, ACM, pp 221–233
53. Wania CE, Atwood ME (2009) Pattern languages in the wild: exploring pattern languages in the laboratory and in the real world. In: Proceedings of the 4th international conference on design science research in information systems and technology, ACM, p 12
54. Wu Z, Jiang Y, Liu Y, Ma X (2020) Predicting and diagnosing user engagement with mobile ui animation via a data-driven approach. In: Proceedings of the 2020 CHI conference on human factors in computing systems, CHI '20, Association for Computing Machinery, New York, NY, USA, pp 1–13
55. Zhang T, Liu Y, Gao J, Gao LP, Cheng J (2020) Deep learning-based mobile application isomorphic gui identification for automated robotic testing. IEEE Softw 37(4):67–74