Rico: A Mobile App Dataset for Building Data-Driven Design Applications

Biplab Deka[1] **Zifeng Huang**[1] **Chad Franzen**[1] **Joshua Hibschman**[2] **Daniel Afergan**[3]
Yang Li[3] **Jeffrey Nichols**[3] **Ranjitha Kumar**[1]
[1]University of Illinois at Urbana-Champaign [2]Northwestern University [3]Google Inc.
{deka2,zhuang45,cdfranz2,ranjitha}@illinois.edu, jh@u.northwestern.edu, {afergan,jwnichols}@google.com, yangli@acm.org

ABSTRACT
Data-driven models help mobile app designers understand best practices and trends, and can be used to make predictions about design performance and support the creation of adaptive UIs. This paper presents Rico, the largest repository of mobile app designs to date, created to support five classes of data-driven applications: design search, UI layout generation, UI code generation, user interaction modeling, and user perception prediction. To create Rico, we built a system that combines crowdsourcing and automation to scalably mine design and interaction data from Android apps at runtime. The Rico dataset contains design data from more than 9.7k Android apps spanning 27 categories. It exposes visual, textual, structural, and interactive design properties of more than 72k unique UI screens. To demonstrate the kinds of applications that Rico enables, we present results from training an autoencoder for UI layout similarity, which supports query-by-example search over UIs.

ACM Classification Keywords
D.2.2 Software Engineering: Design Tools and Techniques

Author Keywords
Mobile app design; design mining; design search; app datasets

INTRODUCTION
Data-driven models of design can scaffold the creation of mobile apps. Having access to relevant examples helps designers understand best practices and trends [13, 14, 23]. In the future, data-driven models will enable systems that can predict whether a design will achieve its specified goals before it is deployed to millions of people, and scale the creation of personalized designs that automatically adapt to diverse users and contexts. To build these models, researchers require design datasets which expose the details of mobile app designs at scale.

UIST 2017, October 22–25, 2017, Quebec City, QC, Canada

ISBN 978-1-4503-4981-9/17/10…$15.00
DOI: https://doi.org/10.1145/3126594.3126651

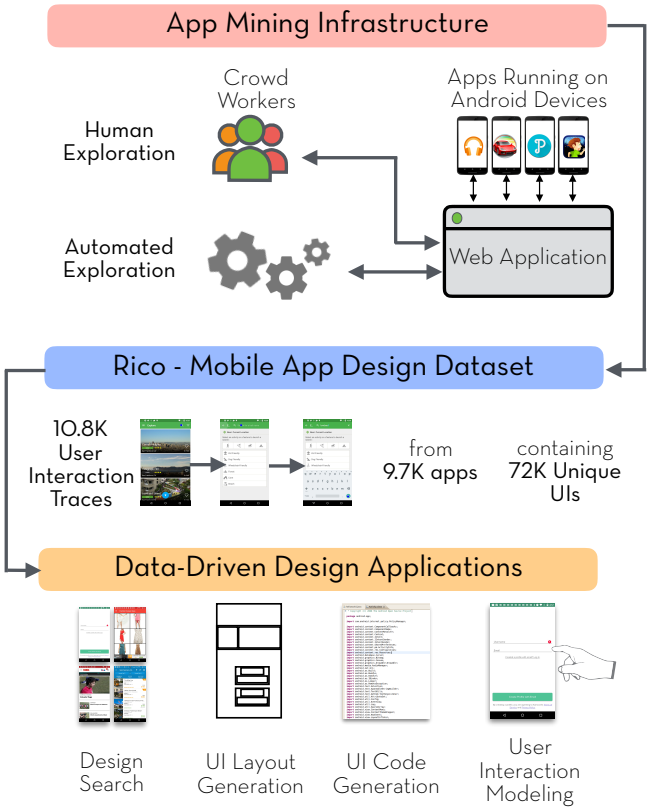

Figure 1: Rico is a design dataset with 72k UIs mined from 9.7k free Android apps using a combination of human and automated exploration. The dataset can power a number of design applications, including ones that require training state-of-the-art machine learning models.

This paper presents Rico[1], the largest repository of mobile app designs to date, comprising visual, textual, structural, and interactive properties of UIs. These properties can be combined in different ways to support five classes of data-driven applications: design search, UI layout generation, UI code generation, user interaction modeling, and user perception prediction.

Rico was built by mining Android apps at runtime via human-powered and programmatic exploration (Figure 1). Like its predecessor ERICA [11], Rico's app mining infrastructure re-

[1]Rico — a Spanish word meaning "rich" — is available for download at http://interactionmining.org/rico, and will be served there until at least 2022.

	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	●	●	●

Figure 2: A comparison of Rico with other popular app datasets.

quires 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.

The Rico dataset contains design and interaction 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*. Additionally, since the Rico dataset is large enough to support deep-learning applications, each UI is annotated with a low-dimensional vector produced by training an autoencoder for UI layout similarity, which can be used to cluster and retrieve similar UIs from different apps.

ANDROID APP DATASETS

Existing Android app datasets expose different kinds of information: Google Play Store metadata (e.g., reviews, ratings) [2, 16], software engineering and security related information [38, 15], and design data [33, 4, 11]. 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 2 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 [33, 4]. ERICA's dataset, on the other hand, is created by *dynamically* mining apps, and captures view hierarchies, screenshots, and user interactions [11].

Like the ERICA dataset, Rico is created by mining design and interaction data from apps at runtime. 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.

DATA-DRIVEN DESIGN APPLICATIONS

Rico was built to support a variety of data-driven applications for mobile app design. The data and representations exposed by Rico are motivated by five classes of design applications, which have been studied in a number of domains (Figure 3).

Design Search

Designers use examples for inspiration and for understanding the landscape of possible solutions. Existing design search systems span domains including web design [32, 19], mobile app flows [11], 3D modeling [17], interior design [7], fashion [22], and programming [10]. These systems often support keyword or query-by-example search, and return visual galleries of results that can easily be reviewed by the designer.

To support keyword search over mobile app designs, Rico exposes app-level metadata from the Google Play Store and element-level metadata contained within the Android view hierarchies. For each UI element, the view hierarchy exposes the text contained within the element, as well as the `classname` and `resource-id` properties specified by the app creator. This textual data often provides semantic clues about the element's functionality (e.g., search icon, login button), which can serve as *weak supervision* to classify semantic parts of mobile app design. Yi et al. leverage a similar form of weak supervision — artists' annotations contained in scene graphs — to label semantic parts of 3D models [39]. Designers can perform keyword searches over these functional semantic classes to find relevant elements or screens in a user interaction trace. To facilitate query-by-example search, Rico exposes a vector representation for each UI that encodes layout. Rico provides search engines with several visual representations that can be served up as results: UI screenshots, flows, and animations.

UI Layout Generation

Design datasets are also useful for training generative models of design. Prior work has learned generative models for arranging design elements and defining their attributes in domains such as graphic design [30] and 3D modeling [35].

	Design Search	Mobile Layout Generation	UI Code Generation	User Interaction Modeling	User Perception Prediction
UI Screenshots	●	●	●	●	●
View Hierarchies	●	●	●	●	○
User Interactions	●	○	○	●	○
Animations	●	○	○	○	●
UI Similarity Annotations	●	○	○	○	○
App Store Metadata	●	●	○	○	●

Figure 3: The five classes of design applications that Rico supports, correlated with the parts of the dataset intended to support them.

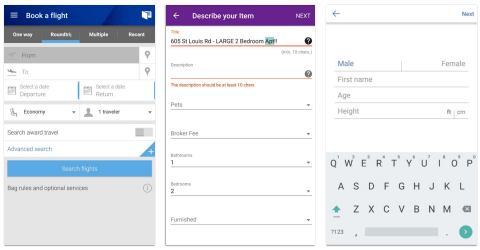

Figure 4: Automated crawlers are often stymied by UIs that require complex interaction sequences, such as the three shown here.

Researchers can similarly use the Rico dataset to train probabilistic generative models of UI layouts. The Android view hierarchy exposes all the elements comprising a UI screen, their attributes (e.g., position, dimensions), and the structural relationships between them. By combining screenshots and view hierarchies, researchers can compute visual features such as the color contrast between nested elements. Additionally, Play Store metadata can be leveraged to create specialized training sets. For example, app ratings and download metrics can be used as proxies for design quality, so that models can be optimized to emulate "good" layouts. Similarly, separate models can be trained for different app categories.

UI Code Generation

Once a mobile app is designed, implementing its interfaces and interactions in code can be a time consuming process. As a result, prior work has studied how different code components are used to implement UIs in popular apps [33] and developed models to automatically generate code from UI designs [29].

Design systems can leverage Rico to reverse engineer UIs. Since Rico's view hierarchies specify the Android components comprising a UI screen, a system could leverage a hierarchy and screenshot to generate Android source code that reproduces both the visual look and interactivity present in the original UI.

User Interaction Modeling

A key component of a mobile app's design is the interactivity of its various UIs and elements [11]. Modeling how users interact with different UIs can support better automated testing for apps as well as app optimizations that pre-fetch data by predicting a user's next action.

Like the ERICA dataset, Rico contains user interaction data captured while an app is being used. Each user trace for an app contains every user interaction event annotated with its type (such as "tap" or "scroll") and the UI element that reported it. By finding the same UI element in the corresponding view hierarchy, models can learn from a richer set of features based on the element's properties and metadata. Instead of predicting that a user will click on an element in the top-right corner of the screen, models can predict that a user will *check out*.

User Perception Prediction

Models of user perception help designers get early feedback on their designs. Prior work has explored models for predicting users' first impressions of web pages [31, 18], mobile app screens [25, 24], and mobile app icons [27]. Other research has focused on predicting longer-term perception based on animations [36], menus [21], and visual diversity and consistency between different screens [37, 26].

To build perceptual models of mobile design using the Rico dataset, systems can compute features over UI screenshots and animations, and correlate them with Play Store metrics. For example, researchers could examine correlations between an app's color palette and its average rating. In the future, researchers could serve these screenshots and animations to crowdsource additional perceptual annotations.

MINING APP DESIGNS

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 [6, 11]. Automated agents, on the other hand, can be used to exhaustively process the interactive elements on a UI screen [9, 34]; however, they can be stymied by UIs that require complex interaction sequences or human inputs (Figure 4) [5].

This paper proposes a hybrid approach for design mining mobile apps that combines 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.

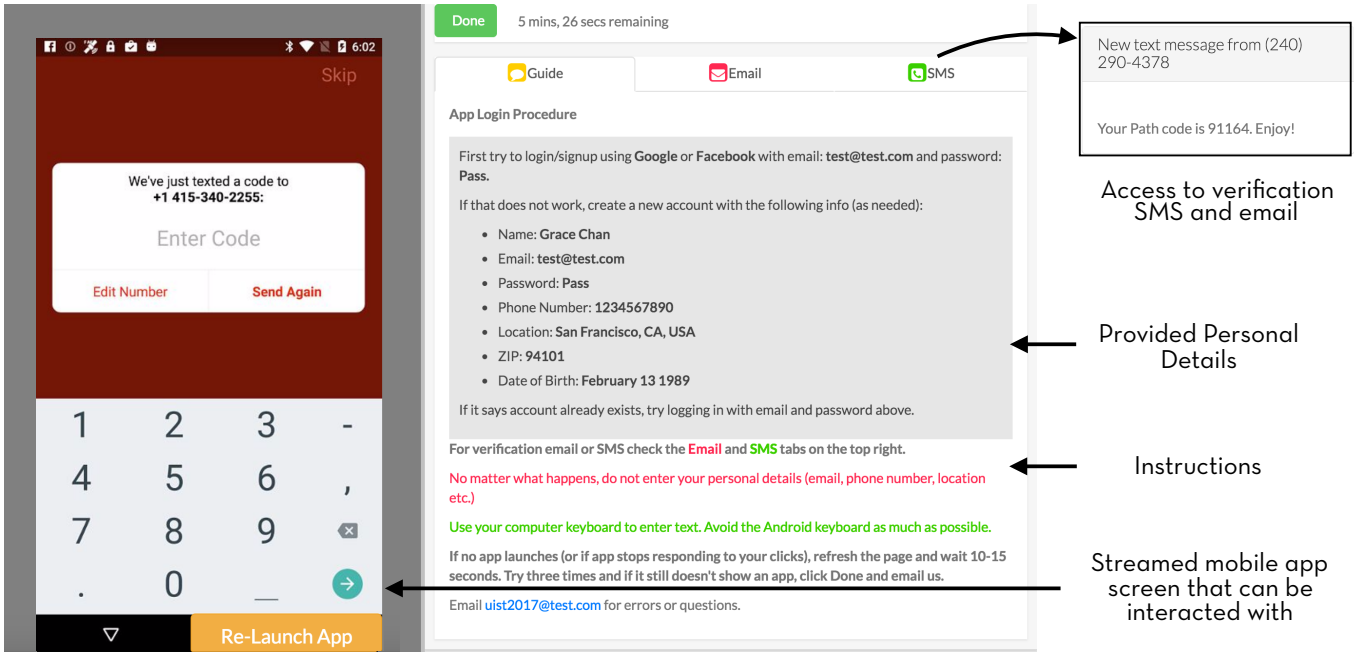

Figure 5: Our crowd worker web interface. On the left, crowd workers can interact with the app screen using their keyboard and mouse. On the right, they are provided instructions and details such as the name, location, phone number, and email address to use in app. The interface also allows workers to access SMS and email messages sent to the provided phone number and email to complete app verification processes.

Crowdsourced Exploration

The crowdsourced mining system uses a web-based architecture similar to ERICA [11]. 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 Figure 5. 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.

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 [20, 5]. We achieve a similar result by leveraging the interaction data contained within the collected user traces: when the crawler encounters a 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 [20, 5], 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.

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 (Figure 6). Therefore, we employ a *content-agnostic similarity heuristic* to compare UIs.

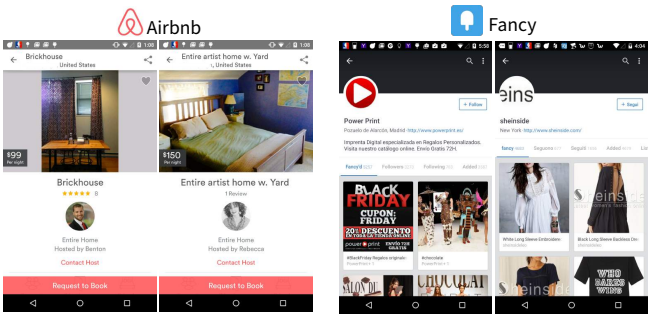

Figure 6: Pairs of UI screens from apps that are visually distinct but have the same design. Our content-agnostic similarity heuristic uses structural properties to identify these sorts of design collisions.

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 $\beta = 1$ produces 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.

Coverage Benefits of Hybrid Exploration

To measure the coverage benefits our hybrid exploration approach, we compare Rico's crawling strategy to human and automated exploration alone. We selected 10 apps (Figure 7) 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, 6, 11] 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 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 8 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.

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

Figure 7: The Android apps used in our evaluation. Each had a rating higher than 4 stars (out of 5) and more than 1M downloads on the Google Play store.

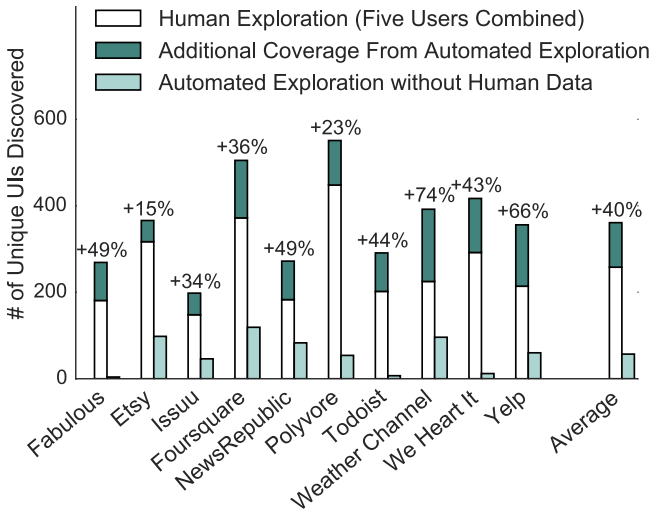

Figure 8: The performance of our hybrid exploration system compared to human and automated exploration alone, measured across ten diverse Android apps.

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 the gated experiences that pure, automated approaches cannot handle. For instance, Todoist and WeHeartIt hide most of their features behind a login wall.

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 (Figure 9). We excluded from our crawl categories that primarily involve multimedia (such as video players and photo editors) as well productivity and personalization apps. Apps in the Rico dataset have an average rating of 4.1 stars, and data pertaining to 26 user interactions.

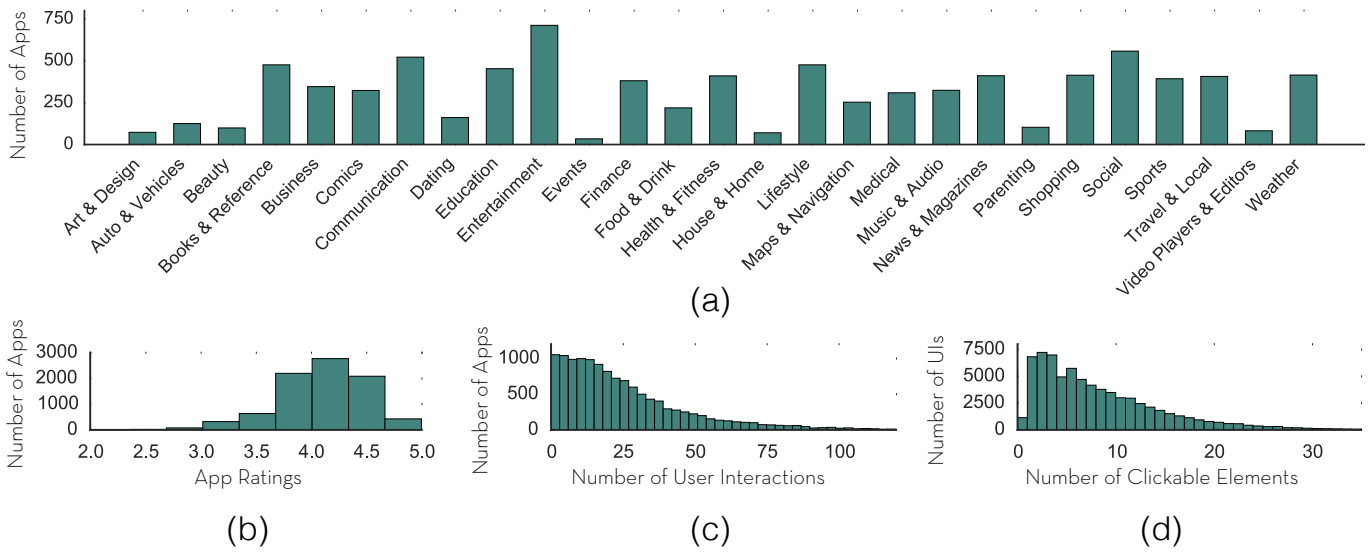

Figure 9: Summary statistics of the Rico Dataset: app distribution by (a) category, (b) average rating, and (c) number of mined interactions. (d) The distribution of mined UIs by number of interactive elements.

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 minutes.

In total, workers spent 2,450 hours 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.

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 (Figure 10). 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 interaction, 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 3M elements, of which approximately 500k are interactive. On average, each UI comprises eight interactive elements.

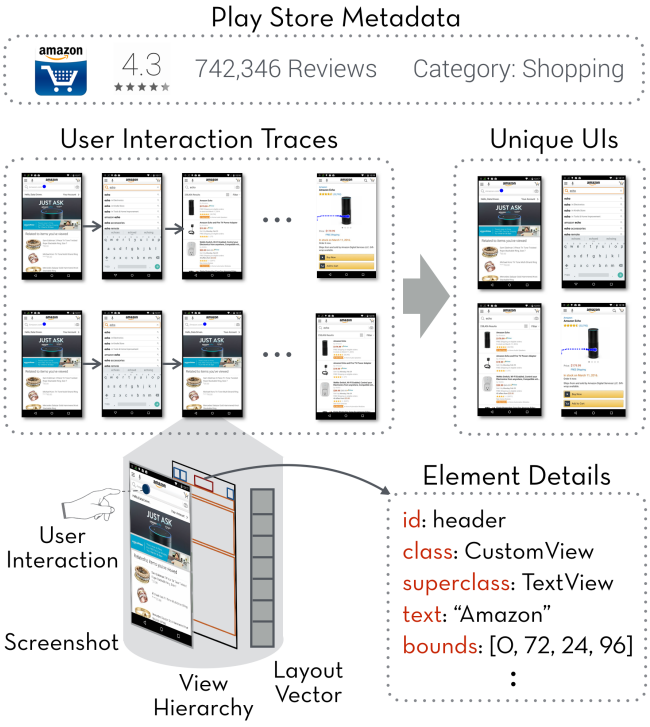

Figure 10: The Rico dataset contains Google Play Store metadata, a set of user interaction traces, and a list of all the unique UIs discovered during crawling.

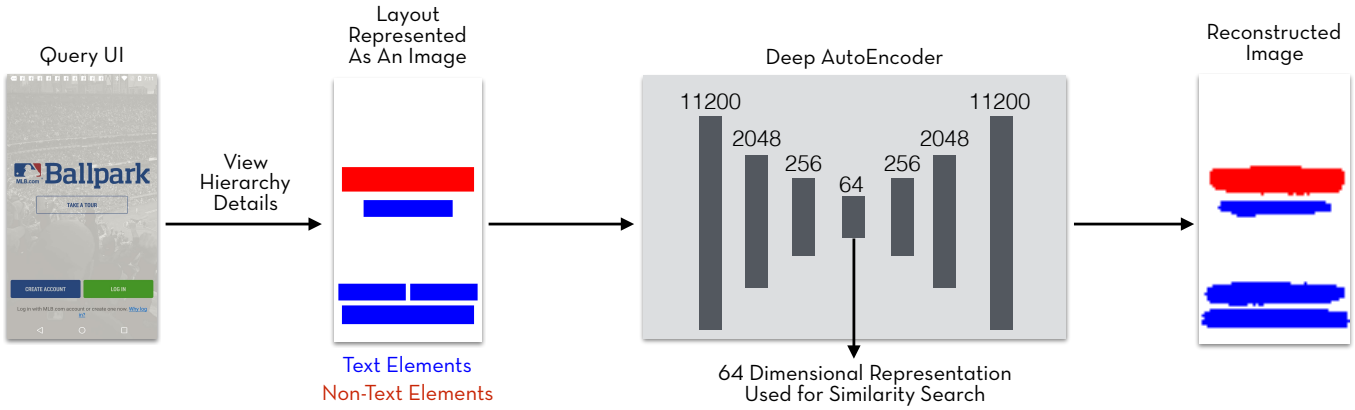

Figure 11: We train an autoencoder to learn a 64-dimensional representation for each UI in the repository, encoding structural information about its layout. This is accomplished by creating training images that encode the positions and sizes of elements in each UI, differentiating between text and non-text elements.

Training a UI Layout Embedding

Since the Rico dataset is large and comprehensive enough to support deep learning applications, we trained an 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 (Figure 12).

An autoencoder is a neural network that involves two models — an encoder and a decoder — to support the *unsupervised* learning of lower-dimensional representations [8]. 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.

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 (Figure 11). Rico's view hierarchies obviate the need for noisy image processing or OCR techniques to create these inputs. In the future, if we can predict functional semantic labels for elements such as *search icon* or *login button*, we can train embeddings with even richer semantics.

The encoder has an input dimension of $11,200$, an output dimension of 64, and uses two hidden layers of dimension $2,048$ and 256 with ReLU non-linearities [28]. The decoder has the reverse architecture. We trained the autoencoder with 90% of our data and used the rest as a validation set, and found that the validation loss stabilized after 900 epochs or approximately 5 hours on a Nvidia GTX 1060 GPU. Once the autoencoder was trained, we used the encoder compute a 64-dimensional representation for each UI, which we expose as part of the Rico dataset.

Figure 12 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.

FUTURE WORK

The are a number of opportunities to extend and improve the Rico dataset. New models could be trained to annotate Rico's design components with richer labels, like classifiers that describe the semantic function of elements and screens (e.g., search, login). Similarly, researchers could crowdsource additional perceptual annotations (e.g., first impressions) over design components such as screenshots and animations, and use them to train newer types of perception-based predictive models.

Unlike static research datasets such as ImageNet [12], Rico will become outdated over time if new apps are not continually crawled and their entries updated in the database. Therefore, another important avenue for future work is to explore ways to make app mining more sustainable. One potential path to sustainability is to create a platform where designers can use apps and contribute their traces to the repository for the entire community's benefit.

To download the Rico dataset — or learn more about the project — visit `http://interactionmining.org/rico`.

ACKNOWLEDGMENTS

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`.

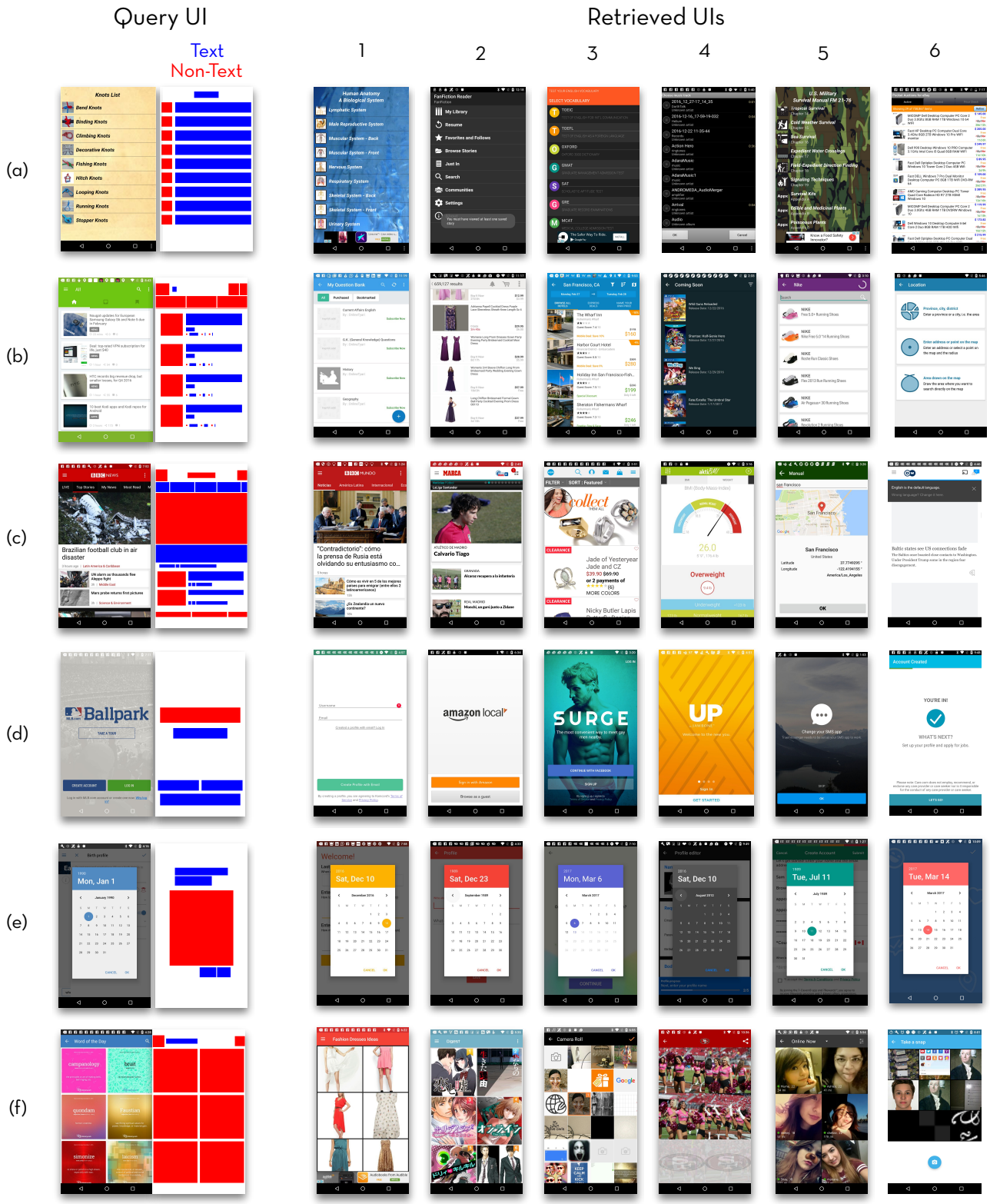

Figure 12: The top six results obtained from querying the repository for UIs with similar layouts to those shown on the left, via a nearest-neighbor search in the learned 64-dimensional autoencoder space. The returned results share a common layout and even distinguish between layout nuances such as lists composed of smaller and larger image thumbnails (a,b).

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. Alharbi, K., and Yeh, T. Collect, decompile, extract, stats, and diff: Mining design pattern changes in Android apps. In *Proc. MobileHCI* (2015).
5. Amini, S. *Analyzing Mobile App Privacy Using Computation and Crowdsourcing*. PhD thesis, Carnegie Mellon University, 2014.
6. Azim, T., and Neamtiu, I. Targeted and depth-first exploration for systematic testing of android apps. In *ACM SIGPLAN Notices* (2013).
7. Bell, S., and Bala, K. Learning visual similarity for product design with convolutional neural networks. *ACM TOG* (2015).
8. Bengio, Y. Learning deep architectures for ai. *Foundations and Trends in Machine Learning 2*, 1 (2009).
9. Bhoraskar, R., Han, S., Jeon, J., Azim, T., Chen, S., Jung, J., Nath, S., Wang, R., and Wetherall, D. Brahmastra: Driving apps to test the security of third-party components. In *Proc. SEC* (2014).
10. Brandt, J., Dontcheva, M., Weskamp, M., and Klemmer, S. R. Example-centric programming: integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2010), 513–522.
11. Deka, B., Huang, Z., and Kumar, R. ERICA: Interaction mining mobile apps. In *Proc. UIST* (2016).
12. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR* (2009).
13. Eckert, C., and Stacey, M. Sources of inspiration: A language of design. *Design Studies 21*, 5 (2000), 523–538.
14. Eckert, C., Stacey, M., and Earl, C. References to past designs. *Studying Designers 5* (2005), 3–21.
15. Frank, M., Dong, B., Felt, A. P., and Song, D. Mining permission request patterns from android and facebook applications. In *Proc. ICDM* (2012).
16. Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J., and Sadeh, N. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proc. KDD* (2013).
17. Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., and Jacobs, D. A search engine for 3D models. *ACM TOG* (2003).
18. Koch, J., and Oulasvirta, A. Computational layout perception using gestalt laws. In *Proc. CHI (Extended Abstracts)* (2016).
19. Kumar, R., Satyanarayan, A., Torres, C., Lim, M., Ahmad, S., Klemmer, S. R., and Talton, J. O. Webzeitgeist: Design mining the web. In *Proc. CHI* (2013).
20. Lee, K., Flinn, J., Giuli, T., Noble, B., and Peplin, C. Amc: Verifying user interface properties for vehicular applications. In *Proc. Mobisys* (2013).
21. Leuthold, S., Schmutz, P., Bargas-Avila, J. A., Tuch, A. N., and Opwis, K. Vertical versus dynamic menus on the world wide web: Eye tracking study measuring the influence of menu design and task complexity on user performance and subjective preference. *Computers in human behavior 27*, 1 (2011), 459–472.
22. McAuley, J., Targett, C., Shi, Q., and Van Den Hengel, A. Image-based recommendations on styles and substitutes. In *Proc SIGIR*, ACM (2015), 43–52.
23. Miller, S. R., and Bailey, B. P. Searching for inspiration: An in-depth look at designers example finding practices. In *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (2014).
24. Miniukovich, A., and De Angeli, A. Visual impressions of mobile app interfaces. In *Proc. Nordic CHI* (2014).
25. Miniukovich, A., and De Angeli, A. Computation of interface aesthetics. In *Proc. CHI* (2015).
26. Miniukovich, A., and De Angeli, A. Visual diversity and user interface quality. In *Proc. British HCI* (2015).
27. Miniukovich, A., and De Angeli, A. Pick me!: Getting noticed on google play. In *Proc. CHI* (2016).
28. Nair, V., and Hinton, G. E. Rectified linear units improve restricted boltzmann machines. In *Proc. ICML* (2010), 807–814.
29. Nguyen, T. A., and Csallner, C. Reverse engineering mobile application user interfaces with REMAUI. In *Proc. ASE* (2015).
30. ODonovan, P., Agarwala, A., and Hertzmann, A. Learning layouts for single-page graphic designs. *IEEE TVCG* (2014).
31. Reinecke, K., Yeh, T., Miratrix, L., Mardiko, R., Zhao, Y., Liu, J., and Gajos, K. Z. Predicting users' first impressions of website aesthetics with a quantification of perceived visual complexity and colorfulness. In *Proc. CHI* (2013).
32. Ritchie, D., Kejriwal, A. A., and Klemmer, S. R. d. tour: Style-based exploration of design example galleries. In *Proc. UIST* (2011).
33. Sahami Shirazi, A., Henze, N., Schmidt, A., Goldberg, R., Schmidt, B., and Schmauder, H. Insights into layout patterns of mobile user interfaces by an automatic analysis of Android apps. In *Proc. EICS* (2013).
34. Szydlowski, M., Egele, M., Kruegel, C., and Vigna, G. Challenges for dynamic analysis of iOS applications. In *Open Problems in Network Security*. Springer, 2012, 65–77.

35. Talton, J., Yang, L., Kumar, R., Lim, M., Goodman, N., and Měch, R. Learning design patterns with bayesian grammar induction. In *Proc. UIST*, ACM (2012).

36. Tractinsky, N., Inbar, O., Tsimhoni, O., and Seder, T. Slow down, you move too fast: Examining animation aesthetics to promote eco-driving. In *Proc. AutoUI*, ACM (2011), 193–202.

37. van der Geest, T., and Loorbach, N. Testing the visual consistency of web sites. *Technical communication 52*, 1 (2005), 27–36.

38. Viennot, N., Garcia, E., and Nieh, J. A measurement study of google play. In *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, ACM (2014), 221–233.

39. Yi, L., Guibas, L., Hertzmann, A., Kim, V. G., Su, H., and Yumer, E. Learning hierarchical shape segmentation and labeling from online repositories. In *Proc. SIGGRAPH* (2017).