# Towards Never-ending Learning of User Interfaces

**Jason Wu** [1 2] **Rebecca Krosnick** [3 2] **Eldon Schoop** [2] **Amanda Swearngin** [2] **Jeffrey P. Bigham** [2] **Jeffrey Nichols** [2]

## Abstract

Machine learning models have been trained to predict semantic information about user interfaces (UIs) to make apps more accessible, easier to test, and to automate. Currently, most models rely on datasets of static screenshots that are labeled by human crowd-workers, a process that is costly and surprisingly error-prone for certain tasks. For example, workers labeling whether a UI element is "tappable" from a screenshot must guess using visual signifiers, and do not have the benefit of tapping on the UI element in the running app and observing the effects. In this paper, we present the Never-ending UI Learner, an app crawler that automatically installs real apps from a mobile app store and crawls them to infer semantic properties of UIs by interacting with UI elements, discovering new and challenging training examples to learn from, and continually updating machine learning models designed to predict these semantics. The Never-ending UI Learner so far has crawled for more than 5,000 device-hours, performing over half a million actions on 6,000 apps to train a highly accurate tappability model.

## 1. Introduction

Machine Learning (ML) has played an increasingly important role in the domain of mobile User Interfaces (UIs). Recent techniques have used Deep Neural Networks (DNNs) to bridge critical usability gaps and enable new types of evaluations, such as providing missing accessibility metadata to UIs (Wu et al., 2021), giving designers feedback to make UI features more discoverable (Swearngin & Li, 2019; Schoop et al., 2022b), and predicting user engagement with animations (Wu et al., 2020). The enabling research artifacts behind these interactions are large datasets of mobile UI screenshots annotated by human crowdworkers (Deka et al., 2017; Kuznetsov et al., 2021). These datasets provide an invaluable volume of data for training DNNs, but they only capture a fixed snapshot of the views of mobile applications and are extremely costly to collect and update. In addition, relying on crowd-workers to estimate certain properties of UI elements from static visual signifiers is known to be error-prone (Schoop et al., 2022b). Inspired by the Never Ending Learning paradigm (Mitchell et al., 2018), we propose an automated method for collecting UI element annotations by *interacting with applications directly* with an automated crawler that continuously improves its own performance and can refresh ML models for other downstream tasks over time.

We built the *Never-ending UI Learner*, an app crawler that formulates UI semantic learning as an active process that uses real interactions on real devices to explore UIs and discover properties which are used to continually train machine learning models. More specifically, our crawler automatically installs real apps from mobile app stores and crawls them to discover new, challenging training examples to learn from (e.g., those that result in low model confidence). During crawling, the Never-ending UI Learner records temporal context (i.e., taking screenshots before, during, and after interactions) that is used by heuristic functions to generate more accurate labels than are possible from human-annotated single screenshots. The resulting data is used to train models that predict semantics from UIs, such as element tappability. Although the process can start with a model trained from human-labeled data, the end-to-end process does not require any additional human-labeled examples.

In contrast to existing data pipelines for data-driven UI modeling (Deka et al., 2017; Zhang et al., 2021; Kuznetsov et al., 2021), our never-ending UI learning paradigm allows data collection, annotation, and model training to be performed without any human supervision and can be run indefinitely. Of course, in this paper the learning is not truly never-ending. Here we present experiments that analyze the performance characteristics of our learner over 5,000 device-hours, in which it performed more than half a million actions on 6,000 apps. The resulting dataset is an order of magnitude larger than existing human-annotated UI datasets (Zhang et al., 2021; Deka et al., 2017) and allowed us to analyze the performance of UI semantic models when

[1]HCI Institute, Carnegie Mellon University [2]Apple Inc. [3]Computer Science and Engineering, University of Michigan. Correspondence to: Jason Wu <jsonwu@cmu.edu>.

*AI & HCI Workshop at the $40^{th}$ International Conference on Machine Learning (ICML)*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

trained with increasing amounts of recently collected examples. Ultimately, we believe this model can be used in a true "never-ending" style, continually crawling the app ecosystem, collecting data from literally all available apps, and experiencing new UI styles and trends as new or updated apps are released.

The specific contributions of our paper are as follows:

1. **The Never-ending UI Learner**, is a system that operationalizes our approach for automatically learning from UIs through never-ending interaction.
2. **An application which demonstrates the use of the Never-ending UI Learner**. We use our crawler to train a model for predicting "tappability", a UI semantics that has been shown by previous work (Schoop et al., 2022a) to be difficult for human annotators to label.

## 2. Related Work

Our work in never-ending learning of UIs aims to supplement UI modeling datasets used to model UIs and user interaction through continual learning. To situate our work, we review related literature in the i) UI modeling datasets, ii) computational models of interaction, and iii) approaches for continual machine learning.

### 2.1. Datasets for Modeling User Interfaces

Several datasets have been collected for the purposes of analyzing and modeling mobile UIs. The Rico dataset is a large dataset of 72,000 mobile UIs and associated metadata including view hierarchies, screenshots, and user interactions, collected from 9,700 publicly available Android apps (Deka et al., 2017). The FrontMatter dataset uses static analysis techniques to predict the purpose of UI elements by determining which system APIs are invoked (Kuznetsov et al., 2021). Large datasets like these have enabled ML-based methods which can perform various tasks involving mobile UIs, including providing accessibility annotations (Wu et al., 2021; Li et al., 2020b), giving design feedback (Huang et al., 2019; Swearngin & Li, 2019; Schoop et al., 2022b; Yuan & Li, 2020), suggesting common interaction flows (Zhou & Li, 2021), summarizing screens (Wang et al., 2021), automating interaction with UIs (Li et al., 2020a; Sereshkeh et al., 2020; Arsan et al., 2021), and creating rich embeddings of UI image and text data for other downstream tasks (Li et al., 2021; Bai et al., 2021; He et al., 2021). Almost all currently available datasets are manually created in some aspect – through manual user interactions with UIs, and/or human annotations. The WebUI dataset (Wu et al., 2023) used screenshots and automatically extracted metadata from web pages to train visual UI models; however, web data was generally noisy and their models needed additional fine-tuning on smaller human-annotated datasets to perform well. Our Never-ending UI Learner produces annotations through the automated crawling of mobile applications. These annotations continually update and refresh the crawler's models, improving its performance, and resulting in a continually updated dataset that can be used to train other models. An important advantage of this approach is that, unlike using mobile UI data collected during a specific time period, data produced by our crawler is always current, and can support updating models to keep up with evolving UI design trends in mobile applications.

### 2.2. Computational Modeling of Interaction

More recently, Reinforcement Learning (RL) has been applied to model user interactions with both physical and digital interfaces. Oulasvirta et al. proposed a general framework based on RL of how users incorporate cognitive facilities, their experiences, and their environment in understanding and interacting with computers (Oulasvirta et al., 2022). Under this context, an important part of knowing how to interact with an interface is by understanding its affordances. Affordances are the functional properties of an object (e.g., UI) that suggest how it should be used (Norman, 1988), and designer commentary suggests that design patterns can make affordance discovery more difficult. Liao et al. used a virtual robot agent equipped with sensors to simulate and learn how humans may discover affordances in physical interfaces (e.g., buttons and sliders) (Liao et al., 2022). Our work aims to achieve similar goals of learning the affordances (e.g., tappability) and capabilities of interfaces. While our work does not directly model the interactions of users through RL techniques, we aim to achieve similar goals of learning of the affordances (e.g., tappability) and capabilities of interfaces through interacting with and inspecting live mobile apps. By applying interaction learning to a mobile app automated crawler, we can scale our experiments to a much larger scale, learning from millions of interactions with UIs.

### 2.3. Continuous Machine Learning

A unique aspect of our work is the intention to continually learn about UIs over time through sustained, potentially endless interaction. Our work is related to active learning (specifically online active learning), which is a field of ML that seeks to improve models using only a limited number of human-labeled examples (Goodfellow et al., 2016). These approaches often identify and prioritize difficult or representative examples to produce the best possible model from a small dataset. Our work is most related to Never-Ending Learning, which is an ML paradigm for creating systems that continually learn from acquired experience rather than a single dataset. It was first applied to web-based knowledge using the NELL system (Mitchell et al., 2018). The system has been running for prolonged peri-

ods of time (years) and has accumulated over 50 million beliefs (i.e., hypothesized knowledge snippets), which is possible only by processing large amounts of data that are prohibitively expensive to annotate. This learning approach introduces unique challenges, such as the need to learn from new data while retaining previously acquired knowledge. There are several techniques in the literature that can be applied to retain previous knowledge that involved i) regularization (Li & Hoiem, 2017; Kirkpatrick et al., 2017), ii) rehearsal-based approaches (Rebuffi et al., 2017), and iii) techniques that address task-recency bias (Castro et al., 2018). From a practical standpoint, implementation also necessitates maintaining large ever-growing datasets collected over time, which could either be addressed through a robust crawling infrastructure or using dataset distillation methods that keep the most relevant samples (Wang et al., 2018; Nguyen et al., 2020; 2021). In this work, we apply the never-ending learning paradigm to benefit automated UI understanding systems by training models "from scratch" and fine-tuning existing models to improve performance.

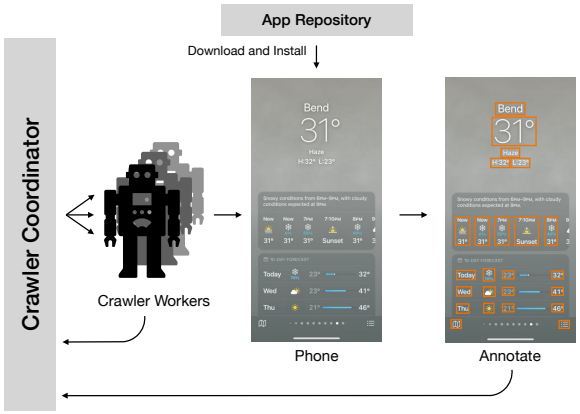


*Figure 1.* Architecture of our Never-ending UI Learner. The Never-ending UI Learner is a parallelizable mobile app crawler which consists of a coordinator-worker architecture. The crawler coordinator distributes crawls to workers and maintains the dataset. Each crawler worker is connected to a programatically controlled mobile device which collects data and runs data post-processing.

## 3. Never-ending UI Learner

To operationalize our approach, we built the Never-ending UI Learner, a system that automatically downloads and crawls publicly available apps using remotely operated devices. Our current implementation and infrastructure is based on iOS. We use stock factory reset devices that are logged in to testing accounts that are not associated with any real user data to avoid privacy concerns.

Note that unlike some crawlers that interact with apps using an OS-provided programmatic interface such as the accessibility API, our crawler interacts with the device through the VNC remote desktop protocol, from which it receives regular updates to the screen and processes them visually and can send raw input events to the device to create tap, swipe and keyboard actions. Using VNC, the Never-ending UI Learner is able to reliably interact with more apps, learn based on the same facilities that a human would and generalize to other platforms. In this section, we describe the crawler's architecture and behavior that enable it to perform never-ending learning.

### 3.1. Architecture Overview

Our crawling architecture is shown in Figure 1. We implemented a distributed crawling architecture which consists of i) a central coordination server and ii) a large pool of workers to parallelize the crawling process.

#### 3.1.1. Coordinator Server

The crawler coordination server maintains a list of app IDs to crawl which are sent to workers. The central server keeps track of successful and unsuccessful crawls, and it automatically retries failed app crawls. App crawlers differ from web crawlers in that they focus only on the app they are asked to crawl, although limited cross-app interaction sometimes does occur (e.g., clicking on a link or permission request dialog). When all app IDs are exhausted, our crawler can schedule itself to be run again after a fixed time period (e.g., weekly). The list of app IDs can be modified between crawls to add new apps or reflect changes in app availability. While the majority of the app IDs remain the same, the apps may change their appearance and behavior due to dynamically updated content and new versions of the software. Re-crawling the same apps regularly can enable our model to adapt to design changes over time.

#### 3.1.2. Crawler Worker

Crawler workers are processes that interface with remotely controlled mobile phones and process the collected data. Each crawler worker downloads and installs a target app whose ID is provided by the central server to the mobile phone and then runs a program that crawls the app. Screenshots are collected during interactions and when the crawler believes it has arrived at a new screen. The program can use a variety of methods to explore the target app and as a part of this paper, we run experiments to determine the best crawling strategy for each of our never-ending learning use-cases. We set a time-limit (5 minutes) on the maximum duration of a crawl for a single app. Afterwards, the worker processes the collected data (e.g., screenshots and interactions) with models and heuristics to generate labels from the observations. Both raw data and processed output is uploaded to a coordinator server.

### 3.2. Machine Learning Components

Our crawler contains a screen-level and element-level model that allow it to understand the content on UIs it encounters. We run these models every time a screenshot is captured to augment it with useful semantics. Furthermore, the three UI semantic models that we trained in using the crawler, are designed as extensions of these base models, improving overall efficiency.

#### 3.2.1. Screen Understanding

To keep track of its crawling progress in the app, our crawler uses a model to generate semantic representations of screens. We used a model introduced by previous work (Feiz et al., 2022) that predicts whether two screenshots belong to the same UI by encoding each as an embedding vector, which the authors shared with us. Because significant variation can be introduced by changes in state, such as a news app that displays new content periodically, the model is designed to learn the underlying structure of UIs. We made minor modifications to the previous work in order to develop a model that could run under our hardware constraints. Instead of their recommended *screen transformer* model architecture, we use their CNN-based model architecture, which is more efficient to run despite somewhat lower performance (Feiz et al., 2022). For further optimization, we use an EfficientNet (Tan & Le, 2019) model architecture as the backbone instead of the original ResNet-18 (He et al., 2016), which has more parameters. During training, we applied a data augmentation approach (Thakur et al., 2020) to increase performance. We followed all other aspects of the original model training and our final CNN-based model achieves a F1-score of 0.636.

#### 3.2.2. Element Understanding

To generate element semantics, we used an object detection model architecture that is similar to CenterNet (Zhou et al., 2019). At a high level, the detection model slides a window (via convolutions) over the image and featurizes image sub-regions using a backbone network (MobileNet-v1 (Howard et al., 2017)), resulting in embeddings for each region. These embeddings are fed into a classification head which produces per-class confidences, and regions with high confidences are returned as detections. The model was trained on the AMP dataset (Zhang et al., 2021), which consists of 77,000 app screens collected and annotated by crowdworkers from 4,000 iPhone apps. In addition to the standard element type classification head, which was trained with the rest of the object detection model, we added an additional head for tappability prediction. The tappability head is trained independently from the rest of the model by first freezing the backbone and training the heads on embeddings corresponding to detected elements.

## 4. Applying Never-ending Learning

In this section, we describe the application of our never-ending learning framework to tappability prediction, and we trained a model completely from crawler-generated data We developed an interaction-based heuristic used by our crawler to automatically generate new training examples for our models. Next, we designed and trained models to predict each of these semantics from a screenshot. Finally, to contextualize these models in the context of never-ending learning, we analyzed their performance over time.

**Experimental Setup.** We conducted experiments on a list of 6,461 free iOS apps. For the purposes of evaluation, all model training and experiments were performed with randomized training (80%), validation (10%) , and testing (10%) splits. We randomly partitioned our list of app IDs, which ensured that all UI screens from an app were contained in the same split. We use the term *crawl epoch* to refer to one complete pass through the list of apps. Note that unlike an *epoch* through a training dataset, the actual contents of a *crawl epoch* might change from time to time, due to the dynamic nature of apps.

Our experiments analyzed two aspects of the crawler's performance: i) crawling strategy and ii) performance over time. We ran three variations of the crawler, which had different crawling strategies: i) randomly selecting elements on each screen (Random), ii) selecting elements that result in low prediction confidence from the current models (Uncertainty Sampled), and iii) a hybrid that for each crawl epoch alternates between Random and Uncertainty Sampled strategies. To evaluate the performance over time, we ran each crawling strategy for five crawl epochs. Note that the first crawl epoch for all strategies uses Random to train an initial confidence-prediction model. In the Hybrid strategy, because alternation happens at the epoch level, the second epoch is crawled using the Uncertainty Sampled strategy and thus through two epochs the inputs and results are identical for both the Uncertainty Sampled and Hybrid strategies. The three strategies fully diverge starting from the third epoch.

The crawler's models were trained and evaluated after each crawl epoch. Models were trained on all data collected so far for 100,000 optimization steps, with early stopping. When training, models were initialized using weights from the last epoch's checkpoint, which improved convergence. In order to maintain a constant validation set across a varying number of epochs, we only use the evaluation data split from the first epoch for calculating performance metrics. Finally, we performed additional sub-epoch evaluations during the first crawl to analyze learning speed.

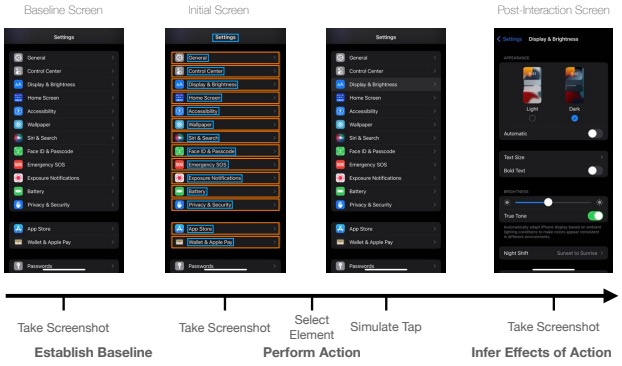


*Figure 2.* This figure visualizes the steps to our tappability heuristic. When the crawler arrives at a new screen, it takes two screenshots separated by 5 seconds as a baseline of visual change. Then, a detected UI element is chosen and sent a tap. After waiting for the screen to settle, a post-interaction screenshot is used to infer the effects of the action.

### 4.1. Tappability

Tapping is the most common interaction on mobile devices, yet it is often difficult to automatically determine if an element is tappable or not due to missing metadata and ambiguous visual cues. Accurate inference of tappability could aid designers in finding ambiguous visual elements and be useful for generating metadata for repairing inaccessible apps. Previous work has used human-annotated UI screenshots to train machine learning models of tappability. However, this process is surprisingly error-prone (Li et al., 2022; Leiva et al., 2021; Swearngin & Li, 2019; Schoop et al., 2022b) due to ambiguous visual cues, which suggests that human-annotated screenshots are an unreliable source of ground-truth for training tappability models. In contrast, our crawler can use additional context from the entire interaction, such as before and after screenshots instead of a single before screenshot, to determine if tapping resulted in an effect. Effects could either be state changes, like flipping a toggle, or a transition to a new screen. We developed a heuristic for inferring tappability from our crawler's recorded interactions and found that it had high agreement with human-annotated videos. We used heuristic-labeled data to train an efficient tappability "head" model purely from crawler-annotated data. After five crawl epochs, the best-performing tappability model reached an F1 score of 0.860.

#### 4.1.1. Tappability Heuristic

We developed a heuristic to infer the tappability of an element based screenshots of the UI taken before, during, and after a tap interaction. A tap may result in several different scenarios, which are captured by our heuristic. First, we use a screen similarity model to compare screenshots taken before and after the tap to determine if the tap led the crawler to a new screen. If a screen change was not detected, the tap could have also changed the screen state. We compute a pixel-based difference of the "before" and "after" screenshots to identify possible visual indications of local or global changes, such as tapping a checkbox or refreshing screen content respectively. Finally, to reduce false positives, the heuristic also uses multiple screenshots captured before the tap to identify dynamic areas of the screen (e.g., videos) whose visual changes are not related to the tap.

To validate the accuracy of our heuristics, we compared its results against human-labeled interaction videos. We used our crawler to save short screen recordings of tap interactions that were collected during crawls. Each example video was approximately 10 seconds long and included the tap location overlaid on the video and temporal context before and after the tap interaction, such as including transition and loading animations.

We randomly sampled a balanced subset of 1000 video clips from our crawls and asked crowd-workers if each video clip contained a tap interaction. We used standard classification metrics to evaluate the accuracy of our heuristics, using the human-annotated labels as ground truth. The tappability heuristic had an overall accuracy of 0.934, and had a similar number of false positives (38 instances) and false negatives (28 instances).

#### 4.1.2. Model Implementation

To predict tappability, we designed a model architecture that operates as a "head" of our existing element detection model (Figure 3). Heads are small sub-networks or set of layers usually located close to the output layer of neural network architectures and generate predictions from featurized representations of the main input produced by a "backbone" network. Since element detection is closely related to tappability, we hypothesized that the previously learned representations are likely to contain relevant information and greatly accelerate tappability learning. Our head model is a simple three-layer feed-forward model with an input size of 128, a hidden size of 64 that we chose through manually tuning, and an output size of 1 that gives tappability confidence. To train it, we first froze the weights of the element detector's backbone network and randomly initialized the parameters of our feed-forward network. While freezing most of the model reduces its capacity, it also results in a significant reduction in training time, since there are fewer parameters to optimize. Then we trained the model to predict the tappability of an element from a screenshot of the UI before the tap, and we used the labels generated by our tappability heuristic as ground truth.

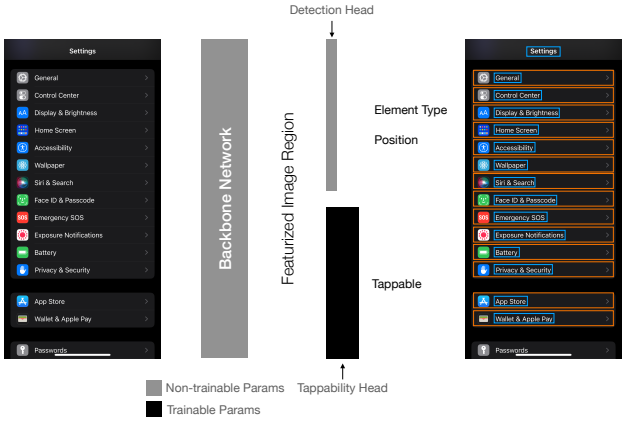


*Figure 3.* Architecture of our tappability model. The tappability model is designed as a "head", which is a sub-network of the UI element detection model. The element detector featurizes image regions in an input screenshot using a sliding window, which results in a featurized image embedding for each detected object. The main branch of the network (top) feeds in the embedding to determine the region's element type and position. We feed in the same element embedding into a separate feedforward network (bottom) to predict the probability that it is tappable.

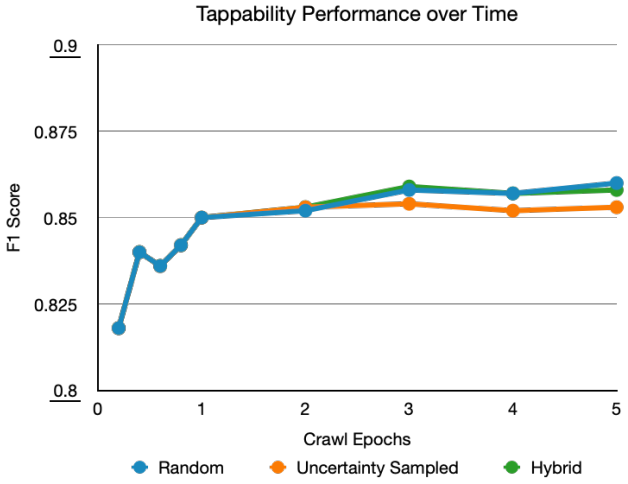


*Figure 4.* Performance of tappability over time. The model performance increases most rapidly during the first crawl epoch and the rate of improvement plateaus afterward. After the final epoch, the random crawler achieves the highest F1 score of 0.860, and the uncertainty sampled crawler has the lowest F1 score of 0.853.

#### 4.1.3. Performance Evaluation

The results of our experiments are shown in Figure 4. While all crawling strategies are successful in improving on the initial model from the first epoch, the Random crawler has the best final performance. In our experiments, the Random crawler reaches the best final F1 score of 0.860 while the Uncertainty Sampled crawler reaches the lowest final F1 score of 0.853. While it is not possible to make a direct comparison with previous work (Schoop et al., 2022a; Swearngin & Li, 2019) because their experiments were run on different datasets, it seems that our tappability model is able to reach similar levels of performance in terms of F1 score after its first epoch.

We also conducted a comparison between the quality of our automatically collected tappability dataset and human-annotated ones, we used the labels provided by the AMP dataset (Zhang et al., 2021). First, we trained our classification head model architecture on AMP, which led to similar performance (F1=0.81) to the originally reported numbers (also F1=0.81), which used a tree-based model architecture. However, when we used the model trained on human-annotated data to predict the tappability of elements in our crawled dataset, we observed significantly degraded performance (F1=0.60), suggesting that the human-annotated and crawler-generated labels disagree with each other. We consider the heuristic-annotated data to be higher quality since its performance was validated by crowdworkers with access to a video clip of the entire tapping interaction, and previous work (Schoop et al., 2022a) has shown predicting element tappability from a single screenshot leads to high variance among raters.

## 5. Discussion

Our experiments revealed that visual UI models could effectively be trained and improved through automated, continual interaction. In this section, we discuss i) the performance of our specific Never-ending UI Learner implementation, ii) other types of interaction-based learning, and iii) the benefits applying these strategies over very long or potentially indefinite period of time

### 5.1. Never-ending UI Learner Performance

In this paper, we conducted experiments that evaluate the Never-ending UI Learner and and its ability to automatically learn tappability. Our experiments investigate two key questions: i) what is the best way for an automated crawler to learn about UIs? and ii) how long would it need to run?

*Crawling Strategy.* Our experiments focused on three crawling strategies for exploring mobile apps: i) randomized crawling, ii) uncertainty sampling, and iii) a hybrid strategy. The random strategy led to the highest final accuracy. We initially hypothesized that uncertainty sampling, an active learning technique that improves sampling efficiency by prioritizing examples with low model confidence, would let the model to learn more efficiently and effectively. However, because our crawler updated its models (which are used to compute the prediction confidences) every epoch

instead of after each sample (as is often done in applications where uncertainty sampling is employed), it led to imbalanced data collection during subsequent crawls, which decreased performance. The hybrid crawler alternated between random and uncertainty sampling strategies, which allowed it learn from low-confidence predictions while also correcting the distribution shift induced by batched uncertainty sampling. Overall, it led to similar performance to pure random crawling.

*Performance over Time.* Even though our crawler is meant to run indefinitely, our experiments focused on a relatively short period of five crawl epochs. Each crawl epoch lasted approximately half a week (clock time) when parallellized across multiple crawler workers and consisted of approximately 500 device-hours of app interaction, data post-processing, and model training. Across all experiments, the Never-ending UI Learner crawled for more than 5,000 device-hours, which was carried out over the span of approximately one month.

Our results show that this window is sufficient to learn an accurate model purely from crawler-collected data. Overall, we found that our model had rapid early learning followed by slower improvement, which is consistent with empirical observations in machine learning research that suggests an exponential relationship between dataset size and model performance (Sun et al., 2017). We believe these small improvements are valuable, since their benefit can be magnified when running over potentially very long periods of time and allow the model to be continuously updated. We plan to continue running the crawler, which doesn't require human supervision, to observe trends over longer periods of time and maximize the potential of our automated learning approach.

## 6. Limitations & Future Work

Our current implementation of a Never-ending UI learner is limited and presents opportunities for future exploration.

First, our current crawler is implemented using a specific set of tools and infrastructure customized for our target platform (iOS). While we did not run experiments on other types of UIs (e.g., Android, web-based interfaces), we expect our results to be generalizable, since our approach does not rely on any platform-specific metadata or APIs, and previous research has shown semantic overlap between mobile and web UIs (Wu et al., 2023). Our experiments primarily focused on free apps that did not require authentication (e.g., registering and making an account), which biased the set of UI screens reached by crawling. We used manually-designed and verified our heuristic for tappability. We believe that many other aspects of UIs and interaction can be formulated using similar methods. A natural question to explore is: what other types of semantics can be learned through interaction? For example, related semantics such as "press-and-hold" functionality can be discovered, and textboxes can be better understood by observing what kind of software keyboard (e.g., email or numeric keyboard) appears when it is tapped on. Could this approach be extended to the problem of UI element detection more generally, which currently relies heavily on human annotation? There are many details that would need to inferred, such as the size and shape of UI elements, and of course the element type. Many more interactions would be needed from the crawler to determine a bounding box for a given element, and it might be difficult to infer complex element types, but a working system that could do this might be able to learn about custom controls and other non-standard elements that current models cannot deal with today. Better automated understanding of UIs can not only benefit downstream applications directly, but also collect better data to train models.

Finally, ever-ending learning also introduces new challenges, like "catastrophic forgetting," the possibility of erasing previously learned information by training on new data, and difficulties associated with large, ever-growing datasets. In this paper, we conduct a preliminary exploration of methods to address some of these challenges, such as uncertainty sampling, which can help prioritize certain types of data. Our literature review uncovered many other possible machine learning techniques that involve training the model training process (Li & Hoiem, 2017; Kirkpatrick et al., 2017; Rebuffi et al., 2017; Castro et al., 2018) or distilling the collected dataset relevant samples (Wang et al., 2018; Nguyen et al., 2020; 2021). We expect that they will be useful for scaling and maximizing the performance of never-ending UI learning.

## 7. Conclusion

In this work, we presented a technique for continuous extraction and modeling of user interface semantics through interactions, which we refer to as "never-ending learning of UIs." We implemented a mobile app crawler that downloads, installs, and crawls thousands of apps to observe UI semantics and affordances in real-world apps, and we use interaction-based heuristics to generate large datasets for training a tappability prediction model, which has previously relied on human-annotate data. We found that our crawler-learned model can be more accurate than those trained from human-annotated screenshots and continue to improve with access to more training examples. The highly automated nature of our approach allows us to apply it indefinitely, with little to no human supervision, which can maximize their performance and utility to downstream applications.

## References


Arsan, D., Zaidi, A., Sagar, A., and Kumar, R. App-based task shortcuts for virtual assistants. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pp. 1089–1099, 2021.

Bai, C., Zang, X., Xu, Y., Sunkara, S., Rastogi, A., Chen, J., and y Arcas, B. A. Uibert: Learning generic multimodal representations for ui understanding. In *International Joint Conference on Artificial Intelligence*, 2021.

Castro, F. M., Marín-Jiménez, M. J., Guil, N., Schmid, C., and Alahari, K. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 233–248, 2018.

Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., and Kumar, R. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pp. 845–854, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349819. doi: 10.1145/3126594.3126651. URL https://doi.org/10.1145/3126594.3126651.

Feiz, S., Wu, J., Zhang, X., Swearngin, A., Barik, T., and Nichols, J. Understanding screen relationships from screenshots of smartphone applications. In *27th International Conference on Intelligent User Interfaces*, IUI '22, pp. 447–458, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391443. doi: 10.1145/3490099.3511109. URL https://doi.org/10.1145/3490099.3511109.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Z., Sunkara, S., Zang, X., Xu, Y., Liu, L., Wichers, N., Schubiner, G., Lee, R., and Chen, J. Actionbert: Leveraging user actions for semantic understanding of user interfaces. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7):5931–5938, May 2021. doi: 10.1609/aaai.v35i7.16741. URL https://ojs.aaai.org/index.php/AAAI/article/view/16741.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

Huang, F., Canny, J. F., and Nichols, J. Swire: Sketch-based user interface retrieval. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pp. 1–10, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300334. URL https://doi.org/10.1145/3290605.3300334.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Kuznetsov, K., Fu, C., Gao, S., Jansen, D. N., Zhang, L., and Zeller, A. Frontmatter: Mining android user interfaces at scale. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, pp. 1580–1584, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385626. doi: 10.1145/3468264.3473125. URL https://doi.org/10.1145/3468264.3473125.

Leiva, L. A., Hota, A., and Oulasvirta, A. Enrico: A dataset for topic modeling of mobile ui designs. In *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '20, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380522. doi: 10.1145/3406324.3410710. URL https://doi.org/10.1145/3406324.3410710.

Li, G., Baechler, G., Tragut, M., and Li, Y. Learning to denoise raw mobile ui layouts for improving datasets at scale. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3502042. URL https://doi.org/10.1145/3491102.3502042.

Li, T. J.-J., Popowski, L., Mitchell, T., and Myers, B. A. Screen2vec: Semantic embedding of gui screens and gui components. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445049. URL https://doi.org/10.1145/3411764.3445049.

Li, Y., He, J., Zhou, X., Zhang, Y., and Baldridge, J. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*, 2020a.

Li, Y., Li, G., He, L., Zheng, J., Li, H., and Guan, Z. 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)*, pp. 5495–5510, 2020b.

Li, Z. and Hoiem, D. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

Liao, Y.-C., Todi, K., Acharya, A., Keurulainen, A., Howes, A., and Oulasvirta, A. Rediscovering affordance: A reinforcement learning perspective. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3501992. URL https://doi.org/10.1145/3491102.3501992.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., and Welling, J. Never-ending learning. *Commun. ACM*, 61(5):103–115, apr 2018. ISSN 0001-0782. doi: 10.1145/3191513. URL https://doi.org/10.1145/3191513.

Nguyen, T., Chen, Z., and Lee, J. Dataset meta-learning from kernel ridge-regression. *arXiv preprint arXiv:2011.00050*, 2020.

Nguyen, T., Novak, R., Xiao, L., and Lee, J. Dataset distillation with infinitely wide convolutional networks. *Advances in Neural Information Processing Systems*, 34: 5186–5198, 2021.

Norman, D. A. *The psychology of everyday things.* Basic books, 1988.

Oulasvirta, A., Jokinen, J. P. P., and Howes, A. Computational rationality as a theory of interaction. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3517739. URL https://doi.org/10.1145/3491102.3517739.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Schoop, E., Zhou, X., Li, G., Chen, Z., Hartmann, B., and Li, Y. Predicting and explaining mobile ui tappability with vision modeling and saliency analysis. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pp. 1–21, 2022a.

Schoop, E., Zhou, X., Li, G., Chen, Z., Hartmann, B., and Li, Y. Predicting and explaining mobile ui tappability with vision modeling and saliency analysis. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022b. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3517497. URL https://doi.org/10.1145/3491102.3517497.

Sereshkeh, A. R., Leung, G., Perumal, K., Phillips, C., Zhang, M., Fazly, A., and Mohomed, I. Vasta: a vision and language-assisted smartphone task automation system. In *Proceedings of the 25th international conference on intelligent user interfaces*, pp. 22–32, 2020.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Swearngin, A. and Li, Y. Modeling mobile interface tappability using crowdsourcing and deep learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pp. 1–11, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300305. URL https://doi.org/10.1145/3290605.3300305.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Thakur, N., Reimers, N., Daxenberger, J., and Gurevych, I. Augmented sbert: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. *arXiv preprint arXiv:2010.08240*, 2020.

Wang, B., Li, G., Zhou, X., Chen, Z., Grossman, T., and Li, Y. Screen2words: Automatic mobile ui summarization with multimodal learning. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pp. 498–510, 2021.

Wang, T., Zhu, J.-Y., Torralba, A., and Efros, A. A. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Wu, J., Zhang, X., Nichols, J., and Bigham, J. P. Screen parsing: Towards reverse engineering of ui models from screenshots. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21, pp. 470–483, New York, NY, USA, 2021. Association for

Computing Machinery. ISBN 9781450386357. doi: 10.1145/3472749.3474763. URL https://doi.org/10.1145/3472749.3474763.

Wu, J., Wang, S., Shen, S., Peng, Y.-H., Nichols, J., and Bigham, J. P. Webui: A dataset for enhancing visual ui understanding with web semantics. *arXiv preprint arXiv:2301.13280*, 2023.

Wu, Z., Jiang, Y., Liu, Y., and Ma, X. 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, pp. 1–13, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080. doi: 10.1145/3313831.3376324. URL https://doi.org/10.1145/3313831.3376324.

Yuan, A. and Li, Y. Modeling human visual search performance on realistic webpages using analytical and deep learning methods. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pp. 1–12, 2020.

Zhang, X., de Greef, L., Swearngin, A., White, S., Murray, K., Yu, L., Shan, Q., Nichols, J., Wu, J., Fleizach, C., et al. Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–15, 2021.

Zhou, X. and Li, Y. Large-scale modeling of mobile user click behaviors using deep learning. In *Proceedings of the 15th ACM Conference on Recommender Systems*, pp. 473–483, 2021.

Zhou, X., Wang, D., and Krähenbühl, P. Objects as points, 2019.