Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning

Xin Zhou
Google Research
Mountain View, CA, USA
zhouxin@google.com

Yang Li
Google Research
Mountain View, CA, USA
liyang@google.com

ABSTRACT

Modeling tap or click sequences of users on a mobile device can improve our understandings of interaction behavior and offers opportunities for UI optimization by recommending next element the user might want to click on. We analyzed a large-scale dataset of over 20 million clicks from more than 4,000 mobile users who opted in. We then designed a deep learning model that predicts the next element that the user clicks given the user's click history, the structural information of the UI screen, and the current context such as the time of the day. We thoroughly investigated the deep model by comparing it with a set of baseline methods based on the dataset. The experiments show that our model achieves 48% and 71% accuracy (top-1 and top-3) for predicting next clicks based on a held-out dataset of test users, which significantly outperformed all the baseline methods with a large margin. We discussed a few scenarios for integrating the model in mobile interaction and how users can potentially benefit from the model.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

User behavior modeling; predictive user interfaces; mobile interaction; click prediction; deep learning.

ACM Reference Format:
Xin Zhou and Yang Li. 2021. Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning. In *Fifteenth ACM Conference on Recommender Systems (RecSys '21), September 27-October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3460231.3474264

1 INTRODUCTION

User interaction behaviors or flows on modern mobile devices mainly consist of a sequence of taps (or clicks[1]) on UI elements presented on the touchscreen, which result in a sequence of screen state transitions in response to these clicks. Modeling user click sequences is important for understanding rich mobile interaction behaviors, e.g., how screens transition and factors that are involved in these transitions [3]. A click-sequence model that predicts which element the user would click next on the touchscreen can create opportunities for improving user experiences in many ways. For example, a mobile system can pre-fetch resources needed based on user action forecasts to reduce response latency (e.g., [15]) or enable adaptive user interfaces (e.g., [5, 20]) by optimizing interfaces to facilitate next user input. In this paper, we focus on the computational modeling of click behaviors that can underpin these domains.

While previous work extensively investigated app-level prediction [3, 8, 10, 15–18, 25], it is relatively sparse on finer-granularity user action prediction. In this paper, we focus on predicting next UI element the user clicks on. Prior work has attempted to address similar problems in various domains, e.g., mobile interaction [9], web surfing [14], advertisement [13], news [23], and shopping [4, 19, 24]. However, previous approaches are inadequate in leveraging rich features that exist in mobile user click-through behaviors and addressing complex interactions between factors that are involved in interaction, which we intend to address in this paper.

There are two major challenges for accurate and scalable click sequence modeling. Firstly, there are a vast number of unique mobile apps today, which come with diverse UI screens and elements. Previous approaches rely on a predefined set of UI elements [9] (similar to word tokens for language models [2]) so as to use conventional sequence modeling techniques such as LSTM [6]. This kind of approach is difficult to scale to address diverse, ever growing UI elements out there. Secondly, user click behaviors can be drastically different, depending on many factors such as each user's usage history and situational factors encountered. Users often switch between apps [3] to carry out a task, which results in sequences spanning multiple apps.

In this paper, we present a novel approach, based on the advance of deep learning techniques [12, 21, 22], for modeling user click sequences at scale. Our approach eliminates the need of using a predefined vocabulary of UI elements and provides an extensible architecture to incorporate a rich set of features, including the user's previous clicks and screens, the structural information of the current screen, and the current time as well as temporal dynamics between click events. We analyzed a large scale dataset of user click behaviors to gain insights for model design. The dataset involves over 20 million clicks from more than 4000 mobile users while using over 13,000 apps in their daily activity. A thorough experiment based on the dataset shows that our model achieves 48% and 71% for top-1 and top-3 accuracy, which significantly outperformed the previous methods and also shows that click prediction is a challenging task. Our approach scales well as it does not involve complex feature engineering. We discuss interaction scenarios where our

[1]We use "tap" and "click" interchangeably in this paper.

RecSys '21, September 27-October 1, 2021, Amsterdam, Netherlands

ACM ISBN 978-1-4503-8458-2/21/09.
https://doi.org/10.1145/3460231.3474264

model can help and design considerations for integrating the model in an interaction flow. We share our insights into the model behavior and limitations, and plans for future work. The paper makes the following contributions.

- An analysis of a large-scale dataset of mobile user click sequences that reveals rich factors and complexity in modeling click behaviors, which contributes new knowledge to understand mobile interaction behaviors.
- A Transformer-based deep model that predicts next element to click based on the user click history and the current screen and time. The model does not rely on a vocabulary of predefined UI elements and provides a general solution for modeling arbitrary UI elements for click prediction.
- A thorough experiment that compares our deep model with multiple alternative designs and baseline models, and an analysis of model behaviors and benefits that the model can bring to improve mobile interaction.

2 RELATED WORK

Previous work has investigated mobile user behaviors based on interaction logs. Böhmer et al. discovered that multiple factors can impact the patterns of mobile app usage [3]. For example, the category of a mobile app and the user context such as the current time and location can influence the usage of mobile applications. Previous work also pointed out the average session with an app lasts less than a minute, even through users may spend a long time using their phones. These findings are aligned with our analyses based on a large-scale dataset.

Previous work has extensively investigated approaches for modeling app usage and predicting next app that are likely to be used, e.g., [1, 15, 18]. Particularly, prior work [7, 25] revealed that time signals such as the hour of the day and the day of the week as well as recent usage history are essential for predicting next app usage. Choonsung et al. [18] developed various ways for integrating prediction models into the mobile phone homescreen that greatly reduced search time for apps. Li proposed app prediction as a general service on mobile devices [10]. App prediction has become available on a variety of mobile platforms that is frequently in use by mobile users for daily app launches.

Compared to app prediction, there is relatively less prior work on finer-granularity user action prediction especially for mobile interaction. Deep learning has been broadly used in various domains, which has shown promise in addressing complex problems without extensive feature engineering. For interaction behavior modeling, sequence models such as LSTMs (e.g., [9, 11]) have been extensively used because of their extensive architecture and the ability to capture long-range dependencies. Convolutional neural networks have also been used for modeling sequences of multiple recommendation tasks [13, 19]. Transformer [21] is a more recent model for sequence modeling that has produced state-of-art results on various tasks. We next focus on recent work that are particularly relevant to our work.

Lee et al. developed a model to predict the element that the user is likely to click on the current screen based on the user's previous click behaviors [9]. To use conventional sequence modeling techniques such as LSTMs [6], the prior work treated user click behaviors as a sequence of tokens where each token is derived from a UI element, which requires a predefined vocabulary of elements. While an important contribution of the prior work was the mechanism for identifying an element to define the vocabulary, this approach is fundamentally limited because mobile apps are fast growing and their UIs are quickly evolving. It is impractical to identify every element in an ever growing collection of UIs. In addition, the previous work only models click sequences within each app and does not handle cross-app transitions where the latter constitute 26% of click behavior based our data analysis. In our work, we deliberately address the representation challenge of UI elements. Without requiring a predefined vocabulary of elements, rather we use neural pointers and contextual representations of elements. Our model is also designed to easily leverage a richer set of features than treating each click as a token, and naturally handle both within and cross-app transitions.

Transformer is employed in previous works like Chen et al. for shopping recommendation [4], and Wu et al. in news recommendation [23]. Their model takes input of embeddings of previously clicked items and a candidate next item, and outputs a score to rank each candidate. Our model, which is also based on Transformer, has several critical differences. First, our model uses a different set of features for click prediction across arbitrary apps, instead of a single app that the previous work focused on. Second, we use a hierarchical Transformer to model both in the context of a screen and a click sequence. Third, we use neural pointer to find next item instead of using a ranking model (i.e., sigmoid output in [4] or inner product in [23]). We extensively experimented with our model in comparison with alternative methods including these deep models. Lastly, in contrast to previous work (e.g., [5, 20]) on human factors and design options for adaptive UIs, we concentrate on computational modeling of user click behaviors, which provides a predictive model basis for adaptive UIs.

3 UNDERSTANDING MOBILE CLICK BEHAVIORS

To develop computational models for predicting next element to tap, we first analyze a large-scale dataset of mobile click behaviors and gain insights into these behaviors. The analyses of the dataset inform the design of our deep model.

3.1 Data Processing

User volunteers were sampled from a consented research panel to be representative of the mobile user population and opted in to participate in multi-week study. The interaction events were collected during their regular usage of device through additional software that captured data from app UIs.

Each event consists of the user input (e.g., a `Click` versus a `Swipe`), the UI element that the input is applied to as well as a structural representation of the entire screen where the event takes place (i.e., the View Hierarchy[2]).

A UI element is considered *actionable* only if its following attributes in the view hierarchy are set to true: `clickable`, `visible`, and `enable`. For each UI element, we extract its text content, type,

[2]https://developer.android.com/reference/android/view/View.html

and bounding box positions on the screen from the view hierarchy. These properties will be used for featurizing the element later for modeling. We extract the text content of an element by looking at its `text` property first. If it is absent or empty, we use its `content-desc` attribute. The last resort is to extract text content from the `resource-id` property of the element.

The click events of each user are sorted chronologically as a sequence. Each event consists of the screen view hierarchy that the click occurs, the element on the screen that is being clicked and the time of the event. We filtered sequences of user participants who had outlier behaviors, such as sporadic usage of their devices or a large number of clicks produced in a short period of time. As a result, the dataset consists of over 20 million clicks, which form click sequences from more than 4,000 Android users using over 13,000 unique apps on their smartphones.

3.2 Data Analysis

3.2.1 Sequence Lengths & Time Spans. The length of these sequences varies from 100 to 25,000 clicks with the time span of each sequence ranging from 2 to 6 weeks (Mean=4.12 and Std=0.62). One factor that contributed to the variance in click numbers is the difference between power users versus novice users, which ranges from 19 clicks to over 5000 clicks per week (Median 833). These results showed that the dataset covers a wide range of user click behaviors that provide a solid basis for developing and evaluating machine learning models.

3.2.2 UI Complexity. The dataset involves a diverse set of UI elements with 24 types (see Figure 1). Overall, for all the actionable UI elements that we analyzed, the number of clicks that each type of element received is well aligned with the total number of elements of each type that exist in the UI screens in the entire dataset, with a few exceptions. For example, `TextView` received relatively few clicks and `TabWidget` received none, which is based on the usage of the set of user participants during the period of data collection.

There are often a large number of actionable elements on each screen in this dataset: Mean=18, Std=12 (see Figure 2). The number of actionable elements on a screen determines how challenging it is for a model to predict the next element to be clicked by the user from these candidates.

3.2.3 Temporal Patterns. The time analysis of click events involves two aspects: one is the temporal dynamics between events, and the other is the temporal regularity. For the temporal dynamics, we analyzed the time intervals between events (see Figure 3). We can see most event transitions have a short time span. Intuitively, the shorter a time span is for an transition, the more relevant the two adjacent events are. For analyzing the temporal regularity, we computed the hour of the day and the day of the week when an event occurred using both the raw timestamp and the timezone information of an event. Figure 4 shows how events are distributed across hours of a day and days of a week. As expected, there are fewer events on the weekends than on the weekdays. Across hours of a day, there is a heavier usage of phones in the late afternoons and evenings. These results show that the dataset sufficiently covers users' daily usage of mobile devices. The example in Figure 5 shows that click behaviors can be highly time-dependent. We intend to use both temporal signals in our deep model.

3.2.4 Click Transitions. A click event triggers the transition from one screen to another, which can be a popup menus overlaying the current screen or a completely new screen. About 30% of these events were generated from popular apps such as Chrome, SMS and YouTube. However, there are a vast number of long-tail apps that play a vital role in users' daily mobile usage. While many of these transitions occur within an app, i.e., the current screen and the next screen after an click both belong to the same app, 26% of transitions do occur across apps, (see Figure 6 for an example). This observation is aligned with previous findings [3] that users tends to transition across apps frequently. This implies that it is important for the model to be able to handle cross-app transitions.

4 MODELING CLICK BEHAVIORS

With the analytical understanding of mobile user click behaviors, we first formulate our modeling task, and then describe the design of our deep model.

4.1 Problem Formulation

As stated in the previous section, a click event consists of a tuple of attributes, including the screen, the element being clicked on the screen, the time that the event occurs, and the app that the screen belongs to. The ith event in a click sequence is formulated as the following (see Equation 1).

$$e_i = [s_i, c_i, t_i, a_i] \tag{1}$$

where s_i is the screen that contains a collection of UI elements $o_{i,j} \in s_i$ where $1 \leq j \leq |s_i|$ and $|s_i|$ represents the number of elements in the screen s_i. c_i denotes the index position of the element being clicked on the screen, in the pre-order traversal of the view hierarchy tree (sibling elements are spatially ordered). Consequently, the element being clicked is represented as the following.

$$o_{i,c_i} = \text{Preorder\_Traversal}(s_i)[c_i] \tag{2}$$

t_i represents the temporal information of the event that consists both temporal regularity, i.e., the hour of the day, r_i, and the day of the week, w_i, and temporal dynamics, i,e., how distant the event is with respect to the prediction time, denoted as v_i. a_i identifies the app that the screen belongs to. A click sequence of a user is thus represented as a sequence of these tuples.

$$e_{1:n} = \{(s_1, c_1, t_1, a_1), (s_2, c_2, t_2, a_2), ..., (s_n, c_n, t_n, a_n)\}$$

So the click prediction problem is defined as the following.

$$c_{n+1} = f(s_{n+1}, r_{n+1}, w_{n+1}, a_{n+1}, e_{1:n}) \tag{3}$$

Given a sequence of previous events $e_{1:n}$ and the current context (i.e., the current screen s_{n+1}, the current hour and day r_{n+1} and w_{n+1}, the current app a_{n+1}), we want to predict which element on the screen, c_{n+1}, is likely to be clicked by the user next, i.e., at the $(i+1)$th step. In this work, we intend to use a deep model to realize function f.

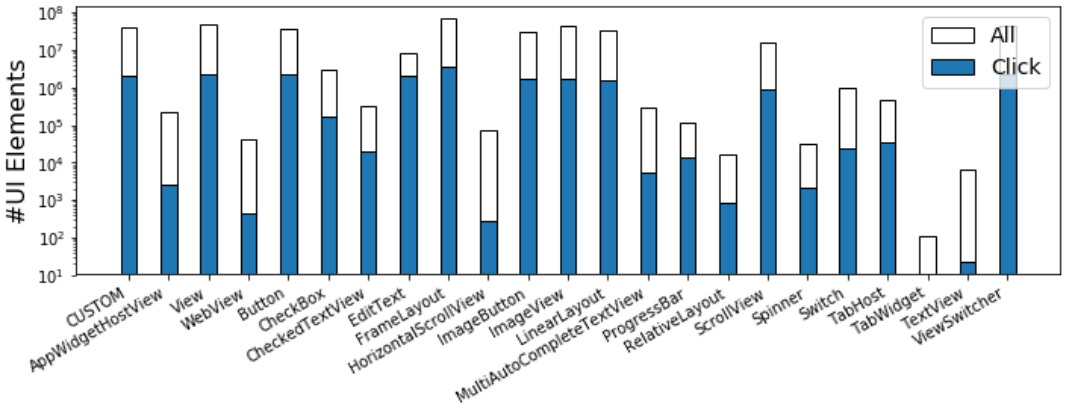


Figure 1: The distribution of the total number of elements versus the clicked ones for each UI type, based on the dataset. The count of UI elements (the Y axis) is displayed at the *Log* scale.

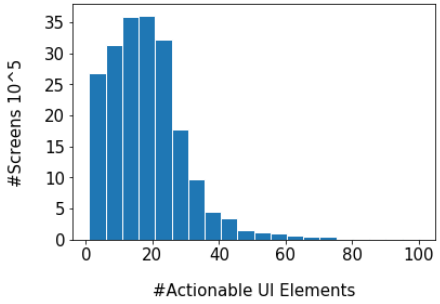


Figure 2: The number of actionable elements on each screen.

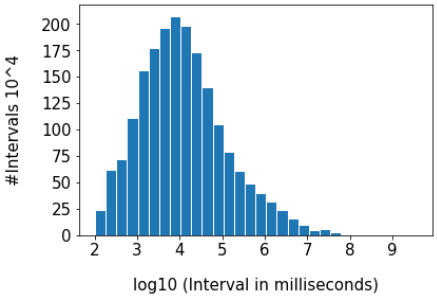


Figure 3: The distribution of time intervals between click events.

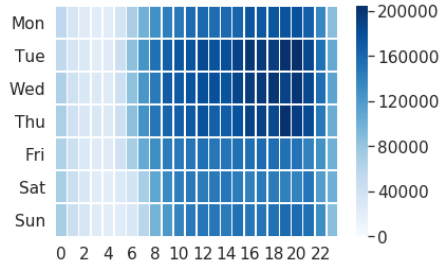


Figure 4: The distribution of click events based on the day of a week and the hour of a day when an event occurred.

4.2 The Design of the Deep Model

There are three tasks for designing the prediction model. First, we need to derive a semantic representation of the clicked element of each event in the context of its screen. Second, we need to model a sequence of click events so as to encode the user's history. Last, based on the encoding of previous events and the current context, we want to design the model to select the element that is most likely to be clicked from all the actionable elements on the current screen. We design our deep model as hierarchical transformers coupled with neural pointers (see Figure 7).

4.2.1 Encoding a UI Element in the Context of its Screen. We use Transformer [21], a neural attention-based model that has led to state-of-the-art results for many problems. Specifically, we use the Transformer Encoder to encode each element in the screen in a similar way to previous work [12] (see Equation 4). The self-attention in Transformer Encoder allows each element to be represented in the context of all the elements on the screen (see [21] for more details about self attention).

$$h_{i,1:|s_i|} = \text{Transformer\_Encoder}(\text{Embed}^o(o_{i,1:|s_i|}), \theta) \quad (4)$$

θ is the trainable parameters. For all the elements on screen s_i, we first compute the embedding of each element, $\text{Embed}(o_{i,1:|s_i|})$, based on its text content, the type attribute, and the bounding box (position and size) on the screen. The text content and the type attribute form the content embedding of an element while the bounding box positions form the positional embedding of the element.

Each word in the text content of an element is represented via a regular word embedding, and the text content of an object, which can contain more than one word, is represented as the average embedding of all the words, similar to a bag of words. The type is a categorical value and also represented as an embedding vector, which is combined with the text embedding to form the content embedding of the element.

The positional embedding of an element is computed based on the bounding box positions of the element: `[left, top, right, bottom]`. We simply normalized each coordinate value to an integer in the range of [0, 100), which is then represented as an trainable embedding vector. The sum of positional and content embeddings forms the input of the Transformer Encoder model.

The embeddings of all the elements on the screen are then fed to a Transformer Encoder (Equation 4), which results in a latent fixed-dimensional representation for each element, $h_{i,j} \in R^d$ where

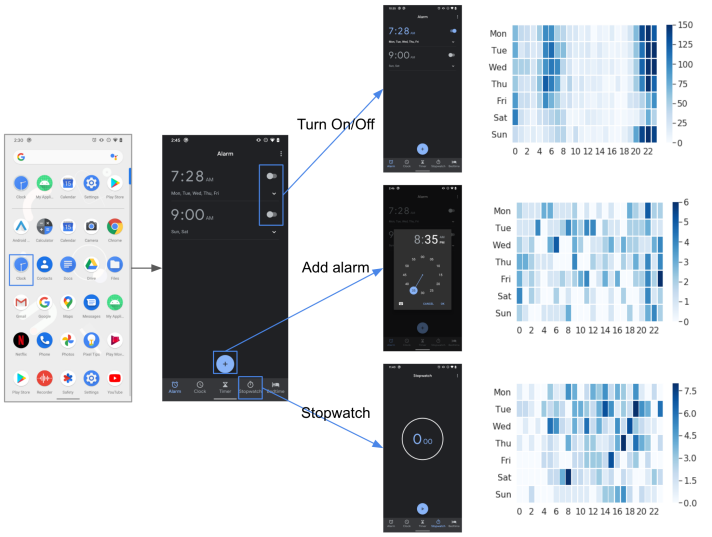


Figure 5: The click-transition sequences show that a user clicked on different elements of the Clock screen in a different hour and day. The toggle buttons of alarms are mostly used in the nights Sunday through Thursday, when the following day is a workday. But the stopwatch is used evenly through the week.

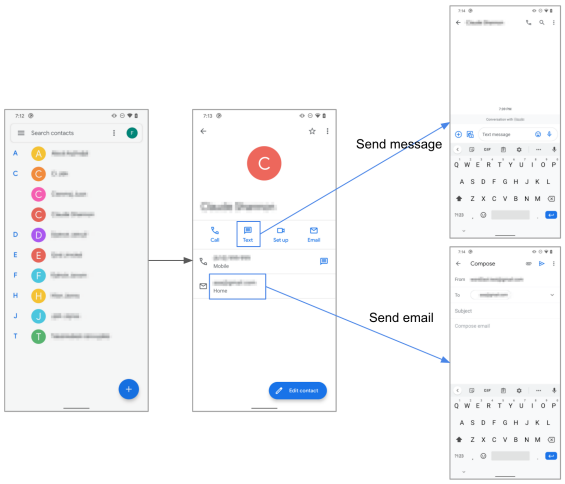


Figure 6: An example of cross-apps click transitions. The "text" and "email" elements in Contacts will lead the user to the Messaging app and the Gmail app respectively.

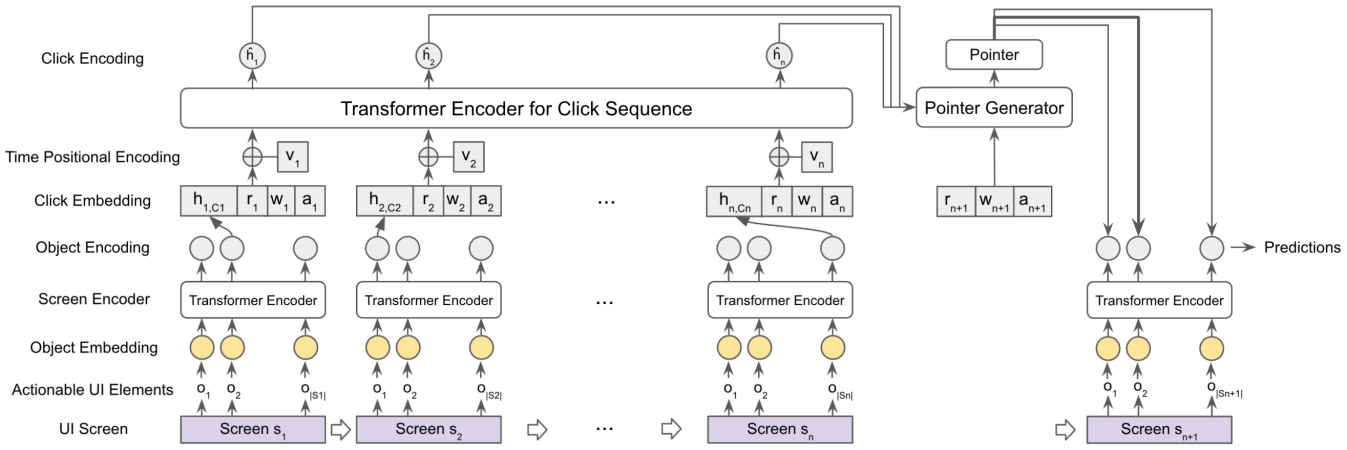


Figure 7: The architecture for our deep model for predicting the element to be clicked on a given screen. App IDs a_i are optional in our model, and our model is app agnostic and provides a general solution for modeling clicks from arbitrary UIs.

$1 \leq j \leq |s_i|$ as defined earlier, and d is the depth of the latent vector. In particular, h_{i,c_i} contextually represents the UI element being clicked on at step i. Such a representation of a clicked element is contextual because the self attention in Transformer Encoder learns to relate an element to all other elements on the screen based on both their content similarity and spatial adjacency.

4.2.2 Encoding a Click Sequence. With each clicked element semantically represented as h_{i,c_i}, we next describe how we model a sequence (history) of click events, $e_{1:n}$ (see Equation 1). For this task, we again design our model using Transformer Encoder.

As revealed in our data analysis, the temporal information, t_i, is an important factor for mobile click behaviors, and there are two aspects to it: temporal regularity and dynamics. For temporal regularity features, i.e., the hour of the day, r_i, and the day of the week, w_i, of an event, we represent them as a one-hot vector of size 7 (for 7 days of a week), and 24 (for 24 hours of a day) respectively, from which we acquire a trainable embedding vector for each feature. Similarly, we represent the app the screen belongs to, a_i, via an embedding vector as well. Along the clicked element representation, h_{i,c_i}, we form the embedding of a click event as the following (Equation 5).

$$c_i^E = [h_{i,c_i}; \text{Embed}^r(r_i); \text{Embed}^w(w_i); \text{Embed}^a(a_i)]W_c \quad (5)$$

We first concatenate all these embedding vectors with the clicked element encoding. The concatenation is then linearly projected via trainable parameters, W_c, to a dimension d, which results in the click event embedding, $c_i^E \in R^d$.

For temporal dynamics, specifically the time interval between an event in the past and the prediction time, v_i, we use the information to derive the positional encoding of each event in the history sequence. Notice that this differs from the positional encoding schema in the original Transformer case where events are evenly distributed along the time dimension. As shown in Figure 3, the time interval between click events can vary drastically. The temporal positional encoding of an event should capture how recent the event is for the current step. We compute this elapsed time-based positional encoding as the following.

$$v_i^E = \text{Embed}^v(\text{Floor}(log(v_i))) \quad (6)$$

where we first take the logarithm of v_i, the elapsed time from the ith step to the current $(n+1)$th step, and then discretize it using its floor value. This again is represented using a vector through the learnable embedding function Embed^v. Finally, we combine both the click event embedding (Equation 5) and its time positional encoding to form the input to the Transformer Encoder by adding them: $e_i^E = c_i^E + v_i^E$. We then feed the event embeddings to another Transformer Encoder to generate the latent representation of each click event, $\hat{h}_{i,1:|s_i|}$. β is trainable parameters in the Transformer Encoder model.

$$\hat{h}_{i,1:|s_i|} = \text{Transformer\_Encoder}(e^E_{i,1:|s_i|}, \beta) \quad (7)$$

4.2.3 Predicting Next Element via Neural Pointer. With the click history represented, we now discuss how we can find the element that is most likely to be clicked on the current screen, s_{n+1}, given the current hour and day r_{n+1} and w_{n+1} as well as the app the screen belongs to, a_{n+1}. First of all, we compute a contextual representation of each element on the current screen, $h_{n+1,1:|s_{n+1}|}$, which is the same way as how we compute the representation of each element in the previous screens (see Equation 4). We then compute a "pointer" to point into the current screen—that evaluates how likely each element would be clicked next. The "pointer" is realized via a M-layer perceptron that takes the current hour and day, r_{n+1} and w_{n+1}, as well as the app, a_{n+1}, as the input, and at the same time attends to the latent representation of each event in the history, $\hat{h}_{i,1:|s_i|}$, via multi-head attention [21].

$$q_{n+1}^0 = [\text{Embed}^r(r_{n+1}); \text{Embed}^w(w_{n+1}); \text{Embed}^a(a_{n+1})] \quad (8)$$

$$q_{n+1}^{m+1} = \text{MultiHead\_Attention}(q_{n+1}^m, \hat{h}_{i,1:|s_i|}), \theta^q) \quad (9)$$

where m denotes the layer index and $0 \le m \le M$. The final output of the pointer generator is thus q_{n+1}^M. We then compute how well the pointer vector q_{n+1}^M matches the latent representation vector of each element on the current screen using a typical neural alignment mechanism.

$$\alpha_{n+1,j} = q_{n+1}^M W^q \cdot h_{n+1,j} \quad (10)$$

where we first linearly project the pointer vector q_{n+1}^M via W^q (learnable parameters), and then perform dot product between the linear projection of the pointer and the latent representation vector of each element. The alignment score $\alpha_{n+1,j}$ is a scalar value, which can then be used to calculate the probability of each element to be clicked using a softmax.

$$\text{Prob}_j = \frac{\exp(\alpha_{n+1,j})}{\sum_{1 \le k \le |s_{n+1}|} \exp(\alpha_{n+1,k})} \quad (11)$$

With Equation 11, we can find the element that is most likely to be clicked, i.e., $c_{n+1} = \text{argmax}_j Prob_j$, which realizes the Equation 3. The entire model can be trained end to end by minimizing the cross entropy loss between the probability distribution of next element to click (Equation 11), and the element that is actually being clicked on the screen—the groundtruth.

Note that our model does not rely on app specific information, except the app identifier a_i. As we will discuss later, our experiments revealed that using the app identifier, a_i, does not significantly improve our accuracy. As a result, the design of our entire model can be app independent, which means that it can handle UIs from arbitrary, unseen apps.

5 EXPERIMENTS

To evaluate the accuracy of our model, we conducted a thorough experiment by comparing our model with multiple baselines based on a set of evaluation metrics.

5.1 Dataset

We randomly split the dataset into 80% of the sequences for training, 10% for validation to tune hyper parameters of the models and 10% for testing. Several baseline methods that we compare with require each screen to be uniquely identified via a static ID. To this end, we compute a hash ID for each screen. To generate a hash ID for a screen, we concatenate the features of each UI element sequentially according to the preorder traversal of the screen's view hierarchy tree. We then map the concatenation to a hash signature using the MurmurHash3 algorithm[3].

5.2 Baselines

We compare our model with 9 alternative methods in our experiment. 3 of these methods are heuristics-based, which offer a baseline for understanding the difficulty of the task. 3 other methods are based on classic ML methods that have been widely used in the recommendation/prediction tasks. We also included 3 recent methods that used deep learning models for click prediction.

5.2.1 Heuristic Methods. Recency is a strong heuristics that has been constantly used in commercial products, e.g., recently used apps or recent calls. We implemented recency to predict next element to click according to the recently clicked elements on the screen. To do so, each unique screen needs to maintain a history of previous clicks. *Frequency* is another commonly used heuristics in suggesting user actions. The method predicts the element to be clicked based on how often the element has been clicked on in the

[3]https://guava.dev/releases/snapshot/api/docs/com/google/common/hash/Hashing.html#murmur3_128--

past. There are two design options: one based on *personal frequency* that is specific to the user, the other using the *global frequency* based on the aggregated frequency across users.

5.2.2 Traditional Machine Learning Methods. For this category of methods, we experiment with *Logistic Regression*, *SVM*, and *Naive Bayesian*. These methods have been commonplaces for machine learning models before deep models became mainstream. They have been widely used in the HCI field. One challenge to apply these methods in our problem is that they cannot easily accommodate variable-length input, which is the case for both click history and the number of objects on the screen. In our experiment, we pair the previously clicked element o_{n,c_n} with each candidate UI element on the current screen $o_{n+1,1:m}$. We featurize an element by concatenating the one-hot vector of each of these UI features: text, type, left, top, right, bottom, day-of-week, hour-of-day, and app name. We then concatenate the feature representation of both the previous and the candidate element along the one-hot encoding of the current hour and day. This forms $x_{n+1,j}$, where $1 \le j \le |s_i|$, a long binary vector that is the input to, Φ, either a Logistic Regression (LR), a Naive Bayesian (NB), or a SVM. The target function, Φ^*, that we want to learn for each model is the following, where N is 0 for LR and NB and -1 for SVM.

$$\Phi^*(x_{n+1,j}) = \begin{cases} 1, & \text{if } j = c_i \\ \text{N}, & \text{otherwise} \end{cases}$$

5.2.3 Recent Deep Learning Methods. We compared our method with three recent approaches that used deep learning methods for click prediction. Liu et al [13] leverage CNN in click prediction. Embedding of elements in history compose a 2D array then apply 1-D row-wise convolution, a flexible p-max pooling, and tanh as activation function. We apply a similar approach on our dataset. Lee et al. [9] employed *LSTM* [6] to model mobile click sequences. A critical step in the previous approach is to tokenize each element by identifying the element using its features. Based on the previous paper [9], we tried our best to replicate the approach in our experiment. Each UI element is represented as a string identifier by concatenating its features such as the text content, type (as a string), and the name of the app the element belongs to. The string identifier is then treated as a token and mapped to an integer ID. With each element tokenized and represented an integer ID, it is straightforward to apply an LSTM to modeling a sequence of tokens. In addition, we compare our method with the approach introduced by Chen et al. [4] that used a Transformer-based model for user click prediction in a shopping app. It has several critical differences with our method as we have elaborated in the Related Work section. We replicate this previous method for next click prediction in our context.

5.3 Evaluation Metrics

We compute multiple metrics to evaluate the prediction quality of each method.

Top-K Accuracy. Similar to a typical model evaluation setting, we compute the top-K accuracy based on how often the target element is within the top-K predictions of a model. Particularly, we report *top-1* and *top-3* accuracy of each model in the experiments. For these measure, the larger the better.

Absolute & Relative Ranking Position. Another important indicator of prediction quality is the ranking position of the target item, referred as *Absolute Ranking*. As we will discuss later, for accessibility scenarios such as Android Switch access[4], which requires a user to sequentially iterate over candidate items, the ranking position would greatly affect user experience. However, Absolute Ranking can be misleading as the difficulty to predict a target item varies based on the number of UI elements on the screen (see Figure 2). It is easier for a model to acquire a smaller Absolute Ranking on screens with few elements than on those with many elements. So we propose another measure *Relative Ranking* by normalizing Absolute Ranking by the total number of elements on the screen, which is in the range of $[0, 1)$. Relative Ranking for a *random* model is expected to be 0.5. For both ranking measures, the smaller the better.

5.4 Model Configurations and Training

We implemented all the models using Python and TensorFlow[5]. For Logistic Regression, SVM or Naive Bayes, we randomly sample a pair of adjacent screens from a click sequence in the training data to feed to the model. The LR model is trained by minimizing the cross entropy loss using GradientDescentOptimizer in TensorFlow. The SVM model uses SGDRegressor and the Naive Bayesian model uses BernoulliNB of the sklearn library[6], which provides `partial_fit()` for training on a large-scale dataset in batches. We observed that positive and negative examples are unbalanced because each screen pair only produces one positive example, since only one element is clicked, but $|s_{n+1}| - 1$ negative examples. To address this issue, we gave a larger weight (5 in our experiments) to positive examples during training.

Both the CNN baseline, LSTM baseline and our Transformer-based models take variable-length click history as the input to the model. Because the length of a sequence can vary drastically, it is not memory efficient to batch train on these sequences directly, which incurs a large number of padding. To address this issue, for the training data, we split each sequence into segments of a fixed size of 100. We used a batch size of 128 and a hidden size of 128 for these models. We also tune the hyper parameters of these models such as the number of layers and the learning rate. We chose to use 2 layers for Transformer Encoder models as well as our pointer generator model (Equation 9). Adding more layers or using a larger hidden size does not significantly improve the accuracy but slows down training. All the neural models including Transformer are trained using the Adam optimizer, with a variable learning rate involving a linear warm-up followed by a exponential decay. We used a dropout ratio of 10%. Each deep model was trained to convergence on a single Nvidia Tesla V100 GPU with 80GB memory, which took roughly 1 day to complete.

5.5 Test Results & Analysis

5.5.1 Experimental Results. We evaluate each model on the test dataset, by testing on every click event in each test sequence. The results are shown in Table 1. Overall, our models outperformed

[4]https://support.google.com/accessibility/android/answer/6122836
[5]https://www.tensorflow.org
[6]https://scikit-learn.org/stable

Model	Top-1 Accuracy	Top-3 Accuracy	Absolute Ranking	Relative Ranking
Recency	.2167	.3573	7.732	.4023
Frequency	.2314	.3684	7.648	.3970
Global Frequency	.1179	.2633	8.777	.4667
Logistic Regression	.2704	.5327	4.839	.2514
SVM	.2196	.3925	7.250	.3684
Naive Bayes	.2599	.4905	4.916	.2503
LSTM	.3453	.5402	5.099	.2663
CNN [Liu et al.]	.3847	.6562	3.441	.1777
Transformer [Chen et al.]	.3761	.6463	3.404	.1780
Our Model	**.4828**	**.7140**	**2.607**	**.1397**
Our Model (all features)	.4817	.7119	2.630	.1407
Our Model (without text)	.2865	.5370	4.447	.2370
Our Model (without type)	.4777	.7087	2.673	.1428
Our Model (without position)	.4828	.7140	2.607	.1397
Our Model (without time)	.4711	.7037	2.728	.1452
Our Model (without app name)	.4780	.7077	2.682	.1431
Our Model (without in-screen attn)	.4373	.6747	3.028	.1560

Table 1: The accuracy of each model on the four metrics. For Top-1 and Top-3 accuracy, the larger the better, and for Absolute and Relative Ranking, the smaller the better. The results of the champion model is highlighted in bold.

all the baseline methods with a significant margin across all the metrics. Our model predicts the target element as the top choice 48% of time and, within the top-3 predictions 71% of time. The ranking position is improved from above 7 of heuristic methods to less than 3.

The method using personal click frequency (Frequency) is consistently better than the one using click frequency aggregated across users (Global Frequency), which indicates that click behaviors are often personal. Recency and Frequency acquired a similar accuracy across the metrics. Machine learning-based methods generally outperformed heuristics-based ones, and deep models such as LSTM and Transformer-based approaches further outperformed traditional methods including Logistic Regression, SVM, and Naive Bayes.

Compared with the previous Transformer-based method [4], our model shows improvement across all the metrics. These results indicate that screen encoding enhances the learning of user behavior—Instead of only considering the clicked items, all the UI elements in the context contribute to the prediction. To further compare our model with this previous method [4], we split the dataset by time that was done in [4]: for each user, the first 80% of the sequence is used for training, the next 10% for evaluation and the last 10% for testing. Splitting by time yields slightly better performance for both methods than splitting by users, because a model has an opportunity to learn specific behavior of each individual user. Our model achieves 51% and 74% for top-1 and top-3 accuracy and 2.38 and 0.12 for absolute and relative rankings. Chen et al.'s model achieves 42% and 68% for top-1 and top-3 accuracy and 3.18 and 0.14 for absolute and relative rankings. Our model still substantially outperformed the previous method.

5.5.2 Model Behavior Analysis. As expected, the more elements there are on a screen the more difficult it is for a model to achieve good accuracy. As shown in Figure 8 , the top-K accuracy decreases and the ranking position is worsen as the number of elements increases on the screen. For UIs with at most 10 elements (28% of screens), our model achieves 61% and 90% for top-1 and top-3 accuracy. The Absolute Ranking grows as the number of elements on the screen increases. However, the Relative Ranking remains stable regardless of the screen complexity.

To understand how the model accuracy varies across different apps, we analyzed the top-1 and top-3 accuracy for the top 20 popular apps in the dataset (see Figure 9). The dashed lines show the average of all apps. These 20 apps contributes 56% of all the screens in the dataset. 13 of them show better accuracy on top-1 than the average and 16 for top-3, which indicates that model performs better on popular apps. Among these apps, the model performed the worst for the Calculator app, which is expected because the user can enter arbitrary sequences (e.g., a multi-digit number) in the Calculator that is hardly predictable.

In 26% cross-app scenarios, i.e. $a_{n+1} \neq a_n$, the model achieves 49.6% and 73.1% in top-1 and top-3 accuracy, which is slightly better than the overall accuracy. This result shows that the model has robust performance in both in-app and cross-app scenarios. There's still room for improvement in prediction of in-app categories.

To further understand the behavior of our model, we look into how our model predicts elements on specific screens. Our testing dataset contains nearly 2 millions screens, and Figure 11 shows the behavior of our model with respect to a specific screen that occurs 105 times. The screen has 12 UI elements, and 6 of these elements were clicked by users on different days and at different hours. These clicks are visualized by the heatmap in Figure 11. As shown in the table on the right, the user clicked on different UI elements at different or the same times and our model is able to handle the variability of user behavior reasonably well. The Top-1 and Top-3 accuracy for this screen is 74% and 97% respectively. By

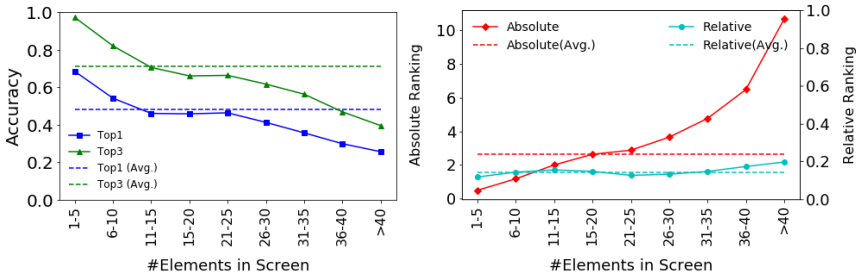


Figure 8: Top-1 and Top-3 accuracy decrease as the number of elements on the screen increases. The Absolute Ranking increases as the number of elements on the screen increases while the Relative Ranking remains stable.

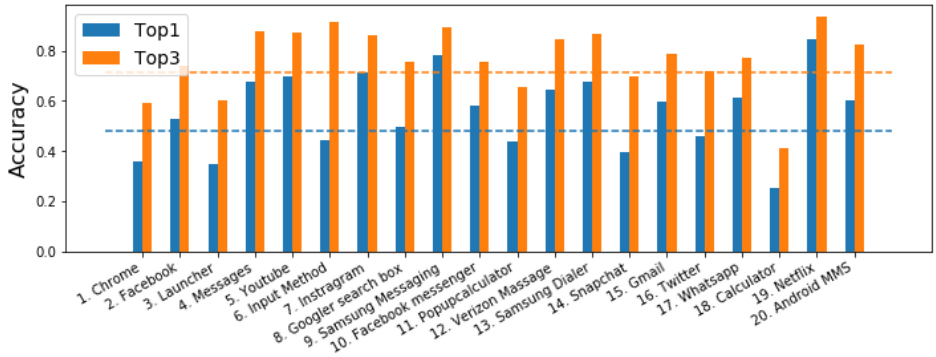


Figure 9: Top-1 and Top-3 accuracy for the top 20 popular apps in the dataset.

breaking down the prediction by elements, we observe that obj_2 was clicked for 13 times and our model gives perfect prediction for all of these cases. For obj_5, our model outputs the correct prediction as the top-1 choice for 3 times out of 4 cases in total, and is able to cover all 4 cases within the top-3 predictions.

5.6 Ablation Study

We seek to understand how each UI feature and context affects the model performances by building variants that: 1) exclude one feature for each, 2) access different lengths of history as model input.

We experiment model variants by picking out each of these features: element text, element type, element position, time, app name. The results (at bottom part of table 1) shows that most features contribute to the performance, in which element text makes the biggest difference. Element position helps very little in our experiment on the validation dataset, and even decreases performance in the testing dataset, as shown in the table. A suspicion is that various devices and app versions cause UI layout differences, thus providing few info for model learning. We also test a variant that uses *clicked* element embedding directly without the Screen Encoder (see Figure 7), the decreased performance shows that attention to other unclicked elements on screen do help in understanding the user click behavior.

The performance of the models varies by history size. The prediction is less effective without any history (as seen in Figure 10. Access of recent screens significantly improves the performance for all metrics, while longer history provides more improvement in performance. We choose 9 as the history size for our model by the memory limit.

6 DISCUSSION

We have deployed our model to Android devices using TF-Lite[7], and the optimized model size is 27MB, which is within a good range for serving on a mobile device. As a proof of concept, we created a prototype feature named *Next Click Overlay* that presents the UI element that is mostly likely to be clicked at the bottom of the screen. This design does not alter the layout of an existing interface, and introduces a small amount of cognitive overload for the user to glance over the predicted item. If the prediction is correct, the user can reach the next click single-handedly. For screens that the model tends to have low accuracy, e.g., Calculator, the overlay will not been shown.

While the feature is targeted for a general mobile interaction scenario, it is particularly relevant to accessibility. Existing accessibility services such as Android Switch Access[8] allows a user with dexterity impairments to "click" on a target by iterating through the actionable elements on the UI one by one via an external device. The traversal is typically based on the spatial ordering of the elements on the screen in a top-down and left-right manner and the

[7]https://www.tensorflow.org/lite
[8]https://support.google.com/accessibility/android/answer/6122836

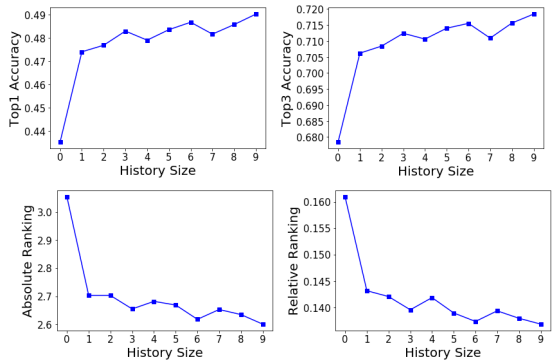
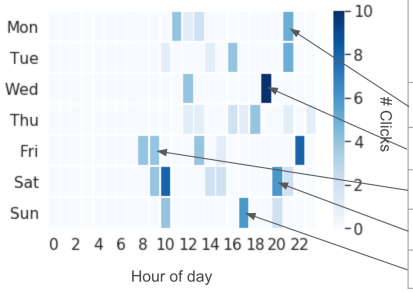


Figure 10: Performance of our Transformer model variants with different history size in model input. The model with access to the most recent screen (history_size=1) performs 4% better in top-1 accuracy than without history info (history_size=0), and other metrics have similar increasement.



		#Clicks	# In Top1	# In Top3
All	obj_2	13	13	13
	obj_3	11	0	11
	obj_4	9	0	6
	obj_5	4	3	4
	obj_8	58	58	58
	obj_10	10	4	10
Mon 9 PM	obj_2	1	1	1
	obj_4	4	4	4
Wed 7 PM	obj_8	9	9	9
	obj_10	1	0	1
Fri 9 AM	obj_5	4	3	4
Sat 8 PM	obj_3	2	0	2
	obj_10	4	3	4
Sun 5 PM	obj_8	6	6	6

Figure 11: An analysis of click behaviors on a specific test screen that contains 12 UI elements. 6 of them were clicked by users: obj_2, obj_3, obj_4, obj_5, obj_8, and obj_10, which account for a total of 105 clicks on different days and hours. The heatmap on the left visualizes the distribution of these clicks across days and hours, with the color intensity corresponding the number of clicks. The table on the right elaborates on a few dense spots.

average number of traversal needed is 9.04 based on our dataset. With the Next Click Overlay, the number can be potentially reduced to 2.61. The estimated gain can be validated via a user study, which is beyond the scope of the paper.

7 CONCLUSION

We presented a novel approach for modeling mobile user click behaviors using deep learning. We analyzed a large-scale dataset of over 20 million clicks from more than 4,000 users, which involved using over 13,000 unique apps in their daily life. Our model predicts the UI element that is likely to be clicked by the user based on the user's click history and the current context, and it provides a general solution for modeling click behaviors on arbitrary UIs. Based on a thorough experiment that compares our approach with multiple alternative methods in the literature, our model significantly outperformed all the baselines. Our experiments also show click prediction is a challenging task. We present our preliminary efforts for integrating the model into mobile interfaces, which shows promises to substantially decrease user effort needed for acquiring next element. The design of our model, our analysis and the experiments provide valuable findings for further investigating the topic.

ACKNOWLEDGMENTS

We would like to thank Xiang Xiao, James Stout, and Ajit Narayanan from the Google Accessibility team for their help and feedback. We would also like to thank anonymous reviewers for their feedback in improving the paper.

REFERENCES

[1] Ricardo Baeza-Yates, Di Jiang, Fabrizio Silvestri, and Beverly Harrison. 2015. Predicting The Next App That You Are Going To Use. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining* (Shanghai, China) *(WSDM '15)*. Association for Computing Machinery, New York, NY, USA, 285–294. https://doi.org/10.1145/2684822.2685302

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* 3, null (March 2003), 1137–1155.

[3] Matthias Böhmer, Brent Hecht, Johannes Schöning, Antonio Krüger, and Gernot Bauer. 2011. Falling Asleep with Angry Birds, Facebook and Kindle: A Large Scale Study on Mobile Application Usage. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (Stockholm, Sweden) *(MobileHCI '11)*. Association for Computing Machinery, New York, NY, USA, 47–56. https://doi.org/10.1145/2037373.2037383

[4] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior sequence transformer for e-commerce recommendation in Alibaba. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. 1–4.

[5] Leah Findlater, Karyn Moffatt, Joanna McGrenere, and Jessica Dawson. 2009. Ephemeral Adaptation: The Use of Gradual Onset to Improve Menu Selection Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. Association for Computing Machinery, New York, NY, USA, 1655–1664. https://doi.org/10.1145/1518701.1518956

[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[7] Ke Huang, Chunhui Zhang, Xiaoxiao Ma, and Guanling Chen. 2012. Predicting Mobile Application Usage Using Contextual Information. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (Pittsburgh, Pennsylvania) *(UbiComp '12)*. Association for Computing Machinery, New York, NY, USA, 1059–1065. https://doi.org/10.1145/2370216.2370442

[8] Jingjing Huangfu, Jian Cao, and Chenyang Liu. 2015. A context-aware usage prediction approach for smartphone applications. In *Asia-Pacific Services Computing Conference*. Springer, 3–16.

[9] S. Lee, R. Ha, and H. Cha. 2019. Click Sequence Prediction in Android Mobile Applications. *IEEE Transactions on Human-Machine Systems* 49, 3 (2019), 278–289.

[10] Yang Li. 2014. Reflection: Enabling Event Prediction as an on-Device Service for Mobile Interaction. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) *(UIST '14)*. Association for Computing Machinery, New York, NY, USA, 689–698. https://doi.org/10.1145/2642918.2647355

[11] Yang Li, Samy Bengio, and Gilles Bailly. 2018. Predicting Human Performance in Vertical Menu Selection Using Deep Learning. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, Article 29, 7 pages. https://doi.org/10.1145/3173574.3173603

[12] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping Natural Language Instructions to Mobile UI Action Sequences. In *Annual Conference of the Association for Computational Linguistics (ACL 2020)*.

[13] Qiang Liu, Feng Yu, Shu Wu, and Liang Wang. 2015. A convolutional click prediction model. In *Proceedings of the 24th ACM international on conference on information and knowledge management*. 1743–1746.

[14] Jiaxin Mao, Cheng Luo, Min Zhang, and Shaoping Ma. 2018. Constructing Click Models for Mobile Search. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (Ann Arbor, MI, USA) *(SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 775–784. https://doi.org/10.1145/3209978.3210060

[15] Abhinav Parate, Matthias Böhmer, David Chu, Deepak Ganesan, and Benjamin M. Marlin. 2013. Practical Prediction and Prefetch for Faster Access to Applications on Mobile Phones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Zurich, Switzerland) *(UbiComp '13)*. Association for Computing Machinery, New York, NY, USA, 275–284. https://doi.org/10.1145/2493432.2493490

[16] J. Shen and M. O. Shafiq. 2019. Learning Mobile Application Usage - A Deep Learning Approach. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. 287–292.

[17] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. 2011. LiveLab: Measuring Wireless Networks and Smartphone Users in the Field. *SIGMETRICS Perform. Eval. Rev.* 38, 3 (Jan. 2011), 15–20. https://doi.org/10.1145/1925019.1925023

[18] Choonsung Shin, Jin-Hyuk Hong, and Anind K. Dey. 2012. Understanding and Prediction of Mobile Application Usage for Smart Phones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* (Pittsburgh, Pennsylvania) *(UbiComp '12)*. Association for Computing Machinery, New York, NY, USA, 173–182. https://doi.org/10.1145/2370216.2370243

[19] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (Marina Del Rey, CA, USA) *(WSDM '18)*. Association for Computing Machinery, New York, NY, USA, 565–573. https://doi.org/10.1145/3159652.3159656

[20] Erum Tanvir, Andrea Bunt, Andy Cockburn, and Pourang Irani. 2011. Improving cascading menu selections with adaptive activation areas. *Int. J. Hum. Comput. Stud.* 69, 11 (2011), 769–785. https://doi.org/10.1016/j.ijhcs.2011.06.005

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[22] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2692–2700. http://papers.nips.cc/paper/5866-pointer-networks.pdf

[23] Chuhan Wu, Fangzhao Wu, Suyu Ge, Tao Qi, Yongfeng Huang, and Xing Xie. 2019. Neural News Recommendation with Multi-Head Self-Attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 6389–6394. https://doi.org/10.18653/v1/D19-1671

[24] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. 2020. SSE-PT: Sequential Recommendation Via Personalized Transformer. In *Fourteenth ACM Conference on Recommender Systems* (Virtual Event, Brazil) *(RecSys '20)*. Association for Computing Machinery, New York, NY, USA, 328–337. https://doi.org/10.1145/3383313.3412258

[25] Jingling Zhao, Yan Li, Jie Wu, and Qing Liao. 2018. Predict What App to Use next Time Only Consider Time and Latest Used App Context. In *Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences* (Wuhan, China) *(ICMSS 2018)*. Association for Computing Machinery, New York, NY, USA, 163–166. https://doi.org/10.1145/3180374.3181345