ActionBert: Leveraging User Actions for Semantic Understanding of User Interfaces

Zecheng He [1], **Srinivas Sunkara** [2], **Xiaoxue Zang** [2], **Ying Xu** [2], **Lijuan Liu** [2], **Nevan Wichers** [2], **Gabriel Schubiner** [2], **Ruby Lee** [1], **Jindong Chen** [2]

[1] Princeton University
[2] Google Research

{zechengh, rblee}@princeton.edu, {srinivasksun, xiaoxuez, yingyingxuxu, lijuanliu, wichersn, gsch, jdchen}@google.com

Abstract

As mobile devices are becoming ubiquitous, regularly interacting with a variety of user interfaces (UIs) is a common aspect of daily life for many people. To improve the accessibility of these devices and to enable their usage in a variety of settings, building models that can assist users and accomplish tasks through the UI is vitally important. However, there are several challenges to achieve this. First, UI components of similar appearance can have different functionalities, making understanding their function more important than just analyzing their appearance. Second, domain-specific features like Document Object Model (DOM) in web pages and View Hierarchy (VH) in mobile applications provide important signals about the semantics of UI elements, but these features are not in a natural language format. Third, owing to a large diversity in UIs and absence of standard DOM or VH representations, building a UI understanding model with high coverage requires large amounts of training data.

Inspired by the success of pre-training based approaches in NLP for tackling a variety of problems in a data-efficient way, we introduce a new pre-trained UI representation model called ActionBert. Our methodology is designed to leverage visual, linguistic and domain-specific features in user interaction traces to pre-train generic feature representations of UIs and their components. Our key intuition is that user actions, e.g., a sequence of clicks on different UI components, reveals important information about their functionality. We evaluate the proposed model on a wide variety of downstream tasks, ranging from icon classification to UI component retrieval based on its natural language description. Experiments show that the proposed ActionBert model outperforms multi-modal baselines across all downstream tasks by up to 15.5%.

Introduction

Given the prevalence and importance of smart devices in our daily life, the ability to understand and operate User Interfaces (UIs) has become an important task for Artificial Intelligence. For instance, a model that can find a UI component by its description can be very useful for voice interfaces, and a model that can predict the expected output of clicking a button can help page navigation. To successfully operate a UI, the models need to understand the user task and intents, and how to perform the tasks in the given UI.

However, UI understanding is a challenging and less-studied area. First, there are various tasks related to UI understanding. Usually, these tasks are cross-modal and cross-domain, e.g., clicking a button through voice command and retrieving an icon via a semantically similar one. Previous works in this field usually target one single task at a time. Training a different complex model for each task is not efficient for on-device model deployment. Moreover, models may suffer from overfitting if the task-specific data is limited. Pre-training models on large-scale datasets to extract features has shown great power in multiple domains, e.g., ResNet (He et al. 2016) in computer vision and BERT (Devlin et al. 2018) in natural language processing. There is no such generic feature representation for user interfaces and it is not clear if a pre-trained feature extraction model can help improve multiple UI related tasks.

Second, the data source and format of UIs are different from natural image and language corpuses. For example, the View Hierarchy (VH) in mobile apps and Document Object Model (DOM) in web pages are tree structures representing the UI layout. The VH and DOM contain structural and semantic information about the UI, however they are not generally visible to the users and they also contain short phrases with hints about functionality. Effectively making use of this domain-specific knowledge for general UI understanding is an unsolved problem.

Third, understanding the functionality of UIs is more challenging than learning about their appearance. It is common that UI elements of similar appearance have very different semantic meanings, and vice versa. In the example in Figure 1, all The icons look similar to each other ("houses"), however, they have different functionalities which can only be interpreted with additional context.

In this paper, we propose ActionBert, a pre-trained transformer-style (Vaswani et al. 2017) model that leverages sequential user action information for UI understanding. Our key intuition is that the semantic meaning of a UI, and the functionality of UI components can be captured by user actions, e.g. a sequence of clicks and their effect on the UI. This model takes advantage of the representation power of transformer models, and integrates domain-specific information like VH and user actions to build embeddings reflecting the functionality of different UI elements. To the best of our knowledge, it is the first attempt to build a generic fea-

Figure 1: Examples where low-level appearance does not reflect the semantics of UI components without context. Shown below are their semantics on the source screen.
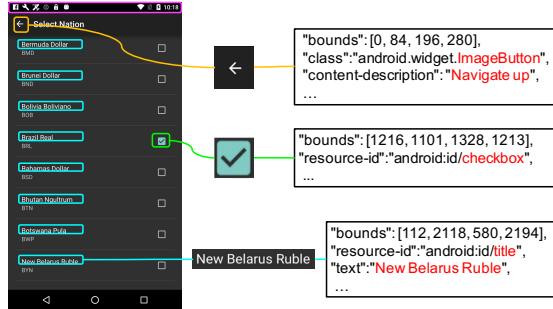

Figure 2: An example of leaf nodes in a view hierarchy. The view hierarchy provides useful semantic information (marked red) for machines to understand the UI.

ture representation in this field. Our main contributions are:

- To the best of our knowledge, we are the first to integrate the powerful transformer models and domain-specific knowledge, e.g., VH and user actions, to improve machines' understanding of UIs.

- We propose ActionBert, a transformer-style multi-modal model to capture the context and semantic meaning of UI elements by introducing new self-supervised pre-training tasks based on user actions and UI-specific features.

- We evaluate the proposed model on four types of UI downstream tasks, capturing various real-life use cases. We show that the proposed model outperforms the existing models on all tasks.

Background of UI View Hierarchy

A View hierarchy is a tree-based representation of a user interface. It has various attributes related to the appearance and functionality of UI components. In this paper, we leverage the content description, resource id, component class, component text fields in the leaf nodes of view hierarchy:

- The content description is a brief description of the functionality of this UI component provided by the developer.

- The component text is the visible text on the component.

- The resource id and component class indicate the type of the component, e.g., button, checkbox or image, and the name of the static files used to render it.

Some of the important information, like content description, is invisible to the user but can be used by applications like Screen Readers to understand the UI. Figure 2 shows examples of leaf nodes in a view hierarchy.
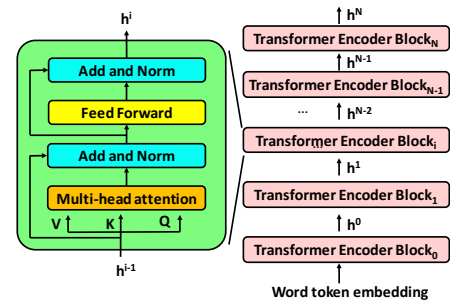

Figure 3: BERT and transformer encoder blocks.

ActionBert

Revisiting BERT

ActionBert is inspired by the great success of BERT (Devlin et al. 2018) in natural language processing (NLP). We briefly review the BERT model, and then extend the concepts to learn UI embedding.

BERT is a transformer-based (Vaswani et al. 2017) bidirectional language model. BERT-style models have shown great success in transferring learned features to multiple NLP tasks. On a high level, BERT takes in the embedding of word tokens and processes them through a multi-layer bidirectional transformer (Vaswani et al. 2017).

BERT is pre-trained with two tasks: masked language modeling (MLM) and next sentence prediction (NSP). In MLM task, some input words are randomly masked out and replaced with a special token [MASK]. The task is to predict the masked word based on the clues from the unmasked words in its context. The NSP task is defined as, given two sentences predict whether one is immediately after the other. To separate the two sentences, a special token [SEP] is inserted between them. A classifier is applied to the BERT embedding and outputs the probability of the second sentence immediately following the first one. More details on BERT can be found in (Devlin et al. 2018).

ActionBert: Semantic UI Understanding with User Actions

Inspired by the BERT model, we adopt the concepts of NLP and extend them to UI understanding. We treat the UI components, e.g., buttons, icons, checkboxes, as the basic building blocks of a user interface. Similar to sentences, which are composed of word tokens, we treat these basic UI components (buttons, icons, etc.) as tokens, and the whole UI as a sentence in NLP. A user interaction trace is a sequence of UIs obtained by starting from a particular UI and interacting with different UI components. Different from sentences, UIs in this trace are linked through a link component, usually a clickable component like a button or an icon. When a user takes an action on that link component, the screen jumps to the next UI. Such a sequence of UIs is analogous a to paragraph in NLP. Table 1 shows a mapping of the concepts between NLP and UI understanding.

Following this analogy, our key idea is that the semantic meaning of a UI component can be identified from components in the same UI and the UI that follows the current one.

Natural Language Processing	UI Understanding
Tokens	UI components (button etc)
Sentences	UIs
Word context	Components in the same UI
Consecutive sentences	Consecutive UIs
Paragraph	Sequence of UIs
Language model	UI embedding model

Table 1: Concepts mapping between NLP and UI understanding.

Figure 4: An example of user actions on UIs. The user clicks on the "tick" button in screen 1 and jumps to the boarding-pass page, screen 2. The semantics of the button, online check-in, can be inferred from components on the homepage (e.g., images of plane, airline name) and the components on the next UI (e.g., QR code, passenger information).

We illustrate this idea in an example in Figure 4. Here, the first UI is the homepage of an airline app. The user clicks the button with a tick and a circle on it. This button links to a new UI with passenger, time, gate information and a QR code on it. From the elements in the current UI and the next UI, the functionality of the button can be interpreted as "online check-in". Similarly, when the user clicks on the "plus" button it links to a UI with more detailed flight information on it. Hence, the "plus" component indicates "show details".

We propose ActionBert that takes a pair of UIs as input, and outputs the contextual embedding of the UIs and the individual UI components. Figure 5 shows the model architecture. It extends the original BERT model by adding vision modality and leverages user-action related tasks for pre-training. Inspired by the recent vision-language model VL-BERT (Su et al. 2019), the ActionBert model uses a uni-stream architecture that allows full-attention across modalities. First, the two input UIs (UI-A and UI-B) are split into four component segments: UI-A text, UI-B text, UI-A vision and UI-B vision. A special token [CLS] is prepended to the component sequence, similar to the original BERT model, whose embedding represents the whole two input UIs. The different segments, representing the text and vision parts of the two UIs are separated with a special token [SEP] and end with another special token [END].

Text embedding Different from BERT and other vision-language models, the text tokens of ActionBert are specifically designed for UI tasks. Each text token (green box in Figure 5) is a concatenation of content description, resource id, component class name and text in a view hierarchy leaf node (Section). The vision segment slots of text tokens are filled with a special token [IMG]. Overall, each text token,

which is a concatenation of the different fields in View Hierarchy, is treated as a sentence and processed through a sentence-level text encoder, e.g. BERT.

Vision embedding Similar to the text tokens, the vision tokens are also specific to the nature of UIs. If the view hierarchy of a UI is available, each vision token is cropped from the UI using the bounding box of a VH leaf node. If the VH is not available, we fine-tune a Faster-RCNN (Ren et al. 2015) to detect UI components in a screenshot and crop components from the detected bounding boxes. Furthermore, a vision encoder, e.g., ResNet-50, is used to generate the input vision embedding from the cropped images. Specifically, from the vision encoder, we take the flattened feature map from the layer just before the fully connected layer as the input vision embedding. Vision tokens of UI-A text, UI-B text and special tokens ([CLS], [SEP] and [END]) are set as the corresponding whole UI screenshots.

Positional embedding Positional embedding represents the geometrical position of UI components in the UI. Unlike word tokens in language, components in a UI are not linearly ordered, but are arranged in a 2D space. We define nine features to represent the positional features of a UI component, i.e. $x_{min}, y_{min}, x_{max}, y_{max}, x_{center}, y_{center}$, height, width and area. x_{min}, y_{min} correspond to the top-left corner and x_{max}, y_{max} correspond to the bottom-right corner of the UI component, respectively. To deal with the different sizes of UIs, we normalize x and y relative to the width and height of the UI, respectively.

Segment embedding Segment embedding indicates whether the corresponding UI component is from UI-A or UI-B, and is a text or vision component. There are four types of segment embedding representing UI-A text, UI-B text, UI-A vision and UI-B vision, respectively. In practice, we define a fifth segment type, padding segment, to pad the input sequences to a fixed-length for batch processing.

The four types of input features are processed through a linear layer followed by a normalization layer (Ba, Kiros, and Hinton 2016). Then they are summed up and passed as input to ActionBert, as a single tensor of shape $L * D_1$, where L is the number of components in the UI pair and D_1 is the input embedding dimension. ActionBert is a uni-stream architecture, allowing attention across components and modalities. The output of ActionBert is a contextual embedding of shape $L * D_2$, where D_2 is the output embedding dimension of the ActionBert. The output embedding at position i represents the contextual embedding of UI component i, while the embedding of the first component [CLS] provides an overall representation of the UI pair.

Pre-training ActionBert

ActionBert is pre-trained on three new tasks that are specifically designed to integrate user actions and UI-specific features: link component prediction, consecutive UI prediction, and masked VH text prediction. The first two pre-training tasks use UI sequences and user actions to learn the connectivity and relationship of two UIs. The last pre-training
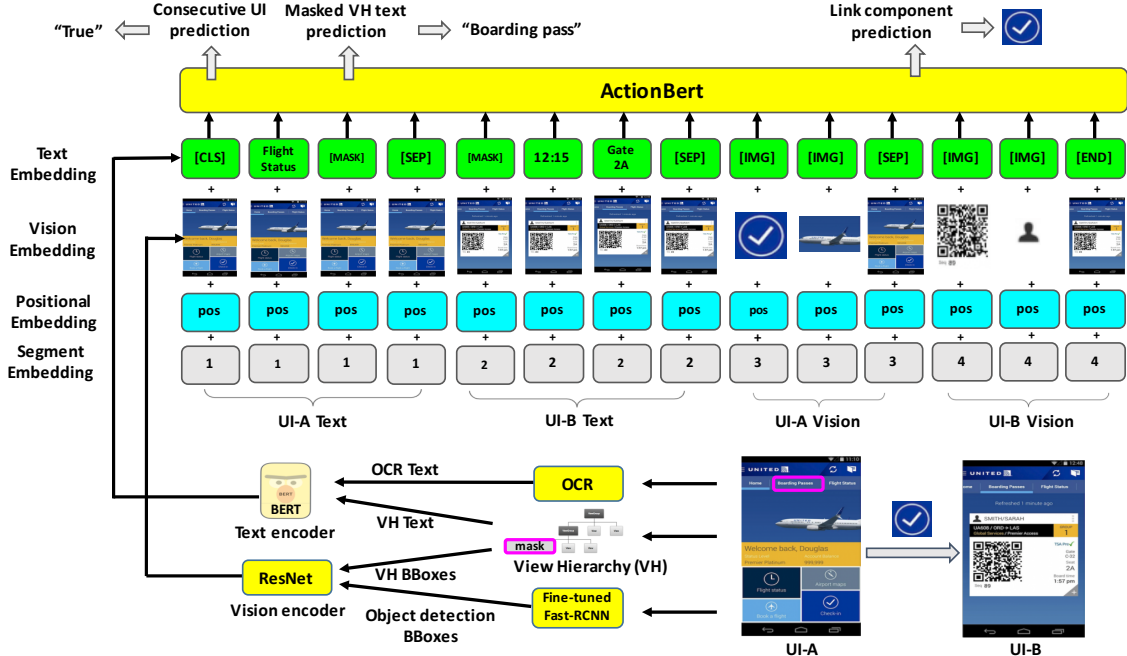
Figure 5: ActionBert model architecture. On a high-level, the ActionBert model takes a pair of UIs, represented by their text, vision, positional and segment embedding as input. Three new UI-specific tasks, i.e. link component prediction, consecutive UI prediction and masked text prediction, are defined to pre-train ActionBert on large-scale UI sequences with user actions.

task learns the relationship between the text features of a UI component and its context (vision and text).

For pre-training, we used a large scale internal dataset obtained by automatically crawling various apps. Our data consists of 60,328 user action sequences on UIs. Each sequence $S=[s_1, s_2...s_T]$ contains T UIs, where T ranges from two (a single click) to hundreds. Each pair of consecutive UIs (s_{i-1}, s_i) also has an action location (x,y), indicating the click position that results in the transition from s_{i-1} to s_i. We extract 2.69M UIs with their view hierarchy from the sequences. We perform a 50%-50% negative sampling to generate non-consecutive UI pairs for the consecutive UI prediction task. Among the negative pairs, half are from the same sequence but not consecutive, while the other half of the negative pairs are from different user sequences. In total, the ActionBert is pre-trained on 5.4M UI pairs with user actions and view hierarchy.

Pre-training task #1: Link component prediction (LCP)
This task is specifically designed to incorporate the user action information from UI sequences. Given two UIs, the task is to predict which component can be clicked on to jump from UI-A to UI-B. The correct link component is obtained via user click position (x,y) during the training data generation. To correctly identify the link components, the model has to learn the semantics of both UIs and find a component whose functionality is to link them. The model takes all text and vision components of both UIs as candidates and selects one from them. The objective can be formulated as

$$p = softmax(MLP(f_\theta(x))), \qquad (1)$$
$$L_{LCP} = -\Sigma_{x \in D}\mathbf{1}_{LC}(x)CE(p,y), \qquad (2)$$

where x is sampled from the training set D. $f_\theta(x)$ represents the embedding generated by the ActionBert model. $MLP(\cdot)$ is a multi-layer perceptron, and p is the predicted probability of each UI component being the link component. $\mathbf{1}_{LC}(x)$ is an indicator function of value 1 if the link component is available in this training pair, i.e., the two UIs are consecutive and the click location refers to a UI component, otherwise 0. $CE(\cdot)$ is a standard multi-class cross-entropy loss.

Pre-training task #2: Consecutive UI prediction (CUI)
Inspired by the next sentence prediction task in BERT pre-training to model the relationship of two sentences, we propose this task to learn the relationship between two UIs. As shown in Table 1, we analogize a UI to a sentence in NLP. The consecutive UIs prediction task predicts whether UI-B can be obtained by a single interaction from UI-A. In pre-training, a UI pair (s_{i-1}, s_i) from the same sequence S is a positive training sample pair. We perform a 50%-50% negative sampling to generate negative samples. Among the negative samples, half of them (25% of total training pairs) are generated by sampling two non-consecutive UIs from the same sequence, i.e. (s_i, s_j) where $i + 1 \neq j$. The other half consists of two UIs from different user interaction sequences, i.e. (s_i, v_j), where s_i is from sequence S and v_j is from sequence V and $S \neq V$. Formally, the loss function is

$$L_{CUI} = -\Sigma_{x \in D}ylog(\hat{y}) + (1 - y)log(1 - \hat{y}) \qquad (3)$$

where x is a training sample (a pair of UIs) from the training set D, and y is the label of whether the two UIs are consecutive. $\hat{y}$=sigmoid(MLP($f_\theta(x)$)) is the model predicted probability that the pair of UIs in x are consecutive. A standard binary cross-entropy loss is applied to it.

Pre-training task #3: Masked VH text prediction This task is similar to the masked language modeling (MLM) task in BERT pre-training. We randomly mask 15% text components from the UI view hierarchy. The main difference is, as each text token in ActionBert is a concatenation of multiple fields (content description, source id, component type and text) from the view hierarchy (Section), it contains more than one word token and we treat it as a text "sentence". Therefore, compared to BERT where each word token is directly predicted, ActionBert predicts the high-dimensional text embedding of the sentence and treats it as a regression task. Formally, the loss for masked VH text prediction is:

$$L_{mask} = \Sigma_{x \in D} \Sigma_{i=0}^{N-1} \mathbf{1}_{mask}(x,i) || f_\theta^i(x_{mask}) - g^i(x) ||_2^2 \quad (4)$$

where D is the training set. x is the unmasked training example and x_{mask} is the training example with masked text. N is the total number of UI components in a training example. f_θ is the ActionBert model with parameter θ, and g is the sentence-level text encoder (we choose BERT in the pre-training), respectively. $f_\theta^i(x_{mask})$ denotes the Action-Bert model output embedding of the i-th component in the masked example x_{mask}. $g^i(x)$ denotes the sentence encoder output of the i-th component in the unmasked example x. $\mathbf{1}_{mask}(x,i)$ is an indicator function whose value is 1 if the component i is masked in example x, otherwise 0.

The overall loss function for pre-training is defined as:

$$L = L_{LCP} + \lambda_{CUI} L_{CUI} + \lambda_{mask} L_{mask} \quad (5)$$

Fine-tuning ActionBert

Similar to how BERT is used as a generic feature representation for different NLP downstream tasks, fine-tuning ActionBert for a variety of UI understanding tasks is relatively easy and does not require substantial task-specific architecture changes nor a large amount of task-specific data. The downstream input to ActionBert needs to be appropriately formatted into segments, as illustrated in Figure 5. During fine-tuning, a task-specific loss function is added on top of ActionBert for training. All parameters, including the text and vision encoder, are jointly tuned in an end-to-end manner to achieve the best performance.

Same as pre-training, we use VH texts and bounding boxes if VH is available in a downstream task. Otherwise, we perform OCR and object detection to extract the text and vision components from the UI, respectively. It is worth noting that, although the ActionBert is pre-trained on UI pairs, it can also handle single-UI and multi-UI tasks.

Single-UI tasks As discussed above, the input data format is designed as [UI-A text, UI-B text, UI-A vision, UI-B vision]. For the downstream tasks which only involve a single UI, the input data can be converted into the ActionBert format by leaving the UI-B text and UI-B vision segments empty. For tasks involving natural language input, e.g., UI component retrieval based on its description, the language input can be passed in the UI-A text segment with the whole UI screenshot being used as the vision token.

Two-UI tasks Since the ActionBert model is pre-trained on UI pairs, it is natural to apply this model on tasks with two UIs as input, e.g. similar UI component retrieval (Section).

We can assign the corresponding text and vision components to UI-A/B text and vision segments in Figure 5.

Extension to multi-UI tasks ActionBert can also be extended to multi-UI (≥ 3) settings, though these types of tasks are not common in practice. Similar to two-UI tasks, different UIs and modalities need to be separated by the [SEP] token. The only difference is that more segment embedding representing the newer UI segments needs to be trained.

Experiments

Pre-training

As described in Section , we pre-train the ActionBert model on large-scale UI sequences with user actions and view hierarchy. We obtained the pre-training action sequences using the Robo app crawler [1]. In total, we used 60,328 UI sequences and extracted 5.4M UI pairs for pre-training. We split this data in the ratio of 80%:10%:10% to obtain the train, dev and test sets, respectively. We prevent leakage of data across these splits by ensuring any app can go into only one of these splits. We pre-train two models of different sizes for comparison: AB_{BASE} and AB_{LARGE} are 6-layers and 12-layers transformer architectures both with 6 heads and 768 hidden dimensions initialized with Xavier initialization.

We use Adam optimizer (Kingma and Ba 2014) with learning rate $r = 10^{-5}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$ and batch size = 128 for training. We set $\lambda_{CUI} = 0.1$ and $\lambda_{mask} = 0.01$ in Eq. (5) during the pre-training. ActionBert is pre-trained with 16 TPUs for three days.

Fine-tuning

ActionBert can be fine-tuned on multiple types of downstream tasks. We evaluate it on four representative UI downstream tasks: similar component retrieval (across app and web UIs), referring expression component retrieval, icon classification and app type classification. Additionally, we perform link component prediction, one of the pre-training tasks, on a different dataset. To understand the performance of ActionBert, we use a benchmark model where we obtain embedding for each UI component by using ResNet to encode the image and BERT to encode the text attributes. The final classification layer added on top of the embedding is same as that added for ActionBert. Additionally, for each of the downstream tasks, we also evaluate the performance of a non pre-trained ActionBert model. This model has the same architecture as a pre-trained model but the model parameters are initialized randomly for each task. This comparison allows us to understand the impact of the transformer architecture and the pre-training tasks. Please refer to the supplementary material for more details regarding the different downstream tasks and the data collection details.

Similar UI component retrieval Similar UI component retrieval focuses on the high-level functionality of UI components. Given an anchor component on a UI, the task is
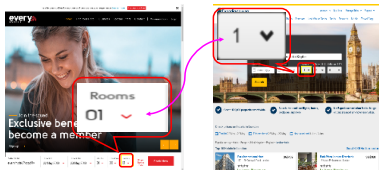
_____
[1]https://firebase.google.com/docs/test-lab/android/robo-ux-test

Figure 6: An example of similar UI component retrieval.

Model	App UI Acc	App UI Gain	Web UI Acc	Web UI Gain
ResNet + BERT	83.4	–	–	–
MobileNet + USE	–	–	50.2	–
$AB_{BASE-NP}$	85.2	+1.8	62.8	+12.6
AB_{BASE}	85.5	+2.1	63.7	+13.5
$AB_{LARGE-NP}$	86.1	+2.7	62.1	+11.9
AB_{LARGE}	**86.4**	**+3.0**	**64.4**	**+14.2**

Table 2: Comparison to the baseline models for similar UI component retrieval on app and web UIs.

to choose the most similar component based on their functionality on the other UI from a list of UI component candidates (Figure 6). After generating the component-level embedding using ActionBert, we use dot-product of the embedding of the anchor and candidate components as the similarity scores and select the component with the highest score as the prediction. We use concatenated embeddings from ResNet + BERT for each UI component as the benchmark.

We perform similar UI component retrieval on app and web UIs. For the app UI task, we collected 900k app UI pairs with annotations of similar components for training, and 32k for validation and testing, respectively. [2] For the web UI task, we collected 65k web UI pairs with annotations of similar components, which are jointly used with the app UIs for training. We manually labeled 2k web UI pairs with similar components for testing. On average, the model chooses the correct component from 10 candidate components on app UIs, and 35 candidates on web UIs.

Table 2 shows the results of similar UI component retrieval. For the app similar component retrieval, the pre-trained ActionBert (AB_{LARGE}) model outperforms the baseline ResNet+BERT model by 3.0% in accuracy. For the web similar component retrieval task, the pre-trained model outperforms the benchmark MobileNet + Universal Sentence Encoder (USE) (Cer et al. 2018) model, which is specifically designed for web similar component retrieval, by 14.2%. Furthermore, the pre-trained models ($AB_{BASE/LARGE}$) achieve higher accuracy than the non pre-trained models ($AB_{BASE/LARGE-NP}$), which shows the benefit of using a generic pre-trained feature representation in UI understanding.

Referring expression component retrieval The referring expression component retrieval task takes a referring expression and a UI screenshot as input, and the model is expected to select the UI component that the expression refers to (Fig-

---

[2] The view hierarchy is not available in this dataset, so we rely on OCR text from the screenshot instead.
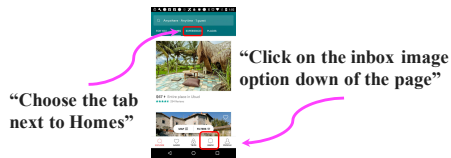

Figure 7: Referring expression UI component retrieval task.

Model	Dev	Test	Test Gain
ResNet+BERT	87.6	86.2	–
$AB_{BASE-NP}$	86.7	85.7	-0.5
AB_{BASE}	90.1	88.4	+2.2
$AB_{LARGE-NP}$	89.5	89.2	+3.0
AB_{LARGE}	89.8	**90.2**	**+4.0**

Table 3: Comparison to the baseline ResNet+BERT embedding for UI referring expression component retrieval.

ure 7) from a list of the UI components detected on the screen. This is a typical task in voice-based control systems where the user can interact with an element on the screen by describing it, e.g., "click settings", "select on the back arrow at the top". To correctly select the UI component, the model not only has to take into account the semantics of the component, but also its relationship to other components.

We collected and manually annotated 16.9k UI components with referring expressions for training, 2.1k for validation and 1.8k for testing. We used a fine-tuned Faster-RCNN to select negative samples (components that are not referred by the expression) in a UI, by selecting the objects with high object detection scores that do not overlap with the correct component. On average, the model chooses the correct component that the expression refers to from 20 UI components.

We present the results of referring expression UI component retrieval in Table 3. We observe that the pre-trained AB_{LARGE} model performs the best. It achieves 4.0% improvement over the ResNet+BERT benchmark model, and outperforms non pre-trained $AB_{LARGE-NP}$.

Icon classification The goal of this task is to identify the type of an icon. Having this information is beneficial for screen readers to understand the type of elements when content description and alt-text are not present. We use the Rico (Deka et al. 2017) dataset for this task. Rico is the largest public mobile app design dataset, containing 72k unique screenshots with their view hierarchies. However, the icons in the dataset are labeled using heuristics and simple ML models relying on view hierarchy attributes making the annotations inaccurate. Hence, we use crowd-sourcing to label icons of this dataset in two degrees of granularity, i.e. 32 top-used icon classes and 77 more detailed icon classes.

We use the ActionBert embedding in the corresponding position as the contextual embedding of the UI components for icon classification. We summarize the icon classification results in Table 4. For icon-32 classification, the pre-trained AB_{LARGE} model achieves the best macro-accuracy of 97.50% and micro-F1 of 0.8766, 0.18% and 1.11% higher than the ResNet+BERT benchmark, respectively. On the finer-granularity (77 classes) icon classification, AB_{LARGE}

	Rico-32		Rico-77	
Model	Acc	Macro F1	Acc	Macro F1
ResNet+BERT	97.32	0.866	91.28	0.626
$AB_{BASE-NP}$	97.38	0.867	91.57	0.630
AB_{BASE}	97.42	0.874	91.60	**0.638**
$AB_{LARGE-NP}$	97.39	0.870	91.56	0.632
AB_{LARGE}	**97.50**	**0.877**	**91.65**	0.631

Table 4: Comparison to the baseline ResNet+BERT embedding for icon classification.

Model	Acc	Gain	Macro F1	Gain
ResNet+BERT	64.3	–	0.598	–
$AB_{BASE-NP}$	78.6	14.3	0.753	0.155
AB_{BASE}	**79.8**	**15.5**	**0.764**	**0.166**

Table 5: Comparison to the baseline ResNet+BERT embedding for app type classification.

obtains 0.37% improvement on accuracy over the baseline. A smaller AB_{BASE} performs slightly better on micro-F1, because of the skewed class distribution in the Rico dataset.

App type classification In this task, the goal of the model is to predict the type of an app, e.g., shopping, communication, arts etc. In total, we examined 27 app types. Similar to icon classification, we use the public Rico dataset for this task. The app type is extracted from the description in the app store. We use 43.5k unique app UIs with their view hierarchies and app types, and split them in the ratio 80%, 10%, 10% for training, validation and testing. Compared to the icon classification above, which is a component-level task, the app type classification is a UI-level task.

From Table 5 we can see that Compared to the ResNet+BERT benchmark model, the pre-trained Action-Bert leads to an increase in accuracy by 15.5% and macro F1 by 16.6%. Furthermore, pre-trained ActionBert outperforms the non pre-trained version by 1.2% and 1.1% on micro accuracy and macro F1 metrics, respectively. This demonstrates the effectiveness of the pre-training tasks.

Link component prediction We also evaluate the performance of ActionBert on link component prediction (Section), one of the pre-training tasks, on the publicly available Rico dataset. Rico also contains 10k user interaction sequences, but typically the length of these sequences is less than the ones we used for pre-training. We extract 90k consecutive UI pairs from the Rico sequences and perform link component prediction on them.

From Table 6, we can see that Pre-trained and non pre-trained ActionBert models significantly outperform the baseline ResNet+BERT model, showing the benefit of the

	Acc	Gain
ResNet+BERT baseline	40.2	–
$AB_{BASE-NP}$	48.2	+8.0
AB_{BASE}	**51.6**	**+11.4**

Table 6: Comparison to the baseline ResNet+BERT embedding on link component prediction (a pre-training task).

unified single-stream attention architecture. The pre-trained model outperforms the non pre-trained one by 3.4%.

In summary, across all the fine-tuning tasks, the pre-trained ActionBert outperforms the baselines and the non pre-trained model, suggesting the effectiveness of our proposed method in generating generic UI embedding.

Related Work

Pretrained embeddings have been studied in various domains, e.g., computer vision (Szegedy et al. 2015; He et al. 2016), natural language processing (Devlin et al. 2018; Vaswani et al. 2017) and security (He, Zhang, and Lee 2019b, 2020, 2019a). Previous works have studied UI embeddings for specific applications, e.g., UI search and design. (Deka et al. 2017) and (Huang, Canny, and Nichols 2019) proposed using auto-encoder and VGG-style architectures to embed UI layout information for similar UI retrieval, respectively. (Liu et al. 2018) used also finer element-level layout information for retrieval. (Wichers, Hakkani-Tür, and Chen 2018) demonstrated an embedding approach to retrieve images from a referring expression. Besides UI search, UI understanding provides insights to the UI designers, e.g., predicting user engagement level (Wu et al. 2020), user impressions of the app (Wu et al. 2019) and perceived functionality of the UI element (Swearngin and Li 2019). These works each targeted a specific UI task, while we are the first to show that a generic pre-trained feature representation can help improve various UI understanding tasks. Furthermore, none of the previous works have investigated user actions to improve UI understanding.

Concurrently, beyond the scope of UI understanding, research work has been proposed on pre-training feature representation for vision-linguistic tasks, e.g., VL-BERT (Su et al. 2019), ViL-BERT (Lu et al. 2019), B2T2 (Alberti et al. 2019), VisualBERT (Li et al. 2019), UNITER (Chen et al. 2020) and ImageBERT (Qi et al. 2020). However, these work focus on natural language and images. They do not consider the domain-specific user actions and UI features, thus are not directly applicable to UI understanding. Our proposed model explicitly integrates UI domain-specific information into the pre-training process.

Conclusion

In this paper, we explore using user actions to build generic feature representations to facilitate UI understanding. We present ActionBert, the first pre-trained UI embedding model that can be applied to multiple UI understanding tasks. ActionBert is pre-trained on a large dataset of sequential user actions and UI domain-specific features. Experiment results show that pre-training helps in improving the performance across four types of representative UI tasks. ActionBert also significantly outperforms multi-modal baselines across all downstream tasks by up to 15.5%. We hope that this study can raise awareness about the importance of pre-trained feature representations in this field and spur the development of useful models for various UI related tasks.

References

Alberti, C.; Ling, J.; Collins, M.; and Reitter, D. 2019. Fusion of Detected Objects in Text for Visual Question Answering. *arXiv preprint arXiv:1908.05054* .

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer Normalization. *arXiv preprint arXiv:1607.06450* .

Cer, D.; Yang, Y.; Kong, S.-y.; Hua, N.; Limtiaco, N.; John, R. S.; Constant, N.; Guajardo-Cespedes, M.; Yuan, S.; Tar, C.; et al. 2018. Universal Sentence Encoder. *arXiv preprint arXiv:1803.11175* .

Chen, Y.-C.; Li, L.; Yu, L.; Kholy, A. E.; Ahmed, F.; Gan, Z.; Cheng, Y.; and Liu, J. 2020. Uniter: Universal Image-text Representation Learning. In *European Conference on Computer Vision (ECCV)*, 104–120.

Deka, B.; Huang, Z.; Franzen, C.; Hibschman, J.; Afergan, D.; Li, Y.; Nichols, J.; and Kumar, R. 2017. Rico: A Mobile App Dataset for Building Data-driven Design Applications. In *ACM Symposium on User Interface Software and Technology (UIST)*, 845–854.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* .

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

He, Z.; Zhang, T.; and Lee, R. 2019a. Sensitive-sample fingerprinting of deep neural networks. In *IEEE on Computer Vision and Pattern Recognition (CVPR)*, 4729–4737.

He, Z.; Zhang, T.; and Lee, R. B. 2019b. Model inversion attacks against collaborative inference. In *Annual Computer Security Applications Conference (ACSAC)*, 148–162.

He, Z.; Zhang, T.; and Lee, R. B. 2020. Attacking and Protecting Data Privacy in Edge-Cloud Collaborative Inference Systems. *IEEE Internet of Things Journal* .

Huang, F.; Canny, J. F.; and Nichols, J. 2019. Swire: Sketch-Based User Interface Retrieval. In *ACM CHI Conference on Human Factors in Computing Systems (CHI)*, 1–10.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* .

Li, L. H.; Yatskar, M.; Yin, D.; Hsieh, C.-J.; and Chang, K.-W. 2019. Visualbert: A Simple and Performant Baseline for Vision and Language. *arXiv preprint arXiv:1908.03557* .

Liu, T. F.; Craft, M.; Situ, J.; Yumer, E.; Mech, R.; and Kumar, R. 2018. Learning Design Semantics for Mobile Apps. In *ACM Symposium on User Interface Software and Technology (UIST)*, 569–579.

Lu, J.; Batra, D.; Parikh, D.; and Lee, S. 2019. Vilbert: Pre-training Task-agnostic Visiolinguistic Representations for Vision-and-language Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Qi, D.; Su, L.; Song, J.; Cui, E.; Bharti, T.; and Sacheti, A. 2020. Imagebert: Cross-modal Pre-training with Large-scale Weak-supervised Image-text Data. *arXiv preprint arXiv:2001.07966* .

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Advances in neural information processing systems (NeurIPS)*, 91–99.

Su, W.; Zhu, X.; Cao, Y.; Li, B.; Lu, L.; Wei, F.; and Dai, J. 2019. VL-BERT: Pre-training of Generic Visual-Linguistic Representations. In *International Conference on Learning Representations (ICLR)*.

Swearngin, A.; and Li, Y. 2019. Modeling Mobile Interface Tappability Using Crowdsourcing and Deep Learning. In *ACM CHI Conference on Human Factors in Computing Systems (CHI)*, 1–11.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 6000–6010.

Wichers, N.; Hakkani-Tür, D.; and Chen, J. 2018. Resolving Referring Expressions in Images with Labeled Elements. In *IEEE Spoken Language Technology Workshop (SLT)*, 800–806.

Wu, Z.; Jiang, Y.; Liu, Y.; and Ma, X. 2020. Predicting and Diagnosing User Engagement with Mobile UI Animation via a Data-Driven Approach. In *ACM CHI Conference on Human Factors in Computing Systems (CHI)*, 1–13.

Wu, Z.; Kim, T.; Li, Q.; and Ma, X. 2019. Understanding and Modeling User-Perceived Brand Personality from Mobile Application UIs. In *ACM CHI Conference on Human Factors in Computing Systems (CHI)*, 1–12.