Lexi: Self-Supervised Learning of the UI Language

Pratyay Banerjee[1*†] **Shweti Mahajan**[2*] **Kushal Arora**[3†] **Chitta Baral**[1] **Oriana Riva**[2]

[1]Arizona State University [2]Microsoft Research [3]McGill University

Abstract

Humans can learn to operate the user interface (UI) of an application by reading an instruction manual or how-to guide. Along with text, these resources include visual content such as UI screenshots and images of application icons referenced in the text. We explore how to leverage this data to learn generic visio-linguistic representations of UI screens and their components. These representations are useful in many real applications, such as accessibility, voice navigation, and task automation. Prior UI representation models rely on UI metadata (UI trees and accessibility labels), which is often missing, incompletely defined, or not accessible. We avoid such a dependency, and propose *Lexi*, a pre-trained vision and language model designed to handle the unique features of UI screens, including their text richness and context sensitivity. To train Lexi we curate the *UICaption* dataset consisting of 114k UI images paired with descriptions of their functionality. We evaluate Lexi on four tasks: UI action entailment, instruction-based UI image retrieval, grounding referring expressions, and UI entity recognition.

1 Introduction

Over the years, humans have learned to parse application user interfaces (UIs). With only a glance at a previously unseen UI they can understand its elements and identify those relevant to a desired task. To some extent, humans have learned the "UI language". The words of this language are UI elements, which can be visual (an icon), textual (a hyperlink), or both (a text button). If machines could parse and understand the UI language in the same way, they could map natural language commands to UI elements, facilitate access to UIs for visually-impaired users, and ultimately operate UIs on behalf of users.

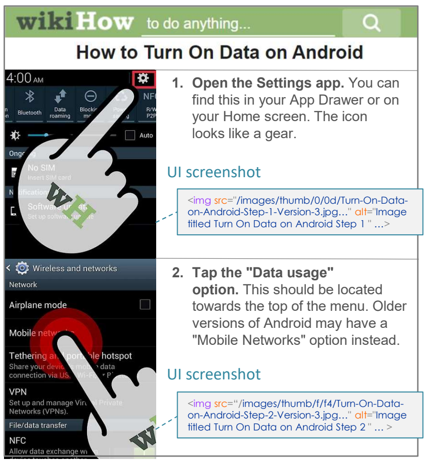
Towards this goal, we are interested in leveraging current vision and language (VL) representation models (e.g., Li et al., 2019; Lu et al., 2019; Chen et al., 2020c) to achieve a generic visio-linguistic feature representation for UIs. However, some key constraints on data availability and data access need to be taken into account. Training VL models requires high-quality paired visio-linguistic datasets. Approaches to collect such datasets for UIs currently involve *(i)* app crawling with human labelling (Li et al., 2020b,a), and *(ii)* use of unlabeled data consisting of UI screens and associated metadata (He et al., 2021; Bai et al., 2021). While the former is hard to scale, in case of the latter, UI metadata, such as accessibility labels and structural representations of a UI screen (referred to as DOM tree in webpages and View Hierarchy in Android) is often missing (Chen et al., 2020a), partially defined, or not accessible for security reasons (on Android, only apps approved as Accessibility tools can access it (XDA, 2021)).

To address these limitations, we propose leveraging a new data source: instruction manuals, tutorials, and how-to guides that abound in the Internet (such as in technical support and how-to websites), exist for many different apps and platforms, and are easy to crawl. In addition to textual instructions, for clarity, they often include visual information. Figure 1 shows two examples where step-by-step instructions from WikiHow include screenshots of the Android UI (left figure) and instructions from the Microsoft Teams manual include images of the referenced icons (middle figure). We posit that these visual and textual resources can be leveraged to learn visually-grounded textual representations of UI screens and their components.

Applying current VL models to such UI data raises some new challenges. VL models are trained using datasets like Conceptual Captions (Sharma et al., 2018), which contain images of the real world (i.e., photographs). In the physical world, objects

[*]The two authors contributed equally.

[†]Work done during an internship at Microsoft Research.



How to Turn On Data on Android

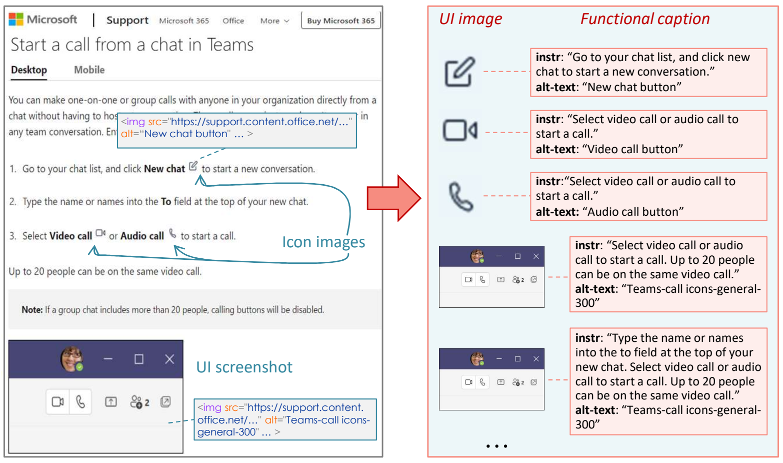
1. **Open the Settings app.** You can find this in your App Drawer or on your Home screen. The icon looks like a gear.

<img src="/images/thumb/0/0d/Turn-On-Data-on-Android-Step-1-Version-3.jpg..." alt="Image titled Turn On Data on Android Step 1 " ...>

2. **Tap the "Data usage" option.** This should be located towards the top of the menu. Older versions of Android may have a "Mobile Networks" option instead.

<img src="/images/thumb/f/f4/Turn-On-Data-on-Android-Step-2-Version-3.jpg..." alt="Image titled Turn On Data on Android Step 2 " ... >

(a) WikiHow Android manual


Microsoft | Support Microsoft 365 Office More Buy Microsoft 365
Desktop
Mobile
Icon images
UI screenshot
UI image
Functional caption

Start a call from a chat in Teams

You can make one-on-one or group calls with anyone in your organization directly from a chat without having to hos... in any team conversation. En...

<img src="https://support.content.office.net/..." alt="New chat button" ... >

1. Go to your chat list, and click **New chat** to start a new conversation.
2. Type the name or names into the **To** field at the top of your new chat.
3. Select **Video call** or **Audio call** to start a call.

Up to 20 people can be on the same video call.

Note: If a group chat includes more than 20 people, calling buttons will be disabled.

<img src="https://support.content.office.net/..." alt="Teams-call icons-general-300" ... >

instr: "Go to your chat list, and click new chat to start a new conversation."
alt-text: "New chat button"

instr: "Select video call or audio call to start a call."
alt-text: "Video call button"

instr:"Select video call or audio call to start a call."
alt-text: "Audio call button"

instr: "Select video call or audio call to start a call. Up to 20 people can be on the same video call."
alt-text: "Teams-call icons-general-300"

instr: "Type the name or names into the to field at the top of your new chat. Select video call or audio call to start a call. Up to 20 people can be on the same video call."
alt-text: "Teams-call icons-general-300"

...

(b) Microsoft Teams manual and generated image-caption pairs

Figure 1: Examples of how-to and support websites. In addition to text, they contain UI screenshots and icon images, which we use to generate image-caption pairs for training Lexi.

of a certain class (cars, bikes, people) share certain appearance features, while the appearance of a UI element is less indicative of its meaning or functionality. UI elements may vary their appearance (in size, color, shape) but have the same functionality, or, vice versa, have different functionality but share similar appearance (e.g., a hand-shaped icon can mean "Raise hand" in a video conference app or "Hand tool" in Photoshop). In other words, the meaning of a UI element is highly *context sensitive*. A second difference is that, compared to photographs, UI screens are *rich in texts*; for some elements, such as text buttons or hyperlinks, text is essential. However, current VL models do not perform text recognition on the visual input. Finally, an out-of-the-box language model like BERT (Devlin et al., 2019) is inadequate to represent manual instructions because *(i)* highly-recurrent technical terms ("click", "tap", "type", etc.) may prevent it from distinguishing instructions, and *(ii)* manuals use terms ("menu", "bar", "button", etc.) that have a very different meaning in the general domain.

To address these challenges we curate a new UI dataset (*UICaption*) and propose *Lexi*, a pre-trained VL model for UI language understanding.

We crawl images of icons and UI screenshots from support and how-to websites, spanning multiple applications and platforms, and create a visio-linguistic dataset by pairing each image with one or multiple captions. To address the context sensitivity of UI elements, we synthesize *functional captions* from instructions co-located with the image in the webpage. Rather than describing the appearance of an image, these captions describe the functionality of one or multiple UI elements, in the *context* of other elements (see Figure 1b).

We build Lexi by replacing the standard object detection methods (e.g., Faster R-CNN (Ren et al., 2015)) used in current VL models with one specifically designed for UI screens. To leverage the text-richness of UIs, we augment an image's region features with textual features of the recognized texts. Finally, to accurately distinguish UI actions, objects, and input parameters in functional captions, we create a new pre-training task based on (noun-verb) POS tagging. We evaluate Lexi on 4 downstream tasks: UI action entailment, instruction-based image retrieval, grounding referring expressions, and entity recognition.

In summary, we make the following contributions: *(i)* a new data source for self-supervised learning of the UI language and a cross-platform dataset of 134k pairs of images and functional captions, which we have released[1]; *(ii)* Lexi, a pre-trained model for UI language understanding which does *not* depend on UI metadata; and *(iii)* an evaluation of Lexi, including baseline comparisons, out of domain performance, and ablation analysis.

2 Related work

VL models. Multiple single- or two-stream transformer-based architectures have been proposed to learn a single feature space from visual and language inputs (e.g., Tan and Bansal, 2019; Lu et al., 2019; Li et al., 2019; Su et al., 2020; Chen et al., 2020c; Qi et al., 2020). Unlike pre-

[1]https://github.com/microsoft/UICaption

training of language models which can use unlimited natural language texts (e.g., Wikipedia), VL models use high-quality paired visio-linguistic datasets, such as Conceptual Captions (Sharma et al., 2018), SBU Captions (Ordonez et al., 2011), and MS COCO (Lin et al., 2014). Our approach extends prior architectures with a vision encoder designed to leverage the text richness of UI images; we revisit VL models in a new problem space, UI language understanding, with a new set of data consisting of UI images and functional captions.

UI representation models. *Supervised* UI understanding methods depend on the collection of interaction traces, app crawling infrastructures, and human labeling. Liu et al. (2018) and Gur et al. (2019) use human demonstrations to train reinforcement learning agents that execute tasks specified in natural language by interacting directly with UIs. Pasupat et al. (2018) and Li et al. (2020a) curate datasets of natural language commands to learn how to ground language into low-level UI elements (buttons, icons, text inputs, etc.) and actions (click, type, select, etc.). All these systems learn at the level of UI elements which can be effective, but only in controlled environments (UI elements do not change over time (Shi et al., 2017)) and for application-specific tasks (Branavan et al., 2009). For better generalization, Mazumder and Riva (2021) map natural language commands to "concept-level" actions, but they require labeled natural language commands and a graph model of the execution environment.

Unsupervised methods include ActionBert (He et al., 2021) and UIBert (Bai et al., 2021), consisting of transformer-based multimodal architectures trained on sequential or single Android UI screens, respectively. While these systems depend on the availability of UI metadata (UI trees and accessibility labels), Lexi assumes only a screenshot is given, thus being widely applicable. By leveraging web data, which can be crawled without requiring app hosting infrastructures, Lexi is trained to handle mobile and desktop UIs from multiple platforms.

Instruction manuals. To the best of our knowledge we are the first using them to learn UI representation models. Yagcioglu et al. (2018) used them to study recipes in a multimodal setting, and Bisk et al. (2020) physical commonsense in a text-only setting. Branavan et al. (2011) used them to inform the strategy of a game-playing agent.

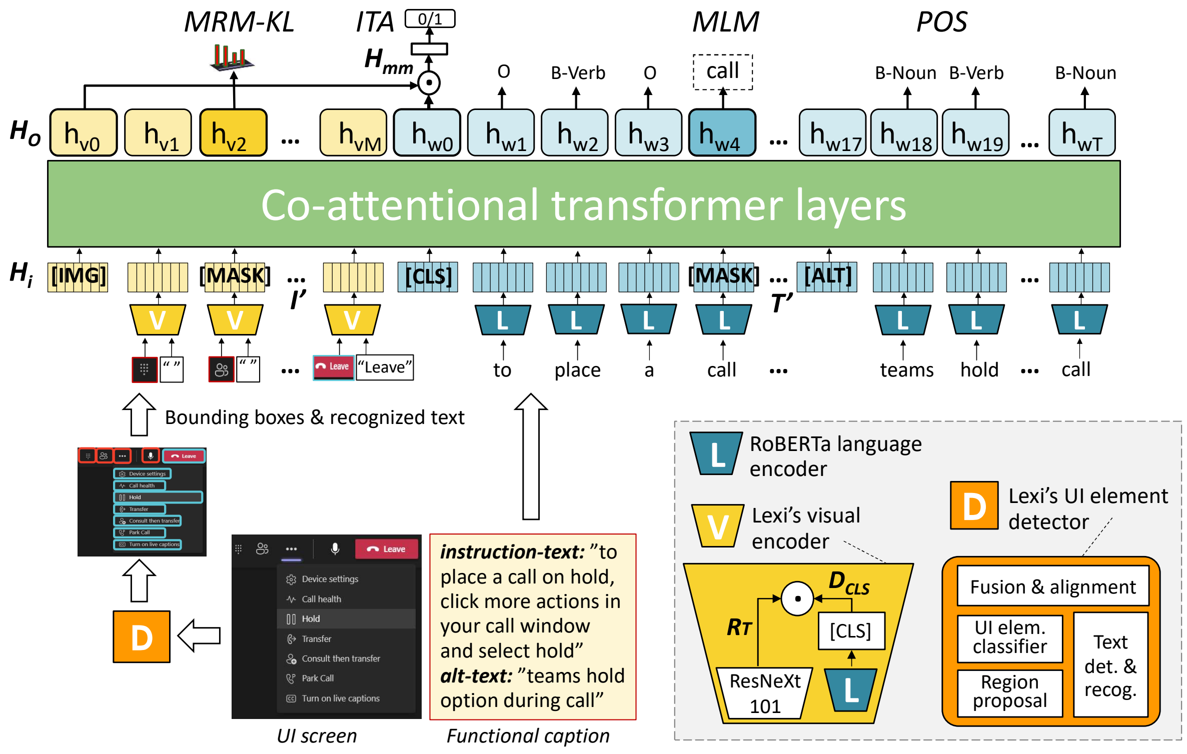
3 UICaption dataset

We curate a new dataset of UI images (screenshots and icons) paired with functional captions.

We collect UI images, alt-text, and instructions accompanying a UI image from three sources: *(i) Instruction manuals* extracted from support websites of 23 Microsoft (MS) apps (Teams, Outlook, Excel, Word, etc.) and Zoom. *(ii) Android support websites*, whose URLs are obtained from the AndroidHowTo dataset (Li et al., 2020a), and *(iii) 58k general tech websites* whose URLs are obtained by submitting search queries containing technical commands (extracted from MS and Android manuals) to Google image search and inspecting the top 60 results. To improve data quality, we perform data filtering, including a simple hash-based and text-based near-duplicate image removal, blurred image removal, and image relevance (e.g., excluding content-unrelated images such as Facebook and Twitter icons).

For each collected image, we use the text surrounding it in the webpage to generate from 1 up to 5 **functional captions** (the number depends on the length and structure of the surrounding text). If available, we also append the alt-text. Rather than describing the appearance of an image, functional captions describe the functionality of one or multiple UI elements, in the *context* of other elements. *Textual* context may translate into *spatial* context because the instructions to accomplish a task often involve co-located UI elements. For example, in Figure 1b, the "Video call" and "Audio call" icons are associated with the caption "Select Video call or Audio call to start a call"; the instruction defines a logical relationship between the two functions, which corresponds to a spatial relationship, i.e., the two icons are next to each other in the UI.

We thus obtain the **UICaption dataset** consisting of 113,971 unique UI images and 133,817 image-caption pairs. This dataset significantly differs from those used in prior UI understanding models (He et al., 2021; Bai et al., 2021). The data is extremely varied in terms of applications (Excel, GMail, Chrome, Photos, etc.), platforms (mobile & desktop), number of UI elements (from single icons to complete desktops), text density (Word documents, photos, or Excel spreadsheets), pixel density, and quality (e.g., screenshots may not be pristine or may have been annotated, such as in Figure 1a). More details on the dataset and the crawling process can be found in Appendix A.



instruction-text: "to place a call on hold, click more actions in your call window and select hold"
alt-text: "teams hold option during call"

Figure 2: Our proposed VL transformer encoder, trained using 4 pre-training tasks: masked language modeling, masked region modeling with KL divergence, image-text alignment, and POS tagging.

4 Method

Lexi follows the two-stream transformer-based architecture of various VL models (Lu et al., 2019; Tan and Bansal, 2019). As illustrated in Figure 2, it consists of an image encoder V, a text encoder L, and co-attentional transformer layers for cross-modal representation learning. Given a text input T represented as a set of word tokens $w_1, ..., w_T$ and an image I represented as a set of region features $v_1, ..., v_M$ augmented with their corresponding detected texts, Lexi outputs final representations $h_{v0}, ..., h_{vM}$ for the visual stream and $h_{w0}, ..., h_{wT}$ for the textual stream. The most distinct aspect of Lexi is the early-infusion of detected text vector representations with corresponding visual features, which is motivated by the frequent presence of text in UI images. Moreover, the vocabulary of functional captions including technical action verbs and specific named entities motivates an additional self-supervision task for pre-training.

UI image representation. An input image I is first resized to a fixed size (224×224), and encoded by extracting bounding boxes of its regions of interest and corresponding visual features v. To this end, VL architectures (Lu et al., 2019) rely on pre-trained object detectors (e.g., Faster R-CNN (Ren et al., 2015) pre-trained on Visual Genome (Krishna et al., 2017)). Pre-training (or fine-tuning) object detectors for UI images requires large-scale datasets of annotated UI screens, which are generally not available.[2] Inspired by Chen et al. (2020b)'s work, we implement a UI element detection framework, the *Lexi detector*, which can handle the cross-platform and cross-app nature of the UI images in our dataset, without requiring annotated UI screens. The design is optimized to leverage the regular layouts and text richness of UI screens. In short, it works as follows. It detects bounding boxes of UI elements using standard computer vision techniques (Canny edge detection (Canny, 1986), contour detection (Suzuki and be, 1985), non-maximum suppression) and classifies them using a ResNeXt-101 model fine-tuned on a 9-class dataset of 263k mobile/desktop UI elements.[3] In parallel, it uses OCR (PP-OCR (Du et al., 2020)) to detect text elements and recognize their text. Then, it combines the two types of prediction by merging boxes based on alignment and distance, to output a set of classified bounding

[2]Only the Android Rico dataset exists (Deka et al., 2017).
[3]We derive the dataset from icon collections (Google, 2021) and mobile app datasets (Deka et al., 2017), and through data synthesis based on UI markup files of Windows apps.

boxes with recognized text, if any. (Appendix B provides further details on the implementation.)

For each image I, Lexi's vision encoder (Figure 2 bottom-right) computes region features using our fine-tuned ResNeXt-101 model. It keeps a maximum of 64 high-scoring boxes, and for each selected region i, it computes v_i (of size 2048) as the mean-pooled convolutional feature from i. In state-of-the-art VL models, text appearing in an input image is not directly extracted. To leverage the text richness of UI screens, each v_i is augmented with textual features of the recognized text (computed as dot-product). Textual features help align with functional captions (as demonstrated by the performance of our baseline ViLBERT-UI-dt, §5). Finally, we encode the spatial location consisting of region position and fraction of the image area covered (not shown in Figure 2). This is projected to match the dimension of the visual/textual features and they are summed. We mark the beginning of the image stream with a special [IMG] token representing the entire image.

UI text representation. Text is encoded as in BERT (Devlin et al., 2019), as a sequence of word tokens pre-pended with the [CLS] token. We concatenate the image's associated instructions and alt-text using a new special token [ALT]. For a given word token, the input representation is a sum of a token, segment, and position embedding. The text encoder is initialized using RoBERTa (Liu et al., 2019). We tokenize the text input using a byte-level BPE (Sennrich et al., 2016) tokenizer.

Formally, given the text $T = w_1, ..., w_T$, the image $I = v_1, ..., v_M$ and the corresponding detected texts $D_v = d_{v1}, ..., d_{vM}$, the image encoder V and the text encoder L, Lexi is represented as follows:

$$I' = V(v_i) = D_{\text{CLS}}(v_i) \cdot \textsc{Rt}(v_i), \;\; T' = L(w_i)$$
$$H_i = [\text{IMG}] : I' : T'$$
$$H_o = \textsc{Co-Att}(H_i)$$
$$H_{mm} = H_{v0} \cdot H_{w0}$$

where, $D_{CLS}(v_i) = L(d_{vi})$ is the [CLS] token representation of the detected text in image I, RT is the fine-tuned ResNeXt-101 visual feature extractor whose weights are frozen, T' is the encoded text input, H_i is the final input vector using concatenation operator :, and H_o is the final output vector after the co-attentional transformer layers CO-ATT. CO-ATT takes as input the vector representations of the textual and visual streams projected to equal dimensions using linear layers. CO-ATT is borrowed from ViLBERT, in which the keys and values of an attention-block from each modality are passed as input to the other modality's multi-headed attention. This leads to attention-pooled features for each modality conditioned on the other, i.e, language-conditioned visual features and visual-conditioned language features. H_{mm} is the final multi-modal hidden representation after a dot-product between the output representation of [IMG] of the visual input I and [CLS] of the textual input T. The hidden representations from H_o are further fed through respective task-specific feed-forward layers.

Pre-training tasks. We train Lexi using four pre-training tasks, three borrowed from prior work (Devlin et al., 2019; Lu et al., 2019) and one new.

Masked Language Modeling (MLM). We borrow this task from BERT. We randomly mask out input words with 15% probability, and replace them with the [MASK] token in the text stream T'. We task the model with predicting them by leveraging the surrounding words and the visual input. We use standard cross-entropy loss for this task.

Masked Region Modeling with KL Divergence (MRM-KL). We mask an image's regions with 50% probability (and only if the image contains at least 4 regions). To mask image regions, we replace their features with zeros – we mask 10% of the features and retain 90% of them. If a masked region contains text, the detected text is automatically masked out too. The model is tasked with reconstructing the masked regions given the remaining regions and the input words. However, the model is unlikely to be able to reconstruct exact image features. Hence, as in ViLBERT, the model predicts a distribution over semantic classes for the corresponding image region, and minimizes the KL divergence between the original class probabilities of the unmasked region and the predicted class probabilities. We use cross-entropy loss also for this task.

Image-Text Alignment (ITA). This task is useful for learning representations for downstream cross-modal tasks. It is a binary classification task: the cross-modal representation H_{mm} is fed into a feed-forward layer to predict a score between 0 and 1. In training, we sample a positive or negative pair at each step. The negative pair is created by pairing the true image with a random caption that is at $[0.5 - 0.9]$ distance away in cosine-similarity with the true caption – due to the prevalence of common technical terms, functional captions tend to be semantically very similar. We apply binary cross-entropy loss to this task.

POS Tagging (POS). A challenge with our text embeddings is that our text inputs have many technical terms, which makes it hard to distinguish them. Some terms ("home","bar", etc.) also have a different meaning in general domain. Moreover, in current VL models the attention in the cross-modality layers focuses on nouns and pronouns because they are the most informative words in current datasets/tasks (Tan and Bansal, 2019), but in our domain verbs play a key role. To help the model learn the "foreign" UI language and attend its syntax correctly, we introduce weakly-supervised part-of-speech (POS) tagging as a new pre-training task, where the model must predict noun and verb spans present in the text stream. Prior work uses POS features as input to a neural network (Kiperwasser and Goldberg, 2016), however, we aim to encode this information in the learned representations during pre-training. We use spaCy's POS tagger's predictions as weak-supervision labels for learning the task, and adopt the BIO scheme. Each token representation is classified as the beginning (B) or the inside (I) of a noun or verb entity, or as other (O), i.e., a non-entity token. We use cross-entropy loss.

In summary, the total pre-training loss $\mathcal{L}$ with scalar coefficients $\alpha, \beta, \gamma, \delta \in (0, 1]$ is given by:

$$\mathcal{L} = \alpha \cdot \mathcal{L}_{MLM} + \beta \cdot \mathcal{L}_{MRM-KL} + \gamma \cdot \mathcal{L}_{ITA} + \delta \cdot \mathcal{L}_{POS}$$

where $\mathcal{L}_x$ is the loss for the pre-training task x.

5 Evaluation on downstream tasks

We evaluate Lexi through 4 downstream tasks. In this section, we describe our experimental setup, introduce datasets and baselines, describe each task's settings and results, and provide an ablation study.

5.1 Experimental setup

To pre-train Lexi we apply the four tasks described in §4 to the UICaption dataset, excluding the Zoom and Teams portions which are used to conduct an out-of-domain (OOD) analysis. We create training (103k UI images, 116k image-text pairs), validation (5k images, 5.6k image-text pairs), and test (6k images, 7k image-text pairs) splits by ensuring no overlaps between UI images and functional captions. In fine-tuning we use a subset of the pre-training dataset and two Android datasets, UIBert (Bai et al., 2021) and AndroidHowTo (Li et al., 2020a).

We test two variants of Lexi with 6 (**Lexi-6**) or 10 (**Lexi-10**) transformer layers in the visual stream. We train on 8 Nvidia V100 GPUs for 100 epochs with a batch size of 128 and a learning rate of 4e-5. We use the Adam optimizer (Kingma and Ba, 2014). For more details see Appendix C.

5.2 Baseline models

We compare Lexi against three baselines. We modify the ViLBERT architecture to encode the input image using the Lexi detector (instead of Faster R-CNN which is not trained on UI images) and the input text using RoBERTa. We test two variants of this baseline depending on whether the text detected in the image is encoded (**ViLBERT-UI-dt**) or not (**ViLBERT-UI**). The third baseline, **ViLBERT-12-1-UI**, is a multi-task ViLBERT model trained on 12 different datasets (4.5M instances) and 6 tasks (Lu et al., 2020); it uses the Lexi detector for image encoding and BERT for text. We fine tune all baselines on our fine-tuning datasets. In the entity recognition task (§5.6), we introduce also a text-only baseline, **RoBERTa-UI**, the RoBERTa model fine-tuned on our dataset. We do not consider single-modality baselines in the other tasks as they cannot produce an output unless both types of input are given. The ActionBert (He et al., 2021) and UIBert (Bai et al., 2021) models require UI metadata (Android View Hierarchy and accessibility tags) as input, hence they are not directly comparable with Lexi (and their source code is not released). However, wherever appropriate, we refer to their performance as upper bounds.

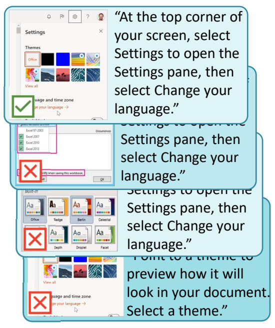
We test in zero-shot (ZS), fine-tuned (FT), and OOD settings. In ZS we directly apply the pre-trained model (i.e., the model is new to the task) to UI data not used in pre-training. In OOD, we test on data from the Zoom and Teams manuals. Both these apps are excluded from pre-training and fine-tuning, and, in general, when crawling UICaption no video conferencing app was specified in the search queries, making this app category completely unknown to the pre-trained model.

5.3 UI action entailment

This task is inspired by the visual entailment task (Xie et al., 2019). Given a natural language instruction (e.g., "Select video call to start a call") and an image (screenshot or icon), the goal is to predict whether the described action can be performed on the image. This task has a practical use in screen readers to locate UI components by their name/functionality in the absence of accessibility labels and alt-texts, as it is often the case.

Model	UI action entailment			Instruction-based UI image retrieval								
	UICaption	T (OOD)	Z (OOD)	UICaption			T (OOD)			Z (OOD)		
	Acc %	Acc %	Acc %	R1	R5	R10	R1	R5	R10	R1	R5	R10
ViLBERT-UI	65.0	31.6	30.7	12.9	42.7	59.3	4.1	17.0	32.0	3.3	16.8	29.5
ViLBERT-UI-dt	68.0	31.8	34.2	14.4	47.1	64.2	5.1	17.4	33.8	3.5	17.6	31.1
ViLBERT-12-1-UI	66.4	33.8	27.6	19.0	52.4	72.7	3.4	16.2	32.0	2.9	12.5	26.8
Lexi-6 (ZS)	52.2	-	-	12.4	21.7	29.3	-	-	-	-	-	-
Lexi-10 (ZS)	57.4	-	-	15.0	22.2	29.0	-	-	-	-	-	-
Lexi-6 (FT)	83.3	**40.1**	**39.6**	34.7	78.4	89.1	**6.5**	25.1	44.3	4.7	**22.7**	**39.9**
Lexi-10 (FT)	**84.1**	39.3	36.1	**36.7**	**80.0**	**91.8**	4.7	**27.5**	**45.8**	**5.3**	21.5	38.2

Table 1: Performance of Lexi on the tasks of UI action entailment and instruction-based UI image retrieval. Models are used zero-shot (ZS) or fine-tuned (FT) on a 77k subset of UICaption. OOD tests are performed on Teams (T) and Zoom (Z).



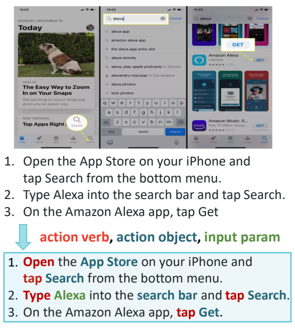
"At the top corner of your screen, select Settings to open the Settings pane, then select Change your language."

Settings pane, then select Change your language."

Settings pane, then select Change your language."

preview how it will look in your document. Select a theme."

(a) Action entailment task.



1. Open the App Store on your iPhone and tap Search from the bottom menu.
2. Type Alexa into the search bar and tap Search.
3. On the Amazon Alexa app, tap Get

action verb, action object, input param

1. Open the App Store on your iPhone and tap Search from the bottom menu.
2. Type Alexa into the search bar and tap Search.
3. On the Amazon Alexa app, tap Get.

(b) Entity recognition task.

Figure 3: Examples of two Lexi's downstream tasks.

While this task is similar to the ITA task used in pre-training, it is a useful diagnostic task to verify Lexi has been pre-trained effectively.

In fine-tuning we use a 4-way multiple-choice setting (Figure 3a). Given an instruction and UI image (the true pair), we generate 3 negative pairs by sampling a hard negative from the 1,000 most similar images to the target image[4], by substituting the true instruction with a random one and the true image with a random one. For each option we compute the alignment score (as in pre-training) and apply a softmax. To make the task more challenging but also more realistic, the image's alt-text is not provided in the input. For training we use a subset of the UICaption dataset used in pre-training (train:77k, val:3.7k, test: 6k[5]).

As Table 1 shows, Lexi (FT) largely outperforms the strongest baseline by 24% demonstrating the efficacy of our pre-training tasks. In ZS settings, Lexi achieves 57.4% accuracy, demonstrating that the model has developed some ability to ground text without requiring any fine-tuning and without relying on the image's alt-text. We test OOD using 1.7k pairs from Teams (T) and 2.5k pairs from Zoom (Z), which were not included in fine-tuning and pre-training. Lexi achieves 39.6–40.1% accuracy (a 16–19% gain over baselines). Lexi-6 outperforms Lexi-10 in the OOD settings, suggesting the model may be able to generalize better with a smaller network depth. In Appendix D, we provide a qualitative analysis of the OOD predictions.

5.4 Instruction-based UI image retrieval

This task is inspired by the image retrieval task used to test VL models. The goal is to identify a UI image from a pool of 50 given a description of its associated instruction(s). We fine-tune the models using a 4-way multiple-choice setting as for UI action entailment (but with alt-text included) on the same 77k subset of UICaption. In-domain and out-of-domain test sets are as in the UI action entailment task.

In Table 1 we report recall@k with k=1,5,10. Our pre-training tasks are effective with significant gains in the FT settings in in-domain tests (1.9$\times$ higher R@1 than the ViLBERT-12-1-UI baseline). While the OOD performance is lower, compared to the strongest baseline R@1 is still 1.4–1.8 percentage points higher. The best ZS performance, achieved by Lexi-10, is inferior to that of the fine-tuned baseline models, indicating it is challenging for the pre-trained model to generalize to this new task. As observed in the previous task, increasing the number of visual layers has a positive effect on in in-domain tests, but the OOD performance of Lexi-6 is better than that of Lexi-10 in some cases.

[4]obtained based on the Euclidean distance of their embeddings, and by discarding images whose captions are contained in the true caption or have a similarity below 0.5 or above 0.9.

[5]The test set was not seen in pre-training.

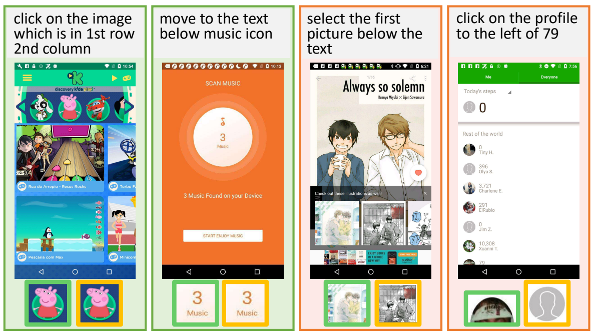

click on the image which is in 1st row 2nd column
move to the text below music icon
select the first picture below the text
click on the profile to the left of 79
Always so solemn
SCAN MUSIC
3 Music Found on your Device
START ENJOY MUSIC
3
Music

Figure 4: Examples of correct (first two) and wrong (last two) predictions for the grounding referring expressions task. The referring expression is above the UI screen, and ground truth UI elements (with a green border) and Lexi's predictions (with an orange border) are below.

5.5 Grounding referring expressions

We borrow this task from ActionBert and UIBert. Given a referring expression in natural language and a UI screen, the goal is to select the UI element referenced by the expression from a set of elements detected in the screen. This task is relevant to voice-guided app navigation agents, where users may issue commands such as "click home at the top of the page" or "play the third song in the list".

Given the referring expression, we re-rank a set of image region proposals. We pass the final representation $h_v i$ for each region v_i into a learned linear layer to predict a matching score for fine-tuning. As we are given pre-computed image regions, the highest-scoring region is used as the prediction. We train using cross-entropy loss on the UIBert dataset, using the official splits (train:15.6k, val:448, test:544). On average, for each expression the model must choose from 20 candidate UI elements.

Lexi outperforms the best baseline by 6.4% (Table 2). If accessibility labels had to be available, the performance would likely increase, as demonstrated by UIBert that, with UI metadata, achieves 90.8% accuracy. Figure 4 provides a qualitative analysis of some predictions (2 correct and 2 wrong). The task is challenging, and the model makes reasonable errors. For instance, for the third referring expression, Lexi correctly returns a picture, but the second rather than the first one under the text. In the last example, Lexi correctly returns a profile icon but not the one next to the text "79", which, slightly cropped at the very bottom of the screen, may be hard to locate also for a human .

	Ref Exp	UI Entity Rec		
Model	UIBert	MS+Android		
	Acc %	Prec	Recall	F1
RoBERTa-UI	-	75.2	80.2	77.2
ViLBERT-UI	51.7	75.7	81.0	77.8
ViLBERT-UI-dt	58.5	76.5	80.1	77.8
ViLBERT-12-1-UI	62.9	79.0	80.4	79.2
Lexi-6	**66.9**	79.0	**82.1**	**80.2**
Lexi-10	65.8	**79.1**	81.7	80.0

Table 2: Performance of Lexi on the grounding referring expressions and UI entity recognition tasks. Models are fine-tuned on UIBert and MS +Android, respectively.

5.6 UI entity recognition

This task is relevant to task completion systems, to translate natural language commands into executable scripts, and was inspired by the AndroidHowTo dataset (Li et al., 2020a).[6] Given a UI image and an instruction, the goal is to extract spans of *UI action verbs*, *UI action objects*, and *input parameters* (see example in Figure 3b). UI action verbs describe operations (click, type, etc.) that a user can take on a UI object (a button or a menu); some UI objects (a text field) require input parameters (e.g., "Enter *a* in b"). A challenge here is that not all instances of a word are of the same entity type (in Figure 3b, "Alexa" is an input, but "Alexa app" is not; "Search" is an object, but may be a verb elsewhere). We model the task as a named-entity recognition task with Begin (B), Intermediate (I), and Other (O) class annotations (BIO scheme). We use a cross-entropy loss function.

We build a dataset for this task as follows. We ask 4 annotators to label the span of words that describe these entity types in instructions extracted from the manuals of 14 MS apps (more details on the process in Appendix E). We combine our annotated instructions with similarly-annotated AndroidHowTo commands adapted to our task (Appendix F) to obtain 43.4k unique image-instruction pairs, for a total of 76.7k action verbs, 79.3k objects, and 1.5k inputs labelled. We refer to this dataset as MS+Android. For training and testing we use an 80-10-10 scheme.

As Table 2 shows, Lexi outperforms the RoBERTa baseline by 3.9% and even the very large ViLBERT-12-1-UI model by 1.3% (based on F1

[6]In the tests we do not consider their model as a baseline because, although related, its goal and metrics are different (entity extraction vs. sequence generation, see Appendix F for more details).

Model	UI action ent Acc %	UI entity rec F1
ViLBERT-UI	65.0	77.8
ViLBERT-UI-dt	68.0	77.8
ViLBERT-12-1-UI	66.4	79.2
Lexi-10	**84.1**	80.0
Lexi-6	83.3	**80.2**
– MLM	83.2	78.6
– MRM-KL	81.7	78.2
– ITA	70.7	79.1
– POS	82.8	78.2
– DetText	75.8	79.2

Table 3: Ablation study of Lexi with respect to the UI action entailment and UI entity recognition tasks.

scores), thus demonstrating the effectiveness of pre-training with POS tagging. Learning with a multi-modal setup improves performance, suggesting the visual input provides further necessary context.

5.7 Ablation analysis

We perform an ablation analysis where Lexi-6 does not include one of our four pre-training tasks or does not use the text detected in the input image. All Lexi variants are pre-trained as the complete Lexi model. In the analysis we use the UI action entailment and UI entity recognition tasks, for which models are fine-tuned on a subset of UICaption (77k) and MS+Android, respectively. As Table 3 shows, among the 4 pre-training tasks, ITA is the most relevant to UI action entailment (its absence causes a 17.8% drop in Lexi-6's accuracy) while MLM and POS are the least relevant. On the other hand, for UI entity recognition, Lexi-6 slightly outperforms Lexi-10, and we observe how POS and MRM-KL tasks are important to achieve high performance (their absence causes a 2.6% drop in F1 scores). The inclusion of the detected text is most beneficial to UI action entailment.

6 Limitations

In this section, we describe some limitations of our method, dataset, and data collection approach.

Our approach depends on pre-training using publicly available web data, which is crawled at scale and for a diverse set of applications. Web data is highly skewed towards popular applications; less popular application might have less or lower-quality documentation and images online. In the future, we will focus on collecting UI data from other sources, such as instructional videos, online tutorials, and bug reports. We also observe that for certain application categories, such as shopping, transportation, weather, or news, instruction manuals are rare or totally missing.

Another limitation that arises from curating pre-training data from the web is how to ensure zero overlap between pre-training and downstream evaluation tasks. A URL-based filter of the crawled websites does not eliminate overlaps as aggregator websites like wikiHow and LifeWire may contain manuals of different apps and platforms. Partitioning data based on application names is also difficult as the name of the application is not explicitly encoded in the webpage source.

Our data and method have been evaluated for English instructions and UIs that contain text in English. Future work should expand to different languages. Currently, the method is trained and evaluated on instructions whose length is at most 512 tokens.

Pre-training the model even on a modest dataset like UICaption already requires significant GPU resources. The obtained model is not optimized to run on edge devices like mobile phones, thus possibly raising privacy concerns. For some scenarios that motivate this work (e.g., accessibility support and voice-based navigation in mobile apps) it would be highly desirable to execute and update the models locally.

7 Conclusion

We propose to learn generic representation models for the UI language from instruction manuals and how-to guides. We apply prior VL models to this data and encounter new challenges due to the text richness and context sensitivity of UI screens. To address them, we build a new dataset based on the concept of functional caption, design a new vision encoder which leverages text detected in UIs, and create a new POS pre-training task. Tested on four downstream tasks, Lexi largely outperforms our baselines, including a multi-task ViLBERT model trained on much larger datasets.

Acknowledgements

We thank Sahisnu Mazumder for early discussion on the Lexi project and for implementing the first prototype of the UICaption web crawler. We thank Weiwei Yang for her feedback on the Lexi architecture, and Kate Lytvynets for helping implement the Lexi UI element detector.

References

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and Blaise Agüera y Arcas. 2021. UIBert: Learning generic multimodal representations for UI understanding. In *Proc. of the 30th International Joint Conference on Artificial Intelligence*, IJCAI'21, pages 1705–1712.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Proc. of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439.

S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement Learning for Mapping Instructions to Actions. In *Proc. of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, ACL '09, pages 82–90. Association for Computational Linguistics.

S. R. K. Branavan, David Silver, and Regina Barzilay. 2011. Learning to Win by Reading Manuals in a Monte-Carlo Framework. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 268–277.

John Canny. 1986. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698.

Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020a. Unblind Your Apps: Predicting Natural-Language Labels for Mobile GUI Components by Deep Learning. In *Proc. of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, pages 322–334.

Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. 2020b. Object Detection for Graphical User Interface: Old Fashioned or Deep Learning or a Combination? In *Proc. of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, pages 1202–1214. ACM.

Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. 2020c. UNITER: UNiversal Image-TExt Representation Learning. In *Computer Vision - ECCV 2020 - 16th European Conference*, volume 12375 of *LNCS*, pages 104–120. Springer.

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proc. of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 845–854. ACM.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, NAACL'19, pages 4171–4186. Association for Computational Linguistics.

Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, et al. 2020. PP-OCR: A practical ultra lightweight OCR system. *arXiv preprint arXiv:2009.09941*.

Google. 2021. Material design icons. https://github.com/google/material-design-icons.

Izzeddin Gur, Ulrich Rückert, Aleksandra Faust, and Dilek Hakkani-Tür. 2019. Learning to navigate the web. In *7th International Conference on Learning Representations (ICLR '19)*.

Zecheng He, Srinivas Sunkara, Xiaoxue Zang, Ying Xu, Lijuan Liu, Nevan Wichers, Gabriel Schubiner, Ruby B. Lee, and Jindong Chen. 2021. ActionBert: Leveraging User Actions for Semantic Understanding of User Interfaces. In *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 5931–5938.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. 2017. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision*, 123(1):32–73.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. 2019. VisualBERT: A Simple and Performant Baseline for Vision and Language. *CoRR*, abs/1908.03557.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020a. Mapping Natural Language Instructions to Mobile UI Action Sequences. In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics*, ACL 2020, pages 8198–8210. Association for Computational Linguistics.

Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020b. Widget Captioning: Generating Natural Language Description for Mobile User Interface Elements. In *Proc. of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP'20)*.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. In *6th International Conference on Learning Representations (ICLR '18)*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. In *Proc. of the Annual Conference on Neural Information Processing Systems 2019*, NIPS'19, pages 13–23.

Jiasen Lu, Vedanuj Goswami, Marcus Rohrbach, Devi Parikh, and Stefan Lee. 2020. 12-in-1: Multi-Task Vision and Language Representation Learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pages 10434–10443.

Sahisnu Mazumder and Oriana Riva. 2021. FLIN: A flexible natural language interface for web navigation. In *Proc. of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL 2021, pages 2777–2788. Association for Computational Linguistics.

Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2020. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221.

Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse engineering mobile application user interfaces with REMAUI. In *Proc. of the 30th IEEE/ACM International Conference on Automated Software Engineering*, ASE '15, pages 248–259. IEEE Press.

Vicente Ordonez, Girish Kulkarni, and Tamara L Berg. 2011. Im2Text: Describing images using 1 million captioned photographs. In *Proc. of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 1143–1151. Curran Associates Inc.

Panupong Pasupat, Tian-Shun Jiang, Evan Zheran Liu, Kelvin Guu, and Percy Liang. 2018. Mapping natural language commands to web elements. In *Proc. of the 2018 Conference on Empirical Methods in Natural Language Processing*, EMNLP'18, pages 4970–4976. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proc of the Annual Conference on Neural Information Processing Systems*, NIPS'19, pages 8024–8035.

Di Qi, Lin Su, Jia Song, Edward Cui, Taroon Bharti, and Arun Sacheti. 2020. ImageBERT: Cross-modal Pretraining with Large-scale Weak-supervised Image-Text Data. *CoRR*, abs/2001.07966.

Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2015. Faster R-CNN: towards real-time object detection with region proposal networks. In *Proc. of the Annual Conference on Neural Information Processing Systems*, NIPS'15, pages 91–99.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics.

Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. 2018. Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning. In *Proc. of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2556–2565. Association for Computational Linguistics.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of Bits: An Open-Domain Platform for Web-Based Agents. In *Proc. of the 34th International Conference on Machine Learning*, ICML '17, pages 3135–3144.

Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. 2020. VL-BERT: pre-training of generic visual-linguistic representations. In *8th International Conference on Learning Representations*, ICLR'20.

Satoshi Suzuki and KeiichiA be. 1985. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46.

Hao Tan and Mohit Bansal. 2019. LXMERT: Learning cross-modality encoder representations from transformers. In *Proc. of the 2019 Conference on Empirical Methods in Natural Language Processing and the*

9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 5100–5111. Association for Computational Linguistics.

XDA. 2021. Google is trying to limit what apps can use an Accessibility Service (again). https://www.xda-developers.com/google-trying-limit-apps-accessibility-service/.

Ning Xie, Farley Lai, Derek Doran, and Asim Kadav. 2019. Visual entailment: A novel task for fine-grained image understanding. *ArXiv*, abs/1901.06706.

Semih Yagcioglu, Aykut Erdem, Erkut Erdem, and Nazli Ikizler-Cinbis. 2018. RecipeQA: A challenge dataset for multimodal comprehension of cooking recipes. In *Proc. of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1358–1368. Association for Computational Linguistics.

Ethical considerations

One possible application of our technology are screen readers for visually-impaired users. As accessibility labels are often missing or incomplete, Lexi could give them access to a much wider range of applications. In this regard, it is important that the pre-training dataset guarantees an unbiased coverage of applications and platforms. Another potential use case of Lexi is task automation, which has societal and security implications: What if an agent clicks the wrong button? Is there an "undo" option or other recourse? A significant frontier for intelligent task completion systems is architecting them such that they can increase human productivity, yet remain amenable to human review.

To test one of our downstream tasks, we asked four annotators to label instruction manuals of Microsoft Office applications. The annotations provided (i.e., spans of verb, object, and input entities) contain little subjectivity and ambiguity. More details on how this dataset was curated can be found in Appendix E.

We have released the UICaption dataset. In line with existing pre-training data sharing policy (Sharma et al., 2018; Li et al., 2020a), we have released the collection of URLs from which we crawled the image-text pairs and the associated scripts to generate training samples.

A UI data collection

As described in §3, to curate the UICaption dataset, we collect UI images from three sources. In the following we provide more details on the first (instruction manuals) and third source (general technical websites). We also describe how functional captions were generated.

Instruction manual crawlers. We built crawlers that use rules specific to a support website to extract comprehensive instruction manuals consisting of sets of instructions along with image occurrences, organized by topic and functionality. We built custom crawlers for 23 Microsoft products, which we collectively call MS apps, whose technical manuals can be found at support.microsoft.com, and for Zoom, whose manual can be found at zoom.com.

Mixed data crawler. We also built a crawler that uses general heuristics to extract UI images along with their alt-text and surrounding texts from a variety of support and how-to websites, for different apps and platforms (hence the name "mixed"). The crawler first collects URLs of technical websites and how-to guides by synthesizing search queries and submitting them to web search.

Search queries are generated in two ways. In one approach, we extract names of UI elements (e.g., "Home button", "Settings icon") from the crawled instruction manuals of MS apps (where UI elements are easily recognizable from their HTML classname attribute) and concatenate them with their corresponding app name (e.g., "Home button, Word"). Note that to create an OOD test set, in this web crawl we avoid using names of UI elements and instructions from Teams and Zoom manuals, although overlaps due to common UI elements and instructions may occur. In a second approach, we split multi-step PixelHelp (Li et al., 2020a) instructions crawled from support.google.com/pixelphone into single-step instructions. PixelHelp instructions relate to four task categories: configuring accounts, Gmail, Chrome, and Photos.

Once search queries are generated, the mixed data crawler submits them to Google image search and parses the top 60 results. It then visits the URL of each result, and extracts images (using HTML tags) as well as preceding and subsequent text.

We found this approach to work well for two key reasons. First, by carefully selecting the source webpages, the process ensured that the large majority of the collected images were indeed icons and UI screenshots. Second, although our seed search queries were specific to MS apps and PixelPhone, they were general enough (e.g., "Account icon OneDrive" derived from the OneDrive manual or "Tap where you can enter text" derived from

PixelHelp) to trigger results for a much larger set of apps and platforms (e.g., Apple and web).

Caption generation. For each crawled UI image we generated from 1 up to 5 captions following simple heuristics. If the preceding text contains a set of numbered instructions we split them into individual steps, and generate captions that contain from 1 up to 5 steps. Otherwise, the entire preceding text goes into one caption. For Android support websites obtained from AndroidHowTo (Li et al., 2020a), we used preceding or subsequent text, based on the annotations provided in the AndroidHowTo dataset. We discarded image-text pairs where the number of words in the text was less than 4 or more than 160. Overall, the average length of captions is 32 words, the average length of alt-text is 4.5 and the average length of instruction text is 29.

UICaption dataset summary. Overall, the UICaption dataset consists of 134k image-caption pairs. Based on the crawling source we distinguish three portions of the dataset: 1) **Mixed** (104k pairs), containing UI data for different apps and platforms, obtained by crawling general technical websites; 2) **Android** (5k pairs), covering Android apps, obtained by crawling support websites listed in AndroidHowTo; and 3) **MS** (12k pairs), **Teams** (1.7k pairs), and **Zoom** (2.5k pairs), covering various MS apps and Zoom, obtained using custom instruction manual crawlers. The third category contains the highest quality data in terms of image-text alignment and instruction details. Teams and Zoom are used only in testing, for an OOD analysis.

B Lexi UI element detector

The design of the Lexi detector is inspired by prior work (Chen et al., 2020b; Nguyen and Csallner, 2015; Moran et al., 2020) that demonstrated how off-the-shelf computer vision algorithms coupled with DNN classifiers can be successfully applied to UI screen understanding, without requiring expensive datasets of labeled UI screens.

The Lexi detector consists of two separate detection streams and a fusion layer. In one stream, it detects the edges of an image's elements using Canny's algorithm (Canny, 1986), applied to a greyscale map of the input image. After a dilation with a 8×8 rectangle kernel, it applies contour detection (Suzuki and be, 1985), and then approximates contours to polygons to obtain rectangular bounding boxes. Finally, it uses non-maximum suppression (with an overlap threshold of 75%) to eliminate overlapping boxes, and it discards boxes whose area is less than 50 pixels or whose width or height is smaller than 5 pixels. However, fully-contained boxes are not eliminated in this stage (e.g., we do not want to eliminate an icon contained in a text button). In a parallel stream, the Lexi detector runs PP-OCR (Du et al., 2020) on the input image[7], followed by another round of non-maximum suppression, this time only for text boxes. In the final fusion stage, the detector merges contour-identified boxes with OCR-inferred boxes. Inspired by REMAUI's approach (Moran et al., 2020), we use the following heuristics: (i) we merge bounding boxes which are vertically or horizontally aligned and separated by no more than 2 pixels, (ii) we merge OCR-inferred boxes with contour-inferred boxes which fully contain them and are of similar area, and (iii) we eliminate contour-detected boxes contained in OCR-inferred boxes only if they are either smaller than 250 pixels or they are at least 85% as big as the containing text box.

Once bounding boxes have been identified, they are fed to a UI classifier. We fine-tune ResNeXt-101 using a collection of 263k mobile/web/Windows UI elements (obtained through data synthesis, by cropping UI elements from annotated Android UI screens (Deka et al., 2017), and from public icon datasets (Google, 2021)). We classify boxes into 9 classes: "text-button", "image", "text-label", "image-button", "radio-button", "check-box", "input-field", "select", and "other".

The final output consists of a set of bounding boxes, each with a class label, probability vector, and recognized text (if any).

C Additional training details

All Lexi models are implemented in PyTorch (Paszke et al., 2019). The text input size is 512 tokens. Alt-text is truncated to 64. The remaining tokens (at least 448) are used for the instruction text. The image region feature size is 2048. Scalar coefficients for the pre-training loss $\mathcal{L}$ are as follows: α=0.01, β=0.8, γ=0.15, and δ=0.2.

In Table 4, we report additional details on the pre-training and fine-tuning process for each task, including datasets, training, validation and test splits, effective batch size (BS), learning rate (LR), and number of training epochs (NE). For fine-tuning

[7]We discard boxes with recognized text shorter than 2 characters and with a confidence lower than 0.7.

Task	Dataset	Split sizes			BS	LR	NE
		Train	Val	Test			
Lexi pre-training (4 tasks)	UICaption (Mixed + MS + Android)	134k	5.6k	7k	128	4e-5	100
UI action entailment	UICaption (Mixed + MS + Android)	77k	3.7k	6k	256	2e-5	50
Instruction-based UI image retrieval	UICaption (Mixed + MS + Android)	77k	3.7k	6k	256	2e-5	50
Grounding referring expressions	UIBert	15.6k	448	544	32	1e-5	50
UI entity recognition	AndroidHowTo + UICaption (MS)	34.5k	4.4k	4.5k	32	1e-5	50

Table 4: Training details including dataset sizes (reported as the number of unique image-instruction pairs), effective batch size (BS), learning rate (LR), and maximum number of epochs (NE). UICaption's Zoom and Teams data are used only in testing, for an OOD analysis.

tasks, NE indicates the maximum number of training epochs, as we have an early stopping using a validation split with patience of 5. Dropout is fixed at 10% for all layers that use dropout.

D Analysis of UI action entailment predictions

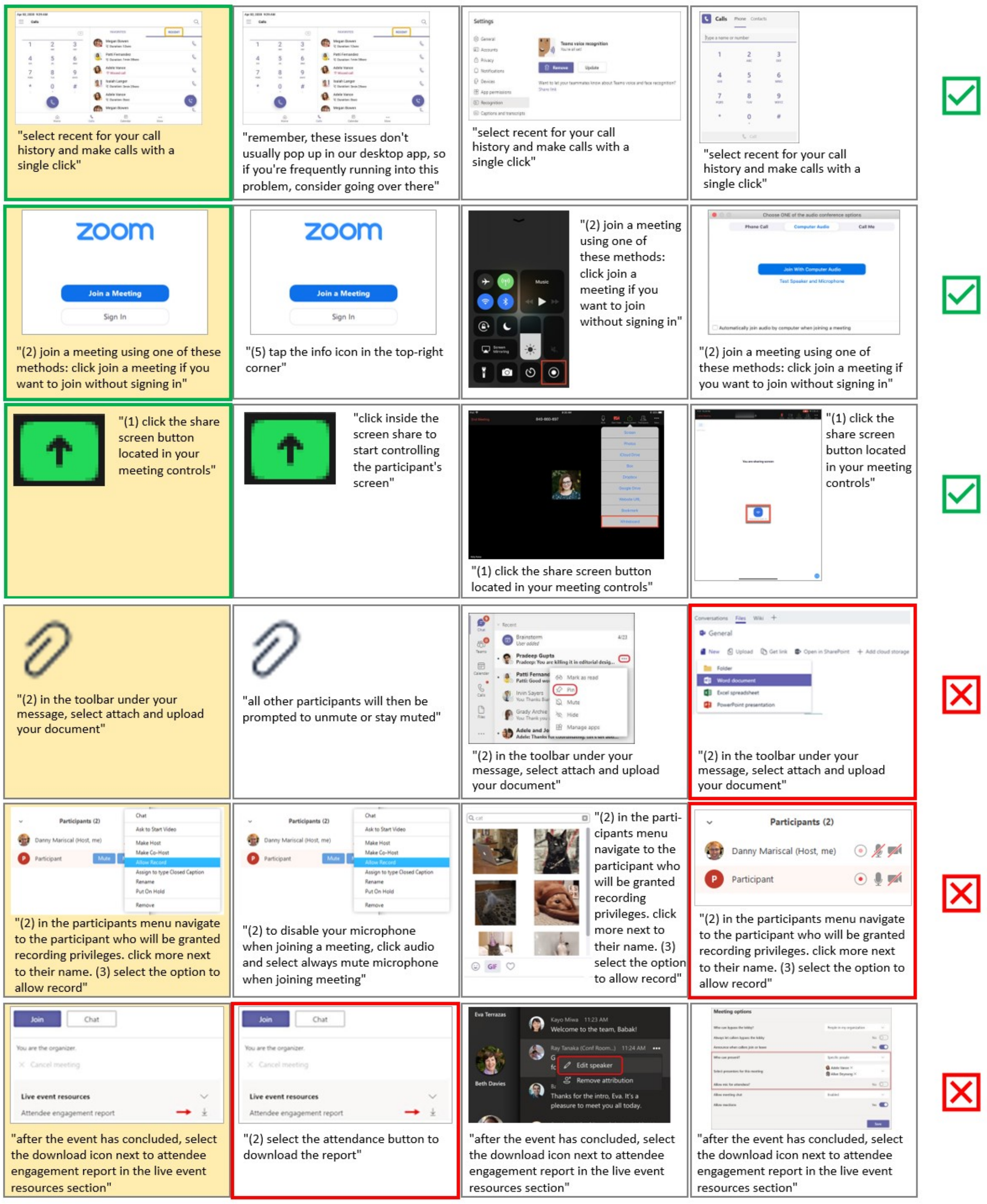
In Figure 5, we share 3 correct and 3 incorrect predictions made by Lexi-6 fine-tuned for the UI action entailment task. The test dataset consists of Teams and Zoom image-text pairs. In the figure, the true pair is always the first one, while the prediction is highlighted in green (correct) or red (wrong). The last pair is the hard negative one. Note that in the UI action entailment task, the model is not provided with the alt-text associated with the UI image, even when it is available.

We observe that the model can select the correct image-instruction pair out of the four given choices even in cases where the alternative choices contain text or UI controls with a high overlap with the instruction text. In the first example in Figure 5 (first row), the instruction describes how to make calls in Teams; the last pair (the hard negative sample) consists of a UI screen with a call pad but the model correctly selects the first UI screen which contains the same call pad but also the "Recent" button mentioned in the instruction. In the second example, the instruction is about joining meetings in Zoom. While both the first and forth UI screen contain "Join" buttons, the model correctly selects the first UI screen which is more appropriate for the instruction given ("join a meeting" vs. "join with computer audio"). Finally, in the third example, the model correctly selects the first pair despite the second and forth pairs both containing a UI screen related to screen sharing and an instruction referring to the screen share function.

In cases where the model chooses incorrect pairs, we observe that it still selects very reasonable pairs. In the fifth example, Lexi selects the last choice where the UI screenshot and caption are related to the participant menu; however, the correct UI screen for that caption (in the first pair) shows another view of the participant menu containing the "more" button mentioned in the instruction. The last example is particularly challenging: the instructions in the first and second pair, which contain the same UI screen, describe the same functionality which can be achieved through two slightly different UI controls (the "download icon" shown in the UI screenshot and the "attendance button").

E Labeling data for UI entity recognition

For the UI entity recognition task we collected a small dataset to train and test our models. We asked 4 annotators with technical backgrounds (2 female and 2 male, English speakers, and payed competitive hourly wages) to annotate some of the MS instruction manuals. Annotators knew their annotations would have been used for a research project. Each annotator was presented with multiple Excel files containing textual instructions for various Microsoft Office applications (e.g., Excel, Word, OneNote, Teams, etc.). Each row in each Excel sheet contained information about a specific functionality, organized as follows: (1) name of the application, (2) title of the manual section (i.e., functionality name), (3) brief description of the functionality, and (4) a set of instructions. The annotator was asked to read the set of instructions for each functionality and label the span of words that described one of three types of entity: (i) UI action verbs (by entering the <verb> and </verb> tags), (ii) UI action objects (by entering the <obj> and </obj> tags), and (iii) input parameters (by entering <input> and </input> tags). Each file was annotated by 3 annotators; in case of disagreement, a forth annotator was consulted. Before starting the annotation task, annotators were given a definition of


"select recent for your call history and make calls with a single click"

"remember, these issues don't usually pop up in our desktop app, so if you're frequently running into this problem, consider going over there"

"select recent for your call history and make calls with a single click"

"select recent for your call history and make calls with a single click"

"(2) join a meeting using one of these methods: click join a meeting if you want to join without signing in"

"(5) tap the info icon in the top-right corner"

"(2) join a meeting using one of these methods: click join a meeting if you want to join without signing in"

"(2) join a meeting using one of these methods: click join a meeting if you want to join without signing in"

"(1) click the share screen button located in your meeting controls"

"click inside the share screen to start controlling the participant's screen"

"(1) click the share screen button located in your meeting controls"

"(1) click the share screen button located in your meeting controls"

"(2) in the toolbar under your message, select attach and upload your document"

"all other participants will then be prompted to unmute or stay muted"

"(2) in the toolbar under your message, select attach and upload your document"

"(2) in the toolbar under your message, select attach and upload your document"

"(2) in the participants menu navigate to the participant who will be granted recording privileges. click more next to their name. (3) select the option to allow record"

"(2) to disable your microphone when joining a meeting, click audio and select always mute microphone when joining meeting"

"(2) in the participants menu navigate to the participant who will be granted recording privileges. click more next to their name. (3) select the option to allow record"

"(2) in the participants menu navigate to the participant who will be granted recording privileges. click more next to their name. (3) select the option to allow record"

"after the event has concluded, select the download icon next to attendee engagement report in the live event resources section"

"(2) select the attendance button to download the report"

"after the event has concluded, select the download icon next to attendee engagement report in the live event resources section"

"after the event has concluded, select the download icon next to attendee engagement report in the live event resources section"

Figure 5: Samples from UI action entailment with Lexi-6 predictions for Zoom and Teams. The correct pair is the first one in each row. Correct predictions are highlighted in green and wrong ones in red.

our entity types and multiple examples, especially ambiguous ones. For instance, we showed them a sample instruction "Specify the PowerPoint presentation that you want to open and then click..." and labeled "PowerPoint presentation" as action object (rather than "the PowerPoint presentation that you want to open").

F Adapting AndroidHowTo to UI entity recognition

Li et al. (2020a) propose a phrase tuple extraction (PTE) model and release the AndroidHowTo dataset, which inspired our UI entity recognition task. In PTE, given a natural language instruction the model extracts a sequence of tuples that best describe each action contained in the instruction. Each tuple consists of a UI operation (of 7 possible types, such as click, input, etc.), a UI object, and additional arguments. The goal of PTE is sequence generation. Given the instruction "to complete your task tap A and B", the goal of the PTE model is to output the 2 tuples <operation:"click", object:"A"> and <operation:"click", object:"B">.

We build a simpler, multi-modal version of this task focused on entity extraction. We introduce equivalent entities (UI action verb, UI action object, and input parameters) and leverage AndroidHowTo to build a dataset to train our model. For the same example instruction above, our model's goal is entity extraction, i.e., tagging "tap" as an action verb, and "A" and "B" as action objects.

An example of AndroidHowTo instruction with annotations is the following.

```
{"instructions": "Keep your phone
    updated: to view the most recent
    security update, open the
    Settings app and tap Security or
    System updates. Go to Settings >
    Google > Security > Google Play
    Protect.",
"operation_list": ["CLICK", "CLICK",
    "CLICK", "CLICK", "CLICK", "CLICK
    "],
"verb_list": ["open", "tap", "go to",
     "go to", "go to", "go to"],
"obj_list": ["the settings app", "
    security or system updates", "
    settings", "google", "security",
    "google play protect"],
"args_list": []}
```

We adapt this data to build a training dataset for our task. We keep obj_list and args_list intact and compute a new action_verb_list derived from verb_list:

```
"action_verb_list": ["open", "tap", "
    go to"]
```