

ILuvUI: Instruction-tuned LangUage-Vision modeling of UIs from Machine Conversations

Yue Jiang
Aalto University
Espoo, Finland

Amanda Sweeningin
Apple
Seattle, USA

Eldon Schoop
Apple
Seattle, USA

Jeffrey Nichols
Apple
Seattle, USA

Abstract

Multimodal Vision-Language Models (VLMs) enable powerful applications from their fused understanding of images and language, but many perform poorly on UI tasks due to the lack of UI training data. In this paper, we adapt a recipe for generating paired text-image training data for VLMs to the UI domain by combining existing pixel-based methods with a Large Language Model (LLM). Unlike prior art, our method requires no human-provided annotations, and it can be applied to any dataset of UI screenshots. We generate a dataset of 353K conversational examples paired with UIs that cover Q&A, UI descriptions, and planning, and use it to fine-tune a conversational VLM for UI tasks. To assess the performance of our model, we benchmark it on UI element detection tasks, evaluate response quality, and showcase its applicability to UI verification.

ACM Reference Format:

Yue Jiang, Eldon Schoop, Amanda Sweeningin, and Jeffrey Nichols. 2025. ILuvUI: Instruction-tuned LangUage-Vision modeling of UIs from Machine Conversations. In *30th International Conference on Intelligent User Interfaces (IUI '25), March 24–27, 2025, Cagliari, Italy*. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3708359.3712129>

1 Introduction

For nearly as long as graphical user interfaces (UIs) have been popular, users have sought methods to verbally describe them for accessibility [25] or automate interaction with them, for example, to execute repetitive tasks [17]. Understanding and automating actions on UIs is challenging since the UI elements in a screen, such as list items, checkboxes, and text fields, encode many layers of information beyond their affordances for interactivity alone. The layout, visual style, and textual content of a UI are often designed to afford user expectations for the domain and capabilities of the application, and the elements themselves may be dynamic or stateful. This inherent complexity makes comprehending and conveying UI-related information through natural language challenging.

Many works in HCI have sought to automate tasks on UIs using programming by demonstration [17, 36, 38], mining similar screenshots through past interactions [2], and Large Language Models

(LLMs) [43, 64, 65]. LLMs, in particular, have demonstrated remarkable abilities to comprehend task instructions in natural language in many domains [16, 52, 53, 63, 72], however, using text descriptions of UIs alone with LLMs leaves out the rich visual information of the UI. Fusing visual with textual information is important to understanding UIs as it mirrors how humans engage with the world. One approach uses Vision-Language Models (VLMs) to accept *multimodal* inputs of both images and text, typically output only texts, and allow for general-purpose question answering, visual reasoning, scene descriptions, and conversations with image inputs [1, 15, 18, 42]. However, the performance of these models on UI tasks falls short compared to natural images because of the lack of UI examples in their training data.

In this paper, we adapt the LLaVA [41, 42] VLM training data generation recipe to the UI domain. Our data generation recipe does not require any human labeling and can be adapted to UIs in existing datasets, such as Rico [20], which do not have existing textual descriptions. We generate a dataset of 353K image-instruction pairs using screenshots from the AMP dataset [71] and additional data from the Crawls interaction trace dataset [26]. In order to generate text pairs for a given screenshot, we employ a UI element detection model [71] that identifies the elements in a screen, converts these detections into a structured text-based representation, and then prompts an LLM with this representation and additional context to generate one or more realistic phrases. We generate six types of phrases: single-step Q&A conversations, detailed descriptions, listing available actions, predicting UI action outcomes, selecting a UI element given a goal, and a goal-based plan.

After generating data using the GPT [50], we use that data to fine-tune a conversational VLM. Our resulting model, ILuvUI, is capable of describing properties of UI elements and screens, contextual help, and planning multi-step interactions. Like many large, unsupervised models, ILuvUI can perform tasks it was not trained to perform. Beyond interpreting the complexities of UIs and following instructions, our model paves the way for using VLMs to enhance UI accessibility by automatically generating descriptions and acting on instructions provided by speech. To better understand ILuvUI's performance, we benchmark against a UI element detection model, evaluate its response quality compared to a baseline VLM, and show selected examples demonstrating ILuvUI's planning and reasoning capabilities. Additionally, we assess the model's effectiveness through UI verification, a process that enables the model to evaluate whether user interactions with the interface align with expected behaviors, thereby identifying both correct and erroneous actions.



This work is licensed under a Creative Commons Attribution 4.0 International License.
IUI '25, Cagliari, Italy
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1306-4/25/03
<https://doi.org/10.1145/3708359.3712129>

Our paper makes the following contributions:

- (1) We adapt the LLaVA method [42] to generate paired text-image data to train VLMs in the UI domain using an LLM and a UI element detector. Our method only uses UI screenshots as input, does not require any human-provided captions, and produces six different types of text pairs.
- (2) We fine-tune an open-source VLM, LLaVA [41, 42], on our UI dataset, leading to the development of a UI-focused instruction-following visual agent ILuvUI.
- (3) We evaluate the performance of ILuvUI using three UI understanding tasks – Element Existence, Type, and Purpose – along with human evaluations of UI summarization and detailed descriptions. Additionally, we assess the model’s effectiveness through UI verification.

2 Related Work

This section focuses on preexisting methods for UI understanding, language models applied to UIs, and generating multimodal data for UIs.

2.1 Computational Understanding of User Interfaces

Pixel-based UI understanding is broadly applicable across diverse domains, including interface adaptation [3, 8, 22, 70], GUI testing [68], data-driven GUI searches [9, 11, 29], prototyping [59], UI code generation to support app development [4, 10, 12, 19, 49], and GUI security [13]. Conventional image processing methods have long been employed to identify UI elements within images, often relying on detecting and aggregating edges [49, 59, 69]. While adept at handling simpler GUIs, these methodologies struggle when faced with images that have gradients, photographs, or intricate UI layouts. Furthermore, template matching techniques [22, 54, 68, 70] require meticulous feature engineering and curated templates, introducing potential limitations when applied to UIs with diverse visual attributes. Many reverse engineering approaches also predict UI elements and layouts [5, 6, 22–24, 31–35, 44, 46, 49, 56, 60, 68], detect hierarchical groupings [67], generate code for UIs [4, 69], and enable platform migration or UI improvement [21, 45, 47, 58].

The surge of deep learning-based approaches has been enabled by comprehensive UI datasets such as Rico [20]. A prominent instance of this trend is witnessed in Pix2Code [4], which employs an end-to-end neural image captioning model to generate descriptions of interface layouts. Correspondingly, Chen et al. [10] use a CNN-RNN model to create a UI skeleton that includes widget types and layouts inferred from screenshots. Other models [9, 66] use object detection techniques to detect GUI widgets within screenshots. Some prior research [57, 61] predicts the tappability of mobile app UI elements. Some deep learning models also use crowdsourcing to generate image captions [27, 28]. Other approaches [12, 14, 48] combine traditional image processing methods, such as edge detection, with deep learning-based classification models to detect UI elements and semantics, including UI types. Zhang et al. [71] infers interface elements and metadata from pixels, including UI types, navigation order, groupings, image descriptions, and icon classes. We use this latter method to produce textual representations of UIs to generate text-screenshot pairs for VLM training.

2.2 Large Language Models and Vision Language Models

Large language models (LLMs) have broad potential in interpreting language, serving as an interface for a versatile AI assistant, and can comprehend explicit task instructions in natural language. ChatGPT [50] and GPT-4 [51] show the proficiency of well-aligned LLMs in following human instructions. This achievement has sparked a surge of interest in developing open-source LLMs. Among these, LLaMA [63] is a downloadable LLM that rivals the performance of GPT-3. Other models like Alpaca [62], Vicuna [16], and GPT-4-LLM [51] have leveraged machine-generated, high-quality instruction-following examples to enhance the alignment capabilities of LLMs. The LLaVA [42] model on which we base our work is itself a combination of the CLIP [30] and Vicuna [16] models.

Spotlight is an example of a VLM trained for UI tasks [37]. This model took as input both an image of UI and a region of interest and outputs text related to the region and was applied to tasks such as widget captioning, screen summarization, command grounding, and tappability prediction. The Spotlight model is quite powerful. However, it can only perform the specific tasks defined in the datasets and not accept text prompts. ILuvUI, in contrast, can perform a broader range of UI tasks, including those requiring logical reasoning. It can also accept a text prompt as input in addition to the UI image, which enables it to provide answers for use cases such as visual question-answering.

3 Generating Multimodal Data for User Interfaces

While general-purpose VLMs perform well on natural images for several tasks, their capabilities are limited when performing tasks with UIs. This is because existing VLMs lack latent knowledge of common UI design principles, tasks, and element types. The availability of training data that ties UI screenshots with descriptions or grounded conversation is comparatively limited. This motivates the need to create a dataset that pairs UIs with natural language instructions and descriptions to enable training a VLM that can understand and act on UIs.

LLaVA [42] is recent work that introduces a method for creating text-image data pairs from an existing image dataset and a multimodal VLM trained on that data, which connects the CLIP vision encoder [30] with the Vicuna LLM [16] for general-purpose visual and language understanding. We adapt both aspects of this work to create a VLM tuned to UI tasks.

A key contribution of LLaVA’s data generation method is using an off-the-shelf LLM to generate a dataset of text-image pairs from the existing COCO image dataset [40]. The COCO dataset pairs images with human-annotated captions and bounding boxes of objects within each image. For each image, the method combines the captions, the list of bounding boxes of objects with the object types, and an additional prompt into a request to a proprietary LLM (experiments were done with both GPT-3.5 and GPT-4). The resulting outputs from the LLM contained realistic text phrases paired with the original image to produce a VLM training set. LLaVA produced three kinds of text pairs that they termed Q&A, detailed description, and complex reasoning.

We adapted the LLaVA dataset generation recipe to UI tasks in several ways to fit our existing UI dataset and different UI tasks. The LLaVA method relied on the COCO dataset [40] to provide human-annotated captions and UI element bounding boxes. For UI data, we use the AMP dataset from Zhang et al. [71], which comprises 80,945 unique screens from 4,239 iPhone apps; and the CRAWLS dataset [26], which contains 750,000 iOS app screens from 6,000 apps. These datasets contain screenshots and annotations of UI element bounding boxes but no human-annotated captions of its UIs. To ensure generalizability, we produced the bounding boxes using an object detection model [71] during data generation since we sought a method that would work with completely unannotated UI examples as might be found in other datasets. The missing UI captions were generated using an LLM for each screen with a prompt that included the UI element detections. We then created text pairs by generating analogous examples to the original recipe (Q&A, detailed descriptions), as well as adapting LLaVA’s original complex reasoning task to multiple different UI tasks: listing possible actions, predicting the outcome of actions, selecting an element that accomplishes a goal, and goal-based planning. Figure 1 provides an example of this process. We generate data from the GPT-3.5-turbo model. In the subsequent discussion, the term “GPT” denotes the GPT-3.5-turbo model. With better GPT models, such as GPT-4, it may achieve more accurate results.

In total, we generate 353K unique language-image instruction-following samples, including 224K in conversations, 32K in detailed description, 32K in detailed description, 32K in logical reasoning, 32K in potential actions, and 1K in UI transition, respectively.

3.1 Detecting UI Elements and Generating Screen Captions

We use the UI detection model from Zhang et al. [71] to generate a list of UI elements with their bounding boxes and types for each screen. UI element information is included in all of our data generation prompts and is formatted as follows:

Label: [type], Text: [text], BoundingBox from (x1, y1) to (x2, y2)

[type] refers to the element category (e.g., text, button, or icon), [text] is the text contained by the UI element extracted via OCR, and the bounding box coordinates are defined by the top-left position and the bottom-right position of the UI element, respectively.

In order to generate screen captions, which are also used in the rest of our data generation prompts, we query GPT with formatted the detected UI elements using the following prompt: “Given the UI screen [Bounding Boxes]. Write a single-sentence usage description for this UI screen.” This prompt was determined experimentally and produced good results. An example of UI element detection and caption is shown in Figure 1.

3.2 Generating LLaVA-Style Data

As shown in Figure 2, LLaVA generates both single-step Q&A conversations and detailed descriptions as part of its data generation procedure, which we also include in our procedure. For both, we use the same few-shot in-context learning method as LLaVA to present GPT with examples of desired output using two “golden examples” we authored. As shown in Figure 4, we design a system message

and select two UI examples with their respective bounding boxes, captions, and desired questions and responses for each data type. Example prompts for each data type are provided in Supplementary Material.

3.2.1 Single-Step Q&A Conversations. The Q&A data type consists of question-answer pairs between an assistant and a user, conditioned on a source UI screenshot. The assistant responds in a way that suggests they view the UI image while answering the question. Various questions are asked about the UI image, including element attributes, placements, relative positions, functionalities, and purpose. Only questions with definite answers are considered.

3.2.2 Detailed Description. The Detailed Description data type is a rich and comprehensive description of a UI image describing all of its elements and their respective functionality. We generate various questions asking the LLM to describe the UI in detail. We randomly select a single question from this list for each UI screenshot example to be the question for the description.

3.3 Generating UI-Specific Data

LLaVA includes a generic “complex reasoning” task as part of its data generation process. It elicits complex responses, including commonsense or historical descriptions of elements in the image and step-by-step descriptions of actions taking place in a scene. We found that this task was too general for UI tasks. Instead, we chose to substitute it with five UI-specific data generation types: identifying UI element types, listing possible actions on a screen, predicting the outcome of taking an action, selecting a UI element capable of a given action, and formulating a plan that accomplishes a goal on the given screen. These tasks were designed to familiarize the VLM with the capabilities of particular UI elements and convey a sense of the affordances signified by common UI design patterns [39]. Figure 2 and Figure 3 show examples to illustrate the data we generate in the UI domain.

3.3.1 Identifying UI Element Types. We randomly sampled 2000 UIs from the AMP dataset [71] and selected 5 elements at random from each UI. For each selected element, we create a UI image annotated with a bounding box around that element. Subsequently, we created a Q&A pair for each element inquiring about its element type. The element types include ‘text’, ‘icon’, ‘picture’, ‘text field’, ‘segmented control’, ‘tab bar’, ‘page control’, ‘dialog’, ‘checkbox’, ‘toggle’, and ‘slider’.

3.3.2 Listing Available UI Actions. We prompt GPT to list all potential actions that can be taken on a particular UI, given its high-level description and the list of recognized elements on the screen. Actions can range from tapping, swiping, or entering text, depending on the functionality of the UI. For this task, we construct a prompt from the UI screenshot caption and its bounding boxes and use the zero-shot result from GPT.

3.3.3 Predicting the Outcome of UI Actions. As shown in Figure 3, in addition to understanding the capabilities of various UI elements, we wanted our VLM to reason about the potential outcomes of acting on UI elements. From the interaction traces in the CRAWLS dataset [26], which overlaps with the AMP dataset, we could establish ground truth correspondences from an element that was



Figure 1: One example to illustrate the input UI image, the UI detection results, formatted bounding boxes and caption.

tapped on one screen and the second screen to which that interaction led. This dataset includes the pixel coordinates of a tap action that caused the UI transition. We run UI element detection on the first screen and match the pixel to the UI element with the smallest bounding box that contains the tap location to identify what specific element was tapped to cause the transition.

We feed the UI elements of the first screen, tapped UI element, and caption of the second screen to GPT and prompt it to formulate a single Q&A pair that asks what action will take place when the

given UI element is tapped and answer with a concise description of the second screen. Since the LLM has access to ground truth in its context for these cases, there is no need to use few-shot examples.

3.3.4 Selecting a UI Element Given a Goal. To complement the above task, we also use the LLM to generate a Q&A pair that asks what element must be tapped on the first screen to arrive at a target view, specified by a concise description of the second screen. The question is prompted not to include any information about the

Conversation:

Question: What are the options available in the navigation bar at the bottom?

Answer: The options available in the navigation bar at the bottom are "Listen Now", "Browse", "Library", and "Search".

Question: What does the "Follow" button do?

Answer: The "Follow" button allows users to follow the Apple Events podcast, which means they will receive updates or notifications when new episodes are released.

Question: What information is provided about the Apple Events podcast?

Answer: The information provided about the Apple Events podcast includes a description of the podcast, its rating, and its category. The description mentions that the podcast is home to the latest keynote addresses, and users can watch announcements of new products and services and browse the archive.

Question: Where is the "Back" icon?

Answer: The "Back" icon is at the top-left corner.

Detailed Description:

Question: Describe the image in detail.

Answer:

This UI screen presents an Apple Events podcast page. At the top, there are three icons: a back button on the left, a 'Follow' button in the middle, and a 'more' icon on the right. Just below these, a refresh icon is positioned on the left, followed by the title 'Apple Events' and a right arrow icon. A 'Resume' button is situated beneath these elements.

The page includes a description of the Apple Events podcast, which reads: "The Apple Events podcast is home to the latest, keynote addresses. Watch announcements of new, products and services and browse the archive MORE, 4.4 (4.8K) Technology". A star icon is present within this text block, likely indicating a rating or favorite feature.

The 'Episodes' section is found below the description, with a 'See All' option on the right. The first episode listed is titled "The next is yet to come. Join US for a special Apple, Event on September 12 broadcasting live from, Apple Park. Listen right here on Apple Podcasts., TRAILER, 14 sec left". This episode entry includes a play icon on the left and a 'more' icon on the right.

Below this episode, there's a button labeled "September 5, 2023, Apple Event, play", which likely represents an upcoming event.

At the bottom of the screen, there are four tabs for navigation: 'Listen Now', 'Browse', 'Library', and 'Search', listed from left to right. These tabs allow users to switch between different sections of the podcast page.

Listing Available UI Actions

Question: What actions can we perform on this user interface?

Answer:

- Tap on the "Button" with the text "add"
- Tap on the "Button" with the text "Resume"
- Tap on the "Icon" (Type: star)
- Tap on the "Text" with the text "See All"
- Tap on the "Icon" (Type: play)
- Tap on the "Icon" (Type: more)
- Tap on the "Button" with the text "September 5, 2023, Apple Event, play"
- Tap on the "Tab" with the text "Listen Now"
- Tap on the "Tab" with the text "Browse"
- Tap on the "Tab" with the text "Library"
- Tap on the "Tab" with the text "Search"

Goal-Based Planning

Question: What should the user do if they want to listen to the latest Apple Event podcast episode?

Answer:

To listen to the latest Apple Event podcast episode, the user should look under the "Episodes" section, which is located towards the middle of the UI screen. The most recent episode is typically listed first. The user can identify the episode by its title, which in this case is "Apple Event". To start listening, the user should click on the "play" icon located within the bounding box of the episode.

Figure 2: One example to illustrate the data showing the four types of responses for UI understanding.

Current UI:

- Notifications
- Sounds & Haptics
- Focus
- Screen Time
- General
- Control Center
- Display & Brightness
- Home Screen
- Accessibility
- Wallpaper
- Siri & Search
- Face ID & Passcode

Subsequent UI:

- APPEARANCE
 - Light
 - Dark
- Automatic
- Text Size
- Bold Text
- BRIGHTNESS
 - Slider
- True Tone

Formatted Bounding Boxes of the Current UI:

Label: Text, Text: Settings, BoundingBox from (417, 157) to (706, 270)

Label: Button, Text: notification, Notifications, right arrow, BoundingBox from (0, 270) to (1125, 419)

Label: Button, Text: speaker, Sounds & Haptics, right arrow, BoundingBox from (0, 419) to (1125, 591)

Label: Button, Text: Focus, right arrow, BoundingBox from (0, 593) to (1125, 764)

Label: Button, Text: Screen Time, right arrow, BoundingBox from (0, 766) to (1125, 938)

Label: Button, Text: settings, General, right arrow, BoundingBox from (0, 1046) to (1125, 1219)

Label: Button, Text: Control Center, right arrow, BoundingBox from (0, 1220) to (1125, 1393)

Label: Button, Text: Display & Brightness, right arrow, BoundingBox from (0, 1395) to (1125, 1567)

Label: Button, Text: calendar, Home Screen, right arrow, BoundingBox from (0, 1568) to (1125, 1742)

Label: Button, Text: Accessibility, right arrow, BoundingBox from (0, 1742) to (1125, 1917)

Label: Button, Text: Wallpaper, right arrow, BoundingBox from (0, 1917) to (1125, 2089)

Label: Button, Text: Siri & Search, right arrow, BoundingBox from (0, 2092) to (1125, 2266)

Label: Button, Text: smile, Face ID & Passcode, right arrow, BoundingBox from (0, 2270) to (1125, 2423)

Transition Action:

Label: Button, Text: Display & Brightness, right arrow, BoundingBox from (0, 1395) to (1125, 1567)

Caption of the Subsequent UI:

This UI screen is a settings page for display and brightness. It allows the user to adjust the appearance, brightness, text size, and other display settings of their device.

Predicting the Outcome of UI Actions

Question: What will the UI look like after tapping on the "Display & Brightness" button?

Answer: The new UI screen might display options for adjusting the display appearance, including light and dark modes, as well as options for automatic brightness and true tone.

Selecting a UI Element Given a Goal

Question: If I want to access the settings for adjusting the appearance and brightness, which option should I choose?

Answer: To access the settings for adjusting the appearance and brightness, you should tap on the "Display & Brightness" option.

Figure 3: One example to illustrate the data generation for UI transition.

second screen. Similar to the above, this Q&A pair also uses the Crawls dataset, so there is no need to use a few-shot since ground truth is available.

3.3.5 Goal-Based Planning. The last UI task we use for data generation is goal-based planning, where a QA pair is generated to ask for directions to accomplish a task in the specified UI. Unlike the pairs generated with the Crawls dataset, the generated directions need not result in opening a new view. Generated directions can interact with the current view (e.g., to change its state), formulate

a multi-step plan across multiple views, or relate a task with the current UI to commonsense knowledge. In practice, most generated examples formulated multi-step plans for completing complex tasks on the given screen. For this task, we use two 2-shot examples. The first example asks what step should be taken to log into a given login screen (first, enter email; then password; then click “sign in”). The second example asks what the user should do when the item shown on a product page is too expensive (open a visible details menu to see if there are promotions or discounts available).

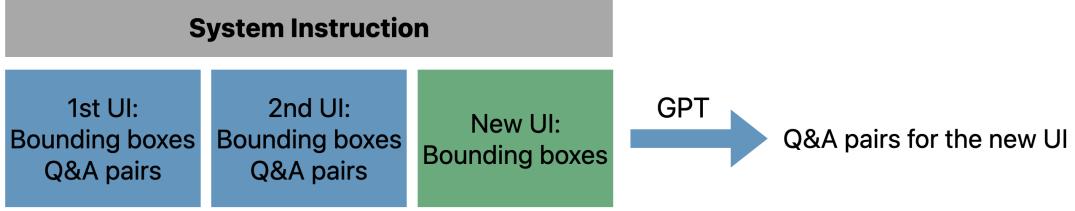


Figure 4: We use two author-created “golden examples” of UI elements and Q&A pairs as few-shot examples for data generation. These examples are prepended to a third list of UI elements, where GPT predicts new, resulting Q&A pairs.

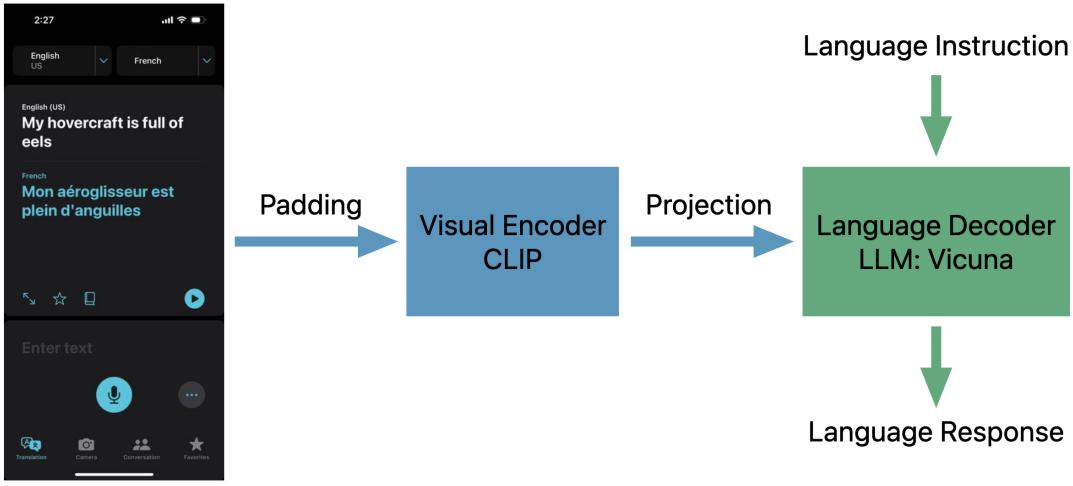


Figure 5: We generate a dataset of Q&A pairs that cover several UI description and reasoning tasks and use it to fine-tune a Vision-Language Model (VLM) that accepts multimodal inputs from screenshot pixels and user-provided text input. Our model, ILuvUI, enables new kinds of UI tasks with conversational VLMs.

4 Vision Language Model

To obtain the UI-focused instruction-tuned VLM, we employ the same network architecture as the LLaVA model [42] with some modifications to the image input step. We found in initial experiments that the default CLIP encoder used by LLaVA used an input of 224x224 and resized input images to fit these dimensions. Resizing these images to such a small size and to fit the square shape causes significant loss of information and distortion, because of the relatively many intricate details of UI images compared to natural images. This in turn led to unsurprisingly poor interpretation of UI elements and text information.

We made two modifications to address this issue. First, we pad the input UI images to square them before resizing to the required input size for the model, which prevents the UI images from becoming overly distorted. Second, we use CLIP-L-336px [55] as the visual encoder, which increases the model input size to 336x336 pixels. We were concerned that this size would still be too small to capture UI detail, but our experiments showed that performance was much improved compared to the smaller image size.

Otherwise, our design matches LLaVA. The visual encoder is followed by the Vicuna-13B LLM [16, 72] as a language decoder as illustrated in Figure 5. Given the input UI image \mathcal{I} , we first pad

and then resize the UI image to fit the visual encoder input. Grid features from before and after the last transformer layers of the pre-trained CLIP visual encoder are then used to output the visual features $h_v = \mathcal{E}(\mathcal{I})$. Following this, we apply a trainable feature alignment projection matrix P to align the visual content into language embeddings $h_p = P \cdot h_v$, which maintain the same dimension as the word embedding in the language model. We then concatenate the projected visual embedding h_p and the word embedding of the input language instruction h_l , and use the language decoder to generate the language response $R = \mathcal{D}(h_p, h_l)$.

4.1 Data Preparation

For each input UI image \mathcal{I} , and the generated pairs of questions and answers, $(Q_1, A_1, \dots, Q_T, A_T)$, where T represents the total number of turns, we arrange them into a sequential format. In this sequence, the user’s instruction during the first turn is randomly selected from either $[Q_1, \mathcal{I}]$ or $[\mathcal{I}, Q_1]$, while for all subsequent turns, Q_t is used as the user instruction. We consider all the corresponding answers, A_t , as the responses provided by the assistant at each respective turn. This ensures that the sequence of user input, whether the UI image or language instruction comes first, does not affect the testing phase.

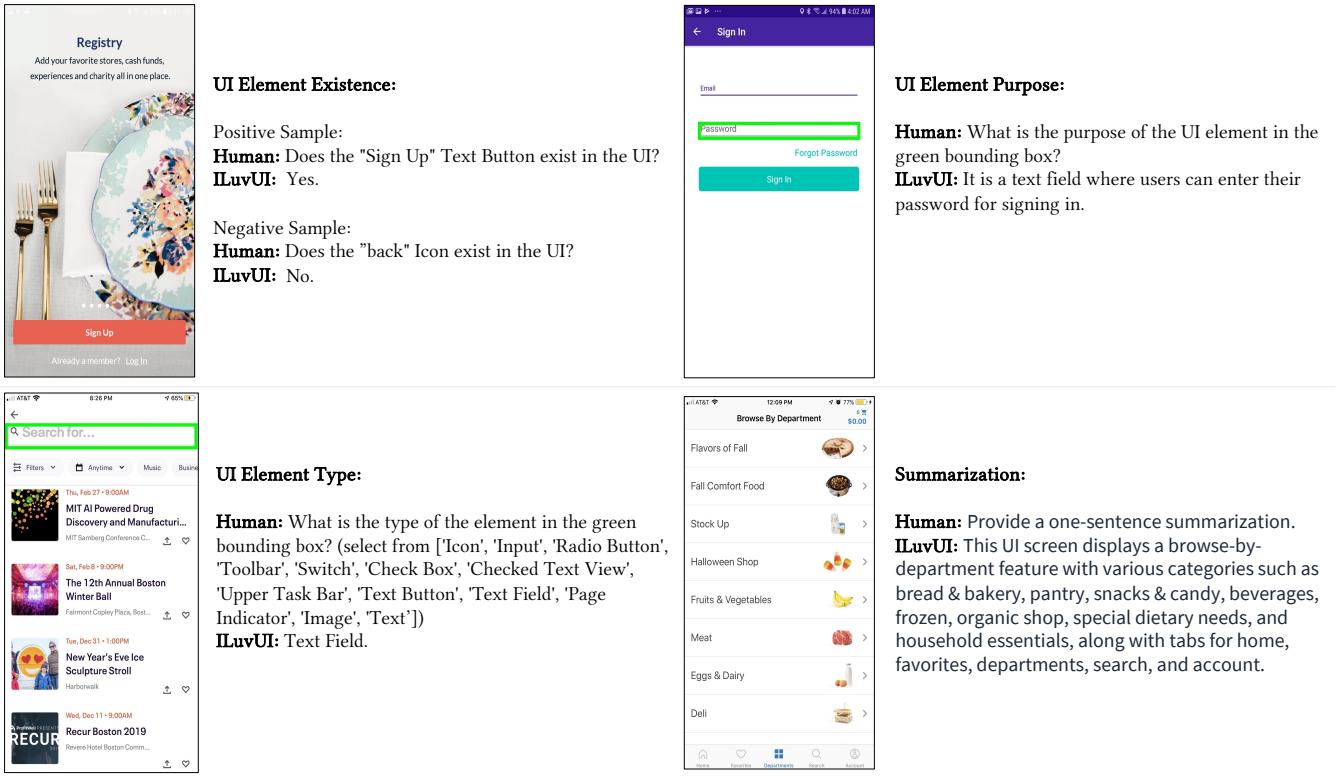


Figure 6: An illustration of four UI tasks in our evaluation including identifying UI element existence, UI element type, and the purpose of the UI element, and generating the UI screen summarization.

4.2 Fine-Tuning Process

ILuvUI uses the pretrained visual encoder \mathcal{E} and an MLP as alignment projection P used in LLaVA V1.5 [41] to align the visual content embedding with the pre-trained LLM word embedding. We use the same language decoder as LLaVA \mathcal{D} , a pretrained Vicuna-13b-v1.5 LLM [72]. We freeze the visual encoder weights while fine-tuning the feature alignment projection matrix and the LLM weights, resulting in a UI-focused instruction-tuned VLM.

5 Evaluation

We qualitatively show the comparison between ILuvUI and LLaVA V1.5 in Table 1 (We also include GPT-4V results for reference), Table 2, and Table 3. Furthermore, we evaluate ILuvUI quantitatively in three ways. We analyze the model’s ability to perform three basic UI Understanding tasks: identifying a UI element’s existence, type, and purpose. Moreover, we compare human preferences for UI screen summarization in one sentence and detailed descriptions generated by the ILuvUI and LLaVA V1.5 models.

We ensure a fair comparison with the LLaVA V1.5 model, we compare ILuvUI with a LLaVA V1.5 model that uses the same 336-pixel-square vision encoder and language decoder. To assess and demonstrate the generalizability of ILuvUI, we randomly sampled 1000 UI images as the evaluation UI set from the VINS dataset [7]. We chose this UI dataset for testing since it contains various types of

UIs from both Android and iOS screens. Moreover, the dataset provides human-annotated ground truth bounding boxes and element types. All the UI images had not been seen by ILuvUI previously during training or fine-tuning. The full results of all evaluations are shown in Table 4.

5.1 UI Understanding Tasks

We evaluate ILuvUI’s understanding of UIs by testing its ability to identify the existence, type, and purpose of UI elements. We randomly sampled 200 UIs in the evaluation set. For each UI, we randomly sample 5 UI elements for evaluation. We draw a green bounding box to specify the sampled UI element. Note that we did not include any training data with bounding boxes. We illustrate four UI tasks in Figure 6. Additionally, to show the necessity of using an MLP as the alignment projection to align the visual content embedding with the LLM word embedding, we also compare ILuvUI to an ablated model, “ILuvUI (linear projection)”, with a linear projection instead of an MLP.

5.1.1 UI Element Existence. In order to evaluate UI element existence, for each screen in the evaluation set, we produce a set of positive and negative detection samples. 5 positive samples are the sampled UI elements on each UI screen. 5 negative examples are randomly sampled from UI elements on other UI screens in the evaluation set, but only after a basic matching check is applied to ensure an equivalent element is not likely to appear on the original

	
Type	Detailed Description
User	Describe the UI in detail.
ILuvUI Response	This UI screen presents weather information for various cities. At the top, the title "Weather" is centrally positioned, while a more options icon is located on the top right corner. Below the title, a search bar allows users to input a city or airport name to find weather information. The main part of the screen displays weather information for different cities. Each city entry includes the city name, the current temperature, and a brief weather description. The first city listed is Toronto, with a temperature of 32° and a heat warning. The next city is Honolulu, which is partly cloudy with a temperature of 28°. The third city is Zurich, which is clear with a temperature of 20°. The last city listed is Shanghai, which is partly cloudy with a temperature of 25°. At the bottom of the screen, a text provides additional information about weather data and encourages users to learn more about it.
LLaVA V1.5 Response	The UI is a weather app displaying the current weather conditions for various cities around the world. The app shows the temperature and weather conditions for each city, with a total of six cities displayed in the image. The app is designed to provide users with a quick and easy way to check the weather conditions in different locations.
GPT-4V Response	The user interface (UI) displays a weather application showing current conditions for multiple cities. Each city is represented by a card with the following details: 1. Toronto: It's 16:18 local time with a temperature of 32°C. A heat warning is in effect. The high is 32°C and the low is 21°C. 2. Honolulu: Local time is 10:18 with partly cloudy weather and a current temperature of 28°C. The high is forecasted to be 29°C and the low 23°C. 3. Zurich: The local time is 22:18. The weather is clear, and the temperature is 20°C, with a high of 27°C and a low of 13°C. 4. Shanghai: It's 04:18 local time with partly cloudy conditions and a temperature of 25°C. The high is expected to be 32°C and the low 25°C. The top of the UI has a search bar for finding weather by city or airport. The top right corner shows the device's connectivity status, battery level, and time (13:18). At the bottom, there's a link to learn more about weather data and map data. The background of each card reflects the weather conditions, with sunlight for day and stars for night.
Type	Summarization
User	Could you give a one-sentence summarization?
ILuvUI Response	This UI screen displays weather information for various cities, including the city name, temperature, and a brief weather description, with an option to search for a city or airport.
LLaVA V1.5 Response	A weather app displays the current weather conditions for various cities around the world, including temperature and precipitation.
GPT-4V Response	The UI shows a weather app with current temperatures and conditions for Toronto, Honolulu, Zürich, and Shanghai, along with local times and daily high and low forecasts.

Table 1: Example prompt demonstrating our agent’s capability compared to the original LLaVA V1.5 agent and GPT-4V.



Type	Logical Reasoning
User	If I want to know whether it is raining in London, what should I do?
ILuvUI Response	To know whether it is raining in London, you should enter "London" in the search bar at the top of the screen.
LLaVA v1.5 Response	To check if it is raining in London, you should look at the weather app and check the current weather conditions for London. The app should display the current weather conditions, including precipitation, if any.
GPT-4V Response	To find out if it's raining in London, you can type "London" into the search bar at the top of the weather app interface where it says "Search for a city or airport," and then review the current weather conditions that are displayed for London.
Type	UI Transition
User	If I click on the top right button, what will happen?
ILuvUI Response	If you click on the top right button, it will likely open up more options or settings related to the weather app.
LLaVA v1.5 Response	If you click on the top right button, it will likely bring up a menu or options for the user to navigate through the app, such as accessing additional weather information, settings, or other features. The exact functionality of the button may vary depending on the specific app you are using.
GPT-4V Response	The top right button, which appears to be three horizontal dots, typically represents a menu or settings option in a user interface. If you click on it, you would likely open a menu with additional options or settings for the weather application, such as preferences, help, or feedback.

Table 2: Example prompt demonstrating our agent’s capability compared to the original LLaVA V1.5 agent.

screen (e.g., close and back icons appear relatively commonly). We query both models for each of the 5 positive and 5 negative samples, asking whether they exist in the input UI image. Our ILuvUI agent achieves an accuracy of 76.9%, outperforming the LLaVA V1.5 model, which has a 58.0% accuracy. Note that the LLaVA V1.5 model nearly always provides positive responses, resulting in a high recall of 99.8% but close to random-chance accuracy.

5.1.2 UI Element Type. In order to evaluate UI element type, we queried both models to identify the types of the sampled UI elements from a list of 13 common types in the dataset, including text, page indicator, text button, image, icon, upper taskbar, edit text, switch, input, checked text view, toolbar, checkbox, radio button. The list of element types is provided in a randomized order for each query. This is a harder task, and our ILuvUI agent achieves an accuracy of 25.1% compared to 8.8% for LLaVA V1.5.

5.1.3 UI Element Purpose. Understanding the purpose of individual UI elements within the UI is important for a comprehensive understanding of the UI. We queried both models regarding the purpose of each sampled UI element and employed human evaluation to validate the accuracy of their responses. Both LLaVA V1.5 and ILuvUI with linear projection failed to recognize the purposes of almost all UI elements. They produced descriptions of the overall UI rather than identifying the specific purpose of each element. This resulted in uniform responses for each UI, regardless of the bounding box’s location. While ILuvUI managed to generate meaningful purposes for specific elements to some extent, it still demonstrated a deficiency in accuracy.

5.2 Human Evaluation of UI Summarizations and Detailed Descriptions

For all 1000 UI images in the evaluation set, both models were queried to generate a one-sentence summarization and a detailed UI

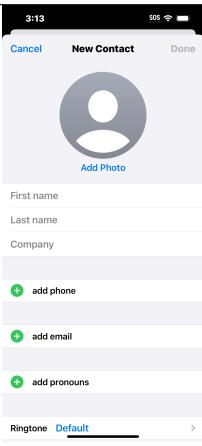
	
Type	Detailed Description
User	Could you provide a detailed description?
ILuvUI Response	<p>This UI screen is designed for creating a new contact. At the top of the screen, there are three text labels: "Cancel" on the left, "New Contact" in the center, and "Done" on the right. These options allow users to either discard the current operation, understand the purpose of the screen, or save the new contact, respectively.</p> <p>In the middle of the screen, there is a placeholder for a photo, indicated by the text "Add Photo". This suggests that users can add a photo of the new contact.</p> <p>Below the photo placeholder, there are three text fields for entering the contact's first name, last name, and company. These fields are arranged vertically, with the first name at the top and the company at the bottom.</p> <p>Further down, there are three buttons for adding contact details. The first button, labeled "Add phone", allows users to add the contact's phone number. The second button, labeled "Add email", enables users to add the contact's email address. The third button, labeled "Add pronouns", provides an option to add the contact's preferred pronouns.</p> <p>At the bottom of the screen, there is a segmented control for selecting the ringtone for the new contact. The options are "Default" and "Ringtone 3", with the "Default" option on the left and the "Ringtone 3" option on the right.</p>
LLaVA V1.5 Response	The image displays a cell phone screen with a contact list, showing a new contact added to the list. The contact is a person with a gray profile picture, and their name is displayed below the picture. The contact information includes their first name, last name, company, and email address. The cell phone is positioned horizontally, with the contact information occupying the majority of the screen.
Type	Logical Reasoning
User	What should I do if I want to add Amy Black at Company ABC with email address 12345@abc.com and phone number +1 1234567890?
ILuvUI Response	To add Amy Black at Company ABC with email address 12345@abc.com and phone number +1 1234567890, you should first enter "Amy" in the "First name" text field located near the top of the UI screen. Below that, in the "Last name" text field, you should enter "Black". In the "Company" text field, you should enter "ABC". Then, in the "Add phone" section, you should click on the "+" icon and enter "+1 1234567890". Similarly, in the "Add email" section, you should click on the "+" icon and enter "12345@abc.com". Once all the information is entered, you can click on the "Done" button located at the top right corner of the UI screen to save the new contact..
LLaVA V1.5 Response	To add Amy Black at Company ABC with the email address 12345@abc.com and phone number +1 1234567890, follow these steps: Open the Contacts app on your cell phone. Tap the "Add Contact" button or the "+" icon to create a new contact. Enter Amy's first name, last name, and company name (Company ABC). In the "Email" field, type "12345@abc.com". In the "Phone" field, type "+1 1234567890". Save the contact by tapping the "Save" or "Add" button.

Table 3: Example prompt demonstrating our agent's capability compared to the original LLaVA V1.5 agent.

Agent	Existence (%)				Type (%)	Purpose (%)
	Accuracy	Precision	Recall	F1 Score	Accuracy	Accuracy
LLaVA V1.5	58.0	56.2	99.8	71.9	8.8	4.0
ILuvUI (linear projection)	55.4	90.4	19.1	31.5	19.2	4.5
ILuvUI	76.9	82.5	72.3	77.0	25.1	19.0

Table 4: Comparison between ILuvUI, ILuvUI with linear projection, and LLaVA V1.5 on UI understanding tasks. The result shows that ILuvUI outperforms the LLaVA V1.5 model and the necessity of using an MLP instead of a linear projection aligning between the visual embedding and text embedding.

Agent	Summarization (%)	Detailed Description (%)
Prefer LLaVA V1.5	27.3	20.1
No Preference	30.5	12.7
Prefer ILuvUI	52.2	67.2

Table 5: Comparison between ILuvUI and LLaVA V1.5 on human evaluations of one-sentence summarization and detailed description, showing that ILuvUI generates more precise and preferable summarization and detailed description.

description for each UI image. We then built a rating user interface on a webpage, where the UI screen image and summarizations from both models were displayed. The summarizations from each model were presented in a randomized order to prevent raters from predicting the source model for a given description. Participants were then asked which summarizations they preferred or if both summarizations were about the same. This process was replicated for the detailed descriptions.

The 1000 UI images were randomly partitioned into 10 sets. Subsequently, 10 participants were tasked with comparing detailed descriptions, evaluating their coverage of information shown in the UI. Another set of 10 participants was assigned to compare one-sentence summarizations, focusing on the precision of the generated summaries for each UI.

As shown in Table 5, in terms of summarization, 40.2% of the responses favored the description generated by ILuvUI, whereas 27.3% preferred LLaVA’s description. The remaining 32.5% were assessed as similar. Regarding detailed descriptions, 67.2% of the responses favored the description generated by ILuvUI, contrasting with 20.1% preferring LLaVA’s description. The remaining 12.7% were considered similar. These results suggest that our ILuvUI produced more comprehensive and preferred detailed descriptions.

6 Application: UI Verification

By fine-tuning our model on the UI transitions from the Crawls interaction trace dataset [26], we enabled it to perform UI verification. This capability allows the model to assess whether user interactions with a UI align with expected behaviors, identifying both correct and erroneous actions.

To establish a benchmark dataset, we conducted an experiment with five participants who were instructed to perform both correct and intentionally incorrect actions on four UIs, resulting in a total of 20 UIs. Participants self-reported their intended targets and evaluated whether they believed their actions were correct. This benchmark includes 100 positive interactions and 100 negative interactions based on user actions across the 20 UIs. We illustrate the positive and negative samples in Figure 7.

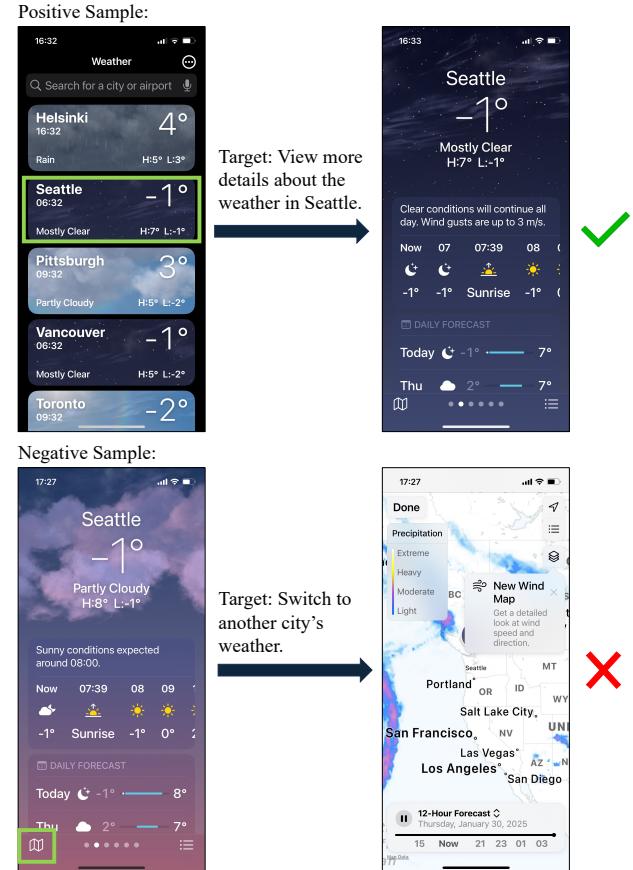


Figure 7: Positive and negative samples for UI verification. Clicking on the selected UI element (green) leads to the target in positive samples but not in negative samples.

We evaluated the model’s performance by examining its ability to verify user interactions across various UI tasks. Specifically, the

Agent	UI Verification (%)			
	Accuracy	Precision	Recall	F1 Score
LLaVA V1.5	51.5	97.0	50.8	66.7
GPT-4V	59.0	61.0	59.0	60.0
ILuvUI	72.0	69.6	78.0	73.6

Table 6: Comparison between ILuvUI, LLaVA V1.5, and GPT-4V on UI verification tasks. The result shows that ILuvUI outperforms the LLaVA V1.5 model and GPT-4V on verifying UI transitions.

model checks the recorded interaction data to determine if the participants' actions matched the expected outcomes. The results are presented in Table 6. LLaVA V1.5 tends to classify transitions as correct, resulting in high precision but low accuracy. Our model demonstrates high accuracy for both positive and negative samples when compared to GPT-4V.

7 Discussion and Future Work

This paper presents an initial promising step toward producing a Vision-Language Model that can perform general UI understanding tasks. Our approach adapts an existing method for the UI domain. We contribute data augmentations that are specific to UIs that go beyond prior work, such as augmentations around the transition from one UI to another and task-specific actions. Our work shows that it can produce UI-centric instruction-tuned behavior despite being trained with synthetic textual data from a large UI dataset of only screenshots. Additionally, training the model on our created specialized dataset of 353K conversational examples paired with UI screenshots enables more UI applications that could not be performed well by previous models such as goal-based planning and UI verification.

Data Quality. Relying on GPT-3.5-turbo for dataset generation may introduce biases and hallucinations. To mitigate this, we carefully designed prompts and employed few-shot examples to guide the model toward producing accurate and contextually relevant outputs. Additionally, our approach used structured inputs for the UI element detection result, to constrain the model's outputs and reduce hallucinations.

Generalizability. The focus of our experimental setup is mobile UIs. However, our data generation pipeline is domain-agnostic and can be extended to other UI domains, such as desktop or web interfaces.

Societal Impact. There are many potential applications in accessibility and software testing for such a model, including providing accessible screen descriptions to vision-impaired users, repairing automated UI tests, or providing contextual help to assist a user with navigating an interface. We have informally completed some of these tasks successfully, and a basic example of our model being used for UI navigation can be seen in Table 7. However, our approach may raise data privacy concerns. Automating UI interactions could inadvertently expose sensitive user data, particularly if the model misinterprets inputs or interacts with unintended UI elements. Future work on UI safety is necessary to ensure responsible deployment and mitigate potential risks.

Evaluation. A more comprehensive evaluation can be designed to fully understand this model's performance and progeny. An open question is: What sort of tasks or benchmarks should be adopted to evaluate the performance of a model on UI tasks? Here, we use some basic UI understanding tasks as initial benchmarks, identifying UI elements' existence, type, and purpose. Further evaluations of this form can include icon recognition and other higher-level tasks like command grounding and tappability prediction. A goal for the research community should be to work towards agreement on a standard set of benchmarks that can be automatically and easily evaluated when new models are produced.

Future Work. Future research could delve deeper into refining the model's reasoning abilities, expanding its knowledge base for UI design, developing novel agents to facilitate effective UI interaction and navigation, and ensuring robustness across broader UI variations.

Future work can focus on the following aspects to further develop the agent:

- Improving dataset quality: While our generated UI dataset has helped in training our ILuvUI agent, there is still room for improving the quality and diversity of the dataset. A deeper analysis of our generated dataset is needed to understand better its flaws, which can lead to improved prompting strategies. Future work on synthetic data could focus on reducing hallucination, adding more annotated data, and varying UIs from different domains. Human-annotated data could also be added, especially to assist with common-sense reasoning and task planning.
- Improving the model: The performance of ILUVUI could be improved by building on top of more advanced vision-language model architectures. In particular, it would be ideal to build upon an image encoder that can function with various input resolutions, including very high ones. For example, such a model could work more easily on traditional desktop user interfaces.
- Support for tools and machine-interpretable output: Currently, ILuvUI seems fairly limited in its ability to produce structured outputs that could be more easily interpreted by a machine, such as JSON. Some applications, particularly UI navigation and software testing, will require the models to work more easily with other traditional software, so additional training and fine-tuning will be required to make this possible and reliable.

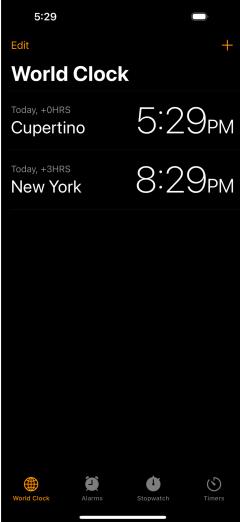
<p>Task: start a stopwatch.</p> 	<p>Human: Describe the screen and its key features, which may be useful for performing various actions on the screen.</p> <p>ILuvUI: [Description of the UI]</p> <p>Human: What action (tap, swipe, enter text) should a user perform on this screen to do the following: start a stopwatch.</p> <p>ILUVUI: To start a stopwatch, the user should tap on the "Stopwatch" tab located at the bottom of the screen.</p>
	<p>Human: Describe the screen and its key features, which may be useful for performing various actions on the screen.</p> <p>ILuvUI: [Description of the UI]</p> <p>Human: What action (tap, swipe, enter text) should a user perform on this screen to do the following: start a stopwatch.</p> <p>ILUVUI: To start a stopwatch, the user should tap on the "Start" button located in the middle of the UI screen. This button is positioned to the right of the "Lap" button.</p>

Table 7: Example demonstrating our agent’s potential capability of planning UI navigation.

8 Conclusion

In this paper, we have shown how to address a significant gap in the current abilities of Vision-Language Models to understand UIs. We proposed a data generation and fine-tuning approach that can adapt VLMs to the realm of UIs. The challenges posed by UIs go beyond static elements and encompass interactive components that introduce both structural and functional complexity. Our approach, building on the data generation and training of LLaVA [41, 42], showcases its versatility in capturing diverse responses relevant to UI understanding and interactions, ranging from individual UI element properties to elaborate UI descriptions, reasoning, potential user actions, and dynamic UI transitions. The performance evaluation of our UI-focused instruction-following visual agent highlights its effectiveness compared to existing models that are not trained on UI data.

References

- [1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Milligan, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. 2022. Flamingo: a Visual Language Model for Few-Shot Learning. arXiv:2204.14198 [cs.CV]
- [2] Deniz Arsan, Ali Zaidi, Aravind Sagar, and Ranjitha Kumar. 2021. App-Based Task Shortcuts for Virtual Assistants. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) (*UIST ’21*). Association for Computing Machinery, New York, NY, USA, 1089–1099. <https://doi.org/10.1145/3472749.3474808>
- [3] Nikola Banovic, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2012. Waken: Reverse Engineering Usage Information and Interface Structure from Software Videos. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (Cambridge, Massachusetts, USA) (*UIST ’12*). Association for Computing Machinery, New York, NY, USA, 83–92. <https://doi.org/10.1145/2380116.2380129>

- [4] Tony Beltramelli. 2018. Pix2code: Generating Code from a Graphical User Interface Screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Paris, France) (*EICS '18*). Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. <https://doi.org/10.1145/3220134.3220135>
- [5] M. L. BERNARDI, G. A. DI LUCCA, and D. DISTANTE. 2009. RE-UWA approach to recover user centered conceptual models from Web applications. *International Journal on Software Tools for Technology Transfer* 11, 6 (2009), 485–501. <https://doi.org/10.1007/s10009-009-0126-1>
- [6] Pavol Bielik, Marc Fischer, and Martin Vechev. 2018. Robust Relational Layout Synthesis from Examples for Android. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 156 (Oct. 2018), 29 pages. <https://doi.org/10.1145/3276526>
- [7] Sara Bunian, Kai Li, Chaima Jemmalni, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. VINS: Visual Search for Mobile User Interface Design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (<conf-loc>, <city>Yokohama/<city>, <country>Japan</country>, </conf-loc>) (CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 423, 14 pages. <https://doi.org/10.1145/3411764.3445762>
- [8] Tsung-Hsiang Chang, Tom Yeh, and Rob Miller. 2011. Associating the Visual Representation of User Interfaces with Their Internal Structures and Metadata. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) (*UIST '11*). Association for Computing Machinery, New York, NY, USA, 245–256. <https://doi.org/10.1145/2047196.2047228>
- [9] Chunyang Chen, Sidong Feng, Zhenchang Xing, Linda Liu, Shengdong Zhao, and Jinsui Wang. 2019. Gallery D.C.: Design Search and Knowledge Discovery through Auto-Created GUI Component Gallery. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 180 (nov 2019), 22 pages. <https://doi.org/10.1145/3359282>
- [10] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. 2018. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. In *Proceedings of the 40th International Conference on Software Engineering (Gothenburg, Sweden) (ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 665–676. <https://doi.org/10.1145/3180155.3180240>
- [11] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xin Xia, Liming Zhu, John Grundy, and Jinsui Wang. 2020. Wireframe-Based UI Design Search through Image Autoencoder. *ACM Trans. Softw. Eng. Methodol.* 29, 3, Article 19 (jun 2020), 31 pages. <https://doi.org/10.1145/3391613>
- [12] Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. 2020. Object Detection for Graphical User Interface: Old Fashioned or Deep Learning or a Combination?. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) (*ESEC/FSE 2020*). Association for Computing Machinery, New York, NY, USA, 1202–1214. <https://doi.org/10.1145/3368089.3409691>
- [13] Sen Chen, Lingling Fan, Chunyang Chen, Minhui Xue, Yang Liu, and Lihua Xu. 2021. GUI-Squatting Attack: Automated Generation of Android Phishing Apps. *IEEE Transactions on Dependable and Secure Computing* 18, 6 (2021), 2551–2568. <https://doi.org/10.1109/TDSC.2019.2956035>
- [14] Sen Chen, Lingling Fan, Ting Su, Lei Ma, Yang Liu, and Lihua Xu. 2019. Automated Cross-Platform GUI Code Generation for Mobile Apps. In *2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile)*. 13–16. <https://doi.org/10.1109/AI4Mobile.2019.8672718>
- [15] Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, Alexander Kolesnikov, Joan Puigcerver, Nan Ding, Kieran Rong, Hassan Akbari, Gaurav Mishra, Linting Xue, Ashish Thapliyal, James Bradbury, Weicheng Kuo, Mojtaba Seyedsosseini, Chao Jia, Burcu Karagol Ayan, Carlos Riquelme, Andreas Steiner, Anelia Angelova, Xiaohua Zhai, Neil Housley, and Radu Soricut. 2023. PaLI: A Jointly-Scaled Multilingual Language-Image Model. *arXiv:2209.06794 [cs.CV]*
- [16] Wei-Lin Chiang, Zhihuo Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyun Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna.lmsys.org (accessed 14 April 2023)* (2023).
- [17] Allen Cypher. 1991. EAGER: Programming Repetitive Tasks by Example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New Orleans, Louisiana, USA) (*CHI '91*). Association for Computing Machinery, New York, NY, USA, 33–39. <https://doi.org/10.1145/108844.108850>
- [18] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. InstructBLIP: Towards General-purpose Vision-Language Models with Instruction Tuning. *arXiv:2305.06500 [cs.CV]*
- [19] Daniel de Souza Baulé, Christiane Gresse von Wangenheim, Aldo von Wangenheim, Jean C. R. Hauck, and Edson C. Vargas Júnior. 2021. Using Deep Learning to Support the User Interface Design of Mobile Applications with App Inventor. In *Proceedings of the XX Brazilian Symposium on Human Factors in Computing Systems* (Virtual Event, Brazil) (*IHC '21*). Association for Computing Machinery, New York, NY, USA, Article 49, 11 pages. <https://doi.org/10.1145/3472301.3484340>
- [20] Biplob 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 *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) (*UIST '17*). Association for Computing Machinery, New York, NY, USA, 845–854. <https://doi.org/10.1145/3126594.3126651>
- [21] G.A. Di Lucca, P. Fasolino, A.R. and IGLINSKI, and P. Tramontana. 2004. Reverse engineering Web applications: the WARE approach. *SOFTWARE MAINTENANCE AND EVOLUTION: RESEARCH AND PRACTICE* 12 (2004), 71–101. <https://doi.org/10.1002/smrv.281>
- [22] Morgan Dixon and James Fogarty. 2010. Prefab: Implementing Advanced Behaviors Using Pixel-Based Reverse Engineering of Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) (*CHI '10*). Association for Computing Machinery, New York, NY, USA, 1525–1534. <https://doi.org/10.1145/1753326.1753554>
- [23] Morgan Dixon, Daniel Leventhal, and James Fogarty. 2011. Content and Hierarchy in Pixel-Based Methods for Reverse Engineering Interface Structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) (*CHI '11*). Association for Computing Machinery, New York, NY, USA, 969–978. <https://doi.org/10.1145/1978942.1979086>
- [24] Morgan Dixon, Alexander Nied, and James Fogarty. 2014. Prefab Layers and Prefab Annotations: Extensible Pixel-Based Interpretation of Graphical Interfaces. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (Honolulu, Hawaii, USA) (*UIST '14*). Association for Computing Machinery, New York, NY, USA, 221–230. <https://doi.org/10.1145/2642918.2647412>
- [25] W. Keith Edwards, Elizabeth D. Mynatt, and Kathryn Stockton. 1995. Access to Graphical Interfaces for Blind Users. *Interactions* 2, 1 (jan 1995), 54–67. <https://doi.org/10.1145/208143.208161>
- [26] Shirin Feiz, Jason Wu, Xiaoyi Zhang, Amanda Swearngin, Titus Barik, and Jeffrey Nichols. 2022. Understanding Screen Relationships from Screenshots of Smartphone Applications. In *27th International Conference on Intelligent User Interfaces*. 447–458.
- [27] Cole Gleason, Amy Pavel, Emma McCamey, Christina Low, Patrick Carrington, Kris M. Kitani, and Jeffrey P. Bigham. 2020. Twitter A11y: A Browser Extension to Make Twitter Images Accessible. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) (*CHI '20*). Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3313831.3376728>
- [28] Darren Guinness, Edward Cutrell, and Meredith Ringel Morris. 2018. Caption Crawler: Enabling Reusable Alternative Text Descriptions Using Reverse Image Search. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) (*CHI '18*). Association for Computing Machinery, New York, NY, USA, 1–11. <https://doi.org/10.1145/3173574.3174092>
- [29] Forrest Huang, John F. Cann, and Jeffrey Nichols. 2019. Swire: Sketch-Based User Interface Retrieval. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland UK) (*CHI '19*). Association for Computing Machinery, New York, NY, USA, 1–10. <https://doi.org/10.1145/3290605.3300334>
- [30] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishaal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. 2021. OpenCLIP. <https://doi.org/10.5281/zenodo.5143773> If you use this software, please cite it as below.
- [31] Yue Jiang, Ruofei Du, Christof Lutteroth, and Wolfgang Stuerzlinger. 2019. ORC Layout: Adaptive GUI Layout with OR-Constraints. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland UK) (*CHI '19*). Association for Computing Machinery, New York, NY, USA, Article 413, 12 pages. <https://doi.org/10.1145/3290605.3300643>
- [32] Yue Jiang, Wolfgang Stuerzlinger, and Christof Lutteroth. 2021. ReverseORC: Reverse Engineering of Resizable User Interface Layouts with OR-Constraints. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3411764.3445043>
- [33] Yue Jiang, Wolfgang Stuerzlinger, Matthias Zwicker, and Christof Lutteroth. 2020. ORCSolver: An Efficient Solver for Adaptive GUI Layout with OR-Constraints. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) (*CHI '20*). Association for Computing Machinery, New York, NY, USA, 1–14. <https://doi.org/10.1145/3313831.3376610>
- [34] T. Katsimpila, Y. Panagis, E. Sakkopoulos, G. Tzimas, and A. Tsakalidis. 2006. Application Modeling using Reverse Engineering Techniques. In *Proceedings of the 2006 ACM symposium on applied computing*. ACM, 1250–1255. <https://doi.org/10.1145/1141277.1141570>
- [35] R. Krosnick, S. W. Lee, W. S. Laseck, and S. Onev. 2018. Expresso: Building Responsive Interfaces with Keyframes. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 39–47. <https://doi.org/10.1109/VLHCC.2018.8506516>
- [36] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & Sharing How-to Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) (*CHI '08*). Association for Computing Machinery, New York, NY, USA, 1719–1728.

- <https://doi.org/10.1145/1357054.1357323>
- [37] Gang Li and Yang Li. 2022. Spotlight: Mobile UI Understanding using Vision-Language Models with a Focus. *ArXiv abs/2209.14927* (2022). <https://api.semanticscholar.org/CorpusID:252595735>
- [38] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) (*CHI '17*). Association for Computing Machinery, New York, NY, USA, 6038–6049. <https://doi.org/10.1145/3025453.3025483>
- [39] Yi-Chi Liao, Kashyap Todi, Aditya Acharya, Antti Keurulainen, Andrew Howes, and Antti Oulasvirta. 2022. Rediscovering Affordance: A Reinforcement Learning Perspective. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) (*CHI '22*). Association for Computing Machinery, New York, NY, USA, Article 362, 15 pages. <https://doi.org/10.1145/3491102.3501992>
- [40] 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 *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 740–755.
- [41] Haotian Liu, Chunyu Li, Yuheng Li, and Yong Jae Lee. 2023. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744* (2023).
- [42] Haotian Liu, Chunyu Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. *arXiv preprint arXiv:2304.08485* (2023).
- [43] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2023. Chatting with GPT-3 for Zero-Shot Human-Like Mobile Automated GUI Testing. *arXiv:2305.09434 [cs.SE]*
- [44] Christof Lutteroth. 2008. Automated Reverse Engineering of Hard-Coded GUI Layouts. In *Proceedings of the Ninth Conference on Australasian User Interface - Volume 76* (Wollongong, Australia) (*AUIC '08*). Australian Computer Society, Inc., AUS, 65–73. <https://doi.org/10.5555/1378337.1378350>
- [45] Melody Moore and Spencer Rugaber. 1997. Using Knowledge Representation to Understand Interactive Systems. In *Proceedings of the 5th International Workshop on Program Comprehension (WPC '97)* (*WPC '97*). IEEE Computer Society, USA, 60.
- [46] Melody M. Moore. 1996. Rule-Based Detection for Reverse Engineering User Interfaces. In *Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96)* (*WCRE '96*). IEEE Computer Society, USA, 42.
- [47] Melody Marie Moore, James D. Foley, and Spencer Rugaber. 1998. *User Interface Reengineering*. Ph.D. Dissertation. USA. AAI9918460.
- [48] 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 (2020), 196–221. <https://doi.org/10.1109/TSE.2018.2844788>
- [49] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse Engineering Mobile Application User Interfaces with REMAUI. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering* (Lincoln, Nebraska) (*ASE '15*). IEEE Press, 248–259. <https://doi.org/10.1109/ASE.2015.32>
- [50] OpenAI. [n.d.]. ChatGPT. <https://openai.com/blog/chatgpt>.
- [51] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).
- [52] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *arXiv:2203.02155 [cs.CL]*
- [53] Baolin Peng, Chunyu Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277* (2023).
- [54] Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. 2011. Pause-and-Play: Automatically Linking ScreenCast Video Tutorials with Applications. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) (*UIST '11*). Association for Computing Machinery, New York, NY, USA, 135–144. <https://doi.org/10.1145/2047196.2047213>
- [55] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.
- [56] A. Sanchez Ramon, J. Sanchez Cuadrado, and J. Garcia Molina. 2012. Model-driven reverse engineering of legacy graphical user interfaces. *Automated Software Engineering* 21, 2 (2012), 147–186. <https://doi.org/DOI:10.1007/s10515-013-0130-2>
- [57] Eldon Schoop, Xin Zhou, Gang Li, Zhourong Chen, Bjoern Hartmann, and Yang Li. 2022. Predicting and Explaining Mobile UI Tappability with Vision Modeling and Saliency Analysis. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) (*CHI '22*). Association for Computing Machinery, New York, NY, USA, Article 36, 21 pages. <https://doi.org/10.1145/3491102.3517497>
- [58] E. Stroulia, M. El-Ramly, P. Iglinski, and P. Sorenson. 2003. User Interface Reverse Engineering in Support of Interface Migration to the Web. *Automated Software Engng.* 10, 3 (July 2003), 271–301. <https://doi.org/10.1023/A:1024460315173>
- [59] Amanda Sweenyngin, Mira Dontcheva, Wilmot Li, Joel Brandt, Morgan Dixon, and Amy J. Ko. 2018. Rewire: Interface Design Assistance from Examples. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) (*CHI '18*). Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3173574.3174078>
- [60] Amanda Sweenyngin, Amy J. Ko, and James Fogarty. 2017. Genie: Input Retargeting on the Web through Command Reverse Engineering. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) (*CHI '17*). Association for Computing Machinery, New York, NY, USA, 4703–4714. <https://doi.org/10.1145/3025453.3025506>
- [61] Amanda Sweenyngin and Yang Li. 2019. Modeling Mobile Interface Tappability Using Crowdsourcing and Deep Learning. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland UK) (*CHI '19*). Association for Computing Machinery, New York, NY, USA, 1–11. <https://doi.org/10.1145/3290605.3300305>
- [62] Rohan Taori, Ishaaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsumori B. Hashimoto. 2023. Stanford Alpaca: An Instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.
- [63] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
- [64] Bryan Wang, Gang Li, and Yang Li. 2023. Enabling Conversational Interaction with Mobile UI Using Large Language Models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) (*CHI '23*). Association for Computing Machinery, New York, NY, USA, Article 432, 17 pages. <https://doi.org/10.1145/3544548.3580895>
- [65] Hao Wen, Yuanchun Li, Guohong Liu, Shanhai Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2023. Empowering LLM to use Smartphone for Intelligent Task Automation. *arXiv:2308.15272 [cs.AI]*
- [66] Thomas D. White, Gordon Fraser, and Guy J. Brown. 2019. Improving Random GUI Testing with Image-Based Widget Detection. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) (*ISSTA 2019*). Association for Computing Machinery, New York, NY, USA, 307–317. <https://doi.org/10.1145/3293882.3330551>
- [67] Jason Wu, Xiaoyi Zhang, Jeff Nichols, and Jeffrey P Bigham. 2021. Screen Parsing: Towards Reverse Engineering of UI Models from Screenshots. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) (*UIST '21*). Association for Computing Machinery, New York, NY, USA, 470–483. <https://doi.org/10.1145/347249.3474763>
- [68] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI Screenshots for Search and Automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (Victoria, BC, Canada) (*UIST '09*). Association for Computing Machinery, New York, NY, USA, 183–192. <https://doi.org/10.1145/1622176.1622213>
- [69] Chen Yongxin, Zhang Tonghui, and Chen Jie. 2019. UI2code: How to fine-tune background and foreground analysis. *Retrieved Feb 23 (2019), 2020*.
- [70] Luke S. Zettlemoyer and Robert St. Amant. 1999. A Visual Medium for Programmatic Control of Interactive Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Pittsburgh, Pennsylvania, USA) (*CHI '99*). Association for Computing Machinery, New York, NY, USA, 199–206. <https://doi.org/10.1145/302979.303039>
- [71] Xiaoyi Zhang, Lilian de Greef, Amanda Sweenyngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, et al. 2021. Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.
- [72] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zuoquan Li, Dacheng Li, Eric Xing, et al. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *arXiv preprint arXiv:2306.05685* (2023).