# Visual Grounding for User Interfaces

**Yijun Qian**[†] **Yujie Lu**[‡] **Alexander G. Hauptmann**[†] **Oriana Riva**[§*]

[†] Carnegie Mellon University
[‡] University of California, Santa Barbara
[§] Google Research

## Abstract

Enabling autonomous language agents to drive application user interfaces (UIs) as humans do can significantly expand the capability of today's API-based agents. Essential to this vision is the ability of agents to ground natural language commands to on-screen UI elements. Prior UI grounding models work by relaying on developer-provided UI metadata (UI trees, such as web DOM, and accessibility labels) to detect on-screen elements. However, such metadata is often unavailable or incomplete. Object detection techniques applied to UI screens remove this dependency, by inferring location and types of UI elements directly from the UI's visual appearance. The extracted semantics, however, are too limited to directly enable grounding. We overcome the limitations of both approaches by introducing the task of *visual UI grounding*, which unifies detection and grounding. A model takes as input a UI screenshot and a free-form language expression, and must identify the referenced UI element. We propose a solution to this problem, *LVG*, which learns UI element detection and grounding using a new technique called layout-guided contrastive learning, where the semantics of individual UI objects are learned also from their visual organization. Due to the scarcity of UI datasets, LVG integrates synthetic data in its training using multi-context learning. LVG outperforms baselines pre-trained on much larger datasets by over 4.9 points in top-1 accuracy, thus demonstrating its effectiveness.

## 1 Introduction

Autonomous language agents that are capable of interacting with real-world applications are emerging (Li et al., 2020; Liu et al., 2018; Kim et al., 2023; Rawles et al., 2023; Zheng et al., 2024). Provided with a task described in natural language, these agents drive application user interfaces as humans do by clicking, typing, scrolling, etc. The myriad of tasks such UI agents could accomplish is potentially unlimited, much beyond what traditional API-based agents can do. In this paper, we focus on a fundamental problem UI agents must solve: *grounding* natural language commands to on-screen elements, i.e., mapping commands such as "enable auto-notification" or "open the second item in the list" to the correct UI action and on-screen element.

Prior work (Bai et al., 2021; Li and Li, 2023) achieves UI grounding by assuming the location bounds and types of UI elements present in a screen are known beforehand. Hence, they define grounding as the problem of ranking a set of UI elements based on the given natural language command. The set of UI elements is computed automatically using developer-provided UI metadata, consisting of UI trees (e.g., web DOM tree or Android View Hierarchy) and accessibility annotations. The issue with this approach is that such UI metadata is often not accessible for security or privacy reasons (XDA, 2021). Developer-provided metadata can also be noisy, corrupted with missing object descriptions or misaligned structure information (Li and Li, 2023). Finally, as others pointed out (Chen et al., 2020a), accessibility labels are generally not provided for all UI elements (see Appendix A.1 for further details). These constraints make these approaches hard to deploy and limit their performance.

Another way of approaching this problem without relying on UI metadata is to train object detection models for UI screens (Chen et al., 2020b; Zhang et al., 2021). This line of work, generally referred to as screen understanding or screen parsing, localizes UI elements in a screen solely from its visual appearance. Elements are labeled with technical terms such as “Button”, “Text-Input”, “Icon”, etc. As these labels carry limited semantic information, they are not sufficient to directly support grounding of natural language commands. This means that a second model, possibly an LLM, must

[*] Work done while at Microsoft Research.

*Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pages 97–107
June 16-21, 2024 ©2024 Association for Computational Linguistics

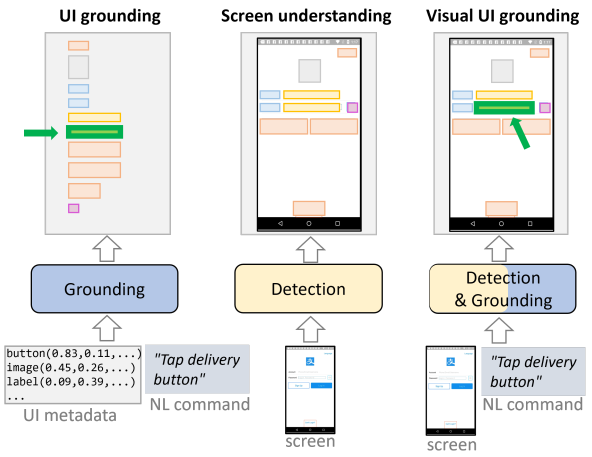


Figure 1: Visual UI grounding unifies the task of UI grounding that relies on the availability of UI metadata and screen understanding which localizes elements in a screen and classifies them into pre-defined types (button, text-label, text-input, icon, etc.). In this new task, a UI element referenced by a natural language command must be localized in the given UI screen, by relying solely on the screen's visual appearance (without assuming UI metadata).

be used to map natural language commands to the detected elements (Yan et al., 2023a). The adoption of a 2-step process causes information loss and increases maintenance and deployment costs.

We address the limitations of both worlds by unifying detection and grounding into the new task of *visual UI grounding*, illustrated in Fig. 1. The model takes as input a UI screenshot (without metadata) and a free-form language expression, and must predict the bounding box of the referenced element. Hence, unlike previously-proposed methods where bounding boxes of candidate UI objects are given as input or pre-computed by a separate model, here a *single* model must perform *both* element detection and action grounding.

In the search for a solution to the problem of visual UI grounding, we first consider recent work on open-vocabulary object detection (Li et al., 2022; Yuan et al., 2021; Gu et al., 2022). These models are trained end-to-end to map natural language expressions to objects in an image. While they represent a perfect fit for our problem, we find that, despite their large training datasets, they do not perform well on UI screens (see baseline comparisons in §5). Our explanation is that these models are trained on real-world scene datasets (Lin et al., 2014; Gupta et al., 2019) where objects of the same appearance (color, shape, size) share similar meanings, whereas UI objects are subject to *application and context sensitivity*. In other words, in UI screens, objects that may look similar have different meanings depending on the application and surrounding UI elements. For example, consider a heart icon which in Facebook loves a post, but in Etsy adds a product to the favorites; if the same icon appears next to a label "click for more" it assumes yet another meaning.

To address the problem of application and context-sensitivity of UI objects, we propose *LVG* (Layout-guided Visual Grounding). We observe that while objects in real-world scenes do not usually follow a regular pattern in their arrangement, UI elements are organized through layouts, which can be key to understanding their meaning. For example the function of an icon or an element in a grid can be better understood by relating it to a nearby text label or to another element spatially aligned to it. Hence, we introduce *layout-guided contrastive learning* where the model learns to classify elements into groups based on their visual containers (headers, lists, tables, etc.). This enforces the target element's features to be closer to those of its sibling elements and far from those of elements in other containers, thus enriching their semantic representations. Application-derived features are then combined with element-specific features. Further, to cope with the lack of UI grounding datasets, we synthetically generate natural language referring expressions paired with original UI screens. We successfully transfer knowledge learned from synthetic to real-user expressions using *multi-context learning*, i.e., forcing the model to generate similar features when synthetic and natural expressions are referring to the same element.

In summary, we make the following contributions: *(i)* we define the task of Visual UI Grounding, *(ii)* we propose a solution, LVG, and introduce layout-based contrastive learning, and *(iii)* we generate a synthetic dataset of diversified language queries and use it effectively through multi-context learning. Overall, LVG surpasses strong baselines by over 4.9 points on top-1 accuracy.

## 2 Related work

**UI grounding** UI grounding models detect UI elements referenced by natural language commands in a screen. Both supervised (Pasupat et al., 2018; Li et al., 2020; Liu et al., 2018; Gur et al., 2019) and unsupervised (He et al., 2021; Bai et al., 2021;

Banerjee et al., 2022) methods rely on deriving the bounding boxes and types of the candidate UI elements (or regions of interest (Li and Li, 2023)) from UI trees (e.g., Android View Hierarchy or web HTML) and often make use of accessibility labels to enhance the UI element representation. This is the case also in recent LLM-based approaches (Wang et al., 2022a; Zheng et al., 2024). The issue with these methods is that UI trees and accessibility labels are often inaccessible (e.g., an Android app cannot access the UI tree of another app) or unavailable (accessibility labels lack both in websites and mobile apps (Chen et al., 2020a)). While web HTML is accessible, raw HTML is large and noisy, often not fitting the input window of LLMs (Zheng et al., 2024), which leads to heuristics being used to reduce its size. For all these reasons, these solution are hard to deploy and scale. LVG performs grounding without depending on UI metadata.

**Screen understanding** Screen understanding (also called screen parsing) models avoid the dependency on UI metadata, by inferring bounding boxes and types of on-screen elements solely from a UI screenshot (Chen et al., 2020b; Zhang et al., 2021; Wu et al., 2021). The inferred class labels ("button", "radio-button", "slider", "text-input", etc.), however, are semantically very limited to directly enable grounding of open-vocabulary referring expressions. For this reason these methods must be paired with a second model, an LLM or VLM, for language grounding. Rawles et al. (2023) use screen understanding techniques based on a combination of OCR and IconNet (Sunkara et al., 2022) to detect elements on the screen and produce a textual representation of the UI. Then, they train a grounding model using behavioural cloning or use LLMs in a zero/few-shot manner to identify the referenced element. Another 2-step approach (Yan et al., 2023a) which involves GPT-4V uses the same screen understanding techniques to identify bounding boxes of relevant elements, which are then represented by visually adding numeric tags to the UI image (Yang et al., 2023). Finally, Pix2Act (Shaw et al., 2023) adopts Pix2Struct (Lee et al., 2022) (consisting of an image encoder and text decoder) to first transform UI screenshots of MiniWob (Shi et al., 2017) synthetic webpages into simplified HTML and then apply behavioural cloning, reinforcement learning or Monte Carlo Tree Search. The main downside of these approaches is that the preliminary step of converting UI screenshots into textual representations or annotating UI images with numeric tags causes information loss. Some elements may be missed, and especially text-only representations are not well suited for visual elements such as icons and symbols. The two-step approach also increases the deployment costs from one model to two. In our approach, one model is trained end to end, thus lowering the deployment costs and avoiding any lossy pre-processing.

**Open-vocabulary object detection** Recent work in the computer vision community tackles the problem of open-vocabulary object detection (Joseph et al., 2021; Li et al., 2022; Zhong et al., 2022; Gu et al., 2022; Kaul et al., 2023), where a model is tasked to detect classes of objects that have not been introduced to it before. RegionCLIP (Zhong et al., 2022) learns a regional visual-semantic space that covers rich object concepts such that it can be used for open-vocabulary object detection. GLIP (Li et al., 2022) unifies grounding and detection tasks by reformulating object detection as phrase grounding, thus being able to learn from both detection and grounding datasets. While related to our goal, these methods are designed for images and objects that represent real-world scenes. When fed with UI datasets their performance is inferior because UI screens exhibit some unique features (see results in §5 and Appendix A.3). To address UI-specific challenges we introduce layout-guided contrastive learning and leverage global-local feature aggregation.

## 3 Method

A key contribution of our work is to address the problem of application and context sensitivity which characterizes UI screens. Application sensitivity occurs with UI elements that despite their similar appearance have different functionality in different applications (e.g., a “hand” symbol in a video call application or in a drawing application have completely different functions). Context sensitivity occurs with UI elements that change their functionality depending on "context", i.e., neighboring UI elements (e.g., a list item must be considered in the context of the other items appearing in the same list or a text-label can change the meaning of a symbol located next to it).

Next, we describe how LVG addresses these challenges. Fig. 2 shows the architecture of LVG. We use SWIN Transformer (Liu et al., 2021) as the

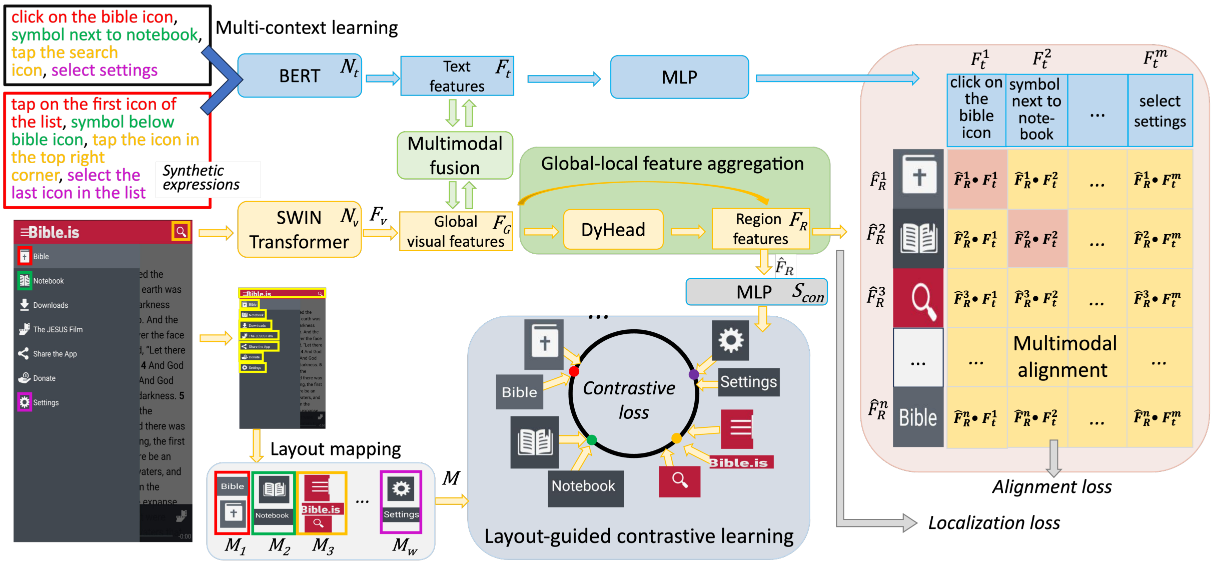

click on the bible icon, symbol next to notebook, tap the search icon, select settings
Multi-context learning
tap on the first icon of the list, symbol below bible icon, tap the icon in the top right corner, select the last icon in the list
Synthetic expressions
BERT $N_t$
Text features $F_t$
MLP
Multimodal fusion
SWIN Transformer $N_v$
$F_v$
Global visual features $F_G$
Global-local feature aggregation
DyHead
Region features $F_R$
$\hat{F}_R$
MLP $S_{con}$
Bible.is
Bible
Notebook
Downloads
The JESUS Film
Share the App
Donate
Settings
Layout mapping
$M$
$M_1$
$M_2$
$M_3$
$M_w$
Contrastive loss
Layout-guided contrastive learning
$F_t^1$
$F_t^2$
$F_t^m$
click on the bible icon
symbol next to note-book
...
select settings
$\hat{F}_R^1$
$\hat{F}_R^2$
$\hat{F}_R^3$
$\hat{F}_R^n$
$\hat{F}_R^1 \bullet F_t^1$
$\hat{F}_R^1 \bullet F_t^2$
$\hat{F}_R^1 \bullet F_t^m$
$\hat{F}_R^2 \bullet F_t^1$
$\hat{F}_R^2 \bullet F_t^2$
$\hat{F}_R^2 \bullet F_t^m$
$\hat{F}_R^3 \bullet F_t^1$
$\hat{F}_R^3 \bullet F_t^2$
$\hat{F}_R^3 \bullet F_t^m$
Multimodal alignment
$\hat{F}_R^n \bullet F_t^1$
$\hat{F}_R^n \bullet F_t^2$
$\hat{F}_R^n \bullet F_t^m$
Alignment loss
Localization loss

Figure 2: LVG architecture.

visual backbone $\mathcal{N}_v$, to extract visual features $F_v$ from UI screens, and BERT (Devlin et al., 2019) as text backbone $\mathcal{N}_t$, to extract textual features $F_t$ from natural language commands.

**Application sensitivity** We fuse visual and text features using a multimodal fusion module (Li et al., 2022). Specifically, we use multiple head attention structures to fuse features from the two modalities. Inspired by the design of the residual block of ResNet (He et al., 2015), to account for application-level information in element recognition, we build a shortcut that concatenates global features (extracted from the whole UI screenshot) with pooled region proposal features generated by Dynamic Head (Dai et al., 2021). Two task specific head modules, which are implemented as Multi-Layer Perceptron (MLP), are designed to perform the regression of bounding box locations and classification of element labels based on the features derived from the pooled region proposals.

We use an attention layer ($Attn$) to get the fused region features $\hat{F}_R \in \mathbb{R}^{n\times d}$ from the global features $F_G \in \mathbb{R}^{1\times D}$ and proposal feature $F_R \in \mathbb{R}^{n\times d}$, where $n$ is the number of region proposals and $d$ is the degree of feature space:

$$\hat{F}_R = Attn(F_G, F_R)[1:] \tag{1}$$

**Context sensitivity** A possible solution to this problem is to augment the features of each region proposal with those of spatially-close regions. We tried different settings such as fusing features of horizontal regions, fusing features of vertical regions, and fusing features of both horizontal and vertical regions. However, none of these approaches worked effectively because features from irrelevant regions were often included. In fact, being two UI elements spatially close does not automatically imply they have a relationship. For example, a caption may be related to the image appearing above or below it, and two text-labels may or may not have a relationship depending on whether they are spatially aligned and on whether they use the same font size and color. Instead, we observe that we have a reliable source of contextual information which has been overlooked by prior work: *UI layouts*. Layouts enforce how UI elements are grouped and organized in visible or invisible *containers*, such as lists, headers, or navigation bars, which are in fact critical to help humans understand and navigate UIs. Layouts not only allow us to identify nearby UI elements that are relevant to a target element but also to exclude elements that despite their spatial closeness are irrelevant.

We leverage UI trees included in public datasets (Deka et al., 2017) to teach the model how to recognize layout structures from visual inputs only. At inference time, the model does not actually take UI trees as input. UI trees provide a hierarchical representation of the UI where each node in the tree may contain any number of nodes. We process UI trees to extract a multi-level tree representation including leaf nodes (the visible UI elements) and containers, such as lists, grids and navigation bars (regardless of whether they are explicitly drawn in the UI). We compute each leaf node's bounding box

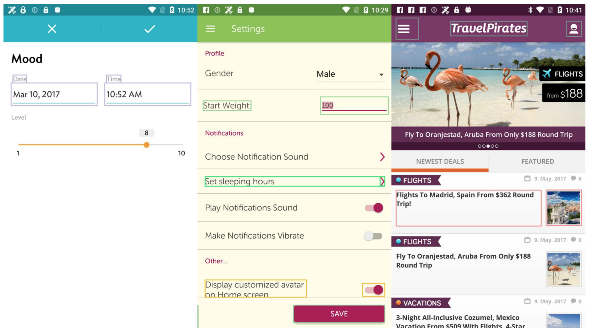

10:52
Mood
Date
Mar 10, 2017
Time
10:52 AM
Level
8
1
10
10:29
Settings
Profile
Gender
Male
Start Weight:
100
Notifications
Choose Notification Sound
Set sleeping hours
Play Notifications Sound
Make Notifications Vibrate
Other...
Display customized avatar on Home screen
SAVE
10:41
TravelPirates
FLIGHTS
from $188
Fly To Oranjestad, Aruba From Only $188 Round Trip
NEWEST DEALS
FEATURED
FLIGHTS
9. May. 2017
6
Flights To Madrid, Spain From $362 Round Trip!
FLIGHTS
9. May. 2017
0
Fly To Oranjestad, Aruba From Only $188 Round Trip
VACATIONS
9. May. 2017
0
3-Night All-Inclusive Cozumel, Mexico Vacation From $509 With Flights, 4-Star

Figure 3: Examples of element groupings as predicted by LVG. The same color represents elements in the same container. We do not report all detected groupings to make the visualization more readable. LVG is able to correctly group together icons, texts and buttons belonging to the same navigation bar as well as date pickers, icons and sliders with the corresponding text labels.

(based on location bounds provided in the UI tree) and use the parent container information extracted from the UI tree to identify its siblings. If a node has no siblings under its direct parent container, we recursively traverse the tree until we find one. Hence, we build a mapping between elements and containers as $\mathcal{M} = \{M_1, M_2, ..., M_w\} \in \mathbb{R}^{w \times c}$, where $w$ is the number of containers and $c$ is the number of elements. Fig. 2 shows some examples of layout mapping where icons are grouped with their associated text labels despite the container not being visible in the UI screenshot. Additionally, Fig. 3 demonstrates the layout grouping capabilities learned by LVG through various examples including header bars, date pickers, and list items.

Then, we introduce *layout-guided contrastive learning*. The contrastive loss aims to separate elements into groups, where each group contains a target element and its siblings. Given the fused region features $\hat{F}_R$ and the element-container mapping $M$, we compute the contrastive loss $\mathcal{L}_{con} = loss_{xe}(S_{con}; \mathcal{M})$, where $S_{con} = \mathcal{N}_{con}(\hat{F}_R)$. $\mathcal{N}_{con}$ is a Multi-Layer Perceptron that projects region features to a probability distribution of layout containers $S_{con} \in \mathbb{R}^{w \times c}$ and $loss_{xe}$ is a cross-entropy function.

In addition to contrastive loss, we implement an alignment loss $\mathcal{L}_{aln} = loss_{xe}(S_{aln}; \mathcal{T})$, where $S_{aln} = \phi(\hat{F}_R)\phi(F_t)^T$ is the probability distribution of alignments between region proposals and referring expressions. Similar to $\mathcal{M}$, $\mathcal{T} \in \mathbb{R}^{n \times m}$ is a mapping dictionary that records the ground truth alignments between elements and phrases ($\phi$ represents the normalization function). Finally, we add a standard localization loss $\mathcal{L}_{loc}$ to optimize the localization task (Ren et al., 2015).

## 4 Datasets and data synthesis

For training, we use the UIBert dataset (Bai et al., 2021),[1] which contains 16,660 referring expressions associated with a total of 5,682 Android UI screenshots. We also complement this human-collected dataset with a synthetic dataset. We obtain UI screens of original Android apps from the Rico dataset (Deka et al., 2017) and use the UI tree information associated with each screenshot to determine a set of cues from which we heuristically generate referring expressions. Our cues extend those proposed in RicoSCA (Li et al., 2020) where every expression consists of an operation (a verb, such as "tap") and a target element. We make various improvements to RicoSCA to increase the diversity of the generated expressions, and add layout-based cues. We generate expressions only for interactable UI elements (buttons, input fields, icons, etc.) through the following process.

First, we assemble a collection of operational phrases such as "click xxx", "select xxx", "type xxx", "tap xxx", "go to xxx". Each phrase consists of a verb and a placeholder xxx. Second, we establish a set of rules to replace "xxx" placeholders with one or multiple object identifying expressions. These expressions are generated using UI tree information. For example, a UI tree may list an object of type "button", with name "Cancel", with location bounds $x_1$,$y_1$,$x_2$,$y_2$, and with property *clickable=true*. We create rules to produce object expressions such as "the button with name Cancel" or "the Cancel button" or simply "Cancel". In general, we identify a target element using its name (accessibility label, textual content), type (class name) or location. We generate location-based object expressions using the location bounds of the object and the neighboring objects to obtain object descriptions such as "at the top of the page" (using absolute location) or "appearing in the menu next to the login button" (using relative location). Third, we create multiple rules based on the object's properties to determine which operational phrases can be applied to an object. For example,

[1] At the time this work was done very few UI datasets existed. The Android in the Wild (AitW) (Rawles et al., 2023) and Mind2Web (Deng et al., 2023) datasets were released recently. While focused on UI automation scenarios, they contain high-level task instructions rather than low-level referring expressions and are therefore not suitable for this study. AitW also does not contain accessibility trees.

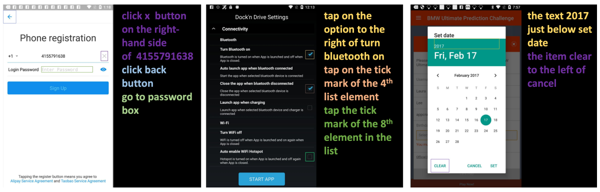

Phone registration
+1
4155791638
Login Password
Enter Password
Sign Up
Tapping the register button means you agree to Alipay Service Agreement and Taobao Service Agreement
click x button on the right-hand side of 4155791638
click back button
go to password box
Dock'n Drive Settings
Connectivity
Bluetooth
Turn Bluetooth on
Auto launch app when bluetooth connected
Close the app when bluetooth disconnected
Launch app when charging
Wi-Fi
Turn WiFi off
Auto enable WiFi Hotspot
START APP
tap on the option to the right of turn bluetooth on
tap on the tick mark of the 4th list element
tap the tick mark of the 8th element in the list
BMW Ultimate Prediction Challenge
Set date
2017
Fri, Feb 17
February 2017
CLEAR
CANCEL
SET
the text 2017 just below set date
the item clear to the left of cancel

Figure 4: Examples of generated synthetic expressions. The expression with a specific color is referring to the element within the bounding box of the same color.

if the object's property "clickable" is set to true and its type is "button", operational phrases such as "click xxx" or "tap xxx" can be applied to it. Finally, for each object we assemble multiple referring expressions. The selected operational phrases are instantiated using one or multiple object expressions. For instance, the phrase "tap xxx" selected for the "Cancel" element described above is instantiated as "tap cancel" or "tap the cancel button" or "tap the cancel button next to login". Fig. 4 shows some examples of generated synthetic expressions.

Overall, we generated 22,617 synthetic expressions for 21,282 Android UI screens. We found that simply mixing UIBert's real-user expressions with synthetic ones did not bring noticeable improvements due to the domain gap (synthetic expressions can be longer and the ratio of referring queries using relative location is higher). We adopt **multi-context learning** which in robotics has been shown to successfully combine together imitation learning datasets of different sizes and nature (Lynch and Sermanet, 2021). We find it is important to generate for each UIBert expression a synthetic counterpart, for the same referred element. This forces the model to map both types of expressions to the same space, and to ultimately leverage the larger size of synthetic data.

## 5 Evaluation

We train and evaluate on the UIBert dataset (Bai et al., 2021) using the official splits. We expand the UIBert train set with 22,617 synthetic expressions. As evaluation metric we use **acc@k** with IoU > 0.5, which measures the fraction of correctly identified UI elements in the top $k$ ranked results.

We compare against 3 baselines: GLIP (Li et al., 2022), OFA (Wang et al., 2022b), and UNINEXT (Yan et al., 2023b). (UNINEXT and OFA currently rank first and third, respectively, in the RefCOCO leaderboard (ref, 2023).) All models are trained on UIBert with or without synthetic data (as specified). For all experimental settings see the Appendix (A.2).

Table 1: LVG performance compared to baselines when trained on UIBert and synthetic data.

| Backbone | Method | Val Acc | | Test Acc | |
|---|---|---|---|---|---|
| | | @1 | @5 | @1 | @5 |
| GLIP | GLIP | 38.42 | 54.98 | 31.27 | 52.33 |
| | GLIP_synt | 40.27 | 55.20 | 33.98 | 54.85 |
| | LVG | 38.85 | 62.05 | 33.74 | 55.92 |
| | LVG_synt | **42.60** | **64.68** | **35.19** | **58.74** |
| OFA | OFA | 37.79 | 55.71 | 37.88 | 56.88 |
| | OFA_synt | 41.80 | 62.19 | 40.27 | 59.26 |
| | LVG | 43.78 | 63.48 | 42.99 | 63.51 |
| | LVG_synt | **45.67** | **64.40** | **45.19** | **65.25** |
| UNINEXT | UNINEXT | 36.11 | 54.82 | 32.19 | 51.93 |
| | UNINEXT_synt | 36.72 | 55.30 | 32.48 | 51.46 |
| | LVG | 38.19 | 57.21 | 34.03 | 53.28 |
| | LVG_synt | **38.22** | **58.20** | **35.67** | **53.88** |

Table 2: Ablation analysis. Models trained on UIBert.

| Method | Val Acc | | Test Acc | |
|---|---|---|---|---|
| | @1 | @5 | @1 | @5 |
| GLIP | 38.42 | 54.98 | 31.27 | 52.33 |
| GLIP + LContast | **40.33** | **62.84** | **33.80** | 55.09 |
| GLIP + Glob-Loc | 39.06 | 60.93 | 32.03 | **58.93** |
| LVG | 38.85 | 62.05 | 33.74 | 55.92 |

**Main results** As shown in Table 1, LVG consistently outperforms the tree baselines on both validation and test sets, demonstrating the efficacy of layout-guided contrastive learning. The best results are obtained with the OFA backbone and synthetic data, where LVG_synt's test acc@1 is 45.2% (acc@5 is 65.3%); this is 4.92 (5.99) percentage points higher than OFA_synt. As the error analysis in §A.3 shows, OFA fails because it does not manage to leverage the spatial context of the target object. We also observe how all tested models benefit from synthetic data, thus demonstrating our multi-context learning approach is successful at transferring knowledge from the synthetic domain to the natural descriptions.

**Ablation analysis** For ablation purposes we use the GLIP backbone because it is less compute intensive. We add layout-guided contrastive learning (LContrast) and global-local feature aggregation (Glob-Loc) to GLIP, and train on UIBert. As Table 2 shows, LContrast surpasses the baseline by 2.53 points in test acc@1 (2.76 in acc@5) demonstrating its effectiveness over traditional contrastive learning for UI tasks. Glob-Loc also surpasses the baseline by 0.76 points in acc@1 (6.6 in acc@5). The full LVG model does not achieve the best performance on all metrics, possibly due to the small size of UIBert, which increases model overfitting as the number of parameters increases. To better appreciate LVG's layout detection capabilities we provide examples of grouping predictions in Fig. 3

## 6 Limitations

LVG was evaluated on an Android dataset. We acknowledge that the dense layouts of desktop UIs may make the visual UI grounding task more challenging. Moreover, there are UI structures such as tables, charts and specialized grids which are not included in our datasets and that may bring additional challenges. Referring expressions can also vary widely. So far we have focused on relatively-short referring expressions. Ideally, LVG should be able to support expressions ranging from very short, under-specified commands (as those characterizing voice-based scenarios) to long and detailed instructions (as those found in instruction manuals). Finally, we acknowledge that the model is trained and tested on referring expressions that are always possible. In real world scenarios a user may refer to a UI element that is not present on the screen.

## 7 Conclusions

We propose the new task of visual UI grounding and present our solution to it. Compared to strong baselines trained on much larger datasets, LVG's layout-guided contrastive learning and multi-context approach for synthetic data demonstrate great improvements in identifying UI elements referenced by NL expressions.

## 8 Ethical considerations

LVG uses some human-labeled data (UIBert dataset), but also demonstrates how synthetic referring expressions can help improve model performance and scale to many different types of application. We think that investing further in synthetic data generation can alleviate the risk of training visual grounding models that work only for certain types of application or platform.

A possible use case for our techniques are screen readers for visually-impaired users. Accessibility labels are often missing or incompletely defined; LVG can enable visually-impaired users to access a much wider range of applications. Another potential use case of LVG is task automation. This use case has tremendous opportunities to advance human productivity. On the other hand, we acknowledge that it also has societal and safety implications (e.g., what if an agent fails in the execution and take irreversible actions?).

## References

2023. Referring expression comprehension on RefCOCO.

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 2021*, pages 1705–1712. ijcai.org.

Pratyay Banerjee, Shweti Mahajan, Kushal Arora, Chitta Baral, and Oriana Riva. 2022. Lexi: Self-supervised learning of the UI language. In *Proc. of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

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.

Xiyang Dai, Yinpeng Chen, Bin Xiao, Dongdong Chen, Mengchen Liu, Lu Yuan, and Lei Zhang. 2021. Dynamic head: Unifying object detection heads with attentions. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7369–7378.

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.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a generalist agent for the web.

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)*, pages 4171–4186. Association for Computational Linguistics.

Google Research Blog. 2023. A vision-language approach for foundational UI understanding. https://ai.googleblog.com/2023/02/a-vision-language-approach-for.html.

Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. 2022. Open-vocabulary object detection via vision and language knowledge distillation. In *International Conference on Learning Representations*.

Agrim Gupta, Piotr Dollar, and Ross Girshick. 2019. LVIS: A dataset for large vocabulary instance segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*.

Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. 2019. Learning to Navigate the Web. In *7th International Conference on Learning Representations (ICLR '19)*.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

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.

K J Joseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. 2021. Towards open world object detection. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5830–5840.

Prannay Kaul, Weidi Xie, and Andrew Zisserman. 2023. Multi-modal classifiers for open-vocabulary object detection.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks.

Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2022. Pix2struct: Screenshot parsing as pretraining for visual language understanding.

Gang Li and Yang Li. 2023. Spotlight: Mobile UI understanding using vision-language models with a focus. In *Proc. of the 11th International Conference on Learning Representations*.

Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. 2022. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10965–10975.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile UI action sequences. In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 8198–8210. Association for Computational Linguistics.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C. Lawrence Zitnick. 2014. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

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)*.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Corey Lynch and Pierre Sermanet. 2021. Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*.

Panupong Pasupat, Tian-Shun Jiang, Evan 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*, pages 4970–4976. Association for Computational Linguistics.

Chris Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. In *NeurIPS 2023 Datasets and Benchmarks Track*.

Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149.

Peter Shaw, Mandar Joshi, James Cohan, Jonathan Berant, Panupong Pasupat, Hexiang Hu, Urvashi Khandelwal, Kenton Lee, and Kristina Toutanova. 2023. From pixels to ui actions: Learning to follow instructions via graphical user interfaces.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of Bits: An Open-Domain Platform for Web-Based Agents. In *34th International Conference on Machine Learning (ICML '17)*, volume 70, pages 3135–3144.

Srinivas Sunkara, Maria Wang, Lijuan Liu, Gilles Baechler, Yu-Chung Hsiao, Jindong Chen, Abhanshu Sharma, and James W. W. Stout. 2022. Towards better semantic understanding of mobile interfaces. In *Proc. of the 29th International Conference on Computational Linguistics*, pages 5636–5650. International Committee on Computational Linguistics.

Bryan Wang, Gang Li, and Yang Li. 2022a. Enabling conversational interaction with mobile ui using large language models. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.

Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. 2022b. OFA: unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International Conference on Machine Learning, ICML 2022*, volume 162, pages 23318–23340.

Jason Wu, Xiaoyi Zhang, Jeff Nichols, and Jeffrey P Bigham. 2021. Screen parsing: Towards reverse engineering of UI models from screenshots. In *Proc. of the 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21, pages 470–483.

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/.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, Zicheng Liu, and Lijuan Wang. 2023a. GPT-4V in Wonderland: Large multimodal models for zero-shot smartphone GUI navigation.

Bin Yan, Yi Jiang, Jiannan Wu, Dong Wang, Zehuan Yuan, Ping Luo, and Huchuan Lu. 2023b. Universal instance perception as object discovery and retrieval. In *CVPR*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v.

Lu Yuan, Dongdong Chen, Yi-Ling Chen, Noel Codella, Xiyang Dai, Jianfeng Gao, Houdong Hu, Xuedong Huang, Boxin Li, Chunyuan Li, et al. 2021. Florence: A new foundation model for computer vision. *arXiv preprint arXiv:2111.11432*.

Xiaoyi Zhang, Lilian de Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, Aaron Everitt, and Jeffrey P Bigham. 2021. Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels. In *Proc. of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Yiwu Zhong, Jianwei Yang, Pengchuan Zhang, Chunyuan Li, Noel Codella, Liunian Harold Li, Luowei Zhou, Xiyang Dai, Lu Yuan, Yin Li, et al. 2022. Regionclip: Region-based language-image pretraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16793–16803.

# A Appendix

## A.1 UI metadata

UI metadata consists of the underlying tree-structured representation of an application UI (called View Hierarchy on Android and DOM on web) and accessibility labels. This metadata is not always available and can be incomplete. Even when available, it may not be accessible.

Technical reasons make UI metadata hard to obtain. On Android, UI metadata is observable through the Accessibility Service. However, for security and privacy reasons, Google heavily restricts who can access it (XDA, 2021). Even when the Accessible Service can be invoked, elements rendered using OpenGL, Unity and Canvas are not included in the retrieved View Hierarchy. This is true also for elements residing inside WebViews which are common in Android apps. View Hierarchies can also present misaligned structure information (Google Research Blog, 2023; Zhang et al., 2021). Accessibility labels are extremely useful to infer the semantics of UI elements. However, they are rare. A previous study reported that more than 77% of 10k Android apps have missing accessibility labels (Chen et al., 2020a).

In desktop apps, accessing UI trees is generally more difficult. For example, the UI tree of common Electron apps like Microsoft Teams are not accessible from the Windows UI Inspector service. Finally, while web DOM trees are generally accessible, they can be very large and noisy, and hence hard to interpret.

## A.2 Implementation details

We train on the UIBert dataset (Bai et al., 2021)[2] using the official splits: train: 4,646 images, 15,624 expressions, validation: 471 images, 471 expressions, test: 565 images, 565 expressions. We expand the train split of UIBert with 22,617 synthetic expressions (no longer than 55 words) generated for 21,282 different Android UI screens.

For experiments with GLIP, we use GLIP-base (SWIN Transformer (Tiny) and BERT) as default backbone. Following the GLIP settings, SWIN-Tiny is pre-trained on ImageNet (Deng et al., 2009), and the input images are resized to 224 × 224 pixels. Models are trained for 100 epochs.

For experiments with OFA, we use OFA-base (ResNet101 and BART-base) initialized with the same pretrained weights. The input images are resized to 384 × 384 pixels. Models are trained for 50 epochs.

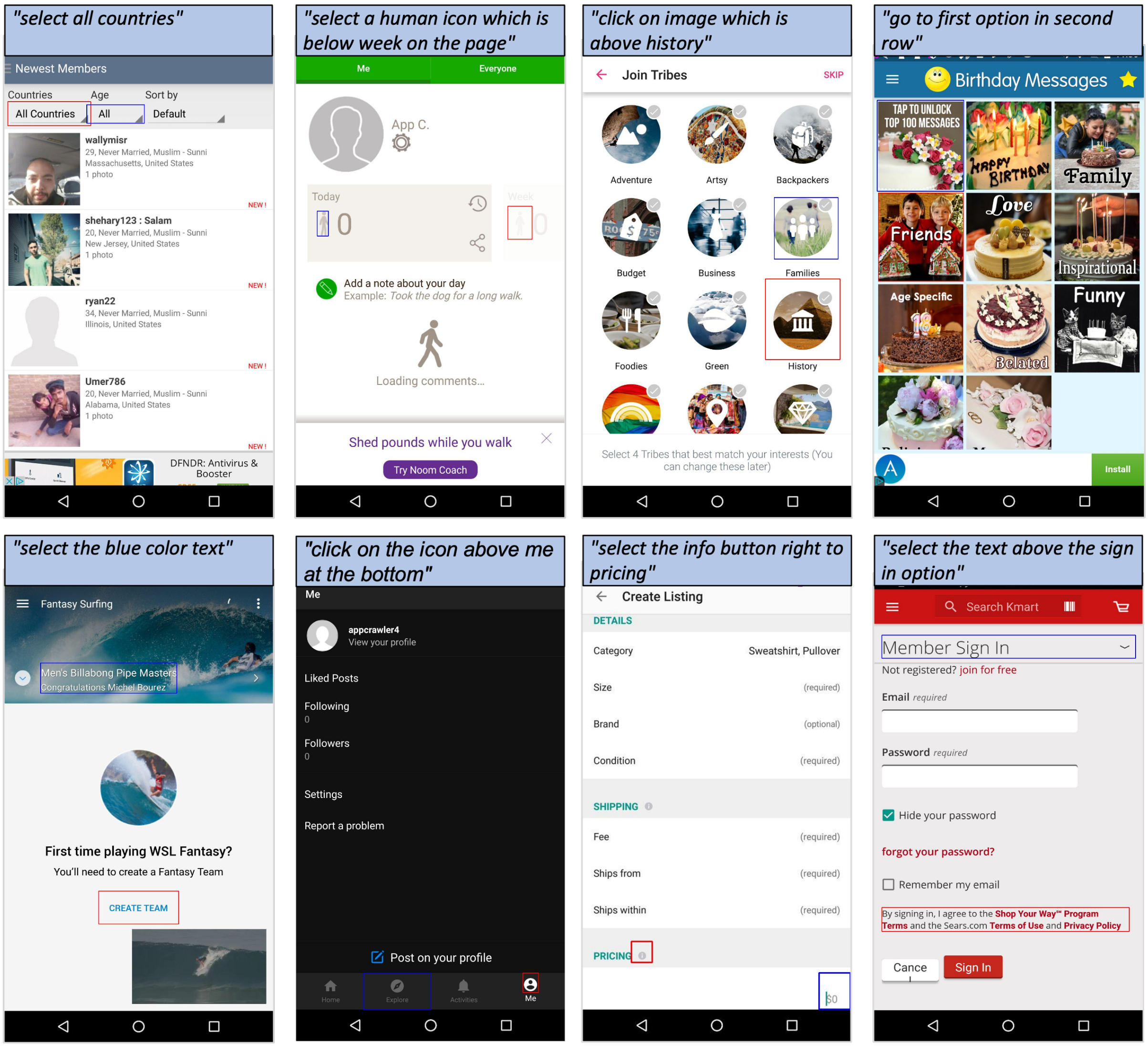
For experiments with UNINEXT, we use UNINEXT-base (ResNet-50 and BERT) as the default backbone, initialized with weights pretrained on Objects365. The images are pre-processed with the same procedure as in UNINEXT. Models are trained for 50 epochs.

For all the settings, the models are optimized by AdamW (Loshchilov and Hutter, 2017) with initial learning rate of $1e^{-4}$, and weight decay of 0.05. The best models are selected based on the results on the validation split.

## A.3 OFA error analysis

In Fig. 5 we show failure cases of the OFA model on the UIBert dataset. Note that in these tests LVG correctly identifies the referenced element. The errors show how OFA does not manage to leverage the spatial context of a target object, which is described by words such as “above”, “below”, or “right to” in the referring expression. Understanding localization in a grid (“first option in second row”) is also challenging. In some cases, the prediction is wrong due to closely-related elements, but also in these cases understanding the spatial layout can help the model (e.g., in the first example, LVG can use layout-guided contrastive learning to group the text “All countries” with “Countries” and the text “All” with “Age”, thus identifying the correct referenced object).

[2] released under license CC BY 4.0



"select all countries"

Newest Members

Countries Age Sort by

All Countries All Default

wallymisr
29, Never Married, Muslim - Sunni
Massachusetts, United States
1 photo
NEW !

shehary123 : Salam
20, Never Married, Muslim - Sunni
New Jersey, United States
1 photo
NEW !

ryan22
34, Never Married, Muslim - Sunni
Illinois, United States
NEW !

Umer786
20, Never Married, Muslim - Sunni
Alabama, United States
1 photo
NEW !

DFNDR: Antivirus & Booster

"select a human icon which is below week on the page"

Me Everyone

App C.

Today 0

Week 0

Add a note about your day
Example: *Took the dog for a long walk.*

Loading comments...

Shed pounds while you walk

Try Noom Coach

"click on image which is above history"

Join Tribes SKIP

Adventure Artsy Backpackers

Budget Business Families

Foodies Green History

Select 4 Tribes that best match your interests (You can change these later)

"go to first option in second row"

Birthday Messages

TAP TO UNLOCK TOP 100 MESSAGES

Happy Birthday Family

Friends Love Inspirational

Age Specific Belated Funny

Install

"select the blue color text"

Fantasy Surfing

Men's Billabong Pipe Masters
Congratulations Michel Bourez

First time playing WSL Fantasy?

You'll need to create a Fantasy Team

CREATE TEAM

"click on the icon above me at the bottom"

Me

appcrawler4
View your profile

Liked Posts

Following
0

Followers
0

Settings

Report a problem

Post on your profile

Home Explore Activities Me

"select the info button right to pricing"

Create Listing

DETAILS

Category Sweatshirt, Pullover

Size (required)

Brand (optional)

Condition (required)

SHIPPING

Fee (required)

Ships from (required)

Ships within (required)

PRICING

$0

"select the text above the sign in option"

Search Kmart

Member Sign In

Not registered? join for free

Email *required*

Password *required*

☑ Hide your password

forgot your password?

□ Remember my email

By signing in, I agree to the **Shop Your Way™ Program Terms** and the Sears.com **Terms of Use** and **Privacy Policy**

Cancel Sign In

Figure 5: Examples of grounding errors of the OFA model. LVG correctly grounds these commands. Red-colored bounding boxes are the ground-truth elements correctly idenfitied by LVG. Blue-colored bounding boxes are the OFA predictions.