UIPro: Unleashing Superior Interaction Capability For GUI Agents

Supplementary Material

7. Details of UIPro Datasets

We list the data sources used in the 20.6M GUI understanding dataset (Sec. 3.2.1) in Tab. 16 and those in the unified GUI agent task (Sec. 3.2.2) in Tab. 18.

GUI understanding data:

Common Crawl: Following [22], we select the pages from the top-200 domains in Common Crawl and design an inhouse web crawler that interacts with elements rendered on the web page and collects interaction trajectories. Subsequently, we use the AutoGUI [22] pipeline to generate functionality grounding and referring tasks.

Android Emulator: We set up virtual Android phones on Android Emulator and collect interaction trajectories on GUIs, including the home page, drop-down panel, settings page, and Apps drawer. Likewise, we use the AutoGUI [22] pipeline to generate functionality grounding and referring tasks.

RICO [13]: We use the element annotations prepared by SeeClick [12] and generate element grounding and referring tasks using RICO element descriptions as referring expressions.

MobileViews [17]: This dataset provides massive GUIs recorded on 20k mobile apps.We generate text localization, OCR, intent grounding, and widget listing tasks using the GUI metadata of this dataset.

WAE [9]: This dataset also provides large-scale GUI metadata, which are originally used for assisting GUI design. Similar to MobileViews, we generate grounding and referring tasks from GUI metadata. As this data source provides accurate element properties. We also generate tasks specific to iconic elements.

WebUI [48]: This dataset also provides large-scale GUI metadata. We generate GUI understanding tasks as we do for WAE.

MultiUI [30]: This dataset provides massive GUI-related Q&A tasks, which are cleaned and incorporated into our dataset to enhance the model's capability of understanding various aspects of GUIs.

GUIEnv [10]: This dataset contains only text localization and OCR tasks, which are both cleaned and incorporated into our dataset

SeeClick-Web [12]: This dataset contains only text localization and OCR tasks in web scenarios, which are cleaned and incorporated into our dataset.

OmniAct [21]: This dataset contains element annotations on web and desktop scenarios, which are used to generate intent grounding tasks.

MOTIF [7]: This dataset contains action trajectories collected on mobile apps. We convert the actions into intent-grounding tasks.

GUI agent task data:

AITW [38]: AITW is released by Google Research, providing massive Android app interaction trajectories. We use the train/test splits provided by SeeClick [12] and incorporate the cleaned training samples into our unified agent task data.

AITZ [57]: This dataset cleans a subset of AITW [38] and uses a proprietary LLM to generate high-quality reasoning and action annotations. Given a step in AITZ, We generate a sample with a reasoning process and one without reasoning to leverage the GUI knowledge entailed in the reasoning content.

AMEX [8]: This dataset expands the General split of the AITW [38] to provide more detailed annotations for each interaction step. The quality of this dataset is high, so we directly reformat their samples and incorporate them into our unified dataset.

AndroidControl [23]: This dataset boasts massive, high-quality interaction trajectories collected over one year by Google Research. As this dataset has not provided the bounding box of target elements, we find the smallest box enclosing target points using the GUI metadata provided.

GUIOdyssey [31]: This dataset provides cross-app interaction trajectories, which can diversify our unified dataset.

WebLINX [32]: This dataset provides dialogue-format human-agent interaction trajectories. We remove all non-action steps and use the action-related steps to generate action prediction samples.

OmniAct-Desktop [21]: This dataset is used in the experiments to assess the transferability of UIPro. As each action plan provided in OmniAct-Desktop is associated with only the starting screenshot, we extract the first action in the plan to generate training and test data.

8. Implementation Details of UIPro

8.1. Training Parameters

The hyper-parameters of training UIPro-SliME and UIPro-Qwen2VL are shown in Tab. 9 and Tab. 10. All experiments are conducted with 8 L20 GPUs, each with 48GB of memory. Pre-training UIPro with the 20.6M GUI understanding data for one epoch took approximately 96 hours on the 8 L20 GPUs; Fine-tuning UIPro with the 380k unified agent task data for the mobile embodiment took approximately 9 hours; Fine-tuning UIPro with the 144.9k unified agent task data for the web embodiment took approximately 3 hours.

Table 9. The training hyper-parameters used for fine-tuning UIPro-SLiME.

Hyper-Parameter	Value
Epoch	1
Global batch size	128
#GPUs	8
Learning rate for all stages	3e-5
weight decay	0.0
ADAM Beta2	0.95
Warm-up ratio	0.03
LR scheduler	Cosine
Model max length	2048
Frozen module	ViT
DeepSpeed	ZeRO-2
Data type	BFloat16

Table 10. The training hyper-parameters used for fine-tuning UIPro-Qwen2VL.

Hyper-Parameter	Value
Epoch	1
Global batch size	128
#GPUs	8
Learning rate for all stages	3e-5
LoRA Rank	128
LoRA Alpha	256
weight decay	0.0
ADAM Beta2	0.95
Warm-up ratio	0.03
LR scheduler	Cosine
Model max length	4096
Frozen module	ViT
DeepSpeed	ZeRO-2
Data type	BFloat16

8.2. Unified Action Spaces

We unify the heterogeneous action definitions from the used data sources and generate three unified action spaces for mobile, web, and desktop scenarios, respectively. Please refer to Tab. 11, Tab. 12, and Tab. 13.

Inconsistency mainly exhibits in swipe, scroll, drag/move, and status (used to signal task completion/impossibility), with substantially different parameter definitions on AITW [38], AndroidControl [23], GUIOdyssey [31], and Mind2Web [14].

8.3. Denoising Procedure

We inspect the used data sources listed in Tab. 16 and Tab. 18, categorize typical noise in the raw GUI data, and develop the following denoising procedure:

1. **Checking the validity of element bounding box coordinates.** An element will be discarded if it is outside of the GUI screenshot or its area is zero.

2. **Removing oversized elements.** As the grounding and referring tasks in our dataset focus on elements that are leaf nodes in the GUI layout hierarchy, we also remove oversized elements that are likely parent nodes. Localizing or referring to these oversized elements probably leads to ambiguity, as the element center often falls in leaf node elements. We remove elements whose area ratios are greater than 0.65. We choose 0.65 as this threshold can achieve an empirically good tradeoff between retaining meaningful elements and removing as many noisy ones as possible.

3. **Removing extremely tiny elements.** According to Google Accessibility Help[3], an element should be large enough for reliable interaction, with a width and height of at least 48dp. Considering that low-density GUI screenshots probably exist in the used GUI data sources, the smallest size of an element is limited to $48 \times (60/160) = 18$pixels. We remove elements whose smaller size is less than this threshold.

4. **Removing blank elements.** It is a common case that the GUI rendering is incomplete due to massive content loading and rendering program bugs. An example is the Mind2Web dataset [14], which contains many incomplete web page HTML scripts. Generating GUI interaction samples from these blank elements is likely to confuse the trained models. To remove these blank elements, we calculate the color deviation using `np.std` for the element region and remove elements whose color deviation is less than 5 (also an empirical value to achieve a good tradeoff between retaining meaningful elements and removing noisy ones as many as possible).

5. **Removing duplicate boxes.** It is also a common case that multiple elements are in the same bounding box. For example, an image button may contain an image element and a button element. We retain only one of the duplicate elements to ensure the dataset's diversity.

6. **Removing invisible textual elements.** Invisible elements, e.g., a hidden menu, will confuse the model. To remove these elements, we utilize `pytesseract` (an efficient OCR toll) to detect the texts for textual elements and remove the elements for which the similarity score between the OCR outputs and their text properties is less than 22.

7. **Denoising for agent task data.** Apart from the noise in GUI grounding data, the used GUI agent task data also contains noise. We list the most occurring noise type in the GUI agent task dataset: 1) AITZ [57]: the action mentioned in the agent's reasoning content does not match the actually taken action. 2) AITW [38]: Apart from the noise recognized by other works [8, 57], we also find that in some cases, one action is repeated multiple times, leading to redundant training samples. 3) AndroidControl [8]: No bounding box associated with the interacted element. 4) Mind2Web [14]:

[3] https://support.google.com/accessibility/android/answer/7101858?hl=en

Action Name	Usage	Definition
click	Click on an element. The target_x and target_y denote the x and y coordinates of the target.	{”action_type”: ”click”, ”target”: ({target_x},{target_y})}
long_press	long-press an element for a duration.	{”action_type”: ”long_Press”, ”target”: ({target_x},{target_y})}
swipe	Simulate a real swipe action to change viewports. The start_x and start_y denote the x and y coordinates of the swipe starting point. The direction has four options: up, down, left, and right. The distance has three options: short, medium, and long.	{”action_type”: ”swipe”, ”start”: ({start_x},{start_y}), ”direction”: ”{direction}”, ”distance”: ”{distance}”}
input_text	Type texts in to an input box.	{”action_type”: ”input_text”, ”text”: ”{text}”}
drag	Press a finger on a target, move the finger to a destination, and finally list the finger.	{”action_type”: ”drag”, ”start”: (x1,y1), ”end”: (x2,y2)}
enter	This action inherits from AndroidControl and Android World. It simulates pressing Enter on a keyboard.	{”action_type”: ”enter”}
navigate_back	Navigate to the previous GUI.	{”action_type”: ”navigate_back”}
navigate_home	Navigate to the home page on the mobile phone.	{”action_type”: ”navigate_home”}
navigate_recent	Open the window showing recently used apps.	{”action_type”: ”navigate_recent”}
wait	Wait for content loading. This is also inherited from AndroidControl and Android World.	{”action_type”: ”wait”}
status	Signal the termination of a task and return an answer if required. The goal_status has two options: ”successful” denoting task completion and ”infeasible” denoting an impossible task. The answer is used to contain the answer generated by the model.	{”action_type”: ”status”, ”goal_status”: ”{goal_status}”, ”answer”: ”{answer}”}

Table 11. The unified action space for mobile scenarios.

Action Name	Usage	Definition
click	Click on an element. The target_x and target_y denote the x and y coordinates of the target.	{”action_type”: ”click”, ”target”: ({target_x},{target_y})}
scroll	Simulate a scroll action to change viewports. The direction has four options: up, down, left, and right. The distance has three options: short, medium, and long.	{”action_type”: ”scroll”, ”direction”: ”{direction}”, ”distance”: ”{distance}”}
input_text	Type texts into an input box.	{”action_type”: ”input_text”, ”text”: ”{text}”}
drag	Press a finger on a target, move the finger to a destination, and finally lift the finger.	{”action_type”: ”drag”, ”start”: (x1,y1), ”end”: (x2,y2)}
move_to	This action simulates moving the mouse and can be used to move the pointer to a location or hover on an element.	{”action_type”: ”move_to”, ”start”: (x1,y1), ”end”: (x2,y2)}
navigate_back	Navigate to the previous webpage.	{”action_type”: ”navigate_back”}
navigate_forward	Undo navigate_back.	{”action_type”: ”navigate_home”}
go_to	Go to a certain URL.	{”action_type”: ”go_to”, ”url”: ”(a certain url)”}
search_google	This action simulates directly typing a search query in the address bar and pressing Enter.	{”action_type”: ”search_google”, ”query”: ”(search query)”}
press_key	This action simulates pressing a key down and then releasing it. Example keys include ’enter’, ’shift’, arrow keys, or function keys.	{”action_type”: ”press_key”, ”key”: ”(key name)”}
hotkey	Press a key combination. The key_comb examples include Ctrl-S or Ctrl-Shift-1 with multiple keys combined with ’-’.	{”action_type”: ”hotkey”, ”key_comb”: ”(key combination)”}
new_tab	Create a new tab in the web browser	{”action_type”: ”new_tab”}
switch_tab	Switch to a tab specified by its index.	{”action_type”: ”switch_tab”, ”tab”: ”(tab index)”}
close_tab	Close the focused tab.	{”action_type”: ”close_tab”}
status	Signal the termination of a task and return an answer if required. The usage of its parameters is the same as mobile.	{”action_type”: ”status”, ”goal_status”: ”{goal_status}”, ”answer”: ”{answer}”}

Table 12. The unified action space for web scenarios.

blank interacted elements.

Our denoising procedure is designed for the data sources used. Nevertheless, one can integrate more rules and adjust the empirical thresholds to extend our procedure to more data sources.

The noise ratios for several data sources are listed in Tab. 17. We find that the noise ratios are unignorable, with 29.0% of WAE [9] elements being noisy.

9. Additional Experiments

9.1. Transfer to Desktop Environments

Transfer UIPro to OmniAct-Desktop Tasks Although UIPro is primarily pre-trained with web and mobile data, we test whether this pre-training can facilitate agent task fine-tuning on out-of-distribution desktop environments, such as Windows and Linux. We fine-tune the pre-trained UIPro with the training split of OmniAct-desktop [21] and evaluate it on the test split. Tab. 14 shows that increasing the pre-training data size consistently improves step SR, with UIPro even surpassing OS-ATLAS[49], which is pre-trained with extensive desktop-domain data and fine-tuned with the same OmniAct data.

9.2. Detailed Performance on ScreenSpot

The grounding accuracy on the three platform splits of ScreenSpot [12] is shown in Tab. 15. The results show that UIPro obtains the highest grounding accuracy and excels at icon grounding.

Action Name	Usage	Definition
click	Click on an element. The target_x and target_y denote the x and y coordinates of the target.	{"action_type": "click", "target": ({target_x},{target_y})}
right_click	Right-click on an element.	{"action_type": "right_click", "target": ({target_x},{target_y})}
double_click	Double-click on an element.	{"action_type": "double_click", "target": ({target_x},{target_y})}
scroll	Simulate a scroll action to change viewports. The direction has four options: up, down, left, and right. The distance has three options: short, medium, and long.	{"action_type": "scroll", "direction": "{direction}", "distance": "{distance}"}
input_text	Type texts in to an input box.	{"action_type": "input_text", "text": "{text}"}
drag	Press a finger on a target, move the finger to a destination, and finnaly list the finger.	{"action_type": "drag", "start": (x1,y1), "end": (x2,y2)}
move_to	This action simulates moving the mouse and can be used to move the pointer to a location or hover on an element.	{"action_type": "move_to", "start": (x1,y1), "end": (x2,y2)}
press_key	This action simulates pressing a key down and then releasing it. Example keys include 'enter', 'shift', arrow keys, or function keys.	{"action_type": "press_key", "key": "(key name)"}
hotkey	Press a key combination. The key_comb examples include Ctrl-S or Ctrl-Shift-1 with multiple keys combined with '-'.	{"action_type": "hotkey", "key_comb": "(key combination)"}
status	Signal the termination of a task and return an answer if required. The usage of its parameters is the same as mobile.	{"action_type": "status", "goal_status": "{goal_status}", "answer": "{answer}"}

Table 13. The unified action space for desktop scenarios.

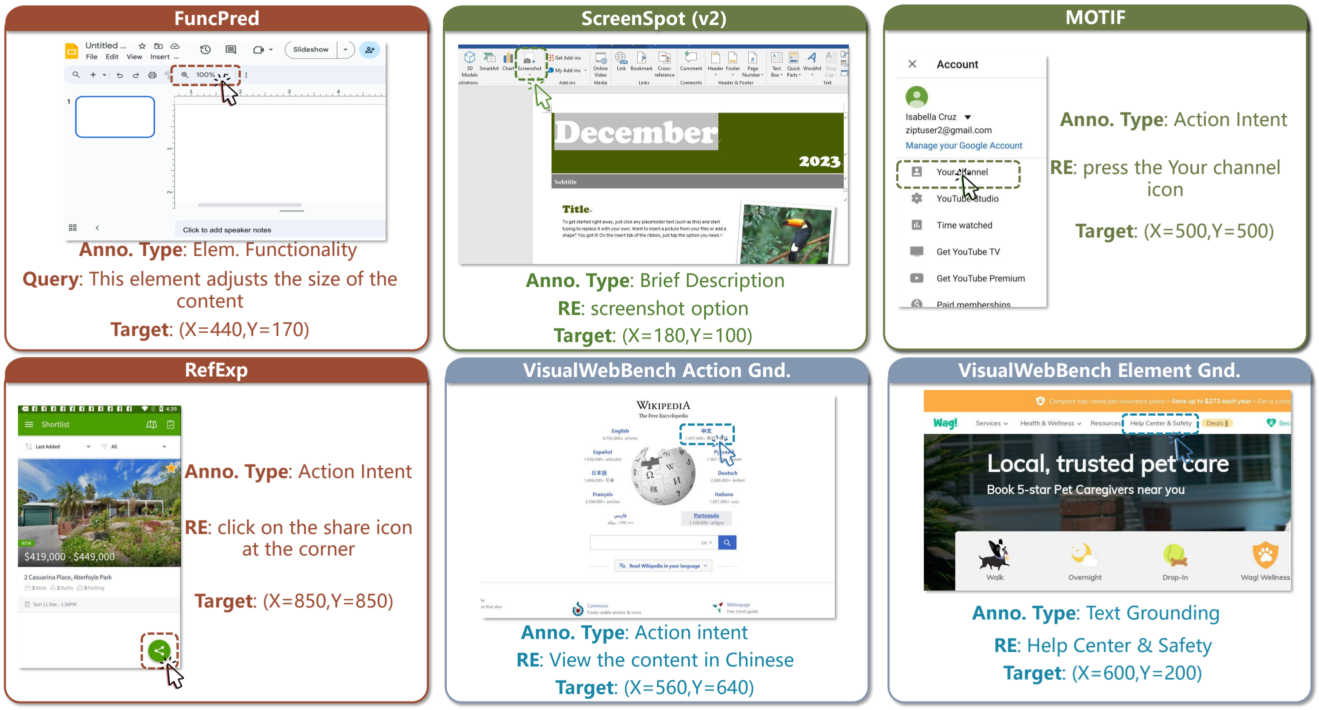
FuncPred
Anno. Type: Elem. Functionality
Query: This element adjusts the size of the content
Target: (X=440,Y=170)
ScreenSpot (v2)
Anno. Type: Brief Description
RE: screenshot option
Target: (X=180,Y=100)
MOTIF
Anno. Type: Action Intent
RE: press the Your channel icon
Target: (X=500,Y=500)
RefExp
Anno. Type: Action Intent
RE: click on the share icon at the corner
Target: (X=850,Y=850)
VisualWebBench Action Gnd.
Anno. Type: Action intent
RE: View the content in Chinese
Target: (X=560,Y=640)
VisualWebBench Element Gnd.
Anno. Type: Text Grounding
RE: Help Center & Safety
Target: (X=600,Y=200)

Figure 7. Examples of the grounding tasks provided by the used GUI element grounding benchmarks in Sec.4.3.

Methods	Size	Step SR
DetAct (GPT-4V) [21]	-	17.0
UGround (GPT-4o) [18]	-	33.4
OS-ATLAS-Base w/ SFT [49]	7B	74.6
UIPro-Qwen2VL	7B	**77.9**
MiniCPM-GUI [10]	3B	11.3
UIPro-SLiME w/o GUI und. PT	3B	15.4
UIPro-SLiME w/ 5.9M GUI und. PT	3B	18.9
UIPro-SLiME w/ 20.6M GUI und. PT	3B	**25.1**

Table 14. **Evaluating UIPro on the desktop software tasks of OmniAct [21]**. Although training samples from desktop domains are scarce in our collected data, the two UIPro models still perform well. Pre-training UIPro-SLiME with more of our GUI understanding data before fine-tuning it on the downstream OmniAct tasks leads to better step SR. OS-ATLAS-Base w/ SFT means OS-ATLAS-Base [49] is finetuned with the OmniAct training split.

Methods	Model Size	Mobile		Desktop		Web		Avg.
		Text	Icon	Text	Icon	Text	Icon	
GPT-4o	-	24.9	24.0	15.5	21.4	12.2	7.3	17.8
Qwen2-VL [46]	7B	78.0	62.9	64.4	50.0	72.2	61.2	66.4
CogAgent [20]	18B	68.5	21.0	73.7	17.9	70.4	33.5	49.8
SeeClick [12]	10B	77.3	52.4	68.6	30.7	59.1	30.6	53.4
UGround [18]	7B	82.8	61.6	84.0	65.1	81.3	71.8	74.8
OS-ATLAS-Base [49]	7B	93.0	72.9	**91.8**	62.9	**90.9**	74.3	82.5
UIPro-Qwen2VL (ours)	7B	**93.1**	**74.7**	89.2	**70.0**	85.7	**74.8**	**82.7**
UIPro-SLiME (ours)	3B	84.3	45.4	70.0	42.7	76.8	29.3	60.8

Table 15. Grounding accuracy on the subsets of the ScreenSpot benchmark [12]. UIPro leads in the overall performance, and achieves the highest accuracy on icon grounding tasks.

GUI Source	#Images	#Tasks	Text		Icon		Other Element		Functionality		Intent Gnd.	Elem. Class.	Widget Listing	QA	Captioning
			Gnd.	Ref.	Gnd.	Ref.	Gnd.	Ref.	Gnd.	Ref.					
Common Crawl[4]†	35.8k	1.3M	134.2k	157.7k	-	-	-	-	314.5k	314.5k	314.5k	-	48.1k	-	-
Android Emulator†	29.6k	629.5k	127.5k	198.1k	-	-	-	-	38.4k	38.4k	189.6k	-	37.7k	-	-
RICO [13]*	34.4k	1.1M	-	-	-	-	939.5k	101.4k	-	-	-	-	-	-	47.2k
MobileViews [17]†	64.6k	1.6M	497.2k	544.3k	-	-	-	-	-	-	495.3k	-	64.0k	-	-
WAE [9]†	372.1k	7.5M	2.1M	2.4M	241.7k	313.5k	-	-	-	-	2.1M	-	345.5k	-	-
AndroidControl [23]†	23.4k	1.1M	186.3k	262.5k	-	-	-	-	23.4k	23.4k	577.0k	-	23.4k	-	-
WebUI [48]†	162.3k	485.2k	-	-	59.6k	113.7k	-	-	-	-	110.7k	40.9k	160.3k	-	-
MultiUI [30]*	1.4M	5.2M		371.5k	-	-	684.7k	-	-	-	1.2M	-	-	2.9M	-
GUIEnv [10]*	70.5k	680.7k	328.3k	352.3k	-	-	-	-	-	-	-	-	-	-	-
SeeClick-Web [12]*	270.8k	1.1M	541.5k	541.5k	-	-	-	-	-	-	-	-	-	-	-
OmniAct [21]*	176	19.1k	-	-	-	-	-	-	-	-	19.1k	-	-	-	-
MOTIF [7]*	55	7.9k	-	-	-	-	-	-	-	-	7.9k	-	-	-	-
TOTAL	2.5M	20.6M	3.9M	4.8M	301.3k	427.3k	1.6M	101.4k	376.3k	376.3k	5.0M	40.9k	679.1k	2.9M	47.2k

Table 16. **Data sources and statistics of UI-Pro GUI understanding dataset.** Approximately two thirds of the tasks are newly generate by the authors while one third is cleaned and included from existing dataset. †means that the authors generate fine-tuning samples from unlabeled raw GUI data. * means that the authors clean the samples provided by the original datasets.

GUI Source	%Invalid Elem
Common Crawl	2.5
Android Emulator	0.4
MobileViews [17]	24.5
WAE [9]	29.0
AndroidControl [23]	11.5
WebUI [48]	14.4
MultiUI [30]	3.0
SeeClick-Web [12]	0.2

Table 17. The percentage of invalid elements detected for the used data source by the proposed denoising procedure.

Scenario	Data Source	#Used Steps	Total
Mobile	AITW [38]	37.6k	380.0k
	AITZ [57]	25.9k	
	AMEX [8]	38.7k	
	AndroidControl [23]	124.0k	
	GUIAct-smartphone [10]	64.3k	
	GUIOdyssey [31]	107.7k	
Web	Mind2Web [14]	7.7k	144.9k
	GUIAct-web [10]	109.3k	
	WebLINX [32]	22.0k	

Table 18. The number of samples included in the merged GUI agen task fine-tuning data from each data source.