A. More Analysis of RegionFocus

Figure 9 shows the difference in number of steps between the combined BaseModel + RegionFocus approach and the BaseModel alone over 400 WebVoyager trajectories, where the BaseModel is UI-TARS-72B. Only the actual browser-interactive steps are counted, excluding RegionFocus overhead. As shown, BaseModel + RegionFocus generally yields 19.74 more steps on average, resulting in overall 34.3% higher success rate.

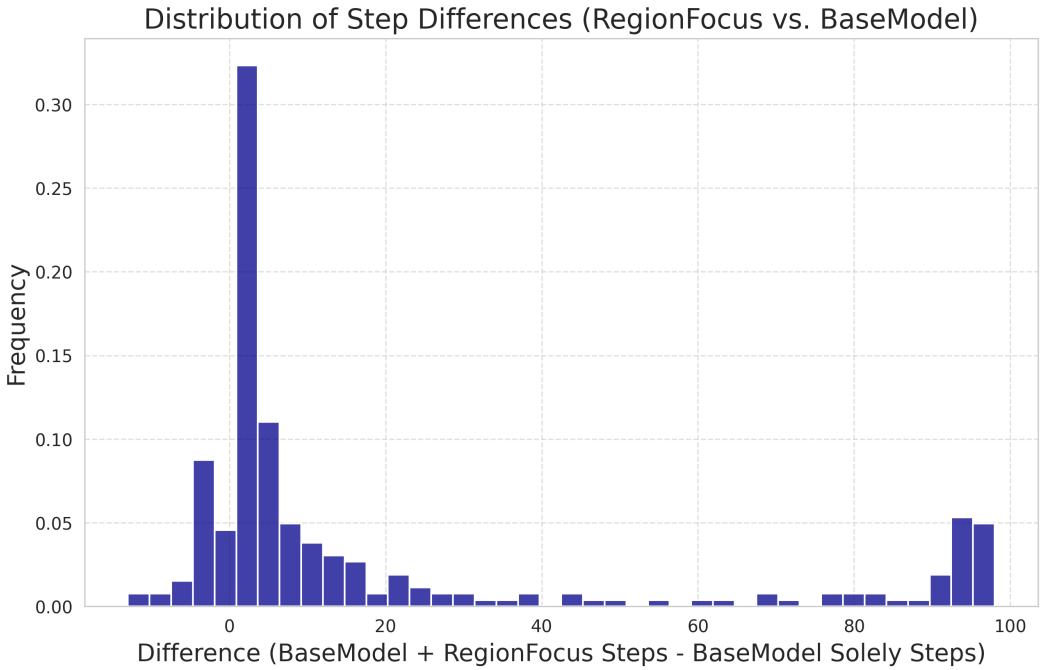
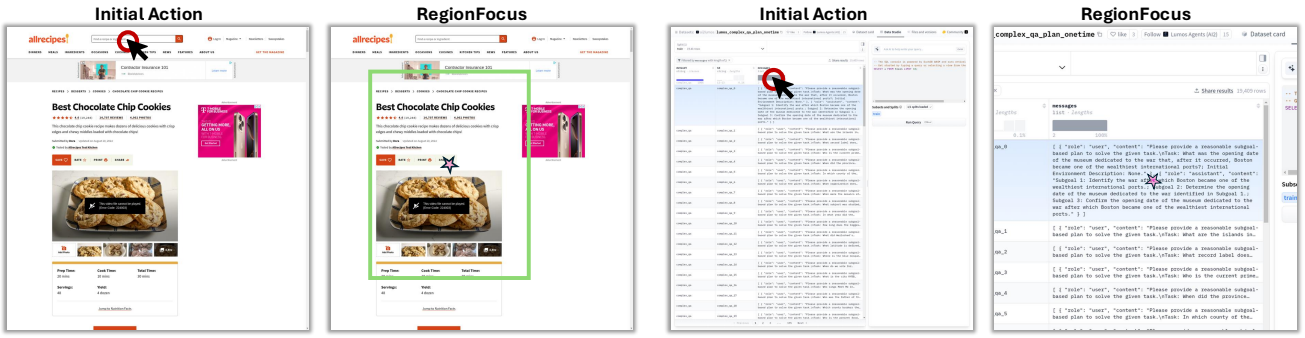


Figure 9. **Histogram of Step Differences on WebVoyager**: Comparing UI-TARS-72B with and without RegionFocus across 400+ trajectories.

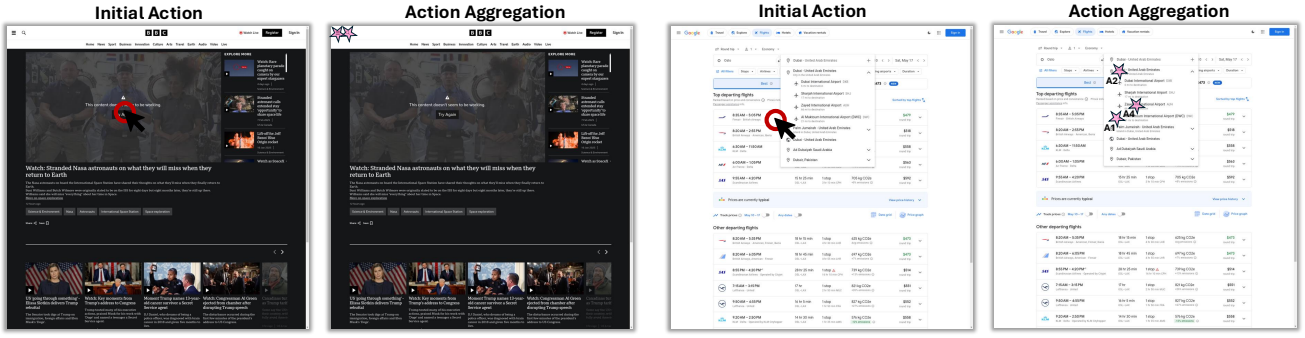
We present several qualitative examples of WebVoyager's performance in Figures 10, 11. In Figure 10 left, the agent initially fails by clicking the 'ingredients' button, which appears in the search bar despite being on the correct page. By highlighting the relevant region with a green bounding box, RegionFocus naturally filters out background noise and draws attention to the primary content. In Figure 10 right, RegionFocus zooms in on the sub-region of interest, enlarging key content and making it easier for the agent to locate the target content. Figure 11 left shows a case where the agent initially clicks an unrelated element. Our pipeline then corrects this mistake by proposing two closely positioned buttons. The image-as-map mechanism allows the agent to distinguish between these nearly identical elements, even though their coordinates differ only slightly. Finally, Figure 11 right illustrates a scenario where the agent mistakenly clicks on an empty area close to the desired element. Once again, RegionFocus highlights the correct button, helping the agent choose it accurately.



Task: Find a Popular recipe for a chocolate chip cookie and list the ingredients and preparation steps.

Task: Check the Dataset Viewer for ai2lumos/lumos_complex_qa_plan_onetime on Hugging face. what is the content corresponding to user in the first message?

Figure 10. **Qualitative Results - RegionFocus.** In these two examples, we illustrate how RegionFocus reduces background noise by emphasizing salient regions of an image. The mouse pointer indicates the agent's initial action prediction, which is suboptimal in both cases. **Left pair of images**: The green window in the second image marks the zoomed-in region. By focusing on this region, we naturally cut out the distracting portion of the first image. **Right pair of images**: The second image is zoomed in, significantly reducing distracting details. This allows the agent to focus on the region with more relevant information.



Task: Search the latest article about space exploration on BBC News and summarize its key points.

Task: Compare the prices and flight durations for economy class flights from Oslo to Dubai, departing on March 8, 2025, and show options with no more than two layovers.

Figure 11. **Qualitative results - image-as-map.** These examples demonstrate how action aggregation, enhanced by the proposed image-as-map, helps distinguish subtle coordinate differences between target elements. The mouse pointer indicates the agent's initial predictions, which were incorrect in both cases. Each star-like landmark is generated during the RegionFocus process before action aggregation. **Left pair of images**: The two top-left landmarks correspond to the home and search buttons. **Right pair of images**: the landmarks correspond to different options in a dropdown menu.

B. More experimental details

In this section, we provide more details about our experimental settings. In our main paper, we examined both the UI-TARS-7B-DPO and UI-TARS-72B-DPO models, as well as Qwen2.5-VL-7B-Instruct and Qwen2.5-VL-72B-Instruct. For WebVoyager, we used a screen resolution of 1440×1440 pixels for the UI-TARS models and 2240×1260 for the Qwen models. For both ScreenSpot-Pro and WebVoyager, our predefined bounding boxes were defined as ratios of the input image size, specifically $[0.5, 0.5]$, $[0.3, 0.3]$, $[0.4, 0.8]$, and $[0.8, 0.4]$. Some of the prompts we used are listed below.

Prompt for Region Focus

```
You are a GUI agent. You are given a task, a current web screenshot, and a history of your previous focused
     points on the same page (indicated by pink stars in the screenshot). Your job is to output the most
     relevant point in the screenshot corresponding to the objective. You must avoid the pink-starred
     coordinates and choose a valid clickable area.

## Other Information
OBJECTIVE: {objective}
URL: {url}

## Output Format
```
(x1, y1)
```
where x1, y1 are the coordinates of the target element, and must differ from any pink-starred coordinates.

## Note
- Ensure the chosen coordinate is a valid clickable area not visibly covered by pink stars in the screenshot.
```

Prompt for Action Prediction – UI-TARS

```
You are a GUI agent. You are given a task and your action history, with screenshots.
You need to perform the next action to complete the task.

## Other Information
OBJECTIVE: {objective}
URL: {url}

## Output Format
```\nThought: ...
Action: ...\n```

Action Space
click(start_box='<|box_start|>(x1,y1)<|box_end|>')
left_double(start_box='<|box_start|>(x1,y1)<|box_end|>')
right_single(start_box='<|box_start|>(x1,y1)<|box_end|>')
drag(start_box='<|box_start|>(x1,y1)<|box_end|>', end_box='<|box_start|>(x3,y3)<|box_end|>')
hotkey(key='')
type(content='') #If you want to submit your input, use "\" at the end of `content`.
scroll(start_box='<|box_start|>(x1,y1)<|box_end|>', direction='down or up or right or left')
wait() #Sleep for 5s and take a screenshot to check for any changes.
finished()
call_user() # Submit the task and call the user when the task is unsolvable, or when you need the user's help.

Note
- Use English in `Thought` part.
- Summarize your next action (with its target element) in one sentence in `Thought` part.
```
```

Prompt for Action Prediction – QWen2.5-VL (part 1)

```
You are a helpful assistant.

# Tools

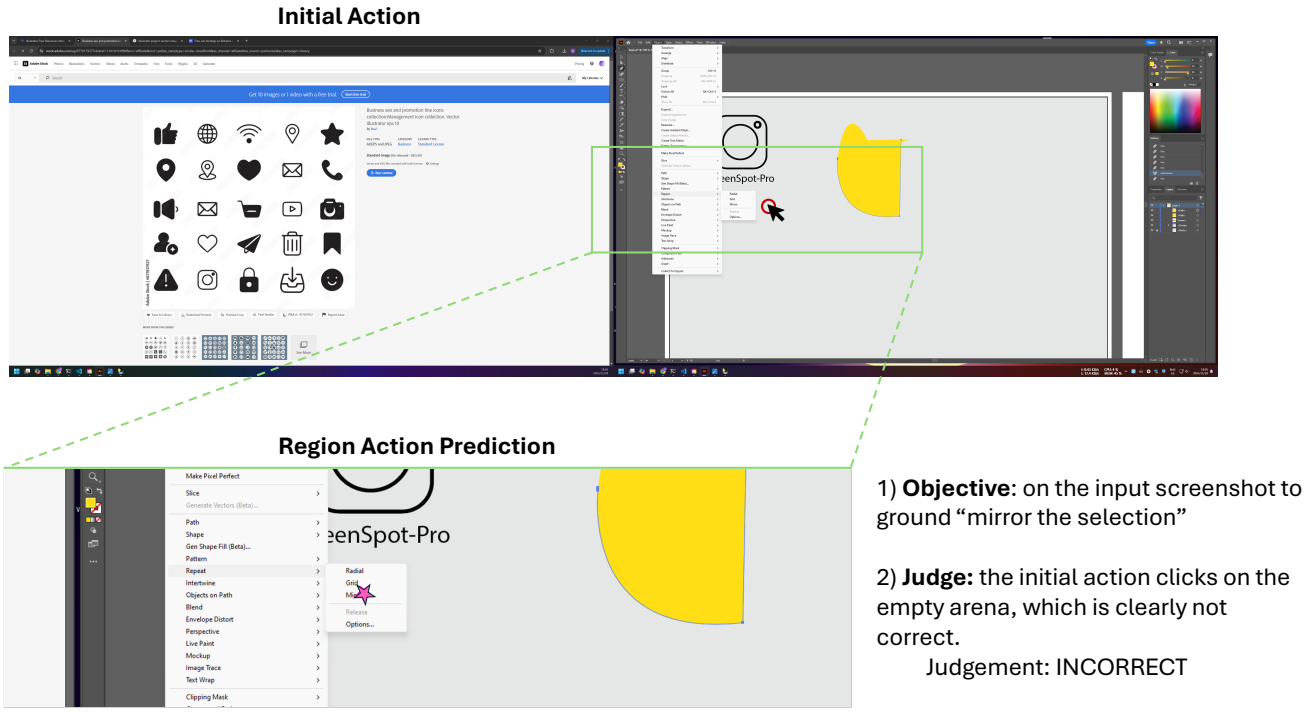
You may call one or more functions to assist with the user query.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{
   "type": "function",
   "function": {
      "name": "computer_use",
      "description": """Use a mouse and keyboard to interact with a computer, and take screenshots.
         * This is an interface to a desktop GUI. You do not have access to a terminal or applications menu.
             You must click on desktop icons to start applications.
         * Some applications may take time to start or process actions, so you may need to wait and take
             successive screenshots to see the results of your actions. E.g. if you click on Firefox and a
             window doesn't open, try wait and taking another screenshot.
         * The screen's resolution is {self.display_width_px}x{self.display_height_px}.
         * Whenever you intend to move the cursor to click on an element like an icon, you should consult a
             screenshot to determine the coordinates of the element before moving the cursor.
         * If you tried clicking on a program or link but it failed to load, even after waiting, try adjusting
              your cursor position so that the tip of the cursor visually falls on the element that you want
             to click.
         * Make sure to click any buttons, links, icons, etc with the cursor tip in the center of the element.
              Don't click boxes on their edges unless asked."""
      "parameters": {
         "properties": {
            "action": {
               "description": """
                  The action to perform. The available actions are:
                  * `key`: Performs key down presses on the arguments passed in order, then performs key
                      releases in reverse order.
                  * `type`: Type a string of text on the keyboard.
                  * `mouse_move`: Move the cursor to a specified (x, y) pixel coordinate on the screen.
                  * `left_click`: Click the left mouse button.
                  * `left_click_drag`: Click and drag the cursor to a specified (x, y) pixel coordinate on the
                      screen.
                  * `right_click`: Click the right mouse button.
                  * `middle_click`: Click the middle mouse button.
                  * `double_click`: Double-click the left mouse button.
                  * `scroll`: Performs a scroll of the mouse scroll wheel.
                  * `wait`: Wait specified seconds for the change to happen.
                  * `terminate`: Terminate the current task and report its completion status.
                  """,
               "enum": [
                  "key",
                  "type",
                  "mouse_move",
                  "left_click",
                  "left_click_drag",
                  "right_click",
                  "middle_click",
                  "double_click",
                  "scroll",
                  "wait",
                  "terminate",
               ],
               "type": "string",
            },
            "keys": {
               "description": "Required only by `action=key`.",
               "type": "array",
            },
            "text": {
               "description": "Required only by `action=type`.",
               "type": "string",
            },
            "coordinate": {
               "description": "(x, y): The x (pixels from the left edge) and y (pixels from the top edge)
                    coordinates to move the mouse to. Required only by `action=mouse_move` and `action=
                    left_click_drag`.",
               "type": "array",
            },
```

Prompt for Action Prediction – QWen2.5-VL (part 2)

```
            "pixels": {
               "description": "The amount of scrolling to perform. Positive values scroll up, negative values
                    scroll down. Required only by `action=scroll`.",
               "type": "number",
            },
            "time": {
               "description": "The seconds to wait. Required only by `action=wait`.",
               "type": "number",
            },
            "status": {
               "description": "The status of the task. Required only by `action=terminate`.",
               "type": "string",
               "enum": ["success", "failure"],
            },
         },
         "required": ["action"],
         "type": "object",
      }
   }
}
For each function call, return a json object with function name and arguments within <tool_call></tool_call>
     XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>
```

C. More qualitative results



1) **Objective**: on the input screenshot to ground "mirror the selection"

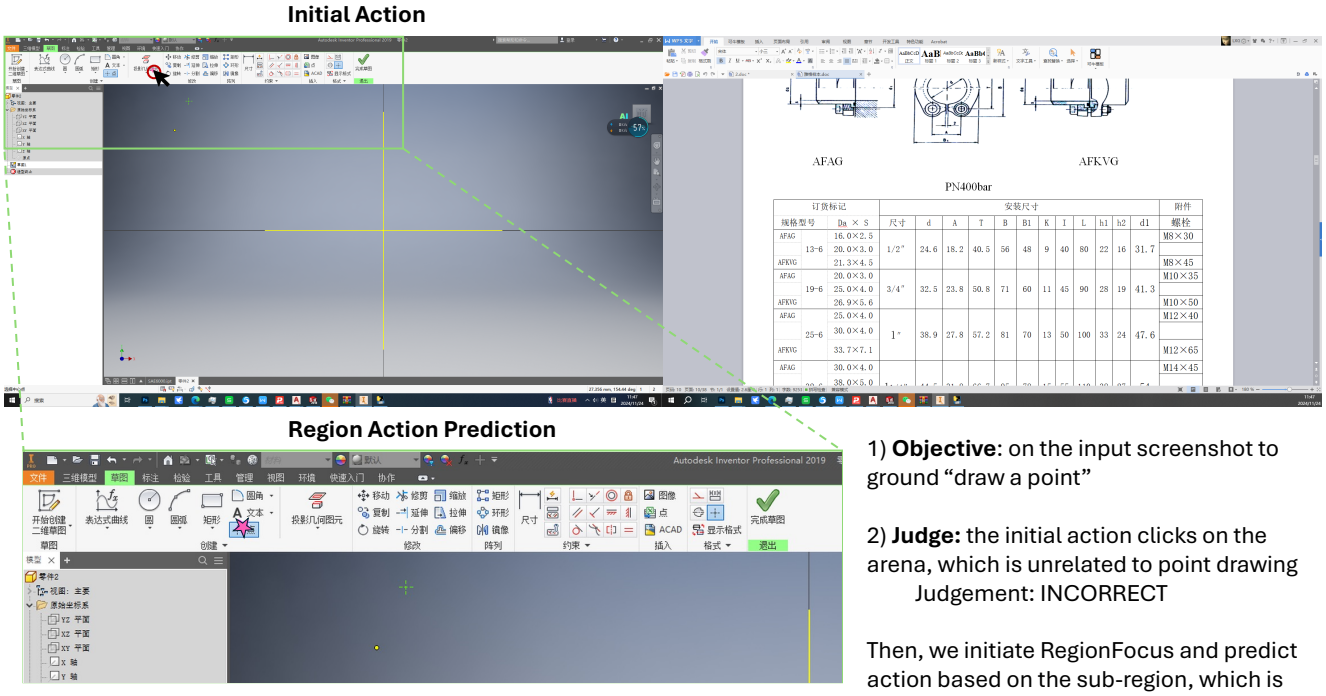
2) **Judge:** the initial action clicks on the empty arena, which is clearly not correct.

Judgement: INCORRECT

Then, we initiate RegionFocus by first proposing a focal point. We use this focal point as a center to generate several potential regions.

One of these regions is shown on the left, and the agent correctly identifies target element position, which is "Mirror".

Figure 12. **Qualitative results - Screenspot-Pro.** In one example from our evaluation, the system successfully rejects the initial action and proposes a correct grounding point based on the zoomed-in view.



1) **Objective**: on the input screenshot to ground “draw a point”

2) **Judge:** the initial action clicks on the arena, which is unrelated to point drawing
Judgement: INCORRECT

Then, we initiate RegionFocus and predict action based on the sub-region, which is the correct output.

Figure 13. **Qualitative results - Screenspot-Pro.** In one example from our evaluation, the system successfully rejects the initial action and proposes a correct grounding point based on the zoomed-in view.

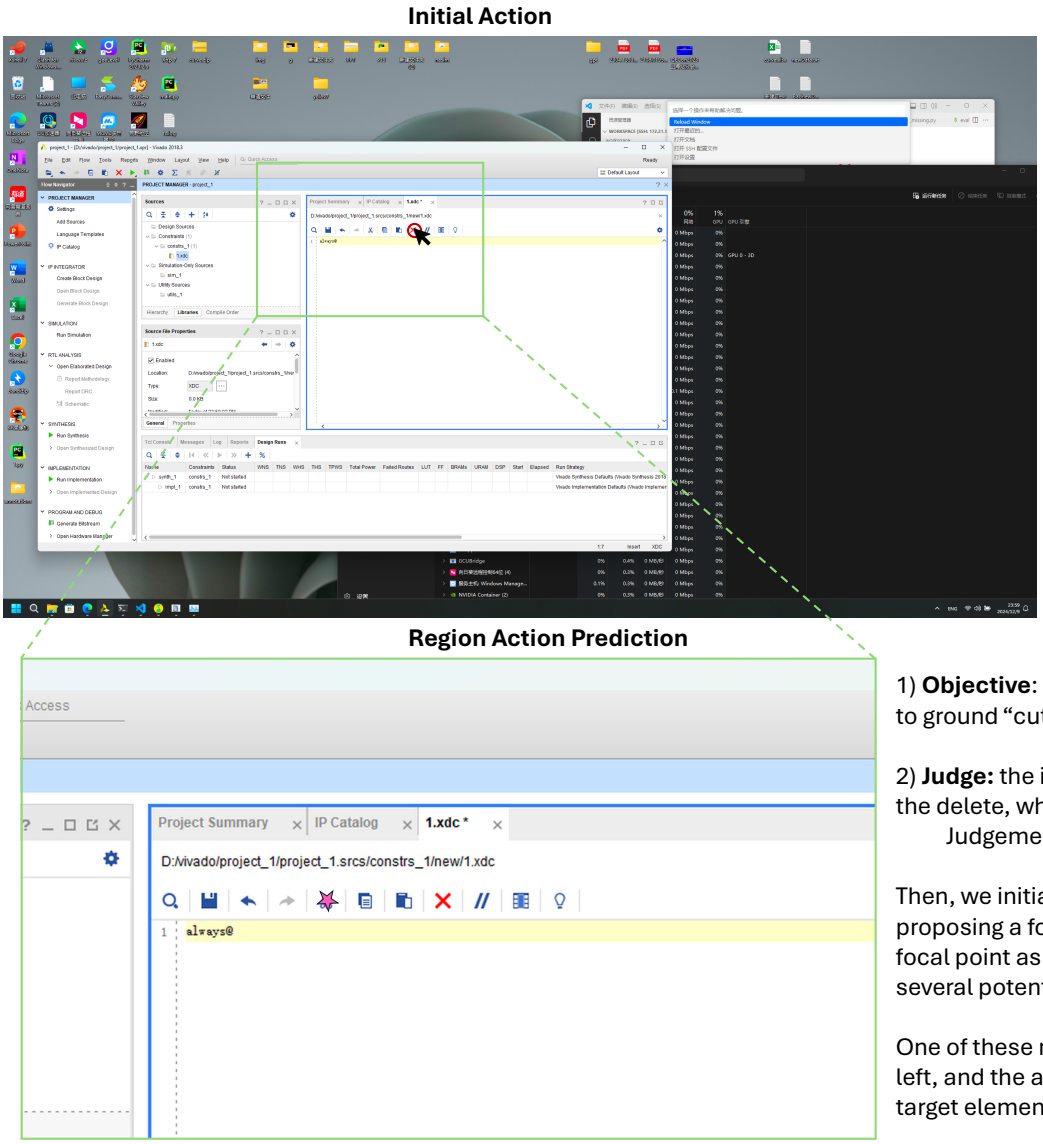


Figure 14. **Qualitative results - Screenspot-Pro.** In one example from our evaluation, the system successfully rejects the initial action and proposes a correct grounding point based on the zoomed-in view.