ShowUI-π: Flow-based Generative Models as GUI Dexterous Hands

Supplementary Material

A. More Results

A.1. Grounding Performance of ShowUI-π

To evaluate the grounding performance of ShowUI-π as a VLA unifying both discrete click and continuous drag actions, we evaluate ShowUI-π on one of the most challenging grounding benchmarks, ScreenSpot-Pro. During the grounding evaluation, we use the first coordinate in the generated action chunk as the model prediction for grounding. As demonstrated in Tab. 1, ShowUI-π achieves performance comparable to much larger baseline models. Due to the small parameter size (450M), the vision capacity of ShowUI-π is weaker than other models, thus restricting the grounding performance.

A.2. Drag Performance of ShowUI-π on public benchmark

To evaluate the drag performance of ShowUI-π on public dataset, we evaluate ShowUI-π on the `Drag` subset of VideoGUI-Action benchmark where drag performance is specifically reported. As the benchmark is image-based, only one screenshot is provided as the visual input. Therefore, we let ShowUI-π generate the entire action chunk at once and use it as the model prediction for dragging, without observations on-the-fly or continuous actions. This restricts ShowUI-π's performance, as its drag capability is trained on the continuous observations from ScreenDrag only, without the drag data from public datasets, thus leading to a gap between training and public benchmark evaluation. However, ShowUI-π still achieves performance comparable to much larger baseline models, as demonstrated in Tab. 2.

A.3. Effects of Co-training with Drag Data and Grounding Data

We also evaluate the effects of training ShowUI-π on the combination of `Drag` data and `Grounding` data. As shown in Fig. 4 and Fig. 3, training on both types of actions leads to the highest performance on both ScreenDrag online evaluation and Screenshot-Pro benchmarks. Interestingly, after training on ScreenDrag drag data, the performance on the `Creative` and `Office` categories in ScreenSpot-Pro are boosted, as there is a large amount of data in ScreenDrag within the corresponding domains, *e.g.,*, PowerPoint and Premiere Pro. Moreover, training on grounding data can help the model learn a better representation, thus bringing the gain. As the grounding data contains only `Click` data, the model trained on grounding data solely is unable to generate valid drag trajectories, thus failing the drag tasks.

B. Setup

B.1. Training Details

We utilize 4 H200 GPUs for training. The batch size per GPU is set to 64, without gradient accumulation. We use `bfloat16` precision for training. The vision encoder and language model part are initialized with the weights from SmolVLM, and other components, *e.g.,*, action state embedding, action expert, *etc*., are randomly initialized. The model is trained end-to-end thanks to its small parameter size. To enhance efficiency, we resize the visual observation to $(1024, 576)$ from the common $(1920, 1080)$, reducing the vision encoder overhead, while maintaining the $16:9$ scale. We leverage DeepSpeed Zero-2 as the training framework. The learning rate is configured to 1e-4.

B.2. Training Data

We use a smaller training corpus for ShowUI-π compared to other models. Specifically, in addition to the training data from ScreenDrag, we use the desktop `Click` subset of GUIAct, WaveUI, UGround, and ShowUI-Desktop. We did not use any `non-Click` data from any public dataset, nor did we use any mobile data for training.

C. Dataset Construction

C.1. How we collect raw data

We construct raw demonstration data across five domains: PowerPoint, OS Desktop and File Manager, Captcha, Handwriting, as well as Adobe Premiere Pro. Across all domains, data are collected on Windows machines where we record high-frequency mouse events and screen video recordings, using our ScreenDrag data pipeline. To obtain the UI metadata, the DOM is used for Captcha on webpages and the

Model	Development			Creative			CAD			Scientific			Office			OS			Avg		
	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg
OSAtlas-7B	33.1	1.4	17.7	28.8	2.8	17.9	12.2	4.7	10.3	37.5	7.3	24.4	33.9	5.7	27.4	27.1	4.5	16.8	28.1	4.0	18.9
UGround (7B)	26.6	2.1	14.7	27.3	2.8	17.0	14.2	1.6	11.1	31.9	2.7	19.3	31.6	11.3	27.0	17.8	0.0	9.7	25.0	2.8	16.5
AriaUI (MOE, 3.9B active)	16.2	0.0	8.4	23.7	2.1	14.7	7.6	1.6	6.1	27.1	6.4	18.1	20.3	1.9	16.1	4.7	0.0	2.6	17.1	2.0	11.3
CogAgent (18B)	14.9	0.7	8.0	9.6	0.0	5.6	7.1	3.1	6.1	22.2	1.8	13.4	13.0	0.0	10.0	5.6	0.0	3.1	12.0	0.8	7.7
ShowUI (2B)	16.9	1.4	9.4	9.1	0.0	5.3	2.5	0.0	1.9	13.2	7.3	10.6	15.3	7.5	13.5	10.3	2.2	6.6	10.8	2.6	7.7
ShowUI-π (450M)	9.7	5.5	7.7	10.1	2.8	7.0	1.0	0.0	0.8	9.7	6.4	8.3	10.2	1.9	8.3	3.7	3.4	3.6	7.5	3.8	6.1
OSAtlas-4B	7.1	0.0	3.7	3.0	1.4	2.3	2.0	0.0	1.5	9.0	5.5	7.5	5.1	3.8	4.8	5.6	0.0	3.1	5.0	1.7	3.7
MiniCPM-V (7B)	7.1	0.0	3.7	2.0	0.0	1.2	4.1	1.6	3.4	8.3	0.0	4.7	2.8	3.8	3.0	3.7	1.1	2.6	4.5	0.7	3.0
Qwen2-VL-7B	2.6	0.0	1.3	1.5	0.0	0.9	0.5	0.0	0.4	6.3	0.0	3.5	3.4	1.9	3.0	0.9	0.0	0.5	2.5	0.2	1.6
SeeClick (7B)	0.6	0.0	0.3	1.0	0.0	0.6	2.5	0.0	1.9	3.5	0.0	2.0	1.1	0.0	0.9	2.8	0.0	1.5	1.8	0.0	1.1
GPT-4o	1.3	0.0	0.7	1.0	0.0	0.6	2.0	0.0	1.5	2.1	0.0	1.2	1.1	0.0	0.9	0.0	0.0	0.0	1.3	0.0	0.8
QwenVL-7B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.1

Table 1. **Performance breakdown of various models across application categories on ScreenSpot-Pro.**

Model	Drag	
	Dist. ↓	Recall ↑
CogAgent (18B)	44.7	0.0
Qwen-VL-Max (72B)	42.0	0.3
Gemini-Pro-Vision	40.8	0.0
Claude-3-Opus	30.6	1.7
GPT-4-Turbo	31.3	1.4
ShowUI-π (450M)	41.6	1.7
GPT-4o	21.9	2.5

Table 2. **Drag performance of ShowUI-π and baseline models.** Following VideoGUI's setting, two metrics are used in the evaluation. The distance (Dist) is normalized and then multiplied by 100, reflecting the average offset between the predicted and ground-truth drag endpoints. The recall (Rec) is the percentage of drags whose predicted start and end points both fall within a 100-pixel threshold of the corresponding ground-truth endpoints. Note that VideoGUI-Action is image-based and only provides the initial screenshot.

UIA framework is used for other domains, so that each trajectory can be paired with precise element locations and attributes.

PowerPoint. For the PowerPoint domain, we start from the official Microsoft template gallery. We crawl and download a diverse set of slide templates spanning different layouts, color schemes, and typography. For each template, we automatically parse the slide metadata, *e.g.,*, textboxes, images and shapes, using UIA, and then design manipulation tasks such as rotating and different types of resizing. The position of rotation handle is calculated using heuristics, as its position is not in the UIA metadata.

Captcha. For Captcha tasks, we build the automated data collection pipeline on an open-source library Go-Captcha. We configure the pipeline to generate interactive Captchas such as sliders and puzzle pieces embedded in a webpage, and the Captcha will be refreshed and regenerated after the previous one is solved. We modify the Captcha library codebase so that the task metadata, *e.g.,*, puzzle piece position, target position, *etc*., and the task status, *e.g.,*, success or failure are accessible at real-time. Therefore, we can filter the successfully solved tasks and collect their trajectories.

Handwriting. For the Handwriting domain, we build upon an open-source handwriting synthesis library. We sample diverse names, short phrases from a Qwen2.5-72B endpoint deployed using VLLM, then the handwriting trajectory from the sampled text will be generated using the handwriting synthesis library with varied stroke styles. Afterwards, the trajectory will be written on the canvas through Win32 mouse interface for fine-grained control.

Adobe Premiere Pro. In the Premiere Pro domain, we capture human demonstration trajectories in creative workflows instead of automation, due to the limitation that Premiere Pro UI metadata is not fully accessible. We recruit two student annotators experienced with video editing to design tasks that reflect common real-world operations, such as trimming clips, adjusting layers on the timeline, arranging clips, and applying effects. The expert demonstrations are recorded using our recorder that can capture high-frequency mouse events with low latency, aligned with the video recording timestamps.

OS Desktop and File Manager. For OS Desktop and File Manager tasks, we build an automated pipeline that creates different types of files and folders on the desktop or file manager windows, then performs drags and records. To increase the task difficulty, we modify the Windows registry so that files and folders created can be placed anywhere on the desktop without automatic arrangement, and each time multiple files and folders with different names will be generated. UIA is used to obtain bounding boxes for desktop icons, folders, and window controls.

Data Recipe	Development			Creative			CAD			Scientific			Office			OS			Avg		
	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg	Text	Icon	Avg
Drag	1.3	0.0	0.7	0.5	2.8	1.5	1.0	1.6	1.1	0.7	0.0	0.4	0.6	3.8	1.3	2.8	0.0	1.5	1.0	1.2	1.1
Grounding	8.4	4.8	6.7	8.1	2.1	5.6	0.5	0.0	0.4	12.5	5.5	9.4	8.5	1.9	7.0	7.5	7.9	7.7	7.3	4.0	6.0
Drag + Grounding	9.7	5.5	7.7	10.1	2.8	7.0	1.0	0.0	0.8	9.7	6.4	8.3	10.2	1.9	8.3	3.7	3.4	3.6	7.5	3.8	6.1

Table 3. **Performance breakdown of models trained using different data recipe across application categories on ScreenSpot-Pro.**

Data Recipe	**Online Evaluation.** Avg. Success Rate(↑).					
	OS	PowerPoint	Premiere	Captcha	Handwriting	Overall
Grounding	0.00	0.00	0.00	0.00	0.00	0.00
Drag	20.79	10.34	4.62	48.15	25.74	21.92
Drag + Grounding	**13.11**	**22.93**	**8.64**	**55.91**	**34.32**	**26.98**

Table 4. **Performance breakdown of models trained using different data recipe across different domains on ScreenDrag online evaluation.**

Data Recording. Across all domains, we use OBS for the screen recording, as it is well optimized and has less latency compared to other screen capture methods, *e.g.,*, FFmpeg.

Data Generation Codebase. To contribute to the community, we will open-source the data generation and recording codebase. Moreover, some of the data crawl sources, *e.g.,*, the Microsoft PowerPoint Template Gallery, have changed and make the template collection more difficult, *i.e.*, only a limited number of templates will be shown without search queries, therefore, we will also provide our collected raw data.

C.2. Data visualization

The visualization of some task trajectories are shown in Tab. 5

C.3. Data-driven Closed-loop Online Evaluation

As mentioned in the main paper, a **data-driven** approach is designed to enable closed-loop rollouts in online evaluation. For each drag task, we store the video recording, task specification, and dense drag trajectory, providing extensive possible GUI states encountered during dragging. During rollouts, the model's predicted action is matched to the nearest recorded state if it falls within a tolerance ϵ (*e.g.,* within 20 pixels of a ground-truth waypoint), upon which the corresponding next observation is retrieved. As shown in Fig. 1, when the model prediction midway can be mapped to a trajectory point, its corresponding UI state will be retrieved from the extensive pre-collected UI states as the next observation, thus the model can perform continuous actions with observations on-the-fly. This data-driven approach largely reduces the complexity of setting up OS and software applications, enhancing reproducibility, while still enabling a closed-loop rollout manner.

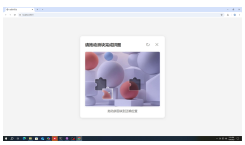
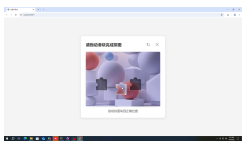
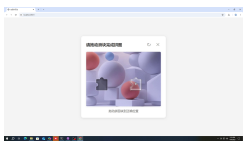
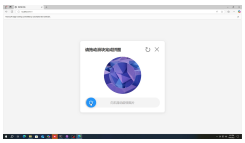
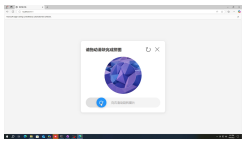
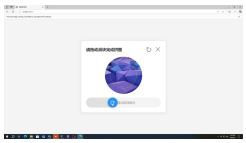
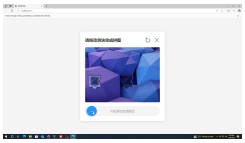
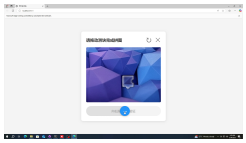
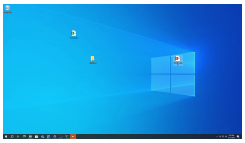
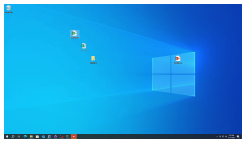
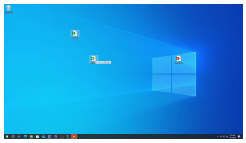
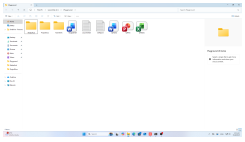
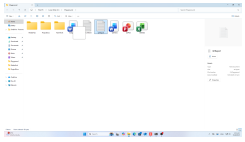
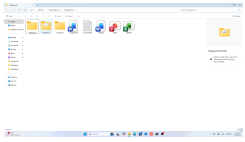
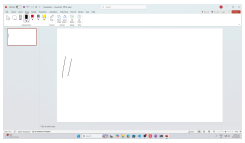
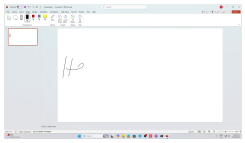
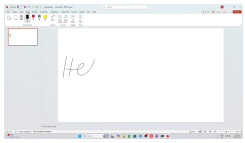
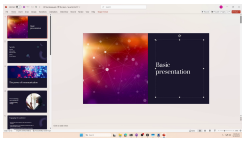
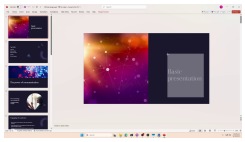
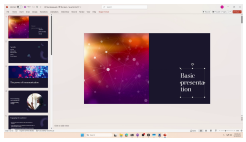
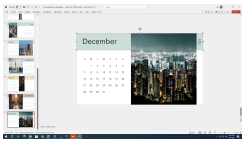
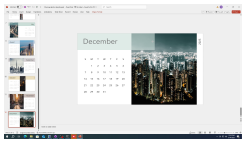
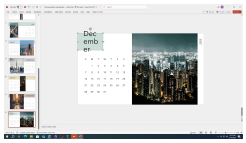
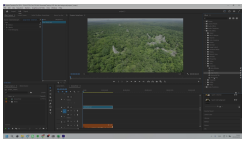
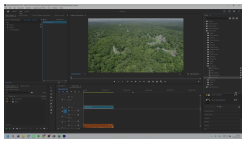
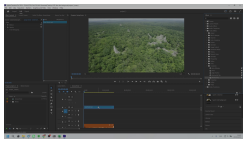
D. Failure Cases of Baseline Models

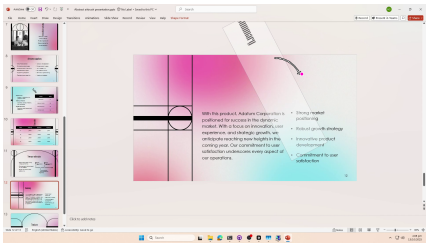
To better illustrate the behavior of baseline models in drag tasks, we analyze eight failure cases observed during evaluation. These qualitative examples highlight problems encountered by existing GUI agents based on language-modeling.

(i) Know-How but does not have the tool. As shown in Fig. 2, the PowerPoint rotation task expects the model to rotate elements using the white rotation handle on the target elements with an arc trajectory. Interestingly, both OpenCUA-7B and Operator frequently produce the correct first step, successfully locating the handle. However, they are only equipped and trained with linear drag tools, *e.g.,*, `Drag((`x_1`,`y_1`),(`x_2`,`y_2`))` and `Linear_Drag((`x_1`,`y_1`),(`x_2`,`y_2`))`, *etc*. Therefore, they are unable to perform rotation even if they know how to rotate, as they do not have the corresponding tool. This highlights a major limitation of GUI agents based on language-modeling: Beyond drag in an arc trajectory, there are various types of free-form drags in real-world scenarios, which are extremely difficult to be fully covered by defining drag tools.

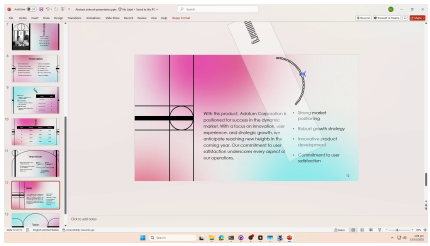
(ii) Gap between discrete tool use and continuous drag. As shown in Fig. 3, on desktop drag tasks, both the Seed-1.6-Vision baseline and Qwen3-VL-32B frequently produce correct high-level plans such as "drag `BudgetNotes.txt` into `TeamDocs`", successfully locate the icon position, and the resulting trajectory starts in the right direction. However, the cursor often stops midway or lands noticeably short of the folder, leaving a large distance to the target. This case illustrates a gap between discrete tool use and continuous drag: language-modeling baselines are designed to predict one discrete tool call each time, *e.g.,*, drag files using one `Drag()` call, instead of

Table 5. **Examples of task trajectories from five domains.** Three frames from the episode are shown for each task.

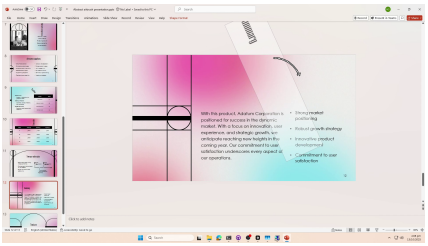
Domain	Task	Initial	Intermediate	Final
Captcha	Solve the Drag-and-Drop Captcha			
	Solve the Rotate Captcha			
	Solve the Slider Captcha			
Desktop	Drag Analysis.xlsx to MyProject			
	Drag Q1Report to projectDocs			
Handwriting	Write “Hello World” on canvas			
PowerPoint	Rotate the Lion counter-clockwise by 45 degrees			
	Resize the title Basic Presentation diagonally by 0.5 from top-left corner			
	Resize width of the textbox December to 0.2 from its right			
Premiere	Apply Vertical Flip effect to Great Forest clip			



(a) The model predicts a coordinate close to the dense trajectory points.



(b) The prediction is mapped to its closest trajectory point.



(c) The model receives the next observation at the mapped trajectory point.

Figure 1. **The visualization of the data-driven closed-loop online evaluation process.** This approach enables models to perform continuous actions with observations on-the-fly, without the complexity to set up OS and software applications, enhancing reproducibility.

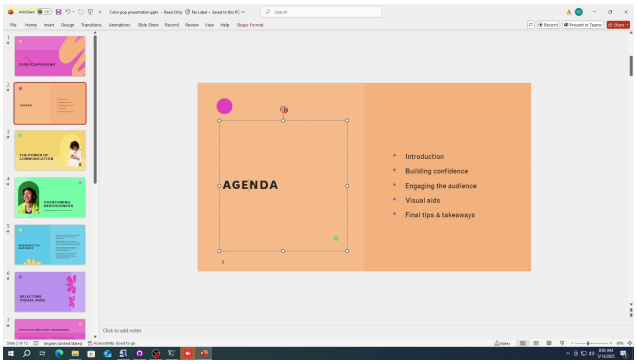


Figure 2. **Know-How but does not have the tool.** The baseline formulates a correct plan to rotate the textbox by dragging the handle above the textbox "AGENDA" with an arc trajectory, however, the baseline is not equipped with such a drag tool, it is only trained and equipped with linear drags, thus failing the task.

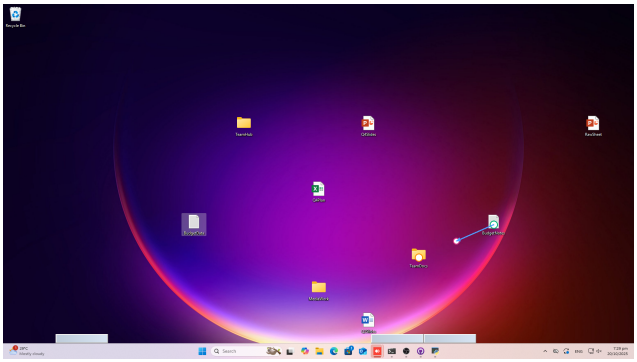


Figure 3. **Gap between discrete tool use and continuous drag.** The baseline formulates a correct plan to drag the file icon to the folder and successfully locates the file icon's initial position, but the execution fails mid-trajectory, leaving the icon stranded far from the target.

continuous actions with on-the-fly observation, thus they cannot finely adjust the cursor along the path, often halting mid-way.

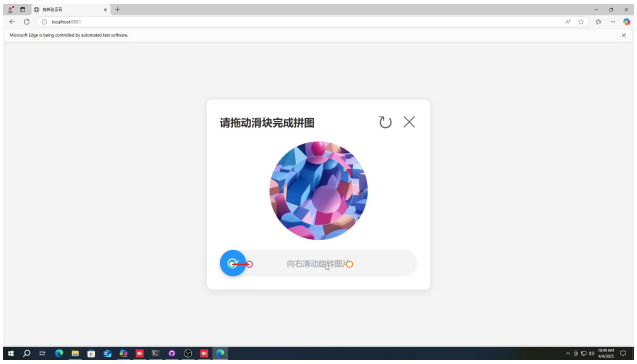


Figure 4. **Safety over action.** The model initiates a drag on the captcha slider but immediately halts and issues a refusal, misinterpreting the standard UI interaction as a safety violation.

(iii) Safety over action. As shown in Fig. 4, under the rotate-captcha tasks, Gemini 2.5 computer-use often recognizes the Captcha and starts a drag, but then halts and emits a safety refusal, declaring that it cannot perform the action. For example, when performing Captcha-solving tasks, it will say *I see that the next action is to interact with a CAPTCHA. I am unable to solve CAPTCHAs and need you to complete it for me.* These behaviors expose an alignment tax: safety filters and RLHF objectives tuned for general-purpose chat misclassify benign UI manipulations, *e.g.*,, captcha solving as risky or inappropriate, so the agent learns that refusing to act is safer than executing the requested drag. However, such tasks are common and valuable for GUI agents.

(iv) Semantic misread. As shown in Fig. 5, in handwriting-style episodes such as "Write Alice Brown on canvas", the Seed-1.6-Vision baseline is instructed to write the name on the canvas, but the trajectory moves toward a window control in the corner instead. Here the instruction is short and unambiguous, the pen tool has already been selected, yet the agent behaves as if the task were to manage the window rather than interact with the canvas. This suggests

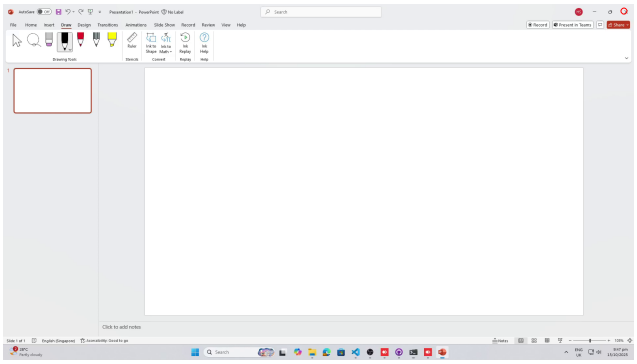


Figure 5. **Semantic misread.** The model misinterprets the visual instruction, moving the cursor to a non-target corner instead of the canvas, indicating a failure in task understanding.

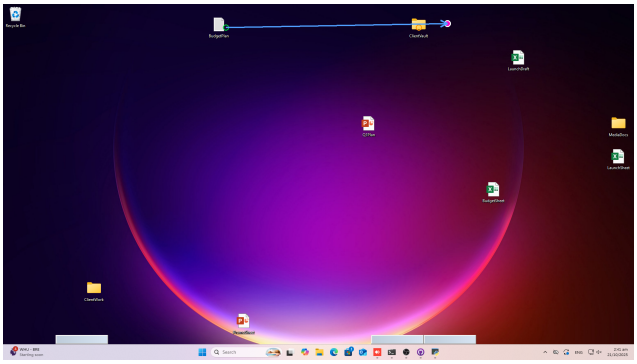


Figure 7. **Geometric precision.** The predicted trajectory follows the correct direction but significantly overshoots the target, highlighting a lack of fine-grained action control.

that strong priors from standard GUI layouts, *i.e.*, menus, close buttons, toolbars, dominate over other UI elements and tasks, leading the model to favor canonical UI elements over the specific target indicated in the prompt.

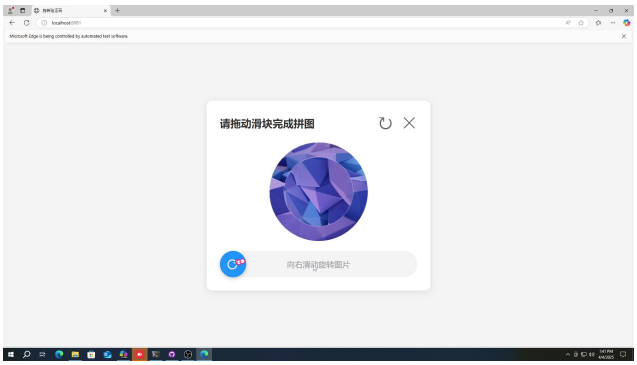


Figure 6. **Wrong primitive choice.** Instead of a continuous drag action required for the slider, the baseline issues a series of discrete clicks, failing to execute the task.

(v) Wrong primitive choice. As shown in Fig. 6, for Qwen3-VL-2B/8B, a common failure mode is to approximate a drag as a series of local `left_click` actions, as illustrated in Fig. 6. On both desktop drag and slider-captcha tasks, the model repeatedly clicks on the element instead of committing to a continuous drag, so the cursor never moves the required distance. This behavior reflects an inductive bias inherited from GUI pre-training data, where discrete clicks are the dominant interaction primitive; when transferred to tasks that require continuous actions, the model keeps reaching for the familiar click action and never fully enters the drag regime.

(vi) Direction right, magnitude wrong. As shown in Fig. 7, OpenCUA-32B typically produce PyAutoGUI tool calls that move in the correct direction but misestimate how far to drag. Handles and icons are moved broadly along the right axis, yet the final position overshoots the target folder.

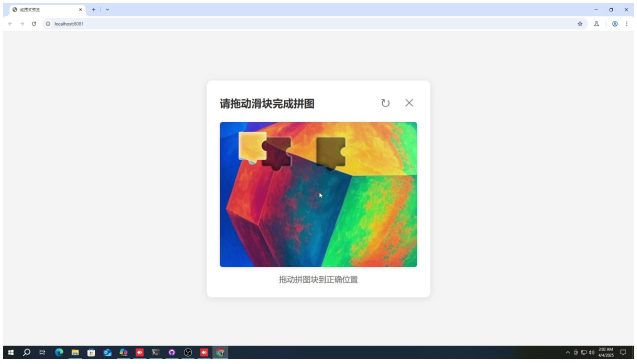


Figure 8. **Dialogue hijacks control.** The model pauses execution to ask for unnecessary clarification, halting progress in an embodied setting where autonomous action is expected.

(vii) Dialogue hijacks control. As shown in Fig. 8, under the Captcha-solving tasks, OpenAI Operator baseline often executes at most one tentative click and then pauses to ask the user to complete the task for it, instead of continuing finishing the task. For example, it will say *I see a drag-and-drop captcha on the screen. Can you please complete it?* This behavior exposes an objective mismatch: RLHF tuning for helpful conversation encourages asking clarifying questions, but benchmarks expect autonomous problem solving with no human in the loop.

(viii) Early termination. In some handwriting tasks, when the model is instructed to write a phrase on the canvas, UI-TARS-1.5-7B sometimes immediately terminates an episode with an "Instruction Unclear" message and executes no further actions. While this is reasonable as a safety mechanism in open-ended dialogue, in benchmarks it produces deterministic failures on tasks that are diverse but still clearly solvable from visual context.

Why ShowUI-π differs. As a lightweight flow-based VLA,

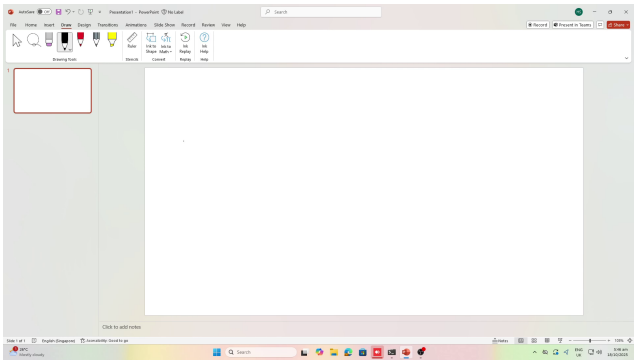


Figure 9. **Early termination.** The model terminates the episode immediately with an "Instruction Unclear" error, refusing to attempt the task.

ShowUI-π is capable of generating continuous actions with observations on-the-fly, performing free-form drags without relying on pre-defined tools. Therefore, it can adjust the fine-grained cursor motion along the task execution, addressing the problems encountered by baseline models, *e.g.*,, problems in Case (i), Case (ii), Case (v), and Case (vi).

E. Limitations and Future Work

We trained ShowUI-π at a small model size and limited training data scale. In our future work, we plan to scale up the model size with more parameters and also larger training data scale from our data collection pipeline and external data. Meanwhile, We will explore text-centric planning integration with ShowUI-π.