SwipeGen: Bridging the Execution Gap in GUI Agents via Human-like Swipe Synthesis

Anonymous ACL submission

Abstract

With the widespread adoption of Graphical User Interface (GUI) agents for automating GUI interaction tasks, substantial research focused on improving GUI perception to ground task instructions into concrete action steps. However, the step execution capability of these agents has gradually emerged as a new bottleneck for task completion. In particular, existing GUI agents often adopt overly simplified strategies for handling *swipe* interactions, preventing them from accurately replicating human-like behavior. To address this limitation, we decompose human swipe gestures into multiple quantifiable dimensions and propose an automated pipeline `SwipeGen` to synthesize human-like swipe interactions through GUI exploration. Based on this pipeline, we construct and release the first benchmark for evaluating the swipe execution capability of GUI agents. Furthermore, leveraging the synthesized data, we propose `GUISwiper`, a GUI agent with enhanced interaction execution capabilities. Experimental results demonstrate that `GUISwiper` achieves a swipe execution accuracy of 69.07%, representing a 214% improvement over existing VLM baselines. Our code, dataset, and model are available at `https://anonymous.4open.science/r/UI-anoy-91BC/`.

1 Introduction

Graphical User Interface (GUI) agents (Chen et al., 2025b; Nguyen et al., 2025) are autonomous systems that can perform human-like GUI interactions according to natural language commands. These agents have been widely adopted in real-world mobile assistants (Wang et al., 2024a; Zhang et al., 2025a), accessibility tools (Peng et al., 2025), and automated GUI testing (Hu et al., 2024; Feng et al., 2025). With recent advances in Vision-Language Models (VLMs) (Wang et al., 2024b; Bai et al., 2025), VLM-based GUI agents (Cheng et al., 2024; Hong et al., 2024; Lin et al., 2025; Wu et al., 2025; Gou et al., 2025; Lu et al., 2025; Luo et al., 2025) have significantly advanced in GUI perception, leading to more reliable and consistent interaction decisions.

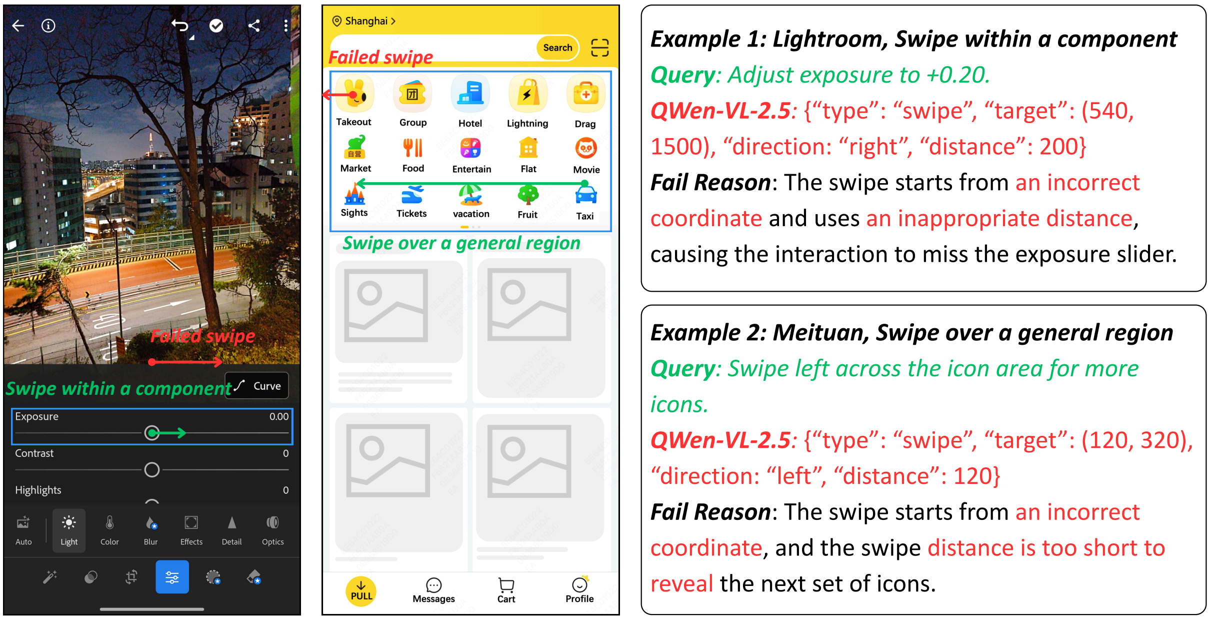
However, such improvements primarily stem from better GUI semantic understanding, while the generated GUI interactions still deviate substantially from human interaction patterns. Specifically, existing GUI agents still struggle to perform *swipe*, one of the most frequently used interactions. A key reason is that most GUI agents *implicitly assume* GUI interactions to be *component-centric*. Specifically, they formulate GUI interaction as an action prediction task that maps a natural language command to an action type t and the coordinate (x, y) of a target GUI component. Some approaches (Cheng et al., 2024; Gou et al., 2025; Chen et al., 2025a; Tang et al., 2025) even simplify this formulation as a pure GUI grounding task, *i.e.*, predicting only the target component's coordinate (x, y).

However, such overly simplified, component-centric interaction strategy does not hold for swipes. As shown in Figure 1, swipes in practice can be broadly categorized into two types: 1) *Swiping within a component* for fine-grained adjustments (*e.g.*, dragging the exposure slider in Lightroom[1]), and 2) *Swiping over a general region* for content exploration (*e.g.*, swiping to reveal additional options in a lifestyle app, Meituan[2]). Unlike clicks and text inputs, swipes are not necessarily anchored to specific components. As a result, existing VLM-based agents struggle to handle swipes under current task formulations, preventing them from completing a wide range of everyday tasks.

Addressing this issue is challenging under existing datasets (Li et al., 2020; Burns et al., 2022;

[1]`https://play.google.com/store/apps/details?id=com.adobe.lrmobile&hl=en`

[2]`https://play.google.com/store/apps/details?id=com.sankuai.meituan&hl=en`



Example 1: Lightroom ***Example 2: Meituan***

Figure 1: Two common types of swipe interactions in mobile GUIs.

Lu et al., 2024a; Cheng et al., 2024; Wu et al., 2025; Chai et al., 2025; Chen et al., 2025b; Zhang et al., 2025c; Li et al., 2025; Rawles et al., 2025). There exist two key challenges: 1) *Biased action distributions.* Current GUI interaction datasets are heavily dominated by component-centric actions such as clicks and text inputs, often accounting for 76.4% to 94.9% of all interactions, while swipes are sparsely annotated. 2) *Improper formulation of swipe.* Unlike clicks, a swipe requires accurately predicting multiple parameters, including the starting position, ending position, direction, and duration. However, the few existing datasets (Li et al., 2020; Burns et al., 2022; Lu et al., 2024a; Zhang et al., 2025c; Chai et al., 2025) that include swipe interactions typically either ignore these parameters or oversimplify them (*e.g.*, containing only a coarse direction). As a result, existing fine-tuned VLMs can hardly ground swipes correctly or generate valid swipe parameters when such interactions are required.

In this paper, we present `SwipeGen`, an automatic pipeline for synthesizing human-like swipe data without relying on predefined human instructions. To properly formulate swipe interactions, `SwipeGen` decomposes each swipe into multiple execution dimensions, including the starting position, direction, distance, and velocity, consistent with widely used mobile automation tools (Developers, 2025a; Appium, 2025; Developers, 2025c). Based on this definition, `SwipeGen` then automatically explores GUIs to synthesize human-like swipes and record all execution-required parameters. Specifically, it first detects scrollable targets (components and regions) on the screen, then executes candidate swipes, and finally verifies their validity by comparing GUI states before and after the interaction. Finally, `SwipeGen` retains only swipes that induce visual changes, enabling high-quality data collection.

Moreover, we demonstrate the effectiveness of `SwipeGen` by fine-tuning open-source VLMs for GUI interaction, resulting in a swipe-capable VLM, `GUISwiper`. Trained on data synthesized by `SwipeGen`, `GUISwiper` achieves a swipe success rate of 69.07% on our swipe benchmark.

Our primary contributions are as follows:

- We propose `SwipeGen`, an automated pipeline for synthesizing human-like and valid swipe interactions for mobile apps. By decomposing swipe execution into multiple dimensions and automatically recording these parameters during exploration, `SwipeGen` addresses improper swipe formulations in existing datasets.

- We introduce `SwipeBench`, the first benchmark for evaluating the quality of swipe interactions generated by GUI agents. `SwipeBench` is constructed using `SwipeGen` and consists of 152 high-quality swipes

Dataset	#Interactions	Action Distribution (%)			Swipe Annotation				
		Click+Text	Swipe	Other	Description	Start Pos	End Pos	Direction	Duration
AndroidHowTo (Li et al., 2020)	136,023	94.9	5.1	10.9	✓	×	×	×	×
MoTIF (Burns et al., 2022)	12,244	94.1	5.9	0.0	×	✓	✓	✓	×
GUI-Odyssey (Lu et al., 2024a)	110,056	89.4	9.5	6.9	×	✓	✓	×	×
CAGUI (Zhang et al., 2025c)	3,916	97.4	2.0	0.6	×	✓	✓	×	×
AMEX (Chai et al., 2025)	35,661	76.4	21.4	2.2	×	✓	✓	×	×

Table 1: **Action distribution and swipe parameter annotation across representative GUI datasets.** Most existing datasets are dominated by component-centric actions (click and text input), while swipes are sparsely annotated. (**Other** denotes dataset-specific actions such as system-level navigation (*e.g.*,, back), long press, or composite gestures.)

collected from 16 newly released mobile apps. All the apps are released after the public release of Qwen2.5 (Bai et al., 2025) to assess agents' out-of-domain (OOD) generalization capability.

- We develop `GUISwiper`, a swipe-capable VLM for GUI agent trained on data synthesized by `SwipeGen`. Experiments show that `GUISwiper` significantly improves swipe execution accuracy by 214%.

2 Related Work

VLM for GUI Agents Despite the strong capabilities of general-purpose large VLMs like GPT-4V (OpenAI, 2023), their performance in understanding and interacting with GUIs remains limited (Yan et al., 2023). This limitation has motivated researchers to fine-tuning open-source VLMs, such as the QWen-VL series (Bai et al., 2023; Wang et al., 2024b; Bai et al., 2025; Team, 2025), to build GUI-specific models that better comprehend user commands and GUI images.

Training these models generally follows two paradigms: early-stage supervised fine-tuning (SFT) and more recent reinforcement learning (RL) approaches. Under the SFT paradigm, models such as SeeClick (Cheng et al., 2024), CogAgent (Hong et al., 2024), OS-Atlas (Wu et al., 2025), UGround (Gou et al., 2025), ShowUI (Lin et al., 2025), and UI-TARS (Qin et al., 2025) demonstrate improved GUI understanding through supervised training on large, labeled datasets. However, SFT heavily depends on high-quality annotations, resulting in high training costs. With the emergence of DeepSeek-R1 (DeepSeek-AI, 2025), researchers found that reinforcement learning with verifiable rewards (RLVR), particularly group relative policy optimization (GRPO) (Shao et al., 2024), is well suited for GUI navigation tasks. Models such as UI-R1 (Lu et al., 2025), GUI-R1 (Luo et al., 2025), and BTL-UI (Zhang et al., 2025b) leverage predefined, task-specific reward functions to train more capable GUI-specific VLMs.

Overall, the performance of GUI-specific VLMs is strongly tied to the distribution and quality of their training data. This observation motivates our work, which targets the lack of diverse and executable swipe data by proposing a scalable pipeline to enhance the swipe capabilities of VLM for GUI agents.

Existing GUI Datasets Existing GUI datasets can be broadly categorized into GUI grounding datasets and GUI navigation datasets. The GUI grounding task aims to locate the target GUI component given a natural language command. Under this component-centric formulation, GUI grounding datasets naturally emphasize click and text input actions, excluding swipes. Representative benchmarks such as ScreenSpot (Cheng et al., 2024) and its subsequent variants (Wu et al., 2025; Li et al., 2025) follow this paradigm and therefore do not annotate swipes.

In contrast, GUI navigation datasets aim to model multi-step interaction trajectories for accomplishing high-level tasks. However, as summarized in Table 1, existing navigation datasets suffer from several limitations that hinder their use for training reliable swipes.

First, the *most critical limitation* is the lack of step-level natural language supervision. Most navigation datasets provide only a single high-level task description (*e.g.*, book a hotel near the city center) paired with a sequence of annotated low-level interactions. Individual actions, including swipes, are *not associated with corresponding descriptions*, making these datasets unsuitable for training single-step swipe prediction models.

Second, the action distribution is heavily skewed toward component-centric interactions such as

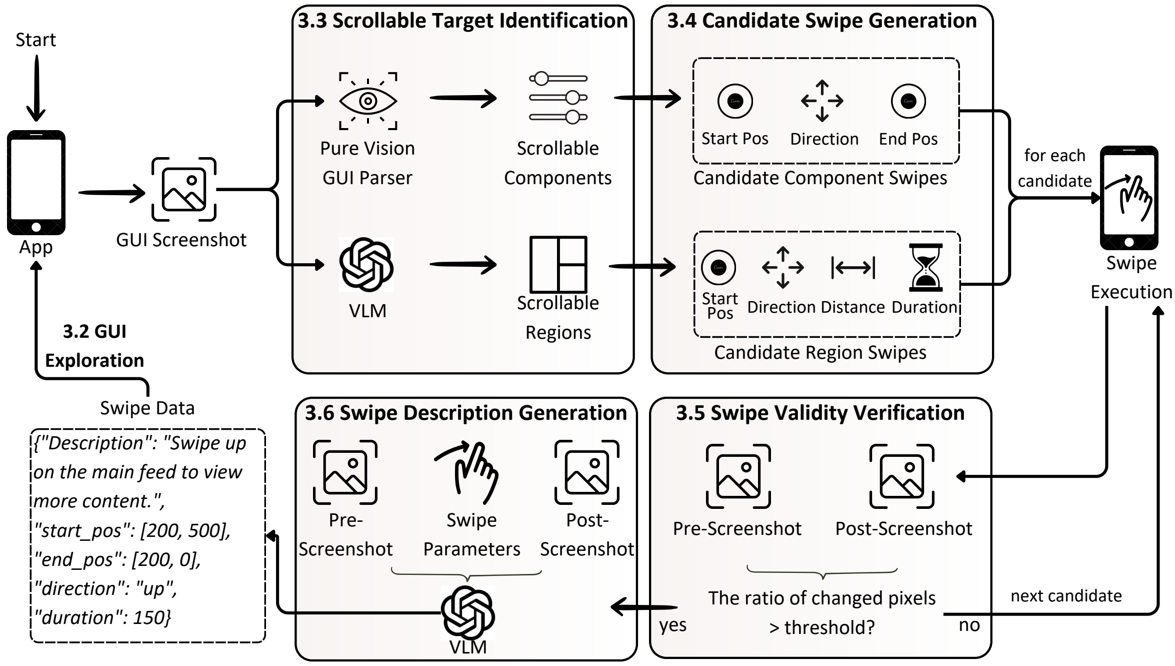


Figure 2: **Overview of the proposed pipeline `SwipeGen`.** It consists of five important modules: GUI Exploration Module, Scrollable Target Identification Module, Candidate Swipe Generation Module, Swipe Validity Verification Module and Swipe Description Generation Module.

clicks and text input.

Third, even when swipes are included, their annotations are often incomplete. As shown in Table 1, existing datasets typically miss one or more parameters required for executable swipes, such as explicit direction or duration. Although the swipe direction can be inferred from the start and end positions, we argue that explicitly annotating direction is important. Moreover, swipe duration directly affects the execution outcome due to OS-level gesture dynamics. We would further discuss the role of these parameters in Section 3.1.

Overall, these limitations motivate the need for a dataset that provides step-level language description and complete parameter annotations for swipes.

3 SwipeGen

This section introduces `SwipeGen`, an automatic pipeline for synthesizing diverse and executable swipe data without relying on predefined GUI commands. Figure 2 provides an overview of the pipeline.

Given a GUI screen, `SwipeGen` first identifies scrollable targets(*i.e.*, components and regions). Based on the identified targets, `SwipeGen` generates and executes candidate swipes under a unified swipe representation. To ensure data quality, each executed swipe is verified by comparing GUI images before and after it. For each valid swipe, `SwipeGen` generates a description based on the pre- and post-swipe images together with the swipe parameters.

3.1 Unified Swipe Representation

Before introducing `SwipeGen`, a fundamental question is unanswered: what parameters should a swipe annotation include for practical deployment?

For `SwipeGen`, our guiding principle is that synthesized swipe data should be *directly executable* in real-world GUI agent systems. In GUI agent systems, the outputs of VLMs are consumed by downstream GUI automation tools to interact with GUIs (Zhang et al., 2025a; Wang et al., 2024a). Therefore, a valid swipe representation must contain sufficient parameters to invoke these tools.

To this end, we survey widely used mobile GUI automation tools for both Android and iOS, including Android Debug Bridge (ADB) (Developers, 2025a), UIAutomator (Developers, 2025c), and Appium (Appium, 2025). As detailed in Appendix B, all these tools parameterize swipes using explicit start and end positions, and two of them further support specifying the swipe duration.

We then analyze how these parameters affect different types of swipe interactions. For component-centric swipes that aim to make fine-grained adjustments (*e.g.*, adjusting a slider), the execution outcome is determined by the start and end positions. In contrast, swipes over general regions exhibit different characteristics. As illustrated in Figure 1, the resulting scroll behavior is mainly effected by the starting position, the swipe direction, and the duration. Notably, duration directly controls the swipe velocity, which in turn determines how far the content scrolls. As a result, even with identical start and end positions, varied swipe duration can lead to different effect. And the exact endpoint is less important for region-level swipes. Therefore, although the direction can be derived from the start and end positions, we explicitly annotate it instead to facilitate learning controllable swipes.

Based on these observations, we unify swipes using four explicit parameters: start position, end position, direction, and duration. An annotated example is provided in Appendix C.1.

3.2 GUI Exploration

To scale data collection across diverse screens, `SwipeGen` adopts a random GUI exploration strategy that expands screen coverage.

The pipeline first identifies clickable GUI components using OmniParser (Lu et al., 2024b), a pure vision tool for parsing GUI screenshots into structured elements. At each step, `SwipeGen` randomly selects an unvisited clickable element and clicks it to trigger navigation. All executed clicks are recorded to avoid repeated navigation.

After each click, `SwipeGen` determines whether a new GUI screen has been reached by applying a state-change verification mechanism (which will be detailed in Section 3.5). Once a new screen is detected, the pipeline resumes swipe synthesis on the newly reached interface.

3.3 Scrollable Target Identification

For each GUI screen, `SwipeGen` identifies two types of scrollable targets: scrollable components and scrollable regions. And these two targets require different identification strategies.

Scrollable components are explicit GUI elements that support swipes, such as sliders or progress bars. There exist multiple approaches for identifying such components, each with different trade-offs. One common approach parses GUI hierarchy files, such as accessibility (a11y) trees or XML layouts. It identifies scrollable components by checking whether a node's corresponding attribute is set to `true` (*e.g.*, `is_scrollable` in `a11y` nodes or `scrollable` in XML nodes), and then localizes the component based on the node's bounding box. While this approach can be accurate when reliable hierarchy files are available, its generalization is limited. Many real-world applications contain WebView components (Developers, 2025b) that are missing or incomplete in GUI hierarchy files, causing such methods to fail. An alternative is purely vision-based GUI parsing. Models such as OmniParser (Lu et al., 2024b) infer component boundaries and supported interaction types directly from GUI screenshots. This approach generalizes better across diverse apps, although it may sacrifice some precision compared to structure-based methods. Since `SwipeGen` is equipped with subsequent swipe validity verification module, it can tolerate false positives. We therefore adopt the pure vision-based approach.

Scrollable regions, in contrast, refer to layout-level areas that support exploratory swipes, such as content feeds, lists, or icon grids. Unlike scrollable components, these regions often do not correspond to explicit GUI elements in the GUI hierarchy file. Instead, they are defined by their visual layout and semantic function, which cannot be reliably captured by structural metadata alone. We therefore leverage a VLM to directly infer scrollable regions from GUI screenshots. Specifically, we use `Qwen3-VL-4B-Instruct` (Team, 2025), an advanced multimodal model with strong GUI understanding capability, to reason over visual appearance. This enables `SwipeGen` to identify scrollable regions in a flexible and app-agnostic manner. The prompt design and an example of identified region are provided in Appendix C.2.

3.4 Candidate Swipe Generation

Given identified scrollable targets, `SwipeGen` generates candidates according to the target type.

Scrollable Components. For scrollable components, we generate candidates as follows. Let $b = (x_1, y_1, x_2, y_2)$ denote the component bounding box. We set the swipe start position to the box center,

$$s = (s_1, s_2) = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right) \quad (1)$$

The swipe orientation is determined by the component aspect ratio: vertical swipes are considered if $(y_2 - y_1) > (x_2 - x_1)$, and horizontal swipes otherwise. A swipe direction dir is then randomly sampled from the valid orientations (up and down for vertical swipes, left and right for horizontal swipes). The swipe distance is randomly sampled as a fraction of the screen size. Specifically, let $\alpha \in (0, 1]$ denote a random scaling factor. For a horizontal swipe, the distance is defined as

$$d = \alpha \cdot W, \tag{2}$$

and for a vertical swipe,

$$d = \alpha \cdot H, \tag{3}$$

where W and H are the screen width and height, respectively. Given the sampled direction, the end position is obtained by translating the start position s along the swipe direction by distance d, while ensuring the end position remains within the screen boundary. For a horizontal swipe, the end position is computed as

$$e = \begin{cases} (\min(s_1 + d,\, W),\, s_2), & \text{if direction is right,} \\ (\max(s_1 - d,\, 0),\, s_2), & \text{if direction is left,} \end{cases} \tag{4}$$

and for a vertical swipe,

$$e = \begin{cases} (s_1,\, \min(s_2 + d,\, H)), & \text{if direction is down,} \\ (s_1,\, \max(s_2 - d,\, 0)), & \text{if direction is up,} \end{cases} \tag{5}$$

As discussed in Section 3.1, duration plays a limited role for such swipes. Therefore, we fix the duration t to a default value of 300ms. Overall, each component yields 2 candidate swipes with different swipe directions, represented as (s, e, dir, t)

Scrollable Regions. Similarly, for each scrollable region with bounding box b, we first compute its center position c. We identify the dominant axis of the region by comparing its height and width. If $(y_2 - y_1) > (x_2 - x_1)$, the region is treated as vertically scrollable. Otherwise, it is treated as horizontally scrollable. Then, we offset the center c along the dominant axis to generate candidate start positions close to the region's boundary. Specifically, let $\alpha \in [0.2, 0.5)$ denote a random offset ratio. For a horizontally scrollable region, the candidate start positions are

$$s = \begin{cases} (c_1 + \alpha(x_2 - x_1),\, c_2), \\ (c_1 - \alpha(x_2 - x_1),\, c_2), \end{cases} \tag{6}$$

and for a vertically scrollable region,

$$s = \begin{cases} (c_1,\, c_2 + \alpha(y_2 - y_1)), \\ (c_1,\, c_2 - \alpha(y_2 - y_1)). \end{cases} \tag{7}$$

For each start position s, the swipe direction is set opposite to the offset direction, reflecting natural scrolling behavior (*e.g.*, offsetting right corresponds to a left swipe). The end position is then obtained by extending the swipe along this direction until reaching the region boundary. For horizontal swipes,

$$e = \begin{cases} (x_1,\, s_2), & \text{if direction is left,} \\ (x_2,\, s_2), & \text{if direction is right,} \end{cases} \tag{8}$$

and for vertical swipes,

$$e = \begin{cases} (s_1,\, y_1), & \text{if direction is up,} \\ (s_1,\, y_2), & \text{if direction is down.} \end{cases} \tag{9}$$

Moreover, the execution outcome of swipes over a region is sensitive to swipe speed. In real-world systems, user control over swipe gestures is inherently coarse-grained (*e.g.*, fast vs. slow swipes), rather than continuous. This is also reflected in OS-level gesture recognizers, which typically rely on velocity thresholds to determine scrolling behavior. Accordingly, we divide swipe duration into two categories: a fast swipe (150ms) and a slow swipe 500ms. Overall, each region yields $2 \times 2 = 4$ candidate swipes with different start points and duration, represented as (s, e, dir, t).

3.5 Swipe Validity Verification

However, not every candidate swipe leads to a GUI response. To ensure data quality, `SwipeGen` adopts an execute-and-verify procedure for swipe selection. For each scrollable target, candidate swipes are executed sequentially and verified one by one, and the first swipe that induces a perceptible GUI change is retained as a valid sample.

To achieve this, we compare the visual changes between the screenshots before and after execution. Specifically, we convert the two screenshots to grayscale, and restrict the comparison within the target area (*i.e.*, the bounding box of the scrollable component or region). We then compute the pixel-wise absolute difference and measure the ratio of pixels whose intensity change exceeds a fixed threshold $\delta = 0.02$. If the swipe is considered effective and retained, `SwipeGen` moves on to the next GUI. Otherwise, the current candidate is discarded, and `SwipeGen` executes the next swipe.

3.6 Swipe Description Generation

Finally, for each validated swipe, `SwipeGen` generates a corresponding step-level natural language description. Specifically, we prompt a VLM with the GUI screenshots before and after the swipe, together with the executed swipe parameters, and ask it to describe the performed interaction in natural language. The prompt and example outputs are provided in Appendix C.3.

4 SwipeBench

We introduce `SwipeBench`, the first benchmark specifically designed for evaluating swipe execution in GUI agents. `SwipeBench` is constructed using `SwipeGen` and consists of 152 executable swipes collected from 16 mobile applications.

`SwipeBench` is designed to evaluate GUI agents under out-of-domain (OOD) scenarios, minimizing potential data leakage from VLM pretraining. Concretely, all selected applications are newly released and avoid overlap with commonly used or previously benchmarked mobile apps, which may already be exposed in large-scale vision-language pretraining corpora. The distribution of `SwipeBench` are provided in Appendix D.

5 Experiments

This section evaluates whether our data generation pipeline can be used to enhance swipe execution capabilities of GUI-specific VLMs.

5.1 GUISwiper

We implement `GUISwiper`, a GUI-specific VLM fine-tuned to perform multiple types of interactions with GUIs, especially swipes. To enhance generalization while keeping training cost low, we employ the widely adopted RLVR method (Lu et al., 2025; Luo et al., 2025; Liu et al., 2025; Chen et al., 2025a; Tang et al., 2025) on a Qwen2.5-VL-3B-Instruct (Bai et al., 2025) base model.

Training Dataset To demonstrate the *quality* of the data generated by `SwipeGen`, we fine-tune `GUISwiper` using a small yet diverse training set consisting of only 185 interaction samples. All training data are automatically generated by `SwipeGen` on popular mobile applications, which is detailed in Appendix E.1. Specifically, the dataset includes 124 swipes and 61 clicks collected during the GUI exploration process. These interactions span multiple domains, including entertainment, shopping, lifestyle, etc.

Reward Design We design the overall reward as a combination of three components: a format reward, an action type reward, and an action accuracy reward. The final reward is linearly normalized to $[-1, 1]$ for stable training.

Format Reward. We adopt a widely used format reward (Lu et al., 2025; Zhang et al., 2025b) to enforce structured outputs. The model is required to produce reasoning enclosed in <think> tags followed by a final answer in a predefined JSON format. Correctly formatted outputs receive a reward of $+1$, while malformed outputs receive -1.

Action Type Reward. To prevent overfitting to swipe actions and preserve general GUI interaction capabilities, we include an action type reward. If the predicted action type (*e.g.*, `swipe`, `click`, `input`) matches the ground truth, the model receives $+0.8$; otherwise, -0.8.

Action Accuracy Reward. For non-swipe actions, we follow the same accuracy reward design as UI-R1 (Lu et al., 2025). As for swipes, we define an accuracy reward $R_{acc} \in [0, 1]$ as the sum of four sub-rewards:

$$R_{acc} = R_{\text{start}} + R_{\text{end}} + R_{\text{dir}} + R_{\text{dur}}. \tag{10}$$

(1) If the Euclidean distance between the predicted and ground-truth start positions is within 220 pixels, a reward of 0.45 is given; otherwise, 0. Following prior work (Liu et al., 2025), we normalize all the image resolutions to $(0, 0, 1000, 1000)$, and set a 220-pixel tolerance for allowing reasonable localization errors. Additionally, for region-level swipes, a constraint is enforced: if the predicted start position lies outside the target bounding box, the start-position reward is set to 0 regardless of distance. Let $\hat{s}$ and s denote the predicted and ground-truth start positions, we define:

$$R_{\text{start}} = \begin{cases} 0.45, & \|\hat{s} - s\|_2 \leq 220 \text{ and } \hat{s} \in B \\ 0, & \text{otherwise,} \end{cases} \tag{11}$$

where B denotes a bounding box:

$$B = \begin{cases} (x_1, y_1, x_2, y_2), & \text{for scrollable regions} \\ (0, 0, W, H), & \text{for scrollable components} \end{cases} \tag{12}$$

Following prior work (Liu et al., 2025), .

(2) If the predicted end position is within 220 pixels of the ground truth, a reward of 0.10 is given. This term is assigned a lower weight since end positions are less important for scrollable regions. Let $\hat{e}$ and e denote the predicted and ground-truth

Model	Model Size	Accuracy (%)
Qwen2.5-VL-Instruct	3B	32.24
GUISwiper (ours)	3B	**69.07**

Table 2: Swipe execution accuracy on SwipeBench.

end positions:

$$R_{\text{end}} = \begin{cases} 0.10, & \|\hat{e} - e\|_2 \leq 220 \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

(3) If the predicted swipe direction (*e.g.*, up) matches the ground truth, a reward of 0.35 is given. Let $\hat{\text{dir}}$ and dir denote the predicted and ground-truth swipe directions:

$$R_{\text{dir}} = \begin{cases} 0.35, & \hat{\text{dir}} = \text{dir} \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

(4) For component-level swipes, the reward of 0.1 is always granted. For region-level swipes, we discretize duration into fast (150 ms) and slow (500 ms). Predicted durations are mapped to these categories using a midpoint threshold, and a reward of 0.10 is given if the predicted category matches the ground truth.

Notably, all accuracy rewards are non-negative: incorrect predictions do not incur penalties, which is crucial for stable training.

Implementations Additional implementation details are provided in Appendix E.2, Appendix E.3, and Appendix E.4.

5.2 Experimental Settings

Benchmarks We evaluate GUI agents on SwipeBench, our newly constructed benchmark, which emphasizes out-of-domain (OOD) generalization.

Baselines We compare GUISwiper with the base model **Qwen2.5-VL-Instruct** (Bai et al., 2025). We can therefore directly assess the effectiveness of our synthesized swipe data and training strategy.

5.3 Experimental Result and Analysis

Table 2 reports the swipe execution accuracy of different models on SwipeBench. For each swipe instruction, we formulate the evaluation as a binary classification task, where a prediction is considered correct only if all four swipe parameters satisfy the same criteria used in the accuracy reward function during training.

As shown in the table, the base model achieves a relatively low accuracy of around 32%, indicating that directly prompting a general-purpose VLM is insufficient for accurate swipe execution. In contrast, GUISwiper, fine-tuned using our synthesized swipe data, significantly improves the swipe execution accuracy by 214%. Since both models share the same size, we can found that the performance gain is primarily attributed to improved training data, rather than increased model capacity.

We further analyze the failure cases of GUISwiper to understand the remaining challenges. First, approximately 17% of the errors are caused by inaccurate swipe distances, where the predicted end point deviates from the expected location. Second, region-level swipe interactions remain particularly challenging. Around 40% of the failures stem from selecting an invalid start point outside the actual scrollable region. This suggests that determining a valid swipe starting location requires jointly reasoning about scrollable regions and user command, which is substantially more difficult than merely identifying scrollable regions. Finally, around 43% of the errors are related to incorrect swipe duration. Distinguishing whether a swipe should be performed quickly or slowly is often ambiguous from static visual information, highlighting a motivation for modeling fine-grained temporal dynamics in GUI interactions.

6 Conclusion

This paper identifies that widely used component-centric interaction strategies adopted by GUI agents often fail to complete GUI interaction tasks due to their inability to replicate human-like swipe interactions. To address this limitation, we decompose human swipe gestures into multiple quantifiable dimensions and propose SwipeGen, an automated pipeline for synthesizing human-like swipe interactions. By deploying SwipeGen on 16 newly released mobile apps, we introduce SwipeBench, the first GUI swipe execution benchmark containing 152 swipe interactions. Furthermore, a swipe execution enhanced GUI agent, GUISwiper, is proposed by fine-tuning a VLM with these synthesized interactions. Experiments prove that GUISwiper achieves higher swipe execution accuracy. We hope that this work highlights the importance of human gesture modeling in GUI agents and could encourage future research to move toward human-like interaction execution.

Limitations

Our work has two limitations that are worth discussing. First, SwipeGen relies on randomized GUI exploration to collect swipe data. As a result, it cannot guarantee exhaustive coverage of all scrollable regions and components within a given app. Nevertheless, this limitation can be partially mitigated by increasing the exploration time.

Second, the natural language descriptions associated with each swipe are automatically generated by a VLM based on GUI context, rather than annotated by humans. Consequently, the quality and precision of these descriptions depend on the capabilities and biases of the underlying VLM. While this reliance may introduce potential risks such as annotation noise and model bias compared to human annotators, these risks can be alleviated by adopting stronger models. We expect future advances in VLMs to further improve the quality and reliability of the generated data.

Ethics Considerations

Our work introduces `SwipeGen`, an automated pipeline for generating swipe interaction data, along with a benchmark `SwipeBench` and a VLM for GUI agent `GUISwiper`. We discuss the relevant ethical considerations below.

Data Privacy and Legality. `SwipeGen` operates exclusively on publicly available mobile apps and interacts with GUIs in a manner consistent with standard user behavior. The pipeline does not collect personal, sensitive, or user-generated data. All GUI screenshots and interaction trajectories are obtained in a controlled environment for research purposes only.

Potential Misuse and Deployment Risks. Like other GUI automation systems, `GUISwiper` could be misused for unintended automation purposes. However, our work focuses on improving the correctness of low-level swipe execution rather than enabling end-to-end task automation. We strongly discourage deploying GUI agents without human oversight, particularly in safety-critical domains such as finance or healthcare.

Overall, we believe the benefits of enabling reliable swipes for GUI agents outweigh the associated risks when appropriate safeguards and research-oriented usage are maintained.

References

Appium. 2025. Appium: Mobile automation framework.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A. Plummer. 2022. A dataset for interactive vision-language navigation with unknown command feasibility. In *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part VIII*, volume 13668 of *Lecture Notes in Computer Science*, pages 312–328. Springer.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Guozhi Wang, Dingyu Zhang, Shuai Ren, and Hongsheng Li. 2025. AMEX: android multi-annotation expo dataset for mobile GUI agents. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 2138–2156. Association for Computational Linguistics.

Liangyu Chen, Hanzhang Zhou, Chenglin Cai, Jianan Zhang, Panrong Tong, Quyu Kong, Xu Zhang, Chen Liu, Yuqi Liu, Wenxuan Wang, Yue Wang, Qin Jin, and Steven Hoi. 2025a. Ui-ins: Enhancing GUI grounding with multi-perspective instruction-as-reasoning. *CoRR*, abs/2510.20286.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2025b. Guicourse: From general vision language model to versatile GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 21936–21959. Association for Computational Linguistics.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9313–9332. Association for Computational Linguistics.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Android Developers. 2025a. Android debug bridge (adb).

Android Developers. 2025b. Webview.

Android Developers. 2025c. Write automated tests with ui automator.

Sidong Feng, Changhao Du, Huaxiao Liu, Qingnan Wang, Zhengwei Lv, Gang Huo, Xu Yang, and Chunyang Chen. 2025. Agent for user: Testing multi - user interactive features in tiktok. In *47th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2025, Ottawa, ON, Canada, April 27 - May 3, 2025*, pages 57–68. IEEE.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for GUI agents. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 14281–14290. IEEE.

Yongxiang Hu, Xuan Wang, Yingchuan Wang, Yu Zhang, Shiyu Guo, Chaoyi Chen, Xin Wang, and Yangfan Zhou. 2024. Auitestagent: Automatic requirements oriented GUI function testing. *CoRR*, abs/2407.09018.

Kaixin Li, Meng Ziyang, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. Screenspot-pro: GUI grounding for professional high-resolution computer use. In *Workshop on Reasoning and Planning for Large Language Models*.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile ui action sequences. In *Annual Conference of the Association for Computational Linguistics (ACL 2020)*.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2025. Showui: One vision-language-action model for GUI visual agent. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pages 19498–19508. Computer Vision Foundation / IEEE.

Yuhang Liu, Zeyu Liu, Shuanghe Zhu, Pengxiang Li, Congkai Xie, Jiasheng Wang, Xueyu Hu, Xiaotian Han, Jianbo Yuan, Xinyao Wang, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025. Infigui-g1: Advancing GUI grounding with adaptive exploration policy optimization. *CoRR*, abs/2508.05731.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024a. GUI odyssey: A comprehensive dataset for cross-app GUI navigation on mobile devices. *CoRR*, abs/2406.08451.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024b. Omniparser for pure vision based GUI agent. *CoRR*, abs/2408.00203.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. 2025. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *CoRR*, abs/2503.21620.

Run Luo, Lu Wang, Wanwei He, and Xiaobo Xia. 2025. GUI-R1 : A generalist r1-style vision-language action model for GUI agents. *CoRR*, abs/2504.10458.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Md. Mehrab Tanjim, and 11 others. 2025. GUI agents: A survey. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 22522–22538. Association for Computational Linguistics.

OpenAI. 2023. Gpt-4v(ision) system card.

Yi-Hao Peng, Dingzeyu Li, Jeffrey P. Bigham, and Amy Pavel. 2025. Morae: Proactively pausing UI agents for user choices. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology, UIST 2025, Busan, Korea, 28 September 2025 - 1 October 2025*, pages 198:1–198:14. ACM.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. UI-TARS: pioneering automated GUI interaction with native agents. *CoRR*, abs/2501.12326.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Fei Tang, Zhangxuan Gu, Zhengxi Lu, Xuyang Liu, Shuheng Shen, Changhua Meng, Wen Wang, Wenqi Zhang, Yongliang Shen, Weiming Lu, Jun Xiao, and Yueting Zhuang. 2025. Gui-g^2: Gaussian reward modeling for GUI grounding. *CoRR*, abs/2507.15846.

Qwen Team. 2025. Qwen3 technical report.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024b. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2025. OS-ATLAS: foundation action model for generalist GUI agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian J. McAuley, Jianfeng Gao, Zicheng Liu, and Lijuan Wang. 2023. GPT-4V in wonderland: Large multimodal models for zero-shot smartphone GUI navigation. *CoRR*, abs/2311.07562.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025a. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, CHI 2025, YokohamaJapan, 26 April 2025- 1 May 2025*, pages 70:1–70:20. ACM.

Shaojie Zhang, Ruoceng Zhang, Pei Fu, Shaokang Wang, Jiahui Yang, Xin Du, Shiqi Cui, Bin Qin, Ying Huang, Zhenbo Luo, and Jian Luan. 2025b. BTL-UI: blink-think-link reasoning model for GUI agent. *CoRR*, abs/2509.15566.

Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, Jie Xie, Wei Zhou, Wang Xu, Yuanheng Zhang, Zhou Su, Zhongwu Zhai, Xiaoming Liu, Yudong Mei, Jianming Xu, and 6 others. 2025c. AgentCPM-GUI: Building mobile-use agents with reinforcement fine-tuning. *arXiv preprint arXiv:2506.01391*.

A Detailed Analysis of Existing GUI Datasets

This section provides a detailed analysis of how swipes are represented and annotated in existing mobile GUI navigation datasets, complementing the summary in Table 1. We focus exclusively on mobile datasets or the mobile subsets of cross-platform datasets, and exclude purely web-based datasets (*e.g.*, GUIAct (Chen et al., 2025b)) or web-only portions.

AndroidHowTo (Li et al., 2020) collects natural language how-to instructions for operating Android devices from web sources. While the dataset includes swipes in its interaction traces, it does not annotate any parameters beyond the action type, such as start position, end position, direction, or duration. Moreover, swipes are not paired with step-level natural language descriptions, further limiting their usability for training swipe prediction models.

MoTIF (Burns et al., 2022) and GUI-Odyssey (Lu et al., 2024a) annotate swipes using explicit start and end positions, and additionally represent the swipe trajectory by uniformly sampling intermediate points along the path. Specifically, MoTIF samples 30 intermediate points between the start and end positions and further provides an explicit swipe direction. GUI-Odyssey, which consists of 7,735 navigation tasks with an average of 14.36 interactions per task, adopts a similar trajectory representation based on start and end positions with sampled intermediate points, but does not annotate swipe direction. Despite these annotations, neither dataset provides explicit swipe duration information, which is critical for determining the actual execution outcome of a swipe gesture under operating system–level inertial effects. Moreover, for both datasets, each trajectory is paired with only a single high-level task description rather than step-level natural language descriptions for individual interactions, rendering them unsuitable for training single-step swipe prediction models.

CAGUI (Zhang et al., 2025c) introduces a unified `DUAL_POINT` action representation that can denote both clicks and swipes. In practice, swipe actions are sparse in this dataset, and their annotations lack explicit direction and duration information. Similarly, AMEX (Chai et al., 2025) which consists of 2,046 navigation tasks with an average of 12.71 interactions per task, represents swipes using start and end coordinates (*i.e.*, `touch_coord` and `list_coord`). But it does not annotate other parameters such as direction or duration, nor provide step-level natural language descriptions for swipes.

Overall, although several datasets include swipes in their trajectories, none provide complete and executable swipe annotations paired with step-level natural language description.

B Swipe Syntax Comparison for Popular GUI Automation Tools

Table 3 summarizes the swipe command syntax for three popular mobile GUI automation tools. All tools require explicit specification of swipe parameters, including start and end positions and, in some cases, duration. This motivates the unified and parameter-complete swipe formulation adopted by `SwipeGen`.

C SwipeGen

C.1 Illustration of the Unified Swipe Representation

```
{
"type": "region",              // Swipe type:
"component" or "region"
"start": [500, 500],        // Normalized start
position (x, y)
"end": [800, 500],          // Normalized end
position (x, y)
"direction": "up",          // One of up, down,
left, right
"duration": 300,            // Swipe duration
in milliseconds
"bbox": [200, 300, 800, 700],// Bounding
box of the scrollable target
"intent": "Scroll down to view more content
in the feed"
}
```

GUI Automation Tool	Swipe Command
Android Debug Bridge (ADB) (Developers, 2025a)	`input swipe (x1, y1, x2, y2, duration)`
UI Automator (Developers, 2025c)	`driver.swipe(x1, y1, x2, y2)`
Appium (Appium, 2025)	`driver.swipe(x1, y1, x2, y2, duration)`

Table 3: Comparison of swipe syntax for popular mobile GUI automation tools.

C.2 VLM Prompt and Output for Scrollable Region Identification

<image>
You are given a screenshot of a mobile app UI.
Your task is to identify all scrollable regions on the screen, *i.e.*, areas that support horizontal or vertical swipes, such as content feeds, lists, carousels, or grid layouts.
Please output at most 6 scrollable regions.
For each region, provide the following information:
1. type: a brief description of the region (*e.g.*, list, feed, carousel)
2. direction: if scrollable, the supported swipe direction ("horizontal", "vertical", or "both")
3. bbox: bounding box coordinates [x1, y1, x2, y2], where x and y range from 0 to 1000
4. description: a short description of the intended interaction
Output the result in JSON format.
Only output valid JSON. Do not include any additional text.

{
"type": "content feed",
"direction": "vertical",
"bbox": [0, 0, 1000, 1000],
"description": "Swipe up or down to view more content in the feed"
},
{
...
}

C.3 VLM Prompt and Output for Swipe Description Generation

<image1> <image2>
You are given two screenshots of a mobile app UI, the screen before a swipe and the screen after it.
You are also given the executed swipe with its parameters.
<swipe>
Your task is to generate a concise, step-level natural language command written in imperative form that accurately describes the performed swipe.
The command should:
1. Describe a single interaction step.
2. Based on the visual change between the two screenshots.
3. Reflect the intent of the swipe (*e.g.*, revealing more content, scrolling a list).
Output the result in JSON format, include the command itself ("command") and the reason why you describe it as so ("reason").

{
"command": "Swipe up on the main feed to view more content.",
"reason": "The swipe moves upward within the main content area, and the after-screenshot shows new items appearing below, indicating that the intent of the interaction is to scroll the feed to reveal more content."
}

D Details of SwipeBench

`SwipeBench` prioritizes data quality and privacy. All swipes are automatically generated and executed by `SwipeGen`, and are subsequently manually reviewed to filter out high-quality data. To address privacy and safety concerns, we additionally anonymize screens that may contain personally identifiable information by masking sensitive regions (*e.g.*, user names or message content).

SwipeBench is detailed in Table 4.

App	Category	#Swipes
Perplexity Comet	Communication	14
Bluesky	Communication	11
Jagat	Communication	4
Viggle Ai	Communication	10
Stellarium	Education	13
Wiser	Education	6
Pingo AI	Education	7
Arts & Culture	Education	3
Focus Friend	Efficiency	13
Gemini	Efficiency	10
Manus	Efficiency	10
Perplexity AI	Efficiency	10
Finch	Tool	19
Arc Search	Tool	8
Perch Reader	Tool	8
OmniTools	Tool	6

Table 4: Distribution of Apps in SwipeBench.

E Details of GUISwiper

E.1 Training Dataset Distribution

The training dataset distribution is detailed in Table 5.

App	Category	#Clicks	#Swipes
YouTube	Entertainment	0	3
Bilibili	Entertainment	0	3
NetEase Music	Entertainment	0	3
Discord	Communication	0	3
Zoom	Communication	8	3
WhatsApp	Communication	7	1
QQ	Communication	9	10
JD Mall	Shopping	0	3
AliExpress	Shopping	2	20
Pinduoduo	Shopping	4	19
Google Calendar	Tool	7	9
Canva	Tool	1	5
Notion	Tool	0	8
Microsoft Translator	Tool	9	6
Google Maps	Navigation	2	9
DeepSeek Chat	Efficiency	6	8
Zhihu	Education	6	8
Meituan	Lifestyle	0	3

Table 5: Training Dataset Distribution of GUISwiper.

E.2 Training Settings

The training settings are detailed in Table 6.

Hyperparameter	Value
Learning rate	from 9.9e-7 to 5.0e-7
Max pixels	1,048,576
Num generations	8
Num train epochs	8
Max prompt length	512
Per-device train batch size	1
Gradient accumulation steps	2
Optimizer	Adam
Data type	BFloat16

Table 6: Training hyperparameters used for fine-tuning Qwen2.5-VL.

E.3 Prompt for GUISwiper

<image>
<command>
You are an assistant that controls a GUI by outputting a single JSON object.
First think privately inside <think> </think>. After </think>, output ONLY one JSON object with no extra text.
Exactly one pair of <think> and </think> must appear, and the JSON must follow this schema:
{
"action": one of "tap", "swipe", "long_press", "text",
"start": [x,y] integers in [0,1000],
"end": [x,y] integers in [0,1000] (required for swipe, optional otherwise),
"direction": "up"|"down"|"left"|"right" (optional; used for swipe when applicable),
"duration": integer milliseconds (required for long$_{p}ress; optionalotherwise)$,
"text": string (required for text; empty string if unknown)
}
Rules:
- Coordinates must be normalized to [0,1000] and integers.
- Do not include any fields not listed above.
- For tap: provide start only, end must be [].
- For swipe: provide start and end; direction is preferred when meaningful.
- For long_press: provide start and duration (ms).
- For text: provide start and text; end should be [].
- Do not use markdown/code fences; output plain JSON after </think>.
Examples:
<think>Locate the search icon and tap it.</think>
{"action":"tap","start":[512,128],"end":[], "direction":null,"duration":0,"text":null}
<think>Scroll the list down to reveal more items.</think>
{"action":"swipe","start":[500,800],"end": [500,200],"direction":"up","duration":300, "text":null}

E.4 GUISwiper Visualization

Figure 3 illustrates the progression of various variables throughout the training process.

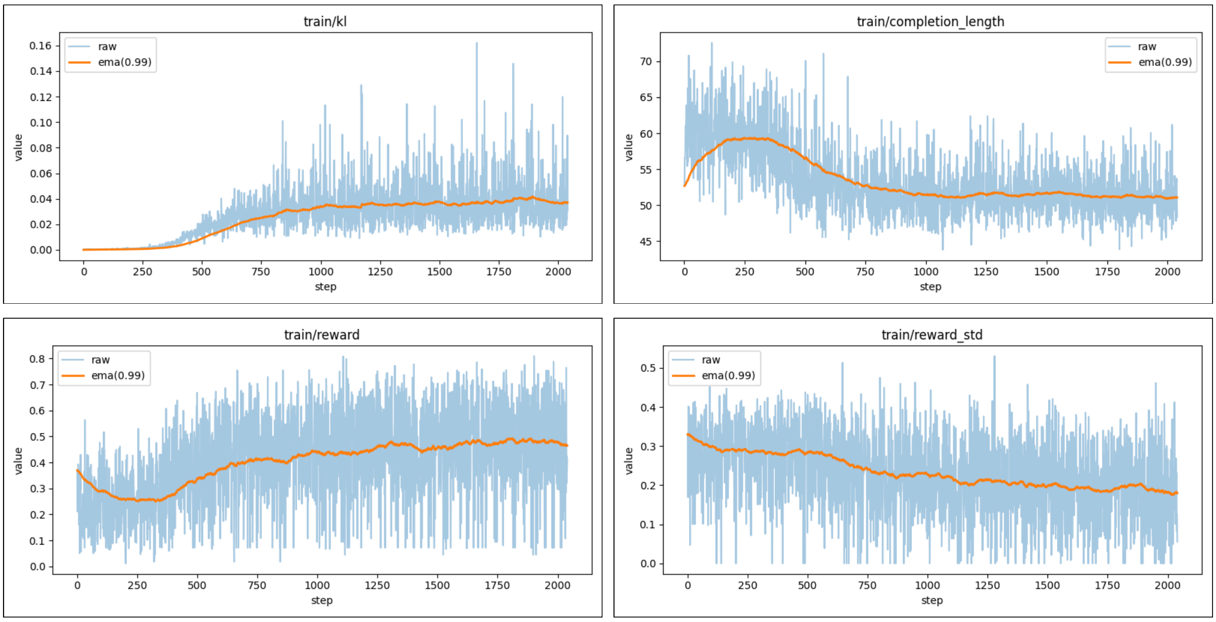


Figure 3: SwipeGen Training Process