SecAgent: Efficient Mobile GUI Agent with *Se*mantic *C*ontext

Anonymous ACL submission

Abstract

Mobile Graphical User Interface (GUI) agents powered by multimodal large language models have demonstrated promising capabilities in automating complex smartphone tasks. However, existing approaches face two critical limitations: the scarcity of high-quality multilingual datasets, particularly for non-English ecosystems, and inefficient history representation methods. To address these challenges, we present SecAgent, an efficient mobile GUI agent at 3B scale. We first construct a human-verified Chinese mobile GUI dataset with 18k grounding samples and 121k navigation steps across 44 applications, along with a Chinese navigation benchmark featuring multi-choice action annotations. Building upon this dataset, we propose a semantic context mechanism that distills history screenshots and actions into concise, natural language summaries, significantly reducing computational costs while preserving task-relevant information. Through supervised and reinforcement fine-tuning, SecAgent outperforms similar-scale baselines and achieves performance comparable to 7B-8B models on our and public navigation benchmarks. We will open-source our dataset and benchmark to advance research in mobile GUI automation.

1 Introduction

Mobile Graphical User Interface (GUI) agents powered by Large Language Models (LLMs) and Multimodal LLMs (MLLMs) have significantly evolved recently (Liu et al., 2025a). Early text-based approaches (Wang et al., 2023; Wen et al., 2024) rely on structured metadata such as accessibility trees, but suffer from platform-specific dependencies and incomplete metadata. To address these limitations, the community has shifted toward vision-based approaches that perceive interfaces directly through screenshots (Lin et al., 2025; Cheng et al., 2024), offering cross-platform compatibility and robust element grounding without structured metadata.

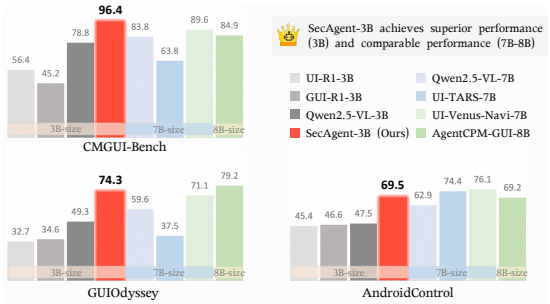


Figure 1: Performance comparison of various agent models on three representative navigation benchmarks. SecAgent-3B achieves superior performance to 3B models and comparable performance to 7B-8B models.

Despite these achievements, current mobile GUI agent research faces two critical limitations. First, existing work predominantly focuses on the English mobile ecosystem, with limited attention to non-English environments. Building robust multilingual GUI agents requires diverse multilingual and cross-cultural data with ecosystem-specific knowledge, such as element functionality and interaction conventions. However, open-source non-English datasets remain scarce. While several recent studies (Zhang et al., 2025c; Tang et al., 2025; Zeng et al., 2025) have constructed large-scale Chinese mobile datasets, they remain proprietary and inaccessible to the research community. To advance research on multilingual GUI agents, we introduce **CMGUI**, a large-scale Chinese mobile GUI dataset comprising 18k grounding samples and 29k navigation episodes with 121k navigation steps across 44 diverse applications, with each navigation step undergoing rigorous human verification and manual bounding box annotation. Building upon CMGUI, we further introduce **CMGUI-Bench**, a high-quality Chinese mobile navigation benchmark where each step is annotated with multiple valid actions when available to accommodate diverse GUI manipulations.

Second, existing methods lack efficient history

representation for multi-step navigation. Informed actions rely on historical information such as prior screenshots and actions. However, the complete history incurs prohibitive computational overhead. Current approaches thus restrict history to recent steps (Wang et al., 2025; Lin et al., 2025) or retain only action sequences (Wu et al., 2024b; Gu et al., 2025b; Luo et al., 2025), failing to fully capture historical progression. To address this challenge, we propose the **Semantic Context Mechanism**, which presents the historical information as a semantic context, a concise natural language-based summary that records critical information from previous interactions. By dynamically maintaining the semantic context across navigation steps, the agent avoids accessing the entire history of screenshots and actions, significantly improving the computational efficiency while preserving the completeness of the historical information. Based on this mechanism, we present **SecAgent**. Through supervised and reinforcement fine-tuning, SecAgent achieves highly competitive performance. As shown in Fig. 1, it outperforms prior methods on CMGUI-Bench and matches 7B-8B models on AndroidControl (Li et al., 2024) and GUIOdyssey (Lu et al., 2025a).

Our contributions are summarized as follows:

- We introduce CMGUI, a large-scale high-quality Chinese mobile GUI dataset, and CMGUI-Bench, a navigation benchmark with multi-choice annotations (Sec. 3).
- We propose the semantic context mechanism for efficient history representation via concise natural language summaries (Sec. 4).
- Extensive experiments demonstrate the superiority of SecAgent across both our proposed and public benchmarks, achieving competitive results with larger models (Sec. 5).

2 Related Works

GUI Benchmarks and Datasets. Numerous benchmarks and datasets have been introduced to advance research on mobile GUI navigation. Some works (Xu et al., 2024; Rawles et al., 2024; Chai et al., 2025) designed benchmarks to comprehensively evaluate agents' navigation capabilities, while others (Rawles et al., 2023; Li et al., 2024; Lu et al., 2025a) curated trajectory datasets to enhance the perception and planning capabilities. Compared to the English mobile ecosystem, fewer studies target the Chinese mobile ecosystem. While several Chinese benchmarks have been proposed (Zhang et al., 2025c; Chen et al., 2025a; Li et al., 2026), there remains a lack of large-scale open-source Chinese trajectory datasets. Recently, Xu et al. (2025) released a Chinese trajectory dataset by annotating instructions for random walk sequences, yet without step-level human verification. In contrast, CMGUI features human-refined instructions and rigorous step-by-step verification with precise bounding box annotations.

GUI Agents. LLM-powered GUI agents generally fall into two paradigms: prompting-based and training-based. Prompting-based agents directly query existing general models (Achiam et al., 2023) to generate GUI actions with carefully constructed prompts that contain the instruction, GUI state, and action space (Yan et al., 2023; Zhang et al., 2025b; Lu et al., 2024). Training-based approaches fine-tune open-source foundation models (Bai et al., 2025b; Chen et al., 2024) to obtain more proficient GUI agents. Through SFT on large-scale and multi-level datasets, the fine-tuned agents demonstrate superior grounding and navigation capabilities across diverse platforms (Lin et al., 2025; Cheng et al., 2024; Wu et al., 2024b). More recently, RFT, combining rule-based rewards with policy updates, is further incorporated to enhance agents' reasoning and generalization capabilities (Liu et al., 2025b; Lu et al., 2025b; Gu et al., 2025a). Our work also employs a two-stage paradigm, incorporating SFT and RFT, but proposes a novel history representation mechanism that differs from existing methods.

3 CMGUI Dataset

In this section, we introduce the CMGUI dataset. Fig. 2 illustrates the collection pipeline.

3.1 Grounding Data Collection

The grounding data is designed to equip agents with the ability to localize and understand specific UI elements based on functional descriptions. The collection pipeline contains the following three stages.

Raw Data Acquisition. To populate the grounding data at scale, we perform the random walk strategy (Wu et al., 2024a) to explore various system surfaces and apps. During this autonomous exploration, the executed action and the device state, comprising the screenshot and the page structure data, are captured at every step. Here, the page structure data provides a structured, hierarchical representation of the UI elements on the mobile

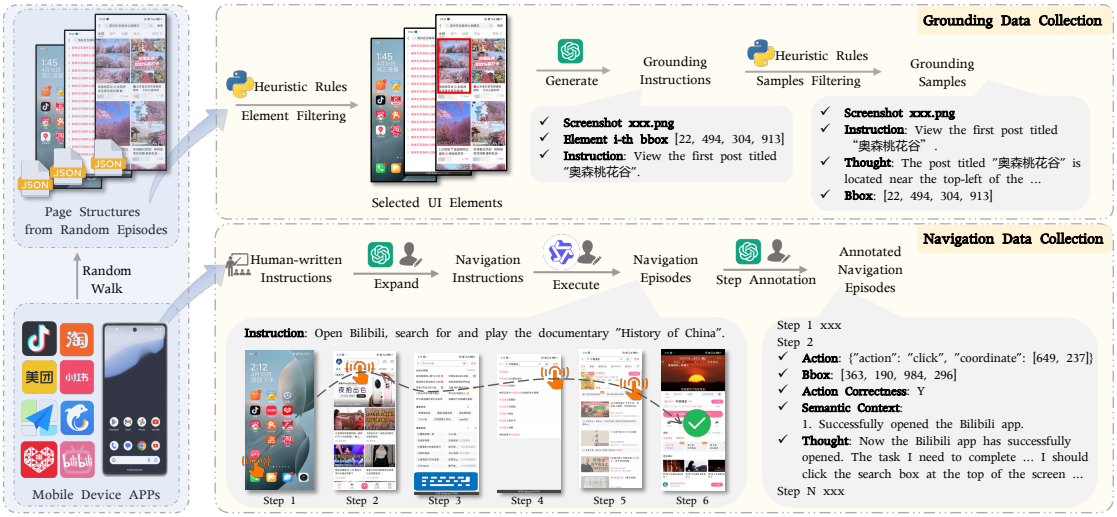


Figure 2: Collection pipeline of CMGUI dataset. Grounding data: UI elements are selected from random walk episodes, with MLLMs generating instructions. Navigation data: episodes are collected using a hybrid human-agent strategy, then humans annotate action correctness and bounding boxes, while MLLMs annotate semantic context and thoughts.

screen, including bounding boxes and descriptive attributes for all potentially interactive UI elements.

UI Element Filtering. To ensure unambiguous and intent-aligned instructions, we implemented a rigorous pipeline for filtering UI elements. Elements are filtered based on geometric (e.g., area $< 6{,}000$ pixels, aspect ratio > 13.5, or occupying $>15\%$ of screen) and visual (near-uniform color) heuristics. Only leaf nodes in the page hierarchy are retained to enforce semantic atomicity. Spatial redundancy is controlled via reservoir sampling, where previously seen elements are downsampled to a 5% selection probability.

Instruction Generation. For each retained element, we overlay a visible marker on the screenshot and use GPT-4 to generate an imperative and unambiguous grounding instruction, along with the rationale of the instruction. Then, a quality check is performed to filter out samples where the generated instructions are shorter than four words or their rationale is fewer than ten words. Finally, we obtain 18,887 high-quality grounding samples, encompassing eight widely used applications such as Taobao and Rednote.

3.2 Navigation Data Collection

The navigation data targets the multi-step planning ability of GUI agents, constructed through a hybrid human-agent collaboration strategy.

Action Space. Our action space contains `CLICK`, `SWIPE`, `TYPE`, `SYSTEM_BUTTON`, `WAIT`, and `TERMINATE`. Details are provided in the supplementary material.

Instruction Generation. We construct high-level instructions through a hybrid human-LLM workflow. First, we prompt GPT-4 with app-specific functional descriptions to generate initial instruction candidates. Then, these candidates are rigorously reviewed and refined by human annotators to ensure feasibility, clarity, and realism. To increase linguistic diversity, we further apply paraphrasing and contextual expansion using GPT-4, resulting in a rich and varied instruction pool.

Hybrid Instruction Execution. To reduce the cost of human demonstration, we employ a hybrid human-agent collaboration strategy to execute instructions. First, a subset of instructions is executed manually by annotators to generate successful navigation episodes with high-quality ground-truth. Subsequently, the annotated episodes are used to train an auxiliary agent by fine-tuning a compact MLLM (Qwen-2.5-VL-3B). It is then deployed to scale data collection through agent-driven execution of the remaining instructions, limited by a maximum of 30 steps per instruction. Here, the auxiliary agent executes each instruction multiple times to increase the diversity of collected episodes.

Episode Annotation. We collect episodes executed by both humans and agents, followed by a step-by-step annotation process. Starting from the first step of each episode, annotators verify the correctness of actions until either termination or the first error is encountered. For each confirmed correct click action, annotators provide a precise bounding box. At the first incorrect step, the correct action and its bounding box are annotated. Notably,

Figure 3: a) Samples from the GUIOdyssey dataset, where SAM2 is used to segment UI elements to obtain bounding boxes, inevitably leading to noise and errors (e.g., the red box is incorrectly sized). b) Samples from our CMGUI data. We employ well-trained annotators to conduct box-by-box annotation and implement a double-check mechanism to ensure precise bounding boxes. c) For our CMGUI-Bench, we conduct step-by-step and box-by-box annotation and consider all reasonable actions at each step, producing multi-choice action annotation.

our human-annotated bounding boxes demonstrate higher precision compared to those generated by autonomous approaches (e.g., SAM2 (Ravi et al., 2024) in GUIOdyssey (Lu et al., 2025a)). Representative examples are illustrated in Fig. 3 (a)-(b). After human verification and correction, we use GPT-4o (Hurst et al., 2024) to augment the annotated steps with two additional fields: (1) semantic context, summarizing critical information from previous steps, and (2) thought, describing the reasoning behind the action. Furthermore, for instructions with a low completion rate, a fallback human execution phase is employed.

To scale data collection efficiently, we adopted a *data flywheel* strategy: in each iteration, after collecting and annotating a new batch of trajectories, we retrain the auxiliary agent, leveraging all previously accumulated data. The detailed training methodology is presented in Sec. 4. In total, we collect 29,711 trajectories comprising 192,666 steps, of which 121,265 steps are annotated. We train exclusively on annotated data; in particular, any step data following the first erroneous action is discarded from the training corpus. The training data covers 44 unique mobile applications across key functional domains such as e-commerce (e.g., Pinduoduo, JD), social media (e.g., Zhihu, Bilibili), and local services (e.g., Meituan, Ele.me).

3.3 CMGUI Benchmark

To evaluate the agent's navigation capability in Chinese mobile applications, we introduce CMGUI-Bench from our curated navigation corpus. The benchmark includes only successful episodes with reliable ground truth. To ensure diversity, we filter episodes using a programming-distance criterion: each instruction must be at least six units distant from others, promoting broad coverage of user intents and interaction patterns while avoiding near-duplicates.

CMGUI-Bench comprises 390 complete episodes and 2,574 steps, with an average episode length of 6.6, spanning 44 widely used apps in the Chinese ecosystem. During curation, we observe that many steps admit multiple valid actions due to interface affordances and app-specific flows (Chen et al., 2025b). Typical cases include submitting a query by either tapping a search button or selecting a suggestion, and reaching a target page via either clicking a tag or swiping horizontally. Some examples are shown in Fig. 3 (c). We therefore recheck all steps and annotate alternative actions to accommodate diverse GUI manipulations. Evaluation focuses on four actions: click, type, swipe, and terminate. A click is correct if it lands within the target bounding box; typed content must match the ground truth; and swipes must align in the direction of the annotated intent.

4 Method: SecAgent

4.1 Formulation

We formalize the agent as a parameterized policy that maps instructions, context, and environment state to actions, following the ReAct paradigm (Yao et al., 2022). Let I denote the instruction, X_t the screenshot at time t, and A_t the action. The baseline agent (Hong et al., 2024) predicts $P(A_t \mid I, X_t)$. Incorporating historical screenshots and actions (Lu et al., 2025a; Wang et al., 2025) yields:

$$P(A_t \mid I, X_t, \{X_{t-i}, A_{t-i}\}_{i=1}^{N}) \qquad (1)$$

where N denotes the history window size. The comparison of existing methods is illustrated in Fig. 4 (Left).

4.2 Architecture

The completion of GUI navigation tasks requires a precise sequence of multi-step actions, whose

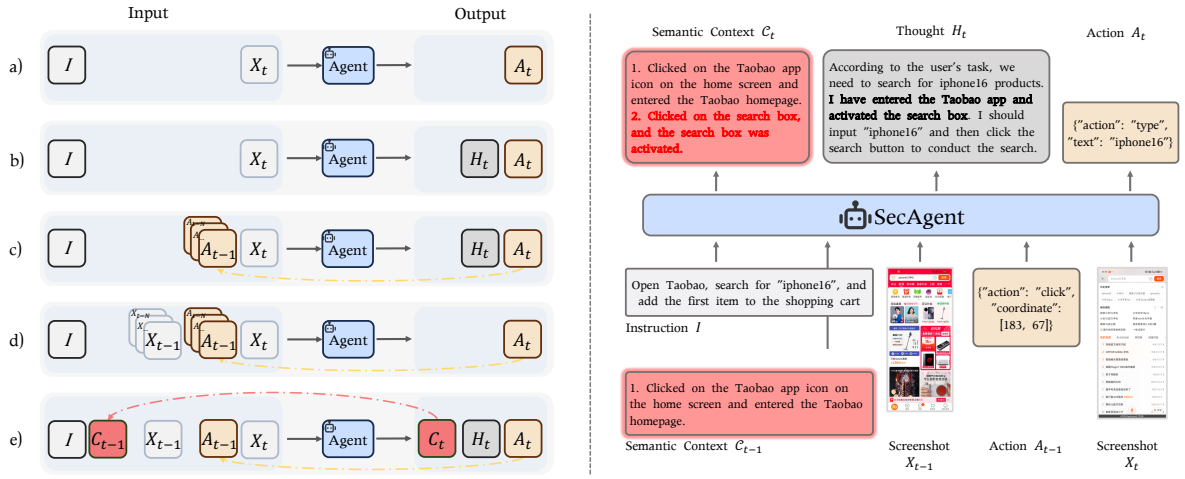


Figure 4: Comparison of different methods. **Left.** a) Baseline: take the instruction and the current screenshot as input and directly output the action. b) AgentCPM (Zhang et al., 2025c) and UI-R1 (Lu et al., 2025b) expect the model to have reasoning capabilities and thus introduce thought before output action. c) UI-Venus (Qin et al., 2025), GUI-R1 (Luo et al., 2025), and OS-Atlas (Wu et al., 2024b) add history actions to the input. d) OdysseyAgent (Lu et al., 2025a) uses both history actions and screenshots in the input. To ensure optimal results, it uses a large number of historical screenshots, for example, $N = 5$. e) Our SecAgent records history information by maintaining a concise and natural language-based summary. Thanks to this, SecAgent can achieve good results using only a historical image and action. **Right.** A demonstration of SecAgent.

correctness critically depends on the *historical information* (e.g., previous screenshots and actions). For instance, in GUI search tasks, clicking a search result depends on both the prior query and the history of visited pages and clicked links. However, incorporating all raw historical information incurs an efficiency challenge (Jiang and Huang, 2025). An action description requires only a handful of tokens, whereas an MLLM-embedded screenshot occupies hundreds of tokens. Stacking all the raw historical information as shown in Eq. (1) leads to substantial *computational overhead*, resulting in increased training costs and detrimental real-time inference latency.

To reduce computational overhead from historical information while preserving essential context, we propose the **Semantic Context Mechanism**, compared with existing methods in Fig. 4 (Left). At each step t, we augment the input with a textual context C_t summarizing critical information from previous steps, and a thought H_t providing the rationale for action A_t. The semantic context is a concise natural language summary recording key operations from previous steps (e.g., clicked labels, typed inputs, searched outcomes), efficiently carrying essential history without processing all past screenshots. With this design, good performance can be achieved using only the last screenshot and action as historical information, i.e., $N = 1$ in Eq. (1). Additionally, we adopt CoT to enhance the agent's reasoning and planning capabilities. Formally, SecAgent predicts the next GUI manipulation as:

$$P(C_t, H_t, A_t \mid I, X_t, C_{t-1}, X_{t-1}, A_{t-1}) \quad (2)$$

SecAgent receives instruction I, current screenshot X_t, previous semantic context C_{t-1}, screenshot X_{t-1}, and action A_{t-1}, then autoregressively generates the updated semantic context C_t, thought H_t, and action A_t. An illustrative example is presented in Fig. 4 (Right).

The semantic context offers three advantages: (1) it requires only single-step history, yielding superior efficiency; (2) natural language representation better leverages the LLM's pre-trained world knowledge; (3) it is more informative than raw parameterized actions (e.g., clicked coordinates), which lack semantic interpretability without their associated screenshots.

4.3 Training

To enhance the perception and reasoning capabilities of SecAgent, we employ a two-stage training framework comprising supervised and reinforcement fine-tuning, as illustrated in Fig. 5.

Stage 1: Supervised Fine-Tuning (SFT). We observe that grounding and navigation tasks inherently require different action output formats:

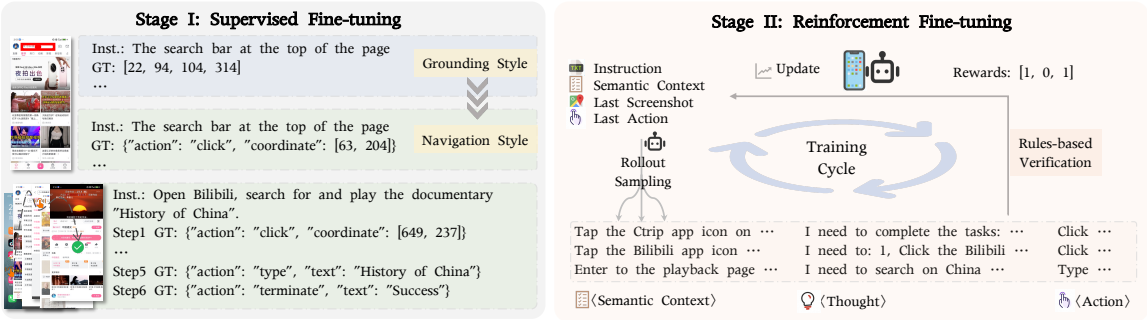


Figure 5: The training framework of SecAgent. We observe that the original grounding and navigation datasets exhibit inherent biases. To mitigate this issue, we do not directly utilize the grounding data for SFT training as in prior work (Zhang et al., 2025c,a). Instead, we transform the grounding annotations into navigation-compatible format by treating the center point of each bounding box as the target click coordinate.

Model	Step Acc.	Task Acc.	Action Acc.			
			Click Acc.	Type Acc.	Swipe Acc.	Terminate Acc.
Closed-source Models						
GPT-4o	48.4	0.0	43.5	68.3	63.7	62.1
Claude-3-Sonnet	81.4	25.8	91.1	30.0	81.3	53.4
Open-source Models (7B and 8B)						
Qwen2.5-VL-7B	83.8	35.6	89.0	75.9	4.4	79.0
UI-TARS-7B	63.8	2.5	80.2	11.1	3.3	8.3
OS-Atlas-7B	69.6	10.0	77.0	76.8	5.5	32.1
GUI-R1-7B	70.5	7.9	84.2	45.5	5.5	8.7
UI-Venus-Navi-7B	89.6	53.1	92.6	91.5	45.0	81.2
AgentCPM-GUI-8B	84.9	39.0	85.8	93.8	54.9	81.2
Qwen3-VL-8B	91.1	59.7	92.6	82.6	69.2	94.6
Open-source Models (3B and 4B)						
Qwen2.5-VL-3B	78.8	25.6	83.2	66.0	5.5	81.9
UI-R1-3B	56.4	1.0	71.2	7.6	5.5	6.5
GUI-R1-3B	45.2	0.5	57.1	8.0	2.2	4.3
Qwen3-VL-4B	90.1	57.4	91.2	84.8	65.9	94.6
SecAgent-3B (Ours)	**96.4**	**80.0**	**96.1**	**99.1**	**87.6**	**99.2**

Table 1: Comparison of various models on the CMGUI benchmark.

grounding tasks require coordinate pairs for bounding boxes, whereas navigation tasks require only a single click coordinate. To address this inconsistency, we deviate from prior work (Zhang et al., 2025c,a) that directly utilizes grounding data for SFT training without format alignment. Instead, we transform the grounding annotations into a navigation-compatible format by computing each bounding box's center point as the click coordinate. This transformation unifies the action space, facilitating better knowledge transfer and generalization between grounding and navigation tasks. During the SFT stage, we train on both transformed grounding and navigation data. The model is optimized to output a structured triplet consisting of semantic context, thought, and action, using the next-token prediction loss. This stage enhances the agent's ability to recognize elements and enables it to imitate human-like decision-making processes.

Stage 2: Reinforcement Fine-Tuning (RFT). Following the SFT stage, we further enhance the agent's reasoning capabilities and generalization performance through reinforcement learning. In the RFT phase, the agent receives a reward for each generated output, and iteratively updates its policy (i.e., model parameters) to maximize the expected reward using the Group Relative Policy Optimization (GRPO) algorithm (Shao et al., 2024). We adopt a rule-based reward comprising format reward and action reward. The format reward encourages structural correctness by assigning a value of 0.5 when the agent successfully generates the required triplet format (semantic context, thought, action), and 0 otherwise. The action reward evaluates functional correctness, providing a reward of 1 for a correct action and 0 otherwise.

Models	AndroidControl	GUIOdyssey
Closed-source Models		
GPT-4o	20.8*	29.3
Claude-3-Sonnet	12.5*	46.5
Open-source Models (7B and 8B)		
Qwen2.5-VL-7B	62.9*	59.6
UI-TARS-7B	74.4*	37.5
OS-Atlas-7B	71.2†	65.2
GUI-R1-7B	51.7‡	41.9
UI-Venus-Navi-7B	<u>76.1</u>†	71.1
AgentCPM-GUI-8B	69.2*	<u>79.2</u>
Qwen3-VL-8B	46.4	53.8
Open-source Models (3B and 4B)		
Qwen2.5-VL-3B	60.1	49.3
UI-R1-3B	45.4‡	32.7
GUI-R1-3B	46.6‡	34.6
Qwen3-VL-4B	47.5	53.4
SecAgent-3B (Ours)	**69.5**	**74.3**

Table 2: Performance comparison on two widely used English benchmarks. We report the step accuracies under high-level instructions. Bold text indicates the best score among 3B and 4B models, and underlined text represents the best score among all baseline models. Superscripts *, † and ‡ indicate results from AgentCPM-GUI, UI-Venus, GUI-R1, respectively.

5 Experiments

5.1 Experimental Setup

Implementation Details. We employ Qwen2.5-VL-3B (Bai et al., 2025b) as the base model to conduct the two-stage training. We adopt LoRA tuning for the SFT stage, and full-parameter tuning for the RFT stage. In addition to our CMGUI dataset, we incorporate data from AndroidControl (Li et al., 2024) and GUIOdyssey (Lu et al., 2025a) datasets to train our agent. More training details are provided in the supplementary material.

Baselines. We consider different types of models for comparison: (1) Closed-source Models: GPT-4o (Hurst et al., 2024) and Claude-3-Sonnet (Anthropic, 2024); (2) General Open-source Models: Qwen2.5-VL (Bai et al., 2025b) and Qwen3-VL (Bai et al., 2025a); (3) GUI-specific SFT Models: OS-Atlas (Wu et al., 2024b) and UI-TARS (Qin et al., 2025); (4) GUI-specific RFT Models: AgentCPM-GUI (Zhang et al., 2025c), UI-Venus-Navi (Qin et al., 2025), UI-R1 (Lu et al., 2025b), and GUI-R1 (Luo et al., 2025).

Metrics. We mainly use two metrics to evaluate the navigation performance of agents: **Step Accuracy (SA)**, defined as the proportion of correct actions, and **Task Accuracy (TA)**, defined as the proportion of episodes where the actions are all correct.

Settings	SA	TA	ITC	TTFT ↓	TPS ↑
N=0	85.3	36.2	1537	0.07	145
N=1	94.8	72.8	2239	0.11	140
N=2	95.1	73.8	2760	0.14	134
N=5	95.5	74.1	3642	0.22	122
N=5 w/o SC	95.0	73.0	3507	0.22	124
N=1 w/o SC	90.6	56.4	2171	0.11	142
N=1 w/o thought	94.0	68.9	2207	0.11	141

Table 3: Performance and efficiency comparison across different history window sizes (N), and the ablation of Semantic Context (SC) and thought. For efficiency evaluation, we report the Input Token Count (ITC), Time to First Token (TTFT), Tokens per Second (TPS), and training hours, following (Lu et al., 2025a). ITC includes both text and vision tokens.

5.2 Benchmark Results

CMGUI-Bench. Tab. 1 shows the comprehensive evaluation of representative GUI agents on our newly constructed CMGUI-Bench. The results unequivocally demonstrate the superior performance of our proposed SecAgent. SecAgent achieves a remarkable SA of **96.4**% and a dominant TA of **80.0**%, significantly outperforming all other closed-source and open-source baselines, regardless of their parameter size. Analyzing the action-specific metrics reveals the source of this performance gain: SecAgent achieves high accuracy on the frequently used `CLICK` action (**96.1**%) and the challenging `SWIPE` action (**87.6**%). This superior performance validates the high quality of the CMGUI dataset and its effectiveness for training agents in the Chinese mobile ecosystem.

AndroidControl and GUIOdyssey Benchmark. To examine effectiveness beyond the Chinese ecosystem, we evaluate on two English mobile navigation benchmarks: AndroidControl and GUIOdyssey. Note that GUIOdyssey has different evaluation versions; we use the updated set and randomly select 100 trajectories to evaluate all methods. As shown in Tab. 2, SecAgent achieves highly competitive results with SA of **69.5**% on AndroidControl and **74.3**% on GUIOdyssey. Our 3B model outperforms other open-source models in its size category and demonstrates performance comparable to larger models, e.g., exceeding AgentCPM-GUI-8B on AndroidControl and UI-Venus-Navi-7B on GUIOdyssey. This highlights that semantic context and the two-stage training scheme yield efficient and competitive navigation capabilities.

Training Data	Training Method	In-Domain Apps				Out-of-Domain Apps			
		Taobao SA / TA	Kuaishou SA / TA	Fliggy SA / TA	JD SA / TA	Rednote SA / TA	Bilibili SA / TA	Baidu Maps SA / TA	Pinduoduo SA / TA
Partial	SFT	91.2 / 56.9	89.1 / 44.8	91.9 / 58.3	91.9 / 61.1	91.7 / 54.5	88.4 / 45.8	86.6 / 41.7	94.4 / 66.7
	SFT + RFT	94.5 / 70.8	90.1 / 48.3	96.0 / 79.2	93.5 / 61.1	96.4 / 80.0	90.2 / 45.8	91.3 / 62.5	97.6 / 83.3
Full	SFT	94.5 / 70.8	91.2 / 58.6	96.1 / 79.2	92.9 / 61.1	97.3 / 87.3	96.3 / 79.2	**96.9 / 79.2**	**97.7** / 83.3
	SFT + RFT	**95.5 / 73.8**	**93.2 / 75.9**	**96.6 / 79.2**	**97.6 / 83.3**	**98.4 / 89.1**	**97.6 / 91.7**	96.6 / **79.2**	97.6 / **88.9**

Table 4: Performance comparison on in-domain and out-of-domain apps. The 'Partial' and 'Full' training data refer to the entire CMGUI dataset, excluding and including the four out-of-domain apps, respectively. For presentation conciseness, the results of four representative in-domain apps are presented.

Training Data	Training Method	SA	TA
Navigation	SFT	93.7	65.8
Grounding + Navigation	SFT	94.4	69.2
Gr2Nav + Navigation	SFT	94.8	72.8
Gr2Nav + Navigation	RFT *	95.2	75.9
Gr2Nav + Navigation	SFT + RFT	**96.4**	**80.0**

Table 5: Comparison of different training data and training methods. 'Grounding' denotes the original grounding data, and 'Gr2Nav' denotes the transformed grounding data with navigation-compatible format. *Direct RFT on the base model fails to converge; thus we apply a SFT with 1k navigation step data before RFT.

5.3 Ablation Studies

Performance and Efficiency Comparison. Tab. 3 examines the impact of history window size N and key components (i.e., semantic context and thought) on performance and efficiency. Expanding from $N = 0$ to $N = 1$ substantially improves both metrics: SA increases by 9.5% while TA nearly doubles, confirming that history is crucial for task completion. However, further expansion yields diminishing returns: $N = 1$ to $N = 5$ improves SA and TA by less than 2%, while ITC grows by 62.7%, TTFT doubles, and TPS declines by 12.9%. Thus, $N = 1$ is adopted as SecAgent's default for its favorable performance-efficiency balance. Moreover, $N = 1$ *with SC* matches $N = 5$ *without SC* at significantly higher efficiency, validating our SC mechanism. Ablation at $N = 1$ shows both SC and thought contribute to performance, with removing SC causing substantial degradation.

Training Data and Methods. Tab. 5 evaluates the impact of different training data compositions and methodologies. First, incorporating grounding data improves SFT-trained agents, demonstrating the quality of our grounding data in enhancing navigational capabilities. Notably, Gr2Nav data provides greater improvement than the original, validating our grounding data transformation approach that enables consistent alignment between grounding-style and navigation-style data. Second, the comparison between SFT and SFT-initialized RFT demonstrates the potential of RFT, which improves TA by 3.1%. Finally, integrating RFT following complete SFT yields the best performance, improving SA by 1.6% and TA by 7.2% over the SFT-only agent, validating our two-stage learning paradigm for complex mobile GUI navigation.

In-Domain (ID) and Out-of-Domain (OOD) Performance. Tab. 4 presents ID and OOD performance under different training data and methods. First, apps share transferable knowledge: RFT-augmented training on partial data achieves reasonable OOD performance, such as 97.6% SA on Rednote. Moreover, OOD data improves ID performance, with Kuaishou TA increasing by 13.8% under SFT. Second, apps possess domain-specific knowledge requiring targeted training. Adding OOD data yields substantial improvements: Bilibili TA increases by 33.4% and BaiduMaps by 37.5% under SFT, attributed to app-specific operations that cannot be inferred from general knowledge alone. Third, with partial data, RFT-augmented training outperforms SFT-only on all OOD apps, demonstrating superior generalization.

6 Conclusion

In this work, we introduce SecAgent, an efficient mobile GUI agent that addresses two critical challenges: the scarcity of multilingual datasets and inefficient history representation. We contribute a large-scale Chinese mobile GUI dataset and a high-quality navigation benchmark with multi-choice action annotations. We also propose a semantic context mechanism that enhances computational efficiency while preserving important historical information. Empirical results demonstrate that SecAgent outperforms 3B baselines and achieves performance comparable to 7B-8B models.

7 Limitations

Our work has the following limitations: First, CMGUI focuses exclusively on the Chinese mobile ecosystem. While this addresses the scarcity of Chinese GUI data, the dataset does not cover other non-English languages. Extending our data collection pipeline to additional languages and cultural contexts remains future work. Second, we only train and evaluate SecAgent at the 3B parameter scale. Although SecAgent-3B demonstrates competitive performance against 7B-8B models, training larger models (e.g., 7B) with our semantic context mechanism and two-stage framework could potentially yield further performance gains, which we leave for future exploration. Third, our evaluation is conducted in offline settings using pre-collected trajectories rather than live device interaction. Real-world deployment may introduce additional challenges such as network latency and unexpected pop-ups that are not captured in our current benchmark.

8 Ethical considerations

Our work presents a mobile GUI agent and a Chinese mobile GUI dataset. We acknowledge the following ethical considerations. First, automated GUI agents could potentially be misused for malicious purposes, such as unauthorized operations on mobile devices. To mitigate this risk, all tasks in our dataset are limited to benign, everyday usage scenarios, and we will release the dataset under a responsible use license. Second, during data collection, we took careful measures to avoid capturing personally identifiable information. All screenshots were collected on test accounts, and sensitive information such as usernames and phone numbers was manually reviewed and removed during the annotation process. Third, all human annotators involved in data collection and verification were fairly compensated and informed about the purpose of the project.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2024. The Claude 3 model family: Opus, sonnet, haiku.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, and 1 others. 2025a. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025b. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guangyi Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. 2025. A3: Android agent arena for mobile gui agents. *arXiv preprint arXiv:2501.01149*.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye Hao, Jun Wang, and Kun Shao. 2025a. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. *arXiv preprint arXiv:2410.15164*.

Liangyu Chen, Hanzhang Zhou, Chenglin Cai, Jianan Zhang, Panrong Tong, Quyu Kong, Xu Zhang, Chen Liu, Yuqi Liu, Wenxuan Wang, and 1 others. 2025b. Ui-ins: Enhancing gui grounding with multi-perspective instruction-as-reasoning. *arXiv preprint arXiv:2510.20286*.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, and 1 others. 2024. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 24185–24198.

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. *arXiv preprint arXiv:2401.10935*.

Jihao Gu, Qihang Ai, Yingyao Wang, Pi Bu, Jingxuan Xing, Zekun Zhu, Wei Jiang, Ziming Wang, Yingxiu Zhao, Ming-Liang Zhang, and 1 others. 2025a. Mobile-R1: Towards interactive reinforcement learning for vlm-based mobile agent via task-level rewards. *arXiv preprint arXiv:2506.20332*.

Zhangxuan Gu, Zhengwen Zeng, Zhenyu Xu, Xingran Zhou, Shuheng Shen, Yunfei Liu, Beitong Zhou, Changhua Meng, Tianyu Xia, Weizhi Chen, and 1 others. 2025b. Ui-Venus technical report: Building high-performance ui agents with rft. *arXiv preprint arXiv:2508.10833*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14281–14290.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Yangqin Jiang and Chao Huang. 2025. Lightagent: Mobile agentic foundation models. *arXiv preprint arXiv:2510.22009*.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154.

Yang Li, Yuchen Liu, Haoyu Lu, Zhiqiang Xia, Hongzhen Wang, Kaiyang Han, Changpeng Yang, Jinyang Wu, Jiaming Xu, Runyu Shi, and 1 others. 2026. Gui-ceval: A hierarchical and comprehensive chinese benchmark for mobile gui agents. *arXiv preprint arXiv:2603.15039*.

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 *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508.

Guangyi Liu, Pengxiang Zhao, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, Hao Wang, and 1 others. 2025a. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025b. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. 2025a. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22404–22414.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. 2025b. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv: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. *arXiv preprint arXiv:2504.10458*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, and 1 others. 2024. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, and 1 others. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Liujian Tang, Shaokang Dong, Yijia Huang, Minqi Xiang, Hongtao Ruan, Bin Wang, Shuo Li, Zhiheng Xi, Zhihui Cao, Hailiang Pang, Heng Kong, He Yang, Mingxu Chai, Zhilin Gao, Xingyu Liu, Yingnan Fu, Jiaming Liu, Xuanjing Huang, Yu-Gang Jiang, and 5 others. 2025. Magicgui: A foundational mobile gui agent with scalable data pipeline and reinforcement fine-tuning. *arXiv preprint arXiv:2508.03700*.

Bryan Wang, Gang Li, and Yang Li. 2023. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI conference on human factors in computing systems*, pages 1–17.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th annual international conference on Mobile computing and networking*, pages 543–557.

Qinzhuo Wu, Weikai Xu, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, and Shuo Shang. 2024a. Mobilevlm: A vision-language model for better intra-and inter-ui understanding. *arXiv preprint arXiv:2409.14818*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024b. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Weikai Xu, Zhizheng Jiang, Yuxuan Liu, Pengzhi Gao, Wei Liu, Jian Luan, Yuanchun Li, Yunxin Liu, Bin Wang, and Bo An. 2025. Mobile-bench-v2: A more realistic and comprehensive benchmark for vlm-based mobile agents. *arXiv preprint arXiv:2505.11891*.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2024. Androidlab: Training and systematic benchmarking of android autonomous agents. *arXiv preprint arXiv:2410.24024*.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, and 1 others. 2023. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Zhixiong Zeng, Jing Huang, Liming Zheng, Wenkang Han, Yufeng Zhong, Lei Chen, Longrong Yang, Yingjie Chu, Yuzhi He, and Lin Ma. 2025. Uitron: Foundational gui agent with advanced perception and planning. *arXiv preprint arXiv:2508.21767*.

Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. 2025a. Tongui: Building generalized gui agents by learning from multimodal web tutorials. *arXiv preprint arXiv:2504.12679*.

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

Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, and 1 others. 2025c. Agentcpm-gui: Building mobile-use agents with reinforcement fine-tuning. *arXiv preprint arXiv:2506.01391*.

A Details of CMGUI

A.1 Action Space

The action space of CMGUI comprises the following actions: `CLICK`, `SWIPE`, `TYPE`, `SYSTEM_BUTTON`, `WAIT`, and `TERMINATE`. The description and format of each action are presented in Tab. 7.

A.2 App Distribution

Tab. 8 presents the detailed breakdown of trajectory counts for each application in CMGUI. To ensure comprehensive coverage of real-world mobile usage scenarios, CMGUI comprises 44 widely-used Chinese mobile applications across diverse functional categories, including social media, e-commerce, entertainment, and productivity.

A.3 Examples

Fig. 6 presents examples of grounding data in CMGUI. To improve the utilization of screenshot resources, we generate multiple grounding data from a single screenshot. To ensure consistency between grounding and navigation data in output format, we transform the grounding data into a single-step navigation data, with the ground-truth click coordinates corresponding to the bounding box center.

Fig. 7 illustrates three evaluation trajectories with multi-choice annotations in CMGUI-Bench. Given the multi-feasible nature of GUI manipulations, a single step may have multiple correct actions that lead to the same goal. To address this, we manually examined each step and annotated all valid actions.

B Experiment Details

B.1 Prompts

Fig. 8 shows the prompt used by SecAgent. The prompt includes the action space specification, user instruction, previous semantic context, current screenshot, and previous action as inputs. The agent is required to output the updated semantic context, its thought, and the action to take. For the baseline models, we strictly follow their respective official guidelines and prompt templates when available.

The prompt used to generate grounding instructions is shown in Fig. 9. In the prompt, we specify the instruction format, require the agent to provide explanations for its generated instructions, and provide descriptions of common UI elements and their meanings.

B.2 RFT Algorithm

Our RFT algorithm is built on the GRPO algorithm (Shao et al., 2024), which stabilizes and accelerates training by estimating the advantage of actions using group-normalized rewards without acquiring a

reward model or critic model. We adopt two modifications to the GRPO algorithm: the token-level loss and different clipping ratios. Given a query q, the current policy $\pi_{\theta_{\text{old}}}$ samples G responses $\{o_i\}_{i=1}^{G}$. Each response is assigned a scalar reward $\{R_i\}_{i=1}^{G}$. Rewards are normalised within the group to produce variance-reduced advantages:

$$\hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_i\}_{i=1}^{G})}{\text{std}(\{R_i\}_{i=1}^{G})} \tag{3}$$

The policy is then updated using a clipped objective with KL divergence penalty:

$$\begin{aligned} J_{\text{GRPO}}(\theta) = & \frac{1}{\sum_{i=1}^{G} |o_i|} \sum_{i=1}^{G} \sum_{t=1}^{|o_i|} \Big\{ \min \Big(r_{i,t}(\theta) \hat{A}_{i,t}, \\ & \text{clip} \left(r_{i,t}(\theta), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}} \right) \hat{A}_{i,t} \Big) \\ & - \beta D_{KL} \left[\pi_\theta \| \pi_{\text{ref}} \right] \Big\} \end{aligned} \tag{4}$$

where ϵ_{low} and ϵ_{high} control the clipping range, β constrains policy divergence from the reference model π_{ref}, and $r_{i,t}(\theta)$ is the probability ratio, calculated as:

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})} \tag{5}$$

B.3 Training Parameters

The hyperparameters used for training SecAgent during SFT and RFT are shown in Tab. 6. We use Qwen2.5-VL-3B as our base model, and set `min_pixels` and `max_pixels` as $200, 704$ and $501, 760$, respectively.

Hyperparameter	Value
Supervised Fine-tuning	
Epoch	6
Batch Size	2
Learning Rate	2e-5
LoRA Rank	8
LoRA Alpha	16
Reinforcement Fine-tuning	
Epoch	2
Batch Size	8
Learning Rate	1e-6
KL Penalty	0.04
Number of Rollouts	16
Clip Range $\epsilon_{\text{low}}, \epsilon_{\text{high}}$	0.2, 0.28

Table 6: The setting of training hyperparameters.

Action	Description	Format
CLICK	Click the point on the screen with coordinate (x, y).	{"name": "mobile_use", "arguments": {"action": "click", "coordinate": [x, y]}}
SWIPE	Swipe from the starting point with coordinate (x, y) to the end point with coordinates2 (x2, y2).	{"name": "mobile_use", "arguments": {"action": "swipe", "coordinate": [x, y], "coordinate2": [x2, y2]}}
TYPE	Input the specified text into the activated input box.	{"name": "mobile_use", "arguments": {"action": "type", "text": "specified_text"}}
SYSTEM_BUTTON	Press the specified system button (e.g., "Back", "Home", "Menu", "Enter").	{"name": "mobile_use", "arguments": {"action": "system_button", "button": "specified_button"}}
WAIT	Wait specified seconds for the change to happen.	{"name": "mobile_use", "arguments": {"action": "wait", "time": "specified_seconds"}}
TERMINATE	Terminate the current task and report its completion status ("success" or "failure").	{"name": "mobile_use", "arguments": {"action": "terminate", "status": "success"}}

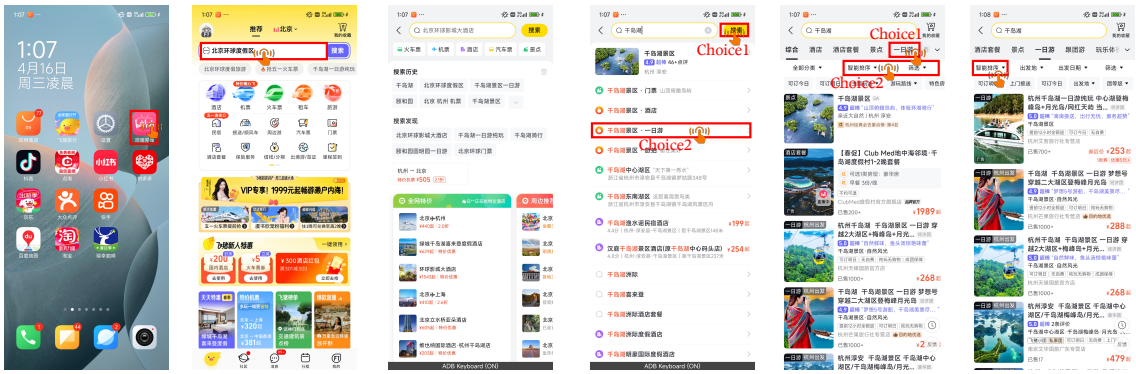
Table 7: The action space of CMGUI.

#ID	App Name	#Traj.	#ID	App Name	#Traj.	#ID	App Name	#Traj.	#ID	App Name	#Traj.
1	Taobao	4,722	2	Kuaishou	1,829	3	Rednote	1,341	4	Baidu Maps	1,213
5	Fliggy	1,184	6	Bilibili	1,157	7	WeChat	1145	8	JD	1,023
9	Pinduoduo	861	10	Weather	817	11	Meituan	777	12	Douyin	757
13	Calendar	741	14	Poizon	714	15	Aliyun Drive	693	16	Amap	686
17	Ele.me	640	18	Quark Browser	589	19	Toutiao	580	20	Tencent Maps	576
21	Calculator	541	22	Baidu	519	23	Tencent Meeting	499	24	Browser	496
25	Tomato Novel	492	26	Didi Chuxing	447	27	Notes	432	28	WeCom	429
29	Message	426	30	Contacts	401	31	DingTalk	393	32	Feishu	380
33	Baidu Netdisk	355	34	Cainiao	329	35	Ctrip	264	36	WPS Office	163
37	QQ	152	38	WeChat Reading	151	39	Weibo	149	40	Zhihu	130
41	Xianyu	129	42	Alipay	120	43	Zhuanzhuan	117	44	Luckin Coffee	48

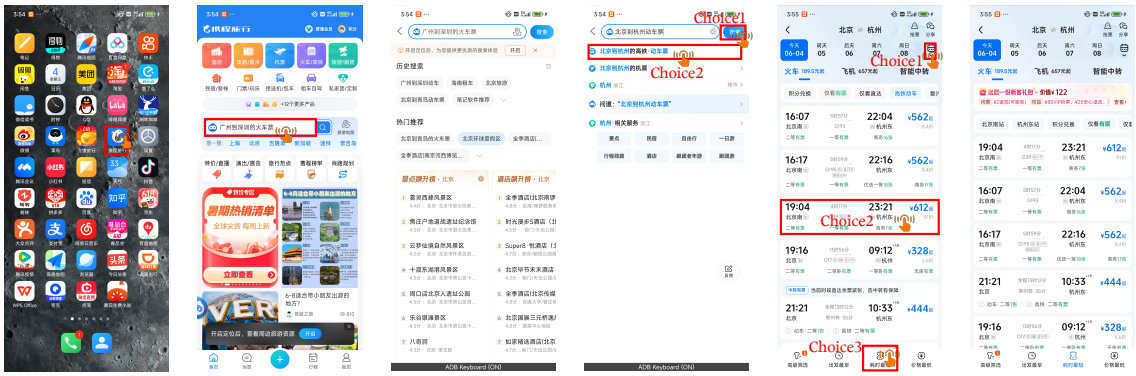
Table 8: Our constructed CMGUI consists of 44 mainstream Chinese apps. The number of trajectories for each app is shown in the table.

Figure 6: Examples of grounding data in CMGUI. For each screenshot, multiple grounding data can be synthesized and then converted into multiple single-step navigation data.

Instruction:打开淘宝，搜索“小米家电洗衣机”，选择一个商品，加入购物车 (Open Taobao, search for “Xiaomi washing machine”, select a product, and add it to your shopping cart.)



Instruction:打开飞猪，搜索千岛湖，筛选一日游，选择智能排序 (Open Fliggy, search for Qiandao Lake, filter by day trips, and select smart sorting.)



Instruction:打开携程旅行，搜索“北京到杭州动车票”，选择耗时最短路线 (Open Ctrip, search for “Beijing to Hangzhou high-speed train tickets”, and select the route with the shortest travel time.)

Figure 7: Example trajectories in CMGUI-Bench. The multi-choice annotations are explicitly drawn on the image.

System Message

You are a mobile GUI agent.

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name_for_human": "mobile_use", "name": "mobile_use", "description": "Use a touchscreen to interact with a mobile device, and take screenshots.\n* This is an interface to a mobile device with touchscreen. You can perform actions like clicking, typing, swiping, etc.\n* 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.\n* The screen's resolution is <width>x<height>.\n* 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:\n* `key`: Perform a key event on the mobile device.\n - This supports adb's `keyevent` syntax.\n - Examples: "volume_up", "volume_down", "power", "camera", "clear".\n* `click`: Click the point on the screen with coordinate (x, y).\n* `long_press`: Press the point on the screen with coordinate (x, y) for specified seconds.\n* `swipe`: Swipe from the starting point with coordinate (x, y) to the end point with coordinates2 (x2, y2).\n* `type`: Input the specified text into the activated input box.\n* `system_button`: Press the system button.\n* `open`: Open an app on the device.\n* `wait`: Wait specified seconds for the change to happen.\n* `terminate`: Terminate the current task and report its completion status.", "enum": ["key", "click", "long_press", "swipe", "type", "system_button", "open", "wait", "terminate"], "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=click`, `action=long_press`, and `action=swipe`.", "type": "array"}, "coordinate2": {"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=swipe`.", "type": "array"}, "text": {"description": "Required only by `action=key`, `action=type`, and `action=open`.", "type": "string"}, "time": {"description": "The seconds to wait. Required only by `action=long_press` and `action=wait`.", "type": "number"}, "button": {"description": "Back means returning to the previous interface, Home means returning to the desktop, Menu means opening the application background menu, and Enter means pressing the enter. Required only by `action=system_button`", "enum": ["Back", "Home", "Menu", "Enter"], "type": "string"}, "status": {"description": "The status of the task. Required only by `action=terminate`.", "type": "string", "enum": ["success", "failure"]}}, "required": ["action"], "type": "object"}, "args_format": "Format the arguments as a JSON object."}}
</tools>

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>

Note:

1, If previous screenshots and actions are available, first summarize what was done in the prior step based on the previous screenshot, the corresponding action, and the current screenshot. Update this summary into a sequential list that records the progress of each step. Output the updated summary within the <summary></summary> tags. If no prior screenshot or action exists, output None.
2, Provide the reasoning process within the <think></think> tags. Note that previously executed actions may not be correct.
3, Output the intended action to be performed within the <tool_call></tool_call> tags.

User

The user instruction: [*user_instruction*]
The previous semantic context: [*previous_semantic_context*]
The previous screenshot: [*previous_screenshot*] (only exists when it is not the first step)
The previous action: [*previous_action*] (only exists when it is not the first step)
The current screenshot: [*current_screenshot*]

Assistant

[*semantic_context*]\n[*thought*]\n[*action*]

Figure 8: The prompt used by SecAgent. We note that the Chinese text has been translated into English for presentation purposes.

You are to act as a mobile application testing agent tasked with generating training data based on the following input:
1.Input: A screenshot of an app interface, where interactive elements are annotated with red and green bounding boxes.
Your task consists of the following steps:
1. Identify interactive elements within the red and green bounding boxes and generate corresponding instructions and reasoning:
  1. If no interactive elements exist within the bounding box, or if multiple interactive elements are present, return an empty response.
  2. If the content inside the bounding box is purely solid-colored (e.g., uniform color without semantic meaning), return an empty response.
  3. Focus exclusively on the content within the red and green bounding boxes; ignore surrounding regions.
2. Generate user action instructions based on the content of the bounding box:
  1. The generated instruction must strictly refer only to the content within the red and green bounding boxes, not adjacent elements.
  2. For a profile avatar enclosed in the bounding box, generate: "Click [User Name]'s avatar".
  3. For text content alone, generate: "Click 'text' text".
  4. Leverage global interface context and structured positional data to infer and produce detailed, natural-language user instructions.
  5. Instructions should follow the format: "[Action verb] + [description of interactive element]". Use diverse action verbs such as "click", "enter", "open", "select", etc.
  6. When multiple identical elements exist in the interface, disambiguate using ordinal indicators (e.g., "the Nth") or distinctive features (e.g., text, icon).
  7. When specific textual content is involved, wrap it in single quotes ' and append a descriptor indicating its type (e.g., post, comment, icon).
  8. Avoid colloquial phrasing; e.g., use "like [content]" instead of "give content a like".
  9. Interpret high-level UI semantics correctly: do not generate "click the magnifying glass symbol"; instead, generate "click the search button" since such icons typically represent search functionality.
  10. For a light gray camera icon, the instruction should relate to photography, camera access, or image upload (e.e., "open the scanner").
  11. For a light gray or white three-line menu icon (hamburger menu), the instruction should reflect access to additional options (e.g., "open more settings").
  12. Ensure the instruction is grammatically correct, unambiguous, and semantically clear.
3. Generate a reasoning process ("thoughts"):
  1. Simulate the cognitive process of an agent executing a user-intended task. Describe how the operation would be understood and completed.
  2. Reasoning may include:
  3. How the target element is identified?
  4. Typical location patterns of such elements?
  5. Relationships between this element and others?
  6. Functional or semantic interpretation of the element?
  7. Do not include pixel coordinates, screen positions, or any numerical spatial references.
  8. Do not explain how the instruction was generated; instead, explain how one would execute it.
4. Output format:
  Return the result in JSON format with two fields: "instructions" and "thoughts".
  1. "instructions": a string containing a single action instruction.
  2. "thoughts": a string describing the reasoning process for executing the instruction.
  3. Strictly adhere to the following JSON schema:
  {
    "instructions": "specific instruction",
    "thoughts": "reasoning process"
    }
  4. Example:
  {
    “instructions”: “Click ‘Login’ button”,
    “thoughts”: “Locate the button labeled ‘Login’ on the interface. Such buttons are typically positioned in the center-right region of the screen and serve as the primary entry point for user authentication. After confirming the button is visible and not obscured, perform the click action."
    }
5. Common UI Semantic Mappings:
  The following table maps common UI elements to their typical functional interpretations. These should guide the generation of semantically accurate instructions.
  Rounded rectangle/circle with a line (possibly): Camera scan area, QR code scanning, scan function
  Light gray camera icon: Take photo, open camera, upload image \n Magnifying glass: Search, find
  House icon: Home screen, main page \n Envelope icon: Messages, email, private messages
  Clock/alarm icon: Alarm, reminder, history \n Bell/speaker icon: Notifications, alerts
  Plus (+) sign: Create, add, publish new content \n Three horizontal lines (hamburger menu): More options, side menu, settings
  Three dots (...): More actions, additional menu \n Gear icon: Settings, system configuration
  Back arrow: Return, go back to previous screen \n Paper airplane: Send, push message
  Speech bubble: Comment, chat, conversation \n Heart icon: Like, favorite, bookmark
  Trash can: Delete, remove \n Eye icon: View, show/hide, privacy-related
  User avatar: Profile, personal center, user-related actions \n Calendar icon: Select date, schedule, event
  Folder/file icon: Directory, category, management \n Shopping cart icon: Shopping, checkout, order
  Bookmark/flag icon: Favorite, tag \n Upload/download arrows: Upload, download, transfer
  Share arrow: Share, forward, export \n Map/location pin: Location, map, find nearby
  Volume icon: Volume control, mute, sound settings \n Refresh symbol: Refresh, update, reload
  Question mark: Help, hint, support \n Lock/unlock icon: Security, privacy, encryption, unlock
  QR code/scanning icon: Scan QR code, payment, identification \n Zoom in/out (+/- or magnifier with +): Zoom in, zoom out, view details
  Scissors/clipboard: Cut, copy, paste \n Pencil/edit icon: Edit, rename, modify
  Document with plus: Create new document, new file \n Lock/unlock icon: Lock operation, authorization, security
  Star symbol: Favorite, mark as important, rate \n Tear-drop/checkmark/x: Selection, selected, read, completed, pending, error
  Cloud with upload/download: Cloud sync, upload to cloud, download locally \n Rotating arrow/bidirectional arrow: Rotate, switch orientation, toggle view (landscape/portrait)
  Sort/filter icon (three vertical lines, with dots/arrows): Sort, filter, change display mode \n Shopping bag: Purchase, place order
  Gift box/red envelope: Give gift, promotion, reward \n Globe/world map icon: Multilingual, international version
  U-shield/key/shield: Authentication, security, protection, permission \n Undo/redo arrow: Undo, redo
  Clock with search icon: Search history \n Book/lecture notes: Learning, reading, tutorial
  Cloud with lightning: Download/update failure, sync warning \n Image + cloud/upload arrow: Upload image/avatar
  Settings panel/slider: Customize, preferences \n Article/news (newspaper/text lines): News, announcement, information
  Play/pause/fast-forward symbols: Play, pause, resume, skip \n Microphone: Voice input, record audio
  Speaker/headphone: Audio output, speaker, mute \n Chain/link: Connect, copy link, hyperlink
  Flame symbol: Popular, trending, hot list \n Flag/target icon: Task, goal, milestone
  Moon/sun: Dark/light mode, eye protection, brightness \n Alarm/clock/sandglass: Timer, alarm, countdown
  Progress ring/circle: Progress, loading, waiting \n Arrow with file (import/export): Import, export
  Bubble with plus / chat with plus: Start new chat, create new conversation \n Info/exclamation mark/alert box: Notice, warning, notification
  Business card: Contact info, share contact \n Crown: Membership, VIP, premium
  Fireworks/firecrackers: Event, celebration, achievement unlocked \n Remote control/device icon: Remote control, hardware pairing

Figure 9: The prompt used to generate grounding instructions. We note that the Chinese text has been translated into English for presentation purposes.