AgentCPM-GUI: Building Mobile-Use Agents with Reinforcement Fine-Tuning

Zhong Zhang[1*], Yaxi Lu[1*], Yikun Fu[1†], Yupeng Huo[2], Shenzhi Yang[2], Yesai Wu[1], Han Si[1†]
Xin Cong[1], Haotian Chen[1], Yankai Lin[2‡], Jie Xie[1], Wei Zhou[1], Wang Xu[1], Yuanheng Zhang[1†]
Zhou Su[3], Zhongwu Zhai[3], Xiaoming Liu[3], Yudong Mei[3], Jianming Xu[3], Hongyan Tian[3]
Chongyi Wang[3], Chi Chen[1], Yuan Yao[1,4], Zhiyuan Liu[1‡], Maosong Sun[1‡]
[1]Tsinghua University [2]Renmin University of China [3]ModelBest Inc. [4]Shanghai Qi Zhi Institute
zhongzhang@tsinghua.edu.cn lyx23@mails.tsinghua.edu.cn

Abstract

Large language model agents have enabled GUI-based automation, particularly for mobile devices. However, deployment remains limited by noisy data, poor generalization, and lack of support for non-English GUIs. In this work, we present AgentCPM-GUI, an 8B-parameter GUI agent built for robust and efficient on-device GUI interaction. Our training pipeline includes grounding-aware pre-training to enhance perception, supervised fine-tuning on high-quality Chinese and English trajectories to imitate human-like actions, and reinforcement fine-tuning with GRPO to improve reasoning capability. AgentCPM-GUI achieves promising performance on five public benchmarks and our proposed Chinese benchmark CAGUI. To facilitate reproducibility and further research, we publicly release all code, model checkpoint, and evaluation data at: https://github.com/OpenBMB/AgentCPM-GUI

1 Introduction

The rapid advancements in Large Language Models (LLMs) and Multimodal Large Models (MLLMs) have catalyzed a new era of autonomous AI agents (Zhao et al., 2023; Wang et al., 2024b). These agents are increasingly capable of understanding complex instructions (Ouyang et al., 2022; Qian et al., 2024), performing multi-step planning (Huang et al., 2024), and interacting with external tools or environments (Qin et al., 2024, 2025a). A critical frontier for deploying these intelligent agents in practical, human-centric applications is enabling them to proficiently operate Graphical User Interfaces (GUIs) (Wang et al., 2024c; Nguyen et al., 2025; Zhang et al., 2025a), particularly within the ubiquitous Android ecosystem, where they serve as the primary interaction layer for a vast array of daily digital tasks. Empowering LLM agents to seamlessly navigate and manipulate these mobile GUIs is essential for transforming them into truly versatile digital assistants capable of automating a wide spectrum of tasks on smartphones, thereby enhancing user productivity and accessibility.

Early GUI agents emerged when Vision-Language Models (VLMs) had limited ability in reliably control GUI widgets. To compensate, researchers augmented model inputs with structured metadata, such as Android view hierarchies and system APIs, and even off-loaded perception and planning to more capable external VLMs (e.g., GPT-4o (Hurst et al., 2024)), thereby improving widget grounding and action execution (Zhang et al., 2025b; Chen et al., 2025a; Chen and Li, 2024; Zheng et al., 2024; Kim et al., 2023; Wang et al., 2024a). Although effective, these hybrid pipelines propagated errors from cross-modal mismatches, incurred round-trip latency, and depended on metadata that many apps do not expose, creating significant challenges for generality and scalability. Recent GUI agents have advanced to resolving interface elements directly from raw pixels, enabling a single end-to-end model to match or even surpass earlier hybrid approaches (Hong et al., 2024; Cheng et al., 2024; Qin et al., 2025b; Xu et al., 2024; Wu et al., 2025; Lin et al., 2025; Zhang and Zhang, 2024). This shift positions purely visual, end-to-end modeling as the most scalable paradigm.

Despite significant progress, current visual GUI agents still face several challenges: **(1) Data quality and scale**. High-quality, fine-grained interaction trajectories that capture realistic user behavior in diverse mobile apps are notoriously difficult to collect at scale. Most publicly available datasets either rely on synthetic generation or emulator-based recordings, both of which can introduce noise and lack semantic diversity. Such imperfect supervision limits the agent's ability to learn precise wid-

[*] Equal contribution.
[†] Internship at Tsinghua University.
[‡] Corresponding authors.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 155–180
November 4-9, 2025

get grounding, compositional reasoning, and long-horizon action planning. **(2) Reasoning generalization**. GUI agents that are trained solely via imitation learning tend to overfit to interface patterns, resulting in brittle planning and poor generalization when task instructions deviate from seen templates or when UI layouts exhibit minor variations. **(3) Language and regional coverage**. Current research concentrates almost exclusively on English GUIs, paying limited attention to the rapidly growing and diverse Chinese mobile ecosystem, whose interface design conventions and linguistic cues differ substantially. These differences limit the generalizability of current agents in multilingual and culturally diverse settings.

To address these challenges, we propose AgentCPM-GUI, a VLM-based agent for mobile GUI understanding and interaction. The key features of this work are as follows.

- **High-quality training data.** We curate a large-scale corpus of 55K trajectories with 470K steps, encompassing a wide variety of Chinese Android apps via targeted collection and meticulous annotation. To enhance generalization and mitigate overfitting, we further incorporate and rigorously de-duplicate multiple public English Android datasets. The resulting unified dataset supports effective training, enabling robust cross-lingual and cross-app behavior modeling.

- **Progressive training for perception, imitation, and reasoning.** We adopt a three-stage progressive training pipeline to equip the agent with strong GUI understanding and reasoning capabilities, consisting of grounding-aware pre-training to enhance visual perception; supervised fine-tuning (SFT) to establish a reliable behavioral prior; and reinforcement fine-tuning (RFT) (OpenAI, 2024; Shao et al., 2024; Trung et al., 2024) to further strengthen reasoning ability, enabling robust performance on long-horizon and compositional tasks. In addition, we optimize the training framework with asynchronous rollout and load balancing to support scalable reinforcement learning.

- **Edge device oriented design.** To reduce decoding overhead, we carefully select action tokens to avoid unnecessary token fragmentation and adopt a compact JSON-based action format, resulting in an average output length of just 9.7 tokens per action. While prior works largely overlook redundancy in action space design, our concise representation significantly improves runtime efficiency, enabling smooth and responsive on-device execution.

- **Comprehensive benchmarking.** We evaluate AgentCPM-GUI on the widely used English GUI agent benchmarks: AndroidControl (Li et al., 2024), GUI-Odyssey (Lu et al., 2024a), and AITZ (Zhang et al., 2024). In addition, we introduce **CAGUI**, the first large-scale <u>C</u>hinese <u>A</u>ndroid <u>GUI</u> benchmark. CAGUI is a representative subset of our corpus designed for public evaluation. AgentCPM-GUI achieves new state-of-the-art performance across all datasets, demonstrating robust multilingual and cross-app generalization.

2 Method

2.1 Architecture Overview

As shown in Figure 1, we adopt a three-stage training framework to transform MiniCPM-V (Yao et al., 2024), a lightweight 8B vision-language model, into a GUI-capable agent. The first stage focuses on visual perception and grounding, using tasks like OCR and widget localization to enhance the model's ability to align GUI elements with language. In the second stage, the model is fine-tuned on supervised GUI trajectories paired with natural language instructions, enabling it to imitate human-like actions. Finally, reinforcement fine-tuning is applied using Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to further improve planning and decision-making.

2.2 Action Space Design

We design a unified and compositional action space that is compact and friendly for language model generation. It consists of six atomic actions, enabling expressive yet efficient GUI control:

- `POINT`: Specifies a normalized coordinate `(x,y)` in `[0,1000]` to perform a tap. Combined with `to` or `duration`, it supports swipes and long presses.

- `to`: Indicates swipe direction or complements `POINT` to define gesture endpoints.

- `TYPE`: Inputs a specified text string into the

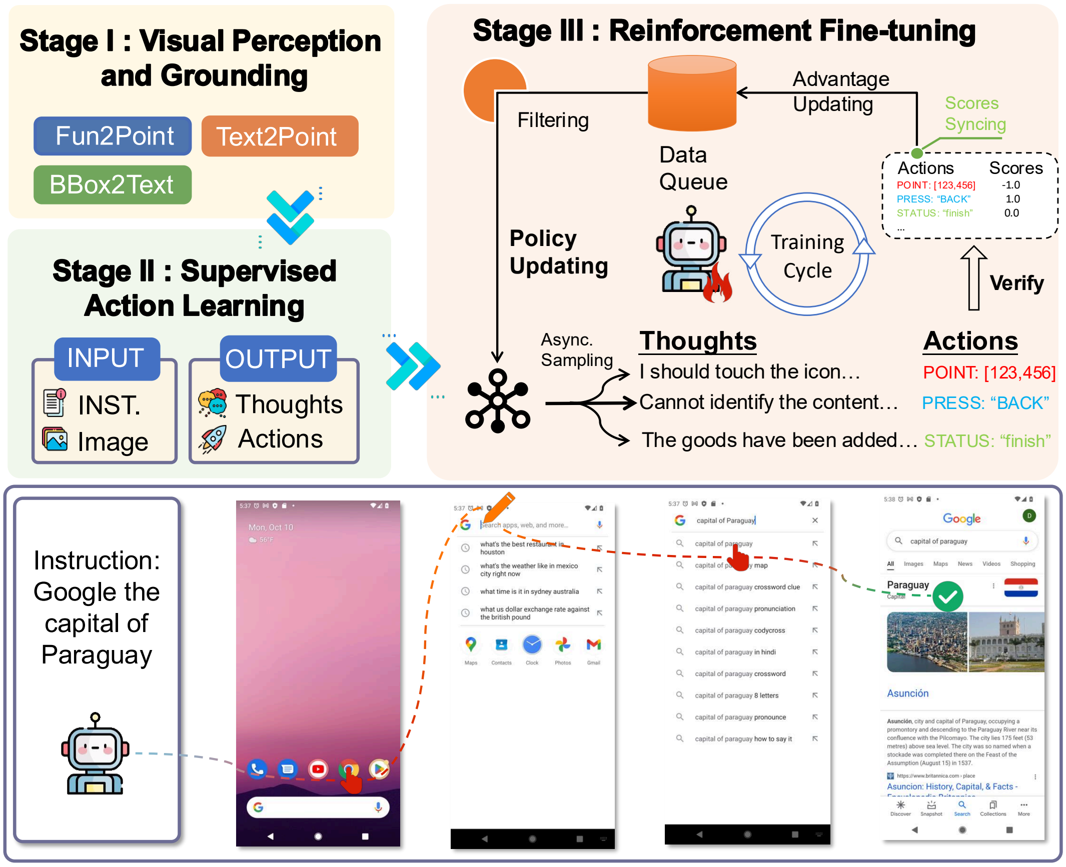


Figure 1: Overview of our training framework.

currently focused input field.

- `PRESS`: Simulates device keys like `"HOME"`, `"BACK"`, or `"ENTER"` for common operations.
- `STATUS`: Communicates task state (e.g., `"continue"`, `"finish"`, `"impossible"`), allowing dynamic control flow.
- `duration`: Time an action lasts. Used alone for delays or with `POINT` for long actions.

To reduce token overhead, we adopt a compact JSON format with no extra whitespace, resulting in a low average token cost of 9.7 per action, enabling fast and efficient execution on edge devices.

2.3 Stage I: Visual Perception and Grounding

For grounding pre-training, we collect Android GUI data by sampling examples from several open-source corpora (AITZ (Zhang et al., 2024), GUICourse (Chen et al., 2025b), OS-Atlas (Wu et al., 2025), UGround (Gou et al., 2025), ScreenSpot (Cheng et al., 2024)) and additional screenshots from our collected Chinese app data. Each image is formulated as either an OCR task that asks the model to write the text in a marked region, or a widget-localization task that asks it to output the bounding box coordinate of a referenced UI element. Grounding batches mix in 50% general multimodal SFT data (e.g., Chat, VQA, Multimodal Reasoning) (Yao et al., 2024), which regularizes the vision module while letting it absorb GUI-specific cues. In total, the grounding pre-training dataset comprises 12M samples.

This pre-training stage plays a crucial role in establishing the model's low-level perceptual and grounding abilities. We observe that, after this stage, the model demonstrates strong proficiency in identifying and locating GUI widgets, especially in accurately predicting coordinates based on visual cues. However, the model at this point still struggles to generate well-formed function calls or to reason over action types, indicating limited understanding of higher-level task semantics and planning. These capabilities are further enhanced in the subsequent SFT and RFT stages.

2.4 Stage II: Supervised Imitation Learning

Due to the scarcity of high-quality open-source datasets for Chinese Android apps, we constructed a large-scale, high-fidelity dataset of GUI interaction trajectories to support supervised imitation learning. The corpus covers over 30 mainstream Chinese apps, spanning eight functional domains: life services, e-commerce, navigation, social, video, music/audio, reading/learning, and productivity. This ensures that the agent is exposed to a wide spectrum of UI layouts, widget types, and task intents. In total, we obtained 55K complete task trajectories comprising 470K atomic steps, approximately 8.5 steps per trajectory.

In order to enhance cross-lingual generalization and reduce over-fitting, we augmented our Chinese corpus with publicly available English-language datasets: AITW (Li et al., 2024), AITZ (Zhang et al., 2024), AMEX (Chai et al., 2025), AndroidControl (Li et al., 2024), and GUI-Odyssey (Lu et al., 2024a). Since AITW is internally redundant, we performed intra-query de-duplication. For each trajectory, we extracted ResNet-50 features from its screenshots and averaged them to produce a trajectory embedding. We then grouped trajectories by shared query and, within each group, removed those whose cosine similarity to any previously retained sample exceeded a fixed threshold. This retained approximately 40% of the original data.

Empirically, training solely on GUI-interaction data led to a pronounced mode collapse during the subsequent RFT stage, manifesting as impoverished and repetitive reasoning thoughts. To mitigate this, we mixed 50% general multimodal SFT data into training batches, which helped stabilize policy optimization. The SFT data comprises a mix of single-turn (system-user-assistant) and multi-turn dialogues. For multi-turn examples, we retained only the last three turns of user-assistant interaction to provide sufficient conversational context while keeping input sequences within tractable length limits. In total, 6.9M instances were used for the SFT stage.

2.5 Stage III: Reinforcement Fine-tuning

We introduce an RFT stage to improve the agent's reasoning ability. To make RFT practical at scale, we further develop a training framework which supports asynchronous rollout and two levels of load balancing to improve efficiency and scalability across distributed environments.

2.5.1 Algorithmic Design

We conduct RFT based on the GRPO (Shao et al., 2024) algorithm. GRPO replaces the value critic of PPO (Schulman et al., 2017) with a group-wise comparison of candidate completions. For reward design and validation, we apply a two-stage validation scheme to evaluate model outputs: (1) format checking and (2) semantic correctness. The reward is mapped to the range $[-1, 1]$. If an output fails the format check (e.g., malformed structure or missing fields), a reward of -1 is assigned. If the format is correct but the answer is semantically incorrect, the reward is 0. If both format and answer are correct, the reward is 1. For action spaces involving continuous goals, such as predicting a `POINT` target, we further define correctness by spatial accuracy: if the predicted point falls within the ground-truth bounding box, a reward of 1 is assigned; otherwise, 0. This fine-grained reward design encourages both syntactic correctness and task-specific accuracy.

2.5.2 System Optimization

Our training system adopts an asynchronous architecture that decouples rollout execution from policy updates. Once a task ID is dispatched from the global task queue, it is sampled n times according to the GRPO algorithm to generate multiple candidate responses per policy. After inference and reward computation for each sample are complete, the main process computes the advantage for the samples using GRPO's variance-reduced estimator. These advantage values are then sent to the node-level main process for policy updating. The global main process collects all necessary statistics and, when synchronization conditions are met, coordinates a unified policy update across nodes. This design ensures tight integration of GRPO's optimization logic within our distributed, asynchronous training framework.

Asynchronous Rollout. In our design, each GPU group performs inference independently and asynchronously. The inference results are first synchronized to the local node's main process. Then, each local main process communicates its inference status with a global main process, which tracks global rollout progress and coordinates training updates. During inference, each GPU group also asynchronously requests the next batch of data required for computing policy gradients. The global main process monitors the overall rollout status and, once a pre-defined synchronization condition is met, broadcasts a signal to all GPU groups to pause rollout and perform a synchronized model update. This asynchronous rollout scheme ensures that GPU groups operate efficiently without wait-

Table 1: GUI grounding accuracy on the CAGUI benchmark over the Fun2Point, Text2Point, and Bbox2Text sub-tasks. **Bold** and underline indicate the best and second-best results.

Models	Fun2Point	Text2Point	Bbox2Text	Average
Closed-source Models				
GPT-4o (Hurst et al., 2024)	22.1	19.9	14.3	18.8
GPT-4o with grounding (Lu et al., 2024b)	44.3	44.0	14.3	34.2
Open-source Models				
Qwen2.5-VL-7B (Bai et al., 2023)	59.8	59.3	<u>50.0</u>	<u>56.4</u>
InternVL2.5-8B (Dong et al., 2024)	17.2	24.2	45.9	29.1
InternVL2.5-26B (Dong et al., 2024)	14.8	16.6	36.3	22.6
OS-Genesis-7B (Sun et al., 2025)	8.3	5.8	4.0	6.0
UI-TARS-7B (Qin et al., 2025b)	56.8	<u>66.7</u>	1.4	41.6
OS-Altas-7B (Wu et al., 2025)	53.6	60.7	0.4	38.2
Aguvis-7B (Xu et al., 2024)	<u>60.8</u>	**76.5**	0.2	45.8
AgentCPM-GUI	**79.1**	**76.5**	**58.2**	**71.3**

ing for each other, thus fully utilizing resources.

Hierarchical Load Balancing. The asynchronous design introduces challenges related to load imbalance, particularly at two levels: intra-node (between GPU groups) and inter-node (between different compute nodes). Intra-node imbalance is addressed by constructing a global task queue from which inference tasks are dynamically dispatched to GPU groups. This design make each GPU group consistently have access to available tasks, thereby minimizing idle time. However, nodes with differing hardware configurations or system loads can result in inter-node imbalance: some nodes may accumulate more rollout results than others. To address this, we implement a work stealing mechanism: underutilized nodes can request inference results from overburdened peers. This approach is particularly suited for large-scale, multi-modal inference outputs, which are often expensive to transmit and manage. Work stealing provides a flexible and scalable solution that avoids the drawbacks of forced synchronization across machines.

3 Experiments

3.1 GUI Grounding Capability

We evaluate GUI grounding on CAGUI through three tasks designed to assess different aspects of visual-language alignment and understanding: **1) Fun2Point**. Given a description of a component's function in the GUI (e.g., "this button opens the website"), the model must locate the correct coordinates of the mentioned component; **2) Text2Point**. The model is required to locate a given textual string appearing within the GUI; **3) Bbox2Text**. The model receives a bounding box location on the GUI and must accurately output the corresponding textual content. Representative examples of these tasks are included in Appendix C.1.

All three grounding tasks are evaluated on the CAGUI benchmark, which was specifically curated for assessing GUI grounding capability in Chinese Android apps. The raw dataset consists of screenshots paired with corresponding XML metadata collected from real-world apps. Each XML file provides fine-grained annotations for GUI widgets, including bounding box coordinates, textual content, and component types. For the Text2Point and Bbox2Text tasks, annotations were directly extracted from the XML metadata by aligning textual content with their corresponding bounding boxes. For Fun2Point, additional function-level labels were constructed to reflect the semantic roles of GUI widgets. To generate these labels, we first overlaid bounding boxes onto the screenshots to explicitly highlight the spatial boundaries of each widget. Then, we prompted a strong VLM Qwen2.5-VL-72B to produce concise functional descriptions, yielding high-quality semantic labels for widgets.

Evaluation procedures were tailored to the input-output formats of each model. InternVL models output bounding boxes, which are evaluated against the ground-truth using the Intersection-over-Union

Table 2: Step-level action prediction performance on five GUI Agent benchmarks, in terms of Type Match (TM) and Exact Match (EM). **Bold** and underline indicate the best and second-best results. *OS-Atlas uses different train/test splits on GUI-Odyssey benchmark and is not directly comparable.

Models	AC-Low		AC-High		Odyssey		AITZ		CAGUI	
	TM	EM	TM	EM	TM	EM	TM	EM	TM	EM
Closed-source Models										
GPT-4o (Hurst et al., 2024)	-	19.5	-	20.8	-	20.4	70.0	35.3	3.67	3.67
Gemini 2.0 (Deepmind, 2024)	-	28.5	-	60.2	-	3.27	-	-	-	-
Claude (Anthropic, 2024)	-	19.4	-	12.5	60.9	-	-	-	-	-
Open-source Models										
Qwen2.5-VL-7B (Bai et al., 2023)	94.1	85.0	75.1	62.9	59.5	46.3	78.4	54.6	74.2	55.2
UI-TARS-7B (Qin et al., 2025b)	**95.2**	**91.8**	**81.6**	**74.4**	86.1	67.9	<u>80.4</u>	<u>65.8</u>	<u>88.6</u>	<u>70.3</u>
OS-Genesis-7B (Sun et al., 2025)	90.7	74.2	65.9	44.4	11.7	3.63	20.0	8.45	38.1	14.5
OS-Atlas-7B (Wu et al., 2025)	73.0	67.3	70.4	56.5	91.8*	76.8*	74.1	58.5	81.5	55.9
Aguvis-7B (Xu et al., 2024)	93.9	89.4	65.6	54.2	26.7	13.5	35.7	19.0	67.4	38.2
OdysseyAgent (Lu et al., 2024a)	65.1	39.2	58.8	32.7	<u>90.8</u>	<u>73.7</u>	59.2	31.6	67.6	25.4
AgentCPM-GUI	<u>94.4</u>	<u>90.2</u>	<u>77.7</u>	<u>69.2</u>	**90.9**	**75.0**	**85.7**	**76.4**	**96.9**	**91.3**

(IoU) metric, with a threshold of 0.5 indicating a successful match. GPT-4o is augmented with OmniParser (Lu et al., 2024b), which extracts layout structures and text/icon segments before the model predicts a target box index. Models including ours generate point coordinates and are assessed by comparing them with ground-truth locations under a predefined spatial tolerance.

The results are summarized in Table 1. AgentCPM-GUI significantly outperforms all baselines across all three tasks. In particular, it achieves a large performance margin in the Bbox2Text task, where most baseline models struggle-largely due to the need for precise alignment between visual regions and text content. Despite the task's difficulty, AgentCPM-GUI attains a 58.2% accuracy, while nearly all competing models score below 5%. This highlights our model's superior grounding ability, especially in mobile interface contexts where visual complexity, small text, and overlapping elements pose unique challenges.

3.2 Action Prediction Capability

We conduct a comprehensive evaluation of AgentCPM-GUI on representative benchmarks: AndroidControl (Li et al., 2024), GUI-Odyssey (Lu et al., 2024a), AITZ (Zhang et al., 2024), and CAGUI, covering diverse GUI interaction patterns across both English and Chinese environments. Each benchmark adopts two standard evaluation metrics: Type Match (TM), which checks if the predicted action type matches the ground truth, and Exact Match (EM), which additionally requires all parameters to be correctly predicted. As shown in Table 2, AgentCPM-GUI achieves state-of-the-art performance across all benchmarks. Notably, it demonstrates strong generalization in complex multi-step scenarios, such as those in GUI-Odyssey and AITZ, significantly outperforming existing models. On the CAGUI benchmark, our model achieves 96.9% TM and 91.3% EM, substantially ahead of other models, highlighting its effectiveness in Chinese-language GUI settings.

All baseline results are from our own re-implementations to ensure fair and reproducible comparisons. We closely followed each model's official instructions and prompts where available, and applied consistent input and evaluation protocols throughout. Notably, OS-Atlas uses a different train/test split on GUI-Odyssey benchmark, so its results are not directly comparable. Our evaluation code and benchmarks are publicly released to support reproducibility and future research.

3.3 Effects of Reinforcement Fine-tuning

To assess the contribution of RFT, we compare our model's performance before and after RFT across all benchmarks, as shown in Table 3. On challenging datasets such as AndroidControl-Low, GUI-Odyssey, and AITZ, RFT brought significant

Table 3: Ablation study comparing AgentCPM-GUI before and after RFT.

Models	AC-Low		AC-High		Odyssey		AITZ		CAGUI	
	TM	EM	TM	EM	TM	EM	TM	EM	TM	EM
AgentCPM-GUI-SFT	87.6	83.1	**78.6**	**69.5**	86.1	66.7	79.0	61.1	**96.9**	**91.5**
AgentCPM-GUI-RFT	**94.4**	**90.2**	77.7	69.2	**90.9**	**75.0**	**85.7**	**76.4**	**96.9**	91.3

improvements, especially in exact match accuracy. This demonstrates its effectiveness in enhancing the model's ability to handle long-horizon reasoning and complex decision-making. However, on datasets like AndroidControl-High and CAGUI, the SFT-only model already performed competitively or even slightly better. This can be attributed to the benchmarks' large and diverse training sets, which expose the model to similar patterns during SFT. As a result, imitation learning alone suffices for effective generalization, with additional reinforcement offering minimal incremental benefits.

4 Related Work

Recent advances in GUI agents have been supported by the development of various datasets and benchmarks, covering both grounding tasks and interaction modeling (Deng et al., 2023; Cheng et al., 2024; Wu et al., 2025; Chen et al., 2025b; Gou et al., 2025; Rawles et al., 2023; Zhang et al., 2024; Li et al., 2024; Lu et al., 2024a; Chai et al., 2025; Rawles et al., 2025). However, most of these focus on English GUIs, limiting cross-lingual generalization. Concurrently, the field has witnessed a transition from modular to end-to-end vision-language agents, with large VLMs trained on millions of screenshots increasingly used for grounding and planning (Wang et al., 2024a; Zheng et al., 2024; Hong et al., 2024; Xu et al., 2024; Qin et al., 2025b; Lin et al., 2025; Yang et al., 2025; Sun et al., 2025). To improve reasoning and adaptability, reinforcement learning techniques have been incorporated, ranging from offline policy training to reward-based fine-tuning and reasoning-centric paradigms (Bai et al., 2024; Wang et al., 2025; Bai et al., 2025; Zhai et al., 2024; Liu et al., 2025b; Tan et al., 2025; Huang et al., 2025; Zhou et al., 2025; Lu et al., 2025; Xia and Luo, 2025; Liu et al., 2025a; Papoudakis et al., 2025).

5 Conclusion

We present AgentCPM-GUI, a VLM-based agent for mobile GUI interaction, trained via a three-stage pipeline that builds grounding, action, and reasoning skills. To support this, we construct a high-quality Chinese Android dataset and incorporate selected English data for cross-lingual generalization. Reinforcement fine-tuning further enhances planning for long-horizon tasks. Experiments on public and CAGUI benchmarks show strong performance, particularly in Chinese settings. All code, data, and models will be released to support future research.

Limitations

While AgentCPM-GUI demonstrates strong performance across both English and Chinese GUI tasks, several limitations remain. First, the model's ability to handle long-horizon interactions is still constrained by limited historical context. Although reinforcement fine-tuning enhances planning and reasoning, the agent only conditions on short, recent trajectories, which can hinder its ability to manage complex, multi-turn tasks requiring memory of earlier states or user preferences. Second, error recovery remains a challenge. The current agent lacks a robust mechanism for detecting failures and autonomously retrying or rolling back actions. While reinforcement training improves overall task success, it does not explicitly teach the model to recover from suboptimal decisions or ambiguous states. Third, our action space, though efficient, assumes deterministic execution and does not yet account for real-time feedback or unexpected UI changes during interaction, which may reduce robustness in deployment. Future work may incorporate memory modules, error-aware execution loops, or uncertainty modeling to further strengthen the agent's autonomy and adaptability in dynamic mobile environments.

Acknowledgments

This work was supported by the Postdoctoral Fellowship Program of CPSF (Grant No. GZC20240831) and the China Postdoctoral Science Foundation (Grant No. 2025M771586).

References

Anthropic. 2024. Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku. https://www.anthropic.com/news/3-5-models-and-computer-use.

Hao Bai, Yifei Zhou, Li Erran Li, Sergey Levine, and Aviral Kumar. 2025. Digi-Q: Learning VLM q-value functions for training device-control agents. In *International Conference on Learning Representations*.

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. DigiRL: Training in-the-wild device-control agents with autonomous reinforcement learning. In *Advances in Neural Information Processing Systems 38*.

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*.

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*, pages 2138–2156.

Wei Chen and Zhiyuan Li. 2024. Octopus v2: On-device language model for super agent. *arXiv preprint*.

Wei Chen, Zhiyuan Li, and Mingyuan Ma. 2025a. Octopus: On-device language model for function calling of software apis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 329–339.

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 models to versatile GUI agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 21936–21959.

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*, pages 9313–9332.

Google Deepmind. 2024. Introducing gemini 2.0: our new ai model for the agentic era. https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36*.

Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Bin Wang, Linke Ouyang, Songyang Zhang, Haodong Duan, Wenwei Zhang, Yining Li, Hang Yan, Yang Gao, Zhe Chen, Xinyue Zhang, Wei Li, Jingwen Li, Wenhai Wang, Kai Chen, Conghui He, and 5 others. 2024. InternLM-XComposer2-4KHD: A pioneering large vision-language model handling resolutions from 336 pixels to 4k HD. In *Advances in Neural Information Processing Systems 38*.

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*.

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*, pages 14281–14290.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. 2025. Vision-R1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint*.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint*.

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*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. In *Advances in Neural Information Processing Systems 36*.

Wei Li, William W. Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on computer control agents. *arXiv preprint*.

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*, pages 19498–19508.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. 2025a. InfiGUI-R1: Advancing multimodal GUI agents from reactive actors to deliberative reasoners. *arXiv preprint*.

Ziyu Liu, Zeyi Sun, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. 2025b. Visual-RFT: Visual reinforcement fine-tuning. *arXiv preprint*.

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

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

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. 2025. UIR1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint*.

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*, pages 22522–22538.

OpenAI. 2024. Reinforcement fine-tuning. `https://platform.openai.com/docs/guides/reinforcement-fine-tuning`.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35*.

Georgios Papoudakis, Thomas Coste, Zhihao Wu, Jianye Hao, Jun Wang, and Kun Shao. 2025. Appvlm: A lightweight vision language model for online app control. *arXiv preprint*.

Cheng Qian, Bingxiang He, Zhong Zhuang, Jia Deng, Yujia Qin, Xin Cong, Zhong Zhang, Jie Zhou, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. Tell me more! towards implicit user intention understanding of language model driven agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 1088–1113.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi R. Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, and 24 others. 2025a. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):101:1–101:40.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.

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. 2025b. UI-TARS: pioneering automated GUI interaction with native agents. *arXiv preprint*.

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*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint*.

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. *arXiv preprint*.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. 2025. Os-Genesis: Automating GUI agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 5555–5579.

Huajie Tan, Yuheng Ji, Xiaoshuai Hao, Minglan Lin, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. 2025. Reason-RFT: Reinforcement fine-tuning for visual reasoning. *arXiv preprint*.

Luong Quoc Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. ReFT: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 7601–7614.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-Agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024b. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Weinan Gan, Xingshan Zeng, Shuai Yu, Xinlong Hao, Kun Shao, Yasheng Wang, and Ruiming Tang. 2024c. GUI agents with foundation models: A comprehensive survey. *arXiv preprint*.

Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. 2025. DistRL: An asynchronous distributed reinforcement learning framework for on-device control agent. In *The Thirteenth International Conference on Learning Representations*.

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*.

Xiaobo Xia and Run Luo. 2025. GUI-R1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024. Aguvis: Unified pure vision agents for autonomous GUI interaction. *arXiv preprint*.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2025. Aria-UI: Visual grounding for GUI instructions. In *Findings of the Association for Computational Linguistics*, pages 22418–22433.

Yuan Yao, Tianyu Yu, Ao Zhang, Chongyi Wang, Junbo Cui, Hongji Zhu, Tianchi Cai, Haoyu Li, Weilin Zhao, Zhihui He, Qianyu Chen, Huarong Zhou, Zhensheng Zou, Haoye Zhang, Shengding Hu, Zhi Zheng, Jie Zhou, Jie Cai, Xu Han, and 4 others. 2024. Minicpm-v: A GPT-4V level MLLM on your phone. *arXiv preprint*.

Simon Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Peter Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, and Sergey Levine. 2024. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. In *Advances in Neural Information Processing Systems 38*.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2025a. Large language model-brained GUI agents: A survey. *Transactions on Machine Learning Research*, 2025.

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*.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024. Android in the zoo: Chain-of-action-thought for GUI agents. In *Findings of the Association for Computational Linguistics*, pages 12016–12031.

Zhuosheng Zhang and Aston Zhang. 2024. You only look at screens: Multimodal chain-of-action agents. In *Findings of the Association for Computational Linguistics*, pages 3132–3149.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2023. A survey of large language models. *arXiv preprint*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*.

Hengguang Zhou, Xirui Li, Ruochen Wang, Minhao Cheng, Tianyi Zhou, and Cho-Jui Hsieh. 2025. R1-zero's "aha moment" in visual reasoning on a 2b non-sft model. *arXiv preprint*.

A Training Details

We list the main hyperparameters for the SFT and RFT stages in Table 4 and Table 5, respectively.

Table 4: Training parameters for Stage II: Supervised Fine-tuning.

Parameter	Default Value	Description
model_max_length	2304	Maximum sequence length
max_line_res	1120	Maximum image resolution for the longest axis
per_device_train_batch_size	1	Training batch size per device
gradient_accumulation_steps	1	Gradient accumulation steps
num_train_epochs	3	Number of training epochs
learning_rate	1e-5	Learning rate
weight_decay	0.1	Weight decay coefficient
adam_beta1	0.9	Adam optimizer beta1 parameter
adam_beta2	0.999	Adam optimizer beta2 parameter
max_grad_norm	N/A	Gradient clipping disabled
lr_scheduler_type	cosine	Learning rate scheduler type
warmup_ratio	0.05	Linear warmup ratio
bf16	True	Use bfloat16 precision
gradient_checkpointing	False	Whether using gradient checkpointing
deepspeed	ZeRO-2	Deepspeed optimization stage

Table 5: Training parameters for Stage III: Reinforcement Fine-tuning.

Parameter	Default Value	Description
max_prompt_length	16384	Maximum prompt length
max_completion_length	512	Maximum completion length
max_line_res	1120	Maximum image resolution for the longest axis
num_generations	8	Number of generations
per_device_train_batch_size	1	Training batch size per device
gradient_accumulation_steps	32	Gradient accumulation steps
learning_rate	1e-6	Learning rate
num_train_epochs	3	Number of training epochs
weight_decay	0.1	Weight decay coefficient
adam_beta2	0.99	Adam optimizer beta2 parameter
max_grad_norm	1.0	Maximum gradient norm for clipping
lr_scheduler_type	cosine	Learning rate scheduler type
beta	0.04	KL divergence coefficient
bf16	True	Use bfloat16 precision

B Evaluation Details

To ensure fair and consistent evaluation across all models, we adopt a unified evaluation framework. Since different models may define their own action formats and conventions, their outputs are first converted into a shared action representation defined by AgentCPM-GUI. This normalization allows us to compare models under the same evaluation criteria and metrics. In the following, we provide representative input prompts for each model, detail the evaluation settings and hyperparameters, and describe how action space conversion is performed when applicable.

B.1 Qwen2.5-VL-7B

B.1.1 Data example

Qwen2.5-VL-7B Data Example

System Message

You are a helpful assistant.

Tools

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

You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name_for_human": "mobile_use", "name": "mobile_use", " description": "Use a touchscreen to interact with a mobile device, and take screenshots.
∗ This is an interface to a mobile device with touchscreen. You can perform actions like clicking, typing, swiping, etc.
∗ 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.
∗ The screen's resolution is 1092x2408.
∗ Make sure to click any buttons, links, icons, etc with the cursor tip in the center of the element. Don't click boxes on their edges unless asked.", "parameters": {"properties": {"action": {" description": "The action to perform. The available actions are:
∗ `key`: Perform a key event on the mobile device.
– This supports adb's `keyevent` syntax.
– Examples: \"volume_up\", \"volume_down\", \"power\", \"camera\", \"clear\".
∗ `click`: Click the point on the screen with coordinate (x, y).
∗ `long\_press`: Press the point on the screen with coordinate (x, y) for specified seconds.
∗ `swipe`: Swipe from the starting point with coordinate (x, y) to the end point with coordinates2 (x2, y2).
∗ `type`: Input the specified text into the activated input box.
∗ `system\_button`: Press the system button.
∗ `open`: Open an app on the device.
∗ `wait`: Wait specified seconds for the change to happen.
∗ `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>

User

The user query: [user_request]
Current step query: low_lew_instruction (included only when low_lew_instruction is defined)
Task progress (You have done the following operation on the current device): [history_actions]
[current_screenshot]

Assistant

[thought_and_action]

B.1.2 Action Space Mapping

Table 6 shows the action space mapping from Qwen2.5-VL-7B to the standardized representation. Two key differences must be addressed during conversion. First, Qwen2.5-VL-7B expresses duration in seconds for actions such as `long_press` and `wait`, whereas AgentCPM-GUI expects time in milliseconds. Second, Qwen2.5-VL-7B produces absolute screen coordinates (in pixels) for spatial actions like `click`, `long_press`, and `swipe`, while AgentCPM-GUI uses normalized coordinates in the range $[0, 1000]$ relative to screen size.

Table 6: Action space mapping from Qwen2.5-VL-7B to AgentCPM-GUI.

Qwen2.5-VL-7B	**Input Parameters**	**AgentCPM-GUI**
`click`	coordinate = (x, y)	`{"POINT":[int(x/width*1000), int(y/height*1000)]}`
`long_press`	coordinate = (x, y), time	`{"POINT":[x,y],"duration": time*1000}`
`swipe`	coordinate = (x1, y1), coordinate2 = (x2, y2)	`{"POINT":[x1,y1],"to": direction}` *Note: direction is computed from two points*
`type`	text	`{"TYPE":text}`
`system_button`	button = Back / Home / Enter	`{"PRESS":BACK/HOME/ENTER}`
`terminate`	None	`{"STATUS":"finish"}`
`wait`	time	`{"duration":time*1000}`

B.1.3 Hyperparameters

We adopt the same hyperparameter settings as used in Qwen2.5-VL-7B for fair comparison, as summarized in Table 7.

Table 7: Inference hyperparameters for Qwen2.5-VL-7B.

Parameter	Default Value	Description
`do_sample`	True	Whether to use sampling (replaces greedy)
`top_p`	0.01	Nucleus sampling threshold
`top_k`	1	Top-k sampling limit
`temperature`	0.01	Controls sampling randomness
`repetition_penalty`	1.0	Penalty factor for repetition
`max_new_tokens`	2048	Maximum number of new tokens to generate

B.2 UI-TARS

B.2.1 Data example

UI-TARS Data Example

System Message

You are a helpful assistant.

User

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

Output Format

```
Thought: . . .
Action: . . .
```

Action Space

```
click(start_box='<|box_start|>(x1,y1)<|box_end|>')
long_press(start_box='<|box_start|>(x1,y1)<|box_end|>', time='')
type(content='')
scroll(direction='down or up or right or left')
press_back()
press_home()
wait()
finished() # Submit the task regardless of whether it succeeds or fails.
```

Note

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

User Instruction

```
[user_request]
```

User

[history_screenshot]

Assistant

[history_thought_and_action]

User

[current_screenshot]

Assistant(included only when low_lew_instruction is defined)

Thought: [low_lew_instruction]
Action:

Assistant

[thought_and_action]

B.2.2 Action Space Mapping

Table 8 shows the action space mapping from UI-TARS to the standardized representation. Since UI-TARS and AgentCPM-GUI define scroll directions oppositely, the direction must be reversed during conversion.

Table 8: Action space mapping from UI-TARS to AgentCPM-GUI.

UI-TARS	Input Format	AgentCPM-GUI
`click(...)`	start_box with (x, y)	`{"POINT":[x,y]}`
`long_press(...)`	start_box with (x, y), time='ms' (optional)	`{"POINT":[x,y],"duration": time (default 1000)}`
`type(...)`	content='text'	`{"TYPE":text}`
`scroll(...)`	direction='up/down/left/right'	`{"POINT":[500,500], "to":reversed direction}` *Note: direction is reversed (e.g., up → down)*
`press_back()`	-	`{"PRESS":BACK}`
`press_home()`	-	`{"PRESS":HOME}`
`wait()`	-	`{"duration":200}`
`finished()`	-	`{"STATUS":"finish"}`

B.3 OS-ATLAS

B.3.1 Data example

OS-ATLAS Data Example

System Message

You are a helpful assistant.

User

You are a foundational action model capable of automating tasks across various digital environments, including desktop systems like Windows, macOS, and Linux, as well as mobile platforms such as Android and iOS. You also excel in web browser environments. You will interact with digital devices in a human-like manner: by reading screenshots, analyzing them, and taking appropriate actions.

Your expertise covers two types of digital tasks:
- Grounding: Given a screenshot and a description, you assist users in locating elements mentioned. Sometimes, you must infer which elements best fit the description when they aren't explicitly stated.
- Executable Language Grounding: With a screenshot and task instruction, your goal is to determine the executable actions needed to complete the task.

You are now operating in Executable Language Grounding mode. Your goal is to help users accomplish tasks by suggesting executable actions that best fit their needs. Your skill set includes both basic and custom actions:

1. Basic Actions
Basic actions are standardized and available across all platforms. They provide essential functionality and are defined with a specific format, ensuring consistency and reliability.
Basic Action 1: CLICK
- purpose: Click at the specified position.
- format: `CLICK <point>[[x-axis, y-axis]]</point>`
- example usage: `CLICK <point>[[101, 872]]</point>`

Basic Action 2: TYPE
- purpose: Enter specified text at the designated location.
- format: `TYPE [input text]`
- example usage: `TYPE [Shanghai shopping mall]`

Basic Action 3: SCROLL
- purpose: Scroll in the specified direction.
- format: `SCROLL [direction (UP/DOWN/LEFT/RIGHT)]`
- example usage: `SCROLL [UP]`

2. Custom Actions
Custom actions are unique to each user's platform and environment. They allow for flexibility and adaptability, enabling the model to support new and unseen actions defined by users. These actions extend the functionality of the basic set, making the model more versatile and capable of handling specific tasks.

Custom Action 1: LONG_PRESS
- purpose: Long press at the specified position.
- format: `LONG_PRESS <point>[[x-axis, y-axis]]</point>`
- example usage: `LONG_PRESS <point>[[101, 872]]</point>`

Custom Action 2: PRESS_BACK
- purpose: Press a back button to navigate to the previous screen.
- format: `PRESS_BACK`
- example usage: `PRESS_BACK`

Custom Action 3: PRESS_HOME
- purpose: Press a home button to navigate to the home page.
- format: `PRESS_HOME`

- example usage: `PRESS_HOME`

Custom Action 4: PRESS_RECENT
- purpose: Press the recent button to view or switch between recently used applications.
- format: `PRESS_RECENT`
- example usage: `PRESS_RECENT`

Custom Action 5: WAIT
- purpose: Wait for the screen to load.
- format: `WAIT`
- example usage: `WAIT`

Custom Action 6: COMPLETE
- purpose: Indicate the task is finished.
- format: `COMPLETE`
- example usage: `COMPLETE`

In most cases, task instructions are high-level and abstract. Carefully read the instruction and action history, then perform reasoning to determine the most appropriate next action. Ensure you strictly generate two sections: Thoughts and Actions.
Thoughts: Clearly outline your reasoning process for current step.
Actions: Specify the actual actions you will take based on your reasoning.

Your current task instruction, action history, and associated screenshot are as follows:
Screenshot:[current_screenshot]
Task: [user_request] You need to: [low_lew_instruction](included only when low_lew_instruction is defined)
History:
[history_low_lew_instruction](included only when low_lew_instruction is defined)

Assistant

[thought_and_action]

B.3.2 Action Space Mapping

Table 9 shows the action space mapping from OS-ATLAS to the standardized representation. When evaluating the AndroidControl-Low setting, we found that the model's predicted scroll direction is often opposite to that indicated in the low-level instruction. Therefore, the scroll direction is reversed during evaluation.

B.4 OS-Genesis

B.4.1 Data Example

For the GUI-Odyssey, AITZ, and CAGUI benchmarks, we construct evaluation prompts following the format described in Data Example. For AndroidControl, we adopt the official evaluation code provided in the benchmark's GitHub repository.

OS-Genesis Data Example

System Message

You are a helpful assistant.

Table 9: Action space mapping from OS-Atlas to AgentCPM-GUI.

OS-Atlas	Input Format	AgentCPM-GUI
`CLICK`	[[x, y]]	`{"POINT":[x,y]}`
`LONG_PRESS`	[[x, y]]	`{"POINT":[x,y],"duration":1000}`
`TYPE`	[text]	`{"TYPE":text}`
`SCROLL`	[direction]	`{"POINT":[500,500],"to":direction}` *Note: if use_low_instruction is True, direction is reversed: up↔down, left↔right*
`PRESS_BACK`	-	`{"PRESS":BACK}`
`PRESS_HOME`	-	`{"PRESS":HOME}`
`PRESS_RECENT`	-	`{"PRESS":RECENT}`
`WAIT`	-	`{"duration":200}`
`COMPLETE`	-	`{"STATUS":"finish"}`

User

You are a GUI task expert, I will provide you with a high-level instruction, an action history, a screenshot with its corresponding accessibility tree.

High-level instruction: [user_request]
Action history:
Accessibility tree:
Please generate the low-level thought and action for the next step.

Assistant

[thought_and_action]

B.4.2 Action Space Mapping

Table 10 shows the action space mapping from OS-Genesis to the standardized representation. Similar to OS-ATLAS, the predicted scroll direction on AndroidControl-Low is often opposite to the instruction, and is therefore reversed during evaluation.

Table 10: Action space mapping from OS-Genesis to AgentCPM-GUI.

OS-Genesis	Input Fields	AgentCPM-GUI
`type`	text	`{"TYPE":text}`
`click`	x, y	`{"POINT":[x,y]}`
`long_press`	x, y	`{"POINT":[x,y],"duration":1000}`
`dismiss`	x, y	`{"POINT":[x,y]}`
`get_text`	x, y	`{"POINT":[x,y]}`
`navigate_home`	-	`{"PRESS":HOME}`
`navigate_back`	-	`{"PRESS":BACK}`
`scroll`	direction	`{"POINT":[500,500],"to":direction}` *Note: If use_low_instruction is True, direction is reversed: up↔down, left↔right*
`wait`	-	`{"duration":200}`

B.5 OdysseyAgent

B.5.1 Data example

Following the official implementation, OdysseyAgent's input consists of the current instruction along with a history of images and their associated actions.

OdysseyAgent Data Example

System Message

You are a helpful assistant.

User

Picture 1: <img>image_path</img>
I'm looking for guidance on how to [instruction]
Previous screenshots: <img>image-history: image_path</img>
Previous Actions: 1. [Action 1]
2. [Action 2].
...

Assistant

[Action]

B.5.2 Action Space Mapping

Table 11 shows the action space mapping from OdysseyAgent to the standardized representation. The output format of OdysseyAgent is largely compatible with AgentCPM-GUI. The only exception is the `RECENT` action, which is not part of the AgentCPM-GUI action space and is therefore ignored during evaluation.

Table 11: Action space mapping from OdysseyAgent to AgentCPM-GUI.

OdysseyAgent	Input Fields	AgentCPM-GUI
`CLICK`	x, y	`{"POINT":[x,y]}`
`LONG_PRESS`	x, y	`{"POINT":[x,y],"duration":1000}`
`SCROLL`	direction	`{"POINT":[500,500],"to":direction}`
`TYPE`	text	`{"TYPE":text}`
`HOME`	-	`{"PRESS":HOME}`
`BACK`	-	`{"PRESS":BACK}`
`COMPLETE`	-	`{"STATUS":"finish"}`
`IMPOSSIBLE`	-	`{"STATUS":"impossible"}`

B.5.3 Hyperparameters

We follow the original implementation for inference, enabling the `image_history` option to incorporate temporal context. Specifically, we store the last 4 actions and their corresponding images. The inference is conducted with the torch seed set to 1234 and the random seed set to 2020 to ensure reproducibility.

B.6 Aguvis-7B

B.6.1 Data Example

Aguvis Data Example

System Message

You are a GUI agent. You are given a task and a screenshot of the screen. You need to perform a series of pyautogui actions to complete the task.
You have access to the following functions:
- {"name": "mobile.swipe", "description": "Swipe on the screen", "parameters": {"type": "object", "properties": {"from_coord": {"type": "array", "items": {"type": "number"}, "description": "The starting coordinates of the swipe"}, "to_coord": {"type": "array", "items": {"type": "number"}, "description": "The ending coordinates of the swipe"}}, "required": ["from_coord", "to_coord"]}}
- {"name": "mobile.home", "description": "Press the home button"}
- {"name": "mobile.back", "description": "Press the back button"}
- {"name": "mobile.wait", "description": "wait for the change to happen", "parameters": {"type": "object", "properties": {"seconds": {"type": "number", "description": "The seconds to wait"}}, "required": ["seconds"]}}
- {"name": "mobile.long_press", "description": "Long press on the screen", "parameters": {"type": "object", "properties": {"x": {"type": "number", "description": "The x coordinate of the long press"}, "y": {"type": "number", "description": "The y coordinate of the long press"}}, "required": ["x", "y"]}}
- {"name": "mobile.open_app", "description": "Open an app on the device", "parameters": {"type": "object", "properties": {"app_name": {"type": "string", "description": "The name of the app to open"}}, "required": ["app_name"]}}

User

Please generate the next move according to the ui screenshot, instruction and previous actions.
Instruction: [Instruction]
Previous actions: [previous_actions]

Assistant

[thought and Action]

Table 12: Action space mapping from Aguvis to AgentCPM-GUI.

Aguvis	Input Fields	AgentCPM-GUI
`pyautogui.click`	x, y	`{"POINT":[x*1000,y*1000]}`
`mobile.long_press`	x, y	`{"POINT":[x*1000,y*1000], "duration":1000}`
`pyautogui.scroll()/hscroll()`	direction	`{"POINT":[500,500],"to": direction}` *Note:* `scroll` *performs vertical, and* `hscroll` *performs horizontal swipes*
`pyautogui.write`	text	`{"TYPE":text}`
`mobile.home()/`	-	`{"PRESS":HOME}`
`mobile.back()`	-	`{"PRESS":BACK}`
`mobile.terminate()`	-	`{"STATUS":"finish"}`
`mobile.open_app`	app_name	-
`mobile.wait`	[time]	`{"duration":3000}`

B.6.2 Action Space Mapping

Table 12 shows the action space mapping from Aguvis to the standardized representation. All coordinates in Aguvis are in the range $[0, 1]$ and are scaled accordingly during conversion. Swipe actions are mapped following the definition in the `pyautogui` package. Since AgentCPM-GUI does not include an "open app" action, it is ignored during evaluation.

B.6.3 Hyperparameters

The hyper parameters are the same as the origin implementation. To be specific, we choose "self-plan" mode during inference, with temperature set as 0 and generate only 1024 new max tokens. Historical actions are not included during inference, as their inclusion leads to abnormal model behavior.

C CAGUI Benchmark

C.1 CAGUI_Grounding

We provide examples from the three tasks that constitute the grounding benchmark, each containing 1,500 samples. The Text2Bbox and Bbox2Text tasks are based on the same dataset. Each bounding box is defined by four absolute coordinates in the format <x_{min}, y_{min}, x_{max}, y_{max}>, with the origin located at the top-left corner of the screen.

Text2Point Data Examples

Text

QQ音乐

Bounding Box

<643, 462, 849, 744>

Prompt of AgentCPM-GUI

你是一个GUI组件定位的专家，擅长输出图片上文本对应的坐标。你的任务是根据给定的GUI截图和图中某个文本输出该文本的坐标。输入：屏幕截图，文本描述输出：文本的相对坐标的中心点,POINT:[...,...]为格式

Bbox2Text Data Examples

Bounding Box

<60, 120, 132, 192>

Bounding Box

返回

Prompt of AgentCPM-GUI

你是一个GUI组件文字识别的专家，擅长根据组件的边界框（bounding box）描述输出对应的文字。你的任务是根据给定的GUI截图和图中某个组件的边界框输出组件的中的文字。输入：屏幕截图，边界框的坐标输出：组件中的文本

Fun2Point Data Examples

Function

UI元素是一个菜单按钮。其主要功能是弹出一个菜单面板，允许用户选择不同的功能选项。通常可以通过点击该按钮触发，点击后会展示一个下拉或侧滑菜单，用户可以在其中进行进一步操作，例如切换功能页面或设置选项。

Bounding Box

<1061, 2424, 1159, 2522>

Prompt of AgentCPM-GUI

你是一个GUI组件定位的专家，擅长根据组件的功能描述输出对应的坐标。你的下一步操作是根据给定的GUI截图和图中某个组件的功能描述点击组件的中心位置。坐标为相对于屏幕左上角位原点的相对位置，并且按照宽高比例缩放到0～1000 输入：屏幕截图，功能描述输出：点击操作，以POINT:[...,...]为格式，其中不能存在任何非坐标字符

C.2 CAGUI_Agent

We present examples of our dataset tasks, each consisting of a query, a screenshot, and the corresponding answer operation. The system prompt used to evaluate AgentCPM-GUI is also included. In total, the benchmark comprises 600 tasks, which together contain 4,516 single-step images. During evaluation, inputs to AgentCPM-GUI follow the standard chat format. Each user message contains both the task query and the associated screenshot, structured as a list with two elements: a text string formatted as "<Question>{query}</Question>\n当前屏幕截图：" and the corresponding image.

Agent Data Examples

Query

请优酷视频根据我的历史记录播放7天内观看超过60%的短视频。

Operation

Action Type: Click
Action Detail: [0.13, 0.61]

System Prompt of AgentCPM-GUI

 # Role
你是一名熟悉安卓系统触屏GUI操作的智能体，将根据用户的问题，分析当前界面的GUI元素和布局，生成相应的操作。

 # Task
针对用户问题，根据输入的当前屏幕截图，输出下一步的操作。

 # Rule
- 以紧凑JSON格式输出
- 输出操作必须遵循Schema约束

 # Schema

```
{
  "type": "object",
  "description": "执行操作并决定当前任务状态",
```

```
  "additionalProperties": false,
  "properties": {
    "thought": {
      "type": "string",
      "description": "智能体的思维过程"
    },
    "POINT": {
      "$ref": "#/$defs/Location",
      "description": "点击屏幕上的指定位置"
    },
    "to": {
      "description": "移动，组合手势参数",
      "oneOf": [
        {
          "enum": [
            "up",
            "down",
            "left",
            "right"
          ],
          "description": "从当前点（POINT）出发，执行滑动手势操作，方向包括向
上、向下、向左、向右"
        },
        {
          "$ref": "#/$defs/Location",
          "description": "移动到某个位置"
        }
      ]
    },
    "duration": {
      "type": "integer",
      "description": "动作执行的时间或等待时间，毫秒",
      "minimum": 0,
      "default": 200
    },
    "PRESS": {
      "type": "string",
     "description": "触发特殊按键，HOME为回到主页按钮，BACK为返回按钮，ENTER为
回车按钮",
      "enum": [
        "HOME",
        "BACK",
        "ENTER"
      ]
    },
    "TYPE": {
      "type": "string",
      "description": "输入文本"
    },
    "STATUS": {
```

```
      "type": "string",
            "description": "当前任务的状态。特殊情况：satisfied，无需操作；impossible，任务无法完成；interrupt，任务中断；need_feedback，需要用户反馈；",
      "enum": [
        "continue",
        "finish",
        "satisfied",
        "impossible",
        "interrupt",
        "need_feedback"
      ],
      "default": "continue"
    }
  },
  "$defs": {
    "Location": {
      "type": "array",
       "description": "坐标为相对于屏幕左上角位原点的相对位置，并且按照宽高比例缩放到0～1000，数组第一个元素为横坐标x，第二个元素为纵坐标y",
      "items": {
        "type": "integer",
        "minimum": 0,
        "maximum": 1000
      },
      "minItems": 2,
      "maxItems": 2
    }
  }
}
```

D Case Study

We demonstrate GUI agent tasks on a real Xiaomi 12S device running MIUI 14.0.11. All interactions with the graphical interface are carried out via ADB control based on the AgentCPM-GUI's predicted actions. The original input and output were in Chinese and translated into English.

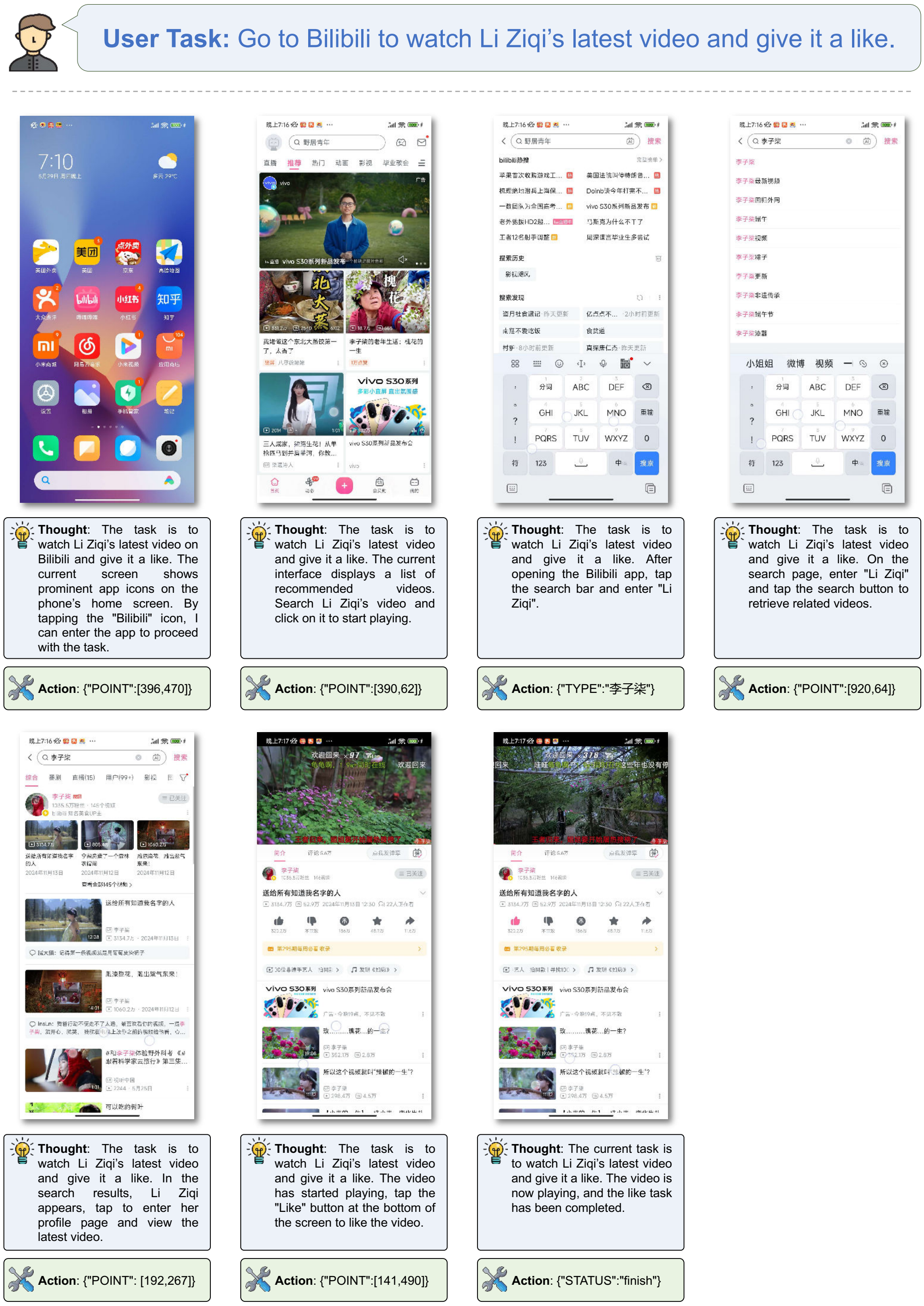
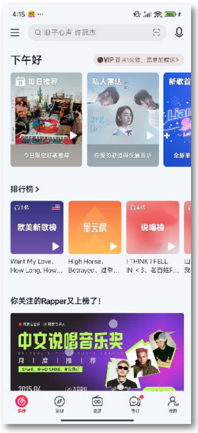


Figure 2: A demo case on the BiliBili.

User Task: Search and play the song "大城小事" on NetEase Cloud Music.



Thought: In the search bar, search "大城小事" and click to enter the playback page.

Action: {"POINT":[356,63]}

Thought: After entering "大城小事" in the search box, tap the search button to look for the song.

Action: {"TYPE":"大城小事"}

Thought: The task is to play "大城小事" via NetEase Cloud Music. The current screen shows the search results, and I need to tap the search button.

Action: {"POINT":[916,59]}

Thought: The search results display multiple related songs. Select "大城小事" for playback.

Action: {"POINT":[550,370]}

Thought: The current screen is the playback interface of NetEase Cloud Music, showing that the song "大城小事" is playing. The task was to search for and play this song, and the interface displays playback information and control buttons, indicating that the song is being played. No further actions are required, the task is complete.

Action: {"STATUS":"finish"}

Figure 3: A demo case on the NetEase Cloud Music.