# $v$SpeedUI: Boosting Mobile GUI Agents via Reusable Transition Graphs

**Xiaohan Zheng**[1], **Yihong Chen**[1], **Haiquan Qiu**[1], **Quanming Yao**[1]
[1]Department of Electronic Engineering, Tsinghua University
zhengxh23@mails.tsinghua.edu.cn, chenyihong@mail.tsinghua.edu.cn, qhq22@mails.tsinghua.edu.cn, qyaoaa@tsinghua.edu.cn

## Abstract

LLM-based mobile GUI agents automate smartphone tasks, but the step-by-step perception and reasoning loop causes high latency. We present $v$**SpeedUI**, an experience-driven system that reuses past trajectories to accelerate execution. $v$SpeedUI builds a Reusable Transition Graph (RTG) whose nodes represent discriminative UI states and whose edges store actions as Semantic Step Summaries capturing step intents and explicit state-level preconditions rather than brittle UI coordinates. At task start, $v$SpeedUI executes Global Look-ahead Planning to retrieve candidate RTG steps and batch-validates and batch-ranks candidate transitions to compile a pre-verified plan. Execution then follows lightweight graph traversal with state localization and action remapping, falling back to a base agent when reuse is infeasible. On HarmonyOS devices, $v$SpeedUI significantly reduces inference latency and total completion time while maintaining strong success rates. Code, extended paper and **Demo Video** are available at https://github.com/lgssstsp-gmail/vSpeed_repo.

## 1 Introduction

LLM-based mobile GUI agents automate smartphone tasks [Xu *et al.*, 2024; Jiang *et al.*, 2025; Qin *et al.*, 2025; Ye *et al.*, 2025; Zhou *et al.*, 2025; Zhang *et al.*, 2025], but most current systems [He *et al.*, 2024; Lu *et al.*, 2025; Dai *et al.*, 2025; Chen *et al.*, 2025] still follow a rigid step-wise perception-reasoning-action loop: they perceive the screen, invoke LLMs to decide the next micro-action, execute, and repeat. Latency/cost scale roughly linearly with task length, limiting real-time use. For usability, *memory matters*. Like human "muscle memory", agents should reuse successful trajectories to avoid repeated perception and reasoning. However, effective reuse is non-trivial: naive approaches like in-context demonstrations [Lee *et al.*, 2024], or per-step retrieval [Guan *et al.*, 2025; Chen *et al.*, 2025] are brittle under UI variation and hard to validate for new goals, relying on superficial similarities rather than semantic preconditions.

We present $v$SpeedUI, boosting efficiency and performance via *structured experience reuse*. We decouple high-level planning from device-level perception using Semantic Step Summaries of step intent and explicit state-level preconditions, and organize them into a Reusable Transition Graph (RTG), where nodes represent discriminative UI states and edges correspond to reusable actions annotated with these semantic constraints. RTG transitions generalize across tasks and interface variations.

On top of the RTG, $v$SpeedUI provides robust execution by combining state matching with action remapping when element positions shift. Crucially, at task initialization, $v$SpeedUI performs Global Look-ahead Planning. Unlike exhaustive search, this acts as a parallel feasibility check; it retrieves candidate transitions and performs batch-validation and batch-ranking to compile a plan that aligns with both explicit preconditions and implicit task logic. During runtime, the agent follows this plan, with safe fallback when reuse is infeasible. Our contributions are summarized as follows.

- We propose an experience-centric execution paradigm for mobile GUI agents, replacing step-by-step LLM reasoning with reusable semantic transitions and enabling Global Look-ahead Planning at initialization.
- We introduce an RTG where actions are stored as semantic intents with state preconditions rather than fixed UI coordinates, allowing cached transitions to be reused across tasks and interface variations.
- Experiments on real devices show substantial reductions in LLM inference latency and task total time, with high reuse rates and maintained or improved success rates under dynamic perturbations.

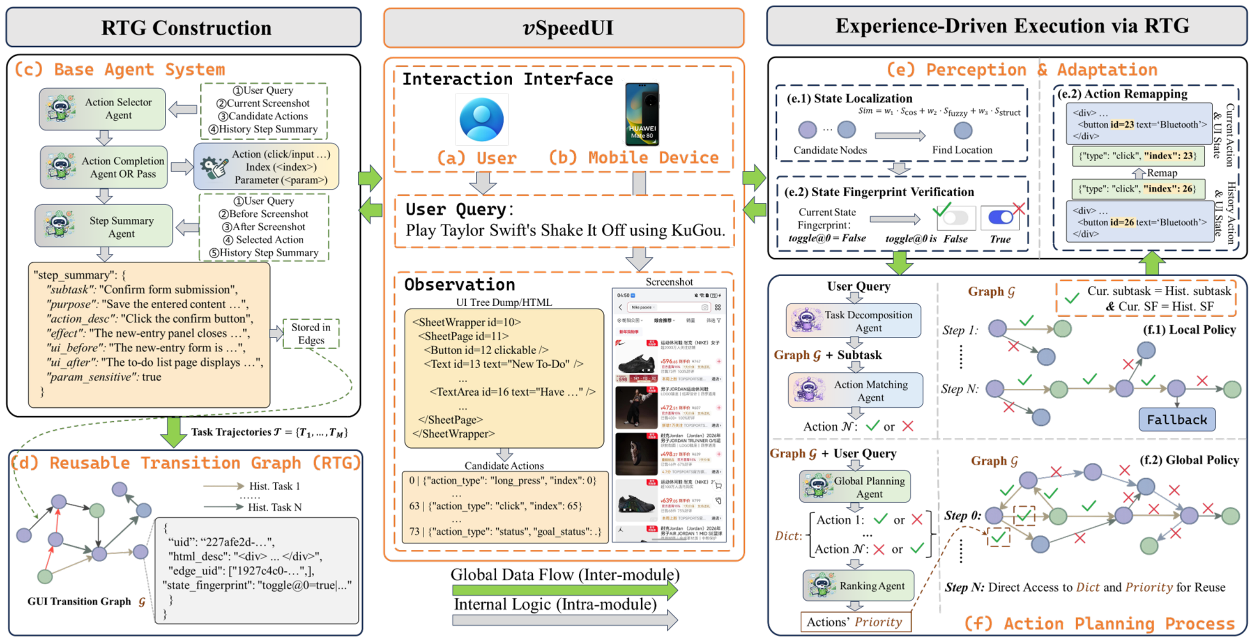
## 2 System Design

Figure 1 illustrates the overall framework of $v$SpeedUI. The system consists of two core modules:

1). *RTG Construction*: It organizes historical trajectories into an RTG for robust reuse under interface variations.

2). *Experience-Driven Execution*: It reuses RTG transitions with state matching and action remapping. At task initialization, the Global Look-ahead Planning batch-validates candidate steps to compile a pre-verified plan, reducing runtime to lightweight graph execution.

**Reusable Transition Graph (RTG) Construction.** We organize historical execution trajectories into a *Reusable Transition Graph (RTG)* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (Fig. 1d). Trajectories are



RTG Construction
vSpeedUI
Experience-Driven Execution via RTG
(c) Base Agent System
Action Selector Agent
①User Query
②Current Screenshot
③Candidate Actions
④History Step Summary
Action Completion Agent OR Pass
Action (click/input …)
Index (<index>)
Parameter (<param>)
Step Summary Agent
①User Query
②Before Screenshot
③After Screenshot
④ Selected Action
⑤History Step Summary
"step_summary": {
"subtask": "Confirm form submission",
"purpose": "Save the entered content …",
"action_desc": "Click the confirm button",
"effect": "The new-entry panel closes …",
"ui_before": "The new-entry form is …",
"ui_after": "The to-do list page displays …",
"param_sensitive": true
}
Stored in Edges
Task Trajectories $\mathcal{T} = \{T_1, \ldots, T_M\}$
(d) Reusable Transition Graph (RTG)
Hist. Task 1
Hist. Task N
GUI Transition Graph $\mathcal{G}$
{
"uid": "227afe2d-…",
"html_desc": "<div> … </div>",
"edge_uid": ["1927c4c0-…",],
"state_fingerprint": "toggle@0=true|…"
}
Interaction Interface
(a) User
(b) Mobile Device
User Query:
Play Taylor Swift's Shake It Off using KuGou.
Observation
UI Tree Dump/HTML
Screenshot
<SheetWrapper id=10>
<SheetPage id=11>
<Button id=12 clickable />
<Text id=13 text="New To-Do" />
…
<TextArea id=16 text="Have …" />
…
</SheetPage>
</SheetWrapper>
Candidate Actions
0 | {"action_type": "long_press", "index": 0}
…
63 | {"action_type": "click", "index": 65}
…
73 | {"action_type": "status", "goal_status": .}
Global Data Flow (Inter-module)
Internal Logic (Intra-module)
(e) Perception & Adaptation
(e.1) State Localization
$Sim = w_1 \cdot S_{cos} + w_2 \cdot S_{fuzzy} + w_3 \cdot S_{struct}$
Candidate Nodes
Find Location
(e.2) State Fingerprint Verification
Current State Fingerprint:
toggle@0 = False
toggle@0 is False
True
(e.2) Action Remapping
<div> …
<button id=23 text='Bluetooth'>
</div>
{"type": "click", "index": 23}
Remap
{"type": "click", "index": 26}
<div> …
<button id=26 text='Bluetooth'>
</div>
Current Action & UI State
History Action & UI State
User Query
Task Decomposition Agent
Graph $\mathcal{G}$ + Subtask
Action Matching Agent
Action $\mathcal{N}$: ✓ or ✗
Graph $\mathcal{G}$
Cur. subtask = Hist. subtask & Cur. SF = Hist. SF
Step 1:
Step N:
(f.1) Local Policy
Fallback
Graph $\mathcal{G}$ + User Query
Global Planning Agent
Dict:
Action 1: ✓ or ✗
Action $\mathcal{N}$: ✗ or ✓
Ranking Agent
Actions' Priority
Graph $\mathcal{G}$
(f.2) Global Policy
Step 0:
Step N: Direct Access to Dict and Priority for Reuse
(f) Action Planning Process

Figure 1: Overview. *v*SpeedUI boosts mobile GUI agents via structured experience reuse with RTG. **Left (RTG Construction):** historical trajectories are converted into RTG transitions annotated with Semantic Step Summaries. **Middle (Environment):** collects user queries and device observations (UI dump tree/screenshot). **Right (Experience-Driven Execution via RTG):** to make RTG reuse reliable under UI changes, we perform state localization, state fingerprint verification, and context-aware action remapping. The Global Policy executes Look-ahead Planning to batch-validate reusable steps at Step 0 to compile a pre-verified plan, which is then executed via lightweight graph traversal with safe fallback.

collected by our Base Agent (Fig. 1c) that interacts with the device and user to produce step sequences [Dai *et al.*, 2025]. For each action $a_t$ taken at state $s_t$, we generate a *Semantic Step Summary* to decouple replayable operations from raw UI details. Specifically, it abstracts raw events into (i) `subtask`, a step intent for cross-task matching, and (ii) `ui_before`, explicit state-level preconditions that specify the required UI context for valid execution.

Given a collected trajectory, we incrementally insert it into the RTG. Each node $v \in \mathcal{V}$ denotes a UI state and stores its UI tree/HTML snapshot plus a *State Fingerprint* that captures task-relevant dynamic attributes (Fig. 1d). This fingerprint enables robust deduplication across repeated runs: a new node is created only when the fingerprint or semantic evidence indicates a meaningful state change. Each directed edge $e \in \mathcal{E}$ denotes a transition induced by an action, and stores its *Semantic Step Summary*, executable parameters, and *origin-task metadata*. When ingesting new trajectories, we merge transitions that match existing nodes/edges and append unseen branches, so graph coverage grows continuously with accumulated experience.

**Experience-Driven Execution via RTG.** We treat the constructed RTG as a library of reusable semantic transitions. At runtime, *v*SpeedUI relies on three key mechanisms: (1) grounding current observations to the RTG, (2) ensuring execution safety via fallback mechanisms, and (3) policies for reuse decision.

*1) Perception & Adaptation (Grounding).* This layer (Fig. 1e) ensures reliable reuse under dynamic UI changes. We first perform *State Localization* to map the current UI to candidate RTG nodes via stable UI-structure similarity. Then, *State Fingerprint Verification* is applied to confirm task-critical micro-states (e.g., whether the Bluetooth switch is already toggled in a "Turn on Bluetooth" task). During execution, we utilize *Context-Aware Action Remapping* to locate targets under UI drift, enabling stored transitions to remain executable beyond brittle coordinates.

*2) Reuse vs. Fallback (Safety).* Grounding serves as a safety gate. If the localized state mismatches or step preconditions do not hold (e.g., no valid outgoing transition exists), *v*SpeedUI aborts reuse and falls back to the Base Agent. This conditional acceleration prevents error accumulation rather than enforcing reuse as a hard constraint.

*3) Local vs. Global (Reuse Decision Policy).* While the Local Policy (Fig. 1 f.1) performs reactive step-wise matching, the Global Policy (Fig. 1 f.2) shifts reasoning to **Step 0**. It retrieves candidate edges and performs **Global Look-ahead Planning** to batch-validate their semantic alignment with the user goal. If ambiguity arises where multiple valid edges co-exist at the same state (e.g., opening different apps on the home screen for cross-app tasks), we invoke a *Ranking Agent* to perform *parallel batch-ranking* to align the execution with the *implicit task logic* based on the global goal and *origin-task context*. By shifting logic to a parallelized batch process, the Global Policy enables *v*SpeedUI to resolve long-term intent at initialization, avoiding the linear latency accumulation inherent in traditional step-wise perception-reasoning loops.

## 3 Experiments

**Settings.** We evaluate *v*SpeedUI on a Huawei Mate 80 smartphone running HarmonyOS 6. Our testbed consists of 10 mobile tasks spanning lifestyle social media (Xiaohongshu), music (QQ Music), e-commerce (JD.com), system applications (Memo), and so on, with an average optimal trajectory length of 8.5 steps, and each task is repeated for 15 trials. We compare $v$SpeedUI against SOTA LLM-based GUI agents, including UI-TARS [Qin *et al.*, 2025], Mobile-Agent-v3 [Ye *et al.*, 2025], and GPT-5-based naïve vision/text agents (Vision: predicting coordinates from screenshots; Text: selecting elements via UI tree dump/HTML). We report four key metrics in Table 1 and Table 2: (i) Success Rate (SR): Percentage of tasks completed within $1.5\times$ the ideal step count. (ii) LLM Inference Latency (LLM IL): The time latency incurred by LLM inference. (iii) Reuse Rate (RR): The ratio of steps driven by the graph versus the optimal step count. (iv) Total Time (TT): The end-to-end time to complete the tasks.

Table 1: Main Results.

| **Method** | **SR (%)**↑ | **LLM IL (s/step)**↓ | **Total Time (s/task)**↓ |
|---|---|---|---|
| UI-TARS (72B) | 44.7 | 17.56 | 191.48 |
| Mobile-Agent-v3 (32B) | 50.7 | 27.73 | 276.95 |
| GPT-5 (Naïve Vision) | 47.3 | 18.10 | 189.10 |
| Mobile-Agent-v3 (GPT-5) | 54.0 | 56.41 | 515.01 |
| GPT-5 (Naïve Text) | 64.0 | 20.67 | 215.37 |
| **Our Base Agent (GPT-5)** | **76.7** | 59.58 | 545.44 |

**Notes. Total time** accounts for **system overhead**, averaging 4.55s/step, which covers hardware-level operations such as screenshotting, UI dumping, and environment settling time.

**Benchmarking Comparison.** We first evaluate our Base Agent against SOTA GUI agents. As shown in Table 1, our Base Agent (GPT-5) achieves the highest Success Rate of 76.7%. However, this superior accuracy comes at a high cost: the Base Agent follows a rigid step-wise reasoning loop, leading to an average Total Time of 545.44s per task. The LLM inference latency reaches 59.58s per step, representing the cumulative latency of the agents shown in Fig. 1c, which is prohibitive for real-time applications.

Table 2: Efficiency with Reuse policies under Same-query Reuse.

| **Reuse Setting** | **SR (%)**↑ | **LLM IL (s)**↓ | **Total (s/task)**↓ | **Reuse (%)**↑ |
|---|---|---|---|---|
| Our Base Agent (GPT-5) | 76.7 | 59.58 /step | 545.44 | – |
| Our Reuse Policy (Local, Qwen2.5) | 82.7 | 1.04 /step | 140.57 | 72.9 |
| Our Reuse Policy (Global, GPT-4) | 86.0 | 11.78 /task | 106.80 | 84.8 |
| **Our Reuse Policy (Global, GPT-5)** | 93.3 | 21.46 /task | 79.51 | 93.2 |

**Effectiveness of Reuse Policies.** Table 2 demonstrates the acceleration performance of $v$SpeedUI through structured experience reuse. Note the unit shift in **LLM IL**: traditional loops incur latency **per step (s/step)**, whereas our policy shifts this to a **one-time initialization (s/task)**. By employing the *Global Policy* (GPT-5), the agent achieves a 93.3% Success Rate, a 16.6 absolute improvement over the Base Agent. Crucially, it slashes the Total Time from 545.44s to 79.51s. This efficiency stems from a high *Reuse Rate (*93.2%*)*, where the Global Policy shifts reasoning to Step 0.

**Handling Query Variations.** To test the resilience of our experience reuse, we introduced four types of query perturbations as shown in Table 3: *Synonym Paraphrasing (Para.)*, *Step Addition/Deletion (Decr./Incr.)*, and *Parameter Perturbation (Param.)*. Our results show that RTG reuse remains robust under perturbations. Both Local and Global policies preserve a high Success Rate (SR) while substantially reducing Total Time (TT), with *Global Policy* being the most stable. The main failure mode appears under *step increase* on the complex task, where Reuse Rate (RR) drops and Local Policy Success Rate degrades, yet Global Policy still maintains strong Success Rate with much lower Total Time than the Base Agent.

Table 3: Robustness results under query perturbations.

| | | **Memo (Simple)** | | | **XHS (Hard)** | | |
|---|---|---|---|---|---|---|---|
| **Method** | **Type** | **RR**↑ | **SR**↑ | **TT**↓ | **RR**↑ | **SR**↑ | **TT**↓ |
| **Ours (Local)** | Para. | 85.7 | 88.7 | 80.2 | 72.7 | 73.3 | 180.0 |
| | Decr. | 85.7 | 93.3 | 78.9 | 72.7 | 86.7 | 179.7 |
| | Incr. | 71.4 | 88.7 | 124.3 | 54.6 | 60.0 | 264.0 |
| | Param. | 85.7 | 93.3 | 76.8 | 63.6 | 73.3 | 234.4 |
| **Ours (Global)** | Para. | 100.0 | 100.0 | 44.3 | 84.6 | 85.7 | 135.0 |
| | Decr. | 100.0 | 100.0 | 44.0 | 76.9 | 92.9 | 171.7 |
| | Incr. | 71.4 | 100.0 | 128.8 | 69.2 | 85.7 | 207.6 |
| | Param. | 100.0 | 100.0 | 42.1 | 76.9 | 92.9 | 170.1 |
| **Base Agent** | Mixed | n/a | 93.3 | 334.6 | n/a | 73.3 | 603.0 |

**Note: RR**: Reuse Rate (%), **SR**: Success Rate (%), **TT**: Total Time (s).

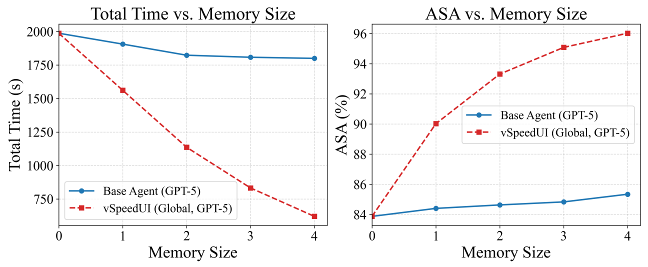


Figure 2: Action Selection Accuracy and time vs. memory size $M$.

**Scaling with More Experience.** We evaluate how $v$SpeedUI scales when the RTG accumulates four diverse tasks ($M$=1...4): social media search (Xiaohongshu), task recording (Memo), music playback (QQ Music), and e-commerce filtering (JD.com). We test the agent on a new, long-horizon cross-app query (e.g., deciding whether to purchase Michael Jackson's *Thriller*). Fig. 2 shows that as $M$ increases, Total Time drops from 1987s to 620s and Action Selection Accuracy (ASA) rises to 96.02%.

**Demonstration.** Our **Demo Video** illustrates $v$SpeedUI solving unseen, complex tasks by dynamically "stitching" transitions from different tasks, showcasing semantic recombination and acceleration over rote replay.

## References

[Chen *et al.*, 2025] Weizhi Chen, Ziwei Wang, Leyang Yang, Sheng Zhou, Xiaoxuan Tang, Jiajun Bu, Yong Li, and Wei Jiang. Pg-agent: An agent powered by page graph, 2025.

[Dai *et al.*, 2025] Gaole Dai, Shiqi Jiang, Ting Cao, Yuanchun Li, Yuqing Yang, Rui Tan, Mo Li, and Lili Qiu. Advancing mobile gui agents: A verifier-driven approach to practical deployment, 2025.

[Guan *et al.*, 2025] Ziyi Guan, Jason Chun Lok Li, Zhijian Hou, Pingping Zhang, Donglai Xu, Yuzhi Zhao, Mengyang Wu, Jinpeng Chen, Thanh-Toan Nguyen, Pengfei Xian, Wenao Ma, Shengchao Qin, Graziano Chesi, and Ngai Wong. Kg-rag: Enhancing gui agent decision-making via knowledge graph-driven retrieval-augmented generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5396–5405. Association for Computational Linguistics, 2025.

[He *et al.*, 2024] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024.

[Jiang *et al.*, 2025] Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. Appagentx: Evolving gui agents as proficient smartphone users, 2025.

[Lee *et al.*, 2024] Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steve Ko, Sangeun Oh, and Insik Shin. Mobilegpt: Augmenting llm with human-like app memory for mobile task automation. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 1119–1133, 2024.

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

[Qin *et al.*, 2025] 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, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025.

[Xu *et al.*, 2024] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction, 2024.

[Ye *et al.*, 2025] Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*, 2025.

[Zhang *et al.*, 2025] Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, Jie Xie, Wei Zhou, Wang Xu, Yuanheng Zhang, Zhou Su, Zhongwu Zhai, Xiaoming Liu, Yudong Mei, Jianming Xu, Hongyan Tian, Chongyi Wang, Chi Chen, Yuan Yao, Zhiyuan Liu, and Maosong Sun. Agentcpm-gui: Building mobile-use agents with reinforcement fine-tuning, 2025.

[Zhou *et al.*, 2025] Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, and Steven Hoi. Mai-ui technical report: Real-world centric foundation gui agents, 2025.