BEYOND DECODING ACCELERATION: FAST PLAN-CONDITIONED GROUNDING FOR GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal GUI agents can automate web, desktop, and mobile interfaces from natural language instructions and screenshots, but their practical use is limited by high per-step latency. While existing acceleration methods mainly reduce subsequent-token decoding cost, GUI grounding typically produces only short actions and coordinates. Our latency analysis shows that the dominant bottleneck is instead the first-token latency caused by repeated visual encoding, multimodal fusion, and prefilling in large vision-language models. To address this, we propose PACE, a Plan-conditioned Action and Coordinate Executor, within a two-speed GUI agent framework. A main model generates concise next-step plans, and PACE executes simple and safe steps by directly predicting the action type and target location from the plan and screenshot. PACE is a lightweight 0.2B dual-stream model with cross-modal fusion and a grid-based grounding head. Across five GUI grounding benchmarks, PACE achieves 64.0% average accuracy with 46.59 ms latency. On OSWorld, integrating PACE reduces main-model calls and total execution time while largely preserving task performance. These results show that low-latency GUI agents require not only faster decoding, but also plan-conditioned delegation that avoids unnecessary heavy multimodal computation.

1 INTRODUCTION

Recent GUI agents have made remarkable progress in computer-use tasks by combining vision-language models with external tools. Modern systems, such as GPT-5.5 (OpenAI, 2026), Claude-4.8-Opus (Anthropic, 2026), Kimi-K2.6 (Team et al., 2026a), and Qwen3.7-Plus (Qwen Team, 2026), can perceive graphical interfaces, reason over user instructions, and execute actions such as clicking, scrolling, typing, and navigation. Despite these advances, the practical deployment of GUI agents remains limited by their high interaction latency. In real-world GUI agent, each action step often requires a large multimodal model to process the current screenshot and generate the next executable action, leading to response times that are quite long. This latency makes current GUI agents difficult to use in scenarios that require fast control (Huang et al., 2026).

The latency of an autoregressive LLM or VLM consists of first-token latency and subsequent decoding latency, and different latency profiles call for different acceleration strategies. Decoding-oriented methods, such as multi-token prediction (Gloeckle et al., 2024; Liu et al., 2024; Aynetdinov & Akbik, 2025) and speculative decoding (Cai et al., 2024; Li et al., 2024; 2025b; Cheng et al., 2025), mainly improve the throughput of autoregressive decoding by generating or verifying multiple future tokens in parallel. These techniques are effective when output sequences are long and decoding dominates the total latency. However, GUI agents typically produce short action outputs, making decoding a limited source of latency. To verify this, we profile four representative VLM-based GUI agents on OSWorld. As shown in Figure 1, time to first-token contributes more than 60% of the total per-step latency for all tested models, suggesting that GUI agent is primarily bottlenecked by repeated multimodal prefilling rather than autoregressive decoding. As a result, decoding-oriented acceleration can only reduce a relatively small portion of the per-step latency in GUI tasks. This latency profile calls for a different acceleration strategy: rather than only accelerating token generation, low-latency GUI agent should avoid unnecessary heavy VLM inference for simple interaction steps, where the required actions are reversible, low-risk, and do not demand strong semantic understanding. Our design turns many low-level GUI interactions from costly VLM generation into efficient plan-conditioned action grounding.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

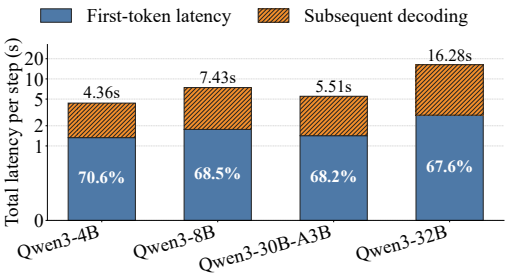

Figure 1: Per-step latency decomposition on OSWorld. We profile four representative VLM-based GUI agents and split action generation latency into time to first-token (TTFT) and subsequent decoding latency. TTFT consistently dominates the total latency, suggesting that decoding-only acceleration is insufficient for low-latency GUI agent.

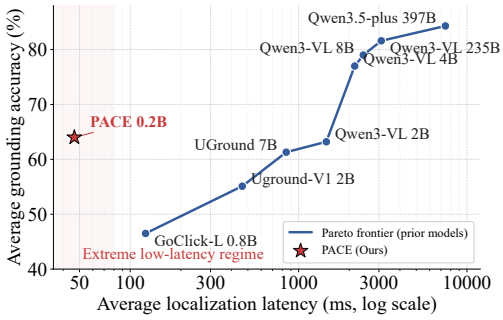

Figure 2: Speed-accuracy Pareto frontier for single-step GUI grounding. PACE achieves competitive accuracy in the extreme low-latency regime compared with Pareto-optimal prior models from Table 1.

Motivated by this observation, we design a complementary acceleration scheme for GUI agents that separates expensive semantic reasoning from lightweight low-level execution. The main VLM is responsible for producing a concise next-step plan and deciding whether the current step can be safely delegated to a lightweight executor. For such simple steps, we introduce PACE, a Plan-conditioned Action and Coordinate Executor, which takes the screenshot and the next-step plan as input and directly predicts both the action type and the target location. PACE adopts a dual-stream text-image architecture with cross-modal fusion, an action classification head for operation prediction, and a convolutional grounding head for spatial localization.

Extensive experiments demonstrate the effectiveness of our approach. On five GUI grounding benchmarks, PACE achieves 64.0% average accuracy with 46.59 ms latency using only 0.2B parameters, establishing a favorable speed-accuracy trade-off among compact GUI grounding models, as shown in Figure 2. On OSWorld, integrating PACE into VLM-based GUI agents reduces main-model calls and total execution time while largely maintaining task performance.

In summary, our contributions are as follows:

- We propose a plan-conditioned acceleration scheme for GUI agents that uses next-step plans to separate expensive semantic reasoning from lightweight low-level execution, reducing unnecessary heavy VLM invocations for simple interaction steps.
- We introduce PACE, a lightweight Plan-conditioned Action and Coordinate Executor, which jointly predicts GUI action types and target locations from screenshots and next-step plans.
- We conduct extensive experiments on GUI grounding benchmarks and OSWorld, showing that PACE achieves a favorable speed-accuracy trade-off and can reduce GUI-agent execution time while largely preserving task performance.

2 Related Work

2.1 Recent Advances in GUI Agents

Recent GUI-agent research has made substantial progress in enabling multimodal models to understand graphical interfaces, follow natural-language instructions, and execute multi-step computer-use tasks. A major line of work focuses on building stronger general-purpose GUI agents through large-scale supervised fine-tuning, reinforcement fine-tuning, or agent-specific training pipelines, as represented by InfiGUIAgent (Liu et al., 2026), AgentCPM-GUI (Zhang et al., 2025), and UI-AGILE (Lian et al., 2025). Another line of work improves the perceptual and grounding capabilities of GUI agents, aiming to better identify interface elements, localize target regions, and produce more

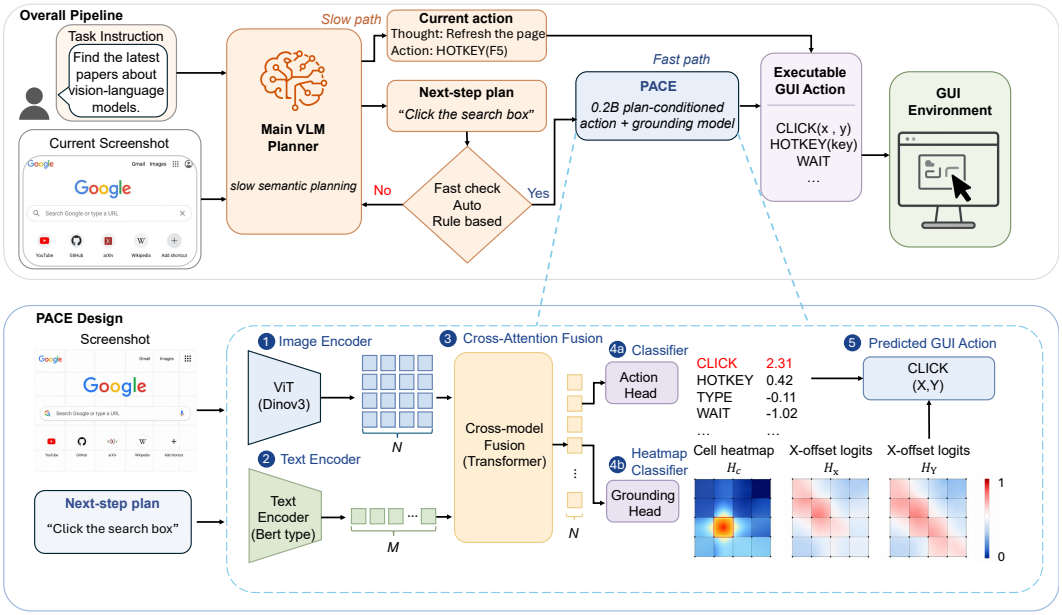

Figure 3: Overview of PACE. The upper panel shows the two-speed GUI agent framework, where the main VLM planner produces a next-step plan and a delegation decision. Simple and safe steps are executed by the fast PACE executor, while complex or risky steps fall back to the main VLM. The lower panel details PACE: screenshot and plan features are fused by cross-attention, followed by an action classifier and a convolutional grounding head that predicts a cell heatmap and intra-cell offsets for target localization.

reliable actions in visually complex environments, including SpiritSight (Huang et al., 2025), TGAgents (Kumbhar et al., 2026), and iSHIFT (Mehrotra et al., 2025). Overall, these studies primarily improve GUI agents by making the underlying models more capable in perception, grounding, reasoning, or task execution. In contrast, PACE studies GUI agents from a latency-oriented perspective. Rather than further increasing the capability of the main VLM, our goal is to reduce unnecessary expensive inference during routine interaction steps by introducing a lightweight plan-conditioned executor that works alongside the main model.

2.2 GUI Grounding and Action Execution

Early end-to-end GUI agents often suffer from inaccurate action localization, motivating recent work on dedicated GUI grounding. Some methods improve grounding by collecting GUI-specific localization data and fine-tuning or reinforcing VLMs on action-coordinate prediction, such as UGround (Gou et al., 2025) and GoClick (Li et al., 2026). Others introduce specialized grounding or action tokens to better align action intents with visual regions, as in GUI-Actor (Wu et al., 2025). While these methods improve spatial accuracy, they still rely on the backbone VLM to observe the screenshot, reason about the next action, and perform grounding at every step. Therefore, they mainly address the precision of grounding rather than the latency caused by repeated heavy VLM inference. In contrast, our work decouples slow planning from fast grounding and delegates simple plan-conditioned actions to a lightweight executor.

3 Method

Figure 3 illustrates the overall design of our method. Our pipeline combines a main VLM planner with PACE, a lightweight plan-conditioned executor. The main VLM remains responsible for semantic reasoning and delegation decisions, while PACE handles simple low-level GUI actions by grounding concise next-step plans in the current screenshot. We next describe the overall pipeline and then introduce the design of PACE.

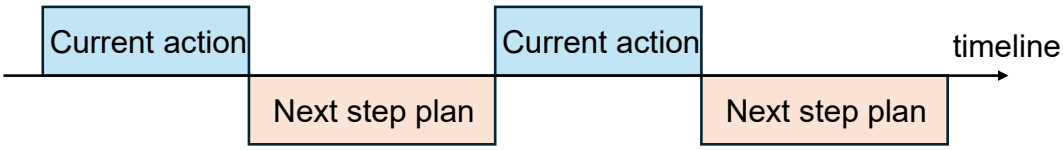

Figure 4: Illustration of the next-step planning mechanism. Each current action is paired with a plan describing the agent's immediately following action.

3.1 Overall Pipeline

To reduce unnecessary heavy VLM inference, we design an overall pipeline that lets the main VLM focus on semantic planning while delegating simple low-level interactions to a lightweight executor, as illustrated in Figure 3. Given the user instruction, interaction history, and current screenshot, the main VLM planner produces two outputs: a normal slow-path action for the current step, and a concise next-step plan that outlines the likely direction of the subsequent interaction, even when the exact action cannot yet be determined, such as *"click the search box"*, *"scroll down to reveal more results"*, or *"click the back button"*. A fast-executable check then determines whether this plan is simple, safe, and directly executable from the current screen. If not, the system follows the slow path and executes the action produced by the main VLM. Otherwise, the system follows the fast path and delegates execution to PACE, a lightweight plan-conditioned executor.

Formally, at step t, let I_t denote the current screenshot, u the user instruction, and h_t the interaction history. The main planner first produces a slow-path action $(a_t^{\text{slow}}, c_t^{\text{slow}})$ and a next-step plan p_t:

$$\big((a_t^{\text{slow}}, c_t^{\text{slow}}), p_t\big) = \text{Main}_{\text{planner}}(u, I_t, h_t), \tag{1}$$

where a_t^{slow} is the action type and $c_t^{\text{slow}} = (x_t^{\text{slow}}, y_t^{\text{slow}})$ is the target coordinate when applicable. The fast-executable check then produces a delegation flag z_t:

$$z_t = \text{FastCheck}(p_t, a_t^{\text{slow}}, I_t), \tag{2}$$

where $z_t = 1$ indicates that the step can be handled by the fast executor. The final executable action is obtained by

$$(a_t, c_t) = \begin{cases} PACE(I_t, p_t), & z_t = 1, \\ (a_t^{\text{slow}}, c_t^{\text{slow}}), & z_t = 0. \end{cases} \tag{3}$$

Here, a_t denotes the final action type and $c_t = (x_t, y_t)$ denotes the target coordinate for spatial actions.

The key interface between the main planner and the fast executor is the next-step plan. Compared with the original task instruction, which can be long, abstract, and temporally extended, the next-step plan specifies only the local action intent at the current step. This substantially narrows the execution problem: PACE does not need to solve the full GUI task, but only needs to ground a short plan in the current screenshot and predict the corresponding executable action. Therefore, fast execution is formulated as a plan-conditioned action grounding problem, where PACE directly predicts both the action type and, when needed, the target location from (I_t, p_t).

To minimize the impact on the original agent performance, we let the main VLM decide whether the current step is suitable for fast execution. The fast path is used only when the main VLM judges that the next-step plan is simple, low-risk, and can be reliably grounded from the current screen. Otherwise, the system keeps the step on the slow path and directly executes the action produced by the main VLM. In this way, the main VLM remains responsible for complex reasoning, ambiguous intents, and safety-sensitive decisions, while PACE accelerates frequent low-level GUI interactions through efficient plan-conditioned action grounding.

3.2 PACE Design

PACE is designed as a lightweight dual-stream vision-language policy model for plan-conditioned GUI execution. Its input consists of the current screenshot I_t and the next-step plan p_t, and its output consists of an action class and a target coordinate when the action is spatially grounded.

Unlike large generative VLMs that autoregressively produce action strings, PACE directly predicts structured GUI actions through discriminative heads, which makes inference substantially faster.

PACE contains three main components: a text encoder, an image encoder, and a cross-modal fusion module. The text encoder maps the next-step plan into a sequence of textual features, $T = E_{\text{text}}(p_t) \in \mathbb{R}^{M \times d_t}$, where M is the number of text tokens. The image encoder maps the screenshot into a sequence of visual patch features, $V = E_{\text{img}}(I_t) \in \mathbb{R}^{N \times d_v}$, where N is the number of visual patches. Since the two encoders may have different feature dimensions, we project both streams into a shared hidden space: $\tilde{T} = TW_T, \qquad \tilde{V} = VW_V$, where $\tilde{T} \in \mathbb{R}^{M \times d}$ and $\tilde{V} \in \mathbb{R}^{N \times d}$.

To align the next-step plan with visual regions, PACE applies a cross-modal fusion module over the projected features. We use visual tokens as queries and text tokens as keys and values, allowing each screenshot patch to be conditioned on the local semantic intent described by the plan:

$$F = \mathcal{F}_{\text{cross}}(\tilde{V}, \tilde{T}) \in \mathbb{R}^{N \times d}. \tag{4}$$

The fused visual features preserve the patch-level spatial layout of the screenshot while incorporating task-relevant textual information. This property is important for GUI grounding, because the target location must be predicted in the original screen coordinate space.

Finally, the fused representation is sent to two prediction heads. The action classification head uses pooled text and visual features to predict the GUI operation type, while the grounding head reshapes the fused visual tokens back into a 2D spatial feature map and predicts the target cell together with intra-cell coordinate offsets. This architecture enables PACE to jointly perform action selection and spatial localization without invoking the main VLM during fast execution.

3.3 Action Classification and Coordinate Grounding

PACE predicts structured GUI actions rather than generating them as text. Specifically, it outputs an action class and, for spatial actions such as clicking or moving, a target coordinate on the screen. This design avoids autoregressive action generation and allows the model to use task-specific prediction heads for action selection and localization.

Action classification. Given the fused visual features F and textual features $\tilde{T}$, we first obtain global representations by pooling over the visual and text tokens: $g_V = \text{Pool}(F), \qquad g_T = \text{Pool}(\tilde{T})$. The action head then predicts a distribution over the predefined GUI action space:

$$P(a_t \mid I_t, p_t) = \text{Softmax}\left(f_{\text{act}}([g_V; g_T])\right), \tag{5}$$

where f_{act} is a lightweight MLP classifier and $[\cdot;;\cdot]$ denotes feature concatenation. The predicted action is obtained as $\hat{a}_t = \arg\max_a P(a \mid I_t, p_t)$. This head determines whether the next step should be executed as a click, scroll, wait, navigation action, or other supported GUI operation.

Spatial feature reconstruction. For coordinate grounding, it is important to preserve the spatial structure of the screenshot. Since the image encoder outputs a sequence of patch tokens, we reshape the fused visual sequence back into a 2D feature map according to the patch layout: $G = \text{Reshape}(F) \in \mathbb{R}^{d \times H \times W}$, where H and W denote the height and width of the visual patch grid. For screenshots with padding or variable aspect ratios, we maintain a valid grid mask and ignore padded regions during prediction. This reconstruction allows the grounding head to operate with convolutional layers over the spatial neighborhood of each visual patch.

Cell and offset grounding. Instead of directly regressing global screen coordinates, PACE predicts the target location in two stages: coarse cell selection and fine intra-cell localization. The grounding head takes the spatial feature map G as input and produces a cell heatmap together with two offset maps: $H_c = f_c(G), \qquad H_x = f_x(G), \qquad H_y = f_y(G)$, where $H_c \in \mathbb{R}^{H \times W}$ indicates the probability of each grid cell containing the target, and $H_x, H_y \in \mathbb{R}^{H \times W}$ estimate the horizontal and vertical offsets inside each cell. We first select the target cell by $(i^*, j^*) = \arg\max_{i,j} H_c(i, j)$. The final normalized coordinate is then decoded by combining the selected cell with its predicted offsets:

$$\hat{x}_t = \frac{j^* + \Delta x_{i^*, j^*}}{W}, \qquad \hat{y}_t = \frac{i^* + \Delta y_{i^*, j^*}}{H}, \tag{6}$$

where $\Delta x_{i^*,j^*}$ and $\Delta y_{i^*,j^*}$ are the intra-cell offsets predicted from H_x and H_y. This cell-plus-offset formulation matches the patch-based structure of vision encoders and enables efficient pixel-level grounding with a lightweight convolutional head.

3.4 Training and Inference

Training objective. PACE is trained with supervised GUI action data. Each training sample consists of a screenshot I_t, a textual execution intent p_t, an action label a_t, and a target coordinate $c_t = (x_t, y_t)$ when the action requires spatial grounding. For non-spatial actions, such as waiting or navigation, only the action classification loss is applied.

The training objective consists of an action classification loss, a cell localization loss, and an intra-cell offset loss. The action head is optimized with cross-entropy over the predefined action space:

$$\mathcal{L}_{\text{act}} = -\log P_{\text{act}}(a_t \mid I_t, p_t). \tag{7}$$

For spatially grounded actions, we convert the target coordinate into a grid cell label (i_t, j_t) and an intra-cell offset target $(\Delta x_t, \Delta y_t)$. The cell heatmap is optimized with a spatial cross-entropy loss:

$$\mathcal{L}_{\text{cell}} = -\log P_{\text{cell}}(i_t, j_t \mid I_t, p_t). \tag{8}$$

The offset head is trained to predict the relative position of the target within the ground-truth cell:

$$\mathcal{L}_{\text{offset}} = \left|\hat{\Delta x}_{i_t,j_t} - \Delta x_t\right| + \left|\hat{\Delta y}_{i_t,j_t} - \Delta y_t\right|. \tag{9}$$

The overall loss is

$$\mathcal{L} = \mathcal{L}_{\text{act}} + \lambda_c \mathcal{L}_{\text{cell}} + \lambda_o \mathcal{L}_{\text{offset}}, \tag{10}$$

where λ_c and λ_o are weighting coefficients. The localization losses $\mathcal{L}_{\text{cell}}$ and $\mathcal{L}_{\text{offset}}$ are applied only to actions with valid target coordinates.

Delegated inference. During end-to-end GUI execution, PACE is used only on the fast path. At each step, the main VLM planner observes the user instruction, interaction history, and current screenshot, and outputs a next-step plan p_t together with a fast-executable flag z_t. If $z_t = 0$, the system follows the slow path and uses the main VLM for execution. If $z_t = 1$, PACE receives (I_t, p_t) and directly predicts the action class and target coordinate:

$$(\hat{a}_t, \hat{c}_t) = \text{PACE}(I_t, p_t). \tag{11}$$

The predicted structured action is then executed in the GUI environment, producing the next screenshot for the following step.

This delegated inference procedure reduces unnecessary invocations of the main VLM on simple steps while preserving the main model for complex, ambiguous, or risky operations. As a result, the system retains the semantic reasoning ability of a strong VLM planner, but avoids repeated heavy multimodal inference for many low-level GUI interactions.

4 Experiment

4.1 Experimental Setup

Training data. We train PACE using GUI grounding data from GUI-Actor (Wu et al., 2025), which provides screenshot-action pairs with target locations for supervised action grounding. To better adapt the fast executor to the OSWorld environment, we further construct a small OSWorld-specific training set. Specifically, we sample 500 interaction trajectories in OSWorld using synthetic tasks, and extract the next-step plan, executed action, and corresponding target coordinate from each step. The resulting data are used to train PACE under the plan-conditioned execution setting, where the model takes the screenshot and next-step plan as input and predicts both the action type and the target location.

Model	Size	ScreenSpot		ScreenSpot-v2		ScreenSpot-Pro		OSWorld-G		MMBench-GUI L2		Average	
		Acc. ↑ (%)	Time ↓ (ms)	Acc. ↑ (%)	Time ↓ (ms)	Acc. ↑ (%)	Time ↓ (ms)	Acc. ↑ (%)	Time ↓ (ms)	Acc. ↑ (%)	Time ↓ (ms)	Acc. ↑ (%)	Time ↓ (ms)
Closed-source & Ultra-Large Models													
Qwen3.5-plus	397B	92.9	6266	96.9	6409	66.2	10880	78.0	6514	87.5	7155	84.3	7445
Qwen3-VL	235B	94.1	3080	97.2	2883	56.5	3490	72.5	2752	87.6	3292	81.6	3099
Models with 4B – 20B Parameters													
SeeClick	10B	55.3	599.2	54.0	595.6	0.9	682.4	10.4	586.4	20.7	614.5	28.3	615.6
Qwen3-VL	8B	91.9	2364	96.2	2313	56.3	3256	66.5	1985	84.0	2201	79.0	2423.8
MiniCPM-V4.5	8B	81.1	1144	84.0	1127	17.9	1293	37.1	1112	64.3	1181	56.9	1171.4
InternVL3.5	8B	85.8	2188	91.4	2221	24.8	2083	44.7	2414	70.5	2725	63.4	2326
OS-ATLAS	8B	73.3	1062	80.0	1095	18.7	1062	33.5	1090	55.5	1138	52.2	1089.4
Qwen2.5-VL	7B	84.7	1274	86.8	1271	26.7	2381	38.6	1095	63.2	1359	60.0	1476
Qwen2-VL	7B	53.5	1169	61.0	1152	4.8	1254	16.5	1294	38.2	1468	34.8	1267
UGround	7B	86.3	848.9	86.9	847.8	31.1	839.3	36.4	841.4	65.7	844.7	61.3	844.4
UI-TARS	7B	89.5	2597	91.6	2593	35.7	2806	47.5	2679	64.3	2774	65.7	2689.8
Qwen3-VL	4B	90.4	2161	92.7	2173	59.6	2719	58.2	1788	84.2	1932	77.0	2154.6
InternVL3.5	4B	82.9	2050	89.1	2100	22.3	1920	41.8	2176	68.6	2606	60.9	2170
Models with 1B – 3B Parameters													
Qwen2.5-VL	3B	55.1	1045	71.5	1040	15.3	1722	28.4	816.1	52.0	1085	44.5	1142
Qwen3-VL	2B	85.0	1370	87.7	1373	32.2	1915	42.9	1247	68.0	1400	63.2	1461.0
UI-TARS	2B	82.3	2376	84.7	2398	27.7	2408	32.9	2398	54.3	2483	56.4	2412.6
ShowUI	2B	76.7	6155	83.1	6183	8.0	6172	20.8	6168	43.6	6175	46.4	6170.6
Uground-V1	2B	77.7	372.4	79.2	377.1	27.7	837.0	34.5	364.4	56.5	364.1	55.1	463.0
InternVL3.5	2B	79.8	1627	86.2	1616	13.2	1542	28.0	1851	56.6	2122	52.8	1751.6
Models $\leq$ 1B Parameters													
InternVL3.5	1B	66.8	948.3	68.6	973.8	5.3	997.1	20.0	1083	41.7	1378	40.5	1076.0
Qwen3.5	0.8B	30.1	1607	29.2	1715	15.2	2056	29.4	2241	39.0	1788	28.6	1881.4
GoClick-L	0.8B	75.9	120.7	77.2	120.1	10.4	126.9	26.3	122.7	42.5	126.5	46.5	123.4
Ours	0.2B	**86.7**	45.03	**87.1**	**45.14**	**35.2**	**51.03**	**42.8**	**43.16**	**68.1**	**48.57**	**64.0**	**46.59**

Table 1: Single-model GUI grounding results. We compare our compact GUI foundation model with proprietary VLMs and open-source GUI grounding base models under a direct grounding setting. Methods that rely on test-time zooming, multi-step refinement, verifier-based reranking, or pipeline-style grounding are excluded from the main comparison. Acc. denotes grounding accuracy, and Time denotes the average end-to-end localization latency.

Implementation details. PACE is trained on 8 NVIDIA RTX 4090 GPUs with 24GB memory each. Unless otherwise specified, we use the same model architecture described in method, including the dual-stream text-image encoders, cross-modal fusion module, action classification head, and convolutional grounding head. The model is trained with the joint objective in Eq. 10, combining action classification, cell localization, and intra-cell offset losses.

Inference and latency measurement. For end-to-end GUI agent evaluation, we use local vLLM services to run the main VLM planners, which allows us to measure inference latency more accurately and avoid fluctuations from remote API services. The vLLM servers are deployed on 4 NVIDIA RTX 4090 GPUs with 48GB memory each. We report both task-level metrics and latency metrics, including the number of main-model invocation steps, total execution time, and average time per step. For single-step grounding evaluation, we measure the end-to-end localization latency of PACE, including screenshot preprocessing, model forwarding, and action-coordinate decoding.

4.2 Research questions.

We organize our experiments around three research questions:

- **RQ1:** Can PACE achieve fast and accurate single-step GUI grounding?
- **RQ2:** Does PACE reduce end-to-end latency when integrated into VLM-based GUI agents?
- **RQ3:** How does the delegation strategy affect the speed-performance trade-off?

Table 2: End-to-end evaluation on OSWorld under different execution strategies. *Baseline* invokes the main VLM planner at every step. + *Next Plan* only asks the planner to produce an explicit next-step plan. + *PACE (Rule)* delegates rule-selected safe steps to PACE, while + *PACE (Auto)* uses automatic delegation. Δ is computed relative to the baseline within each planner group. Lower values are better for steps and time, while higher values are better for score.

Method	Avg. Steps		LLM Steps		Total Time (s)		Time / Step (s)		Score	
	Value	Δ	Value	Δ	Value	Δ	Value	Δ	Value	Δ
Planner: Qwen3-VL-4B-Instruct										
Baseline	11.88	–	11.88	–	31.36	–	2.64	–	19.55	–
+ Next Plan	11.53	-0.35	11.53	-0.35	32.09	+0.73	2.78	+0.14	17.26	-2.29
+ PACE (Rule)	11.71	-0.17	10.62	-1.26	26.51	-4.85	2.26	-0.38	17.21	-2.34
+ PACE (Auto)	11.69	-0.19	**9.66**	**-2.22**	**24.80**	**-6.56**	**2.12**	**-0.52**	17.20	-2.35
Planner: Qwen3-VL-8B-Instruct										
Baseline	13.17	–	13.17	–	42.98	–	3.27	–	25.16	–
+ Next Plan	12.99	-0.18	12.99	-0.18	42.73	-0.25	3.29	+0.02	**26.93**	**+1.77**
+ PACE (Rule)	13.00	-0.17	11.27	-1.90	39.08	-3.90	3.01	-0.26	26.72	+1.56
+ PACE (Auto)	13.05	-0.12	**9.73**	**-3.44**	**36.96**	**-6.02**	**2.83**	**-0.44**	26.86	+1.70
Planner: Qwen3-VL-30B-A3B-Instruct										
Baseline	11.72	–	11.72	–	37.82	–	3.23	–	22.35	–
+ Next Plan	11.43	-0.29	11.43	-0.29	38.16	+0.34	3.34	+0.11	23.28	+0.93
+ PACE (Rule)	11.89	+0.17	10.19	-1.53	33.50	-4.32	2.82	-0.41	23.12	+0.77
+ PACE (Auto)	11.62	-0.10	**9.33**	**-2.39**	**29.48**	**-8.34**	**2.54**	**-0.69**	**23.74**	**+1.39**
Planner: Qwen3-VL-32B-Instruct										
Baseline	11.71	–	11.71	–	85.41	–	7.29	–	29.11	–
+ Next Plan	11.83	+0.12	11.83	+0.12	87.78	+2.37	7.42	+0.13	**29.65**	**+0.54**
+ PACE (Rule)	11.67	-0.04	9.83	-1.88	75.68	-9.73	6.48	-0.81	29.33	+0.22
+ PACE (Auto)	11.77	+0.06	**9.11**	**-2.60**	**70.43**	**-14.98**	**5.98**	**-1.31**	28.90	-0.21

4.3 CAN PACE ACHIEVE FAST AND ACCURATE SINGLE-STEP GUI GROUNDING?(RQ1)

Table 1 compares PACE with proprietary VLMs and open-source GUI grounding models under the direct grounding setting. Overall, PACE achieves 64.0% average accuracy with only 46.59 ms latency, showing a strong speed-accuracy trade-off despite using only 0.2B parameters.

Several observations stand out. First, PACE is substantially faster than all compared models. Even compared with the fastest prior model, GoClick-L, PACE reduces the average latency from 123.4 ms to 46.59 ms, while improving average accuracy from 46.5% to 64.0%. Compared with UGround-V1, PACE improves accuracy by 8.9 points and reduces latency by nearly $10\times$. Second, PACE remains competitive with much larger VLM-based models. It slightly outperforms Qwen3-VL-2B on average accuracy, 64.0% vs. 63.2%, while being over $30\times$ faster. It also surpasses several 4B–8B models such as MiniCPM-V4.5, OS-ATLAS, Qwen2.5-VL-7B, and Qwen2-VL-7B in average accuracy. Third, the gap between PACE and ultra-large VLMs mainly appears on easier grounding benchmarks such as ScreenSpot and ScreenSpot-v2, while PACE remains notably stronger than most compact models on harder benchmarks such as ScreenSpot-Pro and OSWorld-G.

These results indicate that PACE does not aim to replace ultra-large VLMs as a universal visual reasoner. Instead, it occupies a low-latency regime that is well suited for fast execution: it provides sufficiently strong grounding accuracy for simple plan-conditioned GUI steps while reducing localization latency to the millisecond level.

4.4 DOES PACE REDUCE END-TO-END GUI AGENT LATENCY?(RQ2)

Table 2 reports the OSWorld results with four VLM planners. A clear observation is that adding next-step plans alone does not consistently reduce latency. For example, with Qwen3-VL-4B and Qwen3-VL-32B, + *Next Plan* even increases the total time by 0.73s and 2.37s, respectively. This indicates that prompting the main VLM to produce a more explicit plan is not sufficient for acceleration, since the planner is still invoked at every step. In contrast, integrating PACE consistently

reduces the number of LLM steps and total execution time. Under automatic delegation, PACE reduces LLM steps by 2.22, 3.44, 2.39, and 2.60 steps for the 4B, 8B, 30B-A3B, and 32B planners, respectively. This directly confirms that part of the GUI execution can be offloaded from the main VLM to the lightweight executor. The reduction in LLM calls further translates into lower end-to-end latency: total time decreases by 6.56s, 6.02s, 8.34s, and 14.98s across the four planners. The improvement is especially large for the 32B planner, where each VLM call is expensive. The task score is largely preserved for stronger planners. With Qwen3-VL-8B and Qwen3-VL-30B-A3B, PACE (Auto) improves the score by 1.70 and 1.39 points while reducing total time. With Qwen3-VL-32B, the score remains comparable to the baseline, with only a 0.21-point decrease. The 4B planner shows a larger score drop, suggesting that weaker planners may produce less reliable next-step plans or delegation decisions. Overall, these results show that PACE reduces end-to-end GUI latency by avoiding unnecessary main-model invocations, rather than merely changing the prompt format.

4.5 How Does Delegation Strategy Affect the Speed-Performance Trade-off?(RQ3)

We further compare rule-based delegation and automatic delegation. Rule-based delegation is more conservative: it only offloads steps that match predefined safe action types. As a result, it consistently reduces latency while keeping the delegation scope limited. Across the four planners, PACE (Rule) reduces total time by 4.85s, 3.90s, 4.32s, and 9.73s, respectively. Automatic delegation is more aggressive and achieves larger speedups. Compared with the rule-based variant, PACE (Auto) further reduces LLM steps by 0.96, 1.54, 0.86, and 0.72 steps across the four planners. It also further reduces total time by 1.71s, 2.12s, 4.02s, and 5.25s, respectively. This shows that automatic delegation can identify more opportunities for fast execution beyond simple hand-crafted rules. The performance trade-off depends on planner strength. For Qwen3-VL-8B and Qwen3-VL-30B-A3B, automatic delegation achieves better latency with comparable or higher scores than rule-based delegation. For Qwen3-VL-32B, automatic delegation gives the largest latency reduction but slightly lowers the score compared with the rule-based variant. This suggests that automatic delegation provides a more flexible speed-performance trade-off, while rule-based delegation can be preferable when conservative execution is desired.

5 Conclusion

In this paper, we study the latency bottleneck of VLM-based GUI agents and show that GUI action generation is primarily constrained by first-token latency rather than subsequent decoding. Motivated by this observation, we propose PACE, a two-speed GUI agent framework that decouples slow semantic planning from fast plan-conditioned execution. PACE uses a lightweight action-coordinate executor to handle simple and safe GUI steps, while preserving the main VLM for complex reasoning and risky operations. Experiments on single-step GUI grounding benchmarks and OSWorld demonstrate that PACE achieves strong low-latency grounding performance and reduces end-to-end execution time by avoiding unnecessary main-model invocations. These results suggest that efficient GUI agent action should move beyond decoding-only acceleration and instead exploit structured delegation between large planners and compact executors.

AI USE STATEMENT

OpenAI Codex was used to adapt the existing manuscript to the ICLR LaTeX template, arrange the latency and Pareto plots side by side, and check compilation and page layout. The original text, figures, tables, and reported experimental measurements are retained.

REPRODUCIBILITY STATEMENT

The model and training objective are described in Section 3. Experimental settings appear in Section 4.1; implementation details are provided in Appendix A. Delegation prompts and rules are given in Appendix D and Appendix E.4.

REFERENCES

Milad Taleby Ahvanooey, Hassan Dana Mazraeh, and Seyed Hashem Tabasi. An innovative technique for web text watermarking (AITW). *Inf. Secur. J. A Glob. Perspect.*, 25(4-6):191–196, 2016.

Anthropic. Claude opus 4.8 system card. Technical report, Anthropic, June 2026. URL https://www-cdn.anthropic.com/0f0c97ad20d8005706296bd92aa1c27c6b2f4f61/Claude%20Opus%204.8%20System%20Card.pdf. Revised June 17, 2026. Accessed: 2026-06-23.

Ansar Aynetdinov and Alan Akbik. Pre-training curriculum for multi-token prediction in language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 25573–25588. Association for Computational Linguistics, 2025.

Shuai Bai, Ke qin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *ArXiv*, abs/2502.13923, 2025.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, De huai Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. In *International Conference on Machine Learning*, 2024. URL https://api.semanticscholar.org/CorpusID:267061277.

Zicong Cheng, Guo-Wei Yang, Jia Li, Zhijie Deng, Meng-Hao Guo, and Shi-Min Hu. DEER: draft with diffusion, verify with autoregressive models. *CoRR*, abs/2512.15176, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734. PMLR / OpenReview.net, 2024.

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

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

Runxi Huang, Liyu Zhang, Shengzhong Liu, and Xiaomin Ouyang. Mobileexplorer: Accelerating on-device inference for mobile GUI agents via online exploration. *CoRR*, abs/2605.26546, 2026.

Zhiyuan Huang, Ziming Cheng, Junting Pan, Zhaohui Hou, and Mingjie Zhan. Spiritsight agent: Advanced GUI agent with one look. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pp. 29490–29500. Computer Vision Foundation / IEEE, 2025.

Marko Jurmu, Sebastian Boring, and Jukka Riekki. Screenspot: multidimensional resource discovery for distributed applications in smart spaces. In *5th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, MobiQuitous 2008, July 21-25, 2008, Dublin, Ireland*. ICST / ACM, 2008.

Shrinidhi Kumbhar, Haofu Liao, Srikar Appalaraju, and Kunwar Yashraj Singh. Towards GUI agents: Vision-language diffusion models for GUI grounding. *CoRR*, abs/2603.26211, 2026.

Hongxin Li, Yuntao Chen, and Zhaoxiang Zhang. Goclick: Lightweight element grounding model for autonomous GUI interaction. *CoRR*, abs/2604.23941, 2026.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia, MM 2025, Dublin, Ireland, October 27-31, 2025*, pp. 8778–8786. ACM, 2025a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: speculative sampling requires rethinking feature uncertainty. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR / OpenReview.net, 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diago, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *CoRR*, abs/2308.03281, 2023. URL https://doi.org/10.48550/arXiv.2308.03281.

Shuquan Lian, Yuhang Wu, Jia Ma, Zihan Song, Bingqi Chen, Xiawu Zheng, and Hui Li. UI-AGILE: advancing GUI agents with effective reinforcement learning and precise inference-time grounding. *CoRR*, abs/2507.22025, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. Infiguiagent: A multimodal generalist GUI agent with native reasoning and reflection. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2026 - Volume 1: Long Papers, Rabat, Morocco, March 24-29, 2026*, pp. 1035–1051. Association for Computational Linguistics, 2026.

Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. Smolvlm: Redefining small and efficient multimodal models. *CoRR*, abs/2504.05299, 2025.

Sarthak Mehrotra, Sairam VC Rebbapragada, Mani Hemanth Reddy Bonthu, and Vineeth N. Balasubramanian. ishift: Lightweight slow-fast GUI agent with adaptive perception. *CoRR*, abs/2512.22009, 2025.

OpenAI. Openai GPT-5 system card. *CoRR*, abs/2601.03267, 2026. URL https://doi.org/10.48550/arXiv.2601.03267.

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. *CoRR*, abs/2501.12326, 2025. URL https://doi.org/10.48550/arXiv.2501.12326.

Qwen Team. Qwen3.7: The agent frontier. https://qwen.ai/blog?id=qwen3.7, May 2026. Accessed: 2026-06-23.

ByteDance Seed. Ui-tars-1.5. https://seed-tars.com/1.5, 2025a.

ByteDance Seed. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *CoRR*, abs/2509.02544, 2025b. doi: 10.48550/ARXIV.2509.02544. URL https://doi.org/10.48550/arXiv.2509.02544.

Oriane Siméoni, Huy V. Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seung Eun Yi, Michaël Ramamonjisoa, Francisco Massa, Daniel Haziza, Luca Wehrstedt, Jianyuan Wang, Timothée Darcet, Théo Moutakanni, Leonel Sentana, Claire Roberts, Andrea Vedaldi, Jamie Tolan, John Brandt, Camille Couprie, Julien Mairal, Hervé Jégou, Patrick Labatut, and Piotr Bojanowski. Dinov3. *Trans. Mach. Learn. Res.*, 2026, 2026.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026a.

Qwen Team. Qwen3-vl technical report. *CoRR*, abs/2511.21631, 2025. URL https://doi.org/10.48550/arXiv.2511.21631.

Venus Team, Changlong Gao, Zhangxuan Gu, Yulin Liu, Xinyu Qiu, Shuheng Shen, Yue Wen, Tianyu Xia, Zhenyu Xu, Zhengwen Zeng, et al. Ui-venus-1.5 technical report. *arXiv preprint arXiv:2602.09082*, 2026b.

Pavan Kumar Anasosalu Vasu, Fartash Faghri, Chun-Liang Li, Cem Koc, Nate True, Albert Antony, Gokula Santhanam, James Gabriel, Peter Grasch, Oncel Tuzel, and Hadi Pouransari. Fastvlm: Efficient vision encoding for vision language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pp. 19769–19780. Computer Vision Foundation / IEEE, 2025.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, Weiyun Wang, Xiangyu Zhao, Jixuan Chen, Haodong Duan, Tianbao Xie, Chenyu Yang, Shiqian Su, Yue Yu, Yuan Huang, Yiqian Liu, Xiao Zhang, Yanting Zhang, Xiangyu Yue, Weijie Su, Xizhou Zhu, Wei Shen, Jifeng Dai, and Wenhai Wang. Mmbench-gui: Hierarchical multi-platform evaluation framework for GUI agents. *CoRR*, abs/2507.19478, 2025.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*, 2025.

Qinzhuo Wu, Weikai Xu, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, and Shuo Shang. Mobilevlm: A vision-language model for better intra- and inter-ui understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, volume EMNLP 2024 of *Findings of ACL*, pp. 10231–10251. Association for Computational Linguistics, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. Mobile-agent-v3.5: Multi-platform fundamental GUI agents. *CoRR*, abs/2602.16855, 2026. URL https://doi.org/10.48550/arXiv.2602.16855.

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. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025 - System Demonstrations, Suzhou, China, November 4-9, 2025*, pp. 155–180. Association for Computational Linguistics, 2025.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

Yichen Zhu, Minjie Zhu, Ning Liu, Zhicai Ou, Xiaofeng Mou, and Jian Tang. Llava-phi: Efficient multi-modal assistant with small language model. *CoRR*, abs/2401.02330, 2024. doi: 10.48550/ARXIV.2401.02330. URL https://doi.org/10.48550/arXiv.2401.02330.

A Additional Implementation Details

PACE uses a lightweight dual-stream architecture. For language understanding, we use GTE-base (Li et al., 2023) as the text encoder, and for visual perception, we use DINO-v3-base (Siméoni et al., 2026) as the image encoder. Both encoders are fine-tuned during training. The text encoder outputs 768-dimensional token features, and the image encoder outputs 768-dimensional visual patch features. Since the two streams already share the same feature dimension, we set the projection hidden dimension to $d = 768$. The cross-modal fusion module contains 3 cross-attention layers with 8 attention heads. We use visual patch tokens as queries and text tokens as keys and values, so that each visual region is conditioned on the next-step plan. The action prediction head is a 3-layer MLP with hidden size 256. The grounding head is implemented as a 4-layer convolutional network over the reconstructed 2D visual feature map. The full model contains approximately 0.22B parameters, including the text encoder, image encoder, fusion module, action head, and grounding head.

A.1 Input Preprocessing and Coordinate Mapping

For each input sample, PACE takes a screenshot and a next-step plan as input. The maximum text length is set to 256 tokens. Screenshots are resized with aspect-ratio preservation, and the resized height and width are constrained to be multiples of 16 to match the patch size of the image encoder. We use a maximum image resolution of 1024×1536. After image encoding, the screenshot is represented as a sequence of visual patch tokens. For the maximum resolution, this corresponds to a visual patch grid of 64×96. For screenshots with different aspect ratios or sizes, we apply sequence padding after visual tokenization and construct an attention mask to distinguish valid visual tokens from padded tokens. The same valid-token mask is also used in the grounding head, so padded regions are ignored during cell prediction and cannot be selected as target cells. Target coordinates are normalized to $[0, 1]^2$ with respect to the original screenshot size. During training, the normalized coordinate (x, y) is mapped to the valid visual grid. Given a grid of height H and width W, the target cell is computed as

$$j = \lfloor xW \rfloor, \qquad i = \lfloor yH \rfloor,$$

and the intra-cell offset target is

$$\Delta x = xW - j, \qquad \Delta y = yH - i.$$

The cell label (i, j) is used for spatial cross-entropy training, while $(\Delta x, \Delta y)$ is used for offset regression. At inference time, the predicted cell and offsets are decoded into normalized coordinates and then mapped back to the original screenshot coordinate space before evaluation or execution.

A.2 Training Setup

PACE is trained with supervised learning for 5 epochs using the AdamW optimizer. The learning rate is set to 1×10^{-4} and the total batch size is 256. We train the model in FP32 precision on 8 NVIDIA RTX 4090 GPUs. Both the text encoder and the image encoder are fine-tuned together with the cross-modal fusion module and prediction heads. The overall loss follows Eq. 10. We set the action loss weight to 1.0 and use the same default weight for the localization losses. The localization losses are applied only to actions with valid target coordinates. For non-spatial actions, such as waiting or navigation, only the action classification loss is used.

A.3 Latency Measurement

We measure the inference latency of PACE under the single-step GUI grounding setting. All latency results are measured with batch size 1, which matches the online execution setting of GUI agents. The reported latency includes screenshot preprocessing, text tokenization, image encoding, text encoding, cross-modal fusion, action classification, coordinate grounding, and final action-coordinate decoding. For spatial actions, the latency also includes mapping the predicted normalized coordinate back to the original screenshot coordinate space.

We load the model once and then directly measure latency over the evaluation dataset without using additional warm-up runs. The reported value is the average per-sample latency across all evaluated

Table 3: Training hyperparameters of PACE.

Hyperparameter	Value
Optimizer	AdamW
Learning rate	1×10^{-4}
Batch size	256
Epochs	5
Precision	FP32
GPUs	8 × NVIDIA RTX 4090
Action loss weight	1.0

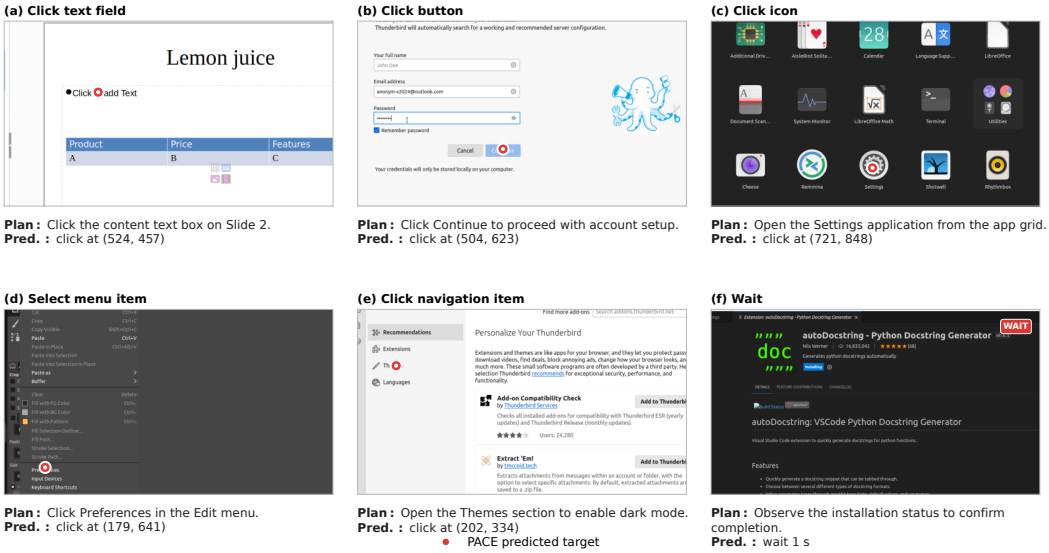

Figure 5: Qualitative case studies of PACE. Given a screenshot and a next-step plan, PACE predicts the action type and target location for diverse low-level GUI interactions, including text fields, buttons, icons, menus, navigation elements, and scrolling.

samples. Model loading time is not included. Since PACE directly predicts structured actions instead of autoregressively generating text, its latency is dominated by encoder forwarding and lightweight prediction heads rather than token-by-token decoding.

For end-to-end OSWorld evaluation, we separately measure the total execution time of each task trajectory. The total time includes main VLM planner inference, PACE inference on delegated steps, action parsing, and environment interaction overhead. We also report the number of main-model invocation steps to quantify how much computation is offloaded from the VLM planner to the fast executor.

A.4 End-to-End Deployment

In end-to-end GUI execution, PACE is integrated into the two-speed agent framework as a fast executor. At each step, the main VLM planner observes the user instruction, interaction history, and current screenshot. The planner outputs a next-step plan and a fast-executable decision. If the current step is not suitable for delegation, the system follows the slow path and uses the main VLM planner to generate and execute the action. If the step is marked as simple and safe, the system follows the fast path and sends the current screenshot and next-step plan to PACE. PACE then predicts a structured GUI action, including the action type and the target coordinate when the action requires spatial grounding. The predicted action is converted into the corresponding OSWorld executable operation and applied to the environment. After execution, the environment returns a new screenshot, and the next decision step starts. We consider two delegation strategies. In rule-based delegation, only predefined low-risk actions are delegated to PACE, such as clicking, scrolling, waiting, and

Table 4: Ablation study of cross-modal fusion.

Fusion Module	Avg. Acc. ↑	Time (ms) ↓
No fusion	46.3	**22.69**
Cross-attention fusion	**64.0**	46.59

Table 5: Ablation study of grounding head design.

Grounding Head	Avg. Acc. ↑	Time (ms) ↓
Direct coordinate regression	26.8	48.72
Cell heatmap only	57.3	**43.71**
Cell + offset	**64.0**	46.59

simple navigation. Operations involving typing, hotkeys, confirmation, submission, deletion, sending, purchasing, or saving are routed back to the main VLM planner. In automatic delegation, the system uses the planner's fast-executable decision to determine whether to call PACE. This strategy allows the agent to delegate more steps beyond hand-crafted rules, but it may also be more sensitive to the quality of the planner's decision. This deployment design preserves the reasoning and safety handling ability of the main VLM for complex or risky operations, while allowing frequent low-level GUI interactions to be executed by the lightweight PACE executor.

B Additional Ablation Studies

We conduct additional ablation studies to better understand the contribution of each component in PACE. Unless otherwise specified, all ablation experiments follow the same training and evaluation protocol as the main model. We report the average grounding accuracy and average inference latency across the GUI grounding benchmarks used in the main experiments.

B.1 Effect of Cross-modal Fusion

We first study the effect of cross-modal fusion between the next-step plan and screenshot features. Since PACE is designed as a plan-conditioned executor, the visual representation should be aware of the local execution intent described by the plan. We compare a variant without explicit cross-modal fusion against the cross-attention fusion used in PACE.

As shown in Table 4, removing cross-modal fusion significantly reduces the average grounding accuracy from 64.0% to 46.3%. This indicates that simply encoding the screenshot and plan separately is insufficient for accurate GUI grounding. In contrast, cross-attention fusion allows visual patch features to be conditioned on the next-step plan, leading to a 17.7-point accuracy improvement. Although cross-attention introduces additional computation, the resulting latency remains below 50 ms, which is still substantially faster than VLM-based grounding models. This confirms that plan-aware visual fusion is important for PACE's accuracy while preserving low-latency execution.

B.2 Effect of Grounding Head Design

We then ablate the coordinate grounding head. The full PACE model predicts a coarse grid cell and an intra-cell offset. To evaluate this design, we compare it with two alternatives: directly regressing the global coordinate from pooled features, and predicting only the target grid cell without offset refinement.

Table 5 shows that the grounding head design has a large impact on localization accuracy. Directly regressing the target coordinate performs poorly, achieving only 26.8% average accuracy. This suggests that global coordinate regression does not effectively exploit the spatial structure of GUI screenshots. Predicting a target cell with a heatmap substantially improves accuracy to 57.3%, showing the benefit of preserving the patch-grid layout. Adding intra-cell offset prediction further improves accuracy to 64.0%, with only a small latency increase over the cell-only variant. These

Table 6: Ablation study of encoder fine-tuning.

Trainable Encoders	Avg. Acc. ↑	Time (ms) ↓
Frozen text & image encoders	15.4	**45.95**
Fine-tune image encoder only	49.7	47.27
Fine-tune text & image encoders	**64.0**	46.59

results support our cell-plus-offset formulation, which combines coarse spatial classification with fine-grained coordinate refinement.

B.3 Effect of Encoder Fine-tuning

We also study whether fine-tuning the pretrained text and image encoders is necessary for GUI grounding. We compare frozen encoders, image-only fine-tuning, and fine-tuning both encoders. This ablation evaluates how much domain adaptation is needed when using general-purpose pretrained encoders for GUI screenshots and plan-conditioned execution.

As shown in Table 6, keeping both encoders frozen leads to poor performance, with only 15.4% average accuracy. This indicates that general-purpose pretrained representations are not directly sufficient for plan-conditioned GUI grounding. Fine-tuning the image encoder improves the accuracy to 49.7%, suggesting that adapting visual features to GUI screenshots is particularly important. Fine-tuning both the text and image encoders achieves the best performance, reaching 64.0% average accuracy. The latency differences among the fine-tuning variants are small because they share the same inference architecture. These results show that end-to-end adaptation of both modalities is important for building an effective lightweight GUI executor.

C Additional Qualitative Case Studies

Figure 5 presents additional qualitative examples of PACE under the plan-conditioned execution setting. Each case shows the current screenshot, the next-step plan, and the action-coordinate prediction produced by PACE. The examples cover common GUI operations such as clicking text fields, selecting buttons, choosing menu items, and interacting with icons or navigation elements.

These case studies illustrate that many GUI interactions become local grounding problems once a concise next-step plan is provided. Instead of performing full task-level reasoning, PACE focuses on mapping the likely direction of the subsequent interaction to an action type and a target location. This supports its role as a fast executor in the two-speed framework, where semantic planning is handled by the main VLM and frequent low-level interactions are delegated to the lightweight model.

For ambiguous, complex, or potentially risky operations, the system does not rely on PACE alone and instead falls back to the main VLM planner, as described in method.

D Additional Analysis of Delegation and Domain Adaptation

D.1 How Automatic Delegation is Decided

In the automatic delegation setting, the main VLM planner is not fine-tuned. Instead, the delegation decision is obtained through zero-shot prompting. As shown in Figure 6, at each interaction step, the planner is required to output a structured response containing four fields: the current action, a next-step plan, a fast-executable decision, and the executable tool call for the current step. We parse the `Fast executable` field into a binary delegation flag z_t:

$$z_t = \begin{cases} 1, & \text{if the field is parsed as } \texttt{Yes}, \\ 0, & \text{if the field is parsed as } \texttt{No} \text{ or cannot be parsed.} \end{cases}$$

When $z_t = 1$, the next-step plan and the next screenshot are sent to PACE for fast execution. When $z_t = 0$, the system follows the slow path and uses the main VLM planner for the next decision step.

Next plan prompt

```
# Response format

Response format for every step:
1) Action: a short imperative describing what to do in the UI.

2) Next step plan: a brief description of what you will do in the next
     step.

3) Fast executable: Yes/No -- Use Yes only for one concrete low-risk
    mouse action (click/open/select/scroll) that requires neither text
    input nor confirmation. Use No for typing, confirmation-required
    actions, submit/save/send/delete/purchase actions, hotkeys,
    dragging, waiting, or any step requiring further observation.

4) A single <tool_call>...</tool_call> block containing only the JSON:
{"name": <function-name>, "arguments": <args-json-object>}.

Rules:
- Output exactly in the order: Action, Next step plan, Fast executable
    , <tool_call>.
- Be brief: one sentence for Action and one sentence for Next step
    plan.
- Do not output anything else outside those parts.
- If finishing, use action=terminate in the tool call.

# Example

Action: Click the File menu in the top-left corner.
Next step plan: Select the Open option from the File menu.
Fast executable: Yes
<tool_call>
{"name": "computer_use", "arguments": {"action": "left_click", "
    coordinate": [42, 68]}}
</tool_call>

# 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>
{tools_def}
</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>
```

Figure 6: Prompt template for generating the current action, next-step plan, and fast-executable delegation flag.

Next plan prompt

```
# Response format

Response format for every step:
1) Action: a short imperative describing what to do in the UI.
2) Next step plan: a brief description of what you will do in the next
     step.
3) A single <tool_call>...</tool_call> block containing only the JSON:
     {"name": <function-name>, "arguments": <args-json-object>}.

Rules:
- Output exactly in the order: Action, Next step plan, <tool_call>.
- Be brief: one sentence for Action and one sentence for Next step
    plan.
- Do not output anything else outside those parts.
- If finishing, use action=terminate in the tool call.

# Example

Action: Click the File menu in the top-left corner.
Next step plan: Select the Open option from the File menu.
<tool_call>
{"name": "computer_use", "arguments": {"action": "left_click", "
    coordinate": [42, 68]}}
</tool_call>

# 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>
""" + json.dumps(tools_def) + """
</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>
```

Figure 7: Prompt template for rule based prompt.

Table 7: Error attribution for failed delegated steps on OSWorld. Error types are non-exclusive, so percentages may not sum to 100%.

Planner	Plan Error	Delegation Error	Grounding Error
Qwen3-VL-4B	14.2%	67.2%	38.8%
Qwen3-VL-8B	10.1%	37.0%	22.9%
Qwen3-VL-30B-A3B	10.0%	26.3%	19.5%
Qwen3-VL-32B	6.7%	11.8%	11.5%

The prompt explicitly defines the scope of fast-executable actions. A step can be marked as `Yes` only when it is a concrete, low-risk mouse operation, such as clicking, opening, selecting, or scrolling, and when it requires neither text input nor confirmation. The planner is instructed to output `No` for typing, confirmation-required actions, submit/save/send/delete/purchase operations, hotkeys, dragging, waiting, or any step that requires further observation. This rule is intentionally conservative: if the delegation field is missing, malformed, or ambiguous, the system falls back to the main VLM rather than calling PACE.

This mechanism makes automatic delegation lightweight and model-agnostic, since it does not require additional training of the main VLM. However, it also makes the quality of delegation dependent on the planner's instruction-following and state-understanding ability. A stronger planner can usually produce more specific next-step plans and more reliable fast-executable decisions, while a weaker planner may delegate ambiguous or unsuitable steps. This explains why automatic delegation provides larger speedups with stronger planners while causing a more noticeable performance drop with the 4B planner.

D.2 Plan Quality and Error Attribution

To better understand the performance drop observed with the 4B planner, we further analyze failed delegated steps on OSWorld. Since a failure can be caused by multiple factors, we use non-exclusive error tags rather than assigning each failure to a single category. Specifically, we consider the following error types:

- **Plan error:** the next-step plan is inconsistent with the task goal or the current GUI state.
- **Delegation error:** the plan is partially reasonable, but the step is ambiguous, risky, or otherwise unsuitable for delegation to the fast executor.
- **Grounding error:** the plan is correct and delegatable, but PACE predicts the wrong action type or target coordinate.

Table 7 shows that the weaker 4B planner produces substantially more delegation-related failures than stronger planners. In particular, 67.2% of its failed delegated steps are associated with delegation errors, compared with only 11.8% for the 32B planner. This suggests that the performance drop in Table 2 is not solely caused by PACE's grounding ability, but is largely related to the reliability of the planner's next-step plan and fast-executable decision. When the planner delegates an ambiguous or unsuitable step, the fast executor is forced to act outside its intended operating regime.

The grounding error rate also decreases as the planner becomes stronger. This is expected because PACE is conditioned on the planner-provided next-step plan. A clearer and more specific plan reduces the semantic ambiguity of the grounding problem, while an underspecified or inaccurate plan can make even a visually simple step difficult to execute. Therefore, the two-speed framework benefits from a planner that can produce reliable local execution intents and conservative delegation decisions.

D.3 Sensitivity to OSWorld-specific Training Data

PACE is a compact executor with only 0.22B parameters. Although GUI interfaces share many common visual patterns, such as buttons, text fields, menus, icons, and navigation bars, the executor can still be sensitive to out-of-domain states and to the language style of the next-step plans produced

Table 8: Sensitivity to OSWorld-specific training data using Qwen3-VL-32B-Instruct as the planner.

Training Data	OSWorld Time ↓	OSWorld Score ↑
GUI-Actor only	90.86	23.25
GUI-Actor + OSWorld synthetic trajectories	70.43	28.90

by the main VLM planner. Therefore, we analyze the effect of the additional OSWorld-specific synthetic trajectories used during training.

As shown in Table 8, training only on public GUI-Actor data is insufficient for robust end-to-end deployment on OSWorld. Although GUI-Actor provides useful general GUI grounding supervision, the resulting executor performs poorly when directly used in the two-speed framework, leading to both higher total execution time and lower task score. This indicates that errors from an out-of-domain executor can accumulate in multi-step GUI tasks: a single incorrect fast action may lead the agent to a wrong state, require additional recovery steps, or cause task failure.

Adding a small number of OSWorld-specific synthetic trajectories substantially improves the end-to-end result, reducing total time from 90.86s to 70.43s and improving the score from 23.25 to 28.90. We attribute this improvement to two factors. First, the additional data adapts PACE to the visual distribution of OSWorld desktop applications. Second, it aligns the executor with the next-step plan style produced by the main VLM planner. Thus, the in-domain data does not make PACE a general-purpose GUI reasoner; rather, it calibrates a small fast executor to the states, actions, and plan-conditioned execution format encountered during deployment.

D.4 Sensitivity to OSWorld-specific Training Data

PACE is trained primarily on GUI grounding data, with an additional small set of OSWorld-specific synthetic trajectories for adapting the executor to the end-to-end environment. To analyze the importance of this in-domain data, we compare two training settings: training only on public GUI-Actor data, and training with the additional OSWorld-specific trajectories.

This comparison measures whether the fast executor can generalize from public GUI grounding data alone, and how much benefit is obtained from a small amount of in-domain OSWorld adaptation. Since OSWorld contains desktop environments, application-specific layouts, and task-dependent interaction patterns, in-domain trajectories can help align the executor with the action distribution encountered during end-to-end deployment.

E Additional Related Work

E.1 Large Vision-Language Models for GUI Agents

Recent GUI agents are typically built upon vision-language models (VLMs), often augmented with external tools, to perceive graphical interfaces and generate executable actions. Existing end-to-end GUI agents can be broadly categorized into two groups.

The first group consists of general-purpose foundation models, such as the GPT-5 series (OpenAI, 2026) and Qwen3-VL series (Team, 2025). These models acquire GUI interaction capabilities by incorporating GUI-related data into large-scale pre-training and post-training, enabling them to perform GUI tasks as part of their general multimodal abilities.

The second group focuses on GUI-specialized agents obtained through continued post-training from pretrained VLM backbones. Representative examples include UI-TARS (Qin et al., 2025), UI-TARS-1.5 (Seed, 2025a), UI-TARS-2 (Seed, 2025b), Mobile-Agent-v3.5 (Xu et al., 2026), UI-Venus (Gu et al., 2025), and UI-Venus-1.5 (Team et al., 2026b), which are typically initialized from strong pretrained models such as Qwen2.5-VL (Bai et al., 2025) or Qwen3-VL (Team, 2025) and further adapted with GUI-centric instruction and trajectory data. Despite their different training strategies, both categories aim to solve GUI tasks in an end-to-end manner, where high-level planning, visual perception, action grounding, and action generation are tightly coupled within a single large model. While this paradigm has achieved strong task-level performance, it also requires

repeated heavy VLM inference at every interaction step, leading to substantial latency in practical GUI agents.

E.2 Efficient Vision-Language Models

Recent work has explored efficient vision-language models to reduce the computational cost of multimodal inference. One line of work builds compact VLMs by pairing small language models with visual encoders, such as LLaVA-Phi (Zhu et al., 2024) and SmolVLM (Marafioti et al., 2025). These models aim to preserve general multimodal capabilities while reducing parameter count and deployment cost. Another line of work improves the efficiency of visual encoding. For example, FastVLM (Vasu et al., 2025) focuses on reducing the cost of processing visual tokens, which is often a major bottleneck in VLM inference. In the GUI domain, MobileVLM (Wu et al., 2024) studies vision-language modeling for intra- and inter-UI understanding, showing the importance of GUI-specific multimodal representations.

These efficient VLMs are closely related to our motivation, as they reduce the cost of multimodal perception and reasoning. However, they still generally follow the standard VLM inference paradigm, where the model observes the screenshot and produces an output at every interaction step. In contrast, PACE addresses GUI-agent latency from a system-level perspective. Rather than only making each VLM call cheaper, it reduces the number of heavy VLM calls by delegating simple and safe plan-conditioned execution steps to a lightweight discriminative executor. Therefore, efficient VLMs are complementary to our approach and can potentially serve as stronger or faster planners within the two-speed framework.

E.3 GUI Benchmarks and Evaluation

A growing number of benchmarks have been proposed to evaluate GUI agents from different perspectives. One line of work focuses on single-step GUI grounding, where the model is asked to locate the target UI element or coordinate given a screenshot and an instruction. ScreenSpot (Jurmu et al., 2008) evaluates GUI grounding across mobile, desktop, and web interfaces, while ScreenSpot-Pro (Li et al., 2025a) further emphasizes professional high-resolution computer-use scenarios with small targets and complex visual layouts. MMBench-GUI (Wang et al., 2025) provides a hierarchical evaluation framework for GUI agents across multiple platforms, including element grounding and higher-level agent skills.

Another line of benchmarks evaluates end-to-end GUI or web agents in interactive environments. OSWorld (Xie et al., 2024) evaluates multimodal agents on open-ended tasks in real computer environments, requiring agents to observe screenshots, reason over task progress, and execute multi-step actions. Mind2Web (Deng et al., 2023) and WebArena (Zhou et al., 2024) focus on web-based task action, while Android in the Wild (Ahvanooey et al., 2016) provides large-scale demonstrations for mobile device control.

Our evaluation follows both directions. We use GUI grounding benchmarks to measure the single-step execution ability of PACE, and use OSWorld to evaluate whether the fast executor can reduce end-to-end latency when integrated into VLM-based GUI agents. This allows us to assess PACE both as a standalone low-latency grounding model and as a component in a full GUI agent system.

E.4 Implementation Details of the Rule-based Delegation Baseline

We implement a rule-based delegation baseline to decide whether a next-step plan should be routed to the fast executor. This baseline is non-parametric and operates only on the textual next-step plan produced by the main planner. Given a plan, we first normalize the text by removing code blocks, bullet markers, redundant whitespace, and minor linguistic variations. The normalized plan is then checked by a set of conservative regular-expression rules. If any rule indicates that the step may be ambiguous, safety-sensitive, or difficult to execute reliably with a lightweight grounding model, the step is kept on the slow path and executed by the main VLM.

Specifically, the rule-based filter skips fast execution for terminal or empty plans, such as *done*, *none*, or *task complete*. It also rejects plans that involve critical UI decisions, including actions associated with confirmation, saving, deletion, overwriting, submission, sending, or task termination.

In addition, we do not delegate operations that require substantial keyboard or text interaction, such as typing, pasting, filling fields, entering text values, pressing hotkeys, using shortcuts, or invoking command palettes. Mouse operations that are difficult to recover from or require continuous control, such as dragging, dropping, or resizing, are also routed to the slow path.

For plans that pass these exclusion rules, the baseline allows fast execution only when the instruction corresponds to a simple and visually grounded GUI operation, such as clicking, selecting, choosing, opening, toggling, or switching a visible interface element. We also include a small exception for menu-selection steps related to export or save-as workflows. In such cases, selecting a concrete menu or submenu option can be safely treated as a local grounding problem, while the actual confirmation or file-writing step remains on the slow path. This exception allows the fast executor to handle common menu navigation actions without delegating irreversible decisions.

After a plan is accepted by the rule-based filter, the fast executor predicts an executable GUI action. We convert the predicted action into the environment action format through a lightweight parser. The parser supports common GUI operations, including left click, double click, right click, middle click, mouse move, drag, scroll, key press, hotkey, text input, wait, and task termination. For coordinate-based actions, predicted screen coordinates are converted to the coordinate convention required by the environment. In our OSWorld experiments, this conversion ensures that actions produced by the fast executor can be directly inserted into the original agent execution loop.

Overall, this rule-based baseline is intentionally conservative. It is designed to reduce unnecessary main-model calls for simple low-level interactions while avoiding delegation for steps that are likely to require semantic reasoning, precise textual input, or safety-sensitive decisions.