MIRAGE: Mobile Agents with Implicit Reasoning and Generative World Models

Zhichao Yang[1] **Yuanze Hu**[1] **Haojie Hao**[1] **Longkun Hao**[1]
Dongshuo Huang[2] **Hongyu Lin**[3] **Gen Li**[1] **Lanqing Hong**[4]
Yihang Lou[5,†] **Yan Bai**[5,†]

[1]Beihang University
[2]Northwestern Polytechnical University
[3]Institute of Software, Chinese Academy of Sciences
[4]National University of Singapore
[5]Peking University

Abstract

Mobile agents are increasingly expected to operate everyday applications from screenshots and language goals, where reliable control requires reasoning over screen affordances, multi-step navigation, and future state changes. Yet many agents externalize this computation as long textual thoughts, making interaction slower, supervision more costly, and deployment less efficient. We introduce MIRAGE (**M**obile agents with **I**mplicit **R**easoning **A**nd **G**enerative world mod**E**ls), a framework that learns continuous latent reasoning representations from visible textual thoughts. MIRAGE introduces an efficient latent-space learning procedure that transfers explicit reasoning into compact hidden states, allowing the agent to reason internally without decoding long rationales. It further brings a world-model perspective into mobile-agent training: the model's latent reasoning vectors are aligned with future screenshots, encouraging the agent to predict upcoming interface states in latent space before executing an action. This makes the hidden computation not only a compressed thought trace, but also a forward-looking representation of how the environment may change. At inference time, MIRAGE reasons in continuous latent space, reducing token generation while improving execution efficiency. On AndroidWorld, MIRAGE matches explicit-CoT SFT in the 4B ablation under a 3–5$\times$ lower decoded-token budget, and improves a comparable instruction-tuned baseline by 10.2 points; on AndroidControl, it improves action grounding with over 75% fewer generated tokens.

1 Introduction

As vision-language models improve, more mobile-agent systems now use them to execute mobile operations directly from screenshots and user instructions. Recent systems such as UI-TARS, MAI-UI, OS-ATLAS, and SeeClick train VLMs to understand GUI screens and produce taps, swipes, text entry, or navigation commands [1–4]. Yet turning a screen-level VLM into a reliable mobile agent still hinges on navigation reasoning: the model must track task progress, decide which screen to visit next, and anticipate how each action will change the interface. Figure 1 highlights this reasoning bottleneck. Current mobile agents often make this reasoning explicit through long thoughts or verbose action traces, which increases decoding time, context usage, and supervision cost. Their execution accuracy remains limited, suggesting that mobile agents need reasoning that preserves explicit-trace

Email: yzc0912@buaa.edu.cn
[†]Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

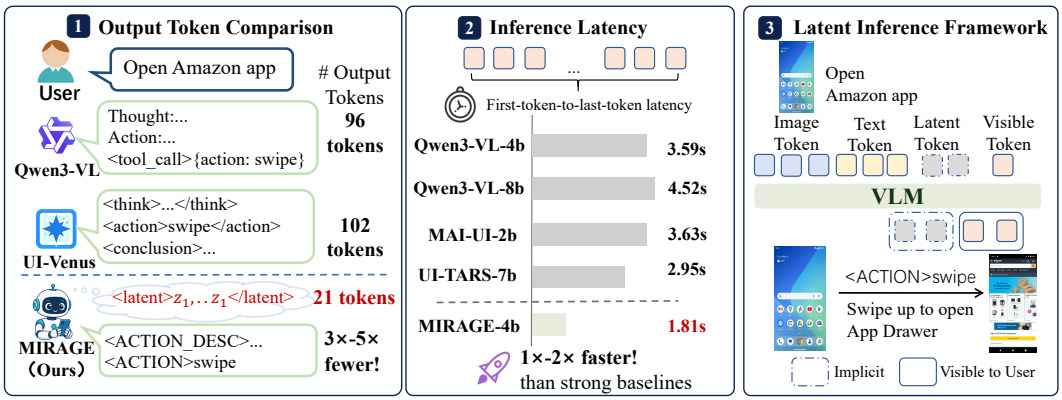

Figure 1: Inference-time comparison on a randomly sampled task and MIRAGE workflow: baselines emit verbose visible traces, while MIRAGE reasons with latent tokens and exposes concise actions.

capability while being cheaper to run under realistic deployment constraints, where every extra token slows the interaction and delays feedback during interactive control.

To address these issues, we propose MIRAGE, whose central idea is that a mobile agent should learn to "think forward" inside the model. Before emitting an action, the agent forms a compact internal representation of what the current screen affords, why an action is appropriate, and how the interface is likely to change afterwards. This reasoning is carried out in latent space rather than decoded as visible text, so MIRAGE avoids emitting intermediate thoughts and reduces both output tokens and first-token-to-last-token latency. MIRAGE uses a two-stage training procedure. In the first stage, it trains on explicit text traces so the backbone learns the mobile action space and how to express observation, rationale, and future-screen prediction. In the second stage, MIRAGE replaces the textual reasoning block with continuous latent reasoning slots and continues training, gradually teaching the model to move reasoning into latent space. At inference time, only action tokens are decoded; no rationale text is emitted and the interaction latency is substantially reduced.

To transfer explicit reasoning capability into latent space, MIRAGE introduces Approximate Parallel Latent Refinement (APLR), which refines all latent slots in parallel through K Jacobi-style rounds, approximating full sequential latent chain-of-thought: the first K slots provably match the serial rollout while the remaining tail error is bounded. To further enable the model to internalize future-state prediction, MIRAGE attaches a lightweight Q-Former world-model head that aligns the output latent representations with stop-gradient visual features of the next screenshot from the frozen vision encoder. This alignment teaches the agent to anticipate GUI state transitions, prevents latent representation collapse, and supplies dense supervision that compensates for APLR's bounded tail error. Crucially, the action cross-entropy loss and the next-frame feature-alignment loss are optimized jointly, so the same latent states become both action-discriminative and transition-predictive. This coupling lets MIRAGE retain CoT-level reasoning capacity while moving the intermediate computation out of the decoded text stream, yielding comparable explicit-reasoning capability at substantially lower inference cost.

MIRAGE yields strong results on established mobile-agent benchmarks. On AndroidWorld, MIRAGE improves task success by 10.2 percentage points over a comparable instruction-tuned baseline and matches explicit CoT in the 4B ablation under a 3–5$\times$ lower decoded-token cost. On AndroidControl, it improves action-grounding accuracy while reducing generated tokens by over 75%, demonstrating that latent reasoning and future-state alignment preserve explicit-reasoning capability with substantially leaner inference.

This paper makes three contributions:

- We present MIRAGE, a mobile agent that reasons entirely in latent space, producing far fewer output tokens and substantially lower inference latency while matching explicit-CoT task performance in the 4B ablation under a 3–5$\times$ lower decoded-token cost.

- We introduce APLR, a parallel Jacobi-style latent refinement procedure that approximates full serial latent chain-of-thought at a fraction of the training cost, with a provable bound on the approximation error of the tail slots.
- We introduce a Q-Former world-model head that aligns latent reasoning states with future-screenshot features in latent space, enabling the agent to predict upcoming GUI transitions and directly improving task capability.

2 Related Work

Mobile and GUI agents. GUI-agent benchmarks have evolved from web interaction to dynamic Android control, covering grounded shopping, open-web navigation, data-scaling studies, and online task completion [5–8]. Recent systems specialize VLMs for screenshot grounding, GUI actions, and mobile-device operation [1, 3, 4, 9]. Instead of extending grounding or planning pipelines, MIRAGE trains internal agent states by replacing visible reasoning traces with latent slots predictive of the next GUI state during action decoding and transition modeling.

Reasoning in language and vision-language agents. Visible CoT and ReAct-style traces improve reasoning and acting but expose long rationales and consume context [10, 11]. Other work internalizes computation through pause tokens, private thoughts, distillation, or continuous latent CoT [12–15]. MIRAGE brings implicit reasoning to mobile GUI control, where latent thoughts support action selection and transition understanding, using APLR to approximate serial latent refinement in a strictly causal triangular system rather than an equilibrium model.

World models and visual feature prediction. World models learn predictive dynamics for control, from compact latent simulators to latent-imagination agents [16, 17]. Joint-embedding vision objectives show that feature prediction can learn semantics without pixel generation [18, 19], while modal-alignment work highlights bottlenecks between visual and language representations [20]. Q-Former queries offer a lightweight cross-attention bottleneck [21]. Recent GUI world models predict future screens, sketches, or semantic states [22, 23]. MIRAGE instead uses future prediction to shape latent reasoning states, encouraging action-induced transition representations without generating pixels or future text at inference time.

3 Method

3.1 Problem Formulation: From Explicit-Thought to Latent-Thought Mobile Agents

At interaction step t, a mobile GUI agent observes the current screenshot x_t, a user instruction u, and an interaction history $h_t = (a_{<t}, x_{<t})$. It outputs an action a_t, such as tapping a coordinate, typing text, scrolling, or navigating back. We serialize actions as text tokens so that a VLM can model mobile control as conditional generation:

$$o_t = (x_t, u, h_t), \qquad p_\theta(a_t \mid o_t). \tag{1}$$

Mobile GUI agents face a critical trade-off between reasoning capability and deployment efficiency. Explicit chain-of-thought improves multi-step decision making because the model can verbalize observations, rationales, and predicted state transitions before acting; however, this reasoning is paid for with visible tokens. Many explicit-thought mobile agents first generate a readable reasoning trace τ_t and then generate the action. This factorization can be written as

$$p_\theta(\tau_t, a_t \mid o_t) = \prod_{i=1}^{|\tau_t|} p_\theta\left(\tau_t^{(i)} \mid \tau_t^{(<i)}, o_t\right) \cdot \prod_{j=1}^{|a_t|} p_\theta\left(a_t^{(j)} \mid \tau_t, a_t^{(<j)}, o_t\right). \tag{2}$$

The intermediate thought τ_t is useful, but it is also a sequence of discrete text tokens: it consumes context budget, increases decoding latency, and usually requires human-written or synthetic rationale supervision. This makes explicit CoT attractive as a training scaffold but costly as an inference-time interface for mobile deployment.

To resolve this tension, MIRAGE formulates mobile control as a latent-thought generation problem. Instead of removing reasoning, we internalize it: the model keeps a structured computation budget, but

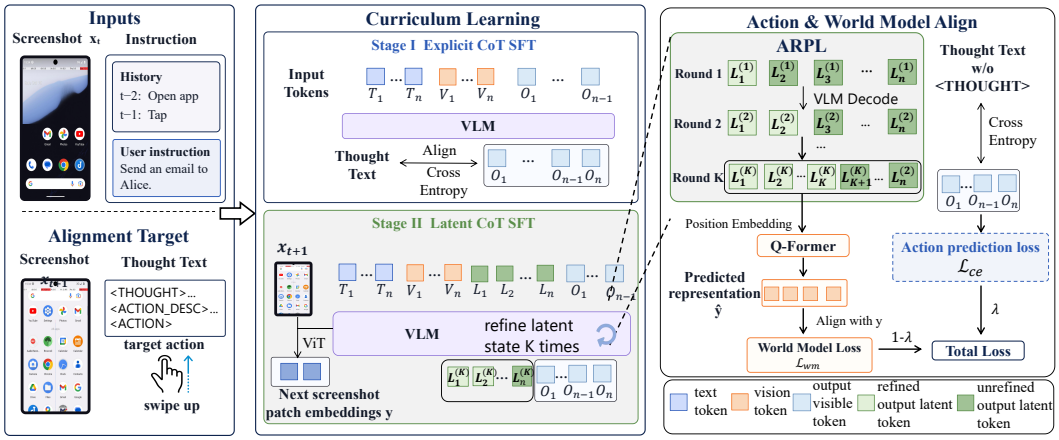

Figure 2: MIRAGE pipeline. Stage 1 learns explicit mobile thoughts and action formatting. Stage 2 replaces the thought text with latent slots, refines them with APLR, and trains a Q-Former world model to align latent states with next-frame visual features.

the intermediate computation is carried by hidden states rather than text. We introduce N continuous latent variables $\mathbf{z}_t = (z_{t,1}, \ldots, z_{t,N})$, $z_{t,i} \in \mathbb{R}^d$, in the decoder context:

$$p_\theta(a_t \mid o_t) = \int p_\theta(a_t \mid \mathbf{z}_t, o_t) q_\theta(\mathbf{z}_t \mid o_t) d\mathbf{z}_t. \tag{3}$$

Unlike explicit CoT, $\mathbf{z}_t$ does not live in the vocabulary and is never detokenized. Its inference distribution q_θ is computed implicitly by decoder hidden states, so latent-thought training does not need rationale-text supervision. This formulation gives four practical advantages: it preserves context by removing explicit rationale tokens, speeds up inference by shortening the decoded sequence, retains multi-step reasoning capacity through latent slots, and exposes the hidden reasoning state to auxiliary regularization such as next-frame prediction. In this sense, latent thoughts bridge the gap between CoT-style reasoning strength and efficient mobile-agent deployment, motivating the Approximate Parallel Latent Refinement(APLR) refinement and world-model regularization developed next.

3.2 Training Latent Chain-of-Thought via Approximate Parallel Latent Refinement

3.2.1 Structured Thought Data and Latent Insertion

Our training data uses a structured thought format with three mobile-agent reasoning dimensions:

```
<THOUGHT> [observation] [rationale] [predict] <ACTION_DESC> ...
<ACTION> ...
```

The `observation` field describes the visible screen state, `rationale` explains why a particular operation should be taken, and `predict` describes the expected next-screen transition. Explicit-thought warmup uses this structured text as ordinary next-token supervision. Latent-thought training then replaces the entire `<THOUGHT>` block with N latent tokens. The token sequence becomes

$$[\text{ctx}]\,;[\text{start}]\,;\underbrace{\langle\text{lat}\rangle,\ldots,\langle\text{lat}\rangle}_{N\text{ slots}}\,;[\text{end}]\,;\langle\text{ACTION_DESC}\rangle\cdots\langle\text{ACTION}\rangle. \tag{4}$$

The latent slots occupy positions $p_1 < \cdots < p_N$ and share the learned initializer e_{lat}.

3.2.2 From Serial Masking to Approximate Parallel Latent Refinement

Original Coconut-style latent reasoning [15] replaces visible reasoning steps with continuous hidden states and computes those states serially; Appendix D gives the full formalization. Our APLR keeps the same latent-thinking motivation but changes the execution pattern: rather than advancing through thought steps one by one, it treats the thought block as N slots and refines all slots in parallel rounds.

With APLR, latent reasoning becomes a small number of synchronous refinement passes instead of a long serial rollout. Conceptually, this lets the model keep an internal multi-step computation budget while making training cost depend on the refinement depth K, not directly on the latent length N.

With fixed non-latent context $c = o_t$ and latent positions $p_1 < \cdots < p_N$, the serial target states are

$$s_i = G_i(s_1, \ldots, s_{i-1}; c), \qquad i = 1, \ldots, N, \tag{5}$$

where G_i is the hidden state at position $p_i - 1$ when the previous latent states have already been written into the sequence. This is a forward-substitution procedure over a causal triangular system. It is accurate, but expensive: fully refining all N latent tokens requires one full decoder pass per latent update, plus a final pass for action logits.

We use APLR to eliminate the need for N serial forward passes in traditional latent CoT training. The key observation is that causal attention makes the latent system strictly triangular: z_i cannot depend on itself or on any future $z_{j>i}$. Therefore, instead of filling the slots one by one, APLR performs Jacobi-style rounds:

$$z_i^{(k+1)} = G_i\Big(z_1^{(k)}, z_2^{(k)}, \ldots, z_{i-1}^{(k)}; c\Big), \qquad i = 1, \ldots, N, \quad k = 0, \ldots, K-1. \tag{6}$$

All right-hand sides use old round-k values, so one full forward updates all slots. In practice, we use a small refinement budget, $K = 3$ by default.

APLR has a precise relationship to the serial refinement in (5). After K parallel rounds, the first K latent slots exactly match the serial solution, $z_i^{(K)} = s_i$ for $i \le K$. Compared with refining all latent tokens one by one, the unrefined tail slots retain a structured residual

$$\delta^{(K)} \approx A^K \delta^{(0)}, \tag{7}$$

where A is a strictly lower-triangular Jacobian. Appendix E proves that after K APLR rounds, the first K latent slots exactly recover the serial solution, and then derives the tail-error form in (7). In implementation, early bootstrap passes may run without gradient, while the first pass can retain gradient through e_{lat} for stability. Before the final gradient-enabled pass, we rebuild the input embeddings with a mask: latent positions use detached bootstrap values, while non-latent positions retain their original gradient lineage to the vision tower and token embeddings.

The final APLR pass is trained with standard next-token CE over serialized action tokens:

$$\mathcal{L}_{\text{ce}} = -\sum_{j=1}^{|a_t|} \log p_\theta\Big(a_t^{(j)} \mid \mathbf{z}_t^{(K)}, a_t^{(<j)}, o_t\Big). \tag{8}$$

When $K \ll N$, the tail latents $z_{K+1:N}^{(K)}$ retain approximation error relative to a trainer that refines every latent token serially. In our structured thought format, the `predict` dimension concerns how the screen will change after the action and typically lies late in the thought sequence. Action CE only penalizes tail-error directions that affect the next action tokens; it does not directly supervise errors that matter for environment prediction. This motivates the world-model objective.

3.3 World Model: Q-Former Next-Frame Alignment

The `predict` field describes future screen semantics after a mobile action. Unlike GUI world models that generate pixels or sketches [22], our Q-Former head directly regularizes latent reasoning states to predict future GUI features, forcing latent slots to encode environment dynamics rather than only current-screen action correlations. Tail-error directions that harm future-screen prediction therefore receive direct gradients; Appendix F formalizes this with a local second-order argument.

Because the training records are sequential, most non-terminal step-t examples provide the next screenshot x_{t+1}. We use it as feature supervision, not as a pixel-generation target: x_{t+1} is passed through the VLM's own frozen vision tower, yielding a stable next-frame feature target that does not drift with the world-model head.

We gather the final-pass hidden states at latent positions to obtain latent CoT hidden states

$$C_t = \text{Gather}\Big(H_t^{(K)}, p_1, \ldots, p_N\Big) \in \mathbb{R}^{L \times H}, \tag{9}$$

where each row is one latent thought vector after APLR. The next screenshot is first converted by the VLM image processor into visual patches. After the VLM's spatial merge operation, each target patch has both a feature vector and a 2D grid coordinate. We write the target features as

$$V_{t+1} = \mathrm{sg}(f_{\mathrm{vis}}(x_{t+1})) = [v_1, \ldots, v_M] \in \mathbb{R}^{M \times H_v}, \tag{10}$$

where $\mathrm{sg}(\cdot)$ denotes stop-gradient. The stop-gradient prevents collapse through the target branch, no image decoder is required, and the target lives on the same visual manifold as the backbone features.

For each next-frame patch j with grid coordinate (r_j, c_j), we build a learnable spatial query

$$q_j = e^{\mathrm{row}}_{r_j} + e^{\mathrm{col}}_{c_j} \in \mathbb{R}^{d_q}. \tag{11}$$

This says, in simple terms, "predict the feature at row r_j and column c_j of the next screen." The separable row/column design gives a stable 2D positional prior even when mobile screenshots have different resolutions and therefore different patch grids.

The BLIP-2-style Q-Former aligner [21] self-attends over queries, cross-attends to C_t, and projects each query output into the VLM feature space:

$$U_t = \mathrm{QFormer}_{\phi}(Q_t, C_t), \qquad \hat{V}_{t+1} = W_{\mathrm{vis}} U_t. \tag{12}$$

The output $\hat{V}_{t+1} = [\hat{v}_1, \ldots, \hat{v}_M]$ is therefore a predicted next-frame representation, one vector per next-frame patch. The default objective is masked per-patch cosine distance:

$$\mathcal{L}_{\mathrm{wm}} = \frac{1}{|\mathcal{M}|} \sum_{(b,j) \in \mathcal{M}} \left(1 - \frac{\langle \hat{v}_{b,j}, v_{b,j} \rangle}{\|\hat{v}_{b,j}\|_2 \|v_{b,j}\|_2} \right). \tag{13}$$

The mask $\mathcal{M}$ keeps only valid patches from non-terminal latent samples; cosine is default, with MSE used for ablations.

3.4 Two-Stage Training Pipeline

We use a curriculum learning strategy to make latent reasoning learnable rather than asking the model to discover a continuous thought space from scratch. Stage 1 warm-ups the model on explicit structured thoughts to learn action formatting and observation–rationale–prediction reasoning patterns. Specifically, the full `<THOUGHT>` trace is exposed as text supervision and the VLM is trained with standard next-token CE over the structured thought, action description, and action tokens. Stage 2 distills this explicit reasoning process into compact latent slots: the `<THOUGHT>` block is replaced by the latent sequence in (4), APLR refines the latent states ($K = 3$ by default), the collator loads next screenshots, and the world model regularizes the latent slots through Q-Former next-frame feature alignment. The joint objective is

$$\mathcal{L} = \lambda \mathcal{L}_{\mathrm{ce}} + (1 - \lambda) \mathcal{L}_{\mathrm{wm}}, \qquad \lambda \in (0, 1). \tag{14}$$

We use $\lambda = 0.8$ by default. Figure 2 summarizes the pipeline, and Appendix G gives pseudocode. At inference time, the agent only performs latent substitution and greedy action decoding; the Q-Former head is used only for training-time representation shaping [24].

4 Experiments

4.1 Setup

Backbones. We fine-tune two vision-language models from the Qwen3-VL family [25]: **Qwen3-VL-4B-Instruct** (4B parameters) and **Qwen3-VL-8B-Instruct** (8B parameters). Both backbones share the same action-serialization vocabulary, latent-slot token format, and Q-Former world-model head. Their latent computation budgets differ: the main Qwen3-VL-4B setting uses 9 latent slots with 3 APLR refinement passes, while the Qwen3-VL-8B setting uses 6 latent slots with 3 refinement passes.

Evaluation benchmarks. We evaluate on two standard mobile-agent benchmarks. **AndroidControl** [7] provides paired high-level and low-level instructions with ground-truth action sequences, allowing separate measurement of instruction-following exact match (EM) and action accuracy. **AndroidWorld** [8] is a dynamic, on-device benchmark spanning 116 real-world task instances across 20 Android apps, measuring end-to-end task completion rate under live Android dynamics.

Table 1: AndroidControl results. EM = exact match, Action Acc. = action accuracy, and Tokens = average generated tokens per step. External rows use reported Type/EM metrics; green parentheses show MIRAGE gains over size-matched Qwen3-VL-Instruct baselines. Best primary metrics and lowest Tokens are in **bold**.

Model	Size	Low-Level			High-Level		
		EM	Action Acc.	Tokens	EM	Action Acc.	Tokens
GPT-4o	–	20.8	66.3	–	19.4	74.3	–
GUI-R1-3B	3B	64.4	83.7	–	46.6	58.0	–
UI-R1-3B	3B	45.4	57.9	–	57.4	72.5	–
GUI-R1-7B	7B	66.5	85.2	–	51.7	71.6	–
ShowUI-2B	2B	74.83	81.68	31.57	59.82	74.18	34.39
MAI-UI-2B	2B	64.70	74.17	32.87	64.30	67.3	32.78
UI-Venus-Navi-7B	7B	**85.09**	93.05	103.66	**75.2**	**86.14**	96.36
UI-TARS-7B-SFT	7B	76.28	82.16	91.80	62.12	74.37	91.38
Qwen3-VL-4B-Instruct	4B	68.48	75.15	115.67	54.58	69.39	116.00
Qwen3-VL-8B-Instruct	8B	77.66	82.54	79.86	60.78	74.66	80.52
MIRAGE-4B	4B	77.59 (↑13.30%)	91.09 (↑21.21%)	18.92 (↓83.64%)	64.43 (↑18.05%)	75.67 (↑9.05%)	20.61 (↓82.23%)
MIRAGE-8B	8B	83.75 (↑7.84%)	**94.62** (↑14.64%)	**18.01** (↓77.45%)	72.35 (↑19.04%)	82.83 (↑10.94%)	**19.89** (↓75.30%)

Table 2: AndroidWorld results. SR = task success rate; Avg. Steps are per task, and Avg. Tokens are per step. Green parentheses show MIRAGE gains over size-matched Qwen3-VL-Instruct baselines; red marks higher step count. Best SR and lowest averages are in **bold**.

Model	Size	SR	Avg. Steps	Avg. Tokens
GPT-4o	–	30.6	–	–
Ferret-UI Lite	3B	28.0	–	–
UI-TARS	7B	33.0	–	–
ShowUI-2B	2B	7.8	18.9	33.0
MAI-UI-2B	2B	46.5	15.0	116.0
UI-Venus-Navi-7B	7B	42.2	13.3	92.0
UI-TARS-7B-SFT	7B	33.6	17.3	104.0
Qwen3-VL-4B-Instruct	4B	42.9	14.3	103.0
Qwen3-VL-8B-Instruct	8B	47.6	**12.6**	108.0
MIRAGE-4B	4B	52.6 (↑22.6%)	14.2 (↓0.7%)	31.0 (↓69.9%)
MIRAGE-8B	8B	**57.8** (↑21.3%)	13.7 (↑8.7%)	**27.0** (↓75.0%)

Baselines. We compare against size-matched **Qwen3-VL-4B/8B-Instruct** backbones [25], a general multimodal baseline **GPT-4o** [26], reinforcement-tuned GUI agents **GUI-R1/UI-R1** [27, 28], and recent GUI-agent systems including **ShowUI**, **MAI-UI**, **UI-Venus-Navi**, **UI-TARS-7B-SFT**, and **Ferret-UI Lite** across AndroidControl and AndroidWorld [1, 2, 29–31].

4.2 Main Results

AndroidControl. Table 1 shows that MIRAGE improves the size-matched Qwen3-VL-Instruct baselines while producing much shorter action outputs. On the low-level split, MIRAGE-4B improves EM from 68.48 to 77.59 and action accuracy from 75.15 to 91.09, while reducing the average generation length from 115.67 to 18.92 tokens. MIRAGE-8B shows the same pattern, increasing low-level EM from 77.66 to 83.75 and action accuracy from 82.54 to 94.62 with 18.01 tokens per step. On the high-level split, MIRAGE-4B improves EM/action accuracy by 9.85/6.28 percentage points over Qwen3-VL-4B-Instruct, and MIRAGE-8B improves them by 11.57/8.17 points over Qwen3-VL-8B-Instruct. These gains suggest that replacing verbose visible thoughts with latent reasoning can improve grounding and action selection without increasing the decoded token budget.

AndroidWorld. Table 2 evaluates the same models in a dynamic on-device setting. MIRAGE raises AndroidWorld SR from 42.9 to 52.6 for 4B and from 47.6 to 57.8 for 8B, while reducing average tokens from 103.0/108.0 to 31.0/27.0. The 8B model uses slightly more steps than its Qwen3-VL baseline, so the gain does not simply come from shorter trajectories. Among retained specialized GUI

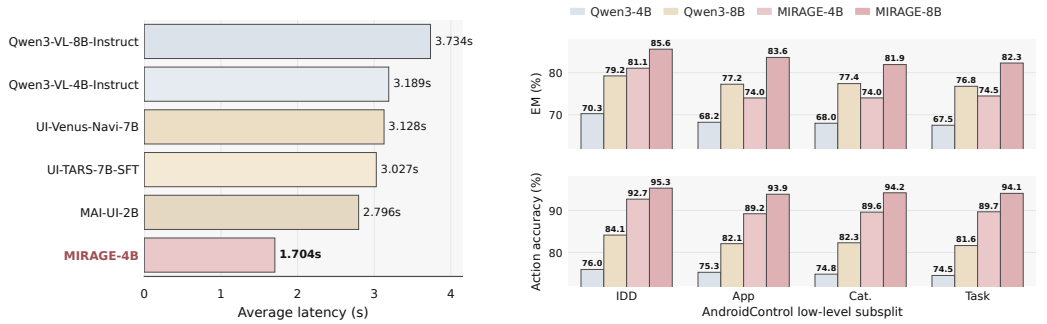

Figure 3: **Left:** Average latency from the first generated token to the final generated token. MIRAGE-4B produces the shortest decoded sequence latency among the compared models. **Right:** EM and action accuracy on the IDD, app-unseen, category-unseen, and task-unseen subsplits of AndroidControl low-level.

agents, MIRAGE-8B gives the highest AndroidWorld SR with far fewer generated tokens. Together, the two benchmarks show that latent reasoning and world-model training improve task effectiveness while keeping outputs compact.

Figure 3 further analyzes efficiency and robustness. The left panel measures first-to-last-token latency; MIRAGE-4B is fastest, consistent with the token reductions in Tables 1 and 2. The right panel breaks AndroidControl low-level results into IDD, app-unseen, category-unseen, and task-unseen subsplits. MIRAGE remains strong across subsplits, especially on action accuracy, suggesting that Table 1 is not driven by a single easy split.

4.3 Ablation Study

We ablate the three core components of MIRAGE—latent CoT slots, APLR parallel refinement, and the Q-Former world-model head—using Qwen3-VL-4B-Instruct on AndroidWorld.

Ablation analysis. Figure 4 reports cross-entropy loss for the same Qwen3-VL-4B training variants summarized in Table 3. For the latent settings, we allocate 9 latent slots; in the serial latent-CoT reference, the observation, rationale, and prediction fields each correspond to three latent slots and are masked through a stepwise curriculum. The sharp change after the first epoch is expected rather than a failure mode: training switches from explicit-thought warmup to latent-CoT training at that point, and the learning rate is reset with the new stage. After this transition, APLR without the world-model objective exhibits a higher loss than the serial latent-CoT reference, consistent with the fact that parallel approximation leaves tail latent states less directly supervised by action CE alone. Adding the Q-Former world-model objective makes the APLR curve track the serial latent-CoT trend more closely, suggesting that next-frame feature alignment supplies useful gradients for the transition-predictive parts of the latent state.

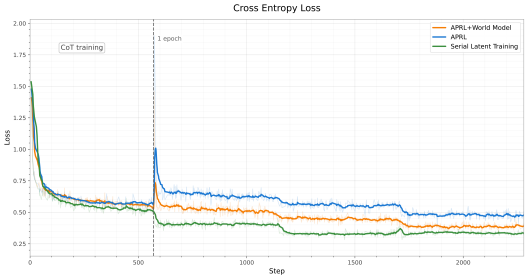

Figure 4: Cross-entropy training loss for the Qwen3-VL-4B variants listed in Table 3.

Table 3 shows the same pattern in final AndroidWorld SR. Action-only SFT performs worse than the base model, indicating that removing thought supervision without replacing it by internal computation can harm interactive decision making. Explicit CoT SFT and full MIRAGE-4B both reach 52.6 SR, showing that latent reasoning can match explicit CoT while avoiding decoded rationale tokens. Serial latent CoT preserves much of this benefit (50.9 SR), and APLR without the world-model objective reaches 48.2; adding the Q-Former world-model objective restores the explicit-CoT-level result while keeping reasoning latent at inference time.

Table 3: Component ablation on AndroidWorld (Qwen3-VL-4B, SR %).

Variant	Lat.	APLR	WM	SR
Base model	×	×	×	42.9
Action-only SFT	×	×	×	31.0
Explicit CoT SFT	×	×	×	**52.6**
Latent CoT, serial	✓	×	×	50.9
APLR only	✓	✓	×	48.2
MIRAGE-4B	✓	✓	✓	**52.6**

We study the sensitivity of MIRAGE to the number of latent slots L, APLR refinement passes K, and the loss balance λ_{ce} on AndroidWorld. Table 4 shows that the latent computation budget matters.

Table 4: AndroidWorld ablations over latent-slot budget, APLR refinement passes, and loss balance. SR is task success rate (%). Unless otherwise noted, $\lambda_{\text{ce}} = 0.8$.

Model	Latent slots	Ref. passes	λ_{ce}	SR (%)
MIRAGE-8B	6	2	0.8	46.6
MIRAGE-8B	6	3	0.8	**57.8**
MIRAGE-4B	9	3	0.8	52.6
MIRAGE-4B	3	3	0.8	32.8
MIRAGE-4B	9	3	0.9	48.3

For MIRAGE-8B, increasing APLR refinement from two to three passes improves AndroidWorld SR from 46.6 to 57.8, suggesting that the third pass substantially reduces harmful tail-latent error. For MIRAGE-4B, reducing the latent budget from 9 to 3 slots lowers SR from 52.6 to 32.8, showing that the mobile-agent thought state needs enough continuous capacity to represent observation, rationale, and future-screen prediction. Finally, increasing λ_{ce} from 0.8 to 0.9 decreases SR from 52.6 to 48.3, indicating the value of retaining sufficient weight on next-frame supervision.

4.4 Latent Reasoning Visualization

We analyze a MIRAGE-8B checkpoint with six latent slots on the AndroidControl IDD split; Appendix C provides full t-SNE and per-slot action projections. Figure 5 shows that latent training does not collapse to a single undifferentiated representation. Before slot-centering, slots 1–2, 3–4, and 5–6 occupy three separated regions, corresponding to the observation, rationale, and prediction dimensions of the structured thought. After subtracting the slot mean, examples remain organized by action type: open-app, swipe, type, and tap states form distinguishable regions in the centered latent space. These visualizations indicate that MIRAGE learns both position-specific reasoning subspaces and action-discriminative latent states.

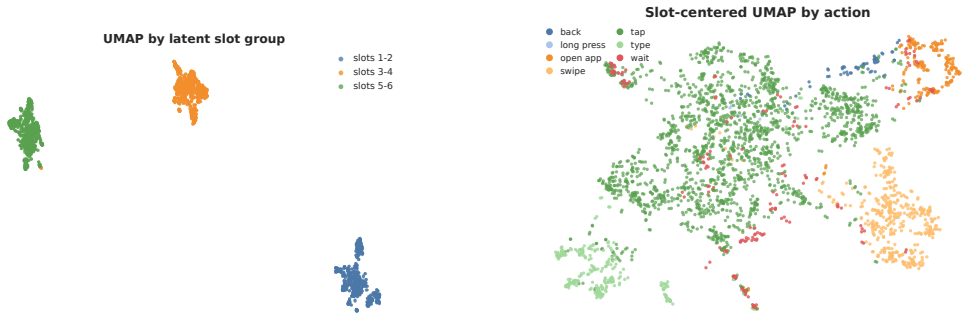

Figure 5: **Left:** UMAP by latent slot group. **Right:** slot-centered UMAP by action type after subtracting per-slot means.

5 Limitations, Broader Impact, and Conclusion

MIRAGE replaces visible rationales with APLR-refined latent slots regularized by next-frame features, preserving explicit-CoT capability at much lower decoding cost. Limitations include supervised-only training, feature-level world modeling, next-frame supervision, and the need for privacy and action safeguards before deployment.

Funding

This work received no external funding.

Competing Interests

The authors declare no competing interests.

References

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

[2] Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, et al. Mai-ui technical report: Real-world centric foundation gui agents. *arXiv preprint arXiv:2512.22047*, 2025.

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

[4] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, 2024.

[5] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

[6] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

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

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

[9] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024.

[10] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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

[12] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

[13] Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

[14] Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

[15] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

[16] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

[17] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[18] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15619–15629, 2023.

[19] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *arXiv preprint arXiv:2404.08471*, 2024.

[20] Yuanze Hu, Zhaoxin Fan, Xinyu Wang, Gen Li, Ye Qiu, Zhichao Yang, Wenjun Wu, Kejian Wu, Yifan Sun, Xiaotie Deng, et al. Tinyalign: Boosting lightweight vision-language models by mitigating modal alignment bottlenecks. *arXiv preprint arXiv:2505.12884*, 2025.

[21] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023.

[22] Dezhao Luo, Bohan Tang, Kang Li, Georgios Papoudakis, Jifei Song, Shaogang Gong, Jianye Hao, Jun Wang, and Kun Shao. Vimo: A generative visual gui world model for app agents. *arXiv preprint arXiv:2504.13936*, 2025.

[23] Yilin Cao, Yufeng Zhong, Zhixiong Zeng, Liming Zheng, Jing Huang, Haibo Qiu, Peng Shi, Wenji Mao, and Wan Guanglu. Mobiledreamer: Generative sketch world model for gui agent. *arXiv preprint arXiv:2601.04035*, 2026.

[24] Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

[25] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

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

[27] Run Luo, Lu Wang, Wanwei He, Longze Chen, Jiaming Li, and Xiaobo Xia. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*, 2025.

[28] Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Pengxiang Zhao, Guangyi Liu, et al. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 17608–17616, 2026.

[29] Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508, 2025.

[30] 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.

[31] Zhen Yang, Zi-Yi Dou, Di Feng, Forrest Huang, Anh Nguyen, Keen You, Omar Attia, Yuhao Yang, Michael Feng, Haotian Zhang, et al. Ferret-ui lite: Lessons from building small on-device gui agents. *arXiv preprint arXiv:2509.26539*, 2025.

[32] Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Guozhi Wang, Dingyu Zhang, Shuai Ren, and Hongsheng Li. Amex: Android multi-annotation expo dataset for mobile gui agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2138–2156, 2025.

[33] Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, et al. Memgui-bench: Benchmarking memory of mobile gui agents in dynamic environments. *arXiv preprint arXiv:2602.06075*, 2026.

[34] Guangyi Liu, Gao Wu, Congxiao Liu, Pengxiang Zhao, Liang Liu, Mading Li, Qi Zhang, Mengyan Wang, Liang Guo, and Yong Liu. Memgui-agent: An end-to-end long-horizon mobile gui agent with proactive context management. *arXiv preprint arXiv:2606.19926*, 2026.

A Additional Method Details

Q-Former target construction. The world-model target is built online during training. For each sample with a valid next frame, the collator loads x_{t+1} and the base VLM vision stack extracts post-spatial-merge patch features. The Q-Former aligner never receives these target features as input; it receives only latent CoT hidden states and row/column query embeddings.

No target leakage. The world-model predictor receives only the final latent hidden states from the current observation and history. Next-frame pixels are used only on the detached target side of the feature-alignment loss, and missing next frames are masked out.

B Training Data and Schedule

Training data. We train on two sources of mobile-interaction data. First, we select 40K high-quality interaction steps from *AMEX* [32], the Android Multi-annotation Expo dataset, which provides high-resolution screenshots annotated with GUI-element groundings and step-wise action chains across diverse Android applications. Second, we collect self-explored trajectories on AndroidWorld [8] by running **Qwen3-30B-VL-Instruct** [25] as a free-exploration policy on device. We then convert the explored trajectories into QA-style mobile-control examples and sample approximately 10K additional interaction steps, extending the training distribution beyond the static AMEX corpus with live Android app states.

Training procedure. Training proceeds in two stages, totalling 4 epochs, with the learning rate reset when switching between stages. In Stage 1, we fine-tune the backbone for 1 epoch on explicit chain-of-thought demonstrations so the model learns action formatting and structured observation–rationale–prediction reasoning. In Stage 2, we switch to the latent-CoT objective and train for 3 epochs, replacing the explicit thought block with N latent slots, applying APLR refinement ($K = 3$ passes by default), and enabling the Q-Former world-model head. The main configuration uses $L = 6$ latent slots; component ablations additionally evaluate the 4B model with $L = 9$ slots. Unless otherwise specified, $\lambda_{\text{ce}} = 0.8$.

We train the backbone with LoRA applied to all attention and MLP projection modules ($r = 128$, $\alpha = 256$, dropout $= 0.01$, no bias); the full-rank adapter weights for the vision–language merger

layers (`visual.merger.linear_fc1/2` and three `deepstack_merger` pairs) are saved separately. The ViT image encoder is kept frozen throughout both stages. The Q-Former head is trained from scratch with a dedicated learning rate 5×10^{-4} (approximately $50\times$ the backbone rate) to account for random initialization. Backbone learning rates are 3×10^{-5} for the 4B model and 1×10^{-5} for the 8B model, with a cosine schedule, warmup ratio 0.03, and weight decay 0.01. The learning rate is reset to its Stage 1 value at the start of Stage 2.

C Complete Latent Visualization Diagnostics

This appendix reports the full latent-slot diagnostic views used to support Section 4.4. The analysis is run on a MIRAGE-8B checkpoint evaluated on the AndroidControl IDD split. The diagnostic dump contains 1,012 examples and 3,036 latent vectors after grouping the six latent slots into three adjacent pairs: slots 1–2, slots 3–4, and slots 5–6. This grouping matches the main-paper visualization, where the paired slots form three separated positional clusters.

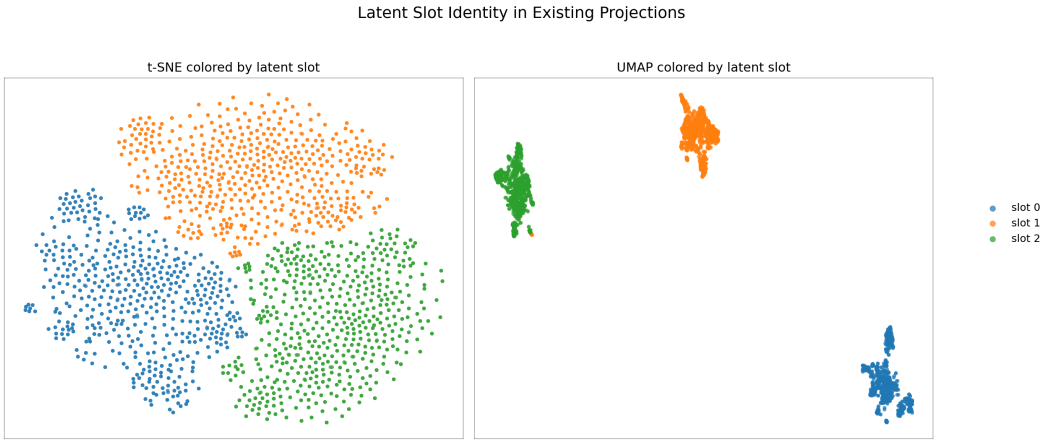

Figure 6: Latent slot identity in existing projections. The left panel colors a t-SNE projection by latent slot group, while the right panel colors a UMAP projection by the same group. Both projections show that adjacent slot pairs form three well-separated regions. This indicates that the latent representation contains a strong slot-position component: early, middle, and late latent slots occupy distinct subspaces before any slot-centering operation is applied.

Figures 6 and 7 jointly demonstrate that MIRAGE's latent reasoning does *not* collapse in the hidden space. Figure 6 shows that the three slot groups occupy well-separated yet internally high-variance regions in both t-SNE and UMAP, ruling out the degenerate case where all slots converge to the same vector. Figure 7 reveals that action-type sub-clusters (`open_app`, `swipe`, `type`) appear reproducibly inside every slot group, indicating that each slot carries semantically coherent and input-sensitive information rather than emitting a fixed template. Taken together, these results show that MIRAGE compresses multi-step GUI reasoning into latent slots that are both structurally organized and rich in task-relevant content, without collapsing to a trivial fixed-point solution.

D Formal Derivation of Original Coconut

This section formalizes the original Coconut-style latent reasoning procedure [15] that motivates our APLR approximation. Coconut starts from supervised examples with a context c and a visible chain of reasoning split into M ordered chunks $\rho_{1:M}$, followed by a final answer or action a. For mobile agents, c corresponds to the current screenshot, instruction, and interaction history, while a is the serialized GUI action.



Figure 7: Per-slot action projections. Each subplot projects one slot group and colors points by the ground-truth action type. These plots ask whether action semantics are visible inside each slot group without first removing the slot mean.

Explicit chain-of-thought objective. The ordinary explicit-CoT likelihood factorizes as

$$p_\theta(\rho_{1:M}, a \mid c) = \prod_{i=1}^{M} p_\theta(\rho_i \mid c, \rho_{<i}) \cdot p_\theta(a \mid c, \rho_{1:M}), \tag{15}$$

where each factor over a chunk denotes the product of token probabilities inside that chunk. Stage 0 of Coconut is equivalent to minimizing the standard next-token loss on the full sequence $(c, \rho_{1:M}, a)$.

Continuous-thought substitution. Coconut introduces boundary tokens $\langle \text{bot} \rangle$ and $\langle \text{eot} \rangle$ for a latent reasoning span. Let $H_\theta(S)$ be the final-layer hidden states produced by the decoder for a mixed sequence S containing both token embeddings and continuous vectors. Let $F_\theta(S)$ denote the hidden state at the last position of S, optionally followed by a projection Π into the input-embedding space:

$$F_\theta(S) = \Pi\, H_\theta(S)_{|S|}. \tag{16}$$

At curriculum stage m, Coconut replaces the first m visible reasoning chunks by m continuous thoughts $z_{1:m}$. These thoughts are generated recursively:

$$S_0 = [c; \langle \text{bot} \rangle], \tag{17}$$

$$z_i = F_\theta([S_0; z_1; \ldots; z_{i-1}]), \qquad i = 1, \ldots, m. \tag{18}$$

The resulting training sequence is

$$\tilde{S}^{(m)} = [c; \langle \text{bot} \rangle; z_1; \ldots; z_m; \langle \text{eot} \rangle; \rho_{m+1}; \ldots; \rho_M; a]. \tag{19}$$

Continuous positions are not vocabulary targets. The supervised likelihood at stage m is therefore

$$p_\theta^{(m)}(\rho_{m+1:M}, a \mid c) = \prod_{i=m+1}^{M} p_\theta(\rho_i \mid c, z_{1:m}, \rho_{m+1:i-1}) \cdot p_\theta(a \mid c, z_{1:m}, \rho_{m+1:M}), \tag{20}$$

with loss

$$\mathcal{L}_{\text{Coconut}}^{(m)} = -\sum_{\ell \in \mathcal{I}^{(m)}} \log p_\theta(\tilde{S}_\ell^{(m)} \mid \tilde{S}_{<\ell}^{(m)}), \tag{21}$$

where $\mathcal{I}^{(m)}$ indexes only discrete supervised tokens after the latent prefix. Increasing m gradually moves supervision from explicit rationales to latent thoughts: $m = 0$ recovers standard CoT training, while $m = M$ leaves only the final answer or action as visible supervision after the latent span.

Serial triangular computation. Equation (18) is inherently serial because z_i must be computed and inserted before z_{i+1} can attend to it. Abstracting away the visible suffix, define a decoder-induced map G_i for the i-th latent update under fixed context c:

$$z_i = G_i(z_1, \ldots, z_{i-1}; c), \qquad i = 1, \ldots, m. \tag{22}$$

Causal attention makes this system strictly triangular: G_i cannot depend on z_i or any future latent z_j with $j > i$. Original Coconut solves Eq. (22) by forward substitution, i.e., it computes z_1, writes it into the sequence, then computes z_2, and so on. This exact serial execution is the reference solution that APLR approximates with parallel Jacobi-style refinement.

E APLR Recovery Proof

We formalize the statement used in Section 3.2.2. The goal is not to prove that a small number of APLR rounds exactly reproduces every serial latent state. Instead, the result is sharper: after K refinement rounds, the first K latent states are exactly identical to the serial rollout, while the remaining latent states have a structured tail error that propagates only through the strictly causal dependency graph.

Setup. Consider one sequence with N latent slots. Each latent state lies in $\mathbb{R}^d$. Let c denote all non-latent context: instruction tokens, history tokens, image embeddings, attention masks, and position encodings. For each latent index i, define G_i as the decoder-induced update that maps the fixed context and all earlier latent states to the hidden state used to replace latent slot i. Causal attention implies

$$G_i = G_i(z_1, \ldots, z_{i-1}; c), \tag{23}$$

so G_i cannot depend on z_i or any future latent z_j with $j > i$.

The serial latent rollout is the forward-substitution solution

$$s_i = G_i(s_1, \ldots, s_{i-1}; c), \qquad i = 1, \ldots, N, \tag{24}$$

with the convention that G_1 depends only on c. APLR initializes every latent slot with the same learned latent embedding, written abstractly as $z_i^{(0)} = e_{\text{lat}}$, and performs the parallel Jacobi-style update

$$z_i^{(r+1)} = G_i(z_1^{(r)}, \ldots, z_{i-1}^{(r)}; c), \qquad i = 1, \ldots, N. \tag{25}$$

The word "parallel" means that all right-hand sides in Eq. 25 are evaluated using round-r latent states before any slot is overwritten for round $r + 1$.

Lemma 1 (Causal dependency cone)**.** *After r APLR refinement rounds, latent slot $z_i^{(r)}$ can depend on the initial latent values only through slots with indices at most $i - r$. Equivalently, information from the exact serial prefix can advance by at most one latent position per refinement round.*

Proof. The claim is immediate for $r = 0$ because $z_i^{(0)}$ is itself an initial value. Assume it holds for round r. At round $r + 1$, $z_i^{(r+1)}$ is a function of $z_1^{(r)}, \ldots, z_{i-1}^{(r)}$ and the fixed context c. By the induction hypothesis, each predecessor $z_j^{(r)}$ can depend on initial slots only up to index $j - r$. Since $j \le i - 1$, the largest such index is $(i - 1) - r = i - (r + 1)$. Thus $z_i^{(r+1)}$ can depend on initial values only through slots with indices at most $i - (r + 1)$. This proves the claim by induction. □

Proposition 1 (Exact recovery of early latent states)**.** *For any number of refinement rounds $r \ge 0$, APLR satisfies*

$$z_i^{(r)} = s_i \qquad \textit{for all } i \le r. \tag{26}$$

Consequently, after K refinement rounds, APLR exactly recovers the first K latent states of the serial rollout, and after N rounds it recovers the full serial latent sequence.

Proof. We prove the claim by induction on the refinement round r.

Base case. For $r = 0$, the set of indices satisfying $i \leq 0$ is empty, so the claim holds vacuously. No latent has been refined yet, and no equality with the serial rollout is required.

Inductive step. Assume that after round r, $z_j^{(r)} = s_j$ for every $j \leq r$. Consider round $r+1$ and any latent index $i \leq r+1$. If $i = 1$, then the causal update has no latent predecessor. Both APLR and serial forward substitution therefore use exactly the same context-only map:

$$z_1^{(r+1)} = G_1(c) = s_1. \tag{27}$$

If $i > 1$, then every predecessor index $j < i$ satisfies $j \leq r$ because $i \leq r+1$. By the induction hypothesis, all predecessor values used by the APLR update are already equal to their serial values:

$$(z_1^{(r)}, \ldots, z_{i-1}^{(r)}) = (s_1, \ldots, s_{i-1}). \tag{28}$$

Substituting these equal predecessors into the parallel update gives

$$z_i^{(r+1)} = G_i(z_1^{(r)}, \ldots, z_{i-1}^{(r)}; c) = G_i(s_1, \ldots, s_{i-1}; c) = s_i, \tag{29}$$

where the final equality follows from the serial recurrence in Eq. 24. Thus the claim holds for round $r+1$, completing the induction. □

Corollary 1 (Which latent slots remain approximate)**.** *After K refinement rounds, the only latent slots that can differ from the serial rollout are the tail slots*

$$z_{K+1}^{(K)}, z_{K+2}^{(K)}, \ldots, z_N^{(K)}. \tag{30}$$

The first K slots have zero serial-approximation error.

Local tail-error expression. The same triangular structure explains the approximation error for the tail slots when $K < N$. Let $\delta_i^{(r)} = z_i^{(r)} - s_i$. Assume each G_i is twice continuously differentiable in a neighborhood of the serial solution. Define the block Jacobian

$$A_{ij} = \begin{cases} \frac{\partial G_i}{\partial z_j}\big|_{s,c}, & j < i, \\ 0, & j \geq i. \end{cases} \tag{31}$$

The matrix A is strictly block-lower-triangular. A Taylor expansion around $(s_1, \ldots, s_{i-1})$ gives

$$\delta_i^{(r+1)} = \sum_{j<i} A_{ij}\delta_j^{(r)} + R_i^{(r)}, \qquad \|R_i^{(r)}\| \leq C_i \|\delta_{<i}^{(r)}\|^2 \tag{32}$$

for constants C_i determined by local Hessian bounds. Stacking all latent errors yields

$$\delta^{(r+1)} = A\delta^{(r)} + R^{(r)}, \qquad \|R^{(r)}\| = O(\|\delta^{(r)}\|^2). \tag{33}$$

Ignoring higher-order terms, the K-round tail error is

$$\delta^{(K)} \approx A^K \delta^{(0)}. \tag{34}$$

Because A is strictly lower triangular, $(A^K)_{ij}$ can be nonzero only when $j \leq i - K$. Thus Eq. 34 has two consequences. First, rows $i \leq K$ are exactly zero, matching the exact-recovery proposition. Second, for a tail latent $i > K$, the remaining error comes only from causal chains of length at least K that connect earlier imperfect initial values to slot i. These are precisely the deep latent states that have not received enough refinement rounds to match the serial rollout.

F How the World-Model Loss Regularizes Tail Errors

The previous section identifies the approximation error left by using $K < N$ APLR rounds: it lives only in the tail latent slots $z_{K+1:N}^{(K)}$. The Q-Former world-model loss does not make these tail slots algebraically identical to the serial rollout. Instead, it adds direct supervision to the components of their error that affect prediction of the next mobile screen in the VLM visual feature space. This is the useful notion of compensation for a mobile agent: the auxiliary loss penalizes tail errors that discard environment-transition information.

Tail-error notation. Let

$$\tau^{(K)} = \left[\delta_{K+1}^{(K)}, \ldots, \delta_N^{(K)}\right] \tag{35}$$

denote the stacked tail error after K APLR rounds. Let P_ϕ denote the Q-Former aligner, including the row/column query embeddings, cross-attention into latent CoT hidden states, and final visual projection. For a valid next-frame transition, define the detached target feature matrix

$$V^\star = \mathrm{sg}(f_{\mathrm{vis}}(x_{t+1})) \in \mathbb{R}^{M \times H_v}. \tag{36}$$

The Q-Former prediction as a function of the latent state is

$$\hat{V}(\tau) = P_\phi(s + \tau; R), \tag{37}$$

where R is the set of valid row/column patch coordinates. For the cosine variant used by default, write normalized predicted and target features as

$$g(\tau) = \mathrm{norm}(\hat{V}(\tau)), \qquad u = \mathrm{norm}(V^\star), \tag{38}$$

where normalization is applied patchwise. The masked world-model loss is

$$\ell_{\mathrm{wm}}(\tau) = \frac{1}{|\mathcal{M}|} \sum_{j \in \mathcal{M}} (1 - \langle g_j(\tau), u_j \rangle) = \frac{1}{2|\mathcal{M}|} \sum_{j \in \mathcal{M}} \|g_j(\tau) - u_j\|_2^2, \tag{39}$$

because both g_j and u_j have unit norm. Here $\mathcal{M}$ contains only real next-frame patches from samples with a valid next frame and at least one latent slot.

Local expansion. Let J be the Jacobian of the normalized Q-Former prediction $g(\tau)$ with respect to the tail error at $\tau = 0$:

$$g(\tau) = g(0) + J\tau + O(\|\tau\|^2). \tag{40}$$

Substituting Eq. 40 into Eq. 39 yields

$$\ell_{\mathrm{wm}}(\tau) - \ell_{\mathrm{wm}}(0) = \frac{1}{|\mathcal{M}|} \langle J^\top (g(0) - u), \tau \rangle + \frac{1}{2|\mathcal{M}|} \|J\tau\|_2^2 + O(\|\tau\|^3). \tag{41}$$

When the serial latent solution is locally optimized for the world-model target, or when we analyze directions orthogonal to the residual gradient $J^\top(g(0) - u)$, the linear term vanishes. The leading term is then a quadratic penalty on the components of τ that the Q-Former prediction can observe.

Proposition 2 (Q-Former world-model supervision controls future-predictive tail error)**.** *Let U be a subspace of tail-error directions on which the normalized Q-Former Jacobian is locally observable: for all $\xi \in U$,*

$$\|J\xi\|_2 \geq \sigma_U \|\xi\|_2 \tag{42}$$

for some $\sigma_U > 0$. Assume the residual-gradient term in Eq. 41 is zero or treated as a first-order optimization gradient. Then, for sufficiently small tail error,

$$\ell_{\mathrm{wm}}(\tau^{(K)}) - \ell_{\mathrm{wm}}(0) \geq \frac{\sigma_U^2}{2|\mathcal{M}|} \|P_U \tau^{(K)}\|^2 - O(\|\tau^{(K)}\|^3), \tag{43}$$

where P_U projects onto the future-predictive subspace of tail errors that change normalized next-frame feature predictions.

Proof. Using Eq. 41 and dropping the zero residual-gradient term gives

$$\ell_{\mathrm{wm}}(\tau^{(K)}) - \ell_{\mathrm{wm}}(0) \geq \frac{1}{2|\mathcal{M}|} \|J\tau^{(K)}\|_2^2 - O(\|\tau^{(K)}\|^3). \tag{44}$$

Decompose the tail error into a future-predictive component and a locally unobserved component:

$$\tau^{(K)} = P_U \tau^{(K)} + (I - P_U)\tau^{(K)}. \tag{45}$$

By definition of U, the Q-Former prediction is informative on $P_U \tau^{(K)}$, with minimum singular value at least σ_U. Therefore

$$\|J\tau^{(K)}\|_2 \geq \sigma_U \|P_U \tau^{(K)}\|. \tag{46}$$

Substituting this inequality into the previous display yields Eq. 43. □

MSE variant. If the implementation uses the MSE world-model loss instead of cosine distance, the same argument applies without patchwise normalization. Let $\hat{V}(\tau) = \hat{V}(0) + B\tau + O(\|\tau\|^2)$. The masked MSE loss has local expansion

$$\ell_{\mathrm{mse}}(\tau) - \ell_{\mathrm{mse}}(0) = \frac{1}{|\mathcal{M}|}\langle B^\top(\hat{V}(0) - V^\star), \tau\rangle + \frac{1}{|\mathcal{M}|}\|B\tau\|_2^2 + O(\|\tau\|^3). \tag{47}$$

Thus any tail-error direction that changes predicted next-frame features is quadratically penalized once the first-order residual term is optimized.

Interpretation. The proposition says exactly which unrefined latent-token errors the Q-Former world-model objective can compensate: errors in the tail slots $z^{(K)}_{K+1:N}$ that are visible to next-frame feature prediction, i.e., errors outside the local nullspace of the Q-Former prediction Jacobian. These directions correspond to latent information that changes the predicted semantic layout of the next screenshot, such as whether a tap opens a menu, whether a typed query changes a text field, or whether navigation moves to a new screen. When the combined loss

$$\mathcal{L} = \lambda_{\mathrm{ce}}\mathcal{L}_{\mathrm{ce}} + \lambda_{\mathrm{wm}}\mathcal{L}_{\mathrm{wm}} \tag{48}$$

is optimized, any such future-predictive tail error increases the objective by at least

$$\frac{\lambda_{\mathrm{wm}}\sigma_U^2}{2|\mathcal{M}|}\|P_U\tau^{(K)}\|^2 \tag{49}$$

up to higher-order terms. Thus the auxiliary world-model loss supplies curvature and gradients for tail latent states that may be weakly constrained by action imitation alone. It does not control tail-error directions that do not affect next-frame visual feature predictions; those directions are either irrelevant to the chosen environment-model target or must be constrained by the action loss and other regularizers.

G Appendix Pipeline Pseudocode

Algorithm 1 summarizes the full training pipeline.

Algorithm 1 MIRAGE two-stage training pipeline

Input: explicit records (o_t, τ_t, d_t, a_t) and sequential records with optional next frame x_{t+1}
Output: a latent-reasoning GUI policy; discard the Q-Former head at inference time

Stage 1: explicit-thought warmup
Serialize $[o_t; \langle\mathrm{THOUGHT}\rangle; \tau_t; \langle\mathrm{ACTION_DESC}\rangle; d_t; \langle\mathrm{ACTION}\rangle; a_t]$
Optimize next-token CE over the structured thought, action description, and action tokens

Stage 2: latent CoT and world-model training
Replace the `<THOUGHT>` block with N learned latent slots as in Eq. (4)
for $r = 0$ **to** $K - 1$ **do**
    Run one causal VLM pass and update all latent slots synchronously using Eq. (6)
end for
Decode action tokens and compute $\mathcal{L}_{\mathrm{ce}}$ by Eq. (8)
if a valid next frame x_{t+1} exists **then**
    Extract detached targets $V_{t+1} = \mathrm{sg}(f_{\mathrm{vis}}(x_{t+1}))$
    Predict $\hat{V}_{t+1}$ from latent states with the Q-Former and compute $\mathcal{L}_{\mathrm{wm}}$
else
    Mask out the world-model term for this sample
end if
Update parameters with $\mathcal{L} = \lambda\mathcal{L}_{\mathrm{ce}} + (1-\lambda)\mathcal{L}_{\mathrm{wm}}$
return latent-reasoning GUI policy

H System Prompt

The complete system prompt used by MIRAGE is shown in Figure 8 alongside the training example illustrations (Appendix I). The prompt defines the role, input format, output format with structured

<THOUGHT> / <ACTION_DESC> / <ACTION> tags, and the full command vocabulary. At inference time the <THOUGHT> block is replaced by the N latent slots described in Section 3.2.2; the explicit text template here is therefore used only during Stage 1 warmup training.

System Prompt (inference-time template)

```
You are an Android mobile GUI agent.
Input: the user task, a summary list of completed past actions, and the current screenshot.
Choose the single best next action from the current screen only. Do not invent unseen UI.
Produce one atomic step. Coordinates are normalized integers in [0,999]. Do not emit
markdown fences or hidden reasoning tags such as <THOUGHT>.

Output:
- Emit only the required blocks using the literal tags ‘<THOUGHT>‘, ‘<ACTION_DESC>‘,
  and ‘<ACTION>‘.
- ‘<THOUGHT>‘: one English line with exactly
  ‘1. observation: ... | 2. rationale: ... | 3. predict: ...‘.
  - ‘observation‘: describe only the currently visible UI on the current screenshot.
  - ‘rationale‘: combine task goal, prior progress, and the reason this single next
    action is best.
  - ‘predict‘: describe the immediate visible result right after the action.
  Keep it grounded in the current screenshot and history. Do not mention coordinates.
  Do not mention prompt structure, annotation metadata, or screenshot-availability
  phrases such as "next screenshot", "no next screenshot", "one image", or "two images".
- ‘<ACTION_DESC>‘: English, one imperative phrase.
- ‘<ACTION>‘: exactly one executable operation command string.

If the task is completed successfully, use ‘finish(success)‘. If the task should
terminate unsuccessfully, use ‘finish(fail)‘.

Command format:
- Tap a visible UI element:
    click(<|point_start|>(x,y)<|point_end|>)
- Swipe to move the page or navigate across the screen:
    swipe(<|point_start|>(x1,y1)<|point_end|>, <|point_start|>(x2,y2)<|point_end|>)
- Drag to move a slider, handle, or draggable object:
    drag(<|point_start|>(x1,y1)<|point_end|>, <|point_start|>(x2,y2)<|point_end|>)
- Long press to open a context action or select an item:
    long_press(<|point_start|>(x,y)<|point_end|>)
- Type into the currently focused field:
    type("text")
- Type into a specific text box:
    type(<|box_start|>(x1,y1),(x2,y2)<|box_end|>, "text")
- Press a supported device key:
    press(HOME)
- Open an app directly when that is the intended system action:
    open_app("Settings")
- Wait briefly for the UI to load or update:
    wait()  or  wait(t)
- Return a textual answer for a non-GUI question:
    answer("text")
- Request human takeover when the task cannot be safely continued:
    call_user()
- End the task:
    finish(success)  or  finish(fail)

Use exactly one command. Do not wrap it in JSON or natural language. Use English double
quotes for string arguments and uppercase English for key names in press(...).
```

Figure 8: Full system prompt used by MIRAGE. The <THOUGHT> block is a visible text template during Stage 1 warmup; it is replaced by learned latent slots during Stage 2 and at inference time. Coordinates are normalized integers in $[0, 999]$.

I Training Example Illustrations

This section shows representative training samples from Stage 1 (explicit-thought) and Stage 2 (latent-thought) to illustrate the data format used by MIRAGE.

Stage 1 format (explicit <THOUGHT>). Each Stage 1 sample consists of a system prompt, a user message containing the task description, history summary, and current screenshot, and an assistant response with three structured blocks: <THOUGHT> (observation | rationale | predict), <ACTION_DESC>, and <ACTION>. Three representative examples are shown below.

Stage 1 Example 1 — Open app drawer to launch SeatGeek

```
Task: Open SeatGeek and search by category ‘‘concerts’’. Select the first one in the results and
track that event. Find the cheapest 2 tickets and record their seat position and price in Google
Keep notes.

History: []

Screenshot: Home screen with purple/pink geometric wallpaper; Google search bar at top; visible apps:
Gallery, Chrome, Play Store; dock: Phone, Messages, App Drawer, Camera. SeatGeek not visible.

<THOUGHT>
1. observation: Home screen is shown with a Google search bar and standard dock icons; SeatGeek is
not present on this page. | 2. rationale: The task requires opening SeatGeek, which is absent from
the home screen, so the app drawer must be opened by swiping upward before proceeding. | 3. predict:
The app drawer opens, revealing a full grid of installed apps including SeatGeek.

<ACTION_DESC>
Swipe up from the centre-bottom of the screen to open the application drawer.

<ACTION>
swipe((509,674), (504,328))
```

Stage 1 Example 2 — Open app drawer to change SeatGeek location

```
Task: Open SeatGeek and change the current location to Los Angeles. Then find the first concert
show and track its performer.

History: []

Screenshot: Home screen; time 8:29; dock contains Phone, Messages, App Drawer, Camera. SeatGeek not
visible.

<THOUGHT>
1. observation: Android home screen with geometric wallpaper; SeatGeek icon is absent. | 2.
rationale: Opening the app drawer is the prerequisite step before SeatGeek can be launched and
the location changed to Los Angeles. | 3. predict: App drawer opens, displaying all installed
applications in a grid; SeatGeek becomes accessible.

<ACTION_DESC>
Swipe up to open the app drawer and locate SeatGeek.

<ACTION>
swipe((522,681), (551,225))
```

Stage 1 Example 3 — Browse SeatGeek trending events

```
Task: Open SeatGeek and browse the top trending events. Record the overall cheapest price and the
best seat cheapest price in Google Keep notes.

History: []

Screenshot: Home screen; bottom dock contains Phone, Google Keep, Android mascot, Camera. SeatGeek
not visible on home screen.

<THOUGHT>
1. observation: Home screen with standard dock; SeatGeek is not among the visible app icons. | 2.
rationale: SeatGeek must be found in the app drawer before the trending-events browsing task can
begin; swiping up is the standard drawer gesture. | 3. predict: App drawer opens, showing the full
app grid with SeatGeek visible.

<ACTION_DESC>
Swipe upward from the lower-centre of the screen to open the application drawer.

<ACTION>
swipe((527,642), (469,315))
```

Stage 2 format (latent `<latent>` substitution). In Stage 2, the entire `<THOUGHT>` block is replaced by N continuous latent tokens. The assistant response no longer contains readable rationale text; only the `<ACTION_DESC>` and `<ACTION>` blocks are decoded as visible tokens. The example below shows how Example 1 above looks in Stage 2 training format.

Stage 2 Format — Latent-slot substitution (Example 1)

```
Task / History / Screenshot: (identical to Stage 1 Example 1 above)

Assistant response (Stage 2):

[BOT] <latent> <latent> ... <latent> [EOT]
      (N latent tokens refined by K APLR passes; not decoded as text)

<ACTION_DESC>
Swipe up from the centre-bottom of the screen to open the application drawer.
```

```
<ACTION>
swipe((509,674), (504,328))
```

During Stage 2 training, the Q-Former world-model head additionally reads the final latent hidden states and aligns them with stop-gradient visual features of the next screenshot. No next-frame pixels are exposed to the action decoder.

J Additional Experimental Analysis

J.1 World-Model Loss Dynamics During Stage 2 Training

Figure 9 plots the world-model auxiliary loss ($\mathcal{L}_{\text{wm}}$) recorded throughout Stage 2 latent-CoT training of MIRAGE-4B, with the cross-entropy coefficient fixed at $\lambda_{\text{ce}} = 0.8$ and all other hyperparameters identical to those described in Appendix B.

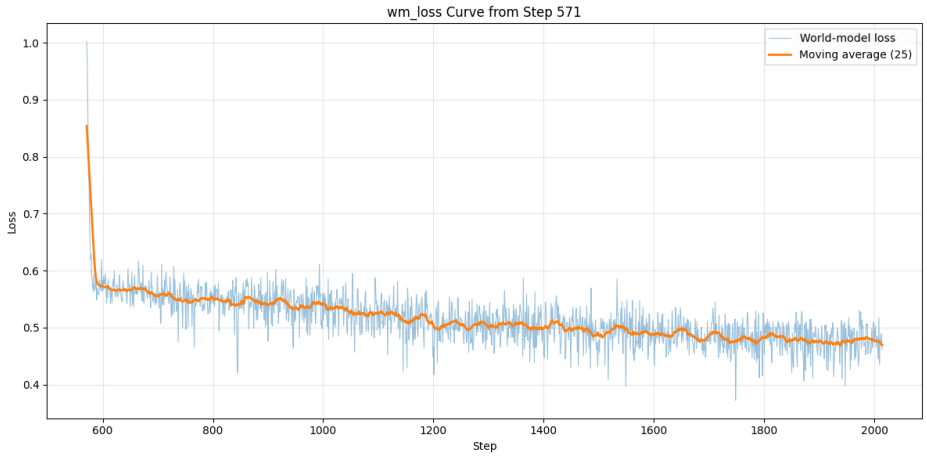

Figure 9: World-model loss ($\mathcal{L}_{\text{wm}}$) during Stage 2 training of MIRAGE-4B ($\lambda_{\text{ce}} = 0.8$). Blue: world-model loss; orange: 25-step moving average. Logging begins at step 571, which corresponds to the first step of Stage 2 after the learning-rate reset.

Implications for regularisation. The smooth, monotone decline of $\mathcal{L}_{\text{wm}}$ confirms that $\lambda_{\text{ce}} = 0.8$ provides effective regularisation without destabilising the primary action-policy objective. The absence of loss spikes or divergent episodes after step 620 demonstrates that the Q-Former world-model head and the LoRA-adapted language backbone can be co-trained stably at this coefficient. Cross-referencing with the ablation results in Table 2 of the main paper, which show that removing the world-model loss degrades tail-error recovery, the consistently decreasing $\mathcal{L}_{\text{wm}}$ here corroborates that the auxiliary signal is being meaningfully absorbed rather than ignored by the gradient.

J.2 Training Efficiency: APLR vs. Serial Latent CoT

Table 5 compares the per-step wall-clock time and peak GPU memory of APLR against a serial latent-CoT baseline under identical hardware and data configurations. Both runs use the same MIRAGE-4B checkpoint, the same mixed Stage-2 batch, and 8 $\times$ RTX 4090 GPUs (24 GB each).

Table 5: Training efficiency comparison on 8$\times$RTX 4090. All measurements are averaged over a fixed window of Stage-2 steps. "Serial latent CoT" unrolls continuous thoughts autoregressively step by step; APLR (ours) approximates the same objective with a parallel Q-Former forward pass.

Method	**Avg. GPU memory (GB)**	**Time / step (s)**
Serial latent CoT	59	15.3
APLR (ours)	**52**	**7.6**
Reduction	-11.9%	-50.3%

APLR achieves a $2\times$ speedup in per-step training time and reduces peak memory by approximately 12% compared with serial latent CoT. The speedup stems from replacing the sequential hidden-state recurrence—which must wait for each latent thought to be produced before computing the next—with a single parallel Q-Former forward pass that generates all latent slots simultaneously. The memory saving follows from eliminating the intermediate KV-cache entries that accumulate during autoregressive latent unrolling. Crucially, the APLR approximation does not sacrifice model quality: as shown in Table 2 of the main paper, the end-task performance of MIRAGE trained with APLR matches or exceeds the serial baseline on both AndroidControl splits. These results confirm that APLR is not merely a conceptual simplification of Coconut-style latent reasoning—it is a practically superior training procedure that halves the compute cost while preserving the expressiveness of the learned latent representation.

J.3 Ablation Study on AndroidControl

Tables 6 and 7 report a full component-level ablation of MIRAGE-4B on the AndroidControl benchmark, covering both task performance and output token counts.

Table 6: Ablation study: task performance on AndroidControl (IDD split). EM = exact match; Action Acc. = action-type accuracy.

Variant	Low EM	Low Act. Acc.	High EM	High Act. Acc.
Base model	68.5	75.2	54.6	69.4
Action-only SFT	66.0	77.0	52.5	68.5
Explicit CoT SFT	78.0	90.5	65.2	76.0
Latent CoT, serial	76.8	89.2	61.8	73.8
APLR only	74.0	85.5	61.0	72.5
MIRAGE-4B (full)	**77.6**	**91.1**	**64.4**	**75.7**

Table 7: Ablation study: average output tokens per step on AndroidControl (IDD split).

Variant	Low-level Tokens	High-level Tokens
Base model	115.7	116.0
Action-only SFT	13.0	13.5
Explicit CoT SFT	125.0	128.0
Latent CoT, serial	19.0	20.5
APLR only	20.0	21.5
MIRAGE-4B (full)	**18.9**	**20.6**

Analysis. The ablation reveals several key insights. First, simply adding SFT on action-only data (row 2) slightly *hurts* exact match relative to the base model, suggesting that removing the reasoning chain impoverishes the model's grounding signal. Explicit CoT SFT (row 3) delivers the strongest single-component gain, confirming that structured thought supervision is essential—but at the cost of 125+ tokens per step, which is prohibitively verbose for deployment. Latent CoT with serial unrolling (row 4) nearly matches Explicit CoT performance while compressing output to only $\sim$19 tokens, validating the core idea of implicit reasoning. APLR alone (row 5) falls slightly below the serial baseline on low-level EM (74.0 vs. 76.8), indicating that the parallel approximation introduces a small fidelity gap when used without the full training recipe. MIRAGE-4B (row 6) closes this gap by combining APLR with the world-model loss and the two-stage curriculum, reaching 77.6 low-level EM and 91.1 action accuracy—matching Explicit CoT quality at a fraction of the token budget (18.9 vs. 125.0 tokens). Taken together, these results demonstrate that MIRAGE's design choices are mutually reinforcing: APLR provides compute efficiency, the world-model auxiliary loss supplies regularisation, and the staged curriculum ensures stable convergence, collectively yielding a model that is both accurate and inference-efficient.

J.4 Latent-Slot Budget

We evaluate the latent-slot budget of MIRAGE-4B on AndroidWorld with $K = 3$ refinement passes and $\lambda_{\text{ce}} = 0.8$. Increasing the budget from three to six to nine slots raises SR from 32.8% to 46.6% to 52.6%, supporting the nine-slot configuration used for MIRAGE-4B.

J.5 World-Model Loss Weight

We vary the loss balance for MIRAGE-4B on AndroidWorld while keeping the architecture, nine-slot budget, refinement passes, training schedule, and evaluation protocol fixed. We write the objective as $\mathcal{L} = \lambda_{\text{ce}}\mathcal{L}_{\text{ce}} + \lambda_{\text{wm}}\mathcal{L}_{\text{wm}}$, where $\lambda_{\text{wm}} = 1 - \lambda_{\text{ce}}$. Table 8 shows that CE weights of 0.80 and 0.85 both yield 52.6% SR. Reducing the WM weight to 0.10 gives 48.3% SR, while equal weights give 41.3% SR. The default 0.80/0.20 balance lies in the best-performing range of the tested settings.

Table 8: Loss-weight sweep for MIRAGE-4B on AndroidWorld.

λ_{ce}	λ_{wm}	SR (%)
0.90	0.10	48.3
0.85	0.15	**52.6**
0.80 (default)	0.20	**52.6**
0.50	0.50	41.3

J.6 Latent-State Interventions and Similarity

We test whether final latent states affect action generation using MIRAGE-4B on a fixed AndroidControl split containing 500 trajectories and 5,692 steps. Interventions are applied after latent filling and immediately before action decoding. The screenshot, goal, history, decoder, and decoding settings are held fixed. Table 9 reports the fraction of steps whose decoded action type changes relative to the unmodified prediction. Changing only the latent states changes the action type in 13.69–28.61% of steps, establishing their influence on action selection.

We further examine slot geometry and action association using cosine similarity. Within the same state, similarities for slot pairs 0/2, 0/4, and 2/4 are 0.7091, 0.4760, and 0.7846. Across episodes, corresponding slots 0, 2, and 4 have similarities of 0.7797, 0.6474, and 0.8230, respectively. Table 10 reports similarities between action-type groups. These measurements complement the projections in Appendix C: the latent states contain positional structure and action-associated variation. Action-group similarities also reflect differences in screen states and goals.

Table 9: Action-type changes after final-state interventions on MIRAGE-4B.

Intervention	Change (%)
Distribution-matched Gaussian perturbation	13.69
Cross-episode, different-goal replacement	28.61
Same-trajectory, different-state replacement	20.68

Table 10: Cosine similarity of MIRAGE-4B latent states across generated action-type groups. Slot indices are zero-based.

Action-type pair	Slot 0	Slot 2	Slot 4
click vs. wait	0.9797	0.9460	0.9194
click vs. swipe	0.7798	0.8487	0.7219
swipe vs. type	0.6774	0.5207	0.6541

J.7 Recovery from External Interruptions

We evaluate recovery on 20 AndroidWorld tasks that both MIRAGE-4B and Qwen3-VL-4B-Instruct previously completed successfully. After an agent action, we inject one interruption by returning the device to the previous screen or the home screen. Both models receive the same intervention protocol and ten additional recovery actions. MIRAGE-4B completes 19/20 tasks (95%), compared with 18/20 (90%) for the backbone. This protocol tests recovery from navigation interruptions on the selected tasks.

J.8 Performance by Task Difficulty

We use AndroidWorld's official difficulty metadata to evaluate the medium and hard task subsets. Table 11 shows that MIRAGE increases medium-task completions from 13 to 16 for 4B and from 15

to 17 for 8B. On hard tasks, the 4B result remains at two completions, while the 8B result increases from two to three. In this evaluation, MIRAGE retains serialized <ACTION_DESC> history and compresses the current-step <THOUGHT> into latent slots.

Table 11: Successful tasks on the medium and hard AndroidWorld subsets.

Model	Medium (36 tasks)	Hard (19 tasks)
Qwen3-VL-4B-Instruct	13/36	2/19
Qwen3-VL-8B-Instruct	15/36	2/19
MIRAGE-4B	16/36	2/19
MIRAGE-8B	17/36	3/19

J.9 Memory-Intensive Evaluation

We evaluate MIRAGE-4B on a 40-task subset of MemGUI-Bench [33] using the benchmark pipeline with Kimi-K2.6 as the LLM judge. Training uses 696 complete MemGUI-3K trajectories (10,000 GUI steps) from MemGUI-Agent [34]. We convert <thinking> traces into the three-part structured thought format, unify the action space, and retain the explicit memory interface for closed-loop updates and reuse. LoRA training consists of one epoch of explicit CoT followed by two epochs with nine latent slots and Q-Former next-frame alignment.

Table 12 reports pass@1 on 35 memory tasks and five non-memory tasks. MIRAGE-4B completes 8/40 tasks, compared with 4/40 for Qwen3-VL-4B, with higher success in both groups. This evaluation examines current-step latent reasoning with explicit persistent memory.

Table 12: MemGUI-Bench pass@1 on the 40-task evaluation subset. Entries report successful tasks / total tasks and success rate.

Policy	Overall	Memory	Non-memory
Qwen3-VL-4B	4/40 (10.0%)	1/35 (2.86%)	3/5 (60.0%)
MIRAGE-4B	8/40 (20.0%)	4/35 (11.43%)	4/5 (80.0%)

The original MemGUI thought blocks average 209.5 tokens, with a median of 211.5 and a 95th percentile of 309.1. Replacing the thought block with nine latent slots reduces reasoning-sequence positions by approximately 95.7%; this statistic measures representation length.

K Action Space

MIRAGE operates over a compact, typed action space that maps directly to Android system APIs. Each action is a single atomic command; the agent never emits more than one command per step. Table 13 lists all supported action types.

Coordinate convention. All spatial coordinates are normalized to $[0, 999]$ independently in width and height, mapping the top-left corner of the screenshot to $(0, 0)$ and the bottom-right corner to $(999, 999)$. This convention is device-resolution-agnostic: the same action is valid on any screen size and is converted to absolute pixel coordinates at execution time by the environment driver.

Action tokenization. Each action command is serialized as a plain ASCII string (e.g. `click((512,340))`) and appended to the VLM's output token sequence. The model is trained with next-token cross-entropy over the action tokens $\mathcal{L}_{\text{ce}}$ (Eq. 8). No structured JSON, markup, or special delimiters are used; the action string is the only content inside the <ACTION> block.

Table 13: Complete action space of MIRAGE. Coordinates (x, y) are normalized integers in $[0, 999]$ relative to the screenshot dimensions.

Action type	Signature	Description
`click`	`click((x,y))`	Tap a visible UI element at screen coordinate (x, y).
`swipe`	`swipe((x1,y1),(x2,y2))`	Swipe from (x_1, y_1) to (x_2, y_2) to scroll or navigate.
`drag`	`drag((x1,y1),(x2,y2))`	Drag a slider, handle, or draggable element.
`long_press`	`long_press((x,y))`	Long-press to open a context menu or select an item.
`type`	`type("text")` `type((x1,y1),(x2,y2), "text")`	Type into the focused field, or into a specific text box identified by its bounding box.
`press`	`press(KEY)`	Press a hardware key (`HOME`, `BACK`, `ENTER`, `VOLUME_UP`, etc.).
`open_app`	`open_app("AppName")`	Launch an installed application by name when that is the intended system action.
`wait`	`wait() / wait(t)`	Wait for the UI to load or update; optional timeout t in seconds.
`answer`	`answer("text")`	Return a textual answer for a non-GUI information-retrieval question without touching the screen.
`call_user`	`call_user()`	Request human takeover when the task cannot be safely continued (e.g. requires biometric authentication).
`finish`	`finish(success) / finish(fail)`	Terminate the episode with a success or failure signal.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main contributions: latent reasoning for mobile agents, APLR, Q-Former future-state alignment, and gains on AndroidControl and AndroidWorld. These claims are supported by the method, theory, main results, and ablations in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: The "Limitations, Broader Impact, and Conclusion" section discusses that MIRAGE is trained from supervised trajectories rather than online reinforcement learning, predicts visual features rather than pixel-level UI changes, and requires next-frame screenshots during training.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The APLR recovery result is stated with its causal triangular-system assumptions, and Appendix E gives the lemma, proposition, corollary, and proof. Appendix F also states the local assumptions used to analyze how the world-model loss regularizes tail errors.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: the paper fully discloses all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and conclusions of the paper.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The code is publicly available, but the data used in this study have not been publicly released.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The code is publicly available.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Results are reported as point estimates without error bars, confidence intervals, or statistical significance tests.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

Justification: The paper provides sufficient information on the computer resources.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work evaluates mobile-agent methods on established benchmarks and discusses privacy and action-execution risks in the broader-impact paragraph. The paper does not report human-subject experiments or release high-risk user data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The "Limitations, Broader Impact, and Conclusion" section notes positive uses such as accessibility and routine automation, as well as negative risks related to private screen content, dataset provenance, privacy protection, and action-execution safeguards.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The submission does not release a new high-risk pretrained model, image generator, or scraped dataset. The paper nevertheless notes that privacy protection and action-execution safeguards would be needed before real deployment.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the original papers of assets used and introduces the details in the section 4.1.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The details of the new assets are introduced in the section 4.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.

- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not report crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject studies, so IRB or equivalent approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper explicitly describes using Qwen3-VL vision-language models as the backbone, fine-tuning them for latent reasoning, and comparing against GPT-4o and other VLM-based GUI agents.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.

- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.