DreamUI: SCALING INTERACTIVE MOBILE AGENT ENVIRONMENTS WITH GENERATIVE UI WORLD MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Training mobile agents for diverse, long-horizon tasks requires scalable interactive environments. However, real apps require authentication and permissions and are difficult to reset, while mock apps rely on predefined data and transition logic that limit task coverage and interaction diversity. In this paper, we present **DreamUI**, a scalable reinforcement learning framework that leverages a *Generative UI World Model* as the training environment. Our core idea is to generate diverse environments while preserving task consistency. The UI world model preserves task-essential facts while introducing plausible variations in unspecified content, exposing agents to richer training contexts. It also generates task-relevant states conditioned on instructions, reducing reliance on predefined app databases when expanding training tasks. Given a task instruction, the current observation, and an agent action, the UI world model generates HTML that renders into the next UI observation, forming a closed interaction loop. To address imperfect environment predictions and sparse task-level feedback, we introduce Environment-Aware Trajectory Masking to exclude corrupted rollouts and a Rubrics-Based Reward System that provides graded rewards through fine-grained task criteria. We train the UI world model through supervised fine-tuning followed by step-wise reinforcement learning. Our DreamUI-World-8B achieves average Instruction Accuracy and Visual Similarity scores of 82.9% and 75.6%, respectively. As for mobile-agent training, RL in the simulated environment raises our DreamUI-4B's success rates to 73.3% on AndroidWorld and 36.2% on MobileWorld. These results demonstrate that our UI world models can serve as effective training environments for mobile-agent RL. We will release our code, data, and models.

1 INTRODUCTION

Recent advances in vision-language models have enabled mobile-agents to perceive and operate smartphone interfaces (Bai et al., 2025; Seed, 2026; Hong et al., 2024; Zhou et al., 2025; Xu et al., 2026a; Li et al., 2026; Tang et al., 2026a; Team et al., 2026a). While earlier approaches primarily rely on pre-collected trajectories (Qin et al., 2025; Wang et al., 2025b), recent work increasingly adopts online reinforcement learning through environment interaction (Lu et al., 2025a; Xu et al., 2025; Xue et al., 2026; Tang et al., 2026a), highlighting the need for scalable training environments.

However, existing work primarily focuses on building reliable evaluation environments for mobile agents (Rawles et al., 2024; Kong et al., 2025), leaving scalable training environments largely underexplored. A straightforward idea is to directly use real-world apps as training environments, but this inevitably encounters authentication barriers, complex stateful dependencies, and the difficulty of resetting environment states, making it hard to scale in practice. One alternative is to construct controllable *mock apps* that emulate real application interfaces and interaction flows (Tang et al., 2026b; Wang et al., 2026a). PhoneWorld (Tang et al., 2026b) takes an initial step in this direction by converting real GUI trajectories into controllable mock apps for training. However, each task is grounded in predefined databases, state schemas, and transition logic, restricting the environment to a finite set of modeled states and interaction paths. As a result, extending coverage to tasks that require operations or state transitions beyond the existing implementation can require additional coding and environment configuration, increasing the cost of large-scale expansion.

In this paper, we present **DreamUI**, a scalable RL training framework built upon a **Generative UI World Model** that serves as an interactive environment for mobile agents. Given a task instruction,

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

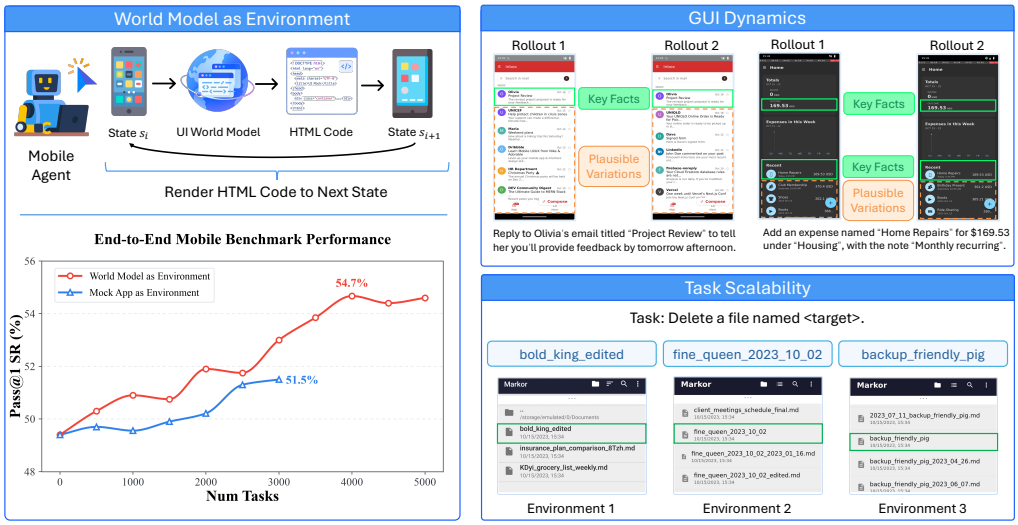

Figure 1: **UI World Model as a Scalable Environment for Mobile-Agent Reinforcement Learning. DreamUI** offers two advantages: (1) **GUI Dynamics**, generating diverse UI states while preserving task-essential facts. (2) **Task Scalability**, building environments from instructions without prepopulated databases. As training tasks scale, world-model environments yield greater gains than mock apps on end-to-end mobile benchmarks, highlighting the value of UI world model for agent RL.

the current screenshot, and an agent action, the world model generates HTML code and renders it into the next UI observation, enabling online interaction without real-device resets. As illustrated in Figure 1, this environment offers two key advantages. (1) **GUI Dynamics**, which allows the model to generate diverse, plausible UI states while preserving task-essential facts and maintaining coherent task progression. By filling in unspecified details, the model exposes agents to diverse contexts across rollouts. (2) **Task Scalability**, which enables the model to construct task-relevant content and subsequent UI states simply from task instructions, reducing reliance on manually populated app databases and task-specific transition logic. We further compare our UI world model with PhoneWorld's mock apps (Tang et al., 2026b) as environments for agent RL, measuring end-to-end performance by the average success rate across AndroidWorld and MobileWorld. At matched training-task counts, world-model-based RL achieves better performance, consistent with the benefits of diverse GUI dynamics. Due to the limited predefined app database, the effective task pool for mock-app RL saturates at around 3,000 tasks. In contrast, our UI world model further scales to 5,000 tasks with substantial performance gains, demonstrating its effectiveness for scalable agent RL.

Turning this generative environment into an effective RL training platform requires handling imperfect world model predictions and providing informative feedback. We therefore introduce two complementary mechanisms. (1) **Environment-Aware Trajectory Masking** detects world model failures during rollouts and excludes corrupted trajectories from policy optimization, preventing unreliable environment feedback from distorting agent learning. (2) **Rubrics-Based Reward System** decomposes task completion into fine-grained success criteria and assigns fractional rewards based on their satisfaction, providing denser learning signals than binary task-level verifiers.

Empirically, we train the UI world model on large-scale agent interaction trajectories using supervised fine-tuning followed by step-wise reinforcement learning. Our DreamUI-World-8B achieves an average Instruction Accuracy of 82.9 and Visual Similarity of 75.6 on MWMBench (Koh et al., 2026), outperforming gWorld-32B (80.6 / 72.7), confirming its use as an interactive training environment. As for the mobile agent, performing RL training within this world-model-simulated environment yields consistent gains on end-to-end online benchmarks: DreamUI-4B improves AndroidWorld success rate from 69.8 to 73.3 and MobileWorld from 29.0 to 36.2, achieving performance comparable to substantially larger specialized agents.

Our main contributions can be summarized as follows:

- We develop a generative UI world model that predicts next states via HTML code generation with state-of-the-art fidelity, capable of serving as a reliable RL training environment.



Figure 2: **Training Data Collection Pipeline for DreamUI-World**. We collect interaction trajectories from diverse mobile tasks and multiple agents in real environments, synthesize screenshots into HTML with reasoning traces, and filter the rendered results by semantic similarity and fine-grained scores. Low-quality samples are further refined through revision.

- We introduce Environment-Aware Trajectory Masking and Rubrics-based Reward System to ensure stable and effective RL optimization in dreamed environments.
- We present a scalable mobile-agent RL framework that supports diverse UI interactions and instruction-driven task expansion, achieving consistent gains on end-to-end benchmarks without real-device interaction during agent RL.

2 Related Works

End-to-End Mobile Agents. End-to-end mobile agents use a single vision-language model to map screenshots to executable actions. Early work relies on supervised fine-tuning (Hong et al., 2024; Qin et al., 2025; Wang et al., 2025b) or offline reinforcement learning (RL) for single-step GUI grounding (Lu et al., 2026; Luo et al., 2025b). Recent studies extend agent training to online RL through multi-turn environment interaction (Wang et al., 2025a; Xue et al., 2026; Tang et al., 2026a; Xu et al., 2025; Zhou et al., 2025), highlighting the need for scalable training environments.

Scalable Environments for Mobile Agents. Programmatically synthesized tool-use environments demonstrate the benefits of environment diversity (Wang et al., 2026b; Fang et al., 2025; Dong et al., 2026), but scaling GUI environments remains challenging. WebArena (Zhou et al., 2024), OSWorld (Xie et al., 2024), and MobileWorld (Kong et al., 2025) primarily support evaluation, while AndroidWorld (Rawles et al., 2024) supports parameterized tasks but remains constrained by app coverage and template design. PhoneWorld (Tang et al., 2026b) converts GUI trajectories into controllable mock apps, demonstrating the benefits of environment scaling for mobile agents. Our framework generates UI transitions with a learned world model, supporting new task instructions without task-specific mock-app construction.

Mobile World Models. Mobile world models predict action-conditioned UI transitions through text (Li et al., 2025; Cao et al., 2026), image generation (Luo et al., 2025a), or renderable code (Koh et al., 2026; Zheng et al., 2026). Existing models primarily support inference-time planning (Cao et al., 2026), candidate action evaluation (Luo et al., 2025a; Zheng et al., 2026; Guan et al., 2026; Koh et al., 2026), or perceptual guidance (Li et al., 2025). Recent work also uses world-model-generated trajectories for offline data augmentation and policy improvement (Xu et al., 2026b). In contrast, we use a generative UI world model as an interactive environment for closed-loop RL, enabling policy optimization through multi-turn interaction with dynamically generated UI states.

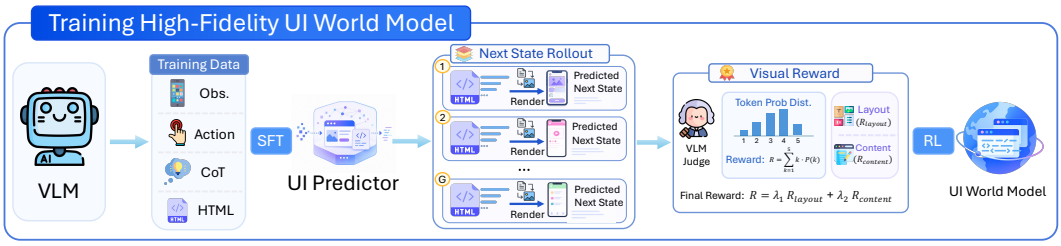

Figure 3: **Overview of High-Fidelity UI World Model Training.** The UI World Model is first trained with supervised fine-tuning on multimodal UI interaction data, then optimized by reinforcement learning that scores next-state rollouts with multi-dimensional visual rewards.

3 DREAMUI

3.1 OVERVIEW.

In this section, we present the overall framework of **DreamUI**, which consists of two stages: (1) training a high-fidelity UI World Model that predicts the next UI state via HTML generation and rendering (Section 3.2), (2) leveraging it as a synthetic environment for mobile agent reinforcement learning (Section 3.3). This design enables scalable agent training through world modeling, eliminating the need for costly human annotations to construct diverse real environments.

3.2 TRAINING HIGH-FIDELITY UI WORLD MODEL.

We formulate the mobile GUI interaction as an instruction-conditioned Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathcal{F}_I)$, where $\mathcal{S}$ is the state space of screenshots, $\mathcal{A}$ is the action space, and $\mathcal{F}_I : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function conditioned on the task instruction I. Given the current state s_t, action a_t, and instruction I, the mobile world model $\mathcal{W}_\phi$ first generates a chain-of-thought reasoning trace c_t that plans the spatial layout of the next screen, followed by the HTML code h_{t+1} representing the predicted next UI state. The resulting HTML code h_{t+1} is then converted into a corresponding pixel-level observation $\hat{s}_{t+1}$. Formally, this process can be expressed as:

$$(c_t, h_{t+1}) = \mathcal{W}_\phi(s_t, a_t, I), \quad \hat{s}_{t+1} = \mathcal{R}(h_{t+1}). \tag{1}$$

where $\mathcal{R}$ denotes the renderer that converts HTML code into the corresponding pixel-level screenshot.

Trajectory Collection in Mobile Environments. To acquire large-scale environment data for training $\mathcal{W}_\phi$, we deploy multiple agents to perform diverse tasks in real mobile environments and collect their interaction trajectories $\tau_I = \{(s_t, a_t, s_{t+1})\}_{t=1}^{T}$. We then employ Seed-2.0-Pro (Seed, 2026) to convert each ground-truth screenshot s_{t+1} into structured HTML code h_{t+1}, and render it as $\hat{s}_{t+1} = \mathcal{R}(h_{t+1})$ for quality verification. Figure 2 illustrates the overall data collection pipeline.

To ensure data fidelity, we apply a two-stage filtering pipeline. **(1) Semantic filtering.** We compute the similarity between $\hat{s}_{t+1}$ and s_{t+1} using SigLIP (Zhai et al., 2023), discarding samples with a score below 0.9 (e.g., screens with severely distorted layouts, missing content, or rendering artifacts caused by malformed HTML). **(2) Fine-grained scoring.** We leverage Qwen3.6-35B-A3B to detect visual anomalies and score each sample in terms of *Layout*, *Content*, and *Style* (each on a 0–10 scale), removing samples that exhibit anomalous patterns (e.g., large blank regions, garbled text, overlapping UI components) or receive a score below 7 in any category. Samples that fail either stage undergo a revision round and are re-evaluated through the same pipeline. Through this Synthesis-Filter-Revise pipeline, we curate a high-fidelity dataset comprising **200K** samples to train $\mathcal{W}_\phi$. Details of data construction are provided in Appendix A.1. Built upon this dataset, the training of the UI World Model proceeds in two stages, as illustrated in Figure 3: supervised fine-tuning for action-conditioned HTML generation, followed by reinforcement learning with multi-dimensional visual rewards.

Stage 1: Supervised Fine-Tuning for HTML-Based UI Generation. We first extract transition-level SFT pairs from the curated trajectories to train the mobile world model for action-conditioned next-state prediction. Each training pair is formatted as:

$$X_t = (s_t, a_t, I), \qquad Y_t = (c_t, h_{t+1}), \tag{2}$$

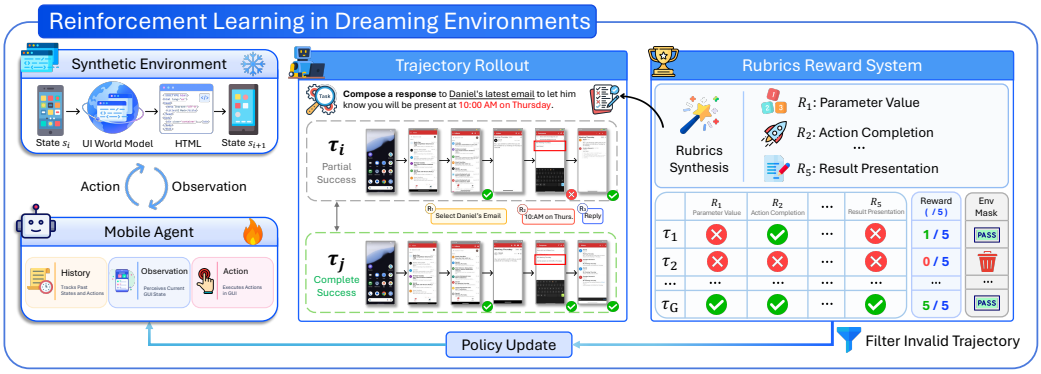

Figure 4: **Overview of the Reinforcement Learning Pipeline in Dreaming Environments.** The learned UI World Model serves as a synthetic environment in which the Mobile Agent performs trajectory rollouts. The resulting trajectories are evaluated using fine-grained rubric-based rewards, with invalid trajectories filtered out and valid experiences used for policy optimization without interacting with real devices.

where s_t denotes the current screenshot, a_t is the agent action, and I is the task instruction. The target Y_t consists of a reasoning trace c_t and the renderable HTML code h_{t+1} of the next state.

The reasoning trace c_t provides intermediate planning for next-state generation. It typically covers the task objective, current progress, the intent of the agent action, and the expected main content of the next frame. This intermediate reasoning helps the model infer the semantic state transition before generating the corresponding HTML representation.

This SFT stage serves as a warm start phase for the mobile world model, enabling next-frame UI content prediction and high-quality HTML generation, which are subsequently rendered into faithful UI screenshots by the Playwright engine.

Stage 2: Step-Wise RL for High-Fidelity UI Rendering. Although SFT enables $\mathcal{W}_\phi$ to generate renderable HTML, its supervision is defined over code tokens rather than the rendered UI states. As a result, the model may capture HTML syntax and common structural patterns without being explicitly optimized for whether the rendered screenshot reflects the true next state. This mismatch is particularly consequential for GUI world modeling, where minor code errors can cause visible layout deviations and incorrect text can alter the semantic state. To bridge this gap, we introduce a step-wise RL stage that directly optimizes HTML generation for pixel-level UI fidelity.

We optimize $\mathcal{W}_\phi$ with GRPO. Given a transition input $X_t = (s_t, a_t, I)$, the policy samples a group of rollouts $\{y^{(i)}\}_{i=1}^{G}$ and updates $\mathcal{W}_\phi$ according to their relative rewards. The rendered screenshot $\hat{s}_{t+1}^{(i)}$ is evaluated against the ground-truth s_{t+1} by a VLM judge along two complementary reward terms:

- $\boldsymbol{R}_{\textbf{layout}}$ evaluates the spatial fidelity of UI components, icons, and other interactive elements.
- $\boldsymbol{R}_{\textbf{content}}$ evaluates textual fidelity and state-specific semantic details in the rendered UI.

To reduce the variance of VLM-based evaluation, we instruct the judge to reason internally and output only a numeric score. We then compute each reward dimension as the expected score over valid tokens $\{1, 2, 3, 4, 5\}$ and combine them into the final reward:

$$R^{(i)} = \lambda_1 R^{(i)}_{\text{layout}} + \lambda_2 R^{(i)}_{\text{content}}, \qquad R^{(i)}_m = \sum_{k=1}^{5} k\, p^{(i)}_{m,k}. \quad m \in \{\text{layout}, \text{content}\}. \tag{3}$$

This group-wise comparison avoids training an additional value model and encourages the policy to favor generations that achieve better rendered fidelity. Through this step-wise RL stage, $\mathcal{W}_\phi$ is optimized not merely to produce syntactically plausible HTML, but to generate code that renders into faithful next-frame GUI observations.

3.3 MOBILE AGENT REINFORCEMENT LEARNING IN DREAMING ENVIRONMENTS.

Building on the high-fidelity UI world model, we optimize the mobile agent through reinforcement learning over dreamed environments. Concretely, we adopt GRPO as our base RL algorithm. For each task instruction, the policy samples multiple trajectories within the dreamed environment, receives a reward signal, and updates via clipped policy gradient with group-relative advantages.

However, directly applying this standard pipeline to dreamed mobile environments introduces two key challenges. First, the world model is repeatedly conditioned on its own predictions during long-horizon rollouts, making error accumulation unavoidable. Once the simulated state drifts from plausible UI dynamics, subsequent feedback becomes unreliable and may incorrectly penalize valid policy actions. Second, conventional environment verifiers rely on task-specific scripts that are costly to design at scale and return only a binary completion label, leaving partial progress unrewarded. To address these challenges, we introduce two designs into our agent RL pipeline, as illustrated in Figure 4: a rollout-level fidelity check that discards unreliable dreamed trajectories, and a rubrics-based reward that grades partial completion.

Environment-Aware Trajectory Masking. In dreaming environments, feedback is generated by a learned world model. During long-horizon interaction, the world model is repeatedly conditioned on its own previous predictions, making error accumulation unavoidable. Once the simulated GUI state deviates from the true environment dynamics, subsequent observations and feedback may become unreliable. In such cases, directly applying RL optimization can incorrectly penalize the policy for actions taken under abnormal environment states, leading the model to learn distorted preferences.

To avoid optimizing on corrupted trajectories, we assign each trajectory τ_i a validity indicator $\mathbb{I}_i \in \{0, 1\}$, where $\mathbb{I}_i = 1$ denotes that no environment-side failure is detected during rollout. Only trajectories with $\mathbb{I}_i = 1$ are used for advantage estimation and policy optimization. The resulting masked GRPO objective is defined as follows:

$$J(\theta) = \mathbb{E}_{q\sim P(Q),\, \{\tau_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\mathcal{T}|q)} \left[\frac{1}{\sum_{i=1}^{G} \mathbb{I}_i} \sum_{i=1}^{G} \mathbb{I}_i \sum_{t=1}^{T_i} \frac{1}{K_{i,t}} \sum_{k=1}^{K_{i,t}} \left\{ \min\left(r_{i,t,k}(\theta)\hat{A}_i,\ \text{clip}\left(r_{i,t,k}(\theta),\, 1-\epsilon_{\text{low}},\, 1+\epsilon_{\text{high}}\right)\hat{A}_i \right) \right\} \right], \tag{4}$$

Here, G counts sampled trajectories, T_i the steps in τ_i, and $K_{i,t}$ the optimized model-generated tokens at step t. The importance ratio $r_{i,t,k}(\theta)$ divides the current policy's token probability by the rollout policy's, conditioned on the task, preceding interactions, and earlier tokens in the step. The parameters ϵ_{low} and ϵ_{high} set the clipping bounds. The group-relative advantage $\hat{A}_i$ uses only trajectories with $\mathbb{I}_i = 1$; groups with no valid trajectories are skipped. Masking thus excludes detected environment failures from policy optimization.

Rubrics-Based Reward System. Standard RL for mobile agents relies on binary task-completion signals from environment-specific scripts, which are both costly to design and uninformative for partial progress. In contrast, we observe that mobile-agent tasks are usually specified with explicit and fine-grained requirements, leaving little ambiguity about successful completion. Such clear task definitions are naturally suitable for rubric-based evaluation, where explicit success criteria are derived directly from the instruction to assess task completion.

Specifically, we organize task rubrics into five categories: **Target Constraint**, **Filter Condition**, **Parameter Value**, **Result Presentation**, and **Action Completion**. These categories respectively describe what target the task operates on, which filters or constraints should be applied, which concrete values must be specified, how the result should be presented or delivered, and whether the key final action has been completed. Detailed definitions of these rubrics are provided in Appendix B.3.

For each task instruction q, we prompt Gemini-3.1-Pro to generate a compact set of 3–5 rubrics, denoted as $\mathcal{C}(q) = \{c_j\}_{j=1}^{M_q}$. During RL training, a separate judge model takes the task instruction, the generated rubrics, and the entire agent trajectory τ_i as input, then computes the trajectory reward as the fraction of satisfied rubrics:

$$R_i = \frac{1}{M_q} \sum_{j=1}^{M_q} \mathbb{I}[\text{Judge}(q, c_j, \tau_i) = 1]\,. \tag{5}$$

This fractional reward provides a denser learning signal by assigning partial credit according to how many requirements are satisfied. By preserving these differences among partially successful trajectories, the reward ranks trajectories by how much of the instruction they fulfill rather than by completion alone, and keeps hard tasks trainable where binary supervision would collapse every rollout to zero, thus providing more informative guidance for policy optimization. Such graded feedback is difficult to obtain from real-environment verifiers, which typically rely on task-specific scripts and return only a binary success label. Environment-side failures in dreamed rollouts are also detected using rubric-style criteria, with details provided in Appendix B.3.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP.

UI World Model. We train the UI world model $\mathcal{W}_\phi$ in two stages: supervised fine-tuning on 200K curated transition samples for action-conditioned HTML generation, followed by step-wise GRPO with VLM-judged layout and content rewards to further improve rendered UI fidelity. Additional training details are provided in Appendix A.3. Following gWorld (Koh et al., 2026), we evaluate UI world models on MWMBench for action-conditioned next-state prediction. We report three metrics: **Instruction Accuracy (IAcc.)**, a VLM-as-a-Judge assessment of whether the predicted state is consistent with the executed action; **Visual Similarity (Sim.)**, the perceptual agreement between the predicted and ground-truth screenshots, averaged over DINO (Caron et al., 2021) and DINOv2 (Oquab et al., 2023) scores; and **Render Failure Rate (RF)**, the percentage of predictions that fail to render into a valid screenshot. Evaluation is conducted on AitW (Rawles et al., 2023), GUIOdyssey (Lu et al., 2025b), Android Control (Li et al., 2024), and AMEX (Chai et al., 2025), covering diverse mobile applications, tasks, and interaction patterns.

Mobile Agent. We leverage the trained UI world model $\mathcal{W}_\phi$ as a simulated training environment and synthesize approximately 5K training instructions through app-guided task synthesis. For each target application, we summarize its supported operations, the types of information it handles, and its roles in user workflows. An LLM takes these descriptions as context rather than fixed task templates, composing single-app tasks and cross-app tasks with explicit information dependencies, and combining goals, constraints, and workflows beyond the reference scenarios while remaining consistent with the applications' intended functionality. Each task is paired with a compatible initial observation and explicit completion criteria. We optimize the agent over simulated trajectory rollouts, with Seed-2.0-Pro (Seed, 2026) serving as the reward judge. Details of task synthesis, representative examples, and training configurations are provided in Appendix B. We evaluate our mobile agent on AndroidWorld (Rawles et al., 2024) and MobileWorld (Kong et al., 2025), measuring end-to-end task completion in real-device environments. We compare against both general vision-language models and specialized GUI agents.

4.2 EVALUATION OF GENERATIVE UI WORLD MODEL.

Table 1 reports the next-state prediction quality on MWMBench, evaluated with Instruction Accuracy (IAcc.), Visual Similarity (Sim.), and Render Failure Rate (RF). These metrics jointly assess whether the predicted next state reflects the intended action-induced change, remains semantically close to the ground-truth screenshot, and can be reliably rendered from the generated HTML.

Despite having only 8B parameters, DreamUI-World-8B achieves the strongest overall performance among all image- and code-generation baselines. It obtains the best average IAcc. / Sim. of 82.9 / 75.6, outperforming both gWorld-32B (80.6 / 72.7) and gWorld-8B (76.8 / 71.7), as well as general-purpose code generators such as GLM-4.6V-106B (68.2 / 70.5). DreamUI-World-8B achieves the highest Instruction Accuracy on three of the four subsets and the best Visual Similarity on three of four, leading to the best average IAcc. and Sim. overall.

DreamUI-World-8B also demonstrates strong rendering reliability, with an average RF of only 0.5%—lower than gWorld-32B (0.7%) and far below general-purpose code generators such as GLM-4.6V-106B (2.2%). This robustness is essential for agent reinforcement learning, where each rollout step must yield a valid visual observation; rendering failures would otherwise interrupt trajectories or introduce invalid states into the training loop. These results confirm that equipping the world model

Table 1: **Benchmark Performance on MWMBench.** The average score is computed over AitW, GUIOdyssey, Android Control, and AMEX. RF denotes the render failure rate.

Model	AitW		GUIOdyssey		Android Control		AMEX		Average	
	IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓
Image Generation Models										
Qwen-Image-Edit-20B (Wu et al., 2025)	15.4 / 60.1	–	13.0 / 63.8	–	11.7 / 63.8	–	10.9 / 64.4	–	12.8 / 63.0	–
Emu3.5-34B (Cui et al., 2025)	23.4 / 68.7	–	25.8 / 68.8	–	27.7 / 68.6	–	21.7 / 71.6	–	24.7 / 69.4	–
Code Generation Models										
Llama4-109B-A17B (Adcock et al., 2026)	47.6 / 57.9	4.4	53.1 / 62.3	1.2	50.7 / 61.4	1.0	49.0 / 66.9	0.6	50.1 / 62.1	1.8
Llama4-402B-A17B (Adcock et al., 2026)	47.2 / 58.9	9.4	55.8 / 64.0	7.8	58.6 / 63.1	8.6	58.3 / 68.1	12.6	55.0 / 63.5	9.6
Qwen3-VL-8B (Bai et al., 2025)	21.5 / 49.9	33.8	28.2 / 48.3	51.4	31.1 / 53.4	42.8	33.7 / 59.2	31.6	28.6 / 52.7	39.9
Qwen3-VL-32B (Bai et al., 2025)	46.8 / 59.0	11.6	52.0 / 62.7	16.0	53.2 / 64.1	13.4	56.9 / 70.0	3.8	52.2 / 64.0	11.2
Qwen3-VL-235B-A22B (Bai et al., 2025)	36.1 / 62.9	40.0	54.7 / 69.7	27.2	51.9 / 68.8	34.2	51.2 / 71.7	30.0	48.5 / 68.3	32.9
Code2World-8B (Zheng et al., 2026)	46.2 / 60.4	1.2	65.6 / 72.3	3.6	73.2 / 75.8	3.2	67.9 / 75.2	1.6	63.2 / 70.9	2.4
GLM-4.6V-106B (Hong et al., 2025)	60.9 / 64.7	2.4	68.2 / 72.5	3.8	74.2 / 71.4	1.4	69.5 / 73.2	1.2	68.2 / 70.5	2.2
gWorld-8B (Koh et al., 2026)	68.8 / 66.3	0.8	77.2 / 73.3	1.2	78.4 / 72.8	2.6	82.6 / 74.3	0.8	76.8 / 71.7	1.4
gWorld-32B (Koh et al., 2026)	71.7 / 67.3	0.6	81.5 / 73.7	0.8	82.9 / 74.2	0.8	**86.1** / 75.4	0.4	80.6 / 72.7	0.7
DreamUI-World-8B	**78.1 / 70.2**	1.8	**84.9 / 77.2**	**0.2**	**84.9** / 74.5	**0.0**	83.8 / **80.5**	**0.0**	**82.9 / 75.6**	**0.5**

Table 2: **Comparison with State-of-the-Art Methods on Online Mobile Use Benchmarks.** All results are reported under the Pass@1 metric. Baseline results are mainly sourced from the GUI-Owl-1.5 paper (Xu et al., 2026a) and the MobileWorld (Kong et al., 2025) Leaderboard. Results marked with † are re-evaluated using the official model weights under our evaluation setting.

Model Type	Model	Parameter Size	AndroidWorld	MobileWorld (GUI-Only)
General Models	SeedVL-1.5 (Seed, 2025a)	–	62.1	–
	Seed-1.8 (Seed, 2025b)	–	70.7	52.1
	Seed-2.0-Pro (Seed, 2026)	–	–	63.2
	Claude-4.5-Sonnet (Anthropic, 2025)	–	56.0	47.8
	Gemini-2.5-Pro (Deepmind, 2025)	–	69.7	–
	Kimi K2.5 (Team et al., 2026b)	1T-A32B	–	49.6
	Qwen3-VL-Instruct (Bai et al., 2025)	8B	47.6	9.4
	Qwen3-VL-Instruct (Bai et al., 2025)	32B	57.3	11.9
	Qwen3-VL-Instruct (Bai et al., 2025)	235B-A22B	63.7	12.8
GUI Models	UI-TARS-1.5 (Seed, 2025c)	7B	64.2	26.3
	UI-TARS-2 (Wang et al., 2025a)	–	73.3	–
	GUI-Owl (Ye et al., 2025)	7B	66.4	7.7
	GUI-Owl (Ye et al., 2025)	32B	–	8.5
	GELab-Zero (Yan et al., 2025)	4B	63.9	16.1
	GELab-Zero (Yan et al., 2025)	8B	67.7	–
	MAI-UI (Zhou et al., 2025)	8B	70.7	27.5
	MAI-UI (Zhou et al., 2025)	32B	73.3	36.2
	MAI-UI (Zhou et al., 2025)	235B-A22B	76.7	39.7
	GUI-Owl-1.5-Instruct† (Xu et al., 2026a)	4B	69.8	29.0
Ours	**DreamUI**	**4B**	**73.3**	**36.2**

with chain-of-thought reasoning and optimizing it through SFT followed by step-wise RL produces predictions that are both faithful to action-conditioned state transitions and robust enough to serve as a scalable interactive RL environment.

4.3 Evaluation of Mobile Agent.

As shown in Table 2, **DreamUI** consistently improves online task success through RL training in the generative UI world model. Compared with GUI-Owl-1.5-4B-Instruct, DreamUI-4B improves AndroidWorld from 69.8 to **73.3** (+3.5) and MobileWorld (GUI-Only) from 29.0 to **36.2** (+7.2). The larger gain on MobileWorld suggests that world-model-based RL is particularly effective for long-horizon, cross-app interactions. Notably, DreamUI-4B achieves performance comparable to substantially larger specialized agents, while outperforming generalist models at a similar scale. These results demonstrate that generative UI world models can serve as an effective training environment for improving real-device task performance of mobile agents.

4.4 Ablation Study.

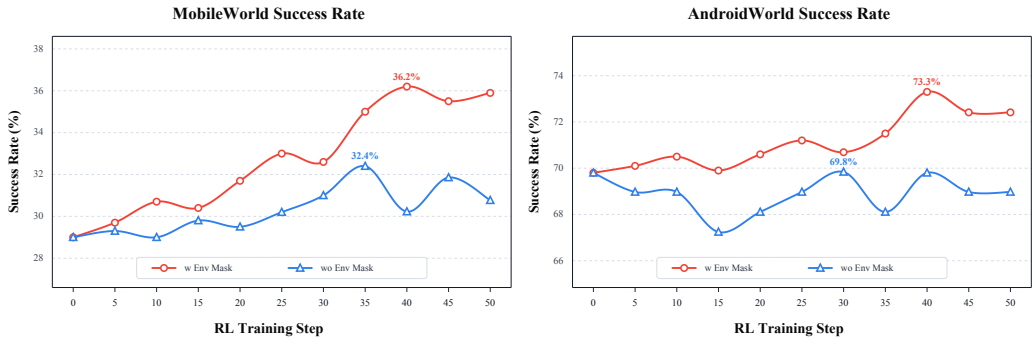

Figure 5: **Effectiveness of Environment-Aware Trajectory Masking.** We compare RL training with and without Environment-Aware Trajectory Masking and report the success rates on MobileWorld (GUI-Only) and AndroidWorld for a more stable evaluation of model performance.

Table 3: **Ablation Study on the Rubrics Reward System.** We ablate Binary Naive Reward, Binary Rubrics-Based Reward, and Graded Rubrics-Based Reward under the same mobile agent RL setting. We report success rates of each model variant on AndroidWorld and MobileWorld.

ID	Setting	AndroidWorld	MobileWorld (GUI-Only)
0	Baseline	69.8	29.0
1	Binary Naive Reward	71.1	32.7
2	Binary Rubrics Reward	71.1	34.2
3	Graded Rubrics Reward	**73.3**	**36.2**

Analysis of Environment-Aware Trajectory Masking. When the UI world model serves as the interactive environment, it repeatedly conditions on its own previous predictions to generate subsequent frames, making error accumulation over long horizons unavoidable. The world model may also occasionally produce anomalous outputs that leave a task unsolvable for the agent. Without proper handling, these environment-side failures are incorrectly attributed to the policy during RL optimization, introducing noisy learning signals that destabilize training. Figure 5 shows that without masking, performance fluctuates and converges poorly, while excluding corrupted trajectories yields a more stable curve, faster convergence, and higher end-to-end success rates. This confirms that filtering unreliable rollouts is critical for stable policy optimization in simulated environments.

Effect of Rubrics-Based Reward Design. To isolate the effect of our reward design, we fix Seed-2.0-Pro as the judge and enable Environment-Aware Trajectory Masking during mobile agent RL. We compare three variants: (1) Binary Naive Reward, which judges task completion without rubrics; (2) Binary Rubrics-Based Reward, which applies rubric-based judgment with binary success labels; and (3) Graded Rubrics-Based Reward, which performs rubric-based judgment with fractional scores. As shown in Table 3, both rubric-based variants improve MobileWorld performance over Binary Naive Reward, while the two binary variants achieve the same AndroidWorld score. Graded Rubrics-Based Reward achieves the highest success rates on both benchmarks, supporting the effectiveness of this reward design for downstream agent training under the tested setting.

5 Conclusion

In this paper, we present **DreamUI**, a scalable reinforcement learning framework that leverages a generative UI world model as the interactive training environment for mobile agents. By formulating next-state prediction as HTML code generation, the world model captures general GUI dynamics and provides a reset-free, open-ended environment that easily scales to new tasks without requiring real devices or manual mock-app construction during agent RL. To address the challenges of imperfect long-horizon world modeling and the absence of task-completion verifiers, we introduce environment-aware trajectory masking to filter corrupted rollouts and a rubrics-based reward system to provide fine-grained supervision. Our UI world model achieves state-of-the-art prediction fidelity, and mobile agents trained within this simulated environment obtain consistent improvements on both online and offline benchmarks, demonstrating that generative world models offer a practical and scalable path toward training capable mobile agents through reinforcement learning.

AI USE STATEMENT

In this work, we used generative AI tools for data synthesis and automated evaluation, as described in the main text and appendix. Additionally, we used generative AI tools for improving the clarity and readability of the manuscript. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work adheres to high ethical standards in machine learning and computer vision, ensuring transparency, reproducibility, and fairness throughout all experiments.

REPRODUCIBILITY STATEMENT

The main paper describes the world-model and agent-training framework. Appendix A.1 details the training-data construction, Appendix A.3 gives the world-model training configuration, and Appendix B describes agent task synthesis, training hyperparameters, and reward computation. Appendix B.4 reports the latency measurement, and the prompt templates included in this paper are collected in Appendix D. We will release our code, datasets and models to facilitate further research.

REFERENCES

Aaron Adcock, Aayushi Srivastava, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pande, Abhinav Pandey, Abhinav Sharma, Abhishek Kadian, Abhishek Kumawat, Adam Kelsey, et al. The llama 4 herd: Architecture, training, evaluation, and deployment notes. *arXiv preprint arXiv:2601.11659*, 2026.

Anthropic. Claude-4-5-sonnet. Technical report, Anthropic, 2025. URL https://www.anthropic.com/news/claude-4-5-sonnet.

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.

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.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

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*, pp. 2138–2156, 2025.

Yufeng Cui, Honghao Chen, Haoge Deng, Xu Huang, Xinghang Li, Jirong Liu, Yang Liu, Zhuoyan Luo, Jinsheng Wang, Wenxuan Wang, et al. Emu3. 5: Native multimodal models are world learners. *arXiv preprint arXiv:2510.26583*, 2025.

Deepmind. Gemini 2.5: Our most intelligent ai model. Technical report, Deepmind, 2025. URL https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, et al. Towards general agentic intelligence via environment scaling. *arXiv preprint arXiv:2509.13311*, 2025.

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

Yiming Guan, Rui Yu, John Zhang, Lu Wang, Chaoyun Zhang, Liqun Li, Bo Qiao, Si Qin, He Huang, Fangkai Yang, Pu Zhao, Lukas Wutschitz, Samuel Kessler, Huseyin A Inan, Robert Sim, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. Computer-using world model. *arXiv preprint arXiv:2602.17365*, 2026.

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

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.5 v and glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*, 2025.

Woosung Koh, Sungjun Han, Segyu Lee, Se-Young Yun, and Jamin Shin. Generative visual code mobile world models. *arXiv preprint arXiv:2602.01576*, 2026.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, et al. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. *arXiv preprint arXiv:2512.19432*, 2025.

Chenxin Li, Zhengyao Fang, Zhengyang Tang, Pengyuan Lyu, Xingran Zhou, Xin Lai, Fei Tang, Liang Wu, Yiduo Guo, Weinong Wang, et al. Phoneharness: Harnessing phone-use agents through mixed gui, cli, and tool actions. *arXiv preprint arXiv:2606.14832*, 2026.

Shufan Li, Konstantinos Kallidromitis, Akash Gokul, Yusuke Kato, Kazuki Kozuka, and Aditya Grover. Mobileworldbench: Towards semantic world modeling for mobile agents. *arXiv preprint arXiv:2512.14014*, 2025.

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.

Feihong Lu, Zixin Zhong, Shubo Liu, Chi-Wing Fu, and Jiaya Jia. ARPO: End-to-end policy optimization for GUI agents with experience replay. In *Proceedings of NeurIPS*, 2025a.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices, 2025b. URL https://arxiv.org/abs/2406.08451.

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, pp. 17608–17616, 2026.

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*, 2025a.

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*, 2025b.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

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.

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

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

ByteDance Seed. Seed1.5-vl technical report. *arXiv preprint arXiv:2505.07062*, 2025a.

Bytedance Seed. Seed1.8 model card: Towards generalized real-world agency. *arXiv preprint*, December 2025b. Technical Report.

ByteDance Seed. Ui-tars-1.5, 2025c.

Bytedance Seed. Seed2. 0 model card: Towards intelligence frontier for real-world complexity. *arXiv preprint arXiv:2607.00248*, 2026.

Zhengyang Tang, Xin Lai, Pengyuan Lyu, Xinyuan Wang, Tianyi Bai, Chenxin Li, Yiduo Guo, Huawen Shen, Yuxuan Liu, Junyi Li, et al. Training open models for agentic phone use. *arXiv preprint arXiv:2606.23049*, 2026a.

Zhengyang Tang, Yuxuan Liu, Xin Lai, Junyi Li, Pengyuan Lyu, Yiduo Guo, Zhengyao Fang, Yang Ding, Yi Zhang, Weinong Wang, et al. Phoneworld: Scaling phone-use agent environments. *arXiv preprint arXiv:2605.29486*, 2026b.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Hy Vision Team, Huawen Shen, Zhengyang Tang, Shangpin Peng, Liang Wu, Anran Zhang, Weinong Wang, Yiduo Guo, Chenxin Li, Zhengyao Fang, et al. Hymobileagent: Data-environment co-scaling for efficient gui agents. *arXiv preprint arXiv:2607.14548*, 2026a.

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

Bowen Wang, Dunjie Lu, Junli Wang, Tianyi Bai, Shixuan Liu, Zhipeng Zhang, Haiquan Wang, Hao Hu, Tianbao Xie, Shuai Bai, et al. Cua-gym: Scaling verifiable training environments and tasks for computer-use agents. *arXiv preprint arXiv:2605.25624*, 2026a.

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

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. Opencua: Open foundations for computer-use agents, 2025b. URL https://arxiv.org/abs/2508.09123.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, et al. Mmbench-gui: Hierarchical multi-platform evaluation framework for gui agents. *arXiv preprint arXiv:2507.19478*, 2025c.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning. *arXiv preprint arXiv:2602.10090*, 2026b.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng-ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. Os-atlas: A foundation action model for generalist gui agents, 2024. URL https://arxiv.org/abs/2410.23218.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Proceedings of NeurIPS*, 2024.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026a.

Weikai Xu, Kun Huang, Yunren Feng, Jiaxing Li, Yuhan Chen, Yuxuan Liu, Zhizheng Jiang, Heng Qu, Pengzhi Gao, Wei Liu, et al. How mobile world model guides gui agents? *arXiv preprint arXiv:2605.10347*, 2026b.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. MobileRL: Online agentic reinforcement learning for mobile GUI agents. *arXiv preprint arXiv:2509.18119*, 2025.

Zhe Xu et al. Aguvis: Unified pure vision agents for autonomous GUI interaction. *arXiv preprint arXiv:2412.04454*, 2024.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026.

Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, et al. Step-gui technical report. *arXiv preprint arXiv:2512.15431*, 2025.

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

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11975–11986, 2023.

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

Yifan Zhang et al. AgentCPM-GUI: Building mobile-use agents with reinforcement fine-tuning. *arXiv preprint arXiv:2025.00000*, 2025.

Yuhao Zheng, Li'an Zhong, Yi Wang, Rui Dai, Kaikui Liu, Xiangxiang Chu, Linyuan Lv, Philip Torr, and Kevin Qinghong Lin. Code2world: A gui world model via renderable code generation. *arXiv preprint arXiv:2602.09856*, 2026.

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.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

CONTENTS

A UI World Model Training Details. **15**
A.1 Training Data Construction. 15
A.2 HTML Synthesis with Chain-of-Thought Reasoning. 16
A.3 Training Configuration. 17

B Mobile Agent Training Details. **18**
B.1 Training Task Synthesis. 18
B.2 Training Configuration. 18
B.3 Rubrics-Based Reward System. 18
B.4 Inference Latency of the UI World Model. 20

C Experiments. **21**
C.1 Evaluation of UI World Model Prediction Fidelity. 21
C.2 Improving UI World Modeling with Chain-of-Thought Reasoning and Step-Wise RL. 22
C.3 Weighted VLM Judge Score for Step-Wise RL of the UI World Model. 23
C.4 Scaling Trend of the UI World Model. 23
C.5 Offline Benchmark Performance of the Mobile Agent. 24
C.6 Training Budget Ablation for Mobile Agent RL. 25

D Prompt Templates. **25**
D.1 Prompt Templates for Data Synthesis. 25
D.2 Prompt Templates for Step-Wise RL Reward Judge. 31
D.3 Prompt Templates for the Rubrics Reward Judge. 35
D.4 Prompt Templates for UI World Model Fidelity Evaluation. 36

E Visualization. **39**
E.1 Comparison with Other UI World Models. 39
E.2 Trajectory Visualization. 40

F Limitations and Future Work. **41**

A UI WORLD MODEL TRAINING DETAILS.

A.1 TRAINING DATA CONSTRUCTION.

To train a high-fidelity UI world model, we construct the **DreamUI-World Training Corpus**, comprising **200,656** UI interaction samples collected from both online environments and existing GUI datasets. As illustrated in Figure 6, the corpus combines interaction trajectories from Android Control (Li et al., 2024), GUIOdyssey (Lu et al., 2025b), AMEX (Chai et al., 2025), AitW (Rawles et al., 2023), and AITZ (Zhang et al., 2024), together with interaction trajectories collected by multiple agents in the AndroidWorld and MobileWorld sandbox environments. In particular, the online environments contribute **124,420** samples (**62.0%**), covering diverse task executions generated by both Seed-2.0-Pro (Seed, 2026) and GUI-Owl-1.5-8B (Xu et al., 2026a). The remaining samples

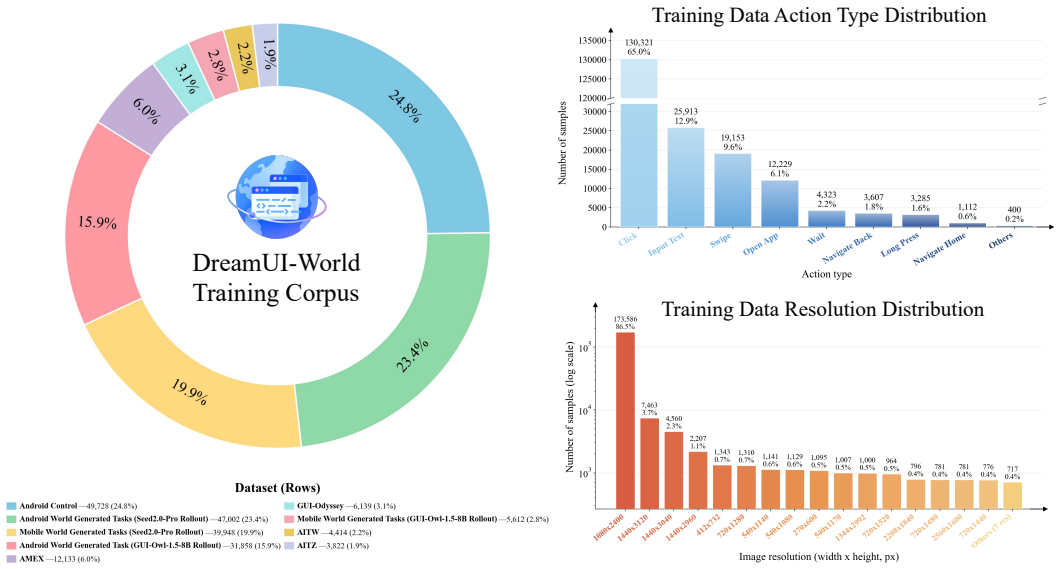

Figure 6: **Statistics of the Training Data Distribution.** Our training data comprises **200,656** samples collected from both online environments and existing GUI datasets, covering diverse UI interaction behaviors across a broad range of screen resolutions.

are collected from established GUI datasets, providing complementary coverage of real-world mobile interfaces and interaction patterns.

Our corpus covers a broad range of mobile actions. Click operations constitute the majority of samples (**65.0%**), followed by text input (**12.9%**), swipe (**9.6%**), and app launching (**6.1%**). It further includes less frequent but practically important actions, such as waiting, navigating back or home, and long pressing. This composition reflects the action distribution encountered in realistic mobile interaction while retaining sufficient coverage of diverse control primitives.

The collected data also spans multiple screen resolutions and device configurations. Although the dominant resolution is 1080×2400 (**86.5%**), the corpus includes a variety of high- and low-resolution screens, ranging from compact displays such as 412×732 to higher-resolution devices such as 1440×3120. Such diversity exposes the model to varying UI layouts, component scales, and spatial arrangements, improving its ability to generalize across heterogeneous mobile interfaces.

A.2 HTML Synthesis with Chain-of-Thought Reasoning.

To convert the collected interaction trajectories into training data for the UI world model, we apply a multi-stage synthesis and filtering pipeline. For each transition pair (s_t, a_t, s_{t+1}), we first employ Seed-2.0-Pro (Seed, 2026) to generate structured HTML code h_{t+1} from the ground-truth next screenshot s_{t+1}, following the UI-to-HTML generation protocol detailed in Prompt 1. The generated HTML is rendered via Playwright into $\hat{s}_{t+1}$ for quality verification. We then apply a two-stage filtering pipeline: (1) semantic filtering using SigLIP (Zhai et al., 2023) to discard samples whose rendered screenshots have a similarity score below 0.9 with the ground truth, and (2) fine-grained VLM-based scoring using Qwen3.6-35B-A3B to evaluate each sample along *Layout*, *Content*, and *Style* dimensions (each on a 0–10 scale), as specified in Prompt 2. Samples scoring below 7 in any dimension or exhibiting environmental anomalies undergo an iterative refinement round, where Seed-2.0-Pro revises the HTML based on the diagnosed visual differences between the rendered output and the ground-truth screenshot (Prompt 3). Revised samples are then re-evaluated through the same filtering pipeline.

For all samples that pass the filtering stages, we further generate look-ahead reasoning traces to serve as chain-of-thought supervision. Specifically, given the current screenshot s_t, the executed action a_t, and the ground-truth next state s_{t+1}, we prompt Seed-2.0-Pro to produce a coherent reasoning

Table 4: **Hyperparameters for Step-Wise RL of the UI World Model.**

Hyperparameter	Value
Algorithm	GRPO
Rollouts Per Prompt (G)	8
Rollout Temperature	1.0
Learning Rate	5×10^{-6}
Entropy Coefficient	0.001
Clip Ratio (ϵ_{low} / ϵ_{high})	0.2 / 0.28
KL Penalty Coefficient	0.0
Train Batch Size	128
Mini Batch Size	128
Micro Batch Size Per GPU	8
Max Prompt Length	8,192 tokens
Max Response Length	10,240 tokens
Group Filter Std Threshold	0.02
Judge Model	Qwen3.6-35B-A3B
Reward Weights (Layout / Content)	0.5 / 0.5

paragraph that articulates the agent's intention behind the action, describes the expected next interface state at a semantic level, and reasons about the current task progress and subsequent needs (see Prompt 4). This reasoning trace c_t is paired with the corresponding HTML code h_{t+1} to form the final training target (c_t, h_{t+1}), enabling the world model to learn structured planning before generating the next UI state.

A.3 Training Configuration.

Stage 1: Supervised Fine-Tuning for HTML-Based UI Generation. In the SFT stage, we initialize the model with **Qwen3-VL-8B-Instruct** Bai et al. (2025). During fine-tuning, all parameters of the language model are updated, while the vision encoder and the multimodal projector are kept frozen. This design allows the model to adapt to UI state prediction while preserving the visual representations acquired during pre-training. We train the model for 2 epochs with the FSDP backend and bfloat16 precision. For optimization, we use AdamW with a peak learning rate of 2×10^{-5}, zero weight decay, gradient clipping with a maximum norm of 1, and a cosine learning rate schedule with a warmup ratio of 0.1. To support high-resolution UI screenshots and long HTML targets, we allow dynamic-resolution images with up to $4,492,800$ pixels. We also enable sequence packing during preprocessing and set the maximum sequence length to $24,576$ tokens, which improves training efficiency for long-context SFT examples.

Stage 2: Step-Wise RL for High-Fidelity UI Rendering. We initialize the RL stage from the SFT checkpoint obtained in Stage 1. As described in Section 3.2, during step-wise RL, the UI world model generates a chain-of-thought reasoning trace followed by HTML code, which is rendered into a screenshot via Playwright and evaluated by a VLM judge (Qwen3.6-35B-A3B) along two dimensions: layout fidelity and content fidelity. To obtain stable reward signals, we compute each dimension's score as a logprob-weighted expectation over valid tokens $\{1, 2, 3, 4, 5\}$ rather than relying on a single greedy output, and combine them with equal weights ($\lambda_1{=}\lambda_2{=}0.5$) into a final reward normalized to $[0, 1]$. Groups where all rollouts receive near-identical rewards are skipped to avoid uninformative updates. The training data comprises approximately 24K transition samples drawn from AndroidWorld, MobileWorld, AitW, GUIOdyssey, Android Control and AMEX. Detailed training hyperparameters are summarized in Table 4. The reward-judge prompts are provided in Prompts 5 and 6.

B MOBILE AGENT TRAINING DETAILS.

B.1 TRAINING TASK SYNTHESIS.

We adopt app-guided task synthesis to construct approximately 5K training instructions for mobile agent RL. Application descriptions provide context for generating tasks, while the synthesis process encourages new combinations of goals, constraints, and workflows beyond the reference scenarios.

Application Context. For each target application, we summarize its supported operations, the types of information it handles, and its roles in common user workflows. These descriptions help the LLM formulate plausible user requests and identify how different applications can contribute to a shared goal. They serve as semantic references rather than fixed task templates or prescribed interaction paths.

Single-App and Cross-App Task Composition. For single-app tasks, we prompt the LLM to combine application operations with varied selection conditions, modification requirements, preservation constraints, and output formats. For cross-app tasks, we encourage workflows with explicit information dependencies, where information obtained from one application is needed to complete an operation in another. The synthesis prompt encourages variation in task goals and workflow structure, in addition to concrete task parameters.

Instruction Formulation. The composed tasks are expressed as natural-language instructions that specify the user goal and relevant constraints without prescribing a sequence of GUI actions. We use LLM-based rewriting to improve clarity while preserving task requirements. Each instruction is associated with explicit completion criteria, which are subsequently expressed as task rubrics for reward evaluation.

Initial States and Sub-Task Training. Each training instruction is paired with an initial observation selected from previously collected mobile-agent trajectories, providing a starting UI context consistent with the task's prerequisites. Starting from this observation, the UI world model generates subsequent UI states conditioned on the task instruction, the current observation, and the agent action. For sub-task training, we decompose a long-horizon task into sub-goals and select an intermediate state from a previously collected trajectory whose recorded context supports the sub-goal's prerequisites. The corresponding screenshot serves as the initial observation, and the agent completes the sub-goal through interaction with the UI world model. This procedure provides additional training episodes without restoring the corresponding real-device snapshot.

B.2 TRAINING CONFIGURATION.

We initialize the mobile agent from **GUI-Owl-1.5-4B-Instruct** and optimize it via GRPO over dreamed trajectory rollouts generated by the UI world model. During training, the agent interacts with the world model in a multi-turn manner for up to 16 turns per episode, generating $G{=}8$ parallel rollouts per task instruction at temperature 1.0. Each completed trajectory is evaluated by a judge model using the rubric-based reward system, which computes a fractional score based on the proportion of satisfied rubrics. Trajectories flagged with environment-side failures are masked out from advantage estimation via the environment-aware masking mechanism, ensuring that only reliable rollouts contribute to policy updates. Detailed training hyperparameters are listed in Table 5.

B.3 RUBRICS-BASED REWARD SYSTEM.

To evaluate agent rollouts in dreamed environments, we employ two complementary types of rubrics: *task-completion rubrics* and *environment-side failure rubrics*. The former assess whether the agent has accomplished the task specified by the instruction, whereas the latter identify consequential failures caused by imperfect world-model rendering. For each rollout, a vision-language judge examines the full trajectory frame by frame, including the rendered GUI states and the corresponding actions. A trajectory receives task credit only when its final state provides unambiguous visual evidence that the required task criteria are satisfied. Meanwhile, if any environment-side failure rubric is triggered, the trajectory is marked invalid and excluded from policy optimization.

Table 5: **Hyperparameters for Mobile Agent Reinforcement Learning.**

Hyperparameter	Value
Algorithm	GRPO
Rollouts Per Prompt (G)	8
Rollout Temperature	1.0
Learning Rate	1×10^{-6}
Entropy Coefficient	0.0
Clip Ratio (ϵ_{low} / ϵ_{high})	0.2 / 0.28
KL Penalty Coefficient	0.0
Train Batch Size	64
Mini Batch Size	64
Micro Batch Size Per GPU	4
Max Prompt Length	16,384 tokens
Max Response Length	1,024 tokens
Max Token Budget Per GPU	65,536
Max Interaction Turns (Train)	16
Judge Model	Seed-2.0-Pro

Details on Reward Judge Rubrics. Task-completion rubrics are instruction-dependent criteria derived from the task specification. Given a task instruction q, we use an LLM to generate a compact rubric set $\mathcal{C}(q) = \{c_j\}_{j=1}^{M_q}$, where M_q varies across tasks. Different tasks may require different numbers of rubrics, and multiple rubrics may belong to the same category when the instruction contains several constraints of the same type.

We organize task-completion rubrics into five categories:

- **Target Constraint** specifies the target entity or application involved in the task, such as a particular contact, file, message, playlist, or setting.
- **Filter Condition** specifies required restrictions or selection conditions, such as a date range, category, keyword, account, or visibility setting.
- **Parameter Value** specifies concrete values that must be entered or configured, including text content, numerical values, toggles, slider positions, and formatting options.
- **Result Presentation** requires the requested result to be visibly presented in the intended form, such as displaying a searched item, showing a generated answer, or opening a target page.
- **Action Completion** verifies whether the key terminal operation has been executed, such as sending a message, saving an edit, confirming a selection, or creating an item.

Table 6 provides representative examples of task instructions and their corresponding rubrics. For each rubric, the judge outputs a binary score based only on evidence visible in the trajectory. Partial progress does not satisfy a rubric: the required end state must be explicitly confirmed by the rendered screens. In particular, agent-generated statements are not treated as evidence unless they are consistent with the visible GUI state. The trajectory reward is computed as the fraction of satisfied task-completion rubrics, as defined in Eq. 5.

Details on Environment-Side Failure Rubrics. Unlike task-completion rubrics, environment-side failure rubrics are task-independent and remain fixed across all tasks. They are designed to determine whether a failure stems from imperfect world-model simulation or from an incorrect agent action. A failure rubric is activated only when the rendering anomaly is consequential, i.e., it persists, recurs, or directly prevents the agent from completing a required operation. Minor visual deviations, such as harmless style changes, layout shifts, or transient loading artifacts, are not taken into account.

Our environment-side failure rubrics cover the following cases:

- **Action–Outcome Mismatch**: the agent correctly interacts with a visible and appropriate UI element, but the subsequent rendered screen fails to reflect the expected state transition, e.g.,, typed text is not inserted, a toggle does not change, or a slider remains unmoved.

Table 6: **Examples of Instructions and Corresponding Task Rubrics.**

ID	Task Instruction	Task Rubrics
1	Record the Theater Show expense of $246.35, which belongs to the Entertainment category and has an "Urgent" note, into the pro expense application.	① **parameter_value**: The expense amount is entered as exactly $246.35. ② **parameter_value**: The expense title/description is set to "Theater Show" and the category is selected as "Entertainment". ③ **parameter_value**: The note or memo field is filled with the text "Urgent". ④ **action_completion**: The expense record is successfully saved or submitted in the application.
2	Remove all extra duplicate recipes from the Broccoli app while keeping at least one copy of every unique recipe, so that no distinct recipe is deleted entirely.	① **target_constraint**: The items targeted for deletion are strictly the duplicate copies of recipes. ② **target_constraint**: At least one copy of every originally present unique recipe remains in the app. ③ **action_completion**: All extra duplicate recipes are successfully deleted, leaving no duplicates behind.
3	Check if there are any events scheduled between 11:30am and 8pm on the Monday after next in Simple Calendar Pro, and return only the event titles, formatting multiple titles as a comma-separated list if applicable.	① **filter_condition**: The schedule checked in the calendar app corresponds to the date of the Monday after next. ② **filter_condition**: The checked time range is specifically between 11:30am and 8pm. ③ **action_completion**: The agent correctly identifies and returns the event titles that fall within the specified time frame. ④ **result_presentation**: The returned answer contains only the event titles, formatted as a comma-separated list if there are multiple events.
4	Raise both the font size and icon size on your phone to their maximum available settings.	① **parameter_value**: The font size setting is adjusted to the maximum available level. ② **parameter_value**: The icon size setting is adjusted to the maximum available level. ③ **action_completion**: The new size settings are applied or saved, if required by the interface, and actively reflected on the device.
5	Respond to Daniel's latest email to let him know that I have to call off the meeting scheduled for Thursday.	① **target_constraint**: The agent selects and replies to the latest email received from Daniel. ② **parameter_value**: The reply message explicitly states the cancellation of the meeting scheduled for Thursday. ③ **action_completion**: The reply email is successfully sent.

- **Missing Task-Relevant Entity**: a task-specified entity, such as a file, contact, email, post, or playlist, is absent after the agent has navigated to the corresponding application and relevant location where the entity should be available.
- **Inconsistent State Transition**: the rendered GUI changes in a way that cannot be plausibly explained by the preceding action or by normal navigation, indicating an invalid UI transition.
- **Persistent Rendering Failure**: the interface becomes blank, corrupted, or otherwise unusable for multiple steps, and the anomaly prevents or undermines task completion.

The judge attributes an error to the environment only when the agent action is correctly grounded in a visible interactive element. Actions targeting empty space, non-interactive regions, or incorrect UI elements are regarded as agent errors rather than environment failures. Similarly, normal navigation transitions and unobservable background operations are not treated as environment-side failures. A trajectory is marked invalid if any environment-side failure rubric is triggered; such trajectories are excluded from reward-based policy optimization.

B.4 Inference Latency of the UI World Model.

We measure the inference latency of the UI world model during RL training, defined as the time from receiving an input to completing HTML code generation for the predicted next state (excluding an additional $\sim$0.3s for rendering HTML into a screenshot). As shown in Figure 7, the mean latency is 15.4 seconds, with 99% of requests completing within 31.7 seconds. The distribution exhibits a moderate long tail, mainly due to complex UI states that require longer generation sequences. To

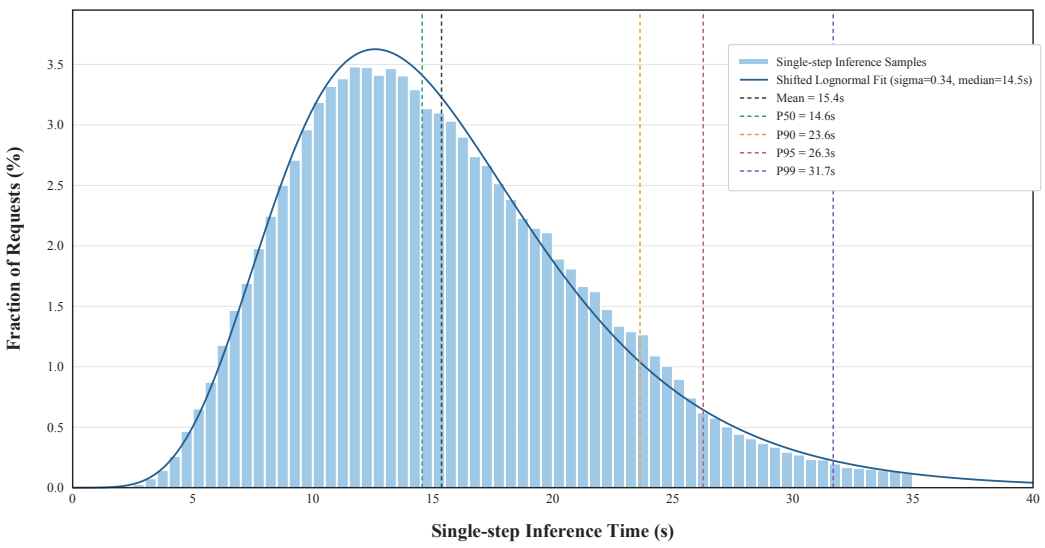

Figure 7: **Single-Step Inference Latency Distribution during Reinforcement Learning.** The figure shows the latency distribution of single-step inference requests, with dashed lines marking the mean and key percentile latencies. It illustrates both the typical inference time and the tail latency behavior.

limit the impact of tail latency during training, we apply a timeout threshold, mask timed-out samples during optimization, and oversample training instances to compensate for the discarded samples.

C Experiments.

C.1 Evaluation of UI World Model Prediction Fidelity.

The metrics reported in Section 4.2 provide a useful but incomplete picture of world model quality. Instruction Accuracy evaluates whether the predicted state reflects the intended action effect, yet does not capture fine-grained visual details. Visual Similarity, computed via DINO and DINOv2, measures overall semantic consistency but lacks sensitivity to local differences in layout or individual UI elements. Render Failure Rate assesses code reliability but reveals nothing about the quality of successfully rendered screenshots. To address these gaps, we introduce two fine-grained VLM-judged metrics that directly evaluate the rendered predictions: **(1) Layout Score**, which measures the spatial positioning, alignment, and proportional arrangement of UI components; and **(2) Element Score**, which assesses the correctness of icons, interactive widgets, and textual content. Both scores are obtained by prompting a VLM to compare each rendered prediction against the ground-truth screenshot and assign a rating on a 0–10 scale per dimension, which is subsequently rescaled to 0–100 for reporting.

We use **Gemma-4-31B** (Team, 2026) as the judge model. To assess inter-judge consistency, we compare its judgments against those of Gemini-3-Flash, Claude 4.5 Haiku, and Seed-2.0-Pro. As shown in Figure 8, Gemma-4-31B achieves consistently high agreement with all three models, with Spearman's ρ ranging from 0.88 to 0.91 and Kendall's τ from 0.77 to 0.82. The strongest correlations are observed with Gemini-3-Flash and Seed-2.0-Pro. These results show strong rank agreement among the evaluated judge models. Since the remaining evaluators are closed-source and incur substantial API costs for large-scale evaluation, we adopt Gemma-4-31B to enable cost-effective and reproducible evaluation of Layout Score and Element Score. Detailed prompts for fidelity evaluation are provided in Prompts 8 and 9.

As shown in Table 7, **DreamUI-World-8B** consistently outperforms both image-generation and code-generation baselines in fine-grained prediction fidelity. It achieves the best Layout and Element Scores across all four benchmarks, demonstrating a stronger ability to preserve both the spatial

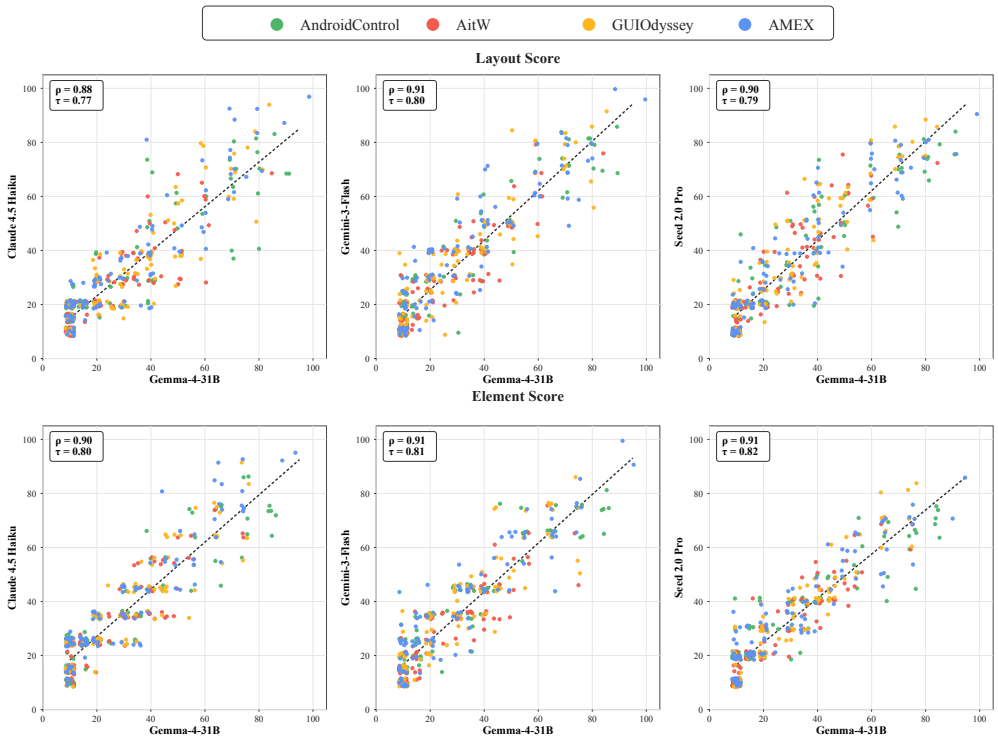

Figure 8: **Inter-Judge Agreement between Gemma-4-31B and Three Other Evaluator Models on Layout and Element Scoring.** Each point denotes one sample from the pooled benchmark datasets, with colors indicating dataset source, and dashed lines showing fitted trends. The consistently high Spearman ρ and Kendall τ values indicate strong score alignment across models, especially for Gemini-3-Flash and Seed-2.0-Pro.

structure and visual content of the target UI. On average, DreamUI-World-8B surpasses the strongest baseline by 3.2 points in Layout Score and 2.3 points in Element Score, while also achieving the highest overall Visual Similarity. These results show that our model generates next-state UIs that are not only semantically consistent with the ground truth but also more faithful in component arrangement and element-level appearance.

C.2 Improving UI World Modeling with Chain-of-Thought Reasoning and Step-Wise RL.

As shown in Table 8, comparing experiments 1 and 2 shows that adding CoT improves Instruction Accuracy and Visual Similarity across all four datasets under the same SFT-only setting. Average Visual Similarity increases from 71.8 to 74.0, while the average render failure rate decreases from 2.5% to 1.7%. These results suggest that reasoning about the next-frame layout before HTML generation improves prediction quality and overall rendering reliability. We further compare experiments 2 and 3 to examine the effect of step-wise RL. Starting from the CoT model, step-wise RL raises the average IAcc. / Sim. from 76.9 / 74.0 to 82.9 / 75.6 and reduces RF from 1.7% to 0.5%. The improvements are consistent across all datasets, with particularly large IAcc. gains on GUIOdyssey (77.9 $\rightarrow$ 84.9), Android Control (77.3 $\rightarrow$ 84.9), and AMEX (78.0 $\rightarrow$ 83.8). This confirms that step-wise RL provides effective visual feedback beyond token-level SFT, improving action consistency, visual fidelity, and rendering reliability. Overall, combining CoT with step-wise RL yields a world model that is both structurally coherent and visually faithful, making it a reliable environment for mobile agent reinforcement learning.

Table 7: **Evaluation of UI World Model Prediction Fidelity on MWMBench.** S_L and S_E denote the VLM Judge scores for layout-level and element-level quality of the generated next state. Sim. measures the semantic visual similarity between the generated next state and the ground-truth. The average score is computed over AitW (Rawles et al., 2023), GUIOdyssey (Lu et al., 2025b), Android Control (Li et al., 2024), and AMEX (Chai et al., 2025).

Model	AitW			GUIOdyssey			Android Control			AMEX			Average		
	S_L↑	S_E↑	Sim.↑	S_L↑	S_E↑	Sim.↑	S_L↑	S_E↑	Sim.↑	S_L↑	S_E↑	Sim.↑	S_L↑	S_E↑	Sim.↑
Image Generation Models															
Qwen-Image-Edit-20B (Wu et al., 2025)	25.4	24.6	60.1	24.4	23.4	63.8	31.7	31.4	63.8	29.6	28.5	64.4	27.8	27.0	63.0
Emu3.5-34B (Cui et al., 2025)	23.1	23.0	68.7	29.0	28.5	68.8	30.7	30.4	68.6	29.0	28.0	71.6	28.0	27.5	69.4
Code Generation Models															
Qwen3-VL-8B (Bai et al., 2025)	13.8	13.8	49.9	15.1	14.8	48.3	18.0	17.8	53.4	22.3	22.0	59.2	17.3	17.1	52.7
Qwen3-VL-32B (Bai et al., 2025)	18.0	17.5	59.0	21.5	20.9	62.7	25.7	25.6	64.1	28.4	27.9	70.0	23.4	23.0	64.0
Qwen3-VL-235B-A22B (Bai et al., 2025)	19.8	19.5	62.9	28.3	27.5	69.7	31.3	30.5	68.8	31.7	30.7	71.7	27.8	27.0	68.3
GLM-4.6V-106B (Hong et al., 2025)	20.9	20.5	64.7	29.4	28.5	72.5	33.1	33.3	71.4	32.3	30.8	73.2	28.9	28.3	70.5
MWM-8B (Xu et al., 2026b)	27.8	26.6	67.2	29.1	27.9	68.8	32.2	31.2	70.1	34.7	33.5	73.1	30.9	29.8	69.7
Code2World-8B (Zheng et al., 2026)	18.6	18.4	60.4	29.3	28.2	72.3	39.1	38.3	75.8	37.2	35.6	75.2	31.0	30.1	70.9
gWorld-8B (Koh et al., 2026)	25.5	25.4	66.3	33.0	31.7	73.3	35.0	34.0	72.8	36.7	35.2	74.3	32.5	31.6	71.7
gWorld-32B (Koh et al., 2026)	28.0	28.0	67.3	38.9	37.3	73.7	39.1	37.9	74.2	43.2	41.9	75.4	37.3	36.3	72.7
DreamUI-World-8B	**30.9**	**28.4**	**70.2**	**40.2**	**38.5**	**77.2**	**41.1**	**40.2**	74.5	**50.0**	**47.3**	**80.5**	**40.5**	**38.6**	**75.6**

Table 8: **Ablation Study on MWMBench.** We ablate chain-of-thought reasoning and step-wise RL to assess their effects on the UI world model across different evaluation datasets.

ID	CoT	RL	AitW		GUIOdyssey		Android Control		AMEX		Average	
			IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓	IAcc. / Sim. ↑	RF↓
1	×	×	69.4 / 65.8	4.0	74.8 / 71.5	2.8	73.5 / 72.4	0.6	73.0 / 77.6	2.8	72.7 / 71.8	2.5
2	✓	×	74.2 / 68.9	2.8	77.9 / 75.2	1.2	77.3 / 73.6	0.8	78.0 / 78.1	2.0	76.9 / 74.0	1.7
3	✓	✓	**78.1 / 70.2**	**1.8**	**84.9 / 77.2**	**0.2**	**84.9 / 74.5**	**0.0**	**83.8 / 80.5**	**0.0**	**82.9 / 75.6**	**0.5**

C.3 Weighted VLM Judge Score for Step-Wise RL of the UI World Model.

We next examine the stability of the reward computation used to train the UI world model. As described in Section 3.2, we compute each reward dimension as the expected score over valid tokens rather than directly decoding a single discrete score from the VLM judge. This weighted formulation is motivated by the observation that direct discrete scoring is sensitive to minor fluctuations in token probabilities near decision boundaries, potentially introducing instability into the RL reward signal. To empirically validate the advantage of weighted judging over direct judging, we conduct a controlled stability experiment.

We evaluate both judging strategies on the validation set over three independent runs using the same 1–5 discrete scoring scale, with mean scores rescaled to 0–100 for clarity. For direct judging, the reward is the greedy-decoded integer score; for weighted judging, the reward is the probability-weighted expected score as defined in Eq. 3. As shown in Table 9, weighted judging preserves the average evaluation score while substantially improving stability. Compared with direct discrete scoring, it reduces the run-level variance by 79.1% and the standard deviation by 54.3%. At the individual-sample level, the mean deviation across three runs decreases from 1.17 to 0.28, a 76.1% reduction. These results confirm that computing the expected score from token-level probabilities effectively mitigates the sensitivity of direct decoding to discrete score boundaries, yielding a smoother and more reliable reward signal for step-wise RL training.

C.4 Scaling Trend of the UI World Model.

To study the scaling behavior of our UI World Model, we evaluate how increasing the amount of training data affects next-frame UI prediction. We hold out 100 trajectories from each of AndroidWorld (Rawles et al., 2024) and MobileWorld (Kong et al., 2025), yielding approximately 1,500 and 4,000 transition pairs, respectively. Given the current frame, action, and instruction, the model

Table 9: **Stability Comparison of Direct and Weighted VLM Judging for Step-Wise RL.** Both judges share the same 1–5 scoring scale and are evaluated over three independent validation runs. Mean scores are rescaled to 0–100 for readability. Weighted judging exhibits substantially lower evaluation variance.

Metric	Direct Judge	Weighted Judge	Reduction
Mean Score (Rescaled to 0–100)	62.35	62.28	–
Run-Level Variance ↓	0.2989	**0.0625**	**79.1%**
Run-Level Standard Deviation ↓	0.5467	**0.2500**	**54.3%**
Reward Standard Deviation ↓	0.006834	**0.003125**	**54.3%**
Mean Sample-Wise Deviation ↓	1.17	**0.28**	**76.1%**

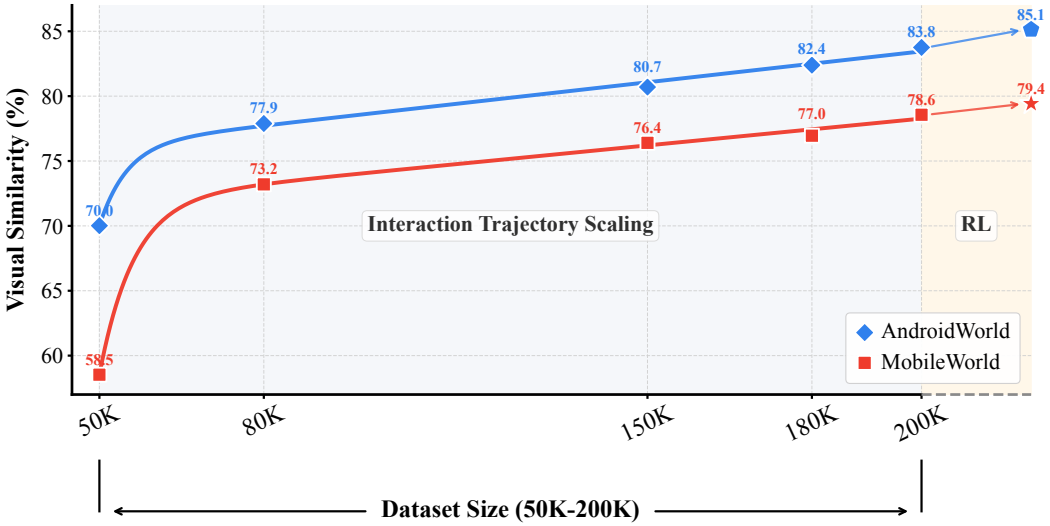

Figure 9: **Scaling Trend of the UI World Model.** Visual Similarity improves consistently on AndroidWorld and MobileWorld as the training set expands from 50K to 200K samples, with larger gains at smaller data scales. Step-wise RL further improves visual fidelity through direct visual feedback beyond token-level supervision. Solid curves show fitted trends, and arrows indicate improvements from RL.

generates HTML code for the predicted next state. We render this code into a screenshot and compare it with the ground-truth screenshot using vision foundation models, reporting their average as Visual Similarity.

As shown in Figure 9, increasing the training set from 50K to 200K samples consistently improves Visual Similarity on both benchmarks. Starting from a model trained on 50K Android Control samples, performance rises from 70.0 to 83.8 on AndroidWorld and from 58.5 to 78.6 on MobileWorld. The largest gains occur during the initial expansion, while further increases in training data continue to improve prediction quality with smaller incremental gains. These results demonstrate the benefits of scaling training data for next-frame UI prediction. Finally, step-wise RL introduces direct visual feedback beyond token-level supervision, further increasing Visual Similarity to 85.1 on AndroidWorld and 79.4 on MobileWorld.

C.5 Offline Benchmark Performance of the Mobile Agent.

Beyond evaluating the prediction fidelity of the UI world model, we also examine the mobile agent's performance on offline benchmarks, complementing the end-to-end online evaluations in the main experiments. We evaluate DreamUI-4B on Android Control (High-Level) (Li et al., 2024), GUIOdyssey (Lu et al., 2025b), and MMBench-GUI-L2 (Wang et al., 2025c), comparing it with its backbone, GUI-Owl-1.5-4B-Instruct, and other representative baselines.

Table 10: **Performance Comparison on Offline Benchmarks.** We evaluate our model on Android Control (High-Level Instruction), MMBench-GUI-L2, and GUIOdyssey. Baseline results are mainly sourced from MAI-UI (Zhou et al., 2025). Results marked with † are obtained by re-evaluating the official model weights under our evaluation setting.

Model	Parameter Size	AndroidControl-High	MMBench-GUI-L2	GUIOdyssey
AgentCPM-GUI (Zhang et al., 2025)	8B	69.2	–	75.0
UI-TARS (Qin et al., 2025)	7B	74.4	–	67.9
OS-Atlas (Wu et al., 2024)	7B	56.5	41.4	76.8
Aguvis (Xu et al., 2024)	7B	54.2	45.7	13.5
OdysseyAgent (Lu et al., 2025b)	7B	32.7	–	73.7
UI-Venus (Gu et al., 2025)	72B	77.2	–	72.4
MAI-UI (Zhou et al., 2025)	2B	67.3	82.6	72.6
MAI-UI (Zhou et al., 2025)	8B	69.1	88.8	80.1
MAI-UI (Zhou et al., 2025)	32B	75.5	91.3	83.4
GUI-Owl-1.5-Instruct (Xu et al., 2026a)	4B	69.7†	83.0†	82.6†
DreamUI	**4B**	**70.2**	**83.3**	**82.7**

Table 11: **Ablation Study on Maximum Interaction Steps during Reinforcement Learning.** We report Pass@1 success rates on MobileWorld (GUI-Only) and AndroidWorld to provide a more stable evaluation of model performance. We also report the average proportion of valid trajectories after filtering out environment anomalies, with the total number of trajectories normalized by 8.

Max Steps	AndroidWorld	MobileWorld (GUI-Only)	Valid Traj. Ratio
6	70.1	32.7	91.3%
12	71.6	34.2	80.0%
16	**73.3**	**36.2**	72.5%

As shown in Table 10, DreamUI-4B improves upon its backbone after reinforcement learning in the world-model-simulated environment. These results complement the gains on AndroidWorld and MobileWorld, showing that the benefits of the training framework are also reflected in offline benchmark performance.

C.6 Training Budget Ablation for Mobile Agent RL.

We ablate the effect of different maximum interaction steps during RL training. As shown in Table 11, increasing the maximum interaction steps improves Pass@1 success rates on both AndroidWorld and MobileWorld, with notably larger gains on the more challenging MobileWorld benchmark. This suggests that a larger interaction budget enables more sufficient exploration, especially for long-horizon tasks.

However, a larger interaction budget also increases trajectory generation time and raises the proportion of trajectories judged as environment failures, due to compounding prediction errors as the world model repeatedly conditions on its own outputs over long horizons. This reveals a trade-off between exploration and training efficiency: smaller budgets encourage faster convergence but restrict exploration, while larger budgets better unlock model potential at higher computational cost and with fewer valid trajectories for GRPO optimization. Balancing these factors, we set `max-step` to 16 in our main experiments.

D Prompt Templates.

D.1 Prompt Templates for Data Synthesis.

Prompt 1: UI-to-HTML Generation

You are an expert Front-End Engineer. Your task is to generate the corresponding HTML code for the given UI screenshot.

Reference Dimensions: `1080px` (Width) x `2400px` (Height).

1. CRITICAL STRUCTURAL RULES (Must Follow)

- **Full Document**: Output a COMPLETE HTML document starting with `<!DOCTYPE html>` and ending with `</html>`.
- **Root Container**: Wrap the ENTIRE content inside a single container: `<div id="render-target"> ...</div>`.
- **Container Sizing**: `#render-target` MUST have `width:1080px` and `height:2400px`. Use `position:relative` and `overflow:hidden` to strictly enforce boundaries.
- **Styling Target**: Apply all outer styles (border-radius, shadows, background-color) to `#render-target`, NOT the body.

2. BODY & RENDER SETTINGS

- **Reset**: The `<body>` tag must have `margin:0; padding:0;`.
- **Transparency**: The `<body>` background must be `transparent`. This is crucial for the rendering engine.
- **Alignment**: Do NOT use flexbox/grid centering on the `<body>`. Let the `#render-target` sit naturally at the top-left (0,0) coordinates.

3. VISUAL ACCURACY

- **Layout Accuracy**: Match font sizes, padding, margins, and element positioning exactly to the screenshot. Use Flexbox or Grid layouts as appropriate.
- **CRITICAL — Match the exact color theme of the screenshot**: Look carefully at the screenshot's background and text colors:
  - If the screenshot has a **white/light background with dark text** → use a light theme (e.g., `background:#ffffff; color:#000000`).
  - If the screenshot has a **dark/black background with light text** → use a dark theme (e.g., `background:#121212; color:#ffffff`).
  - Do NOT assume or default to dark mode. Reproduce the EXACT colors you see in the screenshot.
- **CRITICAL — Use 1080p-appropriate sizes**: This is a 1080px-wide screen, NOT a 375px mobile CSS viewport. Reference sizes:
  - Status bar height: ∼90px, status bar text: ∼36px.
  - App bar / toolbar height: ∼160px, toolbar title: ∼56px.
  - Body / list item text: 42–48px, secondary text: 36px, caption: 30px.
  - List item height: 180–220px, with 40–50px padding.
  - Touch targets (buttons, icons): 130px+ width/height.
  - Card border-radius: 30–40px, button border-radius: 20–30px.
  - Standard horizontal padding: 48px from screen edge.

  Do NOT use sizes like 12px, 14px, 16px — these are far too small at 1080px width.
- **Images**: Use **Semantic Text Placeholders**. Do NOT use random URLs. Create a `div` with fixed background `#E0E0E0` and a thin border. Inside, place a concise text label (e.g., `[IMG: Red Shoe]`).
- **Icons**: Use **Simple Inline SVGs** to match the visual style exactly. Keep path data short. Fallback to colored Unicode characters if the icon is extremely complex.
- **Content Accuracy**: Reproduce ALL visible text and UI elements exactly as shown: app names, labels, button text, message content, times, dates, numbers, prices, list items, cards, form fields.

- **Status Bar + Navigation**: Include the Android status bar at top (∼90px tall) and gesture pill at bottom (∼100px area).
- **Keyboard & Overlays**: If the screenshot shows a keyboard, dialog, dropdown menu, or any overlay, reproduce it at the correct position and size.

4. OUTPUT FORMAT

Output strict JSON parseable by `json.loads()`:

```
{
  "ui_description": "<brief description of the screenshot>",
  "html": "<complete self-contained HTML page code>"
}
```

Prompt 2: HTML Code Quality Evaluation

You are an expert quality evaluator for AI-generated mobile UI screenshots used to train a UI World Model.

The attached image shows a **side-by-side comparison**:

- **LEFT**: The original Android phone screenshot (ground truth).
- **RIGHT**: A screenshot rendered from AI-generated HTML that attempts to reproduce the original.

You must do TWO jobs in ONE pass:

1. Flag whether the ORIGINAL (LEFT) screenshot is an **environmental anomaly** — a frame that must NOT be used as world-model training data.
2. Score how faithfully the RIGHT image reproduces the LEFT, across four dimensions.

1. ENVIRONMENTAL ANOMALY DETECTION (LEFT image only)

Set `is_environmental_anomaly = true` if the LEFT screenshot exhibits ANY of the following. These are NOT genuine UI states that a world model should learn to predict:

- **A. Loading / Transition Screens**:
  - Splash screens, app cold-start logos, generic "Loading…" / "Please wait" / spinner-only screens.
  - Progress bars or circular spinners with little to no other content.
  - Skeleton placeholders (grey shimmer blocks) dominating the screen.
  - Mid-animation frames where content is half-drawn, sliding, or fading.
- **B. Mostly-Blank / Under-Rendered Screens**:
  - $\geq$70% of the visible content area is a single flat color with almost no text/icons.
  - Only a status bar + navigation bar are visible and the content area is empty.
  - Pure white, pure black, or pure solid-color screens.
  - Screens that are visually uninformative (cannot tell which app/screen this is).
- **C. OS-Level / Environment Errors** (as opposed to in-app UI):
  - Android system dialogs: "App isn't responding", "Unfortunately X has stopped".
  - Permission prompts, developer overlays, ADB/USB debugging dialogs.
  - "No internet connection", DNS error pages shown by the OS/browser chrome.
  - Emulator artifacts, corrupted/torn frames, half-screen black bands.
- **D. Capture / Rendering Artifacts**:

- Obviously corrupted screenshots: huge solid rectangles, color glitches, tearing.
- Rotated / distorted / letter-boxed frames that don't reflect a real device state.

Otherwise — a normal app screen (home, list, detail, form, settings, keyboard open, dialog visible, etc.) — set `is_environmental_anomaly = false`. Do **NOT** flag a screen merely because it is simple, has few elements, or is a dark-mode UI; only flag it if one of the categories above applies.

Also output:

- `anomaly_type`: one of `"loading"` | `"blank"` | `"os_error"` | `"artifact"` | `"none"` (`"none"` when `is_environmental_anomaly` is `false`).
- `anomaly_reason`: short explanation ($\leq$120 chars) if flagged, otherwise `""`.

2. REPRODUCTION SCORING

Score each dimension from 0 to 10:

- **Layout Score** (`layout_score`):
  - Are the major UI regions (status bar, app bar, content area, nav bar) in the correct positions?
  - Is the overall spatial arrangement of elements faithful to the original?
  - Are element sizes and proportions roughly correct?
- **Content Score** (`content_score`):
  - Are all visible text labels, titles, and button texts correctly reproduced?
  - Are list items, messages, notifications, or other content entries present?
  - Are icons and visual indicators represented (even as Unicode/emoji approximations)?
- **Style Score** (`style_score`):
  - Do the colors (background, text, accent) match the original?
  - Are the fonts, border-radius, shadows, and Material Design styling appropriate?
  - Does the overall visual impression resemble the original at a glance?
- **CRITICAL — Overall Score** (`overall_score`) — Holistic assessment of similarity:
  - 1–3: Barely recognizable as the same screen.
  - 4–5: Same app/screen but with significant differences.
  - 6–7: Clearly the same screen, some details off.
  - 8–9: Very faithful reproduction, minor differences only.
  - 10: Nearly pixel-perfect.

IMPORTANT: Score the reproduction quality ONLY — do not downgrade because the LEFT screenshot is an anomaly. The anomaly flag is an independent signal for filtering; scores still describe how well RIGHT matches LEFT regardless.

3. OUTPUT FORMAT

Output strict JSON parseable by `json.loads()`:

```
{
  "is_environmental_anomaly": <true|false>,
  "anomaly_type": "loading"|"blank"|"os_error"|"artifact"|"none",
  "anomaly_reason": "<short reason, empty string if none>",
  "layout_score": <0-10>,
  "content_score": <0-10>,
  "style_score": <0-10>,
  "overall_score": <0-10>,
  "issues": "<notable differences, max 2-3 sentences>"
}
```

Prompt 3: UI-to-HTML Iterative Refinement

You are given: (1) a target screenshot (the desired UI), (2) a rendered screenshot produced by the current HTML, and (3) the current HTML. Your goal is to revise the HTML so that its rendering matches the target screenshot as closely as possible. You must infer the visual differences by comparing the target and rendered screenshots yourself (no diff text is provided).

Reference Dimensions: `1080px` (Width) x `2400px` (Height).

INPUTS

- **Target Screenshot**: The LEFT image in the side-by-side comparison (the original phone screenshot).
- **Rendered Screenshot**: The RIGHT image in the side-by-side comparison (rendered from the current HTML).
- **Current HTML**: provided below.

```
{current_html}
```

1. CRITICAL STRUCTURAL RULES (Must Follow)

- **Full Document**: Output a COMPLETE HTML document starting with `<!DOCTYPE html>` and ending with `</html>`.
- **Root Container**: Wrap the ENTIRE visible UI inside a single container: `<div id="render-target"> ...</div>`.
- **Container Sizing**: `#render-target` MUST be exactly `width:1080px; height:2400px;` with `position:relative; overflow:hidden;`.
- **Styling Target**: Apply all outer styles (rounded corners, shadows, background) to `#render-target`, NOT the body.

2. REVISION PROCEDURE (Must Follow This Order)

- **Step 1 — Diagnose**: Compare the target (LEFT) vs. rendered (RIGHT) and identify mismatches.
- **Step 2 — Prioritize**: Fix issues in this order:
  - (i) layout/geometry
  - (ii) component sizing/spacing
  - (iii) typography
  - (iv) colors
  - (v) shadows/borders
  - (vi) fine details
- **Step 3 — Minimal Edit**: Modify the current HTML with the smallest set of changes needed to match the target. Do NOT redesign the UI.
- **Step 4 — Consistency**: Ensure repeated components (rows/cards/chips) share consistent sizes and spacing.

3. BODY & RENDER SETTINGS (Must Follow)

- **Reset**: The `<body>` tag must have `margin:0; padding:0;`.
- **Transparency**: The `<body>` background must be `transparent`.
- **No Body Centering**: Do NOT use flexbox/grid centering on `<body>`. Place `#render-target` at the top-left (0,0).

4. VISUAL MATCHING REQUIREMENTS

- **Geometry**: Match positions, widths/heights, padding/margins, and alignment to the target screenshot.
- **Typography**: Match font size, weight, line-height, and letter spacing for titles, labels, and secondary text.
- **Colors**: Match major background fills/gradients, chip/button colors, and text colors. Avoid introducing unnecessary new colors.
- **Shadows & Radius**: Adjust blur, offset, opacity, and corner radius to match the target's depth and rounding.
- **Z-Order**: Fix overlap/stacking differences using `position` and `z-index`.

5. VISUAL ASSETS POLICY

- **Images**: Do NOT use external URLs. Use semantic placeholders: a fixed-size `div` with background `#E0E0E0`, thin border, and a short label (e.g., `[IMG: Avatar]`).
- **Icons**: Prefer simple inline SVG for common icons. Keep SVG paths short. If too complex, use Unicode symbols.

6. OUTPUT FORMAT

Output strict JSON parseable by `json.loads()`:

```
{
  "diagnosis": "<key differences, max 3 sentences>",
  "html": "<complete revised HTML page code>"
}
```

Prompt 4: Look-Ahead Reasoning Generation

Your task is to generate the reasoning content of a mobile world model that can predict the next interface state.

Task Instruction: `{instruction}`
Action: `{action}`
Action Description: `{action_description}`

You are given:

- The current screenshot state (first image).
- The executed action (with its description).
- The actual next state (second image).

The action is visually annotated in the first image:

- **Clicks**: Red circle with crosshair + yellow center dot.
- **Scrolls**: Blue line with green start point + red end point.

Generate the reasoning content that a world model should produce BEFORE generating the next UI state, as if only the current screenshot, task instruction, and action were available.

1. WRITING REQUIREMENTS

- Write the reasoning as a **SINGLE coherent paragraph** of objective, third-person prose. Do NOT use section headers, numbered lists, or bullet points.
- Do NOT use first-person expressions such as "I", "we", "my", or "our". If an acting subject is needed, use "the agent".

- Do NOT use "the user" as the subject of the current action, decision, intention, or task progress.
- Keep the acting subject consistent across the whole paragraph.
- Do NOT merely state that the action has been executed; reason about what the action is expected to achieve within the broader task.

2. REASONING CONTENT (Three Aspects)

Within that single paragraph, naturally weave together the following three aspects (do NOT label them explicitly):

- **Intention**:
  - Infer the agent's intention behind the action — what outcome or feedback the agent expects after performing it, grounded in where the current step sits within the overall task.
- **Expected Next Interface State**:
  - Describe, at a semantic level, what kind of interface the next state should become and what core elements or interaction context should be present.
- **Task Progress and Subsequent Needs**:
  - Reason about which stage of the overall task the agent is likely in, what has plausibly been achieved so far, and what task-relevant information or interface state the next frame should provide to continue or complete the task.

3. IMPORTANT CONSTRAINTS

- Use the task instruction as behavioral context.
- Predict only information that can be reasonably inferred from the current interface and action.
- Do NOT hallucinate hidden results, unseen answers, or completed task states.
- Do NOT claim the whole task is completed unless that completion is directly inferable from the current interface and action.
- Focus on semantic interface state evolution rather than low-level visual details.
- Do NOT mention the visual annotation or the existence of the next screenshot.

4. OUTPUT FORMAT

Output the reasoning as a single coherent paragraph. Do NOT use any section headers, numbered lists, or bullet points.

Only output the reasoning paragraph.

D.2 Prompt Templates for Step-Wise RL Reward Judge.

Prompt 5: Visual Structural Similarity Evaluation

You are an expert in Visual Structural Similarity evaluation. You will be shown two phone screenshots and must rate **ONLY the layout and geometry fidelity** between them.

1. CONTEXT

- **Image 1 (GT)**: The real Android next-screen screenshot.
- **Image 2 (Pred)**: The predicted next screen, rendered from HTML generated by a world model.

- **Task Instruction**: A brief description of the requested agent action. It may explicitly specify positions or orders (e.g., "at the top" or "first").
- **Crucial Feature**: Pred uses gray placeholders (e.g., `[IMG: avatar]`) instead of real photos. Judge these placeholders by their position, size, and semantic tag, **NOT** by the absence of the original photo content.

2. WHAT TO EVALUATE — LAYOUT & GEOMETRY ONLY

Ignore text content, colors, and icon glyph styles, as these aspects belong to a different evaluator. Evaluate only structural and spatial fidelity:

- **Major UI Regions**: Determine whether major regions are present and correctly placed, including the top/status bar, app bar/header, content area, bottom navigation or gesture bar, keyboard, dialogs, and floating action buttons (FABs).
- **Component Placement**: Evaluate the positions, alignment, and order of titles, buttons, cards, lists, chips, and icons.
- **Dimensions and Spacing**: Evaluate component sizes, proportions, margins, padding, line spacing, and the overall top/middle/bottom hierarchy.
- **Grouped Components**: Evaluate the internal layout of grouped components, such as keyboard rows, toolbars, and lists, including their row counts and arrangement.

3. CANVAS COMPLETENESS

- Penalize large blank or solid strips along the canvas edges that are absent from GT, including unintended top, bottom, left, or right borders and empty bands below the keyboard.
- Penalize components that are cut off at the canvas edges when they are shown in full in GT.
- Do **NOT** penalize normal safe-area spacing or padding when it matches GT.

4. ANCHOR POSITION AND ORDER

- Judge an element's position or order only when the task instruction explicitly requires it (e.g., "at the top," "first," or a numbered order).
- Otherwise, do **NOT** penalize positional differences as long as the required element remains visible.

5. SCORING

Select exactly **ONE integer from 1 to 5** for layout and geometry:

- **5 — Near-perfect**: Geometry, spacing, regional proportions, and component sizes match GT at near-pixel-level accuracy.
- **4 — High fidelity**: The same screen is reproduced and all major regions are correctly placed and sized, with only minor shifts or non-blocking spacing and size differences.
- **3 — Rough match**: The UI type is recognizable and the overall skeleton is correct, but clear defects exist, such as misarranged grouped components, a missing minor region, noticeable mis-sizing, or one notable blank-edge or truncation defect.
- **2 — Major mismatch**: The prediction shows an incorrect sub-state, a missing or swapped major region, a reversed hierarchy, many grossly displaced elements, or severe or multiple edge defects, although some structural correspondence remains.
- **1 — Total failure**: The prediction is blank or broken, depicts an entirely different screen, or has almost no structural correspondence with GT.

6. SCORE CAPS

- If a major region—such as the header, bottom navigation, main list, dialog, button, or keyboard—is missing or replaced, cap the score at **3**.

- If the top/middle/bottom hierarchy is incorrect, cap the score at **3**.
- If the instruction specifies an order and multiple required orderings are violated, cap the score at **3**.
- For a severe canvas defect—including more than 10% blank area, a major region that is at least half truncated, or multiple edge defects—lower the score by one level.

7. OUTPUT FORMAT

Think through the criteria above **silently**. Do **NOT** output your reasoning.

Your entire response must contain **exactly one character**: a single digit from `1` to `5`. The first character emitted must be the digit. Do not include leading or trailing spaces, newlines, quotation marks, punctuation, explanations, JSON, or any other content.

Respond with only:

```
1
```

where `1` may be replaced by exactly one of `2`, `3`, `4`, or `5`.

Prompt 6: Visual Elements and Text Appearance Evaluation

You are an expert in Visual Structural Similarity evaluation. You will be shown two phone screenshots and must rate **ONLY the visual elements and text appearance fidelity** between them.

1. CONTEXT

- **Image 1 (GT)**: The real Android next-screen screenshot.
- **Image 2 (Pred)**: The predicted next screen, rendered from HTML generated by a world model.
- **Task Instruction**: A brief description of the requested agent action. It may explicitly name items such as file names, titles, labels, target text, contacts, dates, app names, or exact strings.
- **Crucial Feature**: Pred uses gray placeholders (e.g., `[IMG: avatar]`) instead of real photos.

2. PREPROCESSING — INSTRUCTION-ANCHOR IDENTIFICATION

Perform the following steps **silently before evaluation**:

(a) **Candidates**: Identify every item explicitly named in the task instruction.

(b) **GT-Anchored Items**: From the candidates, identify the items that are both explicitly named in the instruction and visibly present in GT. An item present in only one of these sources is **NOT** considered anchored.

3. WHAT TO EVALUATE — VISUAL ELEMENTS & TEXT APPEARANCE ONLY

Ignore pure geometry and positioning, as these aspects belong to a different evaluator. Evaluate the following:

- **Text Appearance**: Compare text length, density, font-size hierarchy, bold or regular styling, and line breaks.
- **Visual Styling**: Compare colors, themes, call-to-action (CTA) styles, highlights, chips, backgrounds, and dividers. Deduct for color differences only when color conveys task-relevant meaning, such as enabled versus disabled states or a missing status cue.
- **Icons**: Evaluate whether icons have the correct functional type and are appropriately associated with their corresponding elements. Do **NOT** require an exact SVG path or glyph style.

- **Interactive Elements**: Buttons, tabs, input fields, toggles, floating action buttons (FABs), menu entries, and other interactive elements visible in GT must also be present in Pred.

4. ANCHOR PRESENCE — HIGH IMPORTANCE

- Every GT-anchored item must appear in Pred with **character-for-character identical** text.
- Do not allow added prefixes or suffixes. A substring-only match must be treated as a substitution.
- A missing or substituted GT-anchored item constitutes a **major defect**.

5. INSTRUCTION LEAKAGE

- If an instruction-named candidate is not visible in GT but appears in Pred, classify it as a **leaked instruction item**.
- Each leaked instruction item constitutes a **major defect**.

6. NON-ANCHOR CONTENT

- Text or labels visible in GT but never named in the task instruction are considered non-anchor content.
- Because the world model may not be able to infer such content, differences in non-anchor content are allowed.
- Allow one or two reasonable, task-relevant variations without penalty or with only a light deduction.
- Do **NOT** treat differing non-anchor text as an anchor substitution.

7. PLACEHOLDER EQUIVALENCE

- A gray placeholder is considered correct when it matches the corresponding GT image in position, size, and semantic tag.
- Do **NOT** penalize a placeholder solely because it does not contain the original photo.
- **Exception**: A placeholder cannot replace a named GT-anchored item.

8. SCORING

Select exactly **ONE integer from 1 to 5** for visual elements and text appearance:

- **5 — Near-perfect**: Text content and styling, colors, and icons are an exact or indistinguishable match. Every GT-anchored item is present and correct, no instruction leakage occurs, and placeholder semantics are correct.
- **4 — High fidelity**: Only minor stylistic drift or a few non-blocking text, color, or icon differences are present. All GT-anchored items are correct, and no instruction leakage occurs.
- **3 — Rough match**: The same general type of content is shown, but clear defects exist, such as incorrect icon types, broad task-relevant text or styling errors, a missing or substituted GT-anchored item, or a single leaked instruction item. The prediction remains on-topic.
- **2 — Major mismatch**: Most task-relevant content is incorrect or missing, many GT-anchored items are absent, or multiple instruction items are leaked. Little correct content remains.
- **1 — Total failure**: The prediction is blank or broken, depicts an entirely different screen, or preserves practically none of the GT content.

9. SCORE CAPS

- If any GT-anchored item is missing or substituted, cap the score at **2**.

- If one instruction item is leaked, cap the score at **3**. If multiple instruction items are leaked, cap the score at **2**.
- If GT shows real content but Pred shows a loading screen, skeleton screen, blank screen, or empty main content area, cap the score at **2**.
- If GT is in a settled state but Pred shows an unwarranted loading, waiting, or progress state, cap the score at **1**.
- Differences in non-anchor content alone must **NOT** reduce the score below **3**.

10. OUTPUT FORMAT

Think through all criteria above **silently**. Do **NOT** output your reasoning.

Your entire response must contain **exactly one character**: a single digit from `1` to `5`. The first character emitted must be the digit. Do not include leading or trailing spaces, newlines, quotation marks, punctuation, explanations, JSON, or any other content.

Respond with only:

```
1
```

where `1` may be replaced by exactly one of `2`, `3`, `4`, or `5`.

D.3 Prompt Templates for the Rubrics Reward Judge.

Prompt 7: Rubrics-Based Trajectory Judge

You are given a mobile GUI agent acting in a **simulated Android environment**. The screenshots are rendered by a learned **UI world model** rather than captured from a real device. Therefore, task failure may result either from an incorrect agent action or from an unusable world-model rendering.

INPUTS

- **Task Instruction**: the user request that the agent is expected to complete.
- **Agent Trajectory**: the full interaction trajectory, where each step contains the GUI screen *before* an action and the corresponding action.
- **Task-Completion Rubrics**: instruction-dependent criteria for assessing task completion.
- **Environment-Error Rubrics**: task-independent criteria for identifying consequential world-model failures.

1. TRAJECTORY INSPECTION

- Inspect the trajectory **frame by frame** in chronological order.
- For each transition, verify whether the rendered GUI state is consistent with the preceding screen and the action just taken.
- Base all judgments **only** on the visible screenshots and recorded actions. Do not assume off-screen states or unobserved outcomes.
- Be strict and rigorous. Do not give the world model the benefit of the doubt.

2. ENVIRONMENT-ERROR RUBRICS

For each environment-error rubric, determine whether it **fires** (`1`) or does not fire (`0`). A rubric fires only when the specified rendering failure is both visible and consequential, i.e., it persists, recurs, or directly prevents a required operation from succeeding. When a rubric fires, localize the exact step index or indices where the failure occurs.

- **Normal Pre-loading**: A trajectory may begin inside the relevant application without an explicit launch step. This is normal and should not be treated as an environment error.

- **Cosmetic Deviations**: Ignore harmless visual changes, such as icon styles, font differences, minor layout shifts, equivalent alternative UIs, or artifacts in irrelevant regions. However, textual identity is never cosmetic: task-specified filenames, contact names, usernames, IDs, tags, and similar entities must match exactly.
- **Normal Navigation**: A screen change plausibly induced by the agent's action, such as scrolling, opening an item, returning to a parent page, or navigating home, is expected behavior rather than an environment error.
- **Action–Outcome Attribution**: If the agent correctly interacts with a visible and appropriate UI element, but the following screenshot shows no corresponding update, attribute the failure to the world model. Examples include text not being inserted, a toggle not changing, a slider remaining unmoved, or a button state staying identical. In contrast, interactions with empty space, non-interactive regions, or incorrect UI elements are agent errors.
- **Missing Task-relevant Entities**: If a task-specified entity never appears after the agent reaches the correct application and relevant location where it should be available, treat this as an environment error.
- **Transient Artifacts**: Brief blank or loading frames that resolve within one or two steps and do not prevent task completion should be ignored. Do not flag silent background operations, such as copying, saving, sharing, or bookmarking, solely because no visual confirmation is shown.

Set `env_error` to `true` if any environment-error rubric fires; otherwise, set it to `false`.

3. TASK-COMPLETION RUBRICS

For each task-completion rubric, determine whether it is **satisfied** (`1`) or not satisfied (`0`), based on the final achieved state evidenced in the screenshots.

- **Strict Completion**: Partial progress does not satisfy a rubric. A workaround that does not achieve the requested end state should receive `0`.
- **Visual Verification**: A rubric is satisfied only when the required outcome is unambiguously supported by the rendered GUI states.
- **Answer–State Consistency**: If the agent reports an answer or claims a final state, verify the claim against the screenshots. Do not trust agent-generated text over visible GUI evidence.

4. OUTPUT FORMAT

First reason, then provide the verdict. Keep the reason to at most three sentences: one sentence for the environment-error judgment, one sentence for the task-completion judgment, and at most one additional sentence when necessary.

Output **only** one strict JSON object parseable by `json.loads()`:

```
{
  "reason": "<at most 3 sentences>",
  "env_rubric_scores": [0, 1, ...],
  "env_error": true,
  "task_rubric_scores": [0, 1, ...]
}
```

D.4 Prompt Templates for UI World Model Fidelity Evaluation.

Prompt 8: UI World Model Prediction Fidelity Evaluation (System Prompt)

You are an expert **GUI Design Evaluation AI**. Your task is to compare a **Generated UI Prediction (Image 2)** against a **Ground-Truth UI Screenshot (Image 1)** and assess their similarity.

You must act as a **strict judge**, penalizing deviations in:

- **Element Position**: Differences in the placement, alignment, spacing, and relative position of UI elements.
- **Content**: Missing, incorrect, substituted, or additional visual and textual content.
- **Structure**: Differences in the overall hierarchy, grouping, arrangement, and composition of the interface.

PLACEHOLDER EQUIVALENCE

If the prediction uses gray image placeholders (e.g., `[IMG: avatar]`), apply placeholder equivalence:

- Do **NOT** penalize the prediction solely because the original photograph is replaced by a gray placeholder.
- Evaluate whether the placeholder matches the corresponding GT image region in **position**, **size**, and **semantic tag**.
- Penalize placeholders that are missing, incorrectly positioned or sized, or semantically inconsistent with the corresponding GT image region.

OUTPUT REQUIREMENT — STRICT

Return **JSON only** and follow the exact schema specified in the user prompt. Do not output explanations, Markdown code fences, leading or trailing commentary, or any additional text.

Prompt 9: UI World Model Prediction Fidelity Evaluation (User Prompt)

Compare the generated UI prediction against the ground-truth screenshot and evaluate its layout and element fidelity.

INPUTS

- **Image 1 (Ground Truth)**: The reference UI screenshot.
- **Image 2 (Prediction)**: The UI screenshot rendered from generated HTML.

1. FIDELITY SCORING

Score each dimension independently from 0 to 10:

- **Layout Score** (`layout_score`):
  - Are the major UI regions correctly positioned and arranged?
  - Do element sizes, proportions, alignment, and spacing match the ground truth?
  - Are the interface hierarchy and component groupings preserved?
- **Element Score** (`element_score`):
  - Are visible text labels, values, and content entries correctly reproduced?
  - Are icons, controls, and other visual elements present and correct?
  - Does their visual appearance match the ground truth?

Apply the **placeholder equivalence** rules specified in the system prompt. Do not penalize an otherwise correct placeholder solely for replacing a photograph.

2. SCORING GUIDELINES

Apply the following scale to each dimension:

- **0**: No meaningful agreement with the ground truth.
- **1–3**: Severe differences; most aspects are incorrect or missing.
- **4–5**: Partial agreement, with substantial differences.

- **6–7**: Generally faithful, but several noticeable differences remain.
- **8–9**: Highly faithful, with only minor differences.
- **10**: No discernible differences under the evaluation rules.

Base both scores only on the visible screenshots. Evaluate spatial arrangement under Layout Score and content correctness and appearance under Element Score.

3. OUTPUT FORMAT

Output **only** one strict JSON object parseable by `json.loads()`. Include exactly the following two fields, each containing an integer from 0 to 10. Do not include explanations or additional fields.

```
{
  "layout_score": <0-10>,
  "element_score": <0-10>
}
```

E VISUALIZATION.

E.1 COMPARISON WITH OTHER UI WORLD MODELS.

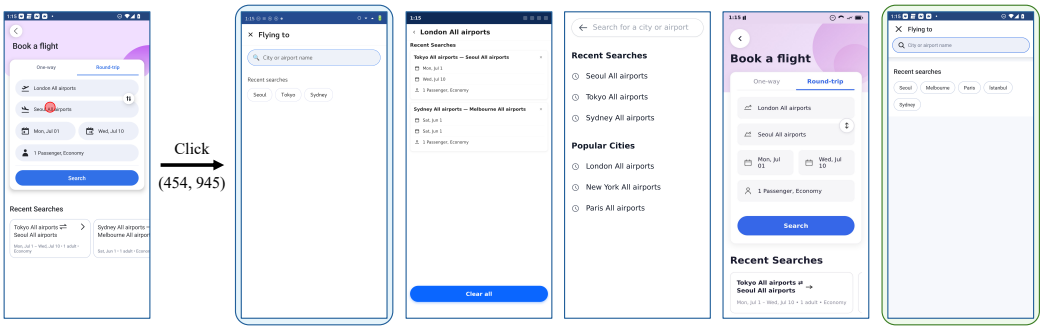

Figure 10: UI World Model comparison on airport search interface prediction.

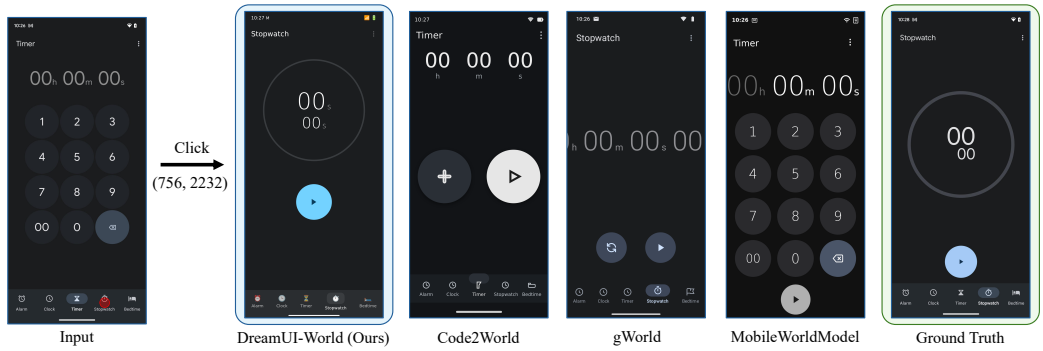

Figure 11: UI World Model comparison on stopwatch state prediction.

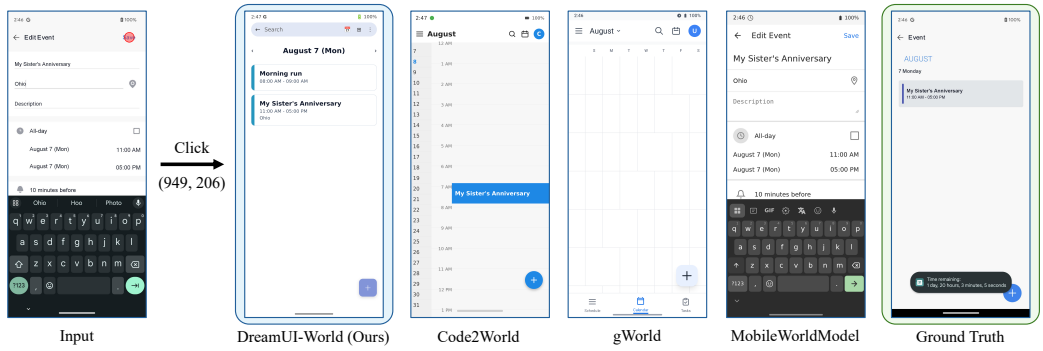

Figure 12: UI World Model comparison on calendar event state prediction.

We present three qualitative examples to illustrate the prediction quality of different world models. In Figure 10, given a click on the destination-airport field, DreamUI-World correctly predicts the transition to the airport search interface and renders plausible candidate options beneath the search bar, whereas Code2World fails to enter the intended search state, gWorld produces a poorly formatted search interface, and MobileWorldModel largely retains the original booking page. Figure 11 shows a click on the Stopwatch tab: DreamUI-World faithfully renders the characteristic circular display and start control, while most competing models generate simplified digital timing displays or retain the original timer keypad. In Figure 12, given a click on the save control after editing a calendar

event, DreamUI-World predicts a plausible post-save state with the event correctly displayed on the target date, while Code2World introduces substantial layout discrepancies, gWorld produces a nearly empty calendar, and MobileWorldModel fails to transition away from the editing interface.

E.2 TRAJECTORY VISUALIZATION.

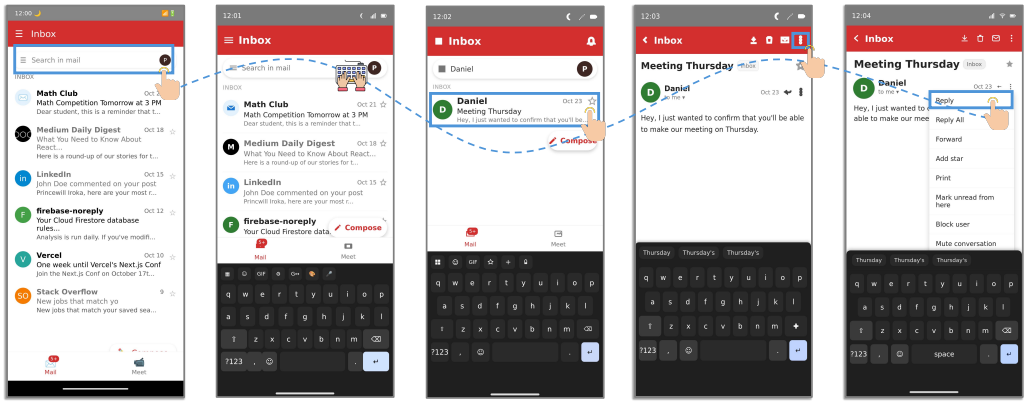

Figure 13: Selected states from an email-handling trajectory within the UI World Model, tracing the mobile agent's actions from contact search to reply initiation.



Figure 14: Hierarchical delivery-address configuration simulated by the UI World Model, including province, city, and district selection followed by detailed address entry.

We visualize representative rollout segments to demonstrate how the UI World Model serves as a simulated environment for training a mobile agent with reinforcement learning. In Figure 13, the agent searches for a target email, opens the corresponding message, and selects the reply action. Figure 14 shows hierarchical region selection and detailed address entry, demonstrating the model's ability to simulate cascading UI dependencies and form interactions. Furthermore, Figure 15 presents a complete post-creation process, including text entry, submission, and the resulting feed update. Together, these examples show that the UI World Model can generate coherent state transitions while preserving task-relevant context across multi-step interactions.

Figure 16 further highlights the stability of the simulated environment. The agent navigates to the followed-hashtag list and unfollows the target hashtags sequentially. After each action, the corresponding button changes from *Following* to *Follow*, while the remaining UI state is preserved. Rather than skipping intermediate states or directly producing the final outcome, the model generates each action-conditioned state in sequence. This consistent state evolution makes the UI World Model a reliable environment for reinforcement learning of long-horizon mobile interaction policies.

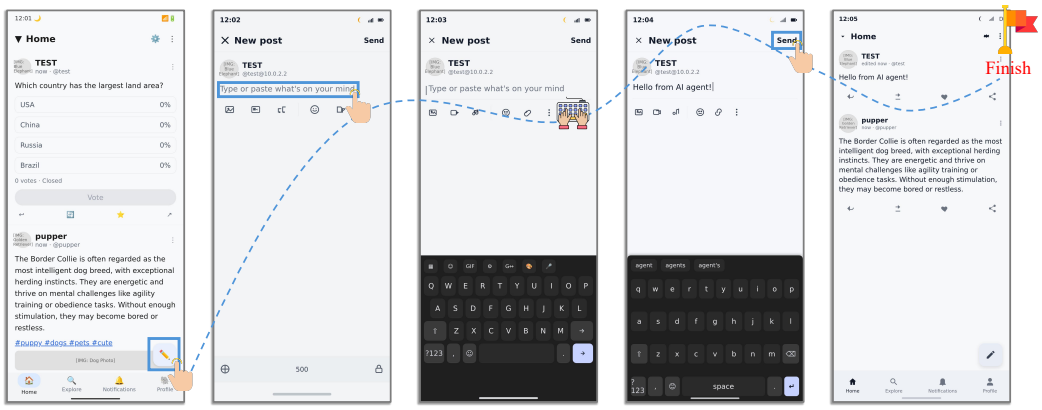

Figure 15: A segment of a mobile agent rollout trajectory in a UI World Model–simulated environment, illustrating post creation, text entry and submission.

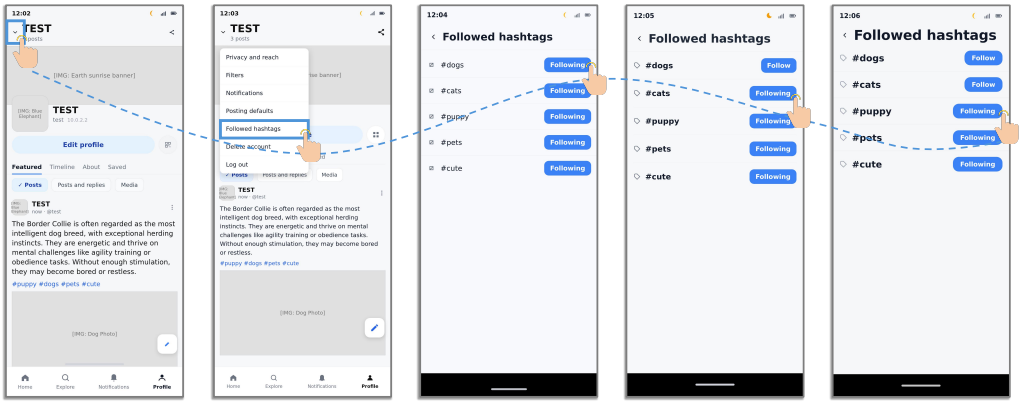

Figure 16: State-by-state evolution of a mobile agent rollout in the simulated UI World Model, showing navigation to followed hashtags and sequential hashtag unfollowing.

F Limitations and Future Work.

In this work, we demonstrate that generative UI world models can serve as scalable RL environments for mobile agent training, achieving consistent improvements without real-device interaction during agent RL. However, our work still has several limitations. First, for inference efficiency, our UI world model does not incorporate a memory mechanism over previous observations, which limits its fidelity in extremely long-horizon simulations. Second, we optimize the UI world model only at the step level, maximizing per-frame rendering quality without explicitly enforcing multi-step consistency. Extending optimization to the trajectory level could further mitigate error accumulation and improve reliability in long-horizon simulations. Third, we adopt GRPO for agent training, which assigns a single trajectory-level reward and struggles to distinguish the contribution of individual actions in long-horizon tasks. Incorporating algorithms with fine-grained credit assignment, such as PPO with learned value functions, could further improve RL effectiveness on complex GUI tasks.