Code2World: A GUI World Model via Renderable Code Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

GUI agents interact with environments by perceiving interfaces and executing actions. As a virtual sandbox, the GUI World model empowers agents with human-like foresight by enabling action-conditioned prediction. However, existing text/pixel-based approaches struggle to achieve both high visual fidelity and fine-grained structural controllability. To this end, we propose **Code2World**, a vision-language coder that simulates the next visual state via **renderable code generation**. Specifically, to address the data scarcity problem, we construct **AndroidCode** by translating GUI trajectories into high-fidelity HTML and refining synthesized code through a visual-feedback revision mechanism, yielding a corpus of **over 80K** high-quality screen-action pairs. To adapt existing VLMs into code prediction, we first perform SFT as a cold start for format layout following, then further apply **Render-Aware Reinforcement Learning** which uses rendered outcome as the reward signal by enforcing visual semantic fidelity and action consistency. Extensive experiments demonstrate that Code2World-8B achieves the top-performing next UI prediction, rivaling the competitive GPT-5 and Gemini-3-Pro-Image. Notably, *Code2World significantly enhances downstream navigation success rates in a flexible manner*, boosting Gemini-2.5-Flash by +9.5% on AndroidWorld navigation.

1 Introduction

Recent advancements in Large Multimodal Models (LMMs) have revolutionized the development of autonomous GUI agents [Hong et al., 2024, Lin et al., 2025b, Qin et al., 2025]. These agents are designed to perceive visual interfaces and execute sequences of actions to complete complex tasks across web and mobile platforms [Xie et al., 2024, Li et al., 2024]. Despite these promising capabilities, a significant gap with human proficiency persists in real-world scenarios, particularly for navigation tasks that demand precise reasoning and error correction [Chae et al., 2024].

A key factor contributing to human superiority is the ability to *mentally simulate the consequences of actions before execution* to evaluate potential outcomes and adjust their strategies accordingly. In contrast, most existing GUI agents operate actions directly without such foresight. Lacking this predictive capability, an erroneous execution at the current step often necessitates costly multi-step corrections or leads to immediate task failure [Wu et al., 2025, Lin et al., 2024]. This is particularly dangerous in high-risk scenarios, such as confirming payments or deleting critical data, where actions are irreversible. To mitigate these risks, it is essential to equip agents with a virtual sandbox to simulate the action-conditioned observation, often referred to as a *world model* which has already demonstrated remarkable success in domains like embodied AI [Lu et al., 2025b].

Recent works have attempted to explore GUI world models under two different representations. **Text-based approaches** [Chae et al., 2024, Gu et al., 2024] leverage Large Language Models (LLMs) to predict state transitions via abstract natural language descriptions. Although these methods capture

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

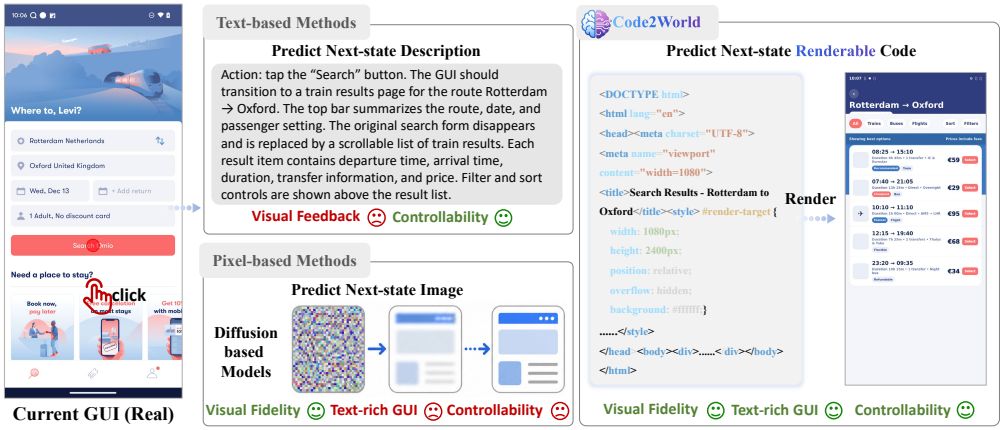


Figure 1: **Illustration of Code2World.** Given a current GUI observation and an action, Code2World predicts the next screenshot via renderable code generation.

semantic intent, they fundamentally lack visual information. Visual feedback provides a more intuitive representation of the environment and has been shown to significantly enhance the reasoning capabilities of agents [Su et al., 2025]. Conversely, **Pixel-based approaches** [Wei et al., 2023, Luo et al., 2025a] predominantly utilize diffusion models to synthesize future screenshots. However, it is non-trivial to model the precise and discrete state transitions of GUIs within a continuous pixel space. Such approaches also face challenges in maintaining fine-grained structural controllability, particularly in text-rich interface [Zhang et al., 2025]. Ultimately, neither text nor pixel representations can simultaneously achieve high-fidelity visual simulation and precise structural controllability.

Motivated by the fact that code can be used to reconstruct visual effects [Lin et al., 2025c, Chen et al., 2025b], and that GUIs are natively constructed from HTML, can we explore code as a representation to learn a GUI world model? Unlike abstract text or raw pixels, symbolic code inherently guarantees structural controllability while enabling high-fidelity visualization via deterministic rendering. Leveraging this insight, we propose a paradigm shift by treating GUI simulation as *renderable code generation*. Under this paradigm, we introduce **Code2World**, as shown in Figure 1, a vision-language coder designed to predict dynamic transitions by synthesizing structured *HTML code* and rendering it into the next GUI state. However, this paradigm faces three significant challenges: ***(i)*** *Data scarcity:* how to curate large-scale, high-quality data that grounds diverse GUI states in faithful structured code; ***(ii)*** *Screenshot-code alignment:* how to build a backbone and devise a learning strategy that effectively aligns textual code generation with rendered visual reality; and ***(iii)*** *Evaluation & Application:* how to rigorously benchmark next UI prediction and effectively assist downstream GUI agents.

To overcome these challenges, ***(i)*** we first construct **AndroidCode**, a large-scale corpus comprising over **80K** samples. Derived from the image-based GUI trajectories in AndroidControl [Li et al., 2024], we synthesize their corresponding HTML representations via GPT-5. Crucially, to ensure data quality, we implement a visual-feedback revision mechanism that refines the generated code based on visual alignment checks. ***(ii)*** Building on a Vision-Language Model (VLM), we begin with Supervised Fine-Tuning (SFT) as a cold start for format following, and then we further employ a **Render-Aware Reinforcement Learning** strategy to enhance the generalization. This strategy integrates dual outcome rewards to align the model with visual reality and action consistency. ***(iii)*** Finally, we establish a holistic evaluation protocol utilizing a VLM-as-a-Judge framework to benchmark functional logic and visual quality. We further validate Code2World's practical value by integrating it as a **plug-and-play** simulator to enhance diverse downstream GUI agents. Extensive experiments demonstrate that Code2World-8B achieves the top-performing next UI prediction, rivaling GPT-5 and Gemini-3-Pro-Image, and significantly enhances downstream navigation success rates, boosting Gemini-2.5-Flash by **+9.5%** on AndroidWorld [Rawles et al., 2025]. Our contributions are as follows:

- We propose Code2World, a novel VLM-based GUI World Model that predicts dynamic transitions via renderable code generation. By utilizing HTML as the native representation, our framework simultaneously achieves high-fidelity visual simulation and precise structural controllability.
- We present AndroidCode, a high-fidelity corpus comprising over 80K samples. We employ a synthesis pipeline with visual-feedback revision to ensure high data quality.
- We introduce a novel Render-Aware Reinforcement Learning strategy to align the textual output with visual reality and ensure the generalization.

- Extensive experiments demonstrate that Code2World achieves top performance in next GUI prediction. As a plug-and-play simulator, it significantly enhances the capabilities of downstream GUI agents, yielding consistent improvements in both offline and online navigation tasks.

2 Related Work

2.1 GUI Agents

The advent of LMMs has shifted GUI agents from metadata dependency to direct pixel-level perception, establishing robust baselines for generalist control [Hong et al., 2024, Qin et al., 2025]. To tackle the complexity of long-horizon navigation, recent research has augmented GUI agents with advanced deliberative capabilities. Innovations in advanced grounding [Nayak et al., 2025, Feizi et al., 2025], efficient context management [Cheng et al., 2025a, Liu et al., 2025a] and self-reflection mechanisms [Wu et al., 2025, Xiao et al., 2025] enable agents to maintain goal consistency and autonomously refine strategies across extended trajectories. Currently, the rapid evolution of Reinforcement Learning (RL) has empowered LMMs with advanced reasoning and planning capabilities in various domains [Dai et al., 2026]. In the context of GUI automation, RL has been widely adopted to optimize decision logic, ranging from reasoning-enhanced "R1-style" fine-tuning [Luo et al., 2025b, Lu et al., 2025c] to online policy optimization [Shi et al., 2025, Lu et al., 2025a]. However, these approaches optimize the agent's policy in the actual environment. Code2World shifts the focus from the agent to the environment, functioning as a learnable virtual sandbox. Unlike RL for the agent itself, we employ a render-aware RL algorithm to optimize our simulator.

2.2 GUI Environments and World Models

Autonomous GUI agents operate within interactive environments, executing actions to elicit feedbacks [Rawles et al., 2025, Xie et al., 2024]. World models internalize these dynamics to function as efficient virtual sandboxes for agent planning. Existing research typically adopts two output modalities. *(i) Text-based approaches* leverage Large Language Models (LLMs) to predict state transitions via symbolic abstraction, ranging from natural language descriptions of state differentials [Chae et al., 2024, Li et al., 2025, Lin et al., 2025a] and structural DOM updates [Gu et al., 2024, Wang et al., 2025] to textual sketches preserving spatial layouts [Cao et al., 2026]. Conversely, *(ii) Pixel-based approaches* utilize image generation models to synthesize future screenshots directly, often adapting diffusion architectures to generate UI layouts and frame sequences [Wei et al., 2023, Rivard et al., 2025], or decoupling text and layout generation [Luo et al., 2025a] for enhanced fidelity. However, text-based methods discard critical spatial-visual details, while pixel-based synthesis struggles with structural controllability and text-rich scenarios. To bridge this gap, we propose a *(iii) renderable code approach* that ensures both visual fidelity and structural integrity.

3 Code2World

We define the task as **Next UI Prediction**, which aims to approximate the deterministic state transition function of a digital environment. Formally, let I_t denote the visual observation (screenshot) at time step t, a_t represent the user action executed on I_t, and $\mathcal{G}$ denotes the task goal. The objective is to predict the subsequent visual state $\hat{I}_{t+1}$. We propose a *renderable code generation* paradigm by targeting the underlying structural representation of the interface rather than the raw pixel space. We define the state transition as a two-step conditional generation process:

$$\hat{C}_{t+1} = \mathcal{M}_\theta(I_t, a_t, \mathcal{G}), \qquad \hat{I}_{t+1} = \mathcal{R}(\hat{C}_{t+1}) \tag{1}$$

where $\mathcal{M}_\theta$ is a multimodal generator parameterized by θ, producing the predicted structured HTML code $\hat{C}_{t+1}$. Subsequently, the predicted visual representation $\hat{I}_{t+1}$ is obtained deterministically via a browser rendering engine $\mathcal{R}$.

3.1 Training Data Synthesis

A primary challenge in training code-based world models is the scarcity of paired GUI trajectories and their corresponding clean HTML representations. Most existing benchmarks, such as Android

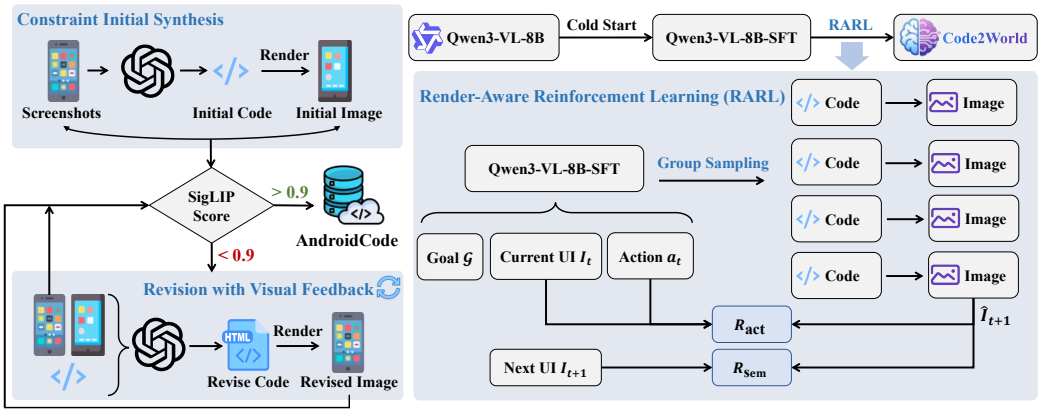


Figure 2: **Left:** Illustration of data curation of AndroidCode. **Right:** Two-stage model optimization.

Control [Li et al., 2024], solely provide raw screenshots and action coordinates. To address this, we synthesize **AndroidCode**, a large-scale, high-fidelity dataset derived from the Android Control corpus. We implement a fully automated pipeline, visualized in the left panel of Figure 2, that transforms raw GUI pixels into semantic HTML code through two hierarchical stages.

Constrained Initial Synthesis. We first utilize GPT-5 to translate each GUI screenshot into structured HTML code, imposing rigorous constraints to ensure deterministic rendering. Specifically, we enforce a standardized root container with fixed dimensions to align the DOM coordinate system precisely with the original screenshot. Furthermore, to mitigate the hallucination of external assets (more analysis in Appendix B.1), we implement a *semantic placeholder strategy*: the model is instructed to replace unreliable image URLs with descriptive text blocks (`[IMG: Red Shoe]`) and render UI icons as inline SVGs. This design guarantees that the generated HTML is a self-contained, dependency-free atomic unit. Full prompts are detailed in Appendix G.1.

Revision with Visual Feedback. Direct code synthesis from screenshots, lacking a verification loop, faces challenges in guaranteeing strict structural alignment and pixel-level precision. To ensure high-quality data curation with computational efficiency, we introduce a *selective revision mechanism with visual feedback*. For each generated sample, we render the code back into an image and compute a visual alignment score using SigLIP against the ground truth. Samples exceeding a high confidence threshold are considered qualified and retained immediately. For samples falling below this threshold, we trigger a refinement loop as shown in Figure 2. Specifically, we feed the ground truth screenshot, the currently rendered image, and the generated HTML code back into GPT-5. The model is instructed to visually compare the two images to identify discrepancies and rectify the code. This mechanism effectively forces the model to revise its own output based on visual evidence, ensuring that AndroidCode maintains high fidelity while optimizing generation efficiency.

3.2 Model Optimization

To empower the multimodal generator $\mathcal{M}_\theta$ with both syntactic precision and visual-level foresight, we employ a two-stage training strategy, schematically shown in the right panel of Figure 2. We organize the synthesized AndroidCode dataset into multimodal training triplets (x, C^*, I^*_{t+1}), where $x = (I_t, a_t, \mathcal{G})$ represents the conditioning input comprising the visual observation, user action, and task goal (detailed in Appendix G.2). The terms C^* and I^*_{t+1} denote the ground-truth HTML sequence and the corresponding target GUI screenshot, respectively. This dual-modal supervision allows the model to learn from both discrete symbolic structures and rendered visual outcomes.

Stage 1: Supervised Fine-tuning for Cold Start. The initial phase focuses on the symbolic mapping from x to C^*, aiming to instill the model with the fundamental syntax of HTML and the logic of UI layout. We adapt the Qwen3-VL-8B-Instruct [Team, 2025b] as backbone by minimizing the standard negative log-likelihood loss over the target code tokens C^*. Through this process, the model learns to generate syntactically valid code structures. However, since this stage treats the code merely as text, the supervised policy, denoted as π_{sft}, remains blind to the final rendered visual outcome.

Stage 2: Render-Aware Reinforcement Learning. To bridge the gap between textual code and visual reality, we use the final rendered outcome as the learning signal to guide optimization. A core innovation of our framework is the design of a composite reward function R_{total} that guides the model

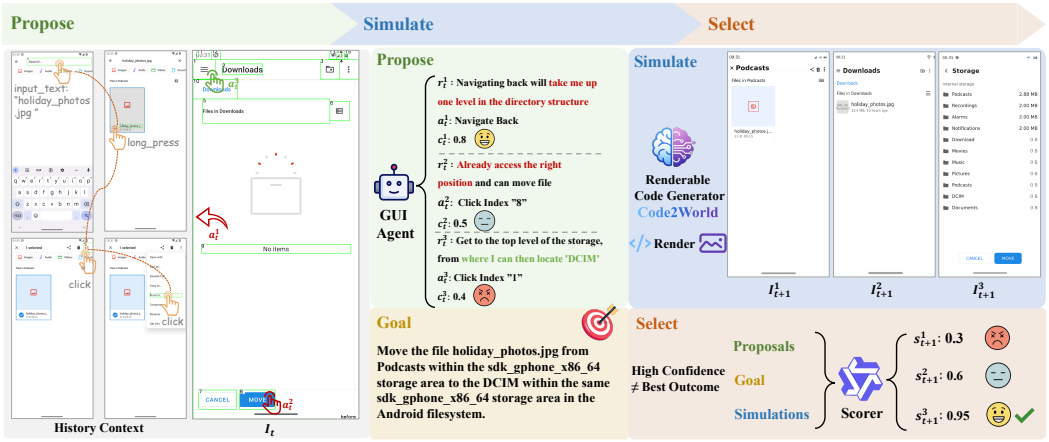


Figure 3: Illustration of the **"Propose, Simulate, Select"** pipeline for Code2World enhanced GUI agent, exemplified by an AndroidWorld task [Rawles et al., 2025].

across two distinct dimensions: visual semantic fidelity and action consistency. Let $\hat{I}_{t+1} = \mathcal{R}(y)$ be the image rendered from a sampled HTML sequence y, and I^*_{t+1} be the ground-truth next state. By leveraging the reasoning capabilities of VLM, we introduce two complementary rewards.

Visual Semantic Reward (R_{sem}). Standard similarity metrics (e.g., CLIP [Radford et al., 2021]) are often brittle—overly sensitive to pixel-level shifts while missing fine-grained details [Yang et al., 2025]. Furthermore, our code generation employs a *rendering abstraction* strategy, where images and complex icons are represented by textual placeholders (e.g., a generic box labeled "Settings") since generating raw assets is infeasible. Although the resulting layout is semantically correct and more concise, embedding-based models severely penalize this visual style discrepancy. To address this, we employ a VLM-as-a-Judge to evaluate the high-level semantic alignment (detailed in Appendix G.4). The VLM is instructed to focus on structural layout and element correspondence, tolerating stylistic abstractions, yielding a score:

$$R_{\text{sem}} = \text{VLM}_{\text{judge}}(\hat{I}_{t+1}, I^*_{t+1}) \tag{2}$$

This ensures the model is rewarded for the correct UI structure rather than superficial pixel matching.

Action Consistency Reward (R_{act}). Visual similarity alone is insufficient; the state transition must logically reflect the execution of the user action a_t. We similarly utilize a VLM-as-a-Judge mechanism to verify the dynamic logic, ensuring the transition adheres to the user's intent (detailed in Appendix G.3). We feed the triplet $(I_t, a_t, \hat{I}_{t+1})$ into the judge model to assess whether the state change in $\hat{I}_{t+1}$ is a valid consequence of executing a_t on the previous state I_t. The VLM outputs a confidence score serving as the logic reward:

$$R_{\text{act}} = \text{VLM}_{\text{judge}}(I_t, a_t, \hat{I}_{t+1}) \tag{3}$$

This effectively penalizes hallucinations where the visual update contradicts the intended action logic.

The final reward is a weighted sum $R_{\text{total}} = \lambda_1 R_{\text{sem}} + \lambda_2 R_{\text{act}}$, where λ_1 and λ_2 balance each component's contribution. To optimize this objective, we adopt *Group Relative Policy Optimization (GRPO)* [Shao et al., 2024], which estimates the baseline using the group average of rewards from G sampled outputs $\{y_1, \ldots, y_G\}$ from the old policy $\pi_{\theta_{\text{old}}}$. For each input x, we compute the advantages A_i by normalizing rewards within the group:

$$A_i = \frac{R_{\text{total}}(y_i) - \text{mean}(\{R_{\text{total}}(y_j)\}_{j=1}^G)}{\text{std}(\{R_{\text{total}}(y_j)\}_{j=1}^G) + \epsilon} \tag{4}$$

The model parameters θ are then updated by maximizing the surrogate objective with a KL-divergence penalty to maintain proximity to the reference SFT policy:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_x \left[\frac{1}{G} \sum_{i=1}^{G} \min\big(\rho_i A_i, \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon) A_i\big) - \beta D_{\text{KL}}(\pi_\theta \parallel \pi_{\text{sft}}) \right], \tag{5}$$

where $\rho_i = \frac{\pi_\theta(\mathbf{y}_i|\mathbf{x})}{\pi_{\theta_{\text{old}}}(\mathbf{y}_i|\mathbf{x})}$ is the importance sampling ratio. Through this process, Code2World iteratively refines its generation policy to maximize both visual fidelity and logical coherence.

4 Evaluation and Application of Code2World

What defines a high-quality GUI world model? We posit that a reliable virtual sandbox in GUI domain must satisfy two critical requirements: (1) **Precise Next UI Prediction**, where the model must simultaneously achieve high visual fidelity and strictly adhere to interaction logic during state transitions, and (2) **Effective GUI Agent Enhancement**, where the world model's foresight effectively enhances GUI agent planning and decision-making. Accordingly, we structure our evaluation to benchmark simulation quality in Section 4.1, followed by an assessment of Code2World's practical impact on assisting GUI agents in Section 4.2.

4.1 Evaluation for Next UI Prediction

To comprehensively assess the capability of next UI prediction, we propose a holistic evaluation protocol tailored for GUI environments. Unlike general image generation tasks, GUI dynamics requires strict adherence to interaction rules and precise structural rendering. Therefore, we design metrics across two complementary dimensions by employing a unified VLM-as-a-Judge framework.

Functional Logic. This dimension evaluates the correctness of state transitions, verifying whether the simulation adheres to the underlying interaction rules.

- **Action Adherence** S_{ad}: Determines if $\hat{I}_{t+1}$ logically ensues from a_t and I_t, preventing hallucinations where the visual update contradicts the intended interaction logic.
- **Action Identifiability** S_{id}: Tests if a_t can be correctly inferred solely from the visual difference between I_t and $\hat{I}_{t+1}$, measuring the causal clarity of the simulation.

Visual Quality. This dimension assesses the fidelity of the rendered interface. Beyond standard semantic metrics like SigLIP [Zhai et al., 2023] and DINO [Oquab et al., 2023] which capture high-level similarity, we introduce two specialized metrics to scrutinize fine-grained details:

- **Element Alignment** S_{ele}: Checks if key UI elements align precisely with the ground truth layout.
- **Layout Integrity** S_{lay}: Evaluates the preservation of layout geometry and structural containment.

The formal formulations and detailed implementations of these metrics are provided in Appendix C.

4.2 Application for GUI Agent

Code2World serves as a plug-and-play module to enhance any existing GUI agent. As illustrated in Figure 3, we seamlessly integrate it as a look-ahead simulator during inference via **"Propose, Simulate, Select"** pattern.

Given the current observation I_t, the task goal $\mathcal{G}$ and the interaction history $\mathcal{H}_t$, the GUI agent generates a set of K candidate proposals $\mathcal{P}_t$, where each proposal comprises a reasoning rationale $r_t^{(k)}$, an executable action $a_t^{(k)}$, and an initial confidence score $c_t^{(k)}$:

$$\mathcal{P}_t = \text{Agent}(I_t, \mathcal{G}, \mathcal{H}_t) = \left\{ \left(r_t^{(k)}, a_t^{(k)}, c_t^{(k)} \right) \right\}_{k=1}^{K}. \tag{6}$$

To avoid unnecessary inference overhead (more discussion in Appendix D), we do not invoke Code2World when exactly one of the K candidate answers satisfies $c_t^{(k)} \geq 0.9$. Otherwise, for each proposal, Code2World predicts the next GUI state by generating structured HTML Code and rendering it into an image:

$$C_{t+1}^{(k)} = \mathcal{M}_\theta(I_t, a_t^{(k)}, \mathcal{G}), \quad \hat{I}_{t+1}^{(k)} = \mathcal{R}\left(C_{t+1}^{(k)} \right). \tag{7}$$

Finally, the scorer selects the most promising action based on the task goal and simulated outcomes:

$$a_t^\star = \arg \max_{a_t^{(i)} \in \mathcal{A}_t} \mathcal{S}\big(I_t, \mathcal{G}, a_t^{(i)}, \hat{I}_{t+1}^{(i)}\big), \tag{8}$$

where $\mathcal{S}(\cdot)$ can be implemented as a VLM-based verifier that judges which predicted next state best advances progress toward the task goal $\mathcal{G}$, effectively filtering out hallucinations or illogical plans.

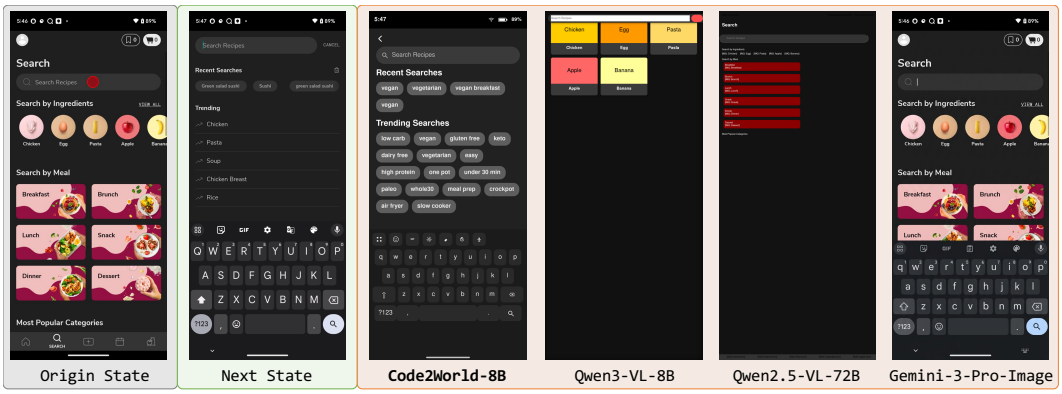


Figure 4: **Qualitative comparison** of next GUI state generation over Code2World and three baselines. The red circle in origin state indicates the user's click position targeting the search bar.

5 Experiments

We structure our experimental analysis to investigate three core research questions: **RQ1:** How effectively can Code2World predict next GUI observation in both in-domain and out-of-distribution settings? **RQ2:** Can Code2World enhance GUI agents navigation? **RQ3:** How does each component of Code2World contribute to the overall performance? Notably, for each setting, we carefully tailor the baselines and metrics to the corresponding question.

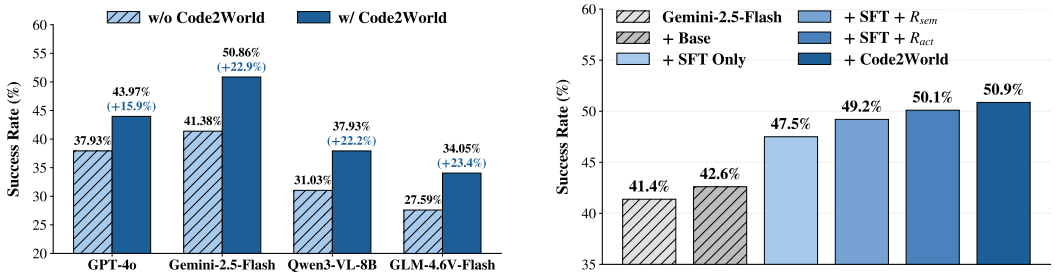
Table 1: **Quantitative Comparison** of various image and code generation models on Android Control **(ID)** and GUI Odyssey **(OOD)**. The best scores are in **bold** while the second best are in underlined.

Model	Android Control (ID)						GUI Odyssey (OOD)					
	Functional Logic		Visual Quality				Functional Logic		Visual Quality			
	S_{ad}	S_{id}	S_{ele}	S_{lay}	*SigLIP*	*DINO*	S_{ad}	S_{id}	S_{ele}	S_{lay}	*SigLIP*	*DINO*
Image Generation Models												
Gemini-3-Pro-Image	92.63	83.67	68.47	63.67	**84.89**	**64.11**	88.50	75.20	45.60	42.30	**78.74**	**48.10**
GPT-Image-1	89.59	71.40	58.78	56.22	77.58	61.00	91.63	71.40	42.04	39.80	76.46	46.64
Doubao-Seedream-4.5	85.36	86.15	59.08	55.82	81.76	52.19	89.30	66.04	42.71	40.75	72.80	37.83
Qwen-Image-Edit-Max	57.55	54.12	54.05	46.33	70.39	53.61	69.05	65.11	38.24	34.28	66.46	47.77
Janus-Pro-7B	58.10	53.68	55.22	45.90	79.85	54.30	68.40	63.30	39.10	33.95	67.05	48.20
Code Generation Models												
GPT-5	94.02	**90.15**	74.13	69.78	78.07	41.74	**93.45**	78.22	**64.41**	**60.33**	70.79	46.35
Gemini-3-Flash	92.65	84.08	**74.52**	69.74	81.17	45.72	92.61	**82.04**	59.43	55.12	71.19	43.27
Claude-Sonnet-4.5	89.60	86.12	65.80	62.25	75.97	36.72	82.81	79.17	43.75	41.56	64.72	34.94
JanusCoderV-7B	57.12	56.05	30.18	31.09	59.89	21.73	86.64	68.11	46.42	43.85	74.82	41.65
Qwen3-VL-8B	59.20	65.80	43.10	42.70	63.88	28.86	34.60	40.00	14.20	14.00	58.95	32.49
Qwen2.5-VL-72B	73.34	70.12	47.15	48.26	68.45	30.12	56.12	54.08	21.45	20.73	55.25	30.20
InternVL3-78B	72.41	67.35	45.06	46.73	62.64	25.42	60.25	58.10	25.30	24.88	60.45	31.50
GLM-4.6V-106B	91.62	74.23	61.95	58.74	67.26	27.42	55.40	52.80	30.15	28.90	62.10	33.25
Code2World-8B (Ours)	**94.28**	88.64	71.35	**70.32**	79.44	49.18	92.73	78.22	55.46	49.78	74.40	46.43

5.1 World Model Ability (RQ1)

Benchmarks and Metrics. We rigorously evaluate the generalization capability of Code2World on unseen applications. We employ two benchmarks: (1) **Android Control** serves as the **In-Domain (ID)** setting, containing applications not seen during training, evaluating generalization within the same mobile device used in training. (2) **GUI Odyssey** represents the **Out-of-Distribution (OOD)** setting, serving as a more challenging *Cross-App* setting with diverse UI styles and domains to test robustness against out-of-distribution device shifts. To strictly quantify performance, we report results using the four VLM-based metrics (S_{ad}, S_{id}, S_{ele}, S_{lay}) defined in our evaluation protocol (Section 4.1), along with standard image similarity metrics (SigLIP, DINO).

Baselines. We compare Code2World against a comprehensive suite of state-of-the-art models, categorized by their generation modality. For **Image Generation Models**, we compare against pixel-space synthesis approaches, specifically Gemini-3-Pro-Image [Google DeepMind, 2025], GPT-Image-1, Doubao-Seedream-4.5, Qwen-Image-Edit-Max, and Janus-Pro-7B [Chen et al., 2025a]. For **Code Generation Models**, we evaluate VLMs in a zero-shot setting where they predict the next GUI state by directly generating HTML. This includes proprietary models such as Claude-4.5-Sonnet, Gemini-3-Flash [Team, 2025a], and GPT-5 [Hurst et al., 2024], alongside leading open-source models



(a) Performance comparison

(b) Ablation analysis

Figure 5: (a) Performance comparison on the AndroidWorld. (b) Ablation analysis of training pipeline design for task success rate (SR) on AndroidWorld.

including JanusCoderV-7B [Sun et al., 2025], Qwen3-VL-8B [Team, 2025b], Qwen2.5-VL-72B [Bai et al., 2025], InternVL3-78B [Zhu et al., 2025], and GLM-4.6V-106B [Team, 2025c].

Quantitative Comparison. As shown in Table 1, Code2World is **lightweight yet powerful**. Despite its compact 8B size, Code2World outperforms open-source baselines scaling **over 10** $\times$ in parameters across both dynamic logic and visual quality dimensions, rivaling proprietary giants like GPT-5 and Gemini-3-Pro-Image. Our analysis identifies distinct limitations inherent to existing *open-source approaches*. Large generalist VLMs (e.g., InternVL3-78B, GLM-4.6V-106B), while capable of inferring semantic state transitions, often lack the specialized UI-to-Code alignment required to faithfully reconstruct the future layout, evidenced by their suboptimal visual quality scores. Conversely, image editing models (e.g., Qwen-Image-Edit, Janus-Pro-7B) struggle to predict precise interface dynamics, prioritizing local texture consistency over the global interaction logic necessary for navigation. Code2World effectively bridges these gaps, delivering high-fidelity simulation in both structure and logic without such shortcomings. This validates the potential of *renderable code generation* paradigm to unlock GUI world modeling capabilities in lightweight VLM architectures.

Robust Generalization. Performance on the GUI Odyssey benchmark (OOD) further validates Code2World's capability to internalize interaction dynamics rather than merely memorizing specific layouts. As expected, the shift to diverse, unseen cross-app environments causes a natural decline in visual similarity metrics across all models compared to the in-domain Android Control. Crucially, however, Code2World exhibits exceptional robustness in dynamic logic, **maintaining** S_{ad} **of 92.73** and S_{id} **of 78.22**—minimal fluctuation relative to the marked performance decay seen in open-source counterparts. This consistency suggests that Code2World has internalized the fundamental interaction dynamics of GUIs, which confirm its viability as a general-purpose world model capable of operating in novel digital environments.

Qualitative Comparison. Figure 4 provides a visual confirmation that directly corroborates the failure modes identified in our quantitative analysis. Reflecting the capacity deficit of smaller models, Qwen3-VL-8B fails to comprehend the transition logic entirely, merely reproducing the initial state with artifacts rather than predicting the future. Exemplifying the reasoning-execution gap, Qwen2.5-VL-72B correctly infers the semantic intent to switch to a "search" interface but fails to translate this into a coherent visual form, resulting in a chaotic layout that lacks clarity. Similarly, verifying the structural rigidity of pixel-based generation, Gemini-3-Pro-Image excels at texture synthesis but remains constrained to the original layout geometry, unable to hallucinate the entirely new page structure required by the navigation. In stark contrast, Code2World bridges these gaps by accurately predicting both the logical jump and fine-grained visual details, achieving near-perfect alignment with the ground truth. A compelling instance is that Code2World even **simulates temporal passage**, evidenced by the time change ($5:46 \rightarrow 5:47$) during the state transition. This intriguing detail exemplifies the model's robust world modeling capabilities, demonstrating its aptitude for capturing latent environmental changes and predicting them with both logical correctness and high visual fidelity. More Qualitative Analysis can be seen in Appendix H.1.

5.2 GUI Agent Enhancement (RQ2)

We conduct online evaluation in AndroidWorld [Rawles et al., 2025]. Unlike offline evaluation (see in Appendix E.1), which focuses on single-step action prediction, AndroidWorld requires agents to generate continuous action sequences and measures performance by the task success rate (SR). We

adopt the *default M3A framework* provided by AndroidWorld and evaluate multiple VLMs, including two closed-source models (GPT-4o, Gemini-2.5-Flash) and two open-source models (Qwen3-VL-8B, GLM-4.6V-Flash [Team, 2025c]). As shown in Figure 5a, Code2World consistently improves task success rates across all evaluated models, demonstrating its plug-and-play effectiveness. This improvement stems from Code2World's renderable code generation and render-aware reinforcement learning strategy, which enable accurate and realistic GUI prediction. By providing agents with foresight into future GUI states, Code2World allows effective exploration of multiple candidate actions while reliably selecting the most advantageous one, thereby substantially enhancing long-horizon reasoning ability in online interaction. More visualization cases showing how Code2World enhance GUI Agent decision-making can be found in the Appendix H.2. Further, we conduct more comparison with the GUI world model with other representations in Appendix E.2.

5.3 Ablation Study (RQ3)

Impact of Components on Next UI Prediction. We investigate the contribution of each training stage in Table 2. Applying SFT instills fundamental HTML syntax and layout rules, establishing a strong foundation that yields substantial improvements across both functional logic and visual quality metrics. However, refining the model solely with the visual reward (R_{sem}) reveals a limitation: while rendering quality improves, functional logic remains stagnant or slightly regresses. This suggests a degree of reward hacking, where the model prioritizes superficial pixel alignment with the ground truth rather than mastering the underlying state transition logic essential for a world model. Conversely, the action reward (R_{act}) primarily boosts dynamic logic by enforcing correct state transitions. While this ensures the generated view corresponds to the correct target state, resulting in moderate visual gains, it lacks the fine-grained feedback needed for high-fidelity rendering. Ultimately, Code2World integrates both rewards to achieve optimal performance, verifying that combining semantic alignment with interaction logic is critical for a robust world model.

Table 2: Comparison of the world model ability of Qwen3-VL-8B and other variants on Android Control.

Model	Android Control					
	Functional Logic		Visual Quality			
	S_{ad}	S_{id}	S_{ele}	S_{lay}	*SigLIP*	*DINO*
Qwen3-VL-8B	59.20	65.80	43.10	42.70	63.88	28.86
Qwen3-VL-8B+SFT	78.45	79.12	58.65	56.30	71.50	39.40
Qwen3-VL-8B+SFT+R_{sem}	78.90	78.85	65.40	61.20	76.10	44.50
Qwen3-VL-8B+SFT+R_{act}	88.20	84.53	60.10	57.80	74.80	40.60
Code2World	**94.28**	**88.64**	**71.35**	**70.32**	**79.44**	**49.18**

Impact of Components on GUI Agent Enhancement. We further analyze how varying qualities of the world model impact the downstream decision-making of the Gemini-2.5-Flash agent on the AndroidWorld benchmark. As shown in Figure 5b, the standalone Gemini-2.5-Flash agent achieves a baseline Success Rate (SR) of 41.4%. Merely equipping it with a naive simulator (+ Qwen3-VL-8B) yields a negligible gain (+1.2%), indicating that low-fidelity predictions fail to provide reliable foresight. In contrast, integrating our SFT model triggers a sharp performance jump to 47.5%, demonstrating that structured, parsable HTML predictions effectively ground the agent's planning. The subsequent RL stages progressively refine this capability: R_{sem} enhances the visual clarity of the simulated future, pushing the SR to 49.2%, while R_{act} ensures the simulator correctly reflects interaction dynamics, further raising the SR to 50.1%. Finally, the full Code2World model, combining all optimizations, achieves a peak SR of **50.9%** (**+9.5%** improvement over the baseline). This strong correlation between world model fidelity and agent success rate underscores that accurate, renderable visual foresight is the key to unlocking robust long-horizon reasoning.

6 Conclusion

In this work, we introduced Code2World, a pioneering code-native GUI world model that fundamentally shifts next UI prediction from raw pixel estimation to *renderable HTML code generation*, uniquely combining high-fidelity visualization with fine-grained structural controllability. We constructed AndroidCode (over 80K screen–action pairs) using a visual-feedback revision loop, and proposed Render-Aware Reinforcement Learning to align code prediction with rendered visual fidelity and action consistency. Functioning as a learnable virtual sandbox, Code2World empowers autonomous GUI agents to navigate complex, dynamic interfaces with human-like foresight. Empirically, Code2World-8B achieves state-of-the-art performance in next UI prediction and, acting as a plug-and-play simulator, significantly enhances downstream agents, boosting Gemini-2.5-Flash by +9.5% on AndroidWorld navigation.

References

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 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.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *arXiv preprint arXiv:2410.13232*, 2024.

Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-pro: Unified multimodal understanding and generation with data and model scaling. *arXiv preprint arXiv:2501.17811*, 2025a.

Yanzhe Chen, Kevin Qinghong Lin, and Mike Zheng Shou. Code2video: A code-centric paradigm for educational video generation. *arXiv preprint arXiv:2510.01174*, 2025b.

Weihua Cheng, Ersheng Ni, Wenlong Wang, Yifei Sun, Junming Liu, Wangyu Shen, Yirong Chen, Botian Shi, and Ding Wang. Mga: Memory-driven gui agent for observation-centric interaction. In *Proceedings of WSDM*, 2025a.

Zihui Cheng, Qiguang Chen, Xiao Xu, Jiaqi Wang, Weiyun Wang, Hao Fei, Yidong Wang, Alex Jinpeng Wang, Zhi Chen, Wanxiang Che, et al. Visual thoughts: A unified perspective of understanding multimodal chain-of-thought. *arXiv preprint arXiv:2505.15510*, 2025b.

Yanqi Dai, Yuxiang Ji, Xiao Zhang, Yong Wang, Xiangxiang Chu, and Zhiwu Lu. Harder is better: Boosting mathematical reasoning via difficulty-aware grpo and multi-aspect question reformulation. In *Proceedings of ICLR*, 2026.

Aarash Feizi, Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Kaixin Li, Rabiul Awal, Xing Han Lù, Johan Obando-Ceron, Juan A Rodriguez, Nicolas Chapados, et al. Grounding computer use agents on human demonstrations. *arXiv preprint arXiv:2511.07332*, 2025.

Google DeepMind. Gemini 3 Pro: The next generation of multimodal ai. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/3-pro, 2025. Accessed: 2026-01-27.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, et al. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.

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 CVPR*, 2024.

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

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 E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on ui control agents. In *Proceedings of NIPS*, 2024.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Qinchen Wu, Mingyi Yan, Zhengyuan Yang, Lijuan Wang, and Mike Zheng Shou. Videogui: a benchmark for gui automation from instructional videos. In *Proceedings of NIPS*, 2024.

Kevin Qinghong Lin, Siyuan Hu, Linjie Li, Zhengyuan Yang, Lijuan Wang, Philip Torr, and Mike Zheng Shou. Computer-use agents as judges for generative user interface. *arXiv preprint arXiv:2511.15567*, 2025a.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *Proceedings of CVPR*, 2025b.

Kevin Qinghong Lin, Yuhao Zheng, Hangyu Ran, Dantong Zhu, Dongxing Mao, Linjie Li, Philip Torr, and Alex Jinpeng Wang. Vcode: a multimodal coding benchmark with svg as symbolic visual representation. *arXiv preprint arXiv:2511.02778*, 2025c.

Tao Liu, Chongyu Wang, Rongjie Li, Yingchen Yu, Xuming He, and Bai Song. Gui-rise: Structured reasoning and history summarization for gui navigation. In *Proceedings of NIPS*, 2025a.

Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners. *arXiv preprint arXiv:2504.14239*, 2025b.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy optimization for gui agents with experience replay. In *Proceedings of NIPS*, 2025a.

Guanxing Lu, Baoxiong Jia, Puhao Li, Yixin Chen, Ziwei Wang, Yansong Tang, and Siyuan Huang. Gwm: Towards scalable gaussian world models for robotic manipulation. In *Proceedings of ICCV*, 2025b.

Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing efficient action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 2025c.

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.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, Christopher Pal, et al. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. In *Proceedings of ICML*, 2025.

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.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *Proceedings of ICML*, 2021.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. In *Proceedings of ICLR*, 2025.

Luke Rivard, Sun Sun, Hongyu Guo, Wenhu Chen, and Yuntian Deng. Neuralos: Towards simulating operating systems via neural generative models. *arXiv preprint arXiv:2507.08800*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yucheng Shi, Wenhao Yu, Zaitang Li, Yonglin Wang, Hongming Zhang, Ninghao Liu, Haitao Mi, and Dong Yu. Mobilegui-rl: Advancing mobile gui agent through reinforcement learning in online environment. *arXiv preprint arXiv:2507.05720*, 2025.

Zhaochen Su, Peng Xia, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li, Kaide Zeng, Zhengyuan Yang, et al. Thinking with images for multimodal reasoning: Foundations, methods, and future frontiers. *arXiv preprint arXiv:2506.23918*, 2025.

Qiushi Sun, Jingyang Gong, Yang Liu, Qiaosheng Chen, Lei Li, Kai Chen, Qipeng Guo, Ben Kao, and Fei Yuan. Januscoder: Towards a foundational visual-programmatic interface for code intelligence. *arXiv preprint arXiv:2510.23538*, 2025.

Google Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025a.

Qwen Team. Qwen3-vl technical report, 2025b. URL https://arxiv.org/abs/2511.21631.

V Team. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025c. URL https://arxiv.org/abs/2507.01006.

Yiming Wang, Da Yin, Yuedong Cui, Ruichen Zheng, Zhiqian Li, Zongyu Lin, Di Wu, Xueqing Wu, Chenchen Ye, Yu Zhou, et al. Llms as scalable, general-purpose simulators for evolving digital agent training. *arXiv preprint arXiv:2510.14969*, 2025.

Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Binbin Xu, Pierre Louis Bernard, and Gérard Dray. Boosting gui prototyping with diffusion models. In *2023 IEEE 31st International Requirements Engineering Conference (RE)*. IEEE, 2023.

Penghao Wu, Shengnan Ma, Bo Wang, Jiaheng Yu, Lewei Lu, and Ziwei Liu. Gui-reflection: Empowering multimodal gui models with self-reflection behavior. In *Proceedings of NIPS*, 2025.

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, et al. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents. In *Proceedings of NIPS*, 2025.

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

Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiele Cheng, Xiaotao Gu, and Jie Tang. UI2Code$^{\text{n}}$: A visual language model for test-time scalable interactive ui-to-code generation. *arXiv preprint arXiv:2511.08195*, 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: Foundamental 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 ICCV*, 2023.

Jianyi Zhang, Yufan Zhou, Jiuxiang Gu, Curtis Wigington, Tong Yu, Yiran Chen, Tong Sun, and Ruiyi Zhang. Artist: Improving the generation of text-rich images with disentangled diffusion models and large language models. In *Proceedings of WACV*, 2025.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of ACL*, 2024.

Yaowei Zheng, Junting Lu, Shenzhi Wang, Zhangchi Feng, Dongdong Kuang, and Yuwen Xiong. Easyr1: An efficient, scalable, multi-modality rl training framework, 2025.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

A Implementation Details

A.1 Experimental Setup

We conduct all experiments on a computing cluster equipped with 8 NVIDIA H20 GPUs (96GB of memory).

We use `Qwen3-VL-8B-Instruct` as the backbone of Code2World and conduct SFT with LLaMA-Factory [Zheng et al., 2024] framework and RL based on the EasyR1 [Zheng et al., 2025] framework. we also use `Qwen3-VL-8B-Instruct` as action scorer in "Propose, Simulate, Select" pipeline.

To ensure rigorous reproducibility, we standardize configurations across all metrics and baselines. For VLM-as-a-Judge in RL reward calculation, we employ `Qwen3-VL-8B-Instruct`. For final evaluation of VLM-as-a-Judge metrics on the test set, we employ `gpt-5-mini` to obtain a more robust and calibrated assessment. Both models are configured with a temperature of 0.1 to ensure deterministic outputs and a maximum generation limit of 1,024 tokens. For visual feature metrics, we compute SigLIP scores using the `google/siglip-so400m-patch14-384` checkpoint and DINO scores using the `facebook/dinov2-giant` model from Huggingface.

For baseline comparisons, the API for proprietary model is accessed via OpenRouter. We fix specific API versions to ensure consistency: `claude-4.5-sonnet-20251120` for Claude and `gpt-4o-mini-2024-07-18` for GPT-4o-mini. Across all generation models (proprietary and open-source), we unify the maximum generation length to 8,192 tokens to accommodate complex HTML structures, with a sampling temperature set to 0.7.

A.2 Training Hyperparams

Data Allocation. We partition the AndroidCode dataset into two disjoint subsets to support the two-stage training strategy. Specifically, we allocate 70% of the samples for Stage 1 (SFT) to establish the foundational policy, while the remaining 30% are employed in Stage 2 (RL) to further align the model with visual and logical rewards.

Stage 1: Code-level Supervised Fine-tuning (SFT). We initialize the backbone with `Qwen3-VL-8B-Instruct`. We perform full parameter fine-tuning on the language model while freezing both the vision encoder and the multimodal projector to preserve pre-trained visual perception capabilities. To optimize memory efficiency and training throughput, we utilize DeepSpeed ZeRO-2 and Flash Attention. The model is trained for 2 epochs with a global batch size of 64 (configured as a per-device batch size of 2 with 4 gradient accumulation steps). We employ a cosine learning rate schedule with a peak learning rate of 2×10^{-5} and a warmup ratio of 0.1. To accommodate the high resolution of smartphone screenshots (1080×2400) and verbose HTML code, we set the cutoff length to 24,576 tokens.

Stage 2: Render-Aware Reinforcement Learning (RL). We align the SFT model using Group Relative Policy Optimization (GRPO) with Flash Attention enabled. For each prompt, the policy generates a group of $G = 4$ candidate outputs with a sampling temperature of 1.0 to encourage exploration. We set the rollout batch size to 16. The max prompt length and max response length are set to 24,576 and 8,192 tokens to support complex UI structures. We set the learning rate to 1×10^{-6} and apply a KL-divergence penalty with a coefficient of $\beta = 0.01$. The visual semantic reward (R_{sem}) and action consistency rewards (R_{act}) are weighted equally during optimization.

B Training Data Construction

B.1 Analysis of HTML Representation and Semantic Placeholders

A key design in CODE2WORLD is to represent GUI states in HTML rather than in pixel space or natural language. This choice is not merely a matter of implementation convenience; it directly matches the nature of GUI pages, which are fundamentally structured, renderable, and governed by explicit layout constraints. In contrast to pixel-level generation, HTML provides a native representation of page hierarchy, widget types, geometry, and spatial relations, thereby offering fine-grained structural controllability while still allowing the predicted state to be deterministically rendered for downstream

visual reasoning. This combination of renderability and structural precision is particularly suitable for GUI world modeling.

Another important advantage of our formulation is the use of semantic placeholders. Exact reproduction of externally hosted assets or server-returned content is often infeasible without direct access to the backend database. Therefore, we explicitly model these missing elements with semantic placeholders such as `[IMG: Red running shoes]` or `[TXT: discount price]`. These placeholders are not empty boxes; they preserve the semantic information that matters for decision making while avoiding unstable, brittle, or excessively long asset synthesis. As a result, the world model is no longer forced to solve an impossible pixel-guessing problem, but instead learns to anticipate the semantically relevant layout and content type of the next state.

This abstraction is also aligned with how humans reason about future GUI states. When anticipating the result of an interaction, humans typically do not predict exact future pixels, precise image assets, or every server-generated string. Instead, they reason about the likely page structure, the function of the next screen, and the semantic category of the newly appeared elements. Therefore, semantic placeholders provide a human-like level of foresight: they preserve the information needed for planning while discarding details that are either unknowable or irrelevant to action selection.

Importantly, our design also avoids the hallucination mode that can arise when a model is required to predict exact external assets or exact dynamic text. Requiring pixel-perfect image recovery or exact recovery of server-returned text is outside the scope of any practical GUI world model without backend access. For this reason, CODE2WORLD does not supervise the model to memorize arbitrary URLs, exact database values, or other brittle surface-form details. Instead, the learning target emphasizes action-conditioned semantic layout, page function, and structural consequence. This turns an inherently impossible reconstruction task into a meaningful semantic anticipation task.

Empirically, this abstraction is not merely principled but effective. We find that the combination of HTML representation and semantic placeholders provides sufficient visual-semantic information to support robust long-horizon planning. Moreover, comparisons against text-based/pixel-based GUI world models (E.2) and matched pipeline variants demonstrate (E.3) that code-based representation consistently yields stronger downstream gains. These results suggest that HTML is a particularly suitable intermediate representation for GUI world modeling: it is expressive enough to preserve the semantics required for reasoning, yet structured enough to support deterministic rendering and controllable generation.

B.2 Synthesis and Revision Algorithm

The detailed construction pipeline for **AndroidCode** is formally presented in Algorithm 1. While the renderable code generation pipeline is outlined in the main text, we specify the operational thresholds and the exact revision logic here. As depicted in Algorithm 1, We utilize a *powerful multimodal coder (GPT-5)* to translate pixel-based screenshots into structured HTML and implement a revision mechanism with visual feedback to ensure high data fidelity. We set the visual similarity threshold $\tau = 0.9$, calculated using the SigLIP cosine similarity between the ground truth screenshot I and the rendered generation $\hat{I}$. Generated samples scoring below this value trigger the revision loop. To ensure training efficiency and prevent infinite loops, we limit the maximum number of revision iterations to $N_{max} = 1$. Crucially, we feed the triplet of *(Ground Truth I, Rendered Image $\hat{I}$, Current Code C)* directly into the multimodal coder. This allows the coder to autonomously perform a visual comparison, identify discrepancies, and rectify the structural code. Samples that fail to meet the quality threshold after N_{max} attempts are discarded to ensure the purity of the final training corpus.

B.3 Multimodal Instruction Formatting

Based on the AndroidCode, constructing a high-fidelity instruction-tuning dataset requires bridging the semantic gap between low-level interaction logs and the high-level reasoning capabilities of VLMs. Raw interaction traces, consisting solely of raw pixels and sparse coordinate metadata, often lack the semantic density required for VLMs to effectively ground user intent. To bridge the gap between low-level execution logs and high-level visual reasoning, we design a meticulous data pre-processing pipeline that augments both visual and textual modalities. This ensures the instruction-tuning data is not only syntactically structured but also semantically explicit.

Algorithm 1 Automated Data Synthesis with Visual-Feedback Revision

Require: Raw GUI dataset $\mathcal{D}_{raw} = \{I_1, \dots\}$, Multimodal Coder $\mathcal{M}_{coder}$, Browser Renderer $\mathcal{R}$, Alignment Metric SigLIP$(\cdot)$, Threshold τ, Max iterations N_{max}.
Ensure: High-fidelity corpus ANDROIDCODE $\mathcal{D}_{syn} = \{(I_k, C_k)\}$.

$\mathcal{D}_{syn} \leftarrow \emptyset$
for all $(I) \in \mathcal{D}_{raw}$ **do**
  $n \leftarrow 0, s \leftarrow 0$
  // Stage 1: Constrained Initial Synthesis
  $C \leftarrow \mathcal{M}_{coder}(I, \text{Prompt}_{\text{init}})$ {Generate initial HTML with semantic placeholders}
  $\hat{I} \leftarrow \mathcal{R}(C)$ {Render code into visual state}
  $s \leftarrow \text{SigLIP}(\hat{I}, I)$ {Compute initial visual alignment score}
  // Stage 2: Revision with Visual Feedback
  while $s < \tau$ **and** $n < N_{max}$ **do**
    $n \leftarrow n + 1$
    {Model visually compares GT and Rendered image to fix code}
    $C \leftarrow \mathcal{M}_{coder}(I, \hat{I}, C, \text{Prompt}_{\text{revision}})$
    $\hat{I} \leftarrow \mathcal{R}(C)$
    $s \leftarrow \text{SigLIP}(\hat{I}, I)$
  end while
  // Filtering & Collection
  if $s \geq \tau$ **then**
    $\mathcal{D}_{syn} \leftarrow \mathcal{D}_{syn} \cup \{(I, C)\}$ {Retain only high-quality pairs}
  end if
end for
return $\mathcal{D}_{syn}$

Visual Prompting. Raw Coordinate-based interaction records are inherently abstract to vision encoders. To explicitly ground the model's spatial attention on the target elements, we adopt a visual prompting strategy, which has been proven effective in enhancing the referring capabilities of VLMs [Cheng et al., 2025b]. Instead of relying on the model to implicitly infer locations from numerical tokens, we render visual markers directly onto the input image I_t. Specifically, for *point-based interactions* (e.g., Click, Long Press), we render a semi-transparent red circle (radius=20px, $\alpha = 0.6$) centered at (x, y), creating a visual anchor that eliminates ambiguity regarding the precise touch position. For *gesture-based interactions* (e.g., Scroll, Swipe), we overlay a directional arrow indicating the finger's movement trajectory (e.g., an upward arrow for a "scroll down" command). This visualizes the dynamic flow of the operation, allowing the model to correlate the static frame with the intended motion.

Instruction Expansion. Raw action primitives (e.g., JSON objects or bare coordinates like `{"action":"click", "loc":[200,300]}`) are disjoint from the natural language pre-training distribution of VLMs, often leading to suboptimal intent understanding. To address this, we implement a deterministic expansion engine that transforms rigid action metadata into rich, descriptive natural language narratives. For instance, abstract coordinates are converted into explicit spatial descriptions such as: *"User performed a CLICK at coordinates (x, y). Expect the button at this location to trigger,"* effectively prompting the model to attend to the causal relationship between the location and the UI element. Similarly, complex dynamics are decomposed into clear semantic instructions: scrolling is described by its consequential effect (e.g., *"The content should move, revealing new items..."*), while text inputs are formatted to emphasize content injection (e.g., *"The focused input field MUST now contain this text"*).

Finally, these augmented visual hints and expanded textual descriptions are integrated into a carefully designed, standardized prompt template (detailed in Appendix G.2), ensuring the model strictly follows the instruction context to simulate interface dynamics accurately.

C Evaluation Metrics of Next UI Prediction

To rigorously assess the capability of GUI World Models in simulating GUI dynamics, we identify that general image similarity scores (e.g., SSIM, LPIPS) are insufficient. They predominantly focus on pixel-level texture, failing to capture the unique requirements of GUI environments: *strict adherence to interaction rules* (e.g., a button click *must* trigger a deterministic state change) and *precise structural rendering* (e.g., elements must remain aligned in the DOM tree).

To bridge this gap, we propose a novel and holistic evaluation protocol tailored for GUI World Models. This protocol introduces four specialized metrics across two complementary dimensions: **Functional Logic** and **Visual Quality**. These metrics are designed to provide a fairer and more granular comparison. We employ a unified *VLM-as-a-Judge* framework to approximate human judgment. The detailed prompts for these VLM-based metrics are provided in Appendix G.4.

C.1 Functional Logic

Functional Logic evaluates the functional correctness of state transitions, verifying whether the world model acts as a reliable simulator.

Action Adherence (S_{ad})
This metric assesses whether the predicted next state $\hat{I}_{t+1}$ is a logically valid consequence of executing action a_t on state I_t. Unlike simple visual coherence, S_{ad} penalizes "hallucinations" where the visual update contradicts the intended interaction (e.g., clicking "Back" but staying on the same page).

Formally, let $\mathcal{J}_{\text{act}}$ be the VLM judge, the score for a test dataset $\mathcal{D}$ is defined as:

$$S_{ad} = \frac{1}{|\mathcal{D}_{test}|} \sum_{(I_t, a_t) \in \mathcal{D}_{test}} \mathcal{J}_{\text{act}}(I_t, a_t, \hat{I}_{t+1}) \tag{9}$$

Action Identifiability (S_{id})
This metric evaluates the causal clarity of the generation. A high-fidelity simulation should allow an observer to infer the cause of a state change solely from the visual outcome. We instruct the VLM to act as an inverse dynamics model $\mathcal{J}_{inv}$, predicting the action type $\hat{a}_t$ based on the visual difference between I_t and $\hat{I}_{t+1}$. Crucially, a high S_{id} ensures that the "Selector" in our agent pipeline can correctly verify whether a simulated outcome matches the planned action. The metric is calculated as the classification accuracy:

$$S_{id} = \frac{1}{|\mathcal{D}_{test}|} \sum_{(I_t, a_t) \in \mathcal{D}_{test}} \mathbb{1}\left[\mathcal{J}_{inv}(I_t, \hat{I}_{t+1}) = \text{type}(a_t)\right] \tag{10}$$

C.2 Visual Quality

Models following the *renderable code generation paradigm* typically employ a *semantic placeholder strategy* to guarantee structural correctness while avoiding the hallucination of external assets. Standard embedding metrics (e.g., SigLIP, DINO), which primarily capture *high-level* semantic similarity, are ill-suited for this nuance; they lack the granularity to explicitly measure *fine-grained* element alignment and structural layout, often penalizing valid stylistic abstractions. To address this, we propose two specialized metrics to disentangle structural fidelity from textural style.

Element Alignment (S_{ele}). This metric verifies the fine-grained positioning of UI components. It measures whether key interactive elements (buttons, inputs) present in the ground truth I^*_{t+1} are accurately reflected in the generation $\hat{I}_{t+1}$ at correct relative coordinates, explicitly tolerating semantic placeholders provided they occupy the correct screen area.

Layout Integrity (S_{lay}). This metric evaluates global layout integrity, penalizing issues common in weak code generation such as CSS collapse, overlapping containers, or misalignment.

Formally, the VLM judge $\mathcal{J}_{vis}$ compares the generated output against the ground truth under specific criteria and provides a composite score:

$$S_{ele/stc} = \frac{1}{|\mathcal{D}_{test}|} \sum_{I^*_{t+1} \in \mathcal{D}_{test}} \mathcal{J}_{vis}(I^*_{t+1}, \hat{I}_{t+1}) \tag{11}$$

D More Deployment Details and Inference Overhead Analysis

To evaluate the additional inference overhead introduced by Code2World, we measured the generation time of the HTML code produced by Code2World over 2,000 inference runs. The statistics are summarized in Table 3.

In practical deployment, to avoid unnecessary overhead, we decide whether to invoke Code2World based on the confidence score output by the agent. Specifically, when among the K predicted actions, one and only one action has a confidence score of ≥ 0.9, we consider the model to be highly certain about its decision, and thus no world model assistance is required. This is both reasonable and intuitive: in the long-horizon reasoning of a GUI agent, most steps are straightforward and unambiguous; only a small number of challenging and ambiguous steps lead to hesitation and errors. We evaluated the Code2World invocation rate and the average task completion time for each model on the AndroidWorld benchmark. The results are detailed below:

Table 3: Statistics of HTML code generation over 2,000 inference runs of Code2World.

	Std	Max	Min	Mean
Token Count	402.4	4,126	2,487	3,184.5
Time (s)	6.54	97.63	43.18	65.24

Table 4: Invocation Rate and Inference Overhead Across Different Base Models on AndroidWorld

Base Model	Invocation Rate (%)	w/o Code2World (s)	w/ Code2World (s)
Qwen3-VL	24.54	224.44	476.18
GLM-4.6V-Flash	21.51	214.51	464.27
Gemini-2.5-Flash	29.03	283.87	515.63
GPT-4o	19.73	347.32	579.48

While Code2World introduces extra runtime, this cost translates into clear downstream gains. Overall, we believe the efficiency–performance trade-off is meaningful in practice.

E More Experiments

E.1 GUI Agent Enhancement on Offline Navigation

We evaluate the effectiveness of Code2World in enhancing agents' capabilities on AndroidControl-High [Li et al., 2024]. This benchmark is highly challenging, as it provides only the user task, requiring the agent to autonomously plan step-wise action, thereby evaluating its single-step decision-making capability. We apply Code2World to enhance both a specialized GUI agent (Mobile-Agent-v3 [Ye et al., 2025]) and a general MLLM (Qwen2.5-VL-7B), and further compare against GPT-4o [Hurst et al., 2024], Gemini-2.5-Flash [Team, 2025a], InfiGUI-R1-3B [Liu et al., 2025b], and UI-TARS-1.5-7B [Qin et al., 2025]. We report *Action Type* accuracy, *Grounding Accuracy*, and the overall *Success Rate (SR)*.

Table 5: Performance on AndroidControl-High.

Model	Type	Grounding	SR
GPT-4o	62.14	31.82	21.2
Gemini-2.5-Flash	67.43	33.29	27.9
GUI-R1-7B	78.45	75.64	67.15
InfiGUI-R1-3B	83.16	74.51	70.98
UI-TARS-1.5-7B	81.36	78.42	69.94
Mobile-Agent-v3-7B	82.05	75.16	67.20
+Code2World	84.13	78.78	68.41
Δ *Improvement*	***+2.09***	***+3.62***	***+1.21***
Qwen2.5-VL-7B	76.66	69.64	65.16
+Code2World	78.74	74.87	66.47
Δ *Improvement*	***+2.08***	***+5.23***	***+1.31***

As shown in Table 5, incorporating Code2World yields a substantial improvement for the Qwen2.5-VL-7B, achieving an improvement of 5.23 in Grounding accuracy, making it comparable to domain-specific GUI agents and even surpassing InfiGUI-R1. Moreover, although Mobile-Agent-v3-7B is trained on task-specific data and already attains strong performance, it still benefits from Code2World, achieving the best accuracy on both Type and Grounding, which validates the plug-and-play versatility of our approach. In summary, by functioning as a *model-agnostic simulator* to enable future GUI

prediction, Code2World empowers the agent to better understand the consequences of each action, thereby selecting actions that more closely align with user intent and significantly improving single-step decision-making performance.

E.2 Comparison with GUI World Models with Other Representations

To validate the claim that renderable code is a superior intermediate representation for GUI world modeling, we compare Code2World against representative text-based and pixel-based GUI world models in terms of downstream agent gains on AndroidWorld (online navigation, SR%). All comparisons follow the same evaluation protocol: we use the T3A agent framework provided by AndroidWorld [Rawles et al., 2025], with Gemini-2.0-Flash as the base agent and integrate each world model variant into the *"Propose, Simulate, Select"* pipeline. The only variable across conditions is the world model representation.

As shown in Table 6, we include comparisons with text-based world modeling methods (e.g., WMA [Chae et al., 2024] and WebDreamer [Gu et al., 2024]) and pixel-based approaches (e.g., ViMo [Luo et al., 2025a], Qwen-Image-Edit-Max, and Gemini-3-Pro-Image [Google DeepMind, 2025]) in terms of downstream agent gains on AndroidWorld. Under these comparisons, Code2World consistently achieves stronger downstream improvement across all representation categories.

Table 6: Comparison of different GUI world model representations on AndroidWorld (SR%). All methods are evaluated under the same T3A agent framework and pipeline. Baseline denotes the T3A agent framework [Rawles et al., 2025] using Gemini-2.0-Flash as the backbone *without any world model*. Note that the results of WMA and ViMo are taken from the reported numbers in the ViMo paper [Luo et al., 2025a].

Setting	Method	SR (%)
Baseline	T3A (without world model)	33.19
(i) Text-based Methods	+ GPT-5-text	36.21
	+ WMA	32.32
	+ WebDreamer	34.48
(ii) Pixel-based Methods	+ViMO	40.95
	+ Qwen-Image-Edit-Max	34.91
	+ Gemini-3-Pro-Image	41.37
(iii) Code-based Methods	+ GPT-5-code	41.81
	+ Code2World (ours)	**42.24**

More importantly, to isolate **the effect of the representation** itself, we further conduct a matched comparison under the same pipeline and backbone: we use GPT-5 as the world model and keep the rest of the pipeline unchanged, while varying only the output representation between **renderable code** and **text**. We observe that code consistently outperforms text under this matched setting. This is the most direct evidence that our gains do not come merely from "using a stronger model" or "adding structure in general," but from the fact that *code is an executable structural representation that better supports downstream GUI reasoning*.

E.3 Comparison with Other Test-Time Scaling Techniques

To further evaluate the efficiency-performance trade-off of Code2World, we compare it with two test-time scaling baselines on AndroidWorld under the same generation configuration. Specifically, we consider two variants as follows.

- **Baseline**: the original Qwen3-VL GUI agent, with no world model and no scorer beyond its default inference.
- **Variant 1**: replacing the base agent with Qwen3-VL-Thinking, i.e., improving the agent itself through test-time reasoning only, without simulation or reranking.
- **Variant 2**: keeping the same agent (Qwen3-VL) but adding a stronger scorer (Qwen3-VL-Thinking) for reranking, still without any world-model simulation.
- **Ours**: keeping the same agent (Qwen3-VL) and the same-strength scorer (Qwen3-VL), with future-state simulation provided by Code2World.

Table 7: Comparison between Code2World and other compute-intensive test-time strategies on AndroidWorld.

Setting	Agent	World Model	Scorer	SR (%)	Time (s)
Baseline	Qwen3-VL	N/A	N/A	31.03	224.44
Variant1	Qwen3-VL-Thinking	Text-based CoT)	N/A	32.76	500.00
Variant2	Qwen3-VL	N/A	Qwen3-VL-Thinking	33.62	289.27
Ours	Qwen3-VL	Code2World	Qwen3-VL	37.93	476.18

The resulting comparison is summarized in Table 7. The results show that other test-time scaling strategies do increase computation, but their performance gains are relatively limited. In particular, simply using a reasoning model (Qwen3-VL-Thinking) consumes even more time, yet only brings a small gain over the base Qwen3-VL agent. Similarly, using a thinking judge for reranking without a world model also leads to only limited improvement. In contrast, CODE2WORLD yields a substantially larger success-rate improvement while remaining competitive in runtime.

We believe the key reason is that these alternatives are still fundamentally pure reasoning-based test-time scaling: they increase computation, but they do not provide explicit future-state grounding for the candidate actions. In other words, they rerank actions only based on textual reasoning or plausibility signals, without simulating the actual consequences of executing them. As a result, their gains are bounded by the base model's internal ability to anticipate outcomes.

By contrast, the additional computation introduced by Code2World is not spent on generating longer reasoning traces, but on producing more decision-informative future UI predictions. This is why Code2World achieves a stronger improvement than pure thinking/reranking strategies. Moreover, Code2World is not invoked at every step: when there is exactly one candidate action with confidence ≥ 0.9, the agent executes it directly and skips world-model/scorer calls. This selective invocation further improves the practical efficiency–performance trade-off.

Overall, these experiments suggest that CODE2WORLD offers a better efficiency–performance trade-off than other common compute-intensive test-time strategies, because it adds not just more computation, but more informative computation with explicit future-state information.

F Human Evaluation

We conduct two complementary human evaluations to validate (1) the quality of the AndroidCode dataset and (2) the visual semantic fidelity of model outputs. Both studies involve 10 annotators.

Study 1: AndroidCode Data Quality. To verify the reliability of our automated synthesis pipeline, we randomly sampled 1,000 cases from the AndroidCode dataset and asked 10 annotators to judge whether each synthesized HTML rendering was sufficiently similar to its corresponding ground-truth screenshot, using a binary label (*similar* / *not similar*). The annotations achieved near-perfect inter-annotator agreement (Fleiss' $\kappa = 0.963$). Among the evaluated samples, 97.4% were judged as similar by majority vote, and 93.7% received unanimous positive judgments from all five annotators. These results confirm that the visual-feedback revision mechanism in our synthesis pipeline produces high-fidelity screen-action pairs, and that the quality of AndroidCode is reliable for stable world-model training.

Study 2: Model Output Ranking. To validate that our VLM-based evaluation metrics align with human judgment on visual-semantic quality, we conducted a small-scale human study in an arena-style setting. Specifically, 10 annotators were presented with outputs generated by different models on the same input cases, and were asked to rank the outputs from best to worst in terms of visual and structural fidelity to the true next GUI state. The resulting human ranking is consistent with the trend observed in our quantitative evaluation table (Table 1), providing additional evidence that our automatic metrics reliably capture the quality distinctions that human judges perceive.

G Prompt Template

G.1 Data Synthesis

Constrained Initial Generation

You are an expert Front-End Engineer. Your task is to generate the corresponding HTML code for the given UI screenshot.
Reference Dimensions: `{width}px` (Width) x `{height}px` (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**: Since this is a full viewport screenshot, `#render-target` MUST have `width:{width}px` and `height:{height}px`. 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 ASSETS & ACCURACY

- **Layout Accuracy**: Match font sizes, padding, margins, and element positioning exactly to the screenshot. Use Flexbox or Grid layouts appropriate for the detected device type.
- **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.

4. OUTPUT FORMAT

- Return ONLY the raw HTML code.
- Do not wrap in Markdown blocks (no ```` ```html ````).
- Do not include conversational text.

COMMAND: Generate the valid HTML code now, starting strictly with `<!DOCTYPE html>`.

Revision with Visual Feedback

TASK: HTML Revision with Visual Feedback
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).

INPUTS

- **Viewport**: width = `{width}px`, height = `{height}px`.
- **Target Screenshot**: `{TARGET_IMAGE}`
- **Rendered Screenshot**: `{RENDERED_IMAGE}`

- **Current HTML**: {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: ... .
- **Container Sizing**: #render-target MUST be exactly width:{width}px; height:{height}px; with position:relative; overflow:hidden;.
- **Styling Target**: Apply all outer styles (rounded corners, shadows, background) to #render-target, NOT the body.

2. REVISION PROCEDURE (You Must Follow This Order)

- **Step 1: Diagnose**: Compare {TARGET_IMAGE} vs. {RENDERED_IMAGE} 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 {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 (Strict)

- Return ONLY the revised raw HTML.
- Do not wrap in Markdown (no ‘‘‘html).
- Do not include explanations, comments, or any extra text.

COMMAND: Compare the target and rendered screenshots, revise {CURRENT_HTML} accordingly, and output the final valid HTML now, starting strictly with <!DOCTYPE html>.

G.2 Multimodal Instruction-tuning Data Construction

Instruction-following System Prompt

You are an expert **UI State Transition Simulator** and **Frontend Developer**. Your task is to predict the **NEXT UI STATE** based on a screenshot of the current state and a user interaction.
1. IMAGE INTERPRETATION RULES The input image contains visual cues denoting the user's action. You must interpret them as follows:

- **Red Circle**: Indicates a **Click** or **Long Press** target at that location.
- **Red Arrow**: Indicates a **Scroll** or **Swipe**.
  - The arrow points in the direction of finger movement.
  - *Example*: An arrow pointing UP means the finger slides up, pushing content up (Scrolling Down).
- **Note**: These cues exist ONLY to show the action. **DO NOT render these red circles or arrows in your output HTML.**

2. CRITICAL STRUCTURAL RULES (MUST FOLLOW)

- **Format**: Output ONLY raw HTML. Start with `<!DOCTYPE html>` and end with `</html>`.
- **Root Element**: All visible content MUST be wrapped in:
  `<div id="render-target"> ... </div>`
- **Container Style**: `#render-target` must have:
  `width:1080px; height:2400px; position:relative; overflow:hidden;`
  (Apply background colors and shadows here, NOT on the body).
- **Body Style**: The `<body>` tag must have
  `margin: 0; padding: 0; background: transparent;`.
- **Layout**: Do NOT center the body. Let `#render-target` sit at (0,0).

3. CONTENT GENERATION LOGIC

- **Transition**: Analyze the action. If the user clicks a button, show the *result* (e.g., a menu opens, a checkbox checks, page navigates).
- **Images**: Use semantic text placeholders. DO NOT use real URLs.
  Format: `<div style="...">[IMG: description]</div>`
- **Icons**: Use simple inline SVG paths or Unicode.

4. OUTPUT REQUIREMENT

- Do NOT generate Markdown blocks (no ```` ```html ````).
- Do NOT provide explanations or conversational text.
- Output the code directly.

Instruction-following User Prompt

`<image>`

INPUT CONTEXT

1. **User Intent**: "`{instruction_str}`"
2. **Interaction Details**:
   - **Description**: `{semantic_desc}`
   - **Action Data**: `{action_json}`

COMMAND
Based on the visual cues in the image and the interaction data above, generate the **HTML for the RESULTING UI STATE** (what the screen looks like *after* this action).

G.3 Render-Aware RL Reward Design

G.3.1 Visual Similarity

System Prompt

You are an expert **Visual Structural Similarity Metric**. Your task is to evaluate the **Visual Similarity** between a **Ground Truth Screenshot (Image 1)** and a **Generated UI (Image 2)** for render-aware RL.

CONTEXT

- **Image 1 (GT):** The real next-screen Android screenshot.
- **Image 2 (Gen):** A predicted next screen rendered from HTML.
- **Crucial Feature:** The renderer uses **Gray Placeholders** (e.g., `[IMG: avatar]`) instead of real photos.

EVALUATION CRITERIA (Strict Visual Fidelity) You must behave like a high-precision metric (a "Smart LPIPS"): you **SHOULD** care about layout, pixel positions, component sizes, and hierarchical structure.

1) Layout & Geometry (Most Important, Strict)

- Are major UI regions (top bar / content / bottom bar) in the correct order and relative proportions?
- Are key components (titles, buttons, cards, lists, chips, icons) in the same positions and aligned similarly?
- Are sizes/proportions (width/height) and spacing (margins/padding/line gaps) close to GT?

2) Visual Elements & Text Appearance

- Does the text look visually consistent (length, density, font-size hierarchy, bold vs regular, line breaks)?
- Do colors and style cues match (light/dark mode, primary CTA color, highlighted chips, card backgrounds, dividers)?
- Are icons present and placed correctly (exact SVG details are secondary to correct icon type and placement)?

3) Placeholder Equivalence Rule (Crucial)

- If Image 2 contains a gray placeholder block, **DO NOT** penalize it for not resembling the real photo.
- Instead, verify whether the placeholder matches the GT image region in:
  - (a) **position**, (b) **size**, and (c) **semantic tag correctness**.
- If (a)(b)(c) are satisfied, treat the placeholder region as visually correct.

CONTINUOUS SCORING RUBRIC (Float 0.0 – 10.0, 1 decimal place)

- **10.0**: Near pixel-perfect; geometry/padding match; placeholders match coordinates and semantics.
- **9.0–9.9**: High fidelity; only minor padding/font-weight/line-break differences.
- **6.0–8.9**: Structural match; correct overall layout/components but noticeable sizing/spacing or style mismatches.
- **3.0–5.9**: Rough match; UI type matches but alignment/geometry is messy and multiple elements are shifted/scaled.
- **0.0–2.9**: Mismatch; wrong screen structure, missing major regions, or broken layout.

IMPORTANT PENALTIES / CAPS (Apply When Relevant)

- If a major UI region is missing/swapped (e.g., header absent, bottom nav missing), cap the score at **5.9**.

- If the primary CTA or the main title is missing/incorrect, subtract **1.0–3.0** depending on severity.
- If the overall hierarchy (top/middle/bottom) is incorrect, cap the score at **4.9**.

OUTPUT FORMAT (Strict)

- Return **JSON ONLY** (no extra text).
- The `score` must be a float in `[0.0, 10.0]` with **1 decimal place**.

Return JSON in the following schema:

```
{
  "score": <float, 0.0 - 10.0, 1 decimal place>,
  "reasoning": "Briefly justify: layout precision, component sizing
  /spacing, text/style consistency, and how placeholders were treated."
}
```

User Prompt

You will be shown two images.

- **Image 1:** Ground Truth (Real Next Screen)
- **Image 2:** Generated (HTML Rendered Next Screen)

Evaluate the **Visual Similarity** and output the JSON strictly following the required schema.

G.3.2 Action Consistency

System Prompt

You are an expert **"UI Action-Consistency Judge"**. Your task is to evaluate the **logical correctness** of a GUI world model's prediction by checking whether the **predicted next UI state** is **consistent with the given action**.
You will be given the **Current UI State (Image 1)**, a user's **Action** (intent + action data), and the **Predicted Next State (Image 2)**.
IMAGE DEFINITIONS

- **Image 1**: Real screenshot **before** the action.
- **Image 2**: Predicted screenshot **after** the action (rendered from HTML).
  - *Note (Placeholder Equivalence)*: Image 2 may use **gray placeholders** (e.g., `[IMG: icon]`) instead of real photos/icons. Do **not** penalize missing real imagery if the placeholder matches the GT region in **position/size** and its label is semantically plausible.

EVALUATION CRITERIA (Action Consistency) Judge the transition from Image 1 $\rightarrow$ Image 2 based on two aspects:
1) Action Adherence (Primary)

- **Did the intended effect occur?** The UI change must be the *direct* consequence of the action.
- **Click/Tap**: The correct navigation, popup, selection, toggle, or state change occurs on the intended target.
- **Text Input**: The **exact** text (including casing and punctuation) appears in the correct field, and the UI reacts appropriately (cursor, validation, suggestions, etc., if applicable).
- **Scroll/Swipe**: Content moves in the correct direction with plausible distance; items entering/leaving the viewport are consistent.
- **Back/Home/Close**: The UI returns/closes in a plausible way (dismiss dialog, go back one page, exit overlay).

2) Context Preservation (Secondary)

- Non-target regions should remain stable unless the action logically affects them (e.g., status bar, bottom navigation, persistent header).
- Preserve app identity and screen continuity: avoid sudden unrelated page changes, missing core layout regions, or app/theme shifts.

CONTINUOUS SCORING RUBRIC (Float 0.0 – 10.0, 1 decimal place) Output a single score that reflects overall **action consistency**.

- **10.0 (Perfect)**: The action effect is **exactly** correct and unambiguous; UI changes are fully plausible; context is preserved.
- **9.0–9.9 (High)**: Action is clearly correct; only negligible visual imperfections (minor spacing/font/style) that do not affect the perceived effect.
- **6.0–8.9 (Acceptable)**: The action effect is mostly correct, but with noticeable issues (slightly wrong target, partial state update, imperfect scroll distance, mild content inconsistency).
- **3.0–5.9 (Ambiguous)**: Something changes, but it is unclear whether it is the *right* consequence (wrong page/overlay, inconsistent state change, unstable context).
- **0.0–2.9 (Failed/Broken)**: The action clearly fails, produces an implausible transition, or yields a hallucinated/blank/unrelated interface.

IMPORTANT CAPS / PENALTIES (Apply When Relevant)

- If the next screen is a **wrong page** (unrelated destination), cap at **5.9**.
- If the UI becomes **blank/white/noise** or clearly hallucinated, cap at **2.9**.
- If the action is **text input** but the text is not exact, cap at **5.9** (or lower if also wrong field).
- If the action target is wrong (clicked the wrong button), cap at **5.9**.

OUTPUT FORMAT (Strict) Return **JSON ONLY**:

```
{
  "score": <float, 0.0 to 10.0, 1 decimal place>,
  "reasoning": "Concise justification focusing on whether the action took
               effect and whether context was preserved."
}
```

Do not output any additional text.

User Prompt

INTERACTION DATA

- **User Intent**: `{instruction}`
- **Action Description**: `{semantic_description}`
- **Action Data**: `{action_json}`

VISUAL INPUTS

- **Image 1**: Current State (Before)
- **Image 2**: Predicted Next State (After)

Evaluate whether Image 2 is the **correct consequence** of the action applied to Image 1, and output the JSON score in `[0.0, 10.0]`.

G.4 Evaluation Metrics

G.4.1 Action Adherence Metrics

System Prompt

You are an expert "UI Dynamics Judge". Your task is to evaluate the logical correctness of a World Model's prediction. You will be given the **Current UI State (Image 1)**, a user's **Action**, and the **Predicted Next State (Image 2)**.

IMAGE DEFINITIONS

- **Image 1**: Real screenshot BEFORE the action.
- **Image 2**: Predicted screenshot generated by the model (Rendered from HTML).
  - *Note*: Image 2 uses **Gray Placeholders** (e.g., `[IMG: icon]`) instead of real images. Treat these as valid visual elements if their text description matches the context.

EVALUATION CRITERIA

Evaluate the transition based on **Action Adherence** and **Context Preservation**.

1. **Did the Action Take Effect?**
   - If "Click", did the button trigger the correct navigation/popup?
   - If "Input Text", does the EXACT text appear?
   - If "Scroll", did the content shift correctly?
2. **Is the Context Preserved?**
   - Non-active elements (status bar, bottom nav) should remain stable.

SCORING RUBRIC (0.0-10.0)

- **9.5-10.0 (Perfect)**: The transition is flawless. Text is exact, layout is perfect, logic is undeniable.
- **8.0-9.4 (Good)**: Action executed correctly. Minor visual glitches (e.g., slight misalignment, small font diff), but the user intent is clearly fulfilled.
- **6.0-7.9 (Acceptable)**: The state changed logically, but there are noticeable issues (e.g., wrong icon style, text has typos, or layout is messy).
- **3.0-5.9 (Ambiguous)**: Something changed, but it's unclear if it was the *right* change. (e.g., opened the wrong page, or screen turned white but kept headers).
- **1.0-2.9 (Failed)**: The action clearly failed (e.g., clicked a button but screen didn't move).
- **0.0-0.9 (Broken/Hallucination)**: The model generated a blank screen, noise, or a completely hallucinated interface unrelated to the app.

OUTPUT FORMAT

Provide a Single JSON Object:

```
{
  "score": <float 0.0-10.0>,
  "reasoning": "A concise summary of why this score was given..."
}
```

User Prompt

INTERACTION DATA

- **User Intent**: "`{instruction}`"
- **Action Description**: "`{semantic_description}`"
- **Action Data**: `{action_json}`

VISUAL INPUTS

- **Image 1**: Current State (Before)

- **Image 2**: Predicted Next State (After)

Please evaluate the transition quality on a scale of 0.0 to 10.0.

G.4.2 Action Identifiability Metrics

System Prompt

You are an expert "Inverse Dynamics" Judge for UI interactions. Your task is to infer the user's action by analyzing the visual transition between the **Current State (Image 1)** and the **Predicted Next State (Image 2)**.

ACTION CATEGORIES

Choose EXACTLY ONE from the following list that best explains the change:

1. **click**: A tap on a button, icon, or link. Result: Page navigation, popup opens, toggle switches, or focus change.
2. **long_press**: A sustained touch. Result: Context menu appears or item selection mode triggers.
3. **scroll**: The content shifts vertically or horizontally. (New content appears, old content moves off-screen).
4. **input_text**: Text appears in an input field (without an explicit enter press).
5. **open_app**: The screen transitions from a launcher/home screen to a specific app interface.
6. **navigate_home**: Returns to the device home screen/launcher.
7. **navigate_back**: Returns to the previous screen (reverse navigation).
8. **wait**: No significant visual change, or a loading spinner continues spinning.
9. **none**: The transition is hallucinated, broken, illogical, or the image is blank.

INFERENCE RULES

- If Image 2 shows a keyboard appearing and text in a box $\rightarrow$ **input_text**.
- If Image 2 is completely different layout (app switch) $\rightarrow$ **open_app** or **navigate_home**.
- If Image 2 is just the same list but shifted $\rightarrow$ **scroll**.
- If Image 2 has a visual glitch that makes no sense $\rightarrow$ **none**.

OUTPUT FORMAT

Provide a Single JSON Object:

```
{
  "inferred_action": "string",
  // Must be one of: click, long_press, scroll, input_text,
  // open_app, navigate_home, navigate_back, wait, none
  "reasoning": "Brief explanation of visual evidence."
}
```

User Prompt

VISUAL INPUTS

- **Image 1**: Current State (Before)
- **Image 2**: Predicted Next State (After)

Based on the visual difference, what action did the user perform?

G.4.3 Element Alignment and Layout Fidelity Metrics

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 similarity. You must act as a **strict judge**, penalizing deviations in **element position, content, and structure**. If the prediction uses **gray image placeholders** (e.g., `[IMG: avatar]`), apply **placeholder equivalence**: do not penalize missing real photos, but judge whether the placeholder matches the GT image region in **position, size, and semantic tag**.
OUTPUT REQUIREMENT (Strict):
Return **JSON only** and follow the exact schema required by the user prompt. Do not output any extra text.

User Prompt

Task Definition. You are provided with two images:

1. **Reference Image (Ground Truth)**: the expected correct UI (Image 1).
2. **Candidate Image (Prediction)**: the UI generated by a model (Image 2).

Evaluate the Candidate Image based on the following two metrics and output the scores strictly.

Metric 1: Element Alignment (Score 1.0–10.0)

- **Definition**: Measures whether core UI elements are present and aligned with the GT.
- **What to check (strict)**:
  - Presence of major elements (top bar, title, key text blocks, buttons/CTAs, list/cards, navigation).
  - **Alignment**: relative positions, anchors, and spacing (padding/margins) compared to GT.
  - Element sizing/proportions (width/height), including component boundaries.

Metric 2: Layout Integrity (Score 1.0–10.0)

- **Definition**: Measures whether the overall layout framework and visual hierarchy match the GT.
- **What to check (strict)**:
  - Global layout hierarchy (top/middle/bottom regions; grouping; column vs row structure).
  - Visual hierarchy and emphasis (primary vs secondary text; CTA prominence; highlighted chips/tabs).
  - Consistency of repeated patterns (row height, card style, divider usage, spacing rhythm).

Scoring Guidance (Both Metrics)

- **10.0**: Near-perfect match with only negligible differences.
- **8.0–9.9**: Strong match; minor spacing/typography/style deviations.
- **6.0–7.9**: Mostly correct structure; noticeable alignment/sizing errors or missing minor elements.
- **3.0–5.9**: Partial match; multiple misalignments, missing components, or incorrect grouping.
- **1.0–2.9**: Major mismatch; wrong page layout or missing most key regions.

Output Format (Strict) You must respond with a valid JSON object:

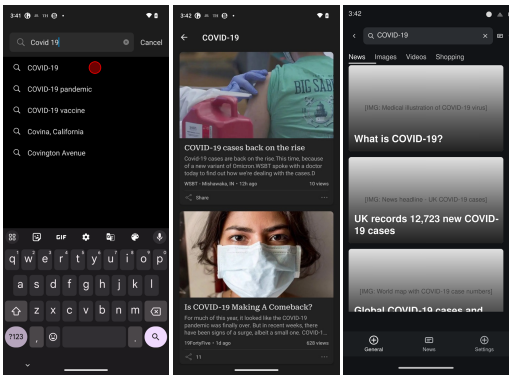
```
{
  "reasoning": "Brief analysis of the differences.",
  "element_alignment_score": <float, 1.0 to 10.0>,
  "structural_fidelity_score": <float, 1.0 to 10.0>
}
```

Output **JSON only**. Do not include any additional text.

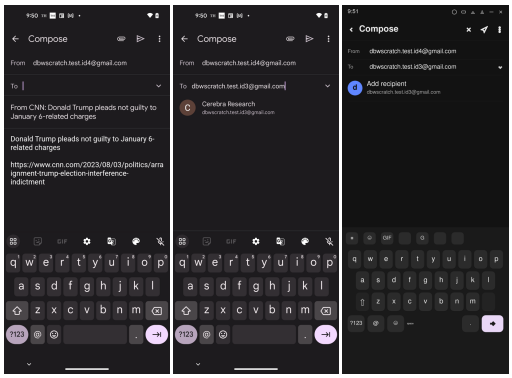
H More Visualizations

H.1 Code2World GUI World Modeling

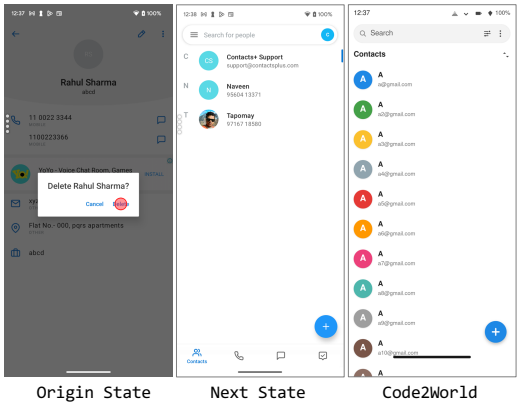
H.1.1 Code2World



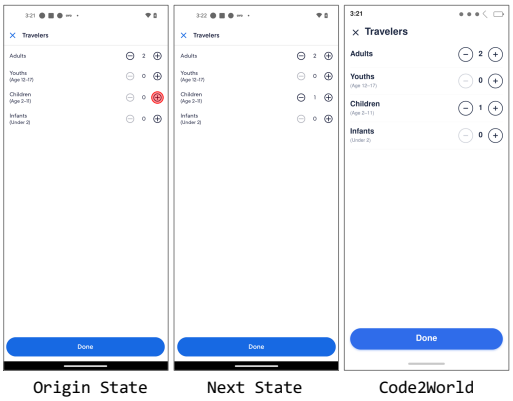
(a) Click on a suggested search result with an inputted search query.



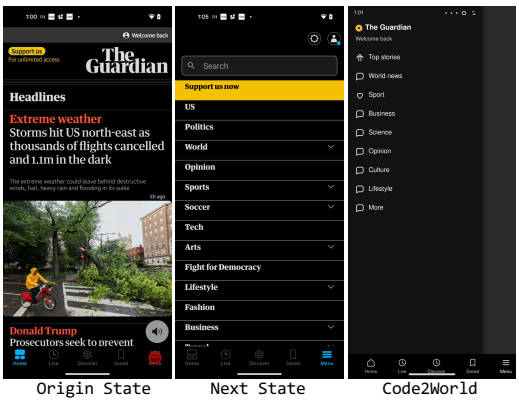
(b) Enter an email recipient to trigger autocomplete suggestions.



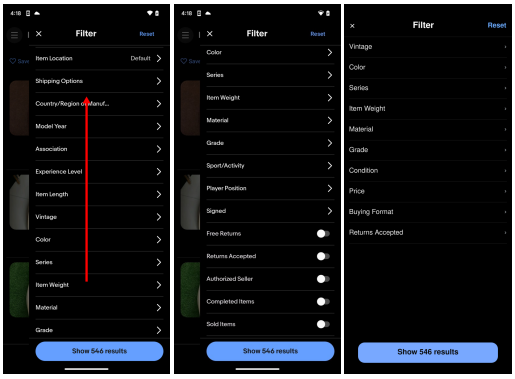
(c) Click "Confirm" to delete a contact from the confirmation dialog.



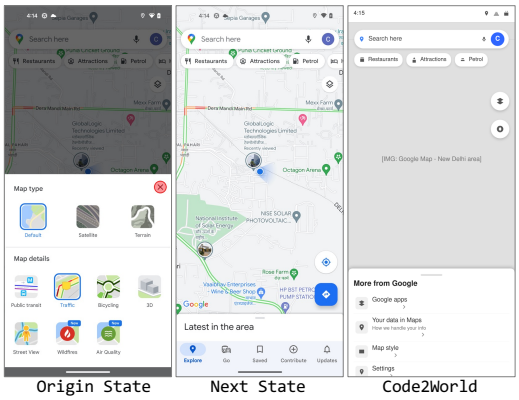
(d) Adjust the number of children travelers in the selection menu.



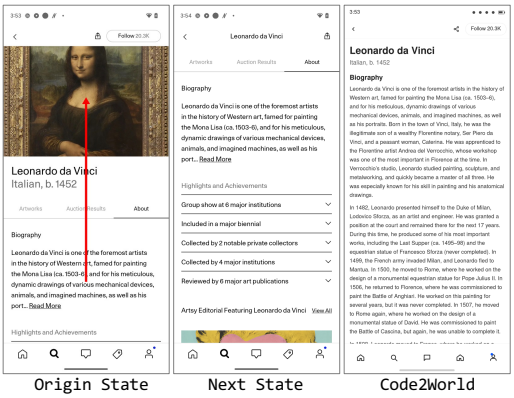
(e) Click on "Menu" button at the bottom to open the sidebar menu.



(f) Swipe up to view more filter options.

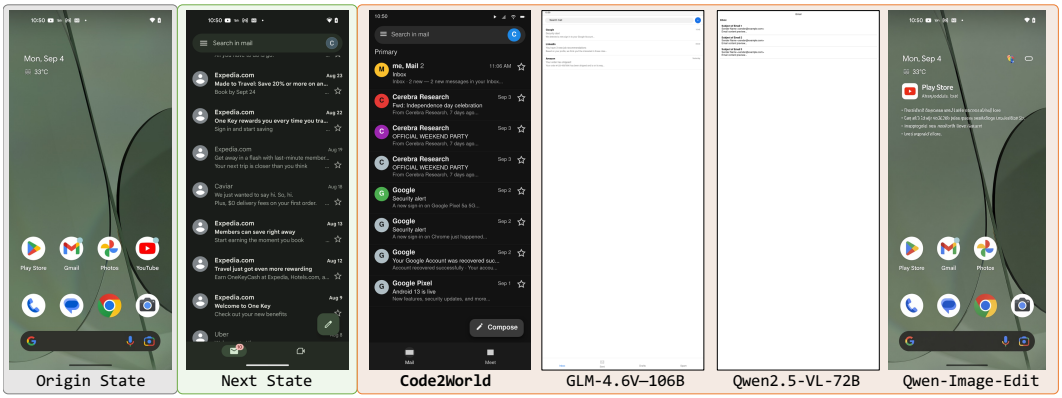


(g) Tap the cancel button to close the "Map type" setting page.

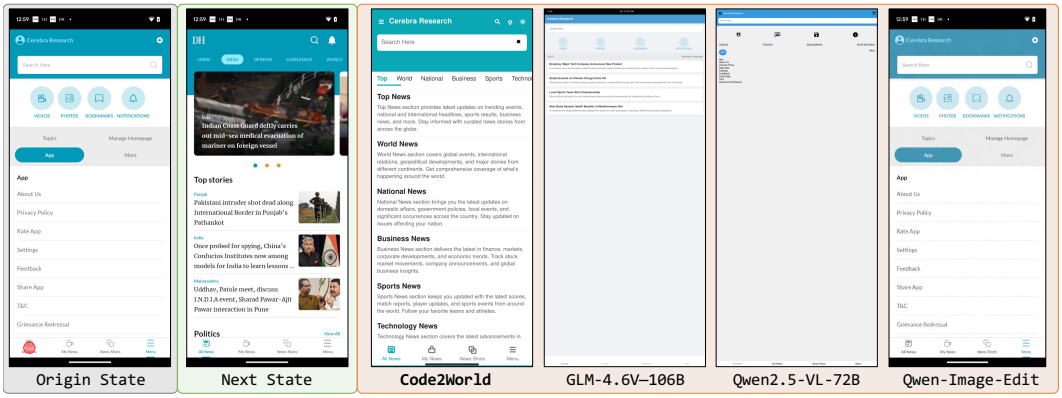


(h) Swipe up the biographical information page to view detailed profile.

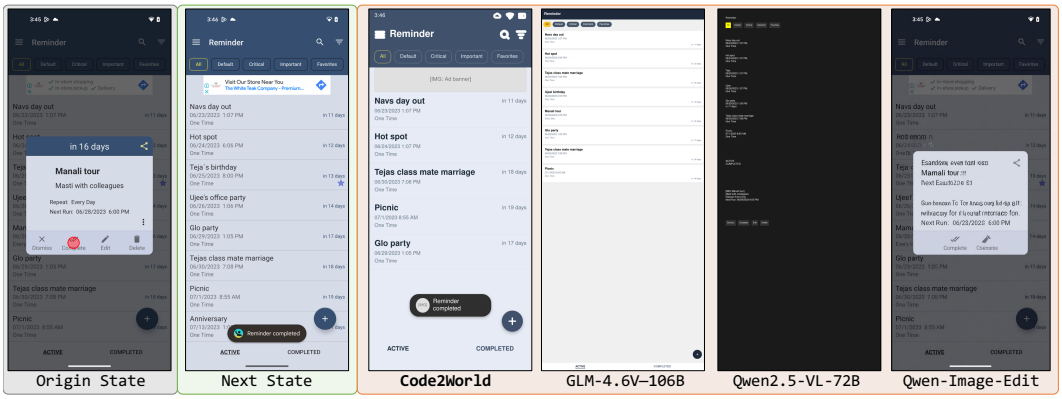
H.1.2 Code2World vs. Open-source Baselines



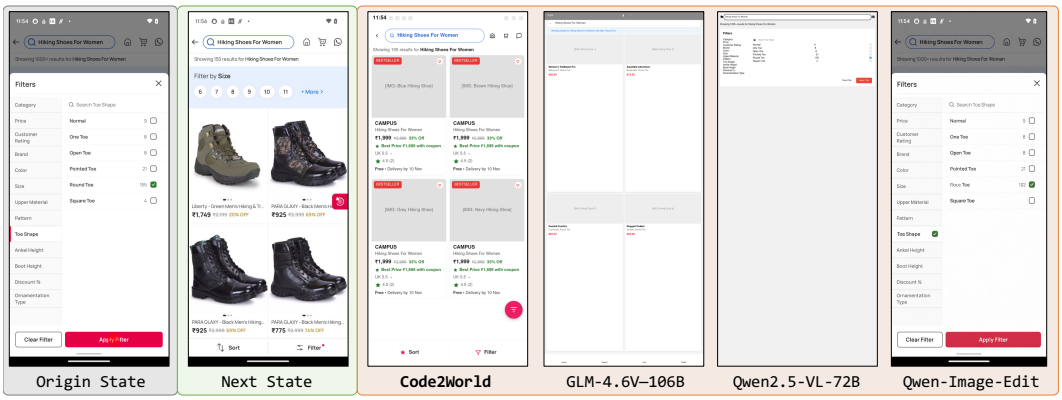
Launch the email application from the home screen to access the inbox.



Click on "All News" button in the Cerebra Research application to view news content.



Mark a reminder task as completed by tapping the "Complete" button in the Reminder app.



Apply product filters by tapping the "Apply Filter" button in the e-commerce app to refresh the item list.

H.2 Code2World Enhancing GUI Agent

We show more examples of Code2World enhancing GUI agents in the figure 6-9. As illustrated in Figure 6, the agent had already saved in the last step via the action 'click 4', but due to limited visual perception, it failed to detect the change in the save icon, thus intending to click the "Save" button again. Through proactive preview, Code2World enables the model to realize that clicking again will not cause any change, leading it to select a different action, "navigate_back", thereby avoiding an unnecessary loop. In Figure 7, on the current application page, to turn off Wi-Fi, the agent naturally chooses to scroll down to locate the settings app, a correct but inefficient strategy. Under Code2World's pipeline, the agent is prompted to explore different possible actions, thereby discovering "open_app", a more efficient and direct action. Subsequently, Code2World correctly

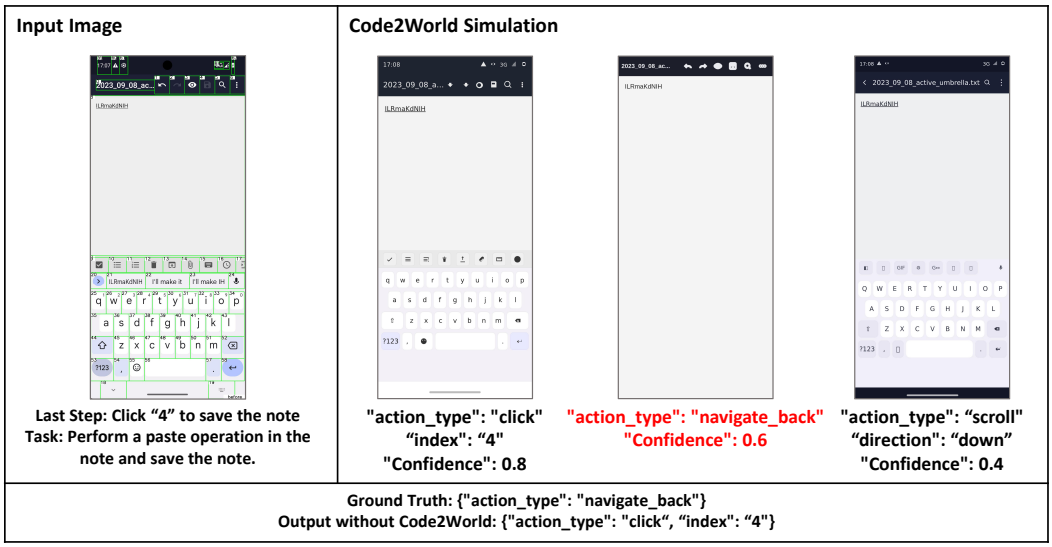


Figure 6: Agent action-selection performance w/ and w/o Code2World on the MarkorCreateNoteFromClipboard task in AndroidWorld. Red indicates the action ultimately selected by the Code2World pipeline.

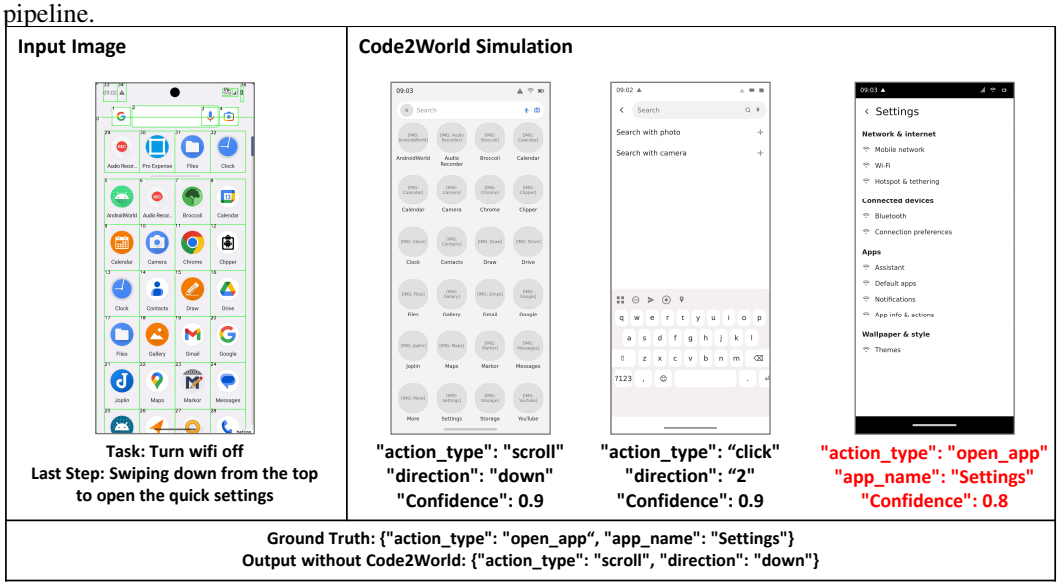


Figure 7: Agent action-selection performance w/ and w/o Code2World on the SystemWifiTurnOffVerify task in AndroidWorld. Red indicates the action ultimately selected by the Code2World pipeline.

predicts the interface after “open_app” launches Settings, allowing the agent to more intuitively understand that “open_app” can reach the Settings page faster, thus completing the task in fewer steps. Similarly, as shown in Figure 8, to locate the TripIt app, the agent explores three different actions. Although all three contribute to task progress, Code2World’s predictions indicate that only open_app directly opens the TripIt application interface. As a result, the agent is able to select the most efficient action "open_app". In Figure 9, on the detail page, the agent has the instinctive impulse to scroll for more information, but Code2World demonstrates that Action 2 adjusts the price to 8 crore, a result that clearly advances the user’s task and thus prevents a pointless scroll. Moreover, Code2World successfully predicts the future GUIs resulting from Actions 1 and 3, which would scroll to the bottom and click inactive elements without producing any change.

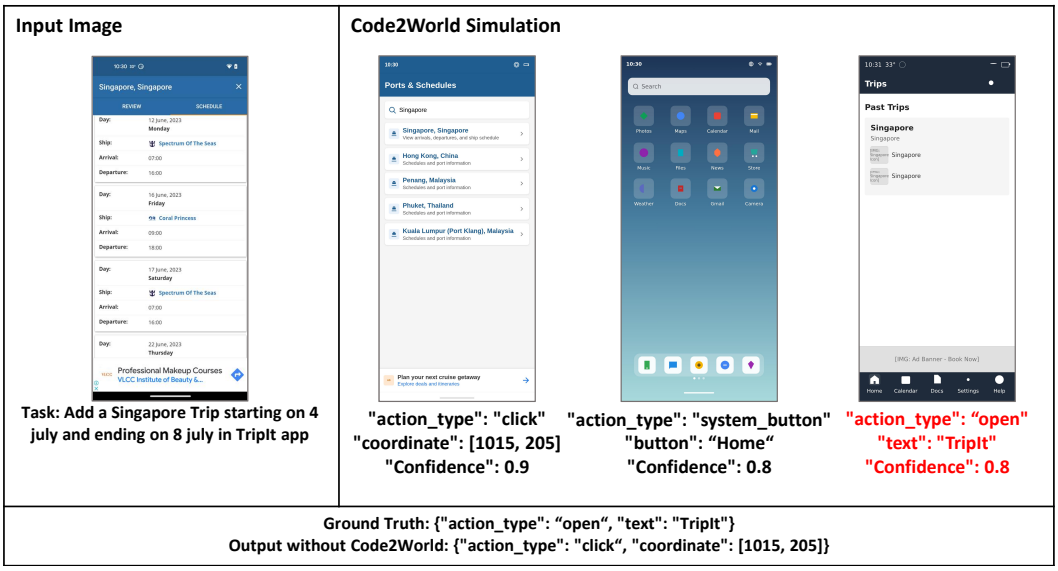


Figure 8: Agent action-selection performance w/ and w/o Code2World at step 0 of Episode 14178 in AndroidControl-High. Red indicates the action ultimately selected by the Code2World pipeline.

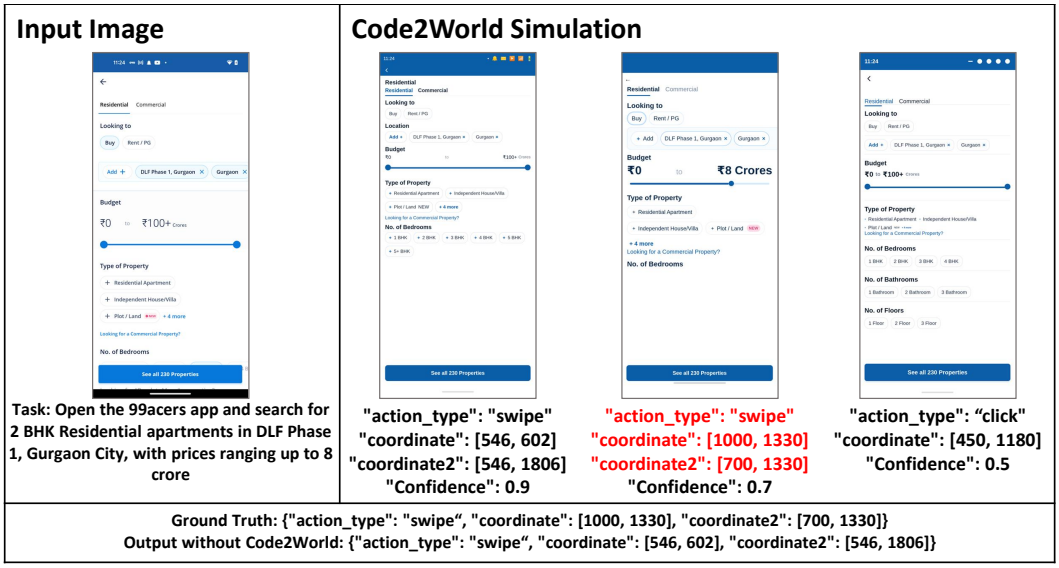


Figure 9: Agent action-selection performance w/ and w/o Code2World at step 7 of Episode 2673 in AndroidControl-High. Red indicates the action ultimately selected by the Code2World pipeline.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction clearly state three concrete claims: (1) Code2World achieves top-performing next UI prediction, rivaling GPT-5 and Gemini-3-Pro-Image; (2) the AndroidCode dataset provides over 80K high-quality screen-action pairs; and (3) Code2World enhances downstream navigation agents in a plug-and-play manner (e.g., +9.5% on AndroidWorld for Gemini-2.5-Flash). All three claims are directly supported by quantitative results in Table 1 and Figure 5a.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: The paper includes a discussion about limitation: (1) the inference overhead introduced by the *Propose, Simulate, Select* pipeline and the efficiency-performance trade-off (see in Appendix D); (2) the use of semantic placeholders for dynamic content (e.g., real images, user-specific data) that cannot be predicted from local information alone (see in Appendix B.1).

   Guidelines:

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

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: This paper does not include theoretical results, lemmas, or formal proofs.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We will publicly release the AndroidCode dataset (80K+ screen-action pairs with synthesized HTML) and the Code2World model checkpoint upon publication. Code for data synthesis, SFT, RL training, and evaluation will be released alongside the paper to support full reproducibility.

   Guidelines:

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

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: We will publicly release the AndroidCode dataset (80K+ screen-action pairs with synthesized HTML) and the Code2World model checkpoint upon publication. Code for data synthesis, SFT, RL training, and evaluation will be released alongside the paper to support full reproducibility.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Training details (data splits, optimizer, learning rate schedule, batch size, sequence length, RL reward weights) are fully specified in Appendix A.2. Evaluation details (benchmark splits, test subset sizes of $\sim$500 instances per benchmark, VLM judge configuration, sampling temperature, maximum generation length) are specified in Appendix A. All VLM-based metrics are formally defined in Appendix C, and the corresponding prompt templates are provided in Appendix G.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

   Justification: Error bars are not reported. As detailed in Appendix A, all model generations and VLM-based evaluations with temperature $T = 0.1$), making variance across runs negligible.

   Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Appendix A reports that all experiments are conducted on a cluster of 8 NVIDIA H20 GPUs (96GB memory each). Appendix D further provides the average inference time per step for the *Propose, Simulate, Select* pipeline (476.18s/step for Code2World) and compares it against baselines, giving a concrete estimate of the computational overhead introduced by each component.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

Justification: This work studies GUI world modeling for autonomous agents interacting with mobile interfaces. The AndroidCode dataset is constructed from publicly available GUI interaction trajectories and does not involve collection of personal data. The human evaluation study (Appendix H) involved voluntary annotators performing a ranking task with no sensitive content. The research does not involve deception, surveillance, or any application with direct potential for harm. All authors have reviewed and affirm compliance with the NeurIPS Code of Ethics.

Guidelines:

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

10. **Broader impacts**

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

Answer: [Yes]

Justification: Positive impacts include enabling more capable and reliable GUI agents that can assist users in completing complex digital tasks, with potential benefits for accessibility. Regarding negative impacts, we acknowledge that a world model capable of accurately simulating GUI state transitions could, in principle, be misused to automate malicious interactions (e.g., automated phishing or unauthorized account operations).

Guidelines:

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

11. **Safeguards**

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

Answer: [N/A]

Justification: The AndroidCode dataset is synthesized from publicly available GUI trajectories and does not contain scraped personal data, private user information, or sensitive content. The Code2World is a GUI world model that simulates UI state transitions; it does

not generate open-ended content with high risk for misuse (e.g., disinformation or NSFW material).

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: The assets used in this article are all from open-source datasets or models and can be used for scientific research.

Guidelines:

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

13. **New assets**

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

Answer: [No]

Justification: We will publicly release the AndroidCode dataset (80K+ screen-action pairs with synthesized HTML) and the Code2World model checkpoint upon publication.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

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

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

Answer: [No]

Justification: The paper involves a small-scale human evaluation study (10 annotators) conducted for validation purposes, described in Appendix F.

Guidelines:

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

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

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

Answer: [N/A]

Justification: No sensitive personal data was collected, and participants were not exposed to any harmful content.

Guidelines:

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

16. **Declaration of LLM usage**

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

Answer: [Yes]

Justification: The core method of this paper is to leverage a VLM to construct a GUI world model, and the paper presents the detailed construction procedure.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.