IMAGINE TO ACT: HIGH-FIDELITY DATA SYNTHESIS VIA IMAGE EDITING WORLD MODEL FOR SCALABLE GUI AGENT TRAINING

Anonymous authors

Paper under double-blind review

ABSTRACT

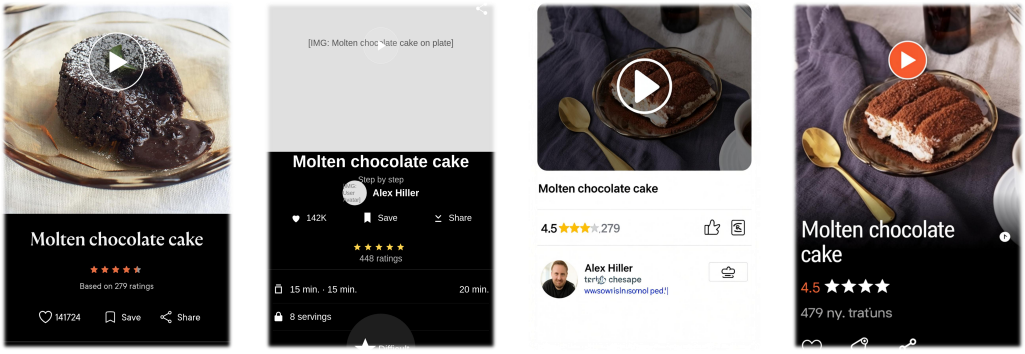
Graphical User Interface (GUI) agents have emerged as a promising paradigm for automating complex digital workflows across diverse applications. However, training highly capable and generalizable agents fundamentally relies on massive, high-fidelity visual-action trajectories, which are notoriously difficult to acquire. While human demonstrations are unscalable, existing GUI world models rely on text descriptions or HTML rendering, discarding crucial pixel-level visual details like icons and layout styles. To address this issue, we introduce Infinite-Dreamer, a simulation-free data synthesis method powered by a pixel-level Image Editing World Model. By conceptualizing GUI transitions as image editing tasks, we leverage Vision-Language Models (VLMs) to describe action-induced UI changes as structured delta-text. We then fine-tune an image editing backbone to controllably synthesize realistic screenshot transitions. We utilize this model to generate both single-frame visual robustness data and multi-step imaginary trajectories. To validate the effectiveness of our approach, we fine-tune the Qwen3-VL baseline solely on the synthesized data to obtain Infinite-Actor, and evaluate it on AndroidWorld, MobileWorld, and AndroidControl-Curated benchmarks. Infinite-Actor consistently outperforms the Qwen3-VL baselines across scales: Infinite-Actor-8B improves AndroidWorld Pass@1 by +4.45 and nearly doubles the MobileWorld Pass@3 success rate, while Infinite-Actor-2B improves Pass@1 by +9.05. Code is available at `https://anonymous.4open.science/r/Infinite-Dreamer-B499/`.

1 INTRODUCTION

Graphical User Interface (GUI) agents have rapidly emerged as a pivotal frontier in artificial intelligence, promising to seamlessly automate increasingly complex digital workflows across diverse operating systems and web applications. Unlike text-based language agents, GUI agents must perceive rich visual screens, effectively reason over layout structures, and accurately execute precise actions just like human users. To achieve robust generalizability across infinitely varied real-world digital environments, these agents must master the intricate mapping between natural language instructions, dynamic screen layouts, and sequential action executions. Ultimately, the development of such highly capable and adaptable GUI agents is fundamentally contingent upon the availability of massive, diverse, and high-fidelity annotated visual-action training data.

Despite its critical importance, acquiring high-quality trajectory data annotated with explicit action intents remains exceptionally difficult. Existing efforts to obtain such data generally suffer from two major limitations: (i) *Human-driven demonstration* is not only financially prohibitive and labor-intensive, but it is also severely bottlenecked by environmental constraints, such as the technical friction of configuring heterogeneous device platforms and the inherent risks of exposing sensitive user privacy during the recording process; (ii) *Automated environment exploration* typically relies on heuristic search or random-walk strategies. Lacking a long-term semantic objective, these automated agents easily become trapped in trivial UI loops, frequently trigger unrecoverable error dead-ends, or generate disjointed action sequences. Consequently, such exploration-based approaches heavily bias the collected data toward short, repetitive interactions, making them incapable of scaling to complex,

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



Ground Truth **HTML-vision** **FireRed-Image-Edit-1.0** **Infinite-Dreamer-9B (Ours)**

Figure 1: **Motivation.** Post-action screenshots predicted by different world models: (1) ground truth; (2) HTML-vision rendering, lacking visual detail; (3) the pretrained FireRed-Image-Edit-1.0, failing to capture GUI-specific transitions; and (4) our Infinite-Dreamer-9B, producing high-fidelity, pixel-level screenshots faithful to the ground truth.

multi-step workflows and utterly failing to maintain goal coherence over long execution sequences. Therefore, how to scalably synthesize GUI interaction trajectories while improving data efficiency has emerged as a core challenge in training advanced GUI agents.

To address this challenge, the paradigm of world models offers a powerful solution for data synthesis. Fundamentally, world models are specifically designed to internalize the underlying transition dynamics of an environment, predicting subsequent states by learning the causal laws that govern how the world responds to specific actions. In the digital ecosystem, a robust GUI world model acts as an unconstrained virtual simulator, emulating how software applications evolve after a click or a scroll. While recent pioneering GUI world models, such as Qwen-AgentWorld and Code2World, attempt to simulate environments via text descriptions or HTML rendering, they inherently struggle to capture diverse layouts and fine-grained visual features, such as custom icons, banners, and dynamic animations (Figure 1). In contrast, pixel-level screenshots represent the natural, raw state space of GUIs, directly reflecting the visual reality experienced by human users. A pixel-level world model can implicitly learn the visual regularities of UI evolution, such as a loading spinner disappearing or a drop-down menu unfolding, thereby preserving both rich graphical details and complex software execution logic. Therefore, leveraging image editing models directly as GUI world models offers a more natural, direct, and high-fidelity paradigm for realistic data synthesis.

To this end, we introduce Infinite-Dreamer, a high-fidelity, simulation-free data synthesis pipeline that generates pixel-level screenshots and continuous action trajectories without relying on complex and restrictive simulation environments. First, we construct a specialized image editing dataset built upon AndroidControl. By employing a Vision-Language Model (VLM) to describe the transitions of UI elements before and after an action, we generate fine-grained delta-text that precisely captures added or removed interface elements. Second, we apply fine-tuning to a generative image editing backbone. The resulting model serves as a pixel-level GUI world model, capable of editing a pre-action screenshot into a post-action screenshot conditioned on the delta-text. Finally, to demonstrate how our synthesized data enhances downstream agent performance, we generate two distinct categories of samples: (i) visual robustness data, where we augment single screenshots with diverse layouts, colors, and styles to boost the agent's adaptability, and (ii) multi-step imaginary trajectories, where the world model continuously edits an initial screenshot to "imagine" multi-step task executions, expanding state coverage beyond the limits of real-world datasets. To summarize, our major contributions are:

- **A Controllable, Pixel-Level GUI World Model.** We repurpose an image editing model into a screenshot-to-screenshot GUI world model conditioned on structured delta-text. This approach effectively enables the controllable generation of diverse, high-fidelity GUI screens without relying on complex costly simulators.
- **Enhanced Data Efficiency and Generalization.** We demonstrate that our synthesis model significantly boosts the utility of existing trajectories. By blending synthetic and real-world

data, the trained agent generalizes far better to unseen UI layouts and novel interaction patterns, outperforming agents trained purely on real demonstrations.

- **Substantial Downstream Performance Gains.** We train a vision-language agent, called Infinite-Actor, using our newly synthesized data. Evaluation on several challenging online benchmarks reveals substantial dramatic performance gains; notably, it nearly doubles the MobileWorld Pass@3 success rate and yields consistent Pass@1 improvements on AndroidWorld across all three model scales (up to +9.20 points).

2 Related Work

Synthetic Data for GUI Agents. Training GUI agents is bottlenecked by the scarcity of high-quality trajectories, since manual annotation is costly and hard to scale. A first line of work derives trajectories from active environment interaction, using random-walk or exploratory policies to collect step-level actions and state transitions, as in OS-Genesis Sun et al. (2025) and GUIrilla Garkot et al. (2026), with GUI-ReWalk Lin et al. (2025) adding reasoning-guided exploration and WebSynthesis Gao et al. (2025b) simulating transitions via an LLM world model with Monte Carlo Tree Search. A second line reuses existing human knowledge artifacts: AgentTrek harvests and filters web tutorials into structured task specifications replayed by a VLM agent Xu et al. (2025), while Video2GUI mines unlabeled Internet videos to extract grounded interaction trajectories Xiong et al. (2026). A third line trains agents in synthetic environments with verifiable rewards, combining auto-synthesized apps with code-native assertions Cao et al. (2026b), self-supervised pre-training Wu et al. (2026), few-shot demonstrations He et al. (2026), and multi-turn reinforcement learning Wang et al. (2025). In contrast, Infinite-Dreamer synthesizes pixel-level trajectories by editing screenshots directly, bypassing both environment interaction and reward design.

World Models for Visual Environments and GUIs. World models learn environment dynamics to support downstream planning, representation learning, and synthetic rollouts. Beyond classical latent-space formulations (Hafner et al., 2022; 2024), recent general world models leverage expressive diffusion models to synthesize high-fidelity visual frames for video generation (Wang et al., 2024; Bruce et al., 2024; Xiang et al., 2024; Che et al., 2024; Gao et al., 2025a), video editing (Rigter et al., 2024), and diverse interactive environment simulation. Some simulate the environment through text or HTML rendering (Zuo et al., 2026), which discards the visual dynamics that a screenshot-based agent actually observes. Others train visual simulators from scratch: ViMo (Luo et al., 2025) builds a generative visual world model for mobile app agents, MobileDreamer (Cao et al., 2026a) uses sketch-based abstractions for efficient rollout imagination, and NeuralOS (Rivard et al., 2026) simulates OS-scale dynamics via recurrent state tracking and neural rendering.

Image Editing Models. Recently, diffusion-based generative models have achieved photorealistic text-to-image synthesis (Rombach et al., 2022; Betker et al., 2023; Black Forest Labs, 2024; Chen et al., 2024; Brooks et al., 2023). Compared with standard text-to-image models, image editing models additionally accept one or more reference images as auxiliary conditioning signals alongside the text prompt, enabling fine-grained controlled modification of existing visuals. This paradigm has rapidly advanced with strong instruction-following editors such as FLUX.1 (Labs et al., 2025), FLUX.2 (Black Forest Labs, 2025), and Qwen-Image-Edit (Wu et al., 2025). However, we observe that while these models can synthesize photorealistic GUI screenshots, they still lack the underlying procedural knowledge of how apps actually operate within GUI environments, thereby precluding them from serving as genuine GUI world models.

3 Methodology

3.1 Problem Formulation

A mobile GUI agent is considered: at step t, the agent observes screenshot s_t and instruction g, executes action a_t (e.g., click, type, swipe; Appendix B.1), and transitions to $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, forming trajectory $\tau = (s_1, a_1, \ldots, a_{T-1}, s_T)$. Collecting real interaction data is bottlenecked by labor costs and exploration dead-ends. We therefore model transition dynamics P with a generative world model to synthesize trajectories $\tilde{\tau}$ for policy learning $\pi_\theta(a \mid s, g)$.

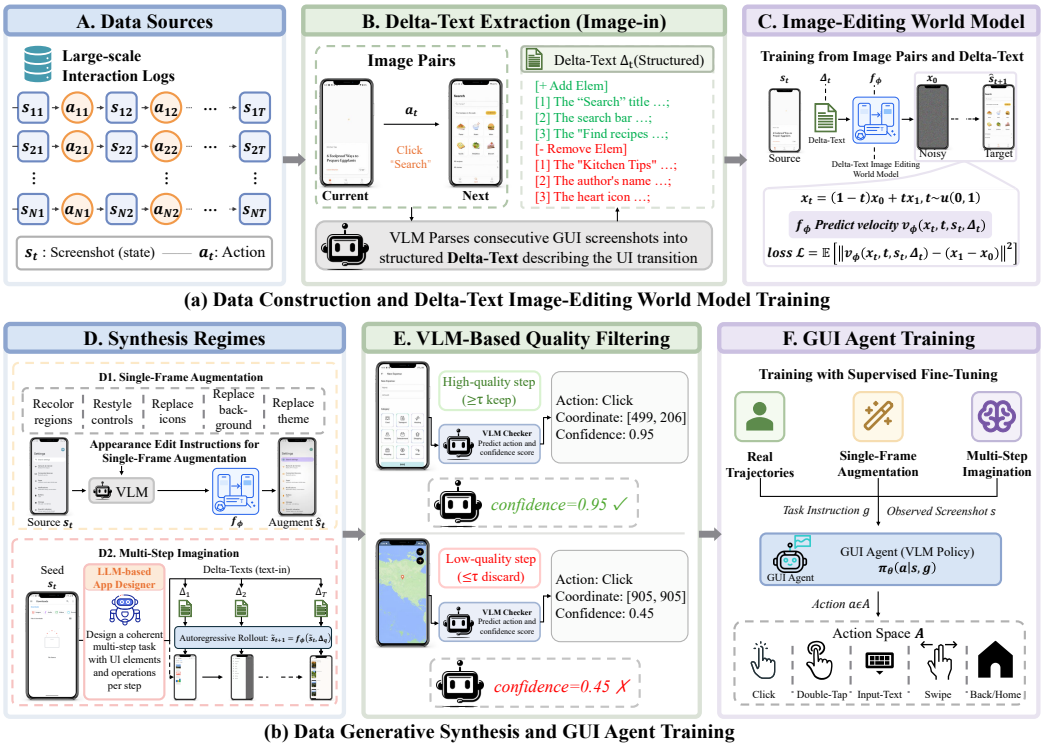


Figure 2: **Overview of our proposed *Infinite-Dreamer*:** (a) Data Construction and Delta-Text Image-Editing World Model Training. We extract structured delta-texts from real UI trajectories and fine-tune an instruction-guided image-editing model as a GUI world model; (b) Data Generative Synthesis and GUI Agent Training. The trained world model enables two synthesis regimes, namely single-frame augmentation and multi-step imagination, whose outputs are filtered by a VLM-based quality checker and used to train the GUI agent.

Because raw action labels carry no visual specification, directly predicting $P(s_{t+1} \mid s_t, a_t)$ is ill-posed. We formulate GUI transition as a *hierarchical world model*: $P(s_{t+1} \mid s_t, a_t) = \sum_{\Delta_t} P_\phi(s_{t+1} \mid s_t, \Delta_t) P_\omega(\Delta_t \mid s_t, a_t)$, where Δ_t is a structured semantic transition specification (*delta-text*). This formulation separates three roles: (i) the **App Designer** (G_ω) plans multi-step goals and UI intents; (ii) the **delta-text** (Δ_t) specifies the transition and affordance contract; and (iii) the **image-editing model** (f_ϕ) simulates continuous visual dynamics, predicting $\hat{s}_{t+1}$.

3.2 Delta-Text Extraction

To build the world model's training set, we use a corpus of real mobile interaction trajectories that provides consecutive (screenshot, action) pairs. For each pair (s_t, s_{t+1}) with action a_t, a vision-language model extractor E_ψ using a template (Appendix C.1) produces a structured description Δ_t, referred to as the *delta-text*, that characterizes the transition:

$$\Delta_t = E_\psi(s_t, s_{t+1}, a_t, a_{t+1}), \tag{1}$$

where a_{t+1} is included so the extractor can identify which UI elements will be relevant for the following step (i.e., the next-step affordances $\mathcal{F}_t$). Concretely, the output is a record $\Delta_t = (o_t, \mathcal{E}_t^+, \mathcal{E}_t^-, \mathcal{F}_t)$ with fields: **Trigger operation** o_t**:** a natural-language paraphrase of the action a_t. **Added elements** $\mathcal{E}_t^+$**:** UI components that appear or become visible. **Removed elements** $\mathcal{E}_t^-$**:** UI components that disappear or become hidden. **Next-step affordances** $\mathcal{F}_t$ **(optional):** task-relevant elements relevant for the subsequent action. The delta-text thus serves as the world model's unified conditioning interface: it is extracted here from real pairs for training, and later authored from synthetic text-only UI descriptions to faithfully drive trajectory imagination.

3.3 IMAGE-EDITING WORLD MODEL TRAINING

We train an image-editing world model f_ϕ that predicts the next screenshot $\hat{s}_{t+1}$ conditioned on both the current screenshot s_t and a delta-text Δ_t:

$$\hat{s}_{t+1} = f_\phi(s_t, \Delta_t). \tag{2}$$

Starting from a pretrained instruction-guided image-editing backbone, we fine-tune it on the paired screenshots, using s_t as the source image and Δ_t as the editing instruction. The backbone follows a standard rectified flow-matching formulation. Let x_0 be the source image and x_1 be the target image; then, with conditioning $c = (s_t, \Delta_t)$ and a velocity network v_ϕ, we sample $t \sim \mathcal{U}(0, 1)$ and minimize the following velocity-prediction objective:

$$\mathcal{L}_{\mathrm{WM}}(\phi) = \mathbb{E}_t \Big[\big\| v_\phi(x_t, t;\, c) - (x_1 - x_0) \big\|_2^2 \Big], \tag{3}$$

where $x_t = (1 - t)x_0 + t\, x_1$ linearly interpolates between source and target images. Unlike conventional renderers that need full scene graphs with exact coordinates, f_ϕ receives coordinate-free semantic diffs and must infer layout reflow, occlusions, and causal state persistence in unedited regions (Appendix A.9); training configurations are in Appendix A.1.

3.4 SYNTHESIS REGIMES

Single-frame Visual Augmentation. Given a screenshot s, we generate an appearance-only variant $\tilde{s}$ that edits the visual appearance while preserving the layout and displayed text:

$$\tilde{s} = f_\phi(s, \Delta^{\mathrm{aug}}), \tag{4}$$

where Δ^{aug} is an appearance-editing delta-text (e.g., recoloring, restyling, or theming; template in Appendix D.1). To enforce this, the prompt constrains the generation so that $\tilde{s}$ approximately preserves the layout and displayed text of s (i.e., $\mathrm{Layout}(\tilde{s}) \approx \mathrm{Layout}(s)$ and $\mathrm{Text}(\tilde{s}) \approx \mathrm{Text}(s)$), thereby keeping the paired task instruction valid.

Multi-step Imagination with an App Designer. To generate high-quality GUI trajectories beyond the coverage of existing data, we introduce an LLM-based App Designer G_ω. Given a seed screenshot s_1, G_ω designs a multi-step task g with per-step UI-element descriptions $\{d_t\}_{t=1}^{T}$ (Appendix D.2). Delta-texts are then authored text-in from these descriptions by a text-only extractor E_ψ^{txt} (Appendix C.2):

$$\Delta_t = E_\psi^{\mathrm{txt}}(d_t, a_t), \quad t = 1, \ldots, T, \tag{5}$$

and the world model subsequently rolls out a trajectory autoregressively, starting from the real seed $\tilde{s}_1 = s_1$:

$$\tilde{s}_{t+1} = f_\phi(\tilde{s}_t, \Delta_t), \quad t = 1, \ldots, T. \tag{6}$$

VLM-based Quality Filtering. Since that autoregressive rollouts can accumulate visual artifacts, we thus score every synthesized step with a VLM: it predicts the intended next action's function call together with a confidence $c_t \in [0, 100]$. Any step with $c_t < \eta$ is discarded (Appendix E), and the retained steps then form the filtered trajectory:

$$\tilde{\tau}^{\mathrm{filtered}} = \{(\tilde{s}_t, a_t) : c_t \geq \eta,\ t = 1, \ldots, T\}. \tag{7}$$

3.5 GUI AGENT TRAINING

Finally, we fine-tune a VLM-based GUI agent over three complementary data sources: (i) real trajectories $\mathcal{D}_{\mathrm{real}}$ collected from mobile interactions, (ii) single-frame augmentations $\mathcal{D}_{\mathrm{aug}}$, and (iii) quality-filtered multi-step imagination trajectories $\mathcal{D}_{\mathrm{gen}}$. We minimize a supervised fine-tuning objective over a selected combination of these sources:

$$\mathcal{L}_{\mathrm{mix}}(\theta) = -\sum_{k \in \mathcal{S}} \mathbb{E}_{(s,g,a) \sim \mathcal{D}_k} \big[\log \pi_\theta(a \mid s, g) \big], \tag{8}$$

where $\mathcal{S} \subseteq \{\mathrm{real}, \mathrm{aug}, \mathrm{gen}\}$ denotes the active sources. The resulting agent's architecture and action-selection prompt are described in Appendices B.2 and B.3, and more detailed training configurations are reported in Appendix A.3.

Table 1: GUI agent performance evaluation. Infinite-Actor (Ours) is trained solely on Infinite-Dreamer synthesis data and paired with its base-model Qwen3-VL counterpart of the same scale; green ↑ / red ↓ indicates whether Infinite-Actor is higher / lower than its counterpart. The **best** and second-best results in each column are **bolded** and underlined, respectively.

Method	AndroidWorld		MobileWorld		AC-Curated-Hard			AC-Curated-Easy		
	Pass@1	Pass@3	Pass@1	Pass@3	Type Acc	Box Acc	Step SR	Type Acc	Box Acc	Step SR
General-Purpose Models										
Claude-Sonnet-4.5	19.25	25.86	33.92	42.11	**73.40**	22.20	30.60	80.90	22.60	35.00
Gemini-3-Flash	34.91	56.03	**43.27**	**52.63**	73.10	48.10	46.00	**84.40**	72.40	68.20
MiniMax-M3	10.34	17.24	14.03	23.68	70.90	47.40	41.60	75.00	45.60	42.50
MiMo-V2.5	30.46	49.14	36.55	**52.63**	73.20	63.40	53.70	81.40	77.10	68.00
Kimi-K2.6	32.18	45.69	40.94	49.12	69.00	48.60	46.50	79.30	69.20	61.20
GUI-Owl-1.5										
GUI-Owl-1.5-2B	29.31	37.93	23.39	29.82	60.20	49.70	30.30	61.50	76.60	47.50
GUI-Owl-1.5-4B	8.05	15.52	23.68	28.07	70.90	74.50	57.70	77.50	95.30	74.70
GUI-Owl-1.5-8B	23.71	36.21	32.46	38.60	66.20	71.10	52.60	69.90	91.80	65.40
Base-Model Qwen3-VL vs. Infinite-Actor (Ours)										
Qwen3-VL-2B	22.56	29.31	5.85	8.77	63.80	56.30	38.30	76.40	92.90	72.10
↪ Infinite-Actor-2B	31.61↑	49.14↑	10.53↑	17.54↑	59.20↓	56.80↑	42.30↑	66.50↓	81.40↓	61.10↓
Qwen3-VL-4B	30.17	43.10	9.65	11.40	68.60	**77.00**	57.10	81.10	**95.50**	78.60
↪ Infinite-Actor-4B	39.37↑	55.17↑	14.91↑	19.30↑	64.40↓	64.70↓	44.20↓	83.70↑	93.20↓	**79.90**↑
Qwen3-VL-8B	39.37	56.90	8.77	11.40	72.30	76.90	**61.60**	78.60	95.30	76.00
↪ Infinite-Actor-8B	**43.82**↑	**62.07**↑	15.20↑	21.93↑	73.30↑	71.30↓	57.90↓	82.40↑	93.80↓	78.80↑

4 Experiment

4.1 Experimental Setup

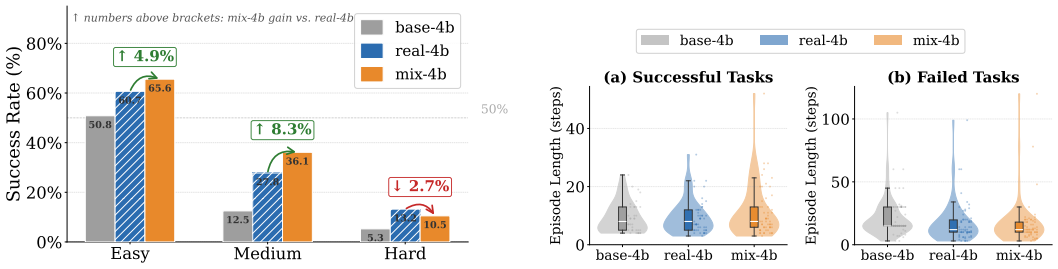
Datasets. The world model is fine-tuned on the AndroidControl (Li et al., 2024) train split and evaluated on its validation split by GLM-4.7v model (using the template in Appendix E.1). The GUI Agent Infinite-Actor is trained solely on the Infinite-Dreamer synthesis data (composition in Appendix A.5), without using any real trajectories. We evaluate our model on AndroidWorld (Rawles et al., 2025) and MobileWorld (Kong et al., 2025), two online benchmarks that require completing end-to-end tasks in dynamic mobile environments, and on AndroidControl-Curated-Hard/Easy (Leung et al., 2025), an additional offline step-level benchmark.

Baselines. For the world model, to evaluate it, we compare five general-purpose image editors (FLUX.1 Kontext, Step1X-Edit-v1.2, LongCat-Image-Edit, Qwen-Image-Edit-2511, FireRed-Image-Edit-1.0) and the un-fine-tuned FLUX.2-Klein backbone (4B/9B), also the starting point for our Infinite-Dreamer-4B/9B; we fine-tune the latter on UI screenshot pairs with delta-text. Importantly, the un-fine-tuned backbone isolates the gain from delta-text fine-tuning, while the general-purpose editors serve as off-the-shelf alternatives. Similarly, for the GUI agent, we compare general-purpose VLMs (Claude-Sonnet-4.5, Gemini-3-Flash, MiniMax-M3, MiMo-V2.5, Kimi-K2.6), GUI-specialized GUI-Owl-1.5 (2B/4B/8B), and base-model Qwen3-VL (2B/4B/8B). Implementation details for all baselines are in Appendix A.7.

Metrics. For world-model synthesis quality, we report four VLM-judged dimensions: Accuracy (Acc.), Completeness (Comp.), Contextual alignment (Cont.), and Visual preservation (Vis.), measuring whether the change is correctly applied, contextually sensible, and leaves unedited regions intact. For the GUI agent, we evaluate from two complementary perspectives: task success rate in dynamic environments and step-level accuracy on static datasets. In AndroidWorld and MobileWorld, we use Pass@1 and Pass@3, the task success rates with one and three attempts, respectively. On the static AndroidControl-Curated (Hard/Easy) dataset, we use Type Acc (action-type accuracy), Box Acc (coordinate grounding accuracy), and Step SR (step-level success rate). Detailed evaluation protocols and world-model inference configurations are in Appendices A.6 and A.2.

Table 2: World-model synthesis quality. Single-step instruction fidelity on the AndroidControl validation set and multi-step rollout quality, both scored across four VLM-judged dimensions (Accuracy, Completeness, Contextual alignment, Visual preservation). Higher is better.

	Single-Step Fidelity				Multi-Step Rollout		
Method	**Acc.**	**Comp.**	**Cont.**	**Vis.**	**1 Step**	**5 Steps**	**10 Steps**
FLUX.1 Kontext (12B)	33.33	33.01	36.05	34.34	34.18	22.16	27.69
Step1X-Edit-v1.2 (12B)	34.27	34.11	36.85	30.60	33.96	48.95	39.61
LongCat-Image-Edit (6B)	33.79	32.66	36.34	40.65	35.86	20.89	32.69
Qwen-Image-Edit-2511 (20B)	50.30	60.50	57.44	55.52	55.94	39.44	26.15
FireRed-Image-Edit-1.0 (20B)	64.34	67.73	71.24	58.89	65.55	47.46	40.96
FLUX.2 Klein 4B	45.13	46.02	51.21	46.02	47.09	24.96	28.65
↪ Infinite-Dreamer-4B (Ours)	60.10	64.50	67.84	60.28	63.18	54.06	40.57
FLUX.2 Klein 9B	62.44	65.17	68.95	59.43	64.00	42.53	29.61
↪ Infinite-Dreamer-9B (Ours)	**68.13**	**72.27**	**75.87**	**66.55**	**70.71**	**64.21**	**48.84**



(a) Success rate by task difficulty.

(b) Distribution of trajectory lengths.

Figure 3: AndroidWorld analyses for base-4b, real-4b, and mix-4b.

4.2 World Model Quality

Single-step Synthesis Fidelity. As Table 2 shows, Infinite-Dreamer-9B leads on all four metrics, outperforming both the unfine-tuned FLUX.2 Klein 9B backbone and the strongest baseline, FireRed-Image-Edit-1.0. The 4B variant also surpasses its FLUX.2 Klein 4B counterpart, confirming that delta-text-guided fine-tuning on UI data improves instruction fidelity and layout preservation. Qualitative single-step editing comparisons against all baselines are shown in Appendix F.1.

Multi-step Rollout Stability. Table 2 shows that Infinite-Dreamer-9B maintains the highest fidelity across all rollout horizons. At 10 steps it scores 48.84, versus 40.96 for FireRed-Image-Edit-1.0 and 29.61 for the unfine-tuned FLUX.2 Klein 9B, demonstrating better stability under recursive application. Figure 4 shows a representative qualitative multi-step rollout: Infinite-Dreamer-9B faithfully reproduces the ground-truth UI transitions across a 6-step GUI interaction, preserving layout, text content, and visual coherence. Additional qualitative comparisons appear in Appendix F.2. We also discuss failure modes and limitations with examples in Appendix G.3.

4.3 GUI Agent Performance

Task Success Rate on AndroidWorld and MobileWorld. As Table 1 shows, on AndroidWorld, Infinite-Actor-8B achieves 43.82 Pass@1 and 62.07 Pass@3, improving over its base-model counterpart Qwen3-VL-8B (39.37 and 56.90) by +4.45 and +5.17 points, respectively. The gains hold at smaller scales: Infinite-Actor-4B matches the 8B base-model on Pass@1 (39.37) with half the parameters, and Infinite-Actor-2B improves over Qwen3-VL-2B by +9.05 on Pass@1 and +19.83 on Pass@3. On MobileWorld, Infinite-Actor-8B reaches 15.20 Pass@1 and 21.93 Pass@3, against 8.77 and 11.40 for Qwen3-VL-8B base-model, nearly doubling the success rate. These results confirm

Table 3: Data source ablation on AndroidWorld Pass@1 (%), using Qwen3-VL-4B as the fixed backbone. Real Data: real collected trajectories; Gen Data: multi-step imagination from the world model; Aug Data: single-frame visual augmentation.

Real Data	Gen Data	Aug Data	SR (%)
×	×	×	30.17
×	✓	✓	39.37
✓	×	×	42.67
✓	✓	×	44.83
✓	×	✓	47.41
✓	✓	✓	48.28

Table 4: World model design ablation on instruction fidelity. Both variants use FLUX.2-Klein-9B as the backbone. "w/o delta-text" indicates that the world model is conditioned only on the trigger action, completely omitting fine-grained descriptions of added or removed UI elements.

Method	Acc.	Comp.	Cont.	Vis.
Infinite-Dreamer-9B	**68.13**	**72.27**	**75.87**	**66.55**
w/o delta-text	21.56	19.72	26.14	25.57

that Infinite-Dreamer synthesis data provides effective supervision for mobile tasks. More results and analysis are provided in Appendix A.8.

Step Level Accuracy on AndroidControl-Curated (AC-Curated). Table 1 shows that Infinite-Actor achieves strong action-type accuracy (73.30 on Hard and 82.40 on Easy for 8B) but regresses in coordinate grounding (Box Acc: $77.00 \rightarrow 64.70$ on 4B Hard). The underlying box predictions are displaced rather than marginally noisy: over the $5{,}016$ Easy and $5{,}074$ Hard annotated-box steps, the median distance between predicted and annotated box centers grows from 0.06 to 0.13 annotated-box widths on Easy and from 0.18 to 0.38 on Hard, with 90th percentiles of $0.83 \rightarrow 0.88$ and $1.90 \rightarrow 4.73$. These errors are not near misses: 88% (Easy) and 97% (Hard) of the model's incorrect boxes place their center more than 50 pixels outside the annotation. Because the offline metric accepts a box only when its center falls inside the annotation, the loss concentrates on compact targets: on Hard, Box Acc is 46.7% for the smallest third of annotations against 75.4% for the largest third (base: 67.5% vs. 87.0%). Online, padded touch targets and post-action feedback absorb part of the same displacement, so an offset tap need not end an episode (Appendix G.3).

4.4 Ablation Study

Synthesis Regimes Analysis. We additionally vary the inclusion of real trajectories from OpenMobile (Cheng et al., 2026) to isolate the contribution of each data source on AndroidWorld. Adding multi-step imagination to real-only training (42.67) lifts SR to 44.83, and adding augmentation instead reaches 47.41; combining all three sources gives the best result, 48.28. These two synthesis regimes therefore contribute in distinct ways: augmentation mainly improves robustness to surface-level UI variation, while multi-step imagination expands the coverage of state transitions. More visualizations appear in Appendices G.1 and G.2.

Performance Across Task Complexities and Hard Tasks. Stratifying AndroidWorld by difficulty (Figure 3a) shows that mixed training improves Easy (+4.9) and Medium tasks (+8.3), where expanded state coverage aids exploration. On Hard tasks, however, mix-4b underperforms real-4b by -2.7 points (10.5 vs. 13.2). Hard tasks demand long, strictly ordered interactions (e.g., cross-application state transfer and precise system dialogs), and world-model fidelity decays over such horizons (Table 2), injecting causal noise that narrower real demonstrations avoid. In the Hard-task logs, 41.2% of the failures exhaust the step budget and 35.3% falsely declare completion (Appendix G.3).

Trajectory-length Analysis. Figure 3b compares AndroidWorld episode lengths by task outcome for base-4b, real-4b, and mix-4b. On successful tasks the three models occupy a similar band of short episodes, and the upper tail separates them: mix-4b's successful episodes run past 50 steps, where real-4b stops near 30 and base-4b near 25. Multi-step synthetic rollouts therefore extend the horizon

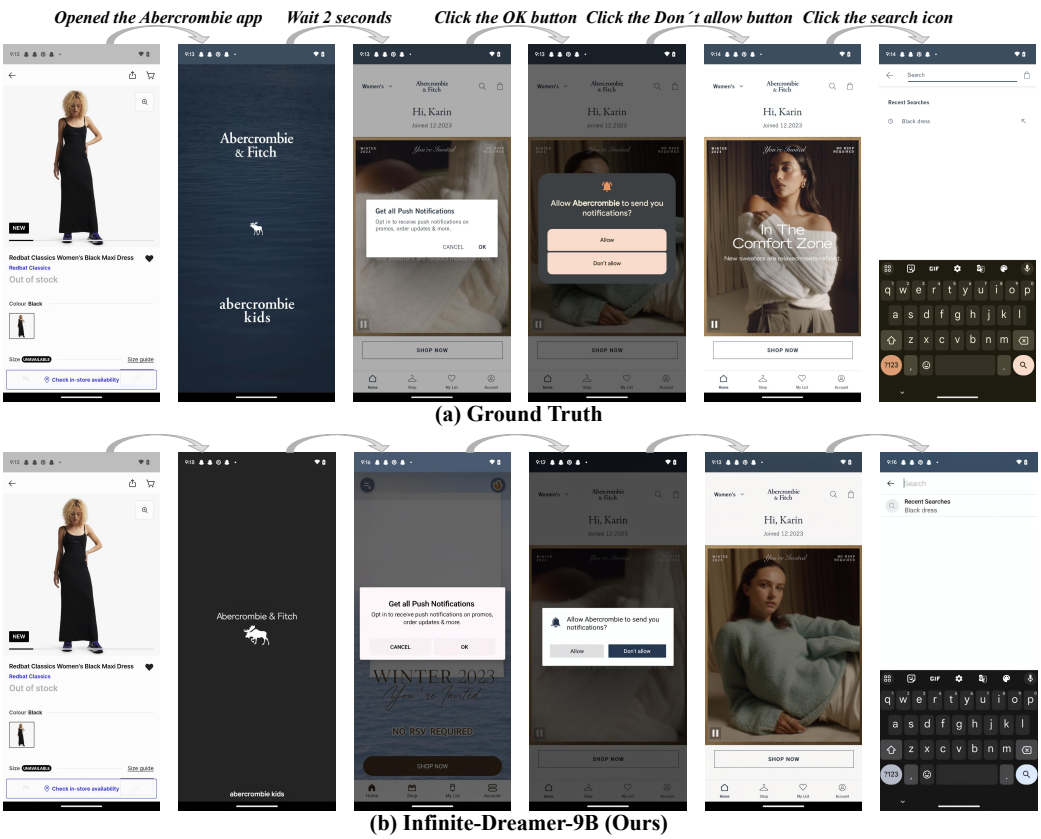


Figure 4: Qualitative synthesis comparison. (a) Ground Truth trajectory (top) versus (b) Infinite-Dreamer-9B (Ours) (bottom) over a 6-step GUI interaction sequence (from opening "Abercrombie" to clicking the search icon). Our model faithfully reproduces the real UI state transitions, maintaining layout accuracy, text fidelity, and visual coherence across all steps.

the policy can sustain. Failed episodes run longer than successful ones for every model, consistent with runs that exhaust the step budget, and mix-4b's failed distribution tracks the baselines.

Necessity of Delta-text Conditioning and Error Dynamics. Table 4 shows that removing delta-text collapses synthesis quality (Acc. drops from 68.13 to 21.56), confirming its indispensable role in transition grounding. However, this dependence exposes sensitivity to specification errors: training delta-texts come from real image pairs (E_ψ), while rollout delta-texts are authored text-only (E_ψ^{txt}). During imagination, the text-only extractor must infer the diff from a planned UI description without ground-truth visual feedback, so underspecified diffs (e.g., a missing keyboard dismissal or background dimming) can produce visual distortions that compound autoregressively (Table 2). Our VLM quality filter ($\eta = 90$) prunes such steps before training (formal error dynamics in Appendix E.2; qualitative ablation in Appendix A.8).

5 Conclusion

In this paper, we present Infinite-Dreamer, a simulation-free data synthesis framework leveraging an image-editing world model for scalable GUI agent training. By conditioning generation on structured delta-text, our framework controllably synthesizes screenshot transitions and multi-step rollouts. Extensive experiments demonstrate that this synthetic supervision substantially enhances agent generalization, yielding large task-level gains for Infinite-Actor on AndroidWorld and MobileWorld. More broadly, our work highlights a promising direction where generative world models and autonomous agents co-evolve: high-fidelity world models expand state coverage, while agents identify transitions critical for long-horizon competence.

AI USE STATEMENT

This paper uses large vision-language models as instruments inside the proposed method. Concretely, GLM-4.7v serves as the delta-text extractor that produces the conditioning descriptions (Section 3.2) and as the judge for the synthesis-quality metrics (Appendix E.1), and Qwen3.5-397B-A17B serves as the quality filter applied to generated trajectories (Appendix E). All of these components are described in the Method and in Appendix A, and every model output that enters the training data is filtered and was reviewed by the authors.

We take responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

This paper studies data synthesis for GUI agent training. All screenshots and trajectories used in the experiments come from the public AndroidControl, AndroidWorld, MobileWorld, and AndroidControl-Curated benchmarks. We collect no new human data and conduct no user studies, so no IRB approval was required. The synthesized screenshots are derived from these public sources and will be released under a research-permissive license.

Two risks deserve note. First, synthesized screenshots inherit the distribution of their source data, so the application coverage and language mix of our synthesized corpus reflect those of AndroidControl; agents trained on it may therefore generalize unevenly across regions and applications. Second, a world model that predicts GUI transitions at pixel level is dual-use, since the same capability that supports agent training can also automate harmful workflows. We restrict the study to standard mobile-assistant benchmarks and release only the synthesized screenshots together with the generation scripts, not any device-control harness.

REPRODUCIBILITY STATEMENT

We specify the full experimental configuration in the appendix so that the results can be reproduced. Appendix A lists the world-model and GUI-agent training hyperparameters and the inference configuration, and Appendix A.6 defines the evaluation metrics. Every prompt template used in the pipeline is reproduced verbatim: delta-text extraction (Appendices C.1 and C.2), single-frame augmentation (Appendix D.1), multi-step imagination (Appendix D.2), quality filtering (Appendix E), and synthesis-quality evaluation (Appendix E.1). Appendix A.4 records the random seeds, the number of runs per configuration, and the computing infrastructure. Code is available at `https://anonymous.4open.science/r/Infinite-Dreamer-B499/`. The synthesized dataset and the generation and training scripts will be released upon publication.

REFERENCES

James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ou, Juntang Zhuang, Joyce Lee, Yufei Guo, Aditya Ramesh, et al. Improving image generation with better captions, 2023. OpenAI Technical Report.

Black Forest Labs. FLUX.1: Frontier text-to-image generation model. `https://blackforestlabs.ai`, 2024.

Black Forest Labs. FLUX.2: Frontier Visual Intelligence. `https://bfl.ai/blog/flux-2`, 2025.

Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions, 2023. URL `https://arxiv.org/abs/2211.09800`.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments, 2024. URL `https://arxiv.org/abs/2402.15391`.

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, 2026a. URL https://arxiv.org/abs/2601.04035.

Yuan Cao, Dezhi Ran, Mengzhou Wu, Yuzhe Guo, Xin Chen, Ang Li, Gang Cao, Gong Zhi, Hao Yu, Linyi Li, Wei Yang, and Tao Xie. GUI-GENESIS: Automated synthesis of efficient environments with verifiable rewards for GUI agent post-training, 2026b. URL https://arxiv.org/abs/2602.14093.

Haoxuan Che, Xuanhua He, Quande Liu, Cheng Jin, and Hao Chen. Gamegen-x: Interactive open-world game video generation, 2024. URL https://arxiv.org/abs/2411.00769.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *International Conference on Learning Representations (ICLR)*, 2024.

Kanzhi Cheng, Zehao Li, Zheng Ma, Nuo Chen, Jialin Cao, Qiushi Sun, Zichen Ding, Fangzhi Xu, Hang Yan, Jiajun Chen, Anh Tuan Luu, Jianbing Zhang, Lewei Lu, and Dahua Lin. OpenMobile: Building open mobile agents with task and trajectory synthesis, 2026. URL https://arxiv.org/abs/2604.15093.

Shenyuan Gao, Siyuan Zhou, Yilun Du, Jun Zhang, and Chuang Gan. Adaworld: Learning adaptable world models with latent actions, 2025a. URL https://arxiv.org/abs/2503.18938.

Yifei Gao, Junhong Ye, Jiaqi Wang, and Jitao Sang. WebSynthesis: World-model-guided mcts for efficient webui-trajectory synthesis, 2025b. https://arxiv.org/abs/2507.04370.

Sofiya Garkot, Maksym Shamrai, Ivan Synytsia, and Mariya Hirna. Guirilla: A scalable framework for automated desktop ui exploration, 2026. URL https://arxiv.org/abs/2510.16051.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2022. URL https://arxiv.org/abs/2010.02193.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. URL https://arxiv.org/abs/2301.04104.

Yanheng He, Jiahe Jin, and Pengfei Liu. Efficient agent training for computer use, 2026. URL https://arxiv.org/abs/2505.13909.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. MobileWorld: Benchmarking autonomous mobile agents in agent-user interactive and MCP-augmented environments, 2025. URL https://arxiv.org/abs/2512.19432.

Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Müller, Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. FLUX.1 Kontext: Flow matching for in-context image generation and editing in latent space, 2025. URL https://arxiv.org/abs/2506.15742.

Ho Fai Leung, Xiaoyan Xi, and Fei Zuo. AndroidControl-Curated: Revealing the true potential of GUI agents through benchmark purification, 2025. URL https://arxiv.org/abs/2510.18488.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on UI control agents, 2024. URL https://arxiv.org/abs/2406.03679.

Musen Lin, Minghao Liu, Taoran Lu, Lichen Yuan, Yiwei Liu, Haonan Xu, Yu Miao, Yuhao Chao, and Zhaojian Li. GUI-ReWalk: Massive data generation for GUI agent via stochastic exploration and intent-aware reasoning, 2025. URL https://arxiv.org/abs/2509.15738.

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, 2025. URL https://arxiv.org/abs/2504.13936.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents, 2025. URL https://arxiv.org/abs/2405.14573.

Marc Rigter, Tarun Gupta, Agrin Hilmkil, and Chao Ma. AVID: Adapting video diffusion models to world models, 2024. URL https://arxiv.org/abs/2410.12822.

Luke Rivard, Sun Sun, Hongyu Guo, Wenhu Chen, and Yuntian Deng. Neuralos: Towards simulating operating systems via neural generative models, 2026. URL https://arxiv.org/abs/2507.08800.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. OS-Genesis: Automating GUI agent trajectory construction via reverse task synthesis, 2025. URL https://arxiv.org/abs/2412.19723.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Aoyan Li, Bo Li, Chen Dun, Chong Liu, Daoguang Zan, Fuxing Leng, Hanbin Wang, Hao Yu, Haobin Chen, Hongyi Guo, Jing Su, Jingjia Huang, Kai Shen, Kaiyu Shi, Lin Yan, Peiyao Zhao, Pengfei Liu, Qinghao Ye, Renjie Zheng, Shulin Xin, Wayne Xin Zhao, Wen Heng, Wenhao Huang, Wenqian Wang, Xiaobo Qin, Yi Lin, Youbin Wu, Zehui Chen, Zihao Wang, Baoquan Zhong, Xinchun Zhang, Xujing Li, Yuanfan Li, Zhongkai Zhao, Chengquan Jiang, Faming Wu, Haotian Zhou, Jinlin Pang, Li Han, Qi Liu, Qianli Ma, Siyao Liu, Songhua Cai, Wenqi Fu, Xin Liu, Yaohui Wang, Zhi Zhang, Bo Zhou, Guoliang Li, Jiajun Shi, Jiale Yang, Jie Tang, Li Li, Qihua Han, Taoran Lu, Woyu Lin, Xiaokang Tong, Xinyao Li, Yichi Zhang, Yu Miao, Zhengxuan Jiang, Zili Li, Ziyuan Zhao, Chenxin Li, Dehua Ma, Feng Lin, Ge Zhang, Haihua Yang, Hangyu Guo, Hongda Zhu, Jiaheng Liu, Junda Du, Kai Cai, Kuanye Li, Lichen Yuan, Meilan Han, Minchao Wang, Shuyue Guo, Tianhao Cheng, Xiaobo Ma, Xiaojun Xiao, Xiaolong Huang, Xinjie Chen, Yidi Du, Yilin Chen, Yiwen Wang, Zhaojian Li, Zhenzhu Yang, Zhiyuan Zeng, Chaolin Jin, Chen Li, Hao Chen, Haoli Chen, Jian Chen, Qinghao Zhao, and Guang Shi. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2509.02544.

Xiaofeng Wang, Zheng Zhu, Guan Huang, Boyuan Wang, Xinze Chen, and Jiwen Lu. Worlddreamer: Towards general world models for video generation via predicting masked tokens, 2024. URL https://arxiv.org/abs/2401.09985.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Sheng ming Yin, Shuai Bai, Xiao Xu, Yilei Chen, Yuxiang Chen, Zecheng Tang, Zekai Zhang, Zhengyi Wang, An Yang, Bowen Yu, Chen Cheng, Dayiheng Liu, Deqing Li, Hang Zhang, Hao Meng, Hu Wei, Jingyuan Ni, Kai Chen, Kuan Cao, Liang Peng, Lin Qu, Minggang Wu, Peng Wang, Shuting Yu, Tingkun Wen, Wensen Feng, Xiaoxiao Xu, Yi Wang, Yichang Zhang, Yongqiang Zhu, Yujia Wu, Yuxuan Cai, and Zenan Liu. Qwen-image technical report, 2025. URL https://arxiv.org/abs/2508.02324.

Mengzhou Wu, Yuzhe Guo, Yuan Cao, Haochuan Lu, Songhe Zhu, Pingzhe Qu, Xin Chen, Kang Qin, Zhongpu Wang, Xiaode Zhang, Xinyi Wang, Wei Dai, Gang Cao, Yuetang Deng, Zhi Gong, Dezhi Ran, Linyi Li, Wei Yang, and Tao Xie. UI-Oceanus: Scaling GUI agents with synthetic environmental dynamics, 2026. URL https://arxiv.org/abs/2604.02345.

Jiannan Xiang, Guangyi Liu, Yi Gu, Qiyue Gao, Yuting Ning, Yuheng Zha, Zeyu Feng, Tianhua Tao, Shibo Hao, Yemin Shi, Zhengzhong Liu, Eric P. Xing, and Zhiting Hu. Pandora: Towards general world model with natural language actions and video states, 2024. URL https://arxiv.org/abs/2406.09455.

Weimin Xiong, Shuhao Gu, Bowen Ye, Zihao Yue, Lei Li, Feifan Song, Sujian Li, and Hao Tian. Video2GUI: Synthesizing large-scale interaction trajectories for generalized GUI agent pretraining, 2026. URL https://arxiv.org/abs/2605.14747.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. AgentTrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations (ICLR)*, 2025. Spotlight.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, Yuxuan Liu, Tianyi Tang, Xiaomeng Hu, Yang Su, Qingfeng Lan, Yantao Liu, Qin Zhu, Yinger Zhang, Bowen Yu, Haiquan Zhao, Haiyang Xu, Jianxin Yang, Jiayang Cheng, Junyang Wang, Lianghao Deng, Mingfeng Xue, Tianyi Bai, Yang Fan, Yubo Ma, Yucheng Li, Zeyu Cui, Zhihai Wang, Zhihui Xie, Zhuorui Ye, An Yang, Dayiheng Liu, Jingren Zhou, and Ning Ding. Qwen-agentworld: Language world models for general agents, 2026. URL https://arxiv.org/abs/2606.24597.

A Experimental Details

This section provides detailed experimental configurations for reproducibility, organized into training settings for the world model and GUI agent, evaluation protocols, and baseline implementations.

A.1 World Model Training Settings

We build the image-editing world model on FLUX.2-Klein backbones (4B and 9B variants) and fine-tune both on paired UI screenshots with delta-text conditioning. All training configurations are summarized in Table 5.

Table 5: World model training configuration. Both 4B and 9B models share the same hyperparameters unless otherwise noted.

Configuration	**Value**
Backbone	FLUX.2-Klein (4B / 9B)
Training data	AndroidControl train split
Fine-tuning method	LoRA
LoRA rank	32
Training epochs	3
Batch size	1
Learning rate	1×10^{-4}
LR schedule	Cosine decay
Warmup steps	N/A
Optimizer	AdamW
Weight decay	0.01
Resolution	672×1536
Precision	bfloat16
Hardware	4$\times$ NVIDIA A100 80GB
Random seed	42
Software	PyTorch 2.9.1

A.2 Inference Performance of World Models

We evaluate the inference performance of the FLUX.2-Klein 4B and 9B variants as image-editing world models. The evaluation is conducted under the configuration summarized in Table 6. The inference performance results are reported in Table 7.

Table 6: Inference configuration for performance testing.

Configuration Item	Value
Image Width	672 px
Image Height	1536 px
Inference Steps	40 steps
Guidance Scale	4.0

Table 7: Inference performance comparison between FLUX.2-Klein 9B and 4B models.

Metric	Infinite-Dreamer-9B	Infinite-Dreamer-4B
Average Time per Image	57.202 s	28.779 s
Average Time per Step	1430.0 ms	719.5 ms
Throughput (Img/s)	0.01748	0.03475
Pixel Throughput (Mpx/s)	0.01804	0.03587

A.3 GUI Agent Training Settings

We fine-tune Infinite-Actor agents from Qwen3-VL-2B/4B/8B checkpoints solely on synthesized data, namely single-frame augmentations and quality-filtered multi-step imagination trajectories, without using any real trajectories. All training configurations are summarized in Table 8.

A.4 Reproducibility Details

Hyperparameter selection. We did not conduct an extensive hyperparameter search; a single setting per model was used, selected by preliminary validation on the AndroidControl validation split. The final values are listed in Tables 5 and 8; all other parameters follow the backbone defaults.

Random seeds. All data generation, training, and evaluation procedures use a fixed random seed (seed $= 42$) to ensure reproducibility.

Algorithm runs. All reported results are obtained from a single training run per model configuration; Pass@3 results are computed over three independent inference attempts, as described in Appendix A.6.

Computing infrastructure. All experiments are run on Ubuntu 22.04 LTS with Python 3.11. The world model is implemented in PyTorch 2.9.1; the GUI agent is fine-tuned with PyTorch 2.9.1 and transformers 4.57.6. The pipeline VLMs (delta-text extractor, quality filter, and synthesis-quality judge) are served via vLLM 0.16.0. GPU models and memory for each stage are listed in Tables 5 and 8.

Preprocessing pipeline. The complete data preprocessing pipeline—delta-text extraction (Appendix C.1 and C.2), single-frame augmentation (Appendix D.1), multi-step imagination (Appendix D.2), and quality filtering (Appendix E)—is fully specified by the prompt templates and configuration tables included in this appendix.

Table 8: GUI agent training configuration. All model sizes (2B/4B/8B) share the same hyperparameters.

Configuration	Value
Backbone	Qwen3-VL (2B / 4B / 8B)
Training data	Infinite-Dreamer synthesis data (aug, gen)
Fine-tuning method	Full fine-tuning
Training steps	1,000
Batch size	32
Learning rate	1.0×10^{-5}
LR schedule	Cosine decay
Warmup steps	30
Optimizer	AdamW
Weight decay	0.05
Resolution	672×1536
Precision	bfloat16
Hardware	4× NVIDIA A40 48GB
Random seed	42
Software	PyTorch 2.9.1, transformers 4.57.6

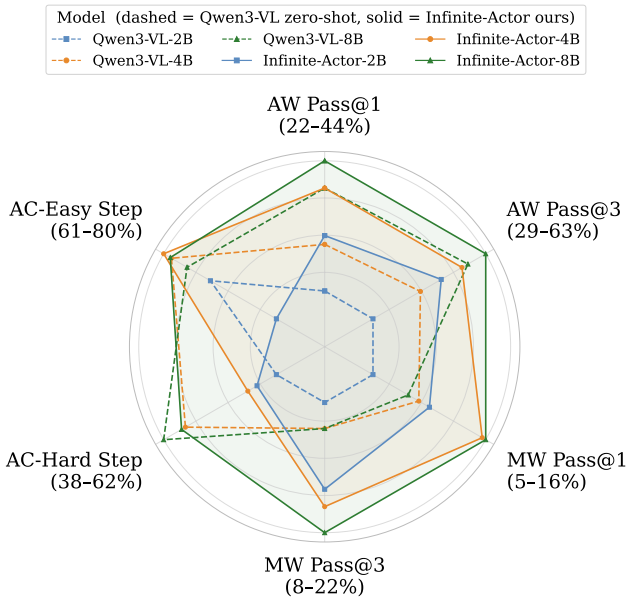


Figure 5: Performance comparison between zero-shot Qwen3-VL and Infinite-Actor agents across model scales (2B/4B/8B) on six evaluation metrics from Table 1: AW Pass@1 (AndroidWorld Pass@1), AW Pass@3, MW Pass@1 (MobileWorld Pass@1), MW Pass@3, AC-Easy Step SR (AC-Curated-Easy Step SR), and AC-Hard Step SR (AC-Curated-Hard Step SR). Infinite-Actor models consistently outperform their zero-shot counterparts on task-level benchmarks.

A.5 Synthesized Data

The novel data introduced by this paper is the Infinite-Dreamer synthesis set used to train the GUI agent (Section 3, GUI Agent Training), composed of two complementary subsets:

- **Single-frame augmentations $\mathcal{D}_{\text{aug}}$:** appearance-only variants of real screenshots (theme, color, and icon restyling) that preserve layout and text, generated by the world model under the augmentation delta-text (Appendix D.1).
- **Multi-step imagination trajectories $\mathcal{D}_{\text{gen}}$:** autoregressive rollouts produced by the App Designer and the world model (Appendix D.2), retained step-by-step after VLM quality filtering with confidence threshold $\eta = 90$ (Appendix E).

Each sample is stored as a (s_t, g, a_t) triple—a screenshot, the task instruction, and a `mobile_use` action—at resolution 672×1536. Both subsets are regenerated deterministically from the prompt templates and configuration tables included in this appendix, and the full dataset together with its generation scripts will be released under a research-permissive license upon publication.

A.6 Evaluation Protocols

Task-level benchmarks. For AndroidWorld and MobileWorld, we report Pass@1 and Pass@3 metrics following the standard evaluation protocol. Each task executes up to the maximum allowed steps. For Pass@3, we run three independent attempts and consider a task successful if any attempt completes correctly.

Step-level benchmarks. For AndroidControl-Curated (Hard/Easy), we report Type Acc (action-type accuracy), Box Acc (coordinate grounding accuracy), and Step SR (step-level success rate) computed over all annotated steps. Box Acc is computed over the steps whose ground-truth action requires a coordinate (click and long press); a predicted box counts as correct when its center falls inside the ground-truth box, after rescaling the model's 1000×1000 output to the screenshot resolution. Type Acc and Step SR are computed over all steps, and steps whose ground-truth action requires no coordinate are scored on action type alone.

A.7 Baseline Implementations

Proprietary models. We use official API endpoints for Claude-Sonnet-4.5 (Anthropic), Gemini-3-Flash (Google), MiniMax-M3, MiMo-V2.5, and Kimi-K2.6.

Open-source VLMs. For Qwen3-VL and GUI-Owl-1.5, we use publicly released checkpoints from Hugging Face with the same prompt template as Infinite-Actor agents.

Image-editing baselines. For baselines in Table 2 (FLUX.1 Kontext, Step1X-Edit-v1.2, LongCat-Image-Edit, Qwen-Image-Edit-2511, FireRed-Image-Edit-1.0), we use default inference settings from official implementations with the same delta-text prompts as Infinite-Dreamer for fair comparison.

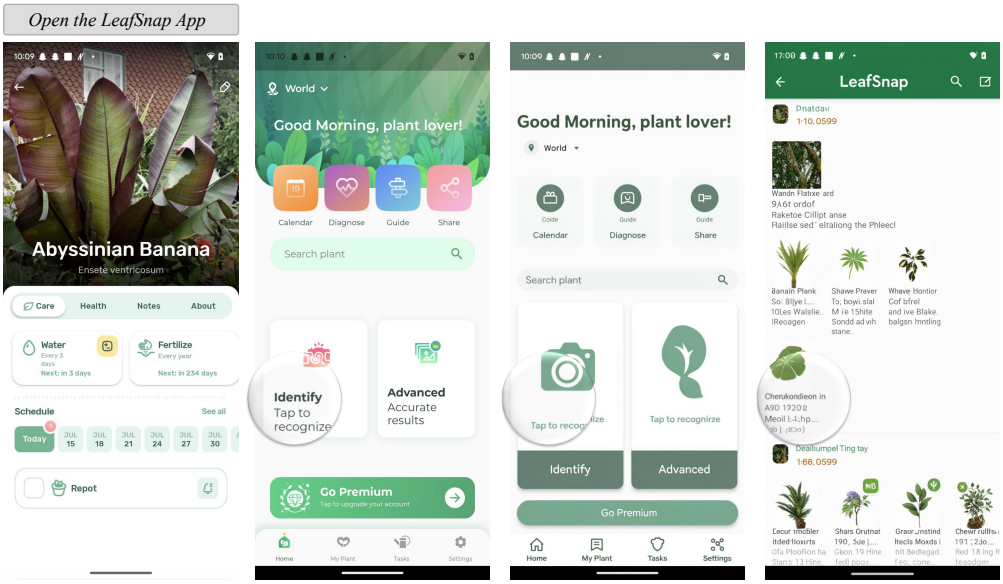
A.8 Additional Experimental Results

Qualitative Ablation: Effect of Delta-Text Conditioning. Figure 6 illustrates the qualitative effect of delta-text conditioning on a representative example. Given the source UI (leftmost) and the delta-text action "Open LeafSnap App", Infinite-Dreamer-9B (third column) closely matches the ground-truth transition (second column), whereas the variant trained without delta-text (rightmost) collapses into an unrelated screenshot that diverges substantially from the intended target UI. This confirms the quantitative finding reported in Table 4.

Comparison: Zero-Shot vs. Ours. Figure 5 provides a visual summary of the performance gap between zero-shot Qwen3-VL and Infinite-Actor across model scales (2B/4B/8B) on the six evaluation metrics reported in Table 1: AW Pass@1, AW Pass@3, MW Pass@1, MW Pass@3, AC-Easy Step SR, and AC-Hard Step SR. Infinite-Actor models consistently outperform their zero-shot counterparts on task-level benchmarks.

A.9 Conceptual Disentanglement: World Model vs. Conditional GUI Renderer

A central conceptual question is whether our image-editing model acts merely as a conditional graphic renderer rather than a genuine world model. We systematically distinguish between these



Source UI | Ground Truth | Infinite-Dreamer-9B (Ours) | w/o delta-text

Figure 6: Qualitative effect of delta-text conditioning. Given the source UI (leftmost) and the delta-text action "Open LeafSnap App", Infinite-Dreamer-9B (third) closely matches the ground-truth transition (second), whereas the variant without delta-text (rightmost, which only receives the trigger action/operation without any fine-grained descriptions of added or removed elements) collapses into an unrelated screenshot that diverges substantially from the intended target UI.

two paradigms across operational mechanisms, representations, and functional roles in agent training, summarized in Table 9.

Table 9: Conceptual and structural comparison between conventional GUI renderers and the Infinite-Dreamer image-editing world model.

Dimension	**Conventional GUI Renderer (e.g., WebKit)**	**Infinite-Dreamer World Model (f_ϕ)**
Input Conditioning	Exhaustive scene graph (DOM tree, precise (x, y, w, h) coordinates, CSS style properties, typography).	Source screenshot s_t and abstract delta-text Δ_t without spatial coordinates or component sizes.
Spatial Physics & Layout	Explicit, hard-coded geometry calculations and constraint-solver passes.	Implicit visual physics: learned neural inference of layout reflow, occlusions, background dimming, and responsive alignment.
State Persistence	Redraws the canvas or invalidates explicit tree nodes; each render pass is independent of the previous frame.	Causal state persistence: conditioning on s_t keeps unaffected regions (status bars, background cards) visually consistent with the previous frame.
Affordance Modeling	Purely passive visual output for human display; oblivious to action viability.	Active affordance preservation: synthesizes interactive visual hot-spots enabling subsequent policy execution.
Operational Role	Open-loop visual presentation terminal.	Closed-loop state transition simulator $P(s_{t+1} \mid s_t, a_t)$ for trajectory imagination.

1. Coordinate-Free Spatial Generalization. Conventional graphical rendering pipelines (e.g., Blink, Gecko, or the Android ViewServer) are deterministic rasterization engines. They require complete, unambiguous specifications of every visual element, including exact screen coordinates, width, height, margin, padding, border radius, and color codes. In stark contrast, delta-text conditioning provides only high-level semantic intent (e.g., "Added: search results dropdown; Removed: search suggestion pill"). The image-editing model f_ϕ must infer where and how to render these elements: it predicts valid spatial coordinates, typography, responsive element reflows, and background dimming purely from visual context. This requires internalizing the underlying spatial grammar and visual physics of digital software.

2. Causal State Persistence and Invariance. A critical requirement of an environmental world model is state persistence: when an action affects a localized region of the screen, the rest of the environment must remain causally invariant. A conventional renderer treats every render pass as an evaluation of the full DOM. In contrast, f_ϕ conditions on the prior visual observation s_t and updates only causally relevant regions, so UI areas unrelated to the action (e.g., device status bars, system navigation icons, background cards) stay visually consistent with the previous frame.

3. Hierarchical Decoupling in World Modeling. Rather than conflating high-level task logic with low-level pixel synthesis, our framework adopts a hierarchical world modeling formulation:

$$T = T_{\text{vis}} \circ T_{\text{sem}}, \tag{9}$$

where $T_{\text{sem}} : (s_t, a_t) \mapsto \Delta_t$ models the symbolic transition dynamics (governed by the App Designer G_ω), and $T_{\text{vis}} : (s_t, \Delta_t) \mapsto s_{t+1}$ models the visual dynamics (governed by f_ϕ). This hierarchical decoupling mirrors standard world-model architectures in robotics and game simulation, where high-level semantic transition intent is explicitly separated from continuous visual realization to prevent catastrophic visual hallucinations.

B GUI Agent Architecture

This section describes the architecture of our mobile GUI agent. The agent interacts with an Android device by observing screenshots, selecting actions from a unified action space (Appendix B.1), and executing them via ADB (Android Debug Bridge). At each step it receives the current screenshot, the user's goal, and a running task-progress history, selects a single action, and appends the new screenshot together with an updated task-progress entry to the trajectory for the next step; no separate outcome-summarization phase is required.

B.1 Action Space Definition

We define a unified action space for mobile GUI interaction that covers common touchscreen operations. Table 10 summarizes all 15 action types and their associated parameters.

Each action is invoked via the `mobile_use` function with a required `action_type` field and action-specific optional parameters. At deployment time the agent acts through a compact `mobile_use` tool-call interface (Appendix B.3) with eight actions; several raw actions are renamed or merged into this vocabulary—for example `input_text` as `type`, `navigate_home`/`navigate_back`/`keyboard_enter` as `system_button`, `scroll`/`drag_and_drop` as `swipe`, and `status` as `terminate`—so the policy emits a single JSON tool call per step.

B.2 Action-Selection Decision Loop

At each step the agent receives the current screenshot, the user's goal, and a task-progress string summarizing the actions taken so far. The vision-language model outputs a brief Thought explaining its reasoning, a short Action description, and a single JSON `mobile_use` tool call specifying the action type and parameters. After parsing the model output, coordinates are normalized from the model's input resolution to the device's native resolution; the action is then executed on the device via ADB, and the agent waits briefly for the UI to stabilize. The newly captured screenshot together with an updated task-progress entry is appended to the trajectory and provided as context for the next

Table 10: Action space for mobile GUI agents. Each action type is listed with its required and optional parameters. Coordinates are specified as (x, y) pixel positions on a 1000×1000 screen.

Action Type	Parameters	Description
Task Control		
`status`	`goal_status`^{R}	Report task completion. `goal_status` $\in$ {`complete`, `infeasible`}.
`answer`	`text`^{R}	Provide a textual answer to a query.
`wait`	—	Wait for the UI to update or an app to load.
Touch Interactions		
`click`	`coordinate`^{R}	Single tap at (x, y).
`double_tap`	`coordinate`^{R}	Double tap at (x, y).
`long_press`	`coordinate`^{R}	Long press at (x, y) to trigger context menus.
`drag_and_drop`	`coordinate`^{R}, `coordinate2`^{R}	Drag from (x_1, y_1) to (x_2, y_2).
Text Input		
`input_text`	`text`^{R}, `coordinate`^{O}	Type text; optionally tap `coordinate` to focus first.
`keyboard_enter`	—	Press the Enter/Return key.
`keyboard_delete`	—	Press the Backspace/Delete key.
Navigation		
`navigate_home`	—	Press the Home button.
`navigate_back`	—	Press the Back button.
`open_app`	`app_name`^{R}	Launch an application by name.
Scrolling & Swiping		
`scroll`	`direction`^{R}	Scroll the page. `direction` $\in$ {`up`, `down`, `left`, `right`}.
`swipe`	`direction`^{R}, `coordinate`^{O}	Swipe gesture; optionally from a specific point.

^{R}Required parameter. ^{O}Optional parameter.

action-selection step. This closed loop lets the agent maintain situational awareness across multi-step tasks directly from its action history.

B.3 Action-Selection Prompt Template

The complete action-selection prompt is shown below. It defines the `mobile_use` tool with eight compact actions (`click`, `long_press`, `swipe`, `type`, `answer`, `system_button`, `wait`, `terminate`), fixes the output order as Thought → Action → `<tool_call>`, and injects the user query {{ `query` }}, the running task progress {{ `task_progress` }}, and a `<image>` token for the current screenshot.

Action-Selection Prompt

```
# Tools

You may call one or more functions to assist with the user query.
```

```
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "mobile_use", "description": "Use a
touchscreen to interact with a mobile device, and take screenshots.\n* This is an
interface to a mobile device with touchscreen. You can perform actions like
clicking, typing, swiping, etc.\n* Some applications may take time to start or
process actions, so you may need to wait and take successive screenshots to see the
results of your actions.\n* The screen's resolution is 999x999.\n* Make sure to
click any buttons, links, icons, etc with the cursor tip in the center of the
element. Don't click boxes on their edges unless asked.", "parameters":
{"properties": {"action": {"description": "The action to perform. The available
actions are:\n* `click`: Click the point on the screen with coordinate (x, y).\n*
`long_press`: Press the point on the screen with coordinate (x, y) for specified
seconds.\n* `swipe`: Swipe from the starting point with coordinate (x, y) to the
end point with coordinates2 (x2, y2).\n* `type`: Input the specified text into the
activated input box.\n* `answer`: Output the answer.\n* `system_button`: Press the
system button.\n* `wait`: Wait specified seconds for the change to happen.\n*
`terminate`: Terminate the current task and report its completion status.", "enum":
["click", "long_press", "swipe", "type", "answer", "system_button", "wait",
"terminate"], "type": "string"}, "coordinate": {"description": "(x, y): The x
(pixels from the left edge) and y (pixels from the top edge) coordinates to move
the mouse to. The coordinates should be values from 0 to 999. Required only by
`action=click`, `action=long_press`, and `action=swipe`.", "type": "array"},
"coordinate2": {"description": "(x, y): The x (pixels from the left edge) and y
(pixels from the top edge) coordinates to move the mouse to. Required only by
`action=swipe`.", "type": "array"}, "text": {"description": "Required only by
`action=type` and `action=answer`.", "type": "string"}, "time": {"description":
"The seconds to wait. Required only by `action=long_press` and `action=wait`.",
"type": "number"}, "button": {"description": "Back means returning to the previous
interface, Home means returning to the desktop, Menu means opening the application
background menu, and Enter means pressing the enter. Required only by
`action=system_button`", "enum": ["Back", "Home", "Menu", "Enter"], "type":
"string"}, "status": {"description": "The status of the task. Required only by
```

```
`action=terminate`.", "type": "string", "enum": ["success",
↪  "failure"]}},
"required": ["action"], "type": "object"}}}
</tools>

For each function call, return a json object with function name and
↪  arguments
within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

# Response format

Response format for every step:
1) Thought: one concise sentence explaining the next move (no
↪  multi-step
reasoning).
2) Action: a short imperative describing what to do in the UI.
3) A single <tool_call>...</tool_call> block containing only the JSON:
↪  {"name":
<function-name>, "arguments": <args-json-object>}.

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

The user query: {{ query }}.
Task progress (You have done the following operation on the current
↪  device): {{
task_progress }}.
<image>
```

C Delta-Text Extraction Templates

We extract a structured delta-text Δ_t for each screenshot transition with one of two templates, chosen by the downstream use of the resulting data. Both produce the same core fields—a trigger operation, added elements, reduced (removed) elements, and the elements required for the next operation—but differ in their inputs and in how much of the surrounding UI they re-state. The image-in variant consumes the before/after screenshots and is used to build the world model's training set, whereas the text-in variant consumes textual UI-element descriptions and is used to build the agent's imagined training data. The full templates are included below.

C.1 Image-In Extraction for World-Model Training

Given two consecutive screenshots (`<image><image>`) together with the current action {`curr_action`} and the next action {`next_action`}, this template asks the vision-language model to describe how the operation changes the page. In addition to the core delta fields it re-states the current and next page elements, yielding six output fields. The resulting delta-texts are paired with the screenshot pairs to construct the training set of the image-editing world model (see the Method section).

Image-In Delta-Text Extraction Prompt

```
<image><image>### Task Description
You are an expert in UI analysis. Please analyze the two given mobile
↪  GUI pages and
describe how a user operation causes the page to change from one state
↪  to another.
```

```
At the same time, identify and describe the UI elements required for the next
operation.

### Possible user operations and corresponding explanations
Possible user operations are represented as JSON dictionaries with an `action_type`
key. The other fields in the action dictionary are unique for each `action_type`
and  store parameter values for that action (e.g., "open_app" actions have an
"app_name" key, but no "x" or "y" keys). The `action_type` field will have one of
the following values:
* `click`: click at a specific point on the screen using the coordinates found in
the `'x'` and `'y'` fields of the action.
* `long_press`: long press at a specific point on the screen using the coordinates
found in the `'x'` and `'y'` fields of the action.
* `scroll`: scroll in a specific direction set in the `'direction'` field (one of
'up', 'down', 'left', or 'right').
* `open_app`: open an app with the name stored in the `'app_name'` field. This name
should map to an activity name which can be used to open the app using adb.
* `input_text`: types the text in the `'text'` field. It does not perform a click
or press enter at the end.
* `navigate_home`: go to the home screen.
* `navigate_back`: go back to the previous screen.
* `wait`: wait a set number of seconds. This is used when no action should be taken
other than to wait for something on screen (e.g., a loading bar).
Note: When it comes to position coordinates, please use a normalized coordinate
system between 0 and 1. Keep the coordinate values to four decimal places. The
coordinate origin is in the upper left corner of the image, with the X-axis
pointing to the right and the Y-axis pointing downward.

### Current user operation
The operation performed by the user on the current page is: {curr_action}

### Next user operation
The operation performed by the user on the next page is: {next_action}

### Task
Please describe in detail the page changes caused by this operation based on the
above information. If there are text elements on the GUI page, clearly indicate
what text may appear. If there are image elements on the GUI page, clearly describe
what images may appear. Please include the following:
1.Trigger Point of Change: Clearly indicate what operation of the user on the
current page triggered the page changes.
2.Elements on the Current Page: Describe in detail the UI elements of the current
page.
```

```
3.Elements on the Next Page: Describe in detail the UI elements of the
↪  next page.
4.Added Elements on the Page: Describe in detail which UI elements
↪  have been added
to the screen from Page 1 to Page 2. If there are no added UI
↪  elements, please
indicate so.
5.Reduced Elements on the Page: Describe in detail which UI elements
↪  have been
reduced on the screen from Page 1 to Page 2. If there are no reduced
↪  UI elements,
please indicate so.
6.Elements for Next Operation: Describe in detail the UI elements
↪  required for the
operation performed by the user on the next page and clearly indicate
↪  the position
of this UI element (e.g., {{\"index\":\"x,y\"}}). If the operation
↪  performed by the
user on the next page does not have the required UI elements, please
↪  still maintain
the structured output.

### Output format
Please output your analysis in a structured manner using the following
↪  title:
- ** Trigger operation **: [descriptions]
- ** Current elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Next elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Added elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Reduced elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Elements for next operation **:
{{\"coordinate\":\"x,y\",\"descriptions\":\"[elements
↪  descriptions]\"}}
```

C.2 TEXT-IN EXTRACTION FOR TRAJECTORY IMAGINATION

Given textual descriptions $\{\texttt{ui_elements}\}$ of the two consecutive pages (no images), this template identifies the trigger operation and enumerates the added, reduced, and next-operation elements, yielding the four core delta fields. It is applied to the UI-element descriptions designed by the App Designer (Appendix D.2) to obtain the delta-texts $\Delta_{1:T}$ that condition the world-model rollout during multi-step imagination, thereby constructing the agent's synthesized training data.

Text-In Delta-Text Extraction Prompt

```
You are an expert in UI analysis. Please analyze the UI elements
↪  descriptons on the
```

```
two given mobile GUI pages and describe how the first operation causes
↪  the GUI page
to change from one state to another. At the same time, identify and
↪  describe the UI
elements required for the second operation.

### Possible operations and corresponding explanations
Possible operations are represented as JSON dictionaries with an
↪  `action_type` key.
Other fields are **action-specific** and store parameter values
↪  required by that
action. Fields not relevant to a given `action_type` **must not
↪  appear** in the
action dictionary. The `action_type` field will have **one of the
↪  following
values**:

#### `status`
Reports the final status of the task.
This action **ends the interaction**.
- **Fields**
  - `goal_status`: one of
    - `"complete"` { the task has been successfully completed
    - `"infeasible"` { the task cannot be completed

#### `answer`
Returns a textual response to the user (used when no further UI
↪  interaction is
needed).
- **Fields**
  - `text`: the answer content

#### `click`
Clicks at a specific point on the screen.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates

#### `double_tap`
Performs a double tap at a specific point on the screen.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates

#### `long_press`
Performs a long press at a specific point on the screen.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates

#### `input_text`
Types the provided text at the given location.
This action **does not** automatically press Enter.
- **Fields**
  - `text`: text to input
  - `coordinate`: `x,y` normalized screen coordinates

#### `keyboard_enter`
Simulates pressing the Enter / Return key on the keyboard.
- **Fields**
  - *(none)*

#### `scroll`
Scrolls the screen in a given direction.
- **Fields**
  - `direction`: one of `"up"`, `"down"`, `"left"`, `"right"`

#### `swipe`
```

```
Performs a swipe gesture starting from a specific point in a given direction.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates
  - `direction`: one of `"up"`, `"down"`, `"left"`, `"right"`

#### `open_app`
Opens an application.
- **Fields**
  - `app_name`: name of the app to open (must map to an activity that can be
  launched via adb)

#### `navigate_home`
Navigates to the home screen.
- **Fields**
  - *(none)*

#### `navigate_back`
Navigates back to the previous screen.
- **Fields**
  - *(none)*

#### `wait`
Waits without performing any interaction.
Used when the system needs time to update (e.g., loading, transitions).
- **Fields**
  - *(none)*

### Coordinate system note
- All screen positions use a **normalized coordinate system**:
  - Values range from **0 to 1**
  - Format: `x,y`
- Coordinates must be rounded to **four decimal places**
- Origin `(0, 0)` is the **upper-left corner** of the screen
- X-axis increases to the **right**
- Y-axis increases **downward**

### UI Elements Descriptons on the GUI Page
{ui_elements}

### UI Element Changes
Please describe in detail the UI element changes caused by the first operation
based on the above information. If there are text elements on the GUI page, clearly
indicate what text may appear. If there are image elements on the GUI page, clearly
describe what images may appear. Please complete the following task:
1. Trigger Point of Change: Clearly indicate what the first operation on the first
GUI page triggered the GUI page changes.
2. Added UI Elements: Describe in detail which UI elements have been added from GUI
Page 1 to GUI Page 2. If there are no added UI elements, please indicate so.
3. Reduced UI Elements: Describe in detail which UI elements have been reduced from
GUI Page 1 to GUI Page 2. If there are no reduced UI elements, please indicate so.
4. Elements for Second Operation: Describe in detail the UI elements required for
the operation on the GUI page 2 and clearly indicate the position of this UI
```

```
element (e.g., {{\"coordinate\":\"x,y\"}}). If the operation on the next page does
not have the required UI elements, please still maintain the structured output.

### Output format
Please output your answer in a structured manner using the following title:
- ** Trigger operation **: [descriptions]
- ** Added elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Reduced elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Elements for next operation **:
{{\"coordinate\":\"x,y\",\"descriptions\":\"[elements descriptions]\"}}
```

D TRAJECTORY SYNTHESIS TEMPLATES

The two synthesis regimes of the Method section are each driven by a prompt template: single-frame appearance-only augmentation and multi-step imagination with an App Designer. Both are included below.

D.1 SINGLE-FRAME AUGMENTATION PROMPT

Given a single screenshot, this template generates 3–5 appearance-only editing instructions that preserve the layout pixel-for-pixel. Each instruction draws from a distinct category—Color, Control Style, Icon, Background, or Theme—and alters only the visual skin (recoloring, icon or control restyling, background or theme replacement), leaving all text content and structure unchanged. Because only the visual skin is altered, the paired task instruction remains valid as-is.

Single-Frame Augmentation Prompt

```
You are a UI instruction generator for structure-consistent, appearance-only image
editing.

INPUT: A mobile GUI screenshot.
OUTPUT: Exactly 3 to 5 editing instructions. Each instruction must be a single,
self-contained line.

GLOBAL APPEARANCE-ONLY RULE:
All instructions must alter only the visual skin of the UI. The underlying
skeleton|layout, spacing, size, and position of every component|MUST remain
pixel-perfect to the original image.

TEXT CONTENT PRESERVATION RULE:
All displayed text (titles, labels, buttons, placeholders, etc.) must remain
exactly as shown. You may only change text color, font, or weight. The textual
string itself must never be altered.
```

```
ALLOWED VISUAL TRANSFORMATIONS:
- Recoloring any UI region or element.
- Replacing icons or images with new ones that share the same semantic
↪  purpose.
- Applying a new visual style to controls (e.g., changing a switch's
↪  appearance).
- Replacing the background layer.
- Applying an overall theme (e.g., Dark Mode).

FORBIDDEN STRUCTURAL CHANGES (do not generate instructions that
↪  imply):
- Adding, removing, moving, or resizing ANY element.
- Changing the text content.
- Altering margins, padding, or alignment.

ANALYSIS & DESCRIPTION RULES (for generating precise {placeholders}):
1. **Specificity:** When referring to a UI element, always use a
↪  distinguishing
feature (e.g., \the blue ‘Continue’ button at the bottom" not just
↪  \the button").
2. **Style Hints for Icons:** When replacing an icon, include a brief
↪  style
descriptor to ensure visual coherence (e.g., \a flat, minimalist
↪  calendar icon" not
just \a calendar icon").
3. **Color Descriptions:** Be specific. Instead of just \blue", use a
↪  concrete
value or vivid description (e.g., \#2C3E50", \a deep navy blue", or \a
↪  vibrant
coral gradient").

INSTRUCTION DIVERSITY RULE:
- Generate exactly 3, 4, or 5 instructions.
- Each instruction MUST use a DIFFERENT category from the list below.
- Prefer instructions that create a meaningful visual contrast (e.g.,
↪  pairing a
color change with a style change) over superficial ones.

AVAILABLE CATEGORIES AND EXACT TEMPLATES (replace only
↪  {placeholders}):

[Color]
Change the color of {specific UI element with location/feature} to
↪  {specific color
value/description}.

[Control Style]
Change the style of {specific UI control with location/feature} to
↪  {target style
description}.

[Icon]
Replace {original icon description} in {specific location} with {new
↪  icon
description including style}.

[Background]
Replace the background with {new background description}.

[Theme]
Convert the UI to {target mode/theme}.

OUTPUT FORMAT RULES:
- One instruction per line. No markdown, no numbering, no bullet
↪  points.
```

```
- Separate instructions with exactly one blank line.
- Do NOT wrap the output in any way (no code blocks, no quotes).
- The output must contain only the instruction lines and the blank
↪  lines between
them.

EXAMPLE OUTPUT:
Change the color of the gradient header background to a solid deep
↪  indigo #1A1A2E.

Change the style of the prominent 'Get Started' button at the bottom
↪  to a
pill-shaped button with a soft shadow.

Replace the generic user avatar in the top-right profile card with a
↪  minimalist,
abstract geometric portrait icon.

Convert the UI to a high-contrast Dark Mode.

Now, based on the provided GUI image, generate the
↪  instructions.<image>
```

D.2 Multi-Step Imagination (App Designer) Prompt

Given a task, a difficulty level, a target step count, and the initial page's UI elements, the App Designer imagines a coherent sequence of GUI pages and the operation on each. It outputs, per step, the designed UI elements, the step goal, and an operation drawn from the same action vocabulary as the text-in delta-text template, terminating with a `status: complete` action. The delta-texts conditioning the world-model rollout are then extracted from consecutive designed UI-element descriptions (Appendix C.2).

Multi-Step Imagination (App Designer) Prompt

```
You are a UI design expert. Please give full play to your imagination
↪  and
creativity. Given a task that can be completed on a mobile
↪  phone:<{task}>, with a
difficulty of <{difficulty}> and a number of steps of <{num_steps}>,
↪  design a
series of coherent, reasonable and rich GUI pages to simulate the
↪  process of
completing this task, and describe in detail the operations performed
↪  during the
process of completing this task.

### Possible operations and corresponding explanations
Possible operations are represented as JSON dictionaries with an
↪  `action_type` key.
Other fields are **action-specific** and store parameter values
↪  required by that
action. Fields not relevant to a given `action_type` **must not
↪  appear** in the
action dictionary. The `action_type` field will have **one of the
↪  following
values**:

#### `status`
Reports the final status of the task.
This action **ends the interaction**.
- **Fields**
  - `goal_status`: one of
    - `"complete"` { the task has been successfully completed
```

```
    - `"infeasible"` { the task cannot be completed

#### `answer`
Returns a textual response to the user (used when no further UI
↪  interaction is
needed).
- **Fields**
  - `text`: the answer content

#### `click`
Clicks at a specific point on the screen.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates

#### `double_tap`
Performs a double tap at a specific point on the screen.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates

#### `long_press`
Performs a long press at a specific point on the screen.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates

#### `input_text`
Types the provided text at the given location.
This action **does not** automatically press Enter.
- **Fields**
  - `text`: text to input
  - `coordinate`: `x,y` normalized screen coordinates

#### `keyboard_enter`
Simulates pressing the Enter / Return key on the keyboard.
- **Fields**
  - *(none)*

#### `scroll`
Scrolls the screen in a given direction.
- **Fields**
  - `direction`: one of `"up"`, `"down"`, `"left"`, `"right"`

#### `swipe`
Performs a swipe gesture starting from a specific point in a given
↪  direction.
- **Fields**
  - `coordinate`: `x,y` normalized screen coordinates
  - `direction`: one of `"up"`, `"down"`, `"left"`, `"right"`

#### `open_app`
Opens an application.
- **Fields**
  - `app_name`: name of the app to open (must map to an activity that
  ↪  can be
  launched via adb)

#### `navigate_home`
Navigates to the home screen.
- **Fields**
  - *(none)*

#### `navigate_back`
Navigates back to the previous screen.
- **Fields**
  - *(none)*
```

```
#### `wait`
Waits without performing any interaction.
Used when the system needs time to update (e.g., loading,
↪  transitions).
- **Fields**
  - *(none)*

### Coordinate system note
- All screen positions use a **normalized coordinate system**:
  - Values range from **0 to 1**
  - Format: `x,y`
- Coordinates must be rounded to **four decimal places**
- Origin `(0, 0)` is the **upper-left corner** of the screen
- X-axis increases to the **right**
- Y-axis increases **downward**

### Initial GUI page
## Step 0
{init_ui_elements}

### GUI page design and task planning
Based on the information above, You need to analyze the given task and
↪  plan the
operation steps for the user to complete this task. At the same time,
↪  exert your
imagination and creativity to design a series of coherent, reasonable
↪  and rich GUI
pages. Then, describe in detail the operations performed by the user
↪  during the
process of completing this task and the GUI pages you designed. If
↪  there are text
elements on the GUI page, clearly indicate what text may appear. If
↪  there are image
elements on the GUI page, clearly describe what images may appear.
↪  Please complete
the following task:
## Step id
1. UI Elements: Describe in detail all the UI elements on the GUI page
↪  you designed
from top to bottom and from left to right.
2. Step Goals: Describe in detail the goals you have planned for the
↪  current step
to complete the given task.
3. Operation: Clearly indicate the operations you have planned for the
↪  current step
to complete the given task (e.g., {{\"action_type\": \"click\",
↪  \"coordinate\":
\"x,y\"}}).

### Output format
Please output your answer in a structured manner using the following
↪  title:
## Step 0
{init_ui_elements}
- ** Step Goals **: [goals]
- ** Operation **: [operation] If the task is completed, then here is
{{\"action_type\": \"status\", \"goal_status\": \"complete\"}}.

## Step 1
- ** UI Elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Step Goals **: [goals]
```

```
- ** Operation **: [operation] If the task is completed, then here is
{{\"action_type\": \"status\", \"goal_status\": \"complete\"}}.

## Step 2
- ** UI Elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Step Goals **: [goals]
- ** Operation **: [operation] If the task is completed, then here is
{{\"action_type\": \"status\", \"goal_status\": \"complete\"}}.

continue...

## Step N
- ** UI Elements **:
1. [descriptions 1]
2. [descriptions 2]
3. [descriptions 3]
continue...
- ** Step Goals **: Task completed.
- ** Operation **: {{\"action_type\": \"status\", \"goal_status\":
↪  \"complete\"}}.
```

E Quality Filtering Details

To retain only trajectories whose synthesized screenshots are actionable, we apply a VLM-based quality filter. For each generated step the VLM receives the task goal, the task progress, the intended step action, a reference action type, and the synthesized screenshot, and must emit a `mobile_use` tool call of that same action type together with a confidence score from 0 to 100 reflecting how clearly the screenshot supports executing the action. Steps with low confidence—screenshots that lack the UI elements needed to carry out the reference action—are discarded, and trajectories containing such steps are removed from the training set. We set the threshold to $\eta = 90$ (on a 0–100 scale), discarding steps whose confidence falls below this value. The full filter template is included below.

Quality Filtering Prompt

```
You are an expert mobile UI analyst. Your task is to evaluate whether
↪  a generated
GUI screenshot contains sufficient and accurate information for an
↪  agent to execute
a specific action.

# ROLE & OBJECTIVE

Given:
1. A task objective (high-level goal)
2. Task progress (what has been completed so far)
3. A step action description (what the user should do next)
4. A reference action type (the TYPE of action that must be taken,
↪  e.g., click,
type, swipe, terminate)
5. A generated GUI screenshot

You must:
1. Analyze the screenshot to understand the current UI state
2. Determine the specific parameters for the reference action type
↪  based on the
screenshot
3. Output a tool_call with the SAME action type as the reference
```

```
4. Provide a confidence score (0-100) indicating how well the
↪  screenshot supports
the action

# OUTPUT FORMAT

Your response must include:

1. **Analysis**: Brief analysis of the screenshot and whether it
↪  supports the
reference action type
2. **Predicted Action**: A tool_call in XML format with the SAME
↪  action type as
provided in the reference
3. **Confidence**: A confidence score (0-100) based on:
   - How clear the UI elements are
   - Whether the screenshot matches the expected state for this action
   - How well the UI supports executing the action

# TOOL CALL FORMAT

The tool_call must be in this exact XML format:
<tool_call>
{"name": "mobile_use", "arguments": {"action": "ACTION_TYPE",
↪  ...other_fields...}}
</tool_call>

Where ACTION_TYPE can be: click, long_press, swipe, type, answer,
↪  system_button,
wait, terminate

For click/long_press: include "coordinate": [x, y]
For swipe: include "coordinate": [x1, y1], "coordinate2": [x2, y2]
For type: include "text": "..."
For system_button: include "button": "Back|Home|Menu|Enter"
For wait: include "time": seconds
For terminate: include "status": "success|failure"

# CONFIDENCE SCORING GUIDELINES

- **90-100**: Screenshot clearly shows all necessary UI elements,
↪  agent can
confidently execute the action
- **70-89**: Screenshot shows most necessary elements, minor ambiguity
↪  but action
is clear
- **50-69**: Screenshot shows some relevant elements but lacks clarity
↪  or has
missing information
- **30-49**: Screenshot is partially relevant but significant
↪  information is
missing
- **0-29**: Screenshot does not contain sufficient information or is
↪  irrelevant to
the task

# CRITICAL RULES

- You MUST output a tool_call with the SAME action type as the
↪  reference action
type provided
- DO NOT change the action type - if reference says "click", your
↪  output must be
"click"
```

```
- Determine the specific parameters (coordinates, text, etc.) based on
↪  the
screenshot content
- The confidence score reflects how well the screenshot supports the
↪  action
- Be realistic about what information is visible in the screenshot

# TASK OBJECTIVE
{task_goal}

# TASK PROGRESS
{task_progress}

# CURRENT STEP ACTION
{step_action}

# REFERENCE ACTION TYPE
{action_type}

# SCREENSHOT
<image>

---

Based on the screenshot above, analyze whether it contains sufficient
↪  information
for the agent to execute the described action. Provide:

1. **Analysis**: Is the screenshot appropriate for this step? Does it
↪  show the
necessary UI elements?
2. **Predicted Action**: Output a tool_call with action type
↪  "{action_type}" -
determine the specific parameters (coordinates, text, etc.) based on
↪  the
screenshot.
3. **Confidence**: How confident are you (0-100) that the predicted
↪  action is
correct?

Output format:
**Analysis**: [Your analysis here]

**Predicted Action**:
<tool_call>
{{"name": "mobile_use", "arguments": {{...}}}}
</tool_call>

**Confidence**: [0-100]
```

E.1 Synthesis-Quality Evaluation Template

World-model synthesis quality (Table 2) is judged by GLM-4.7v with the template below. Given a before screenshot, an after (generated) screenshot, a ground-truth after screenshot, and the editing instruction, the judge returns four 0–10 scores. We report these as Acc. (Accuracy), Comp. (Completeness), Cont. (Contextual alignment), and Vis. (Visual preservation) in the main table, matching its column headers.

Synthesis-Quality Evaluation Prompt

```
<image><image><image>You are an expert in evaluating the performance
↪  of a mobile
```

```
emulator. The mobile emulator is designed to navigate the UI change based on human
instruction. Given a series of instructions and the corresponding screenshots
before and after the action, your task is to assess how accurately the emulator has
followed the instructions.

### Input Format
- 'before' screenshot: The screenshot of the app's UI before the action is taken.
- 'after' screenshot: The screenshot of the app's UI after the action is taken.
- 'ground turth' screenshot: The turth screenshot of the app's UI after the action
is taken.
- instructions: A natural language description of the action that should be
performed on the app's UI before the action is taken.

### Input
- 'before' screenshot: The first screenshot.
- 'after' screenshot: The second screenshot.
- ''ground turth' screenshot': The third screenshot.
- instructions: {instructions_text}

### Task
Based on the information above, for each instruction, compare the 'before' and
'after' screenshots to determine if the action described in the instruction has
been correctly executed. Consider the following criteria in your evaluation:
1. Accuracy: Does the 'after' screenshot reflect the changes specified in the
instruction?
2. Completeness: Are all aspects of the instruction addressed in the 'after'
screenshot?
3. Contextual Understanding: Does the change make sense within the context of the
app's UI and functionality?
4. Visual Consistency: Are there any unintended changes in the 'after' screenshot
that were not part of the instruction?
Provide a detailed evaluation for each instruction, highlighting any discrepancies
or errors observed. Conclude with an overall assessment of the emulator's
performance in following the instructions. Please include the following:
** Reasons **: A reasoned analysis of the emulator's performance.
** Accuracy Metrics **: Specific metrics or scores that quantify the accuracy of
the emulator's actions. Score instructions on a scale from 0 to 10, where 10
indicates perfect execution.
** Completeness Metrics **: Specific metrics or scores that quantify how completely
the instructions were followed. Score instructions on a scale from 0 to 10, where
10 indicates full completeness.
** Contextual Understanding Metrics **: Specific metrics or scores that quantify
```

```
the emulator's understanding of the context. Score instructions on a
↪  scale from 0
to 10, where 10 indicates perfect contextual understanding.
** Visual Consistency Metrics **: Specific metrics or scores that
↪  quantify the
visual consistency of the 'after' screenshot. Score instructions on a
↪  scale from 0
to 10, where 10 indicates perfect visual consistency.

### Output Format
Please provide your evaluation in the following JSON format:
{{\"reasons\": <reasoned analysis>, \"metrics\": {{\"accuracy\":
↪  <accuracy score>,
\"completeness\": <completeness score>, \"contextual_understanding\":
↪  <contextual
understanding score>, \"visual_consistency\": <visual consistency
↪  score>}}}}
```

E.2 Error Propagation Dynamics in Recursive Rollouts

While Infinite-Dreamer achieves strong single-step fidelity (Table 2), multi-step rollout quality exhibits a progressive decline over extended horizons (Table 2: from 70.71 at 1 step to 48.84 at 10 steps). Here we formalize the mathematical dynamics of error compounding during recursive generation, analyze the distribution shift between the two extraction templates, and describe the protective role of VLM-based quality filtering.

Formal Model of Compounding Drift. Let s_t^* denote the ground-truth reachable GUI screenshot at step t, and let $\tilde{s}_t = s_t^* + \mathbf{e}_t^{\text{vis}}$ denote the generated observation produced by recursive application of f_ϕ. Similarly, let Δ_t^* denote the ideal transition specification, and let $\tilde{\Delta}_t = \Delta_t^* + \mathbf{e}_t^{\text{sem}}$ denote the delta-text authored by the App Designer. The visual state at step $t+1$ evolves as:

$$\tilde{s}_{t+1} = f_\phi(\tilde{s}_t, \tilde{\Delta}_t) = f_\phi(s_t^* + \mathbf{e}_t^{\text{vis}},\, \Delta_t^* + \mathbf{e}_t^{\text{sem}}). \tag{10}$$

Taking a first-order Taylor expansion around the ground-truth transition (s_t^*, Δ_t^*), the visual error vector $\mathbf{e}_{t+1}^{\text{vis}}$ at step $t+1$ satisfies:

$$\mathbf{e}_{t+1}^{\text{vis}} = \tilde{s}_{t+1} - s_{t+1}^* \approx \underbrace{\nabla_s f_\phi(s_t^*, \Delta_t^*) \cdot \mathbf{e}_t^{\text{vis}}}_{\text{Autoregressive Error Compounding}} + \underbrace{\nabla_\Delta f_\phi(s_t^*, \Delta_t^*) \cdot \mathbf{e}_t^{\text{sem}}}_{\text{Semantic Mismatch Distortion}} + \delta_{\text{flow}}, \tag{11}$$

where $\nabla_s f_\phi$ is the Jacobian of the world model with respect to the input image, $\nabla_\Delta f_\phi$ captures sensitivity to delta-text specifications, and δ_{flow} represents numerical integration error from flow-matching sampling.

Equation 11 reveals two distinct driving sources of rollout error:

1. **Semantic Specification Shift ($\mathbf{e}_t^{\text{sem}}$):** During world-model training, the extractor E_ψ consumes ground-truth image pairs (s_t, s_{t+1}) to extract Δ_t. During multi-step imagination, however, the text-only extractor E_ψ^{txt} must infer Δ_t solely from high-level UI descriptions authored by the App Designer G_ω, without ground-truth visual feedback. When E_ψ^{txt} underspecifies secondary UI effects (e.g., background dimming or keyboard dismissal), $\mathbf{e}_t^{\text{sem}}$ induces visual distortions in the generated frame.
2. **Visual Memory Amplification ($\mathbf{e}_t^{\text{vis}}$):** Once $\tilde{s}_t$ departs from the manifold of clean training screenshots, the corrupted canvas is fed back into f_ϕ at step $t+1$. Because diffusion models lack explicit self-correcting mechanisms, slight font blurring or boundary shifts are often misinterpreted by f_ϕ as intentional UI structures and further amplified by $\nabla_s f_\phi$.

Rollout Survival under Quality Filtering. To characterize how error compounding affects synthesized data, we measure two quantities on the validation cache of the multi-step imagination pipeline (1,599 imagined trajectories, 9,504 validated generated screenshots; the initial page at step 0 is excluded). At horizon t we report the fraction of evaluated steps at that horizon that pass the VLM

quality filter ($\eta \geq 90$), and the cumulative fraction of trajectories reaching horizon t whose steps $1, \ldots, t$ all pass. The at-risk set falls from $1{,}599$ trajectories at $t = 1$ to 302 at $t = 10$, because imagined tasks end once their final action is reached (91% of trajectories end with a `terminate` or `answer` action).

Table 11: Survival of synthesized rollouts under VLM quality filtering ($\eta = 90$) across horizons $t = 1, \ldots, 10$, measured over $1{,}599$ imagined trajectories. *Step validity*: share of evaluated steps at horizon t that pass. *Cumulative survival*: share of trajectories reaching horizon t whose steps $1, \ldots, t$ all pass. Step 0 (the initial page) is excluded.

Rollout Step (t)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Step Validity (%)	84.6	84.9	82.8	84.7	79.0	78.4	80.4	79.4	81.2	79.5
Cumulative Survival (%)	84.6	72.0	60.6	50.3	39.8	30.4	24.1	18.7	14.1	9.3

Step validity stays within a narrow band (78.4–84.9%) across all horizons, so the filter is not systematically stricter on later steps. Cumulative survival, by contrast, falls from 84.6% at step 1 to 9.3% at step 10, because a trajectory remains usable only while every one of its steps passes. The VLM quality filter therefore acts as a *circuit-breaker*: compounding degradation, rather than a collapsing per-step pass rate, is what removes trajectories, and filtering keeps those with degraded UI elements out of the final policy training set $\mathcal{D}_{\text{gen}}$.

F Additional Qualitative Examples

This section provides qualitative comparisons between Infinite-Dreamer and baseline image-editing models across single-step and multi-step scenarios.

F.1 Single-Step Editing Comparison

We present three representative single-step editing examples comparing Infinite-Dreamer-9B against several baseline image-editing models (FLUX.1 Kontext, Step1X-Edit-v1.2, LongCat-Image-Edit, Qwen-Image-Edit-2511, FireRed-Image-Edit-1.0). The examples cover a plant-care settings screen, a recipe-filter page, and a schedule-meeting form. For each example, the source UI is transformed according to a delta-text instruction specifying UI elements to add, remove, or modify. As shown in Figure 7, Infinite-Dreamer-9B produces edits that more accurately reflect the intended transitions while preserving the overall UI structure and unmodified regions. Baseline models often fail to apply the requested changes correctly or introduce artifacts in unedited regions.

F.2 Multi-Step Rollout Comparison

Figure 8 shows a five-step rollout in the LeafSnap plant app across all compared models. Part I presents three general-purpose baselines (FLUX.1 Kontext, Step1X-Edit-v1.2, LongCat-Image-Edit), Part II compares Qwen-Image-Edit-2511, FireRed-Image-Edit-1.0, and Infinite-Dreamer-4B (Ours), and Part III shows the ground-truth trajectory alongside Infinite-Dreamer-9B (Ours). Our models faithfully track the ground-truth UI state transitions, while baselines accumulate compounding errors across steps.

Across all five steps, our Infinite-Dreamer models preserve plant names, UI text, and the toggled reminder state correctly, while baselines show corrupted text, duplicated or missing UI elements, and inconsistent backgrounds—demonstrating the importance of domain-specific fine-tuning for GUI state synthesis.

G More Visualizations

This section showcases the diversity and quality of images generated by Infinite-Dreamer under two synthesis regimes: single-frame visual augmentation and multi-step trajectory imagination.

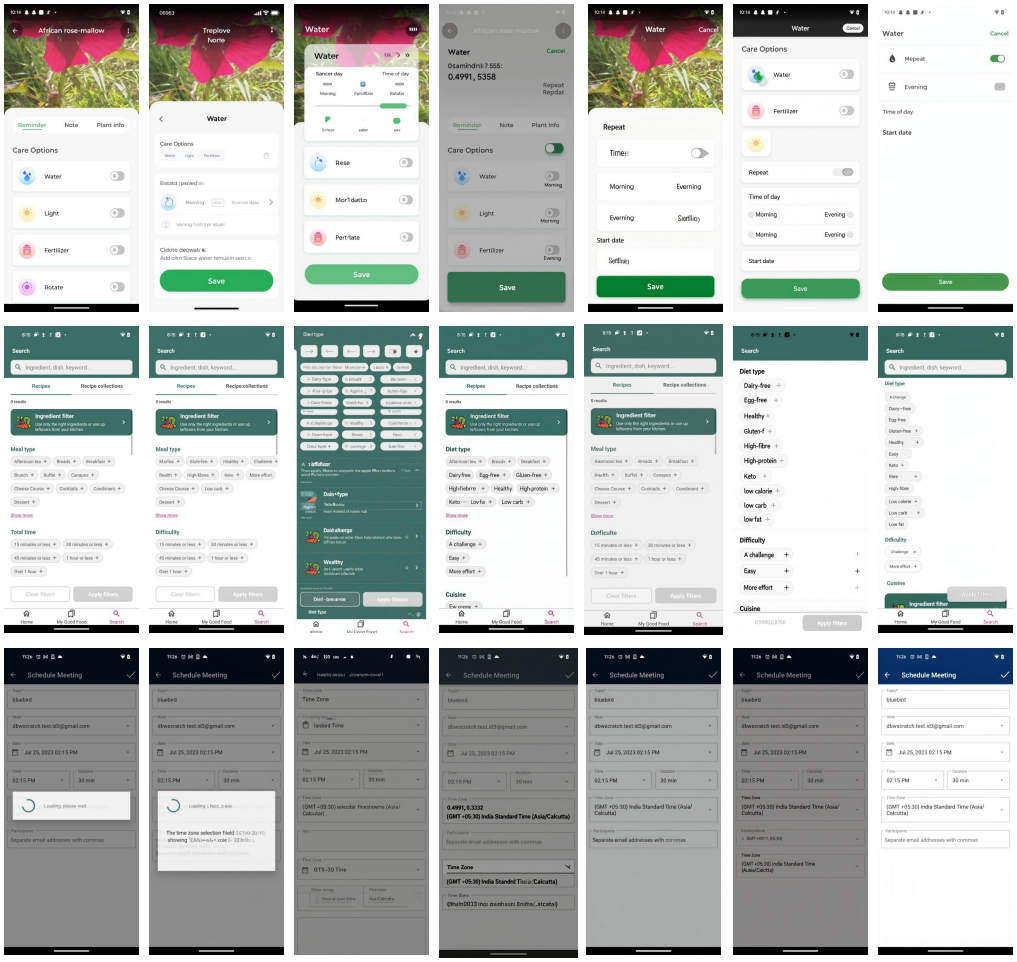


Figure 7: Qualitative comparison of single-step UI editing. Three examples (rows) show the source UI and outputs from FLUX.1 Kontext, Step1X-Edit-v1.2, LongCat-Image-Edit, Qwen-Image-Edit-2511, FireRed-Image-Edit-1.0, and Infinite-Dreamer-9B (rightmost). Infinite-Dreamer-9B most faithfully executes the requested edits while preserving layout and unmodified regions.

G.1 Single-Frame Augmentation Results

Single-frame augmentation generates visually diverse variants of a source screenshot while strictly preserving the layout and all text content. Figure 9 shows source–variant pairs spanning multiple app categories, including a recipe editor, a dialer, a system Display settings page, a file manager, an event scheduler, a file-explorer document preview, a messaging composer, a hiking route planner, and a Battery Saver settings page. The variants exercise dark/light theme swap, accent-color and background recoloring, and icon/keyboard restyling, while every button, list item, and text label from the source is reproduced verbatim. These augmentations improve the agent's robustness to surface-level UI variations without requiring new task labels.

G.2 Multi-Step Imagination Results

Multi-step imagination generates complete trajectory rollouts from a seed screenshot using the App Designer and world model. Figure 10 shows three independent rollouts, each starting from an Initial state and unfolding over five steps. Every step is conditioned on the previous screenshot together with a delta-text action label—for example "Click the *Create new folder* Button", "Input the Text

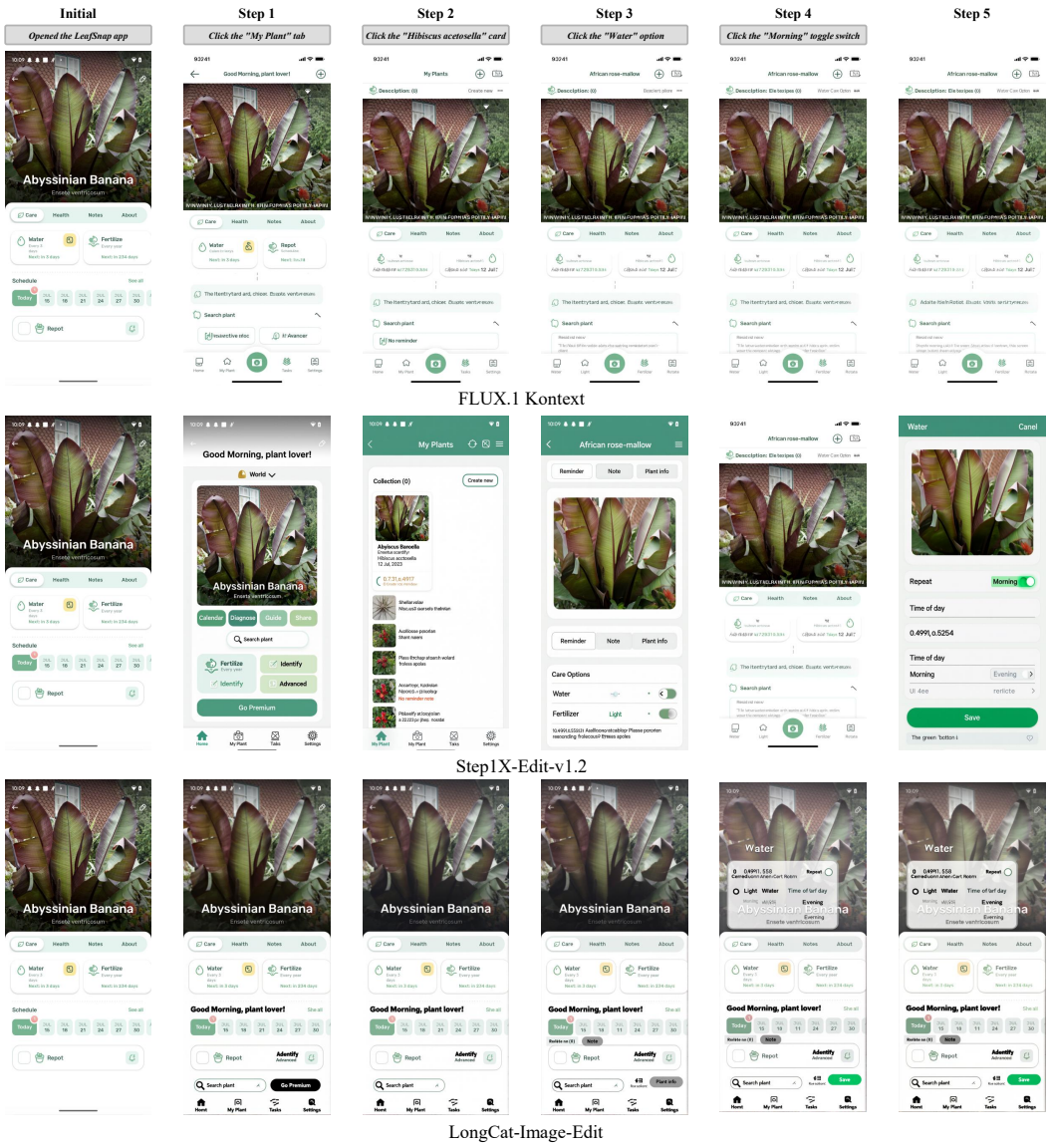


Figure 8: Qualitative comparison of multi-step rollout. (a) Part I shows outputs from FLUX.1 Kontext, Step1X-Edit-v1.2, and LongCat-Image-Edit for the five-step LeafSnap watering task; each column is a step. These baselines drift away from the intended UI state.

daily_goals", or "Click the Send Button". Across the three trajectories, Infinite-Dreamer opens dialogs, types into text fields, and dismisses modals while preserving the surrounding UI chrome.

G.3 Detailed Failure Mode Analysis, Diagnostics, and Mitigations

While Infinite-Actor demonstrates marked improvements on end-to-end task success (AndroidWorld Pass@1 +4.45 to +9.05), two performance anomalies warrant deeper investigation: (i) regressions in coordinate grounding (Box Acc) on AC-Curated, and (ii) a -2.7 point regression on AndroidWorld Hard tasks when mixing synthetic data with real demonstrations (Figure 3a). Here we measure the failure outcomes of the mixed-data model from its evaluation logs, relate them to tier-dependent regressions, and discuss candidate mechanisms together with potential mitigations.

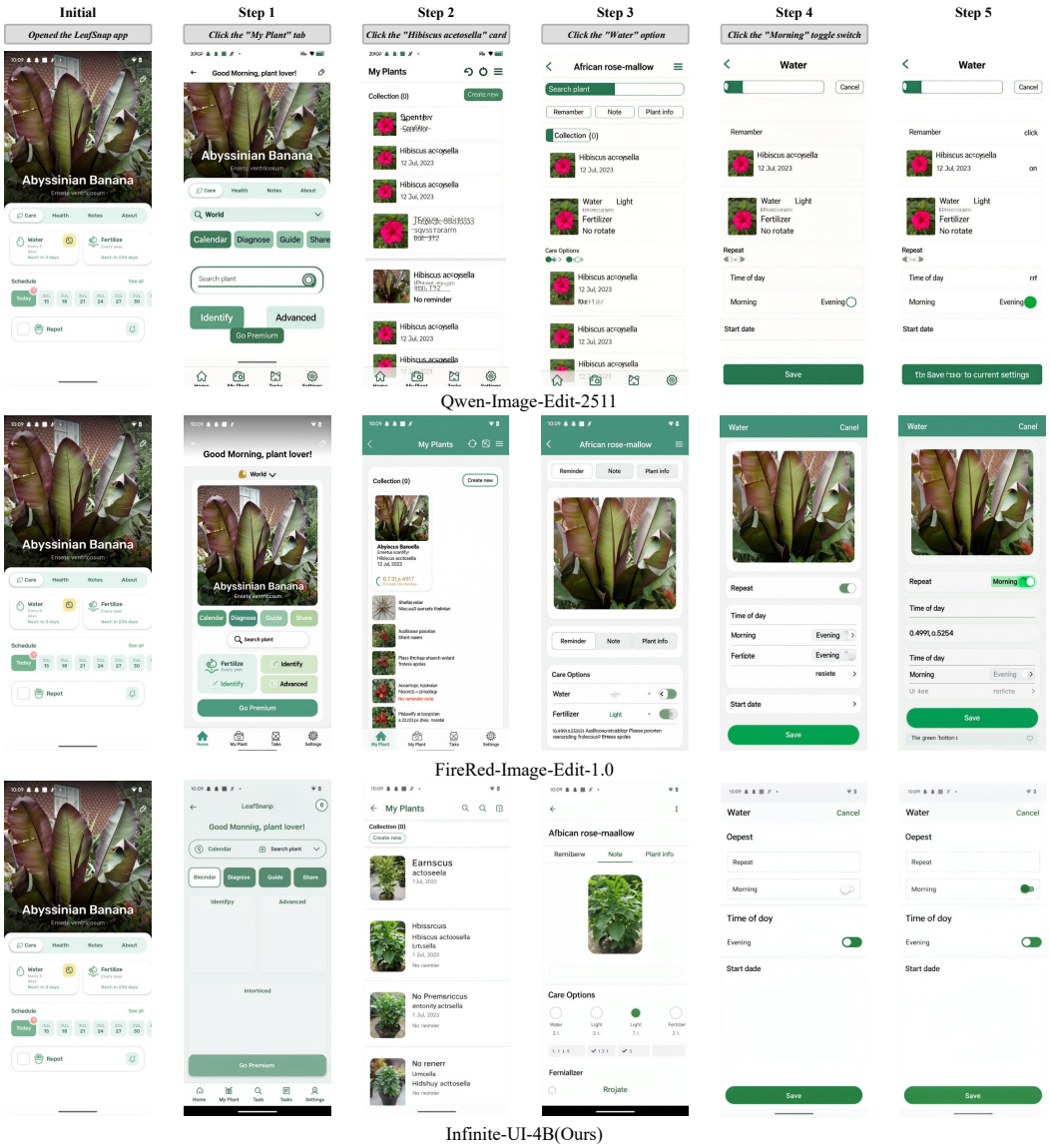


Figure 8: (b) Part II shows outputs from Qwen-Image-Edit-2511, FireRed-Image-Edit-1.0, and Infinite-Dreamer-4B (Ours) for the same rollout. Infinite-Dreamer-4B preserves visual coherence while the baselines introduce artifacts and layout drift.

Measured Failure Outcomes on AndroidWorld Hard Tasks. To understand why synthetic data fails to boost Hard tasks, we analyze the evaluation logs of `mix-4b` on the 19 AndroidWorld Hard tasks (one episode per task: two successes and 17 failures). Each failure is assigned to exactly one terminal outcome recorded in the log and listed in Table 12. The episode either exhausted the interaction budget the harness allocates ($10\times$ the task complexity score), terminated with `goal_status=complete` although the verifier still failed the task, ended with an `answer` action that failed verification, or stopped mid-interaction without terminating. None of the 17 failures ends with an explicit infeasible verdict.

Two process-level symptoms accompany these outcomes. First, $6/17$ failed episodes contain a run of at least three consecutive identical actions, meaning the agent repeats an action or oscillates between screens instead of progressing, and the same number of episodes contain the agent's own statement that it is stuck in a loop. Second, $7/17$ episodes execute at least one action after which

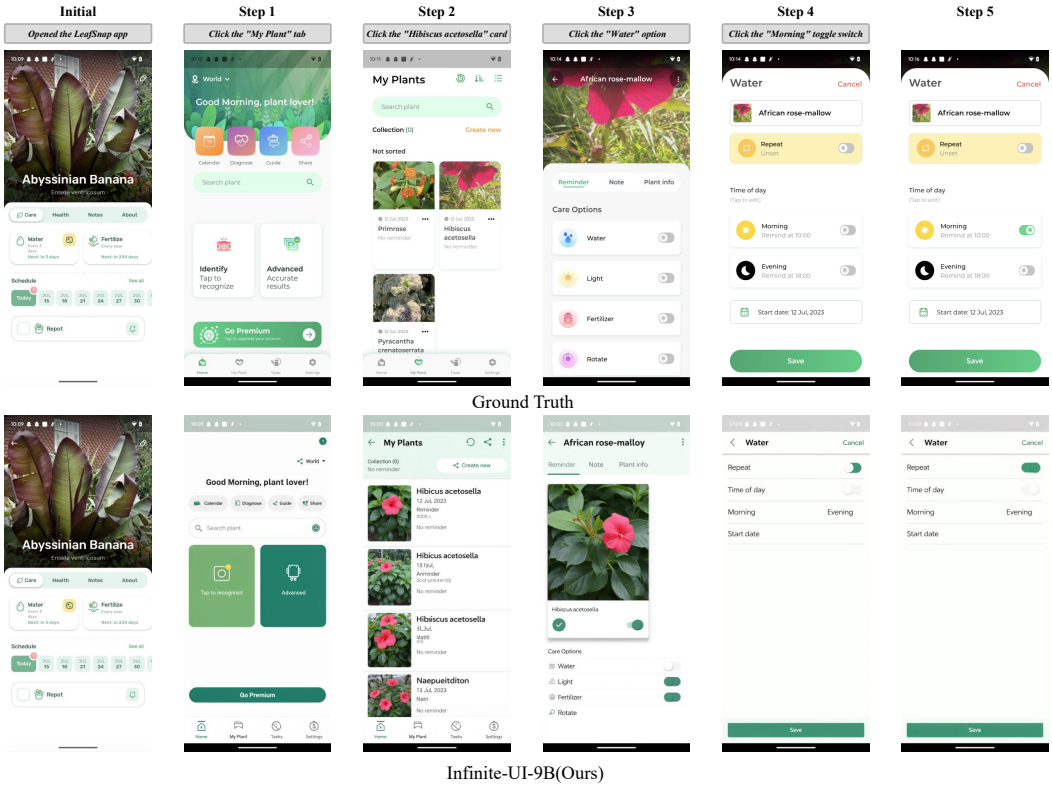


Figure 8: (c) Part III compares the ground-truth trajectory with Infinite-Dreamer-9B (Ours). Infinite-Dreamer-9B closely follows the ground-truth layout and text across all five steps.

Table 12: Measured failure outcomes of `mix-4b` on the 19 AndroidWorld Hard tasks (17 failures, one episode per task). Categories are mutually exclusive and are assigned from the terminal event recorded in the evaluation log.

Failure Mode	**Share**	**Operational Definition (from the evaluation log)**
Step-budget exhaustion	41.2% (7/17)	Ran to the allocated interaction budget (10× task complexity) while still acting; no terminal action was issued.
False completion	35.3% (6/17)	Issued a `terminate` call with `goal_status=complete`, yet the task verifier marked the episode failed.
Incorrect final answer	17.6% (3/17)	Ended with an `answer` action whose response failed verification (three information-retrieval tasks).
Abrupt stop	5.9% (1/17)	Episode ended mid-interaction; the log records no `terminate` or `answer` call.

the screenshot is byte-identical, and 3/17 issue clicks that produce no screen change at all. These symptoms concentrate in the Hard tier: among failed episodes, false completion rises from 14.3% (Easy) and 30.4% (Medium) to 35.3% (Hard), and repetition or oscillation rises from 19.0% and 21.7% to 35.3%. Both Hard successes do terminate with `goal_status=complete` and pass verification, so the discriminating factor is not whether the agent terminates but whether its claimed state matches the environment.

From Measured Outcomes to Mechanisms. Table 12 describes how Hard episodes fail, not which property of the synthetic corpus caused the failure. We consider four candidate mechanisms; each would require a controlled study to confirm. First, *coordinate displacement*: on AC-Curated,

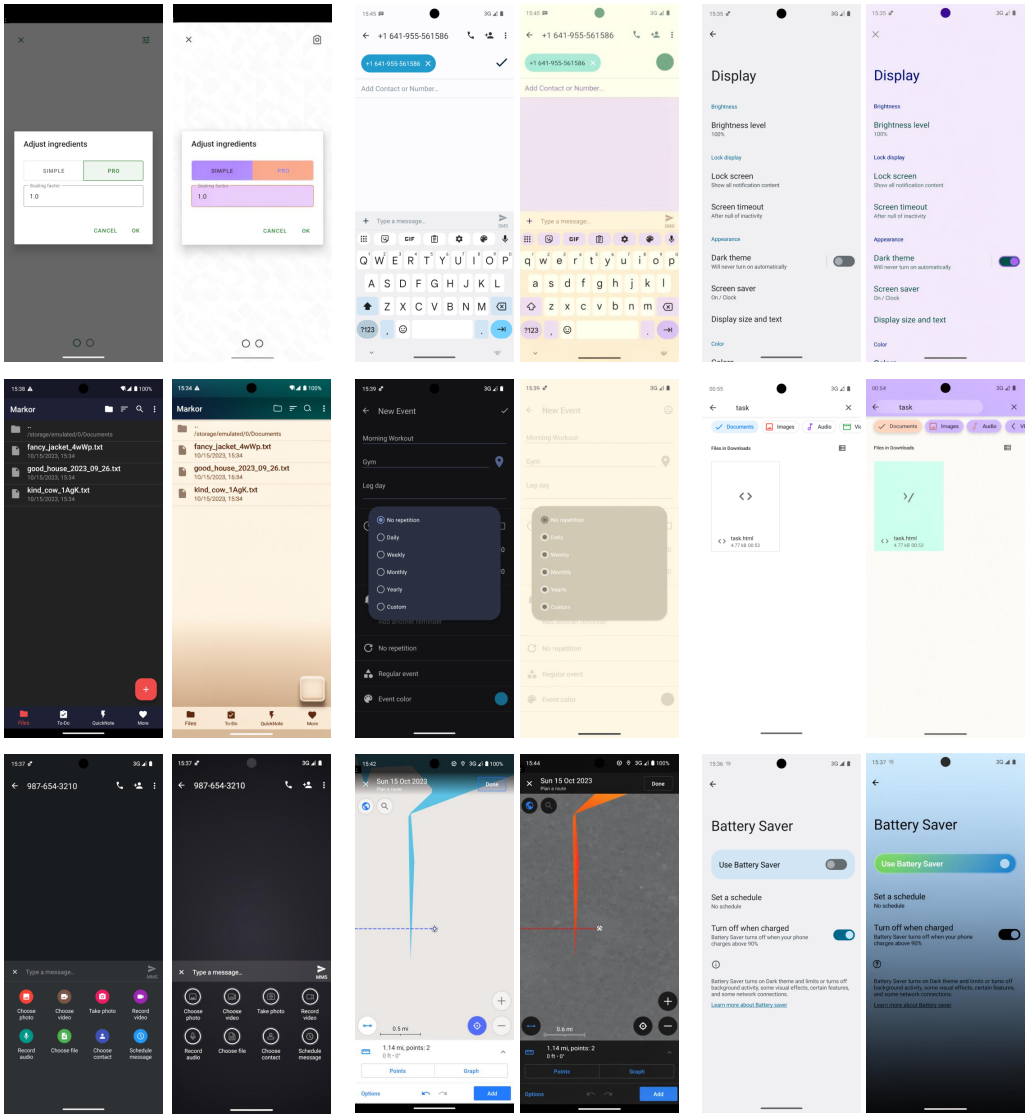


Figure 9: Single-frame augmentation examples. For each pair, the left column shows the source screenshot and the right column shows a visually distinct variant produced by Infinite-Dreamer. The pairs span multiple app categories and cover dark/light theme swap, accent-color and background recoloring, and icon/keyboard restyling, while widget positions and text content are held fixed.

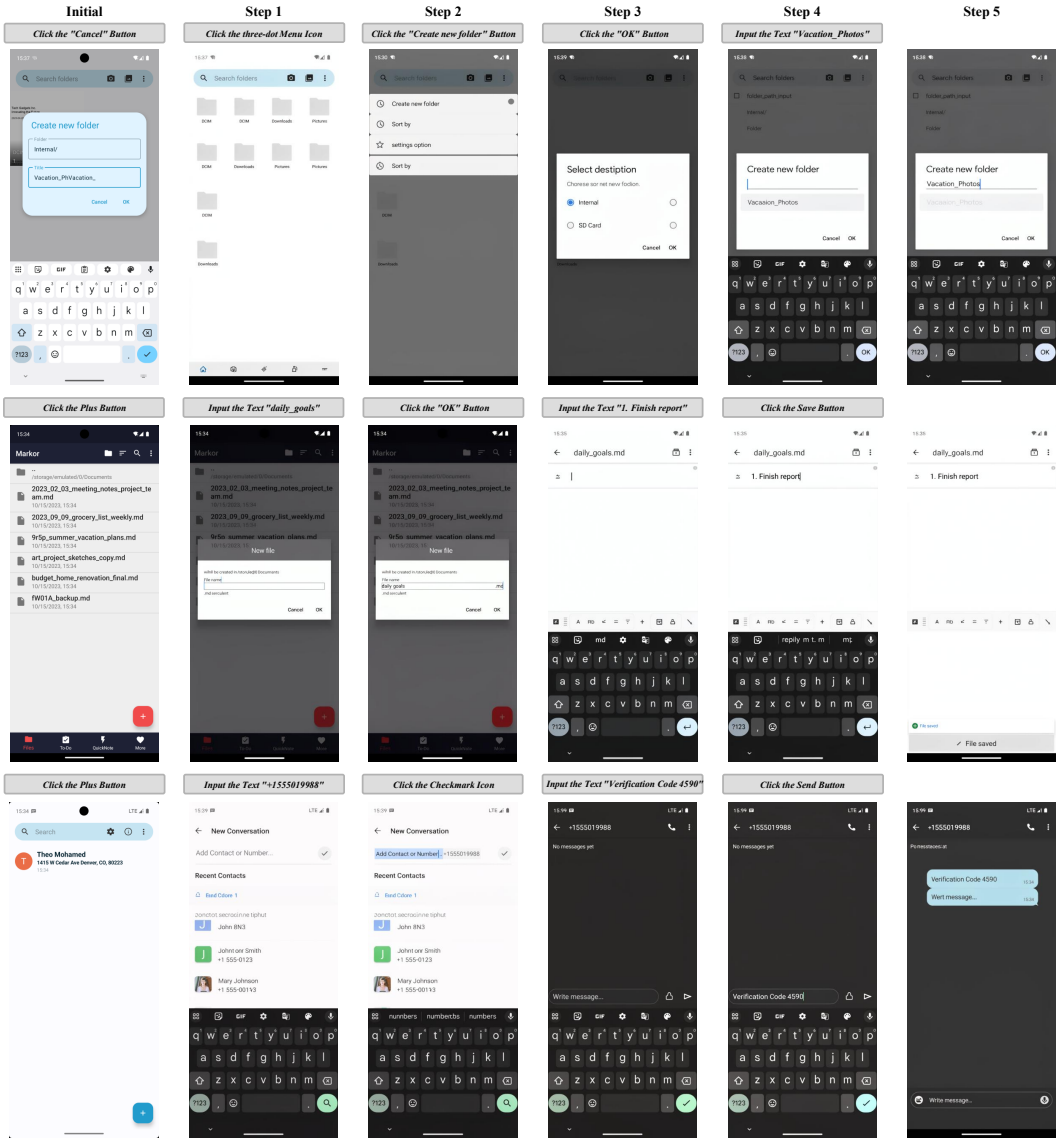
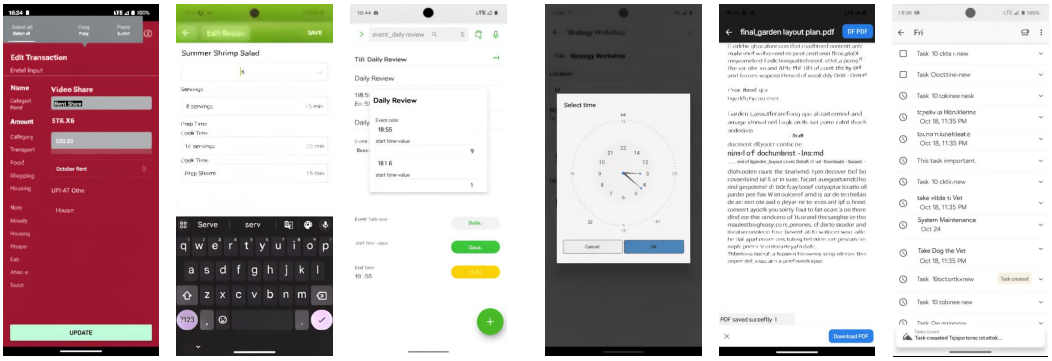


Figure 10: Multi-step imagination examples. Each row is an independent trajectory: starting from an Initial screenshot, Infinite-Dreamer autoregressively applies the delta-text action printed above each column (e.g., "Click the *Create new folder* Button", "Input the Text *daily_goals*", "Click the Send Button") to produce Step 1 through Step 5. The imagined screenshots keep the surrounding UI chrome stable while correctly rendering the widget or text that the action introduces.

Infinite-Actor-4B's predicted regions are displaced from the annotations rather than marginally noisy, and the loss concentrates on compact targets (Section 4). That comparison changes both fine-tuning and the data sources, so it is consistent with, but does not isolate, an effect of synthetic supervision; separating boundary changes in the generated screenshots from the policy's visual encoding would require a controlled study. Its behavioural symptom in the Hard logs would be ineffective clicks, which occur in $3/17$ failures. Second, *long-horizon causal noise*: world-model fidelity decays with rollout length (Table 2), and only 9.3% of trajectories that reach step 10 survive quality filtering intact (Table 11), so long synthetic rollouts are both scarce and biased toward easy horizons. A policy trained on such supervision can adopt over-optimistic beliefs about progress and termination, which is consistent with the dominance of budget exhaustion and false completion above. Third, *dense-UI confusion*: as shown in Figure 11b, the world model can merge adjacent widgets or misplace insertions in tightly packed layouts, which in principle teaches incorrect element attributes. Fourth, *transient system-dialog inconsistency*: ephemeral runtime permission dialogs and system keyboards are unlikely to appear in imagined screenshots, so a policy trained on such data may mishandle them on a real device. The evaluation logs record actions but not whether a dialog was on screen, and they contain no element-level annotations, so the third and fourth mechanisms are not measurable from them and remain hypotheses.



(a) Long-horizon drift. **(b) Dense UI challenges.**

Figure 11: Representative failure cases of Infinite-Dreamer. (a) Long-horizon drift: over many generated steps, visual inconsistencies such as garbled text, shifted buttons, and spurious elements accumulate. (b) Dense UI challenges: tightly packed widgets (e.g., grid layouts, complex lists) can cause the model to merge adjacent elements or misplace insertions and removals.

Constructive Mitigation Strategies. These findings point to actionable avenues for advancing generative GUI world models:

- **Termination calibration and progress verification:** Since false completion accounts for 35.3% of Hard failures, training should verify task post-conditions before accepting a `terminate` call, for example by re-checking the environment state or by filtering synthetic trajectories whose terminal state was never observed in real data.
- **Box-preserving attention guidance:** Incorporating explicit bounding-box coordinate losses or masking during world-model fine-tuning can anchor element boundaries more tightly and reduce the coordinate displacement of predicted regions.
- **Keyframe re-anchoring in long horizons:** Since only 9.3% of rollouts that reach step 10 remain fully valid, periodically re-anchoring the visual state to real seed screenshots or a template library can prevent compounding visual and causal drift before it exhausts the horizon.
- **Consistency verification for Hard tasks:** Applying bi-directional rollout verification (e.g., checking whether reverse actions reconstruct the prior state) before admitting long-horizon synthetic trajectories into the training corpus can suppress spurious transitions.