GUI-CC: Benchmarking Contextual Consistency of GUI World Models as Agent Environments

Lin Fu[1*] **Zheyuan Yang**[2*] **Tianhui Zhang**[3*] **Jinbiao Wei**[4]
Guo Gan[1] **Boxu Liu**[5] **Yilun Zhao**[4✉] **Yu Rong**[6]

[1]Zhejiang University [2]Tongji University [3]University of California, San Diego
[4]Yale University [5]China University of Geosciences [6]DAMO Academy, Alibaba Group

Codebase Dataset

Abstract

GUI world models are increasingly evaluated as one-step next-screen predictors, yet their intended use is often as multi-step environments for GUI agents. This mismatch leaves a key requirement under-tested: generated states must remain contextually consistent when they are repeatedly reused for future interaction. We introduce GUI-CC, a benchmark that evaluates contextual consistency of GUI world models as agent environments rather than isolated next-screen predictors. GUI-CC contains two complementary tracks: an offline reference-action track that rolls models along real mobile GUI trajectories, and an online agent-loop track that lets fixed probing agents interact with model-generated UIs. We construct 500 offline trajectory tasks from GUIOdyssey and 200 emulator-verified online tasks across 30 mobile apps. GUI-CC evaluates transition fidelity, transition plausibility, contextual consistency, and task progress. Experiments show that plausible single-step generation does not guarantee reliable environment simulation: current models often produce usable-looking screens while failing to preserve task-relevant context or support executable multi-step rollouts.

1 Introduction

Autonomous GUI agents are defined by interactive execution: they observe graphical interfaces, issue UI-grounded actions, and complete tasks through environment state changes. Live web, desktop, and mobile benchmarks instantiate this setting with task initialization and execution-based success checks (Zhou et al., 2024; Xie et al., 2024; Rawles et al., 2025), while large-scale mobile datasets provide scalable logged demonstrations for training and evaluation (Rawles et al., 2023; Li et al., 2024; Lu et al., 2025). Yet neither resource type alone provides cheap, reproducible counterfactual rollouts at scale: live environments are interactive but costly to reset and execute (Gan et al., 2026a; Chen et al., 2025), while logged trajectories are scalable but fixed and cannot simulate unseen actions. GUI world models address this gap by learning action-conditioned transition dynamics, serving as surrogate environments for simulated rollouts, model comparison, and planning (Chae et al., 2025; Luo et al., 2025; Zheng et al., 2026; Koh et al., 2026; Guan et al., 2026; Xu et al., 2026).

This surrogate-environment use changes what should be evaluated. A generated next screen may be locally plausible, but once it is fed back as the next state, future actions depend on whether app identity, navigation context, task-relevant entities, selected or created state, and action affordances remain coherent over the rollout. We call this requirement *contextual consistency*. Prior GUI world-model evaluations (Li et al., 2025; Koh et al., 2026; Li et al., 2026; Zheng et al., 2026) typically measure semantic or visual transition quality, scripted GUI generation, or isolated world-model outputs, but do not directly diagnose whether generated states remain coherent under repeated state reuse. As shown in Figure 1, visually plausible rollouts can still drift to the wrong app, delay action effects, lose saved state, or regenerate a launcher without the target app, leaving the agent in a plausible-looking but invalid environment.

We introduce GUI-CC, a benchmark for evaluating *contextual consistency* of GUI world models as agent environments. GUI-CC contains two complementary tasks. The *offline trajectory task* starts from a real initial UI and rolls out the world model along reference semantic action sequences, providing a controlled setting for testing whether generated states remain executable across real interaction trajectories. The *online agent-loop task* lets fixed probing GUI agents act on model-generated UIs, directly testing whether the generated envi-

*Equal contributions. ✉Corresponding author: yilun.zhao@yale.edu.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 7798–7828
October 24-29, 2026 ©2026 Association for Computational Linguistics

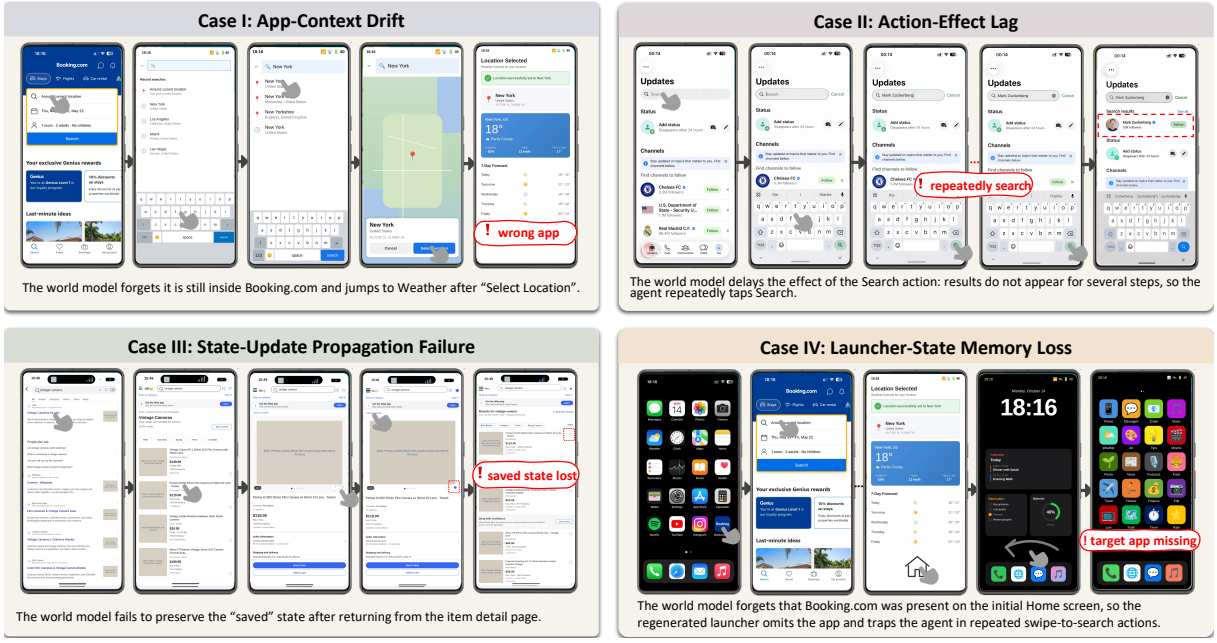


Figure 1: Representative failure cases of GUI world models under long-horizon rollouts. The examples illustrate app-context drift, action-effect lag, state-update propagation failure, and launcher-state memory loss.

ronment can support closed-loop task progress. We construct 500 offline trajectory tasks based on GUIOdyssey (Lu et al., 2025) and 200 emulator-verified online agent-loop tasks over 30 mobile apps and 18 task templates. Our evaluation dimensions include transition fidelity, transition plausibility, contextual consistency, and task progress, separating local prediction quality from trajectory-level environment utility.

Our evaluation shows that plausible single-step prediction does not imply reliable rollout behavior. Models can obtain reasonable local transition scores while still losing task-relevant state, drifting across apps or pages, or failing to support later reference actions. History-conditioned inputs improve consistency in some cases, but the overall task-progress scores remain limited, especially in longer rollouts and closed-loop agent interaction. These results suggest that GUI world models are promising as learned rollout environments, but current systems still fall short of the contextual consistency needed for scalable agent training and evaluation.

2 Related Work

GUI Agent Benchmarks and Interaction Datasets. Live GUI agent benchmarks evaluate agents through execution in real web (Zhou et al., 2024), desktop (Xie et al., 2024), and mobile (Rawles et al., 2025; Wei et al., 2026c,b,a; Zhou et al., 2026; Gan et al., 2026b) applications, but their reliance on live execution makes large-scale rollout costly and hard to reproduce. Large-scale mobile interaction datasets (Rawles et al., 2023; Li et al., 2024; Lu et al., 2025) provide instructions, screenshots, actions, and semantic annotations from real GUI trajectories, yet they are demonstrations rather than learned environments and do not by themselves test whether a model can replace the transition function during agent rollout.

GUI World Models. GUI world models predict the consequences of actions in graphical interfaces, letting agents reason about future UIs without executing every action on a live device. Existing models differ mainly in how the predicted state is represented: textual or semantic transitions expressed in language (Chae et al., 2025; Li et al., 2025; Cao et al., 2026), structured UI representations such as DOM or accessibility trees (Xiao et al., 2026), image-level screenshot synthesis (Luo et al., 2025; Xu et al., 2026), renderable code that is rendered back into screenshots (Zheng et al., 2026; Koh et al., 2026), and hybrid pipelines combining textual transitions with visual realization (Guan et al., 2026). Serving as an environment additionally requires generated states to remain mutually consistent when reused across the rollout.

Evaluating GUI World Models. Existing evaluations of GUI world models score one-step next-state fidelity inside model papers (Chae et al.,

Evaluation	Release	Domain	Output State	Multi-Step	Pred. Fed Back	Agent-in-Loop	Direct WM Eval	Consistency Eval
WMA (Chae et al., 2025)	2024-10	Web	Text	✓	✗	✗	✗	✗
MobileWorldBench (Li et al., 2025)	2025-12	Mobile	Text	✗	✗	✗	✓	✗
gWorld (Koh et al., 2026)	2026-02	Mobile	HTML code	✗	✗	✗	✓	✗
GEBench (Li et al., 2026)	2026-02	Mobile / desktop	Image	✓	✗	✗	✓	✗
Code2World (Zheng et al., 2026)	2026-02	Mobile	HTML code	✗	✗	✗	✓	✗
CUWM (Guan et al., 2026)	2026-02	Desktop	Text / image	✗	✗	✗	✓	✗
WebWorld (Xiao et al., 2026)	2026-02	Web	Structured	✓	✓	✓	✓	✗
MobileWorld (Xu et al., 2026)	2026-05	Mobile	Code / image / text	✓	✓	✓	✓	✗
GUI-CC Offline Track	2026-05	Mobile	Code / image	✓	✓	✗	✓	✓
GUI-CC Agent-Loop Track	2026-05	Mobile	Code / image	✓	✓	✓	✓	✓

Table 1: Comparison of GUI-CC with prior GUI world-model evaluations. **Pred. fed back**: predicted states are reused as the next environment state during the evaluation rollout. **Agent-in-loop**: the rollout is driven by an agent acting on generated states. **Direct WM eval**: the headline metric is computed on the world model's output rather than on downstream agent or search outcomes. **Consistency eval**: the evaluation directly tests cross-step state-reuse consistency. GUI-CC measures reference-action executability and context persistence in the offline track, and milestone progress, loops, and stagnation in the agent-loop track. GEBench reports an image-level temporal coherence score but does not test state-reuse consistency.

2025; Zheng et al., 2026; Koh et al., 2026; Guan et al., 2026), semantic next-state prediction in language (Li et al., 2025), multi-step GUI generation from scripted or instruction-conditioned inputs (Li et al., 2026), or downstream planning utility for an agent (Xiao et al., 2026). None of these settings tests whether predicted states remain a coherent environment when repeatedly fed back as the next input, where task-relevant latent state such as app identity, navigation history, created entities, selected options, and launcher layout must remain mutually consistent. GUI-CC fills this gap with an offline track that measures whether a real semantic action sequence remains executable under autoregressive prediction and an agent-loop track that measures whether fixed probing agents can make progress inside the generated environment.

3 GUI-CC Benchmark

This section presents GUI-CC, a benchmark evaluating GUI world models as agent environments under multi-step action-conditioned state reuse; Figure 2 provides an overview of the data construction pipeline and track statistics.

3.1 Problem Formulation

GUI-CC evaluates whether a GUI world model can serve as an interaction environment, not merely whether it can generate a plausible next screen. A real GUI environment maintains a latent system state s_t containing apps, the navigation stack, settings, user-created entities, and other hidden context. At each step an agent chooses $a_t \sim \pi(\cdot \mid \tau_t, g)$ from history $\tau_t = (o_0, a_0, \ldots, o_t)$ and instruction g, and the environment transitions via

$$(s_{t+1}, o_{t+1}) = E(s_t, a_t).$$

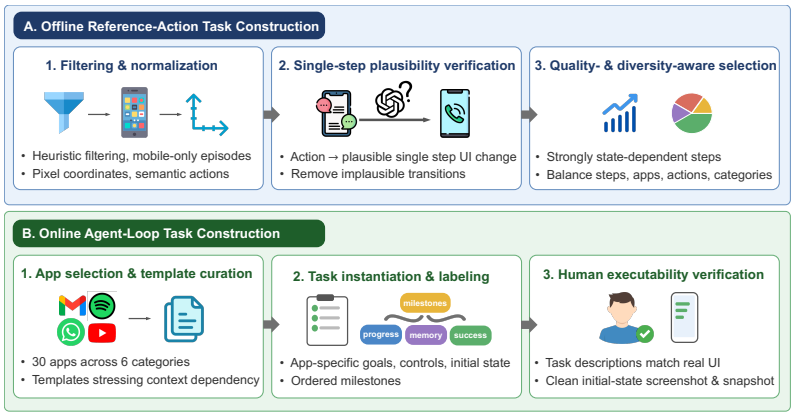
A GUI world model replaces this transition with a learned function

$$\hat{o}_{t+1} = W(H_t, a_t),$$

where $H_t = (o_0, a_0, \hat{o}_1, \ldots, a_{t-1}, \hat{o}_t)$ is the generated history and each $\hat{o}_t$ becomes the input on which later actions are chosen or evaluated; W never observes s_t, so whatever continuity the rollout exhibits must be reconstructed from the generated UIs. A rollout is *contextually consistent* if there exists a plausible latent-state sequence $(s_0, \ldots, s_T)$ under which each $\hat{o}_t$ is a rendering of s_t and each action a_t remains executable on s_t. GUI-CC operationalizes this property at the trajectory level: it measures whether the entire rollout admits such a latent-state explanation across multiple steps of action-conditioned reuse, rather than scoring isolated single-step predictions.

3.2 Benchmark Tracks

GUI-CC instantiates contextual consistency through two complementary tracks. The **Offline Reference-Action Track** uses real GUI trajectories $(o_0, a_0, o_1, \ldots, a_{T-1}, o_T)$ as references: starting from o_0, the world model rolls out autoregressively along the reference semantic action sequence, and the reference screenshots are used only to score the predicted rollout, never as rollout inputs. The **Online Agent-Loop Track** replaces the fixed action sequence with a frozen probing agent that chooses each a_t on the previously generated rollout, until task completion, unusable output, or stagnation.



Statistic	Offline	Online
# Tasks	500	200
# Transitions	4,905	—
Avg. steps / budget	9.81	15.74
Min / max steps / budget	7 / 14	6 / 24
# Apps	130	30
# App categories	6	6
# Templates	—	18
Avg. milestones	—	4.0
Min / max milestones	—	2 / 5

Figure 2: **(Left)** Data construction pipeline of GUI-CC. Both tasks are designed to stress contextual consistency under multi-step rollout. The offline trajectory task selects high-quality real mobile trajectories from GUIOdyssey, while the online agent-loop task creates emulator-verified tasks with ordered milestones for closed-loop agent evaluation. **(Right)** Statistics of the two tasks in GUI-CC. Dashes indicate statistics that are not applicable to the corresponding track: the offline reference-action track does not use task templates or ordered milestones, while the online agent-loop track is evaluated with a step budget rather than a fixed set of reference transitions.

The two tracks share the autoregressive rollout structure but probe different aspects of environment consistency: the offline track fixes the action sequence to isolate the world model from agent behavior, while the online track lets the agent react to predictions to test whether the generated environment can sustain closed-loop interaction.

3.3 Action Representation

GUI-CC represents every action in a semantic schema rather than as raw pixel coordinates. Eight interaction types are shared by both tracks: *tap* and *long_press* carry pixel coordinates and a natural-language target description; *scroll* carries a direction and normalized start and end points; *type_text* carries the input text; *open_app* carries the target app name; and *navigate_back*, *navigate_home*, and *wait* require no additional parameters. The agent-loop track adds two terminal actions, *terminate* and *answer*; offline reference trajectories instantiate the five types that occur in GUIOdyssey (*tap*, *scroll*, *navigate_home*, *navigate_back*, *long_press*). Coordinates are restored from GUIOdyssey's normalized $[0, 1000]$ range to pixel coordinates using the screen resolution, and target descriptions are drawn from low-level instructions where available and labeled by a VLM otherwise. This representation lets the offline track score reference-action executability based on whether the affordance remains visible in the predicted UI, decoupled from pixel-level reproduction.

3.4 Offline Reference-Action Track

The offline track is constructed from GUIOdyssey (Lu et al., 2025), a large-scale mobile GUI navigation dataset that provides screenshots, action records, and semantic annotations for multi-step cross-app interactions.

Stage 1: Filtering and Normalization. We first heuristically filter trajectories for world-model evaluation, keeping completed phone episodes with moderate steps and meaningful actions, while removing failed, repetitive, launcher-dominated, or structurally abnormal trajectories. We then normalize each retained trajectory into a canonical sequence $(o_0, a_0, o_1, \ldots, o_T)$ by restoring coordinates from the normalized $[0, 1000]$ range to pixels and converting raw action records into semantic actions. This yields a unified trajectory format for autoregressive rollout and evaluation.

Stage 2: Single-Step Plausibility Verification. We verify normalized trajectories at the transition level to ensure each recorded action corresponds to a plausible single-step UI change. Although they come from real interactions, logs may contain missing operations or off-record state changes. We use GPT-5.5 to inspect each before-action-after triple and judge whether the after state can result from the recorded action. Trajectories with implausible transitions are removed, so the remaining references can serve as action-conditioned rollouts.

Stage 3: Quality- and Diversity-Aware Selection. From the verified pool, we select 500 offline tra-

jectory tasks based on data quality and diversity. For data quality, we use GPT-5.5 to inspect the full trajectory and estimate the number of strongly state-dependent steps. We prioritize trajectories with more such steps, since they are more informative for evaluating contextual consistency. For diversity, we balance trajectory steps, apps, action types, task categories, and interaction patterns, ensuring the final set covers a broad range of rollout structures and GUI behaviors.

3.5 Online Agent-Loop Track

The online track covers 30 mobile apps, spanning six categories: communication, travel & navigation, shopping & delivery, finance, media & entertainment, and system & utility.

Stage 1: App Selection and Task Template Curation. Annotators first brainstorm AI-assisted task templates, then manually review and refine them for executability and app-specific applicability. These templates stress contextual consistency through multi-step patterns, including cross-app navigation, state persistence after context changes, and revisiting task-relevant entities.

Stage 2: Task Instantiation and Labeling. Each selected template is instantiated by grounding it in the concrete UI flow of a target app. Annotators specify the task goal, app-specific pages or controls, required account or environment conditions, and initial-state description, turning a generic template into an executable task with a specific starting state and expected interaction path. Annotators also provide an ordered milestone sequence for evaluation. The milestones mark key states to be reached during execution, including intermediate progress, state persistence after context changes, and terminal task success. They allow the online task to measure whether the model-generated environment supports both task advancement and retention of established context.

Stage 3: Human Executability Verification. Finally, each online task is manually verified in an Android emulator. Annotators execute the task from the specified initial state, confirm that the instruction and milestones match the real UI, and save a clean initial-state screenshot and emulator snapshot for evaluation. Tasks with incorrect UI descriptions, unreachable milestones, or unstable conditions are revised or removed. The step budget is set from human completion length, typically around twice the number of human steps.

3.6 Quality Control

After construction, we perform an independent second-pass audit on both tracks. In the offline track, we re-check whether the screenshots, semantic actions, and transition sequence are coherent. In the online track, we re-check whether the instruction and initial state are clear, the milestones are observable, and the step budget is reasonable. Tasks with minor annotation issues are revised, while unreliable tasks are removed. This second-pass audit helps ensure that benchmark failures mainly reflect world-model inconsistency rather than data or task artifacts. For reproducibility, all VLM judges run with fixed model versions and frozen prompts, which we release together with the benchmark. More details of the data construction process are provided in Appendix E.

3.7 Evaluation Metrics

We evaluate GUI world models along four dimensions: *transition fidelity*, *transition plausibility*, *contextual consistency*, and *task progress*. The first two assess local prediction quality, extending Code2World (Zheng et al., 2026) with GUI state usability; the latter two assess coherence and executability across multi-step rollouts.

Transition Fidelity. This dimension evaluates whether the predicted next UI matches the reference UI in visual content and structure. SigLIP (S_{sig}) and DINO (S_{dino}) measure global similarity, while two fine-grained metrics assess UI details: (1) *Element Alignment* (S_{ele}) checks whether key UI elements are correctly present and aligned; (2) *Layout Integrity* (S_{lay}) checks whether the page layout and visual hierarchy are preserved.

Transition Plausibility. This dimension evaluates whether the predicted transition is a reasonable action-conditioned GUI update via three metrics: (1) *Action Adherence* (S_{ad}) checks whether the predicted UI reflects the intended effect of the action; (2) *Action Identifiability* (S_{id}) checks whether the action can be inferred from the visual change; (3) *GUI State Usability* (S_{use}) checks whether the predicted UI is readable, coherent, and usable for further interaction.

Contextual Consistency. This dimension evaluates whether rollout remains consistent with history

#	World Model	Setting	Transition Fidelity				Transition Plausibility			CC		TP	Overall
			S_{ele}	S_{lay}	S_{sig}	S_{dino}	S_{ad}	S_{id}	S_{use}	S_{cp}	S_{rd}	S_{rap}	
Code / HTML-output world models													
1	Claude Opus 4.7	w/ history	16.8	15.6	73.4	47.1	**69.5**	**79.0**	**99.6**	**65.0**	**93.8**	**16.4**	**57.6**
2	GPT-5.5	w/ history	**17.8**	**16.6**	**75.0**	**49.8**	63.4	68.2	99.4	59.5	80.9	15.3	54.6
3	Claude Opus 4.7	w/o history	13.0	11.9	71.0	45.9	62.0	78.3	99.6	46.9	78.1	13.1	52.0
4	GPT-5.5	w/o history	13.1	11.8	71.5	47.5	53.3	63.7	98.2	35.9	55.1	14.7	46.5
5	Code2World	w/ history	9.8	8.3	68.9	36.0	49.1	61.5	96.0	38.0	58.9	9.5	43.6
6	MobileWorld-html-8B	w/ history	10.3	9.0	70.3	40.6	47.6	60.2	93.4	30.7	53.8	8.0	42.4
7	MobileWorld-html-8B	w/o history	10.2	8.8	69.4	37.5	44.0	57.6	91.0	28.7	54.3	9.3	41.1
8	Code2World	w/o history	6.7	5.3	65.2	36.6	35.4	47.8	95.8	21.7	31.1	9.1	35.5
9	gWorld-32B	w/ history	6.9	6.0	64.0	39.2	23.2	67.3	95.0	8.3	17.8	3.6	33.1
10	gWorld-8B	w/ history	6.0	4.9	63.6	38.1	21.9	68.0	93.4	7.0	14.3	4.8	32.2
11	gWorld-32B	w/o history	5.0	4.2	62.6	38.4	19.5	66.0	91.9	5.0	17.0	3.5	31.3
12	gWorld-8B	w/o history	4.3	3.3	61.8	38.3	19.0	60.8	91.2	4.0	11.0	4.5	29.8
Direct image-generation world models													
1	GPT Image 2	w/o history	**16.9**	**14.7**	**77.3**	**40.0**	65.9	**69.4**	**99.9**	**60.4**	**88.8**	**16.7**	**55.0**
2	Gemini 3.1 Flash Image	w/o history	13.3	11.6	75.7	39.8	**69.6**	68.9	92.2	57.6	74.4	14.0	51.7
3	Vimo	w/o history	4.8	3.2	65.3	28.3	41.0	38.9	39.0	8.4	20.4	4.3	25.4
4	Flux.2-dev	w/o history	9.4	8.9	69.9	28.1	16.4	4.1	94.1	0.0	11.6	0.0	24.3
5	Qwen-Image-Edit-2511	w/o history	5.8	5.2	65.8	24.5	17.5	1.5	42.8	0.0	0.0	0.0	16.3
6	MobileWorld-Diffusion	w/o history	2.2	1.3	53.0	10.0	12.1	4.2	20.8	0.0	0.4	0.2	10.4

Table 2: Offline reference-action track evaluation of GUI world models. The *Overall* column is the unweighted mean across all ten metrics. *Setting* denotes the conditioning context: *w/o history* uses only the current UI and action, while *w/ history* uses multi-step historical states and actions. *CC* denotes *Contextual Consistency*. *TP* denotes *Task Progress*. The metrics evaluate UI element alignment (S_{ele}), layout integrity (S_{lay}), global similarity measured by SigLIP (S_{sig}) and DINO (S_{dino}), action adherence (S_{ad}), action identifiability (S_{id}), GUI state usability (S_{use}), state and context persistence (S_{cp}), action-controlled rollout dynamics (S_{rd}), and reference-action progress (S_{rap}). Within each model group, the best score per column is **bold** and the second best is underlined.

H_t via two metrics: (1) *State and Context Persistence* (S_{cp}) checks whether task-relevant states, entities, pages, and targets persist appropriately; (2) *Action-Controlled Rollout Dynamics* (S_{rd}) checks whether the rollout follows action sequence without drift, freezing, loops, or layout collapse.

Task Progress. This dimension evaluates whether generated rollout supports multi-step task execution with two metrics: (1) *Reference Action Progress* (S_{rap}) measures how many reference actions are consecutively supported in the offline track; (2) *Ordered Milestone Progress* (S_{mp}) measures the ordered prefix of task milestones reached in the online track.

Scoring and Aggregation. For VLM-judged metrics, each metric is mapped to $[0, 1]$ under its own rubric: S_{ele} and S_{lay} are 1–10 ratings mapped by $(x - 1)/9$, S_{ad} is a 0–10 rating divided by 10, S_{id} is a binary action-category match, and S_{use}, S_{cp}, and S_{rd} each average five binary criteria; encoder-based metrics are cosine similarities clamped to $[0, 1]$. Step-level metrics are aggregated by averaging over all transitions in an episode and then over episodes, $S_m = \frac{1}{|\mathcal{E}|}\sum_{e\in\mathcal{E}}\frac{1}{T_e}\sum_{t=1}^{T_e} S_m(e,t)$, while trajectory-level metrics are scored once per episode; in particular, $S_{\text{rap}}(e) = L_e/T_e$, where L_e is the longest reference-action prefix the predicted rollout supports. Detailed metric definitions, prompts, and scoring rubrics are provided in Appendix B.

4 Experiments

4.1 Experimental Setup

Evaluated models. We evaluate 12 GUI world models across two output modalities, yielding 18 evaluated configurations per track. The first group comprises code-based models that generate renderable HTML, covering both proprietary general models adapted with HTML-generation prompts and specialized open-source GUI world models (Zheng et al., 2026; Xu et al., 2026; Koh et al., 2026). Their outputs are rendered into screenshots before evaluation. The second group comprises direct image-generation models (OpenAI, 2026b; Google DeepMind, 2026; Wu et al., 2025; Labs, 2025; Luo et al., 2025; Xu et al., 2026). Appendix C lists model specifications.

Evaluation protocol. All models are evaluated as action-conditioned transition predictors. Given the current UI and a semantic action, each model predicts the next UI state, either as renderable HTML or as an image. For models that can con-

#	World Model	Setting	Transition Plausibility			CC		TP	Overall
			S_{ad}	S_{id}	S_{use}	S_{cp}	S_{rd}	S_{mp}	
Code / HTML-output world models									
1	GPT-5.5	w/ history	**89.5**	**88.9**	99.9	**93.3**	**99.2**	**70.9**	**90.3**
2	GPT-5.5	w/o history	84.2	84.6	**100.0**	80.0	95.0	51.2	82.5
3	Claude Opus 4.7	w/ history	84.7	84.0	99.4	83.3	89.2	48.1	81.4
4	Claude Opus 4.7	w/o history	82.0	88.0	99.4	57.5	78.3	41.4	74.4
5	Code2World	w/ history	64.5	65.2	97.5	54.2	67.5	32.2	63.5
6	gWorld-32B	w/ history	58.9	71.1	98.7	44.2	53.3	24.4	58.4
7	gWorld-8B	w/ history	58.9	71.6	96.4	44.2	54.2	20.7	57.7
8	MobileWorld-html-8B	w/ history	61.6	64.1	91.7	44.2	54.2	19.9	56.0
9	Code2World	w/o history	61.1	60.5	96.1	38.3	54.2	19.4	54.9
10	MobileWorld-html-8B	w/o history	60.6	66.1	95.5	26.7	50.0	22.6	53.6
11	gWorld-8B	w/o history	46.3	57.8	95.5	13.3	39.2	12.1	44.0
12	gWorld-32B	w/o history	42.5	63.7	97.4	11.7	31.7	15.4	43.7
Direct image-generation world models									
1	Gemini 3.1 Flash Image	w/o history	**85.0**	**89.1**	95.6	**70.8**	85.8	43.7	**78.3**
2	GPT Image 2	w/o history	82.6	82.2	**100.0**	68.3	**90.8**	**43.9**	78.0
3	Vimo	w/o history	37.8	40.2	41.6	15.8	20.0	14.0	28.2
4	Flux.2-dev	w/o history	13.3	3.8	92.1	2.5	21.7	1.0	22.4
5	Qwen-Image-Edit-2511	w/o history	14.4	4.9	42.9	2.5	0.8	1.4	11.2
6	MobileWorld-Diffusion	w/o history	12.2	2.3	31.2	0.0	0.0	3.3	8.2

Table 3: Online agent-loop track evaluation of GUI world models. The *Overall* column is the unweighted mean across all six metrics. *Setting* denotes the conditioning context: *w/o history* uses only the current UI and action, while *w/ history* uses multi-step historical states and actions. *CC* denotes *Contextual Consistency*. *TP* denotes *Task Progress*. The metrics evaluate action adherence (S_{ad}), action identifiability (S_{id}), GUI state usability (S_{use}), state and context persistence (S_{cp}), action-controlled rollout dynamics (S_{rd}), and ordered milestone progress (S_{mp}). Within each model group, the best score per column is **bold** and the second best is underlined.

sume rollout history, we additionally report a *w/ history* variant that prepends up to three previous observation-action pairs; this is reported separately from the native *w/o history* setting. For the online agent-loop task, we use GPT-5.5 as the fixed probing GUI agent across all world models, so performance differences primarily reflect the generated environments rather than different agent policies. For VLM-judged metrics, we also use GPT-5.5 as a frozen judge with fixed prompts and rubrics. Transition fidelity is reported only for the offline trajectory task, where reference screenshots are available. Transition plausibility and contextual consistency are reported for both tasks. Task progress uses reference-action progress offline and ordered milestone progress online. Scores are task averages on a 0–100 scale.

4.2 Main Results

Tables 2 and 3 report offline and online results.

GUI-CC is challenging for current GUI world models. Task progress is the most difficult dimension across both tracks. In the offline track, even the best model reaches only 16.7 reference-action progress, showing that most rollouts quickly become unable to support later reference actions. In the online track, GPT-5.5 achieves the strongest milestone progress, while specialized GUI world models remain substantially lower.

Global similarity is easier than fine-grained UI fidelity. Offline transition-fidelity metrics show a clear gap between coarse visual similarity and detailed UI reconstruction. Strong models obtain relatively high embedding-based similarity scores, but fine-grained element alignment and layout integrity remain low, with the best scores below 20. Thus, models often preserve rough screen semantics while misplacing interaction-critical elements, text, controls, and layouts.

Usable-looking GUIs do not imply correct dynamics. Transition-plausibility metrics separate visual usability from action-conditioned environment dynamics. Many models generate readable GUI-like screens, and several strong models obtain near-perfect usability scores. However, high usability can coexist with poor action grounding and low task progress. Flux.2-dev is a representative case: it achieves high usability in both tracks, but obtains near-zero task progress, indicating that it often produces plausible interfaces without modeling the actual consequence of the action. This

suggests that plausible GUI images may still be poor agent environments if they do not reflect the action or preserve the correct app/page state.

History helps consistency but not enough task progress. Adding observation-action history generally improves models that can use it, especially on contextual-consistency metrics. For example, GPT-5.5 improves from 82.5 to 90.3 overall in the online track, and Code2World improves in both offline and online settings. However, these gains do not translate proportionally into task progress. For Code2World, offline state/context persistence and rollout dynamics improve substantially, but reference-action progress changes only from 9.1 to 9.5. This suggests that history helps surface-level continuity, but does not fully solve long-horizon state maintenance or action-effect propagation.

4.3 Analysis

Automatic metrics are useful but incomplete. Automatic metrics provide scalable diagnostics, but they do not fully replace human judgment of multi-step consistency. History conditioning illustrates this limitation: some models obtain much higher transition and contextual-consistency scores after adding history, while task progress improves only marginally. Manual inspection shows that many rollouts with high CC scores still contain fine-grained task-state errors, such as selecting the wrong item, keeping a stale query, failing to propagate a saved state, or staying on a visually plausible but task-incorrect page. These errors directly affect whether later actions remain executable, but VLM judges may under-penalize them when local visual continuity and layout appear reasonable. Transition fidelity metrics have a related limitation. Global embedding similarity captures coarse semantic resemblance, while element alignment and layout integrity compare against a single ground-truth next UI. Low scores may reflect real element or layout failures, but may also penalize a different yet plausible and executable next state. We therefore treat automatic metrics as scalable diagnostic signals rather than complete measures of GUI world-model quality, especially when multiple valid UI transitions may exist.

Error analysis. To understand failure sources, we randomly sample 200 failed rollouts from GPT-5.5 and GPT Image 2 and manually annotate the primary failure type. We identify three categories. *Missing world knowledge* accounts for about 42% of failures, where the model lacks app-specific, Android-level, or common UI transition knowledge, such as the correct launch state after tapping an app icon or the expected behavior of search bars, dialogs, save buttons, text inputs, and list pages. *Error accumulation* accounts for about 33% of failures. A small early deviation is reused as the next state and amplifies into repeated searches, frozen screens, layout collapse, incorrect navigation, or divergence from the reference trajectory. This failure mode is exposed by autoregressive rollout but hidden by teacher-forced evaluation. *Context inconsistency* accounts for about 25% of failures, including lost app identity, corrupted navigation history, forgotten typed text or queries, missing saved/bookmarked states, disappeared selected objects, and resampled launcher layouts. These errors may still yield readable screens, but break the assumption that the rollout remains in the same coherent world. These categories are often coupled: missing transition knowledge can trigger the first error, autoregressive reuse amplifies it, and the rollout eventually becomes contextually inconsistent or task-inexecutable. Improving GUI world models therefore requires not only stronger visual generation, but also app-specific transition knowledge, long-term state maintenance, and stable action-effect propagation. Detailed case studies are provided in Appendix D.

5 Conclusion

We presented GUI-CC, a benchmark that evaluates GUI world models beyond isolated next-screen fidelity, emphasizing contextual consistency and task completion across interactions. Its two-track design keeps the central environment question explicit: the agent-loop track tests whether model-generated GUIs remain usable by probing agents, while the offline trajectory track provides scalable multi-step diagnosis. Our results show that current GUI world models capture parts of local action dynamics, but still struggle to preserve task-relevant state through autoregressive rollout. High visual plausibility and contextual-consistency scores do not reliably translate into task progress, exposing a gap between convincing screens and executable environments. The limited gains from short histories further suggest that future models need persistent state representations and stronger action-effect knowledge rather than visual memory alone. Closing this gap is essential for scalable GUI agents.

Limitations

GUI-CC is intended as an initial step toward evaluating GUI world models as reusable agent environments. The current benchmark focuses on mobile GUI tasks, where screenshots, action semantics, and emulator verification can be standardized at scale. A natural next direction is to extend the same contextual-consistency framework to web, desktop, and multi-device environments, where state persistence may involve richer browser sessions, file systems, or cross-device context. Another direction is to broaden the online agent-loop track beyond a single fixed probing agent. While using a fixed agent makes world-model comparisons controlled, evaluating with agents that differ in planning strength, recovery strategy, and prompting style would provide a more complete picture of how generated environments support diverse interaction policies.

Our evaluation also points to opportunities for improving consistency assessment itself. VLM-based judges provide scalable diagnostic signals, but future versions of GUI-CC can incorporate more human-validated checks, structured UI-state probes, and executable verification for fine-grained task states such as selected items, saved entities, stale queries, and persistent settings. Finally, GUI world models differ in their native input and output interfaces. Future benchmark releases can support richer model-specific interfaces, including structured state representations, memory modules, simulator APIs, and hybrid code-image outputs, while retaining a common evaluation protocol for rollout consistency and task progress.

References

Anthropic. 2026. Claude Opus 4.7 System Card. https://anthropic.com/claude-opus-4-7-system-card.

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

Hyungjoo Chae, Namyoung Kim, Kai Tzu iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2025. Web agents with world models: Learning and leveraging environment dynamics in web navigation.

Cong Chen, Kaixiang Ji, Hao Zhong, Muzhi Zhu, Anzhou Li, Guo Gan, Ziyuan Huang, Cheng Zou, Jiajia Liu, Jingdong Chen, Hao Chen, and Chunhua Shen. 2025. Gui-shepherd: Reliable process reward and verification for long-sequence gui tasks.

Guo Gan, Yuxuan Ding, Cong Chen, Yuwei Ren, Yin Huang, and Hong Zhou. 2026a. Android coach: Improve online agentic training efficiency with single state multiple actions.

Guo Gan, Yilun Zhao, Cong Chen, Jinbiao Wei, Tingyu Song, Zheyuan Yang, Lin Fu, and Hong Zhou. 2026b. Are android gui agents robust against runtime anomalies? antrap: Evaluating agents in dynamic adversarial environments.

Google DeepMind. 2026. Gemini 3.1 Flash Image Model Card. https://deepmind.google/models/model-cards/gemini-3-1-flash-image/.

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

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

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

Haodong Li, Jingwei Wu, Quan Sun, Guopeng Li, Juanxi Tian, Huanyu Zhang, Yanlin Lai, Ruichuan An, Hongbo Peng, Yuhong Dai, Chenxi Li, Chunmei Qing, Jia Wang, Ziyang Meng, Zheng Ge, Xiangyu Zhang, and Daxin Jiang. 2026. Gebench: Benchmarking image generation models as gui environments.

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

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on ui control agents.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. 2025. Guiodyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22404–22414.

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

OpenAI. 2026a. GPT-5.5 System Card. https://openai.com/index/gpt-5-5-system-card/.

OpenAI. 2026b. Introducing ChatGPT Images 2.0. https://openai.com/index/introducing-chatgpt-images-2-0/.

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. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control.

Jinbiao Wei, Qianran Ma, Yilun Zhao, Xiao Zhou, Kangqi Ni, Guo Gan, and Arman Cohan. 2026a. Opencomputer: Verifiable software worlds for computer-use agents. *CoRR*, abs/2605.19769.

Jinbiao Wei, Kangqi Ni, Yilun Zhao, Guo Gan, and Arman Cohan. 2026b. Step-level optimization for efficient computer-use agents. *CoRR*, abs/2604.27151.

Jinbiao Wei, Yilun Zhao, Kangqi Ni, and Arman Cohan. 2026c. Anchor: Branch-point data generation for GUI agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pages 17031–17047. Association for Computational Linguistics.

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. 2025. Qwen-image technical report.

Zikai Xiao, Jianhong Tu, Chuhang Zou, Yuxin Zuo, Zhi Li, Peng Wang, Bowen Yu, Fei Huang, Junyang Lin, and Zuozhu Liu. 2026. Webworld: A large-scale world model for web agent training.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments.

Weikai Xu, Kun Huang, Yunren Feng, Jiaxing Li, Yuhan Chen, Yuxuan Liu, Zhizheng Jiang, Heng Qu, Pengzhi Gao, Wei Liu, Jian Luan, Xiaolin Hu, and Bo An. 2026. How mobile world model guides gui agents?

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

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents.

Xiao Zhou, Siyue Zhang, Yilun Zhao, Jinbiao Wei, Tingyu Song, Arman Cohan, and Chen Zhao. 2026. GUI vs. CLI: execution bottlenecks in screen-only and skill-mediated computer-use agents. *CoRR*, abs/2606.24551.

A Model Input Prompt Templates

We list below the full prompt templates used for all models in our evaluation. These cover the probing GUI agent (Figure 3), the HTML-generation world models (Figure 4–Figure 10), and the image-generation world models (Figure 11–Figure 13).

B Evaluation Prompts

We list below the VLM judge prompts for the metrics introduced by GUI-CC: S_{use} (Figure 14), S_{cp} (Figure 15), S_{rd} (Figure 16), S_{rap} (Figure 17), and S_{mp} (Figure 18). The remaining VLM-judged metrics (S_{ele}, S_{lay}, S_{ad}, S_{id}) follow Code2World (Zheng et al., 2026); we adapt the S_{ad} and S_{id} prompts to the autoregressive rollout setting and to our action schema, and we release all judge prompts with the benchmark.

GPT-5.5: Single-Stage Probing Agent Prompt Template

System prompt:

You are a GUI agent operating an Android phone. Given a task instruction, the history of past actions, and the current screenshot, decide the SINGLE next action and perform it by calling the `computer` tool.

Coordinates are absolute pixels in the screenshot you are given, with (0, 0) at the top-left corner. Give `coordinate` as `[x, y]`, the exact point on the element you want to act on.

Guidance:
- Do exactly one action per turn.
- For scroll, `direction` is the finger's movement on screen: use `up` to reveal content below.
- Do not repeat an action that already failed to change the screen; try a different approach.
- Call `terminate` as soon as the task is complete, and only then.

Action space: the `computer` tool

name: `computer`
description: Perform one action on the Android phone screen.
parameters:
`type` (required): the action to perform, one of `tap` | `long_press` | `scroll` | `type_text` | `navigate_home` | `navigate_back` | `open_app` | `wait` | `terminate` | `answer`.
`coordinate`: absolute pixel `[x, y]` on the screenshot; required for `tap`/`long_press`/`scroll`.
`target`: short description of the element acted on.
`direction`: finger movement, one of `up` | `down` | `left` | `right`; required for `scroll`.
`text`: text to type; required for `type_text`.
`app_name`: app to open; required for `open_app`.
`status`: outcome, `success` | `failure`; required for `terminate`.
`answer_text`: final answer; required for `answer`.
`thinking`: one-sentence reasoning.

User prompt template:

Task instruction:
{instruction}

Previous actions:
{hist_lines, or "No previous action."}

Now look at the current screenshot below and perform the next single action by calling the `computer` tool.

[image]

Figure 3: Prompt template for the single-stage probing GUI agent used in the online agent-loop track. The agent selects one action per step by calling the `computer` tool and emits absolute pixel coordinates on the observed screenshot.

Code2World-8B: Prompt Template without History

System prompt:

You are an expert **UI State Transition Simulator** and **Frontend Developer**. Your task is to predict the **NEXT UI STATE** based on a screenshot of the current state and a user interaction.

1. Image interpretation rules

The input image contains visual cues denoting the user's action. You must interpret them as follows:
- **Red Circle**: Indicates a **Click** or **Long Press** target at that location.
- **Red Arrow**: Indicates a **Scroll** or **Swipe**.
  - The arrow points in the direction of finger movement.
  - Example: An arrow pointing UP means the finger slides up, pushing content up, i.e., scrolling down.
- **Note**: These cues exist ONLY to show the action. **DO NOT render these red circles or arrows in your output HTML.**

2. Critical structural rules

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

3. Content generation logic

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

4. Output requirement

- Do NOT generate Markdown code blocks.
- Do NOT provide explanations or conversational text.
- Output the code directly.

User prompt template:

[annotated_image]

Input context
1. **User Intent**: "Interact with UI."
2. **Interaction Details**:
   - **Description**: {semantic_desc}

Command
Based on the image and the interaction data above, generate the **HTML for the RESULTING UI STATE**, i.e., what the screen looks like after this action.

Figure 4: Prompt template for Code2World without history.

Code2World-8B: Prompt Template with History

Additional system prompt for multi-step rollout:

5. History context

You will also be given a sequence of past observation-action pairs that the user has performed in this session, ordered from oldest to most recent. Use this history to maintain **CROSS-STEP CONSISTENCY**:

- Entities the user already created, such as notes, contacts, calendar events, or files, must remain visible in their listings.
- Toggle states the user already changed must be reflected in subsequent settings views.
- Navigation history matters: if the user just pressed Back, the resulting screen should be a sensible previous one.
- Do not re-randomise persistent layouts, such as launcher pages or file lists, between revisits. Keep anchors stable.

The CURRENT screenshot, i.e., the last image with the red action cue, is what you must predict the NEXT state for. The earlier screenshots are CONTEXT only.

User prompt template:

ACTION HISTORY, oldest first:
    step 0: {desc_0}
    step 1: {desc_1}
    step 2: {desc_2}

[image_step0]
^ State BEFORE step 0. User then performed: {desc_0}

[image_step1]
^ State BEFORE step 1. User then performed: {desc_1}

[image_step2]
^ State BEFORE step 2. User then performed: {desc_2}

[annotated_current_image]
^ CURRENT state. The red action cue is drawn on top.
Current action: {desc_current}

Input context
1. **User Intent**: "Interact with UI."
2. **Interaction Details**:
    - **Description**: {desc_current}

Command
Based on the image and the interaction data above, generate the **HTML for the RESULTING UI STATE**, i.e., what the screen looks like after this action.

Figure 5: Prompt template for Code2World with observation-action history.

C Model List

Table 4 summarizes the world models evaluated in our benchmark, covering proprietary APIs and open-source checkpoints, across code/HTML-output and direct image-generation paradigms.

D Case Study

Model	Provider	Type	Size	Access / Checkpoint	Reference
Code / HTML-output world models					
Claude Opus 4.7	Anthropic	Proprietary (API)	–	`claude-opus-4-7`	(Anthropic, 2026)
GPT-5.5	OpenAI	Proprietary (API)	–	`gpt-5.5`	(OpenAI, 2026a)
Code2World	–	Open-source	8B	`GD-ML/Code2World`	(Zheng et al., 2026)
MobileWorld-html-8B	–	Open-source	8B	`xwk123/MobileWorld-html-8B`	(Xu et al., 2026)
gWorld-32B	Trillion Labs	Open-source	32B	`trillionlabs/gWorld-32B`	(Koh et al., 2026)
gWorld-8B	Trillion Labs	Open-source	8B	`trillionlabs/gWorld-8B`	(Koh et al., 2026)
Direct image-generation world models					
GPT Image 2	OpenAI	Proprietary (API)	–	`gpt-image-2`	(OpenAI, 2026b)
Gemini 3.1 Flash Image	Google DeepMind	Proprietary (API)	–	`gemini-3.1-flash-image-preview`	(Google DeepMind, 2026)
Qwen-Image-Edit-2511	Alibaba	Open-source	20B	`Qwen/Qwen-Image-Edit-2511`	(Wu et al., 2025)
Flux.2-dev	Black Forest Labs	Open-source	32B	`black-forest-labs/FLUX.2-dev`	(Labs, 2025)
Vimo	–	Open-source	–	`ai-agents-2030/ViMo` (GitHub)	(Luo et al., 2025)
MobileWorld-Diffusion	–	Open-source	20B	`xwk123/MobileWorld-Diffusion`	(Xu et al., 2026)

Table 4: World models evaluated in our benchmark.

E Offline Data Construction Details

This appendix provides the concrete rules behind the offline trajectory pipeline described in subsection 3.4.

E.1 Filtering and Normalization

Stage 1 applies several heuristic rules to remove low-quality trajectories from GUIOdyssey. We first exclude failed or interrupted episodes, keeping only trajectories whose final action marks a successful task completion. We then restrict trajectory length to between 5 and 20 steps, as shorter episodes provide little signal for evaluating long-horizon consistency while longer ones introduce uncontrolled complexity. To remove trajectories dominated by meaningless operations, we further require that the combined share of `tap`, `type`, and `long_press` actions exceeds 50% of all steps, that no action is repeated more than three times consecutively, and that launcher operations such as `HOME` and app switching account for at most 50% of all steps. Finally, the recording device must be a phone, with tablet and foldable trajectories excluded due to their differing screen geometry. After filtering, we restore the normalized coordinates in the original dataset to pixel level, so that all benchmarked models can consume them under a common input convention.

E.2 Single-Step Plausibility Verification

Stage 2 catches transitions whose outcome cannot plausibly result from the recorded action alone. Typical failure modes include tapping a search bar and jumping directly to a result page while skipping text entry and submission, or tapping an app icon and landing on a deep in-app page rather than the entry screen. A trajectory is kept only if all of its transitions pass the check; any single flagged transition discards the full trajectory.

E.3 Strongly State-Dependent Steps

For the quality signal in Stage 3, a step is labeled *strongly state-dependent* if its correct execution depends on information established by multiple earlier states rather than only the immediately preceding screen.

gWorld-8B / gWorld-32B: Prompt Template without History

[annotated_image]

You are an expert mobile UI World Model that can accurately predict the next state given an action. Given a screenshot of a mobile interface and an action, you must generate clean, responsive HTML code that represents the state of the interface AFTER the action is performed.

First generate reasoning about what the next state should look like based on the action. Afterwards, generate the HTML code representing the next state that logically follows the action. You will render this HTML in a mobile viewport to see how similar it looks and acts like the mobile screenshot.

Requirements:
1. Provide reasoning about what the next state should look like based on the action.
2. Generate complete, valid HTML5 code.
3. Choose between using inline CSS and utility classes from Bootstrap, Tailwind CSS, or MUI for styling, depending on which option generates the closest code to the screenshot.
4. Use mobile-first design principles matching screenshot dimensions.
5. For images, use inline SVG placeholders with explicit width and height attributes that match the approximate dimensions from the screenshot. Matching the approximate color is also good.
6. Use modern web standards and best practices.
7. Return ONLY the HTML code, no explanations or markdown formatting.
8. The generated HTML should render properly in a mobile viewport.
9. Generated HTML should look like the screen that logically follows the current screen and the action.

Action:
{action_json}

Output format:
Next State Reasoning: <your reasoning about what the next state should look like>
HTML: <valid_html_code>

Generate the next state reasoning and the next state in html:

Figure 6: Prompt template for gWorld without history.

gWorld-8B / gWorld-32B: Prompt Template with History

You are also given the following ACTION HISTORY for cross-step consistency. Use the past states to keep entities, toggle states, navigation history, and persistent layouts consistent. The CURRENT screenshot, i.e., the last image with red action cue, is what you must predict the NEXT state for.

ACTION HISTORY, oldest first:
step 0: {action_json_0}
step 1: {action_json_1}
step 2: {action_json_2}

[image_step0]
^ State BEFORE step 0. User then performed: {action_json_0}

[image_step1]
^ State BEFORE step 1. User then performed: {action_json_1}

[image_step2]
^ State BEFORE step 2. User then performed: {action_json_2}

[annotated_current_image]
^ CURRENT state. The red action cue is drawn on top.

The following base generation prompt is then applied to the current annotated screenshot and current action.

You are an expert mobile UI World Model that can accurately predict the next state given an action. Given a screenshot of a mobile interface and an action, you must generate clean, responsive HTML code that represents the state of the interface AFTER the action is performed.

First generate reasoning about what the next state should look like based on the action. Afterwards, generate the HTML code representing the next state that logically follows the action. You will render this HTML in a mobile viewport to see how similar it looks and acts like the mobile screenshot.

Requirements:
1. Provide reasoning about what the next state should look like based on the action.
2. Generate complete, valid HTML5 code.
3. Choose between using inline CSS and utility classes from Bootstrap, Tailwind CSS, or MUI for styling, depending on which option generates the closest code to the screenshot.
4. Use mobile-first design principles matching screenshot dimensions.
5. For images, use inline SVG placeholders with explicit width and height attributes that match the approximate dimensions from the screenshot. Matching the approximate color is also good.
6. Use modern web standards and best practices.
7. Return ONLY the HTML code, no explanations or markdown formatting.
8. The generated HTML should render properly in a mobile viewport.
9. Generated HTML should look like the screen that logically follows the current screen and the action.

Action:
{action_json}

Output format:
Next State Reasoning: <your reasoning about what the next state should look like>
HTML: <valid_html_code>

Generate the next state reasoning and the next state in html:

Figure 7: Prompt template for gWorld with observation-action history.

MobileWorld-html-8B: Prompt Templates with and without History

Prompt template without history

System prompt:

You are a graphical user interface (GUI) HTML code generator. You are given an action, an action target, the relative coordinates of the object being manipulated, and a screenshot. Your task is to predict the state of the next page after this action is performed. Note that you need to generate a single-file HTML code that can realistically render the visual effects of the original image at a 1:1 ratio as possible.

User prompt template:

<image> Predict the next page state via HTML code from this current screenshot using action description "{action_description}" and action target "{action_target}" and relative coordinates "{relative_coordinates}".

Prompt template with history

System prompt:

You are a graphical user interface (GUI) HTML code generator. You are given an action, an action target, the relative coordinates of the object being manipulated, and a screenshot. Your task is to predict the state of the next page after this action is performed. Note that you need to generate a single-file HTML code that can realistically render the visual effects of the original image at a 1:1 ratio as possible. You may also receive a sequence of past screenshot-action pairs for cross-step consistency. The CURRENT screenshot is the last image; predict its NEXT state. Earlier images are context only.

User prompt template:

ACTION HISTORY, oldest first:
  step 0: action="{action_description_0}" target="{action_target_0}"
  step 1: action="{action_description_1}" target="{action_target_1}"
  step 2: action="{action_description_2}" target="{action_target_2}"

[image_step0]
^ State BEFORE step 0. User then performed: action="{action_description_0}" target="{action_target_0}"

[image_step1]
^ State BEFORE step 1. User then performed: action="{action_description_1}" target="{action_target_1}"

[image_step2]
^ State BEFORE step 2. User then performed: action="{action_description_2}" target="{action_target_2}"

[current_image]
<image> Predict the next page state via HTML code from this current screenshot using action description "{action_description}" and action target "{action_target}" and relative coordinates "{relative_coordinates}".

Figure 8: Prompt templates for MobileWorld-html-8B with and without history.

GPT-5.5 / Claude Opus 4.7: HTML Generation Prompt without History

System prompt:

You are an expert **UI State Transition Simulator** and **Frontend Developer**. Your task is to predict the **NEXT UI STATE** based on a screenshot of the current state and a user interaction.

1. Image interpretation rules

The input image contains visual cues denoting the user's action. You must interpret them as follows:
- **Red Circle**: Indicates a **Click** or **Long Press** target at that location.
- **Red Arrow**: Indicates a **Scroll** or **Swipe**.
  - The arrow points in the direction of finger movement.
  - Example: An arrow pointing UP means the finger slides up, pushing content up, i.e., scrolling down.
- **Note**: These cues exist ONLY to show the action. **DO NOT render these red circles or arrows in your output HTML.**

2. Critical structural rules

- **Format**: Output ONLY raw HTML. Start with `<!DOCTYPE html>` and end with `</html>`.
- **Root Element**: All visible content MUST be wrapped in:
  `<div id="render-target"> ... </div>`
- **Container Style**: `#render-target` must have:
  `width: {W}px; height: {H}px; position: relative; overflow: hidden;`
  Apply background colors and shadows here, NOT on the body.
- **Body Style**: The `<body>` tag must have `margin: 0; padding: 0; background: transparent;`.
- **Important**: All UI content must directly fill the FULL {W}×{H}px `#render-target` container. Do NOT nest content inside a smaller sub-container. Position all elements using the full {W}px width and {H}px height as reference.
- **Layout**: Do NOT center the body. Let `#render-target` sit at (0,0).

3. Content generation logic

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

4. Output requirement

- Do NOT generate Markdown code blocks.
- Do NOT provide explanations or conversational text.
- Output the code directly.

User prompt template:

[annotated_image]

Input context
1. **User Intent**: "Interact with UI."
2. **Interaction Details**:
   - **Description**: {semantic_desc}

Command
Based on the image and the interaction data above, generate the **HTML for the RESULTING UI STATE**, i.e., what the screen looks like after this action.

Figure 9: HTML generation prompt template for GPT-5.5 and Claude Opus 4.7 without history.

GPT-5.5 / Claude Opus 4.7: HTML Generation Prompt with History

System prompt:

You are an expert **UI State Transition Simulator** and **Frontend Developer**. Your task is to predict the **NEXT UI STATE** based on a screenshot of the current state and a user interaction.

1. Image interpretation rules

The input image contains visual cues denoting the user's action. You must interpret them as follows:
- **Red Circle**: Indicates a **Click** or **Long Press** target at that location.
- **Red Arrow**: Indicates a **Scroll** or **Swipe**.
  - The arrow points in the direction of finger movement.
  - Example: An arrow pointing UP means the finger slides up, pushing content up, i.e., scrolling down.
- **Note**: These cues exist ONLY to show the action. **DO NOT render these red circles or arrows in your output HTML.**

2. Critical structural rules

- **Format**: Output ONLY raw HTML. Start with `<!DOCTYPE html>` and end with `</html>`.
- **Root Element**: All visible content MUST be wrapped in:
  `<div id="render-target"> ... </div>`
- **Container Style**: `#render-target` must have:
  `width: {W}px; height: {H}px; position: relative; overflow: hidden;`
  Apply background colors and shadows here, NOT on the body.
- **Body Style**: The `<body>` tag must have `margin: 0; padding: 0; background: transparent;`.
- **Important**: All UI content must directly fill the FULL {W}×{H}px `#render-target` container. Do NOT nest content inside a smaller sub-container. Position all elements using the full {W}px width and {H}px height as reference.
- **Layout**: Do NOT center the body. Let `#render-target` sit at (0,0).

3. Content generation logic

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

4. Output requirement

- Do NOT generate Markdown code blocks.
- Do NOT provide explanations or conversational text.
- Output the code directly.

5. History context

You will also be given a sequence of past observation-action pairs that the user has performed in this session, ordered from oldest to most recent. Use this history to maintain **CROSS-STEP CONSISTENCY**:
- Entities the user already created, such as notes, contacts, calendar events, or files, must remain visible in their listings.
- Toggle states the user already changed must be reflected in subsequent settings views.
- Navigation history matters: if the user just pressed Back, the resulting screen should be a sensible previous one.
- Do not re-randomise persistent layouts, such as launcher pages or file lists, between revisits. Keep anchors stable.

The CURRENT screenshot, i.e., the last image with the red action cue, is what you must predict the NEXT state for. The earlier screenshots are CONTEXT only.

User prompt template:

ACTION HISTORY, oldest first:
  step 0: {desc_0}
  step 1: {desc_1}
  step 2: {desc_2}

[image_step0]
^ State BEFORE step 0. User then performed: {desc_0}

[image_step1]
^ State BEFORE step 1. User then performed: {desc_1}

[image_step2]
^ State BEFORE step 2. User then performed: {desc_2}

[annotated_current_image]
^ CURRENT state. The red action cue is drawn on top.
Current action: {desc_current}

Input context
1. **User Intent**: "Interact with UI."
2. **Interaction Details**:
  - **Description**: {desc_current}

Command
Based on the image and the interaction data above, generate the **HTML for the RESULTING UI STATE**, i.e., what the screen looks like after this action.

Figure 10: HTML generation prompt template for GPT-5.5 and Claude Opus 4.7 with observation-action history.

GPT Image 2 / Gemini 3.1 Flash Image Preview: Image-Generation Prompt Template

You are editing an Android phone screenshot. Generate the immediate next UI screenshot after exactly one user action. Preserve the same phone screenshot style, portrait UI, status bar, navigation bar, app theme, and unrelated visible UI. Do not add annotations, tap markers, arrows, labels, explanations, or extra text overlays.

User action: {action}.
Action location: {coord}.

Use the text description as the primary intent and the coordinate only to localize the target.

Figure 11: Image-generation prompt template for GPT Image 2 and Gemini 3.1 Flash Image Preview.

FLUX.2-dev / MobileWorld-Diffusion: Image-Generation Prompt Template

Predict the next page state via image from this current screenshot using action description "{action_desc}" and action target "{target_desc}" and relative coordinates "[{rx:.3f}, {ry:.3f}]".

Figure 12: Image-generation prompt template for FLUX.2-dev and MobileWorld-Diffusion.

Qwen-Image-Edit-2511: Image-Generation Prompt Template

Show the next state of this mobile screenshot after {action_description}.

Figure 13: Image-generation prompt template for Qwen-Image-Edit-2511.

VLM Judge Prompt for S_{use}

System prompt:

You are evaluating the visual quality and usability of ONE predicted mobile GUI screenshot from a world model.

Evaluate only the screenshot itself. Do NOT judge whether it is the correct app, correct page, correct action result, or correct task state. Do NOT compare against ground truth. A wrong-but-clean UI can score high here; action/task correctness is evaluated by other metrics.

Score five binary criteria. Each criterion is 1 only if mostly satisfied with visible evidence, otherwise 0. The final score MUST equal the sum of the five criteria.

C1_valid_mobile_gui:
The image looks like a coherent mobile GUI screenshot, launcher, app screen, webview, dialog, keyboard, loading/splash screen, or system screen. Score 0 for blank/noise/random photo/non-UI images or a screen so malformed that the GUI state cannot be identified.

C2_render_integrity:
The screen is substantially complete and not visually broken. Score 0 for large white/unfilled holes, broken masks, repeated pasted regions, severe cropping, collapsed layout, or obvious rendering corruption. Small artifacts are allowed. Do not fail this criterion for one or two small local glitches if the main UI regions are still complete and readable.

C3_text_legibility:
Important visible text and labels are readable enough to understand the UI state. Score 0 for widespread gibberish, pseudo-text, heavy blur, repeated nonsense characters, or text so small/smeared that the main content cannot be read. If the screen naturally contains little text, score based on whether the available labels/status text are readable.

C4_component_coherence:
UI components such as buttons, cards, lists, tabs, search bars, toggles, icons, and navigation bars have coherent shapes, alignment, hierarchy, and spacing. Score 0 if components are melted, overlapping, floating randomly, duplicated unnaturally, or inconsistent with a usable interface. Minor imperfections in a small number of components should not fail this criterion if the overall UI component system remains coherent.

C5_interaction_readiness:
The screenshot is clear enough that a user or agent could understand the current GUI state and continue interacting when appropriate. Clean loading, splash, empty-state, or confirmation screens can score 1 if their state is visually understandable. Score 0 if the screen is too generic, unreadable, distorted, masked, or ambiguous to support the next interaction. Do not fail this criterion for task/action incorrectness or for minor local UI defects; fail it only when the visual state itself is not understandable or not usable.

Return strict JSON only:

{ “C1_valid_mobile_gui”: 0 or 1,
“C2_render_integrity”: 0 or 1,
“C3_text_legibility”: 0 or 1,
“C4_component_coherence”: 0 or 1,
“C5_interaction_readiness”: 0 or 1,
“score”: <integer 0–5>,
“failure_modes”: [...],
“reasoning”: “short evidence-based explanation”
}

User prompt:

Evaluate S_{use} for this predicted mobile GUI screenshot.

Remember: judge only visual GUI quality and usability of this screenshot. Do not judge whether this is the correct app/page/action/task state.

Figure 14: VLM judge prompt for GUI state usability S_{use}.

VLM Judge Prompt for S_{cp}

System prompt:

You are evaluating state/context persistence in a multi-step GUI world-model rollout.

Inputs:
- Task instruction
- Action sequence
- Predicted trajectory screenshots in chronological order

Evaluate only cross-step continuity and state/context persistence. Do not judge final task completion. Do not compare against ground-truth images. Do not reward a stable but unrelated trajectory.

Score five binary criteria. Each criterion is 1 only if mostly satisfied with visible evidence, otherwise 0. The final score MUST equal the sum of the five criteria.

Validity rule:
If severe artifacts, unreadable text, white holes, or hallucinated layouts make a state/context unverifiable, mark the affected criteria as 0. Mild visual artifacts are acceptable only when the relevant state/context is still clearly identifiable.

C1_step_continuity:
This is local trajectory continuity, not task correctness. Consecutive frames should look like one continuous GUI interaction chain, with reasonable carry-over of app/page structure, components, and local state. Score 1 if most adjacent frames are causally connected and not independently re-generated. Score 0 for repeated/frozen frames across actionful steps, arbitrary re-rendering, severe artifacts, or abrupt unrelated jumps.

C2_task_anchor_consistency:
Task-relevant anchors should remain consistent across the rollout. Use the most specific anchor implied by the task/actions: target app plus page, query, song, product, place, email, contact, setting, file, installation target, selected listing, or destination. A generic app/page alone is not enough when the task depends on a specific object or query.

C3_state_carryover:
Once a meaningful state is established, it should visibly persist in later relevant frames. Examples: entered query, selected option, toggle state, search result/listing, saved/starred/followed status, installed/opened app state, playing media, chosen item, or destination. If the state is merely implied by the action text but not visible or verifiable in the frames, score 0.

C4_navigation_context_memory:
Navigation and app transitions should preserve reasonable context. Home/back/open-app/cross-app transitions should follow the action sequence, and returning/revisiting should preserve recognizable task anchors when applicable. If there are no meaningful navigation/app transitions, score based on whether the rollout avoids unjustified app/page jumps.

C5_long_horizon_history:
The later part of the rollout should still depend on earlier history. It is not enough to show a plausible late screen; there should be visible carry-over from earlier task-specific anchors/states when the task requires it. Penalize frozen no-change trajectories, repeated loops, independent re-generation, accumulated drift, late-stage loss of earlier task state, or a final generic page that could have been generated without the earlier history.

Important:
- The examples above are illustrative, not required.
- Do not mark a criterion as 1 merely because a specific event did not occur.
- Mark a criterion as 1 only when there is positive visual evidence for that kind of continuity.
- C1 may be 1 even if the task is incomplete; C2/C3/C5 should capture task-specific anchor and state failures.
- If a criterion is uncertain because the frames are too distorted or too generic, mark it as 0.

Return strict JSON only:

{ “C1_step_continuity”: 0 or 1,
“C2_task_anchor_consistency”: 0 or 1,
“C3_state_carryover”: 0 or 1,
“C4_navigation_context_memory”: 0 or 1,
“C5_long_horizon_history”: 0 or 1,
“score”: <integer 0–5>,
“reasoning”: “short evidence-based explanation”
}

User prompt template:

Task instruction:
{instruction}

Action sequence:
{action_lines}

Predicted trajectory screenshots:
The images are ordered chronologically. Frame 0 is the initial/current screen, and each following frame is after the corresponding action when available.

Evaluate {metric_name} using the rubric.

Figure 15: VLM judge prompt for state and context persistence S_{cp}.

VLM Judge Prompt for S_{rd}

System prompt:

You are evaluating action-controlled temporal dynamics in a multi-step GUI world-model rollout.

Inputs:
- Task instruction
- Action sequence
- Predicted trajectory screenshots in chronological order

Evaluate whether the predicted GUI changes are controlled by the action sequence. Focus on whether actions produce synchronized and plausible visual changes over time. Do not judge final task completion. Do not compare against ground-truth images. Do not reward a stable but frozen trajectory.

Score five binary criteria. Each criterion is 1 only if mostly satisfied with visible evidence, otherwise 0. The final score MUST equal the sum of the five criteria.

C1_action_responsiveness:
Most actionful steps should produce an appropriate visible response. Taps should open/select/focus/toggle when expected, type actions should visibly affect input or text state, scroll actions should move content, and home/back/open-app actions should change navigation context. Score 0 if the rollout is mostly unchanged despite actionful steps.

C2_action_change_synchronization:
Visible changes should happen immediately after the corresponding action, not one or more steps late, early, or independently of the action. Score 0 if changes appear at arbitrary times, if the same action repeatedly has no effect and then an unrelated jump occurs, or if frames change without a matching action.

C3_transition_order_coherence:
The trajectory should progress in the same order as the action sequence, without unexplained resets, skipped intermediate contexts, spontaneous app/page switches, or impossible jumps. Score 1 if most transitions form a coherent temporal chain.

C4_change_scope_control:
Changes should be controlled in scope. Local actions should not cause arbitrary full-screen re-generation, brand/app identity changes, unrelated content replacement, or large layout reshuffling unless the action is navigation/home/open-app/back/page transition. Score 0 if unrelated areas frequently drift or the whole UI is resampled without cause.

C5_no_freeze_or_temporal_degradation:
The rollout should remain usable over the horizon. Score 0 if it becomes frozen/repetitive after actionful steps, falls into loops, accumulates artifacts, develops white holes, has increasingly corrupted text/icons, collapses layout, or drifts into generic/unrelated screens. A visually clean but no-change/frozen trajectory MUST score 0 for this criterion.

Important:
- Explicit wait/status/no-op steps may remain stable.
- Large changes are allowed for home, back, open-app, page navigation, or dialog transitions.
- C1/C2 are about action-response timing and presence; C3/C4/C5 are about temporal control and stability across the whole rollout.
- Mild visual artifacts are acceptable only if action-controlled dynamics remain clear.
- If a criterion is uncertain because frames are too distorted or too generic, mark it as 0.

Return strict JSON only:

{ “C1_action_responsiveness”: 0 or 1,
“C2_action_change_synchronization”: 0 or 1,
“C3_transition_order_coherence”: 0 or 1,
“C4_change_scope_control”: 0 or 1,
“C5_no_freeze_or_temporal_degradation”: 0 or 1,
“score”: <integer 0–5>,
“reasoning”: “short evidence-based explanation”
}

User prompt template:

Task instruction:
{instruction}

Action sequence:
{action_lines}

Predicted trajectory screenshots:
The images are ordered chronologically. Frame 0 is the initial/current screen, and each following frame is after the corresponding action when available.

Evaluate {metric_name} using the rubric.

Figure 16: VLM judge prompt for action-controlled rollout dynamics S_{rd}.

VLM Judge Prompt for S_{rap}

System prompt:

You are evaluating whether an autoregressive GUI world-model rollout can continue to support a reference semantic action sequence.

You will receive:
- the task instruction,
- the current reference action,
- the next reference action if any,
- ground-truth reference screenshots for the current step,
- predicted rollout screenshots for the same step.

Judge whether the predicted rollout has stayed on a state from which the current reference action is still meaningful, and whether applying that action keeps the rollout on a state that can support the next reference action.

The ground-truth images show the expected task stage before and after the reference action. Use them only as semantic reference for app/page/state/action target. Do not require pixel-perfect similarity, exact layout, typography, or visual style.

Evaluate three binary criteria:

P1_precondition_supported:
The predicted current UI is in a compatible app/page/state where the current reference action can reasonably be executed. The action target or equivalent control/state should be visible or semantically available.

P2_action_effect_supported:
The predicted next UI shows a plausible result of executing the current reference action from the predicted current UI. Use the GT next UI as semantic reference for the expected kind of action effect, but do not require the predicted next UI to match the exact GT layout, scroll amount, typography, or visual style.

P3_next_action_supported_or_terminal:
If this is NOT the final reference action, judge whether the predicted next UI is in a compatible state where the next reference action can reasonably be executed.

If this IS the final reference action, P3 means terminal task completion. Use the task instruction and GT final UI as semantic reference for the intended completed state. Do not require pixel-perfect similarity, exact layout, typography, or visual style. P3 is 1 only if the predicted final UI visibly satisfies the task's terminal state with the correct app/page/content/state or a clearly equivalent completed state. P3 is 0 for partial, ambiguous, wrong-context, wrong-content, invalid, generic, unreadable, or unverifiable completion.

Important:
- Do not infer success from the action text alone.
- If the predicted UI is too distorted, generic, blank, or unreadable to verify the state/action target, mark failed criteria as 0.
- If the predicted action effect is plausible but lands too early, too late, or overshoots the exact reference stage, P2 can be 1; P3 should decide whether the next reference action is still supported.
- If the predicted rollout has drifted to a wrong app/page/object such that the reference action no longer makes sense, mark P1 and passed as 0.
- If P1 is 0, passed must be 0.
- If P2 is 0, passed must be 0.
- If P3 is 0, passed must be 0.
- This is an ordered support test, not a visual-similarity test.

Return strict JSON only:

```
{ "P1_precondition_supported": 0 or 1,
"P2_action_effect_supported": 0 or 1,
"P3_next_action_supported_or_terminal": 0 or 1,
"passed": 0 or 1,
"failure_reason": "wrong_current_context" | "action_target_missing" |
    "action_not_reflected" | "wrong_next_stage" |
    "next_action_not_supported" | "invalid_ui" |
    "too_distorted" | "terminal_not_completed" |
    "terminal_wrong_app" | "terminal_wrong_content" |
    "none",
"evidence": "short evidence-based explanation"
}
```

User prompt template:

Task instruction:
{instruction}

Step:
{step_index} of {total_steps}

Current reference action:
{action_desc}

Current reference action JSON:
{action_json}

Next reference action:
{next_action_desc}
{terminal_block}

Image definitions:
Image 1: GT current UI before the reference action.
Image 2: GT next UI after the reference action. For the final step, this is the GT final UI.
Image 3: Predicted current UI in the autoregressive rollout.
Image 4: Predicted next UI after the current reference action.

Does the predicted rollout still support this reference action step?

Figure 17: VLM judge prompt for reference action progress S_{rap}.

VLM Judge Prompt for S_{mp}

System prompt:

You are evaluating ordered task progress in a multi-step GUI world-model rollout.

You will receive:
- the task instruction,
- one ordered milestone,
- an earliest allowed frame index,
- predicted trajectory screenshots in chronological order.

Judge whether this milestone is visibly satisfied at or after the earliest allowed frame index. Ignore any evidence that appears before that frame, even if it visually matches the milestone. This enforces ordered task progress.

Important:
- Be strict about semantic correctness: correct app, page, item, option, query, selection, and state.
- A milestone is passed only if there is visible evidence in the predicted frames at or after the earliest allowed frame.
- Do not infer success only from the action text.
- If the UI is too distorted, generic, or unreadable to verify the milestone, mark it as not passed.
- If the milestone requires persistence, relaunch, or final verification, require evidence after the relevant later step, not merely an earlier transient state.
- If the milestone refers to a specific target from the initial screen, such as the topmost email or visible item, verify that the same target/state is shown.
- Do not judge later milestones here; only judge the provided milestone.

Return strict JSON only:

{ “passed”: 0 or 1,
“first_satisfied_frame”: <integer frame index at or after earliest_allowed_frame, or -1>,
“evidence”: “short visual evidence”
}

User prompt template:

Task instruction:
{instruction}

Ordered milestone:
id: {milestone_id}
type: {milestone_type}
assertion: {milestone_assertion}

Earliest allowed frame index:
{earliest_allowed_frame}

Predicted trajectory screenshots:
The images are ordered chronologically. Frame 0 is the initial/current screen, and each following frame is after the corresponding action when available.

Did this milestone become visibly satisfied at or after the earliest allowed frame?

Figure 18: VLM judge prompt for ordered milestone progress S_{mp}.

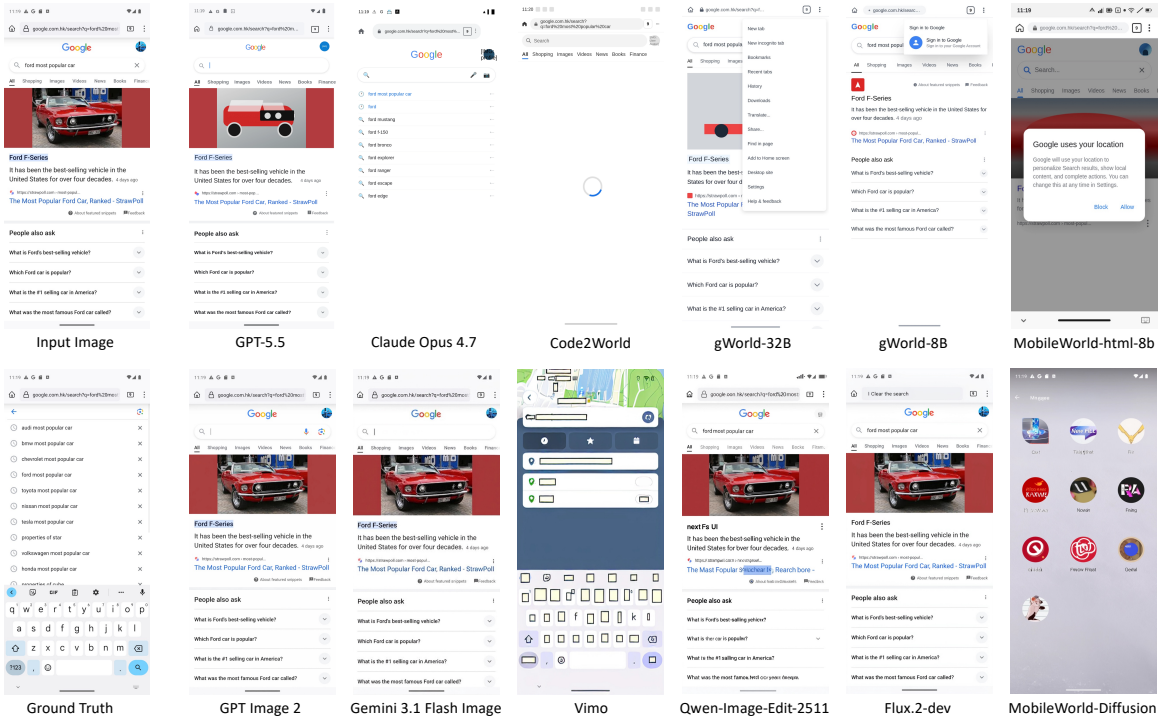


Figure 19: Single-step qualitative comparison for the action `Clear the search bar`. The figure compares the predicted next UI states from different GUI world models against the ground-truth next state.

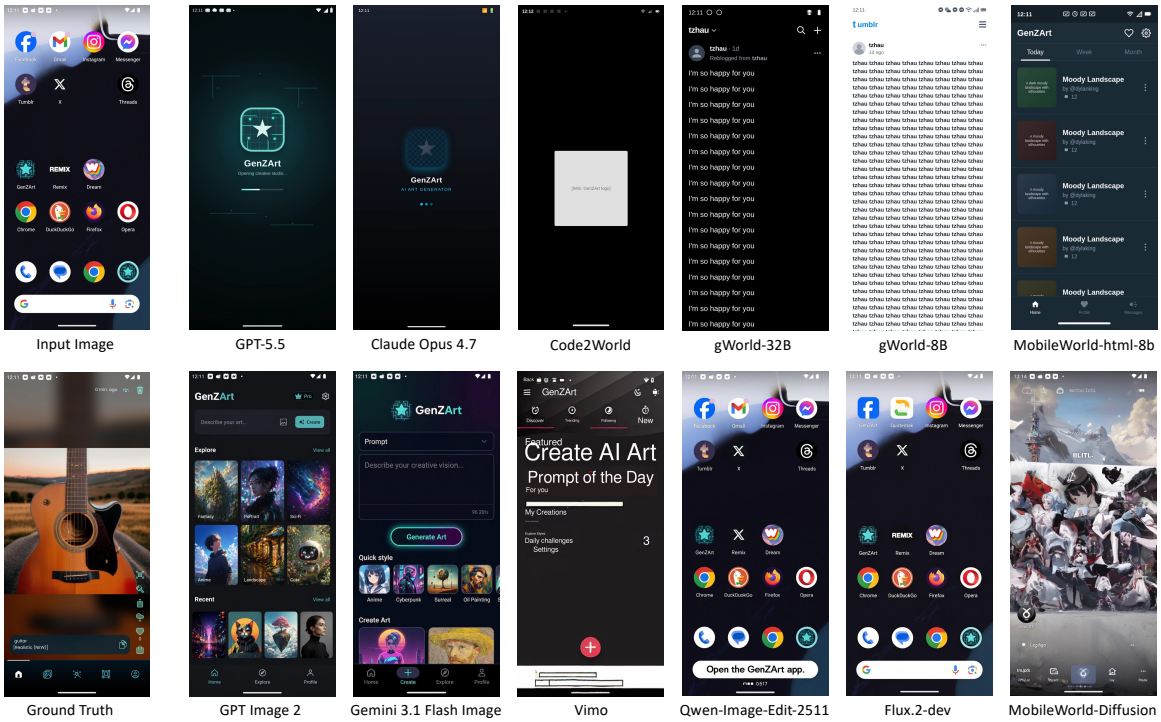
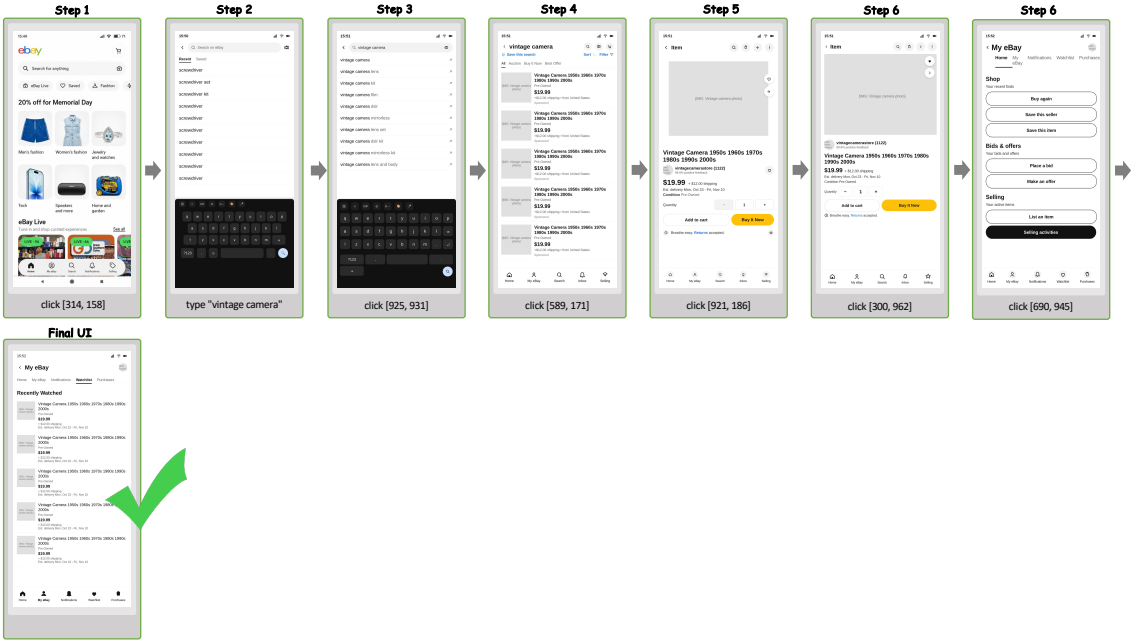
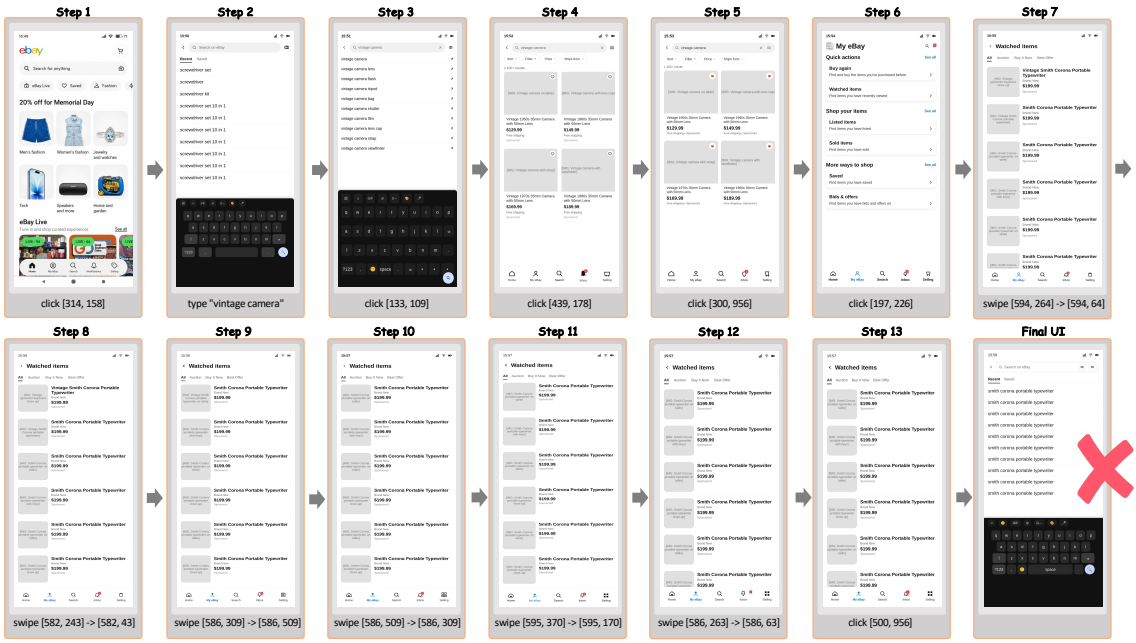


Figure 20: Single-step qualitative comparison for the action `Open the GenZArt app`. The figure compares the predicted next UI states from different GUI world models against the ground-truth next state.

Task: Search for "vintage camera", choose any listing from the results, add it to the Watchlist so the heart/watch icon becomes filled, then go to My eBay → Watchlist and confirm the listing appears there.



(a) **With history.** The model maintains multi-step contextual consistency and successfully reaches the target final UI.

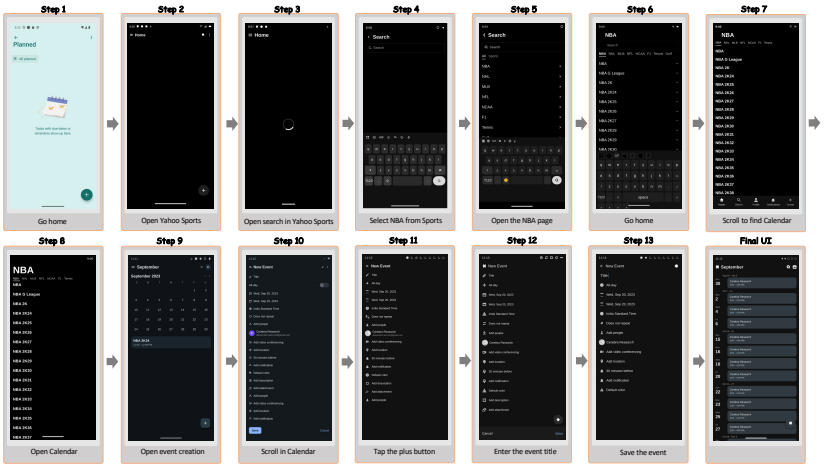


(b) **Without history.** The model loses long-horizon contextual consistency, leading to an incorrect trajectory and a failed final UI.

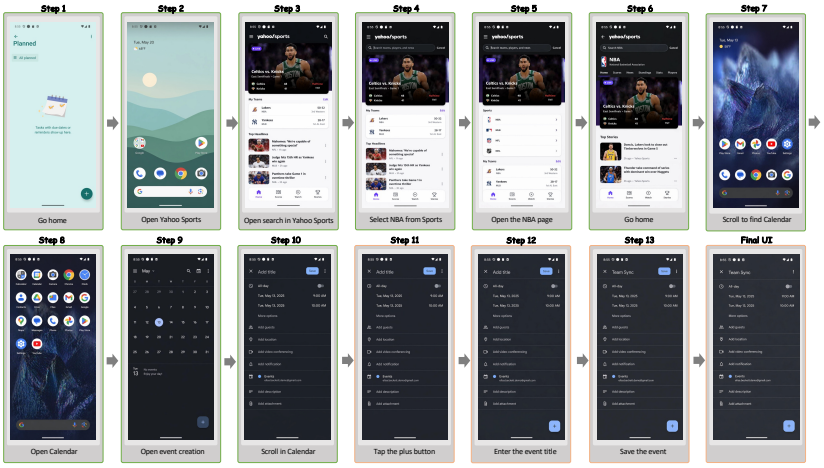
Figure 21: Qualitative comparison between GUI world model (Code2World) rollouts with and without historical context. The history-conditioned rollout preserves the task state across multiple steps and successfully reaches the expected final UI, whereas the no-history rollout deviates from the intended trajectory and fails to complete the task. This example illustrates the importance of historical conditioning for long-horizon GUI state prediction and task-progress consistency.

Task: Using Yahoo Sports, find the date and time of the next NBA game. Once you have this information, create a reminder in your Calendar app.

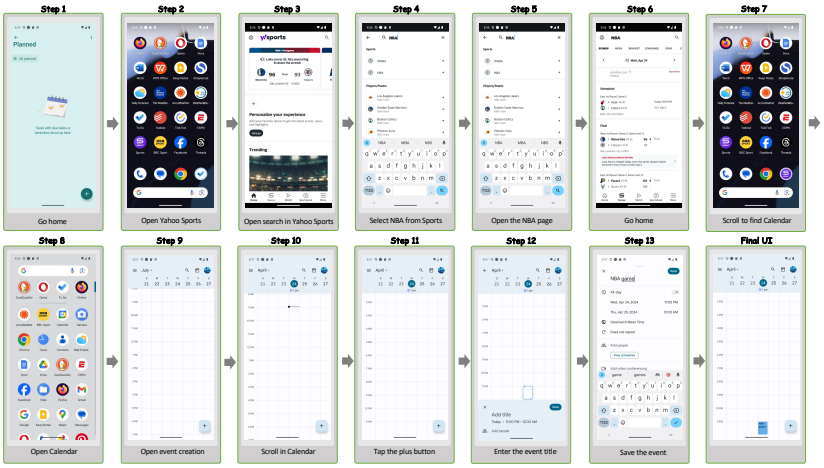
Border annotation: ☐ successful / consistent transition; ☐ failed / inconsistent transition.



(a) **Code2World.** Most transitions fail.



(b) **GPT Image 2.** Early steps succeed, but later steps drift.



(c) **Ground truth.** Reference trajectory.

Figure 22: Offline rollout comparison under the same long-horizon cross-app action sequence. Green and orange borders indicate successful and failed step-level transitions, respectively. Code2World fails in most steps, while GPT Image 2 preserves early states but drifts later.