WISH: A TRAINING-FREE WORLD-MODEL INVOCATION AND SELECTION HARNESS FOR GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents can use world models to compare the predicted outcomes of candidate actions, but the evidence they need varies across screens and candidate actions. Existing methods commonly provide the agent with one world model per run and use a fixed pipeline or external rule to decide when to call it. We introduce WISH, a training-free harness that lets a frozen GUI agent autonomously decide whether to request world-model predictions and which world model each candidate action needs. To guide these decisions, we abstract recurring evidence needs from teacher explanations into conditions that the agent can assess using the current GUI and agent's candidate actions, and combine this guidance with supporting components in WISH. Before requesting a world model's predictions, the agent judges whether those predictions could change its choice among candidate actions. If so, it selects a world model to provide the information each candidate needs. On AndroidWorld, WISH achieves 50.0% task success with Qwen3-VL-8B, exceeding the strongest reproduced baseline by 7.8 percentage points. Beyond accuracy, WISH only requests world-model predictions on 21.7% of steps, compared with 23.1% for that baseline, and improves task success over the evaluated baselines across two GUI agents and three benchmarks.

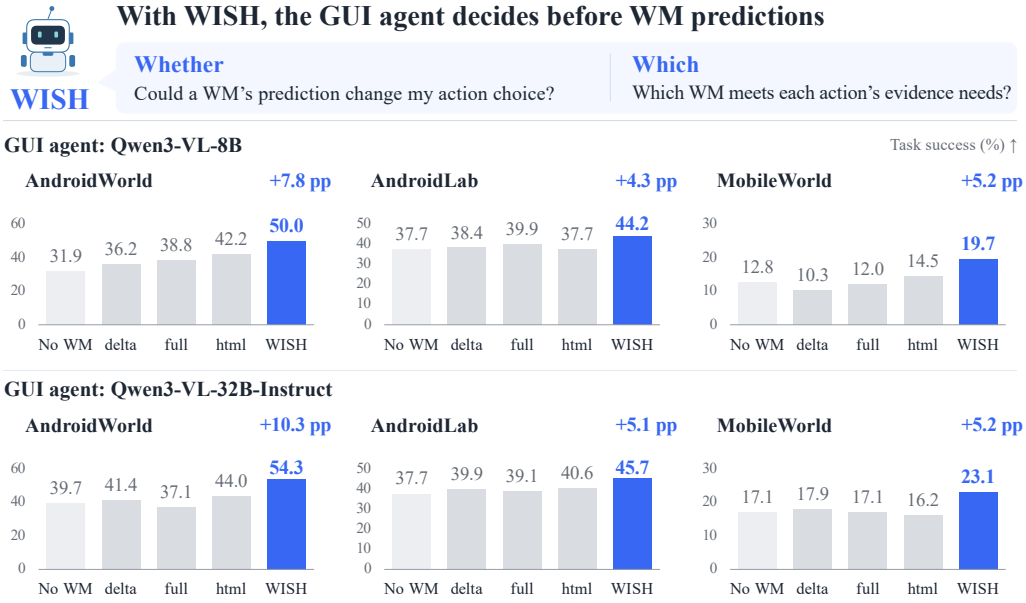

Figure 1: WISH lets the GUI agent autonomously decide whether to request world-model predictions and which world model each candidate action needs before prediction. Grey bars show the no-world-model baseline and entropy-gated Mobile World Model (MWM) self-reflection with delta, full, or html; blue bars show WISH. Blue annotations show gains in percentage points over each setting's strongest baseline.

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.

1 INTRODUCTION

GUI agents choose an action before observing its consequences, so a mistaken deletion, submission, edit, navigation step, or file operation can disrupt later steps (Lee et al., 2026; Guan et al., 2026; Yu et al., 2026). A world model may reduce this risk by predicting the next interface after each candidate action, allowing the agent to compare alternatives before acting (Chae et al., 2024; Gu et al., 2025; Luo et al., 2026). GUI world models predict the next interface as local changes (delta) (Chae et al., 2024; Li et al., 2025), complete semantic states (full) (Cao et al., 2026; Xiao et al., 2026), renderable interface code (html) (Zheng et al., 2026; Koh et al., 2026), or images (diffusion) (Luo et al., 2026; Rivard et al., 2026). These world models provide different information about the next interface, so the most useful model depends on the current interface and the candidate action (Xu et al., 2026b). To improve GUI agent performance, most existing methods use a single world model and follow a fixed pipeline. Static gating metrics, such as model entropy, determine when to call the world model (Chae et al., 2024; Gu et al., 2025; Zheng et al., 2026; Xu et al., 2026b, see Appendix A). To decide when to request world-model predictions, the agent must judge whether they would help it predict its next action. In our preliminary experiments, however, the GUI agent can propose the same wrong action across repeated samples, keeping the entropy gate closed (Section 2.2). These behaviours may stem from the GUI agent's overconfidence, which manifests in the agent almost never calling a world model (Section 2.2). These observations lead to our central questions:

> ***RQ1:*** *How can a GUI agent* ***autonomously decide*** *whether to use a world model?*
> ***RQ2:*** *If so,* ***which model to use*** *for each candidate action?*

To answer these questions, we investigate world-model use in GUI tasks and identify three insights (Sections 2.2 and 2.3). First, the GUI agent can remain confident in wrong actions. When we directly ask it whether to call a world model and which one to use, it declines most calls under some prompts but requests predictions on most steps under others. These observations show why the agent needs guidance for autonomous world-model use. Second, world-model predictions improve task success over the no-world-model baseline, with selective calls providing further gains in some configurations. These comparisons establish the value of world models at test time and the importance of deciding when to call them. Third, different world models succeed on partly different sets of tasks. By comparing their predictions, a separate teacher model recovers tasks missed by runs that use one world model throughout. In aligned traces, the teacher also favours different world models for different candidate actions on the same screen, supporting model choice for each action.

These insights motivate WISH, a training-free **W**orld-model **I**nvocation and **S**election **H**arness that helps a frozen GUI agent autonomously decide *whether* to request predictions and *which* world model each candidate action needs (Figure 1). To develop WISH, we use traces in which a frontier model serving as a teacher and evaluates each candidate action using each world model's prediction and explains why it accepts or rejects that action. By comparing the teacher's explanations for the same state and action, we identify when the current GUI settles the action choice, when predictions could help choose between plausible actions, and which world model each candidate needs. We encode these conditions in fixed prompts and checks that guide the agent through separate whether and which stages using the goal, current screen, history, and prepared candidate actions. At test time, the frozen GUI agent first judges *whether* a world model's prediction could change its choice among candidate actions and skips predictions when the current screen already makes the next action clear. If it requests predictions, it then decides *which* world model to use for each candidate, depending on whether it needs to check a local change, the full content of the next screen, or its layout. The agent makes both decisions autonomously before any world model generates a prediction. WISH supports these decisions by preparing candidate actions, guiding action selection, checking actions before execution, and using real execution outcomes to update memory and support recovery (Figure 2).

Our experiments examine task performance, the agent's whether and which decisions, and the components that support these decisions. WISH achieves higher task success than the baselines across GUI agents and benchmarks. With Qwen3-VL-8B on AndroidWorld, it achieves 50.0% task success, exceeding the strongest reproduced baseline by 7.8 percentage points, and requests world-model predictions on 21.7% of steps. Our contributions are:

1. We investigate world-model use in GUI tasks and identify three insights: the tested GUI agent needs guidance to make autonomous calls, useful predictions can change its action choice, and different world models provide different information to guide that choice.
2. Based on these insights, we introduce WISH, to our knowledge the first training-free harness that lets a frozen GUI agent autonomously decide whether to call a world model and which of several frozen world models each candidate action needs.
3. We evaluate WISH with two GUI agents on three benchmarks and conduct paired component tests. With Qwen3-VL-8B on AndroidWorld, WISH achieves 50.0% task success, exceeding the strongest reproduced baseline by 7.8 percentage points. WISH also achieves higher task success than the evaluated baselines across two agents and all three benchmarks.

2 Understanding World Model Use for GUI Agents

In this section, we investigate whether world models help GUI agents and which models they should use. Do predictions improve task success, and is calling a world model at every step better than selective use? Can the GUI agent make these call decisions autonomously? Together, these questions address whether to use a world model. To study which world model to use, we ask whether choosing among models can help with different tasks and candidate actions. We address these questions through task comparisons, prompt tests, and teacher-guided traces, and use the findings to guide WISH's design.

2.1 Problem Definition

Let π_θ denote a frozen GUI agent with policy parameters θ, and let $\mathcal{M}$ denote a set of available frozen world models. At environment step t, the agent receives the task goal g, the current screenshot I_t, and accessibility elements U_t. The screenshot and accessibility elements form the observation $o_t = (I_t, U_t)$, while the interaction history h_t records earlier actions and observations. Together, these inputs form the current context $x_t = (g, o_t, h_t)$. An action's outcome is the next observation o_{t+1}, namely the screen that appears after the agent executes it. A world model $m \in \mathcal{M}$ predicts this outcome for a proposed action a before execution, and we denote the prediction by $m(o_t, a)$.

The agent proposes an ordered list of K_t candidate actions $A_t = (a_t^1, \ldots, a_t^{K_t})$, where a_t^k is the action at position k. Before requesting any world-model output, the agent faces two decisions. The binary variable z_t indicates *whether* to request predictions ($z_t = 1$) or skip them ($z_t = 0$). If $z_t = 1$, the assignment $m_t^k \in \mathcal{M}$ specifies *which* world model predicts the outcome of candidate a_t^k. Together, these decisions determine the initial prediction set $E_t = \{(a_t^k, m_t^k, m_t^k(o_t, a_t^k))\}_{k=1}^{K_t}$ when $z_t = 1$ and $E_t = \varnothing$ when $z_t = 0$. Autonomous world-model use means that π_θ itself makes both decisions using x_t and A_t. The agent then uses x_t and E_t to choose an action a_t and submits it to the environment, where real execution produces the next observation o_{t+1}. The environment determines from real execution whether the episode trajectory $\tau = (o_1, a_1, o_2, a_2, \ldots)$ achieves the task goal g and returns task success $S(\tau) \in \{0, 1\}$. World-model use should increase the success rate $\mathbb{E}[S(\tau)]$ while θ and the world models remain fixed.

2.2 Whether to Use a World Model

Prediction at every step. We first investigate the value of the GUI world model by the following experiments. A Qwen3-VL-8B (Bai et al., 2025) achieves 31.9% task success on AndroidWorld (Rawles et al., 2025) without any world model, counting tasks with positive environment rewards as successes. This non-world-model usage experiment serves as our baseline, and we compare it with an always-on strategy that calls a fixed world model at every step. The agent proposes three candidate actions, and the world model predicts the next observation for each candidate to help the agent choose a more promising action. We use the released MWM delta, full, and html world models (Xu et al., 2026b) and repeat this experiment five times. All three world models improve task success over the no-world-model baseline, with delta reaching 36.2%. These results show that world models can help the agent choose actions, and that the benefit depends on the world model used.

Selective calls through an external gate. We then investigate non-autonomous usage of current GUI world models. MWM (Xu et al., 2026b) incorporates self-reflection into the reasoning process,

using a world model to assess and choose the next action. The method uses an entropy gate based on three candidate actions to decide whether to request a prediction. In our reproduced experiments, self-reflection yields considerable improvements in task success (Table 1). Adding entropy gating to self-reflection lowers task success with delta and full, but increases the number of successful tasks with html from 44 to 49, reaching a 42.2% success rate. These results highlight the importance of choosing a suitable call strategy when using a world model to support the GUI agent.

Direct prompting for autonomous WM use. Our initial experiments show that self-reflection and, in some configurations, entropy gating can improve the tested GUI agent's performance. However, these two methods are actually non-autonomous in fact. A key question is whether the agent can autonomously decide to use a world model. To test this ability, we directly ask the GUI agent whether world-model predictions would help it choose an action, and it requests predictions in 3.5% of recorded call decisions. In a separate test, we ask whether it is uncertain about its proposed action's outcome or cannot choose between actions with different outcomes. Requesting predictions when the agent reports either condition leads to world-model calls on 82.7% of steps. In a MobileGym example (Wu et al., 2026), the agent declines predictions on all four steps and repeats a menu click that leaves the screen unchanged. These observations show that direct prompts alone cannot provide sufficient guidance for the tested GUI agent to use world models autonomously and reliably. Appendix C.12 gives the details.

Table 1: World-model call strategies on AndroidWorld (Qwen3-VL-8B; 116 tasks). Entries show positive-reward task percentages, with no WM at 31.9%. Each run uses one released MWM checkpoint, and always-on requests predictions at every action-decision step.

Call strategy	delta	full	html
Always-on candidate selection	36.2%	34.5%	32.8%
Always-on self-reflection	37.9%	39.7%	37.9%
Entropy-gated self-reflection	36.2%	38.8%	42.2%

2.3 Which World Model to Use

Different strengths across world models. In Section 2.2, we show that world models help the GUI agent, but different world models give it different information about the next screen (Xu et al., 2026b). delta predicts local changes caused by an action, full predicts the complete next screen in text, and html predicts the next screen as renderable code that captures its text and layout. These differences give the agent information at different scopes for comparing candidate actions. Moreover, these three world models together solve more tasks than any single model. As the diffusion world model solves fewer tasks and takes longer to run, we exclude it (Table 2). In our AndroidWorld runs with one world model at every step, the success sets of MWM delta, full, and html together cover 55/116 tasks, 13 more than the strongest single model. Each world model has unique solved tasks that the other two cannot solve (Appendix C.10), this motivates wiser choice among world models. We next ask who can make this choice.

Choice by a teacher. Since the tested GUI agent lacks the autonomy to use world models (Section 2.2), we let a teacher make this choice. On AndroidWorld, Qwen3-VL-8B proposes three candidate actions at each step, and all three world models predict the outcome of each candidate action. A separate teacher model, Gemini 3.5 Flash (Kavukcuoglu et al., 2026), helps the GUI agent choose an appropriate world model at every step. The agent fails 58 tasks both without a world model and when using any single world model at every step. The teacher rescues five of these tasks by choosing different world models. Choosing among several world models can therefore solve tasks that the fixed methods cannot.

Table 2: Task success with Qwen3-VL-8B on AndroidWorld when each output comes from another world model or from MWM (Xu et al., 2026b). The other world models are Gemini 3.5 Flash (Kavukcuoglu et al., 2026), Code2World (Zheng et al., 2026), and GPT-image-2 (OpenAI, 2026). All runs use always-on candidate selection and count positive-reward tasks out of 116 as successes. Appendix C.10 gives MWM checkpoint names, complete-task counts, and total rewards.

	Other world model		MWM
Output	Model	Success	Success
delta	Gemini 3.5 Flash	35.3%	36.2%
full	Gemini 3.5 Flash	27.6%	34.5%
html	Code2World	28.4%	32.8%
diffusion	GPT-image-2	29.3%	21.6%

2.4 Insights for Harness Design

To summarize, we draw three insights from the preceding ablation studies.

Whether. A call is worth making when the predicted outcomes of the candidate actions could change the upcoming action the agent will take. When the current screen already settles the choice, the agent can act without a prediction. Calling a world model when its predictions could change the action choice, rather than at every step, helps the GUI agent (Section 2.2).

Which. Each world model gives different information about the next screen, so the right choice depends on what the agent needs to know about each candidate action. Choosing the right world model for each action helps the GUI agent (Section 2.3).

Autonomy. The GUI agent is overconfident in its first action, as Xu et al. (2026b) also observe, and rarely judges what a world model could add. It therefore needs explicit guidance to decide whether to call a world model and which one to use. Without such guidance, the tested GUI agent lacks the autonomy to make these two decisions (Section 2.2).

These insights motivate us to propose WISH, a training-free harness for autonomous world-model use through two decisions, **whether** and **which**. The three insights serve as its design principles.

3 WISH

Following the principles in Section 2.4, WISH gives the GUI agent the `guidance` and inputs its needs at each decision stage to autonomously decide whether to call a world model and which one to use. Five modules support these decisions: candidate preparation, action selection, execution checks, memory and progress tracking, and recovery (Figure 2). These modules combine prompts, rule-based checks, and execution records to guide action choices using current GUI facts, world-model predictions, and real execution feedback. We denote the execution records by ξ_t; they track repeatedly ineffective actions and completed goal items. At each decision stage, WISH combines these records with the context x_t (Section 2.1) to form the augmented context $\bar{x}_t = (x_t, \xi_t)$. We first explain how we build `guidance` from teacher traces, then describe each module and its corresponding decision process.

3.1 Building WISH from Teacher Traces

Teacher traces. We build the `guidance` from MobileGym traces (Wu et al., 2026) with a teacher agent T who guides the GUI agent π_θ. At each step t, every candidate world model $m \in \mathcal{M}$ predicts the outcome $m(o_t, a_t^k)$ of every candidate a_t^k in A_t. As in Section 2.3, Gemini 3.5 Flash serves as the teacher and evaluates one candidate with one prediction at a time. It returns a confidence $c \in [0, 1]$, a reason r, and a label y that is 1 if the action helps reach the goal (valid) and -1 otherwise (invalid). We use the signed confidence as the candidate's score:

$$(c, r, y) = T\big(g, h_t, I_t, a_t^k, m(o_t, a_t^k)\big), \qquad s_t^k(m) = y\, c. \tag{1}$$

The teacher also scores the first proposed action without a world-model prediction; we record this evaluation as no_wm. For each world model, we retain the highest-scoring candidate action, choosing the prior candidate in the proposal order when scores tie. We then compare these three scores with the no_wm score, and the environment executes the action with the highest score. Ties at this stage follow the fixed priority no_wm, delta, full, then html (Appendix C.3). The collection contains 209 episodes, 8,966 steps, and 26,894 candidates. For 7,845 candidate actions (29.2%), the teacher judges the same action valid with one world model's prediction and invalid with another. In 788 of 8,966 steps (8.8%), at least two candidate actions have different highest-scoring world models, with each candidate's top two model scores differing by more than 0.05.

From reasons to conditions. During trace collection, the teacher scores candidate actions using the world models' predictions. At test time, the GUI agent π_θ decides whether to request predictions and which world model each candidate needs before seeing those predictions. We therefore turn the teacher's reasons into `guidance` that depends on $\bar{x}_t$ and A_t. To do so, we compare the teacher's judgements across world models for the same state and candidate action and identify the information behind each judgement.

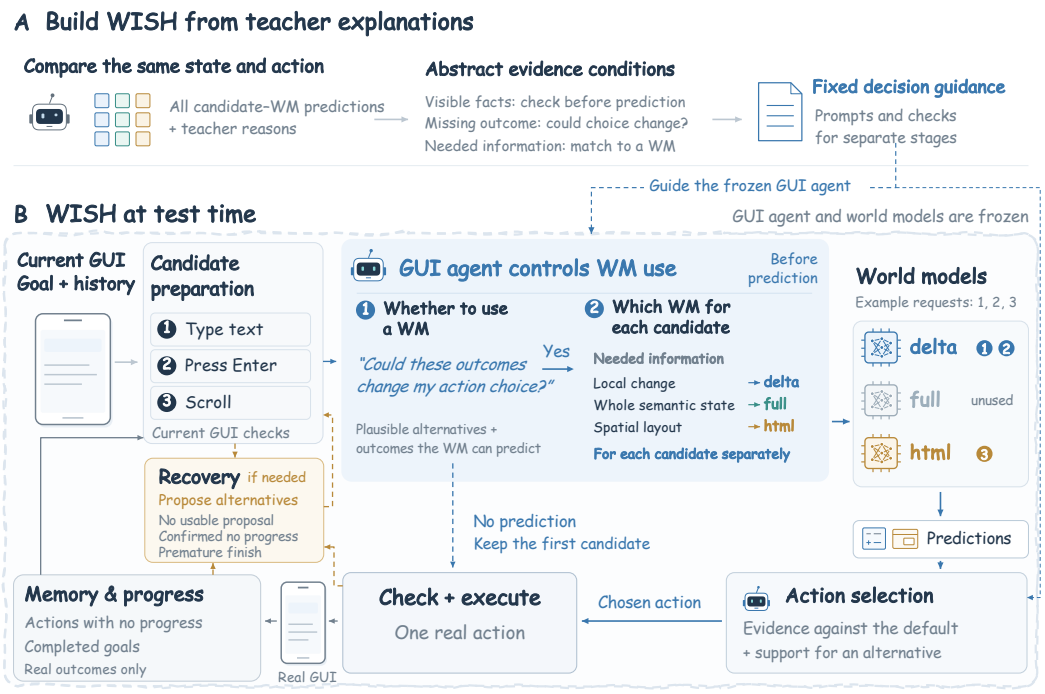

Figure 2: WISH construction and test-time control. **(A)** Teacher explanations inform fixed guidance for each stage. **(B)** The GUI agent autonomously decides whether to request predictions and which world model each candidate needs before generation. Skipping prediction sends the first candidate to execution checks. Both paths execute one real action, whose real outcome updates memory.

Within WISH, we encode these conditions as the guidance G, a set of prompt templates and rule-based checks that remains fixed throughout evaluation (Appendix C.3). Before requesting world-model predictions, the GUI agent could apply rules in G to prepare candidate actions and choose z_t. It can then choose the world model m_t^k using rules about the information a prediction needs to show. After receiving the predictions, the agent could use rules based on their content to select the final action a_t from E_t. At each stage, WISH gives π_θ the stage inputs and relevant rules in G, then applies the corresponding checks to its response.

3.2 Whether to Use a World Model

Candidate preparation. WISH guides the GUI agent to request predictions when they could change its action choice. The agent therefore needs concrete candidate actions to compare before making this decision. WISH asks π_θ to propose up to three ordered candidates in a separate response, each with action parameters, a reason, and a confidence value (Figure 2B). WISH then checks the proposals, and the first candidate that passes the checks becomes the default a_t^1.

Prediction decision. Given $\bar{x}_t$ and A_t, the GUI agent autonomously judges whether seeing the predicted outcomes could change which candidate it prefers. The guidance asks it to classify the step into one of three cases. In case 1, current evidence already ranks the candidates because one action is the standard next step or the relevant outcomes follow from the current screen. In case 2, at least two candidates remain competitive, with outcomes that cannot be read from the current screen but still depend on it, such as what a scroll reveals or whether a tap opens a file or selects it. In case 3, a world model could only guess because the next screen belongs to a new app, a needed value must first be read or calculated, or every candidate already has a known defect. The agent requests predictions only for case 2,

$$z_t = \pi_\theta(\bar{x}_t, A_t; G), \qquad z_t \in \{0, 1\}, \tag{2}$$

and it returns this decision and its reason before any world-model output exists. To limit prediction cost in long tasks, WISH sets $z_t = 0$ after 40 steps, while the agent continues to choose and execute actions. When $z_t = 0$, the default a_t^1 goes directly to the execution checks in Section 3.3.

World models. Following the complementary results in Section 2.3, WISH offers three frozen world models with distinct information scopes. The delta model predicts a local change in text, such as a field update or toggle, while full predicts the whole next screen in text so the agent can assess several fields or settings together. The html model renders the predicted interface to show menu layouts, control locations, and newly visible content. These models form the toolset $\mathcal{M} = \{\text{delta}, \text{full}, \text{html}\}$.

Agent choice. When $z_t = 1$, WISH gives the GUI agent the goal, the current GUI, the candidates with their reasons, and the model-choice guidance. For each candidate, the agent identifies what it needs to know about the outcome and chooses the world model whose scope covers that information. It chooses html when it must see the resulting screen, such as whether the action enters a mode or what a scroll reveals, and delta when a local state change settles the question. It chooses full when it must assess the complete next state, including all relevant fields before submission. The required evidence can differ even for two taps on the same screen, such as changing a setting and opening a menu. The agent returns one main model for every candidate, with a reason, in a single response:

$$(m_t^1, \ldots, m_t^{K_t}) = \pi_\theta(\bar{x}_t, A_t; G), \qquad m_t^k \in \mathcal{M}. \tag{3}$$

Because the agent makes these assignments before any prediction exists, it can request different evidence for different candidates without running all world models (Figure 2B).

3.3 Action Selection and Execution

Action choice. The GUI agent evaluates the candidates using their world-model predictions in E_t and the current screen, giving precedence to current-screen facts when the evidence conflicts:

$$\tilde{a}_t = \pi_\theta(\bar{x}_t, A_t, E_t; G), \qquad \tilde{a}_t \in A_t. \tag{4}$$

WISH keeps the default a_t^1 unless the agent shows both that a_t^1 fails and that another candidate makes progress, citing the current screen or a prediction (Appendix C.5). This rule protects a default that the current screen supports.

Execution checks. Before execution, WISH checks whether the chosen action is executable on the current screen. If it is not, WISH falls back to the first executable candidate when one is available. A few fixed rules then check common task requirements, such as confirming a deletion only for the item named in the goal and saving an edit before finishing. The environment executes the resulting action a_t and returns o_{t+1}, which updates the history and memory (Appendices C.4 and C.5).

3.4 Memory and Recovery

Memory and progress. WISH builds the records ξ_t from real execution. After each action, it compares the screens before and after execution and records actions that repeatedly leave the screen unchanged, completed goal items, and unfinished operations. WISH passes these records to the agent when it proposes later actions, so the agent avoids repeating ineffective actions or completed work. Before model choice, WISH also removes candidates that have failed to make progress at least twice from the same screen, as long as other candidates remain. Only real execution updates these records, and the environment alone judges task completion.

Recovery. When no candidate is usable or the agent keeps making no progress, WISH asks the same GUI agent for new candidate actions. The agent sees the reason for the request, the goal, the current screen, the history, and the existing candidates, and proposes alternatives in up to three new samples. WISH checks these alternatives, merges them into the candidate list, and runs the whether, which, and action-selection stages again before executing anything. Recovery can therefore add a decision round within one environment step (Appendix C.6).

4 Experiments

We evaluate WISH's task success, examine the GUI agent's autonomous world-model decisions, and assess the components that support these decisions. The main table compares Qwen3-VL-8B and Qwen3-VL-32B-Instruct (Bai et al., 2025) on AndroidWorld (Rawles et al., 2025), AndroidLab (Xu et al., 2024), and MobileWorld (Kong et al., 2026). MobileWorld adds tasks with long sequences of actions across applications. The detailed analysis of autonomous world-model decisions and supporting components uses the Qwen3-VL-8B AndroidWorld runs.

Evaluation setup. We evaluate WISH with frozen Qwen3-VL-8B and Qwen3-VL-32B-Instruct agents (Bai et al., 2025), using frozen MWM delta, full, and html world models (Xu et al., 2026b). In the Qwen3-VL-8B AndroidWorld runs analysed below, the agent samples ordered action proposals at temperature 0.7. To give date-dependent tasks the same temporal reference, we fix the AndroidWorld device time to 15:34 UTC on 15 October 2023 for all evaluated methods. The no-world-model and entropy-gated baselines follow the procedures in Section 2.2; Appendix C.10 records their detailed settings. The primary AndroidWorld metric is the fraction of 116 tasks with positive environment-verified reward. We also report complete tasks (reward $= 1$) and total reward in the appendix. The WM-step rate measures the fraction of logged steps with at least one WM request.

4.1 Main Results

Table 3 compares WISH with no-world-model and entropy-gated self-reflection baselines on three benchmarks and two GUI agents. WISH achieves the highest task success in all six settings. With Qwen3-VL-8B, it completes 50.0% of AndroidWorld tasks, 7.8 percentage points more than the strongest baseline, entropy-gated self-reflection with html (42.2%). On AndroidLab and MobileWorld, it reaches 44.2% and 19.7%, 4.3 and 5.2 percentage points above the strongest baseline on each benchmark. With Qwen3-VL-32B-Instruct, WISH leads the strongest baseline by 10.3, 5.1, and 5.2 percentage points on AndroidWorld, AndroidLab, and MobileWorld, respectively. Using a fixed world model yields less consistent gains. Entropy-gated self-reflection sometimes falls below the no-world-model baseline, for example with delta on MobileWorld for the 8B agent (10.3% versus 12.8%) and with full on AndroidWorld for the 32B agent (37.1% versus 39.7%). Table 12 reports variation across five complete WISH runs with Qwen3-VL-8B on AndroidWorld, and Table 5 records the additional harness configurations.

Table 3: Main results across GUI agents on AndroidWorld, AndroidLab, and MobileWorld. AndroidWorld task success counts positive-reward tasks out of 116, AndroidLab task success counts successful tasks out of 138, and MobileWorld task success counts tasks with a positive score out of 117. MWM self-reflection uses entropy gating and the indicated world-model checkpoint.

Method	AndroidWorld Task success	AndroidLab Task success	MobileWorld Task success
GUI agent: Qwen3-VL-8B			
No world model	31.9%	37.7%	12.8%
MWM self-reflection (delta)	36.2%	38.4%	10.3%
MWM self-reflection (full)	38.8%	39.9%	12.0%
MWM self-reflection (html)	42.2%	37.7%	14.5%
WISH	**50.0%**	**44.2%**	**19.7%**
GUI agent: Qwen3-VL-32B-Instruct			
No world model	39.7%	37.7%	17.1%
MWM self-reflection (delta)	41.4%	39.9%	17.9%
MWM self-reflection (full)	37.1%	39.1%	17.1%
MWM self-reflection (html)	44.0%	40.6%	16.2%
WISH	**54.3%**	**45.7%**	**23.1%**

4.2 Autonomous World Model Use

Beyond task success, WISH's traces show that the GUI agent makes both decisions itself before any world-model prediction exists. The counts below come from the 8B AndroidWorld run that completes 58 tasks, and the examples link each decision to the information available at that step.

Whether decisions. WISH requests world-model predictions on 349 of 1,606 steps (21.7%) under the request limits in Section 3.2. Entropy-gated self-reflection requests predictions on a similar share of steps (20–23%) while retaining one world model throughout each run. WISH's traces record the GUI agent's reasons for requesting or skipping predictions. In a Markor editing task, it judges that opening the visible target file is the clear next action and skips prediction. In a note-creation task, it is unsure which of three candidate actions will lead to a new note, so it requests predictions to compare tapping the creation button, opening the "More" page, and opening the menu. The agent explains both decisions before receiving any world-model output, distinguishing a choice that the current GUI already settles from one that needs the predicted outcomes of the candidate actions.

Which decisions. The GUI agent assigns different primary world models to candidates on 208 of the 349 prediction steps (59.6%), counting only the agent's own assignments. In one Markor editing step, it assigns delta to text entry and pressing Enter to check whether the requested text would appear and whether Enter would commit the input. For scrolling, it chooses html because it needs a rendered view of the content's position after the action. The agent gives these reasons before prediction, applying the guidance to different information needs within the same screen. These traces show that the agent makes both decisions according to the action choice it faces. Appendix C.11 records the explanations.

4.3 Supporting Components

Each supporting component targets a failure that would otherwise limit the agent's decisions or its final action. We describe the problem that each component addresses and the evidence from WISH's traces in the run analysed in Section 4.2; Table 4 reports the paired task results.

Candidate preparation. The whether and which decisions require alternatives to compare, but a GUI agent that proposes actions and decides whether to request predictions in one response often returns a single candidate. This joint response yields one candidate on 516 of 1,411 steps (36.6%), and separate proposal reduces this share to 0.5%. WISH's checks then correct the proposals: they move a defect-free action to the front on 91 steps, remove malformed candidates on 8 steps, correct target elements 9 times, and replace existing text instead of appending on 19 steps.

Table 4: Paired task outcomes with and without supporting components. Entries show successful runs / paired runs, where success means positive reward. Each row uses its own task set and control; pairs include both mirrored rounds. Appendix C.8 reports direct gains and losses under the matching criteria.

Component	With	Without
Separate candidate proposal	21/40	9/40
Extra candidates (same order)	10/40	8/40
Action repair	16/38	11/38
Source-file recovery	13/18	10/18
State–action memory	22/50	16/50
Recovery after no progress	17/22	15/22

Action selection. After seeing predictions, the agent may replace its default with an action that does not serve the goal. In a music-queue task that requires the songs in a given order, a selector without WISH's rule inspects the predictions and clicks the second requested song instead of looking for the first song (Appendix C.9). WISH therefore lets the agent replace the default only when it shows evidence that the default fails and that the alternative makes progress toward the goal.

Execution checks. A plausible action can still break a task requirement, and some mistakes cannot be undone, such as confirming the deletion of a similarly named file. The checks catch such cases before execution. They blocked one destructive confirmation that named the wrong file and led the agent to the exact source file named in the goal in two file-move tasks, keeping the moves on track. They also switched to an untried candidate on 10 steps where navigation kept repeating.

Memory and recovery. The agent can stall by repeating actions that leave the screen unchanged, wasting steps needed for the task. WISH uses records from real execution to break such loops. In the 116-task AndroidWorld run with Qwen3-VL-8B, WISH removed candidates with confirmed lack of progress on 40 steps in 13 tasks and requested new candidates on 33 steps in 11 tasks after the agent proposed such an action again. In a separate component test, recovery helps the agent leave a stalled "Open with" dialog during file deletion (Appendix C.9).

5 Conclusion

WISH is a training-free harness that gives a frozen GUI agent autonomous control over whether to request world-model predictions and which world model each candidate action needs. The whether decision identifies candidate actions whose unresolved outcomes could change the agent's choice, while the which decision matches each candidate's information needs to a world model. We abstract teacher explanations into conditions for these decisions and combine the guidance with supporting components throughout the control loop. With Qwen3-VL-8B, WISH achieves 50.0% task success on AndroidWorld while requesting world-model predictions on 21.7% of steps. Its traces show that the agent requests predictions for unresolved action choices and selects different world models for candidates on the same screen. These findings support organising world-model use around the evidence needed for an action choice, with the GUI agent deciding what to request.

AI USE STATEMENT

We used AI coding assistants to aid or polish writing and draft sections of the paper.

ETHICS STATEMENT

This work studies automation in sandboxed Android emulators and uses no private human data.

REPRODUCIBILITY STATEMENT

We provide the method, evaluation settings, prompts, and experimental protocols throughout the main text and appendices. Appendix C gives implementation details, tested configurations, and the audit process.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Yilin Cao, Yufeng Zhong, Zhixiong Zeng, Siran Dai, Liming Zheng, Jing Huang, Haibo Qiu, Peng Shi, and Wenji Mao. Mobiledreamer: Generative sketch world model for gui agent, 2026. URL https://arxiv.org/abs/2601.04035.

Hyungjoo Chae, Namyoung Kim, Minju Gwak, Gwanwoo Song, Jihoon Kim, Kai Tzu iunn Ong, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. World models for web agents. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*, 2024. URL https://openreview.net/forum?id=NWU6AaJv48.

Yurun Chen, Zeyi Liao, Ping Yin, Taotao Xie, Keting Yin, and Shengyu Zhang. Safepred: A predictive guardrail for computer-using agents via world models, 2026. URL https://arxiv.org/abs/2602.01725.

Tianqing Fang, Hongming Zhang, Zhisong Zhang, Kaixin Ma, Wenhao Yu, Haitao Mi, and Dong Yu. WebEvolver: Enhancing web agent self-improvement with co-evolving world model. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 8959–8975, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.454. URL https://aclanthology.org/2025.emnlp-main.454/.

Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your LLM secretly a world model of the internet? model-based planning for web agents. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=c6l7yA0HSq.

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

Yu Han and Tianwen Qian. Wm-r1: Training gui agents to reason and leverage world models with reinforcement learning, 2026. URL https://arxiv.org/abs/2608.27508.

Koray Kavukcuoglu, Jeff Dean, Oriol Vinyals, and Noam Shazeer. Gemini 3.5: Frontier intelligence with action. Google Blog, May 2026. URL https://ai.google.dev/gemini-api/docs/models/gemini-3.5-flash.

Woosung Koh, Sungjun Han, Segyu Lee, Se-Young Yun, and Jamin Shin. Generative visual code mobile world models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ZI6dgwPpoE.

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. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6142–6167, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.278. URL https://aclanthology.org/2026.acl-long.278/.

Juyong Lee, Dongyoon Hahm, June Suk Choi, W. Bradley Knox, and Kimin Lee. Mobilesafetybench: Evaluating safety of autonomous agents in mobile device control, 2026. URL https://arxiv.org/abs/2410.17520.

Kelvin Li, Dhruv Pendharkar, Anish Pahilajani, Chuyi Shang, Leon Oks, Leonid Karlinsky, Rogerio Feris, Trevor Darrell, and Roei Herzig. Discriminative world models for web agents, 2026. URL https://arxiv.org/abs/2609.02885.

Shufan Li, Konstantinos Kallidromitis, Akash Gokul, Yusuke Kato, Kazuki Kozuka, and Aditya Grover. Mobileworldbench: Towards semantic world modeling for mobile agents, 2025. URL https://arxiv.org/abs/2512.14014.

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. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=mWoMyDEfbM.

Kai Mei, Jiang Guo, Shuaichen Chang, Mingwen Dong, Dongkyu Lee, Xing Niu, and Jiarong Jiang. R-wom: Retrieval-augmented world model for computer-use agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=5ZaoXB3MdP.

OpenAI. GPT-Image-2 model. OpenAI API documentation, April 2026. URL https://developers.openai.com/api/docs/models/gpt-image-2.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=il5yUQsrjC.

Luke Rivard, Sun Sun, Hongyu Guo, Wenhu Chen, and Yuntian Deng. NeuralOS: Towards simulating operating systems via neural generative models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=TE2Vu7WJki.

Zhouzhou Shen, Xueyu Hu, Xiyun Li, Tianqing Fang, Juncheng Li, and Shengyu Zhang. World-model-augmented web agents with action correction, 2026. URL https://arxiv.org/abs/2602.15384.

Yiming Wang, Da Yin, Yuedong Cui, Ruichen Zheng, Zhiqian Li, Zongyu Lin, Di Wu, Xueqing Wu, Chenchen Ye, Yu Zhou, and Kai-Wei Chang. Llms as scalable, general-purpose simulators for evolving digital agent training, 2025. URL https://arxiv.org/abs/2510.14969.

Jiaming Wei, Zekun Wu, Adriano Koshiyama, and Maria Perez-Ortiz. Routing is least learnable where it is most valuable: Bounds on representation routing for web agents, 2026. URL https://arxiv.org/abs/2608.06171.

Dingbang Wu, Rui Hao, Haiyang Wang, Shuzhe Wu, Han Xiao, Zhenghong Li, Bojiang Zhou, Zheng Ju, Zichen Liu, Lue Fan, and Zhaoxiang Zhang. Mobilegym: A verifiable and highly parallel simulation platform for mobile gui agent research, 2026. URL https://arxiv.org/abs/2605.26114.

Zikai Xiao, Jianhong Tu, Chuhang Zou, Yuxin Zuo, Zhi Li, Peng Wang, Bowen Yu, Fei Huang, Junyang Lin, and Zuozhu Liu. Webworld: A large-scale world model for web agent training, 2026. URL https://arxiv.org/abs/2602.14721.

Weikai Xu, Yunren Feng, Haoxiang Lei, Kun Huang, Yuxuan Liu, Kang Zhao, Xiaolin Hu, Shuo Shang, and Bo An. Appdeltaworld: Transition-grounded delta code world model for mobile gui agents, 2026a. URL https://arxiv.org/abs/2608.05891.

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. How mobile world model guides gui agents?, 2026b. URL https://arxiv.org/abs/2605.10347.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents, 2024. URL https://arxiv.org/abs/2410.24024.

Zhichao Yang, Yuanze Hu, Haojie Hao, Longkun Hao, Dongshuo Huang, Hongyu Lin, Gen Li, Lanqing Hong, Yihang Lou, and Yan Bai. Mirage: Mobile agents with implicit reasoning and generative world models, 2026. URL https://arxiv.org/abs/2606.04627.

Xue Yu, Bo Yuan, Kailin Zhao, Pengshuai Yang, Hong Hu, and Junlan Feng. Seerguard: A safety framework for mobile gui agents via world model prediction, 2026. URL https://arxiv.org/abs/2607.15550.

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

A Related Work

GUI world models. GUI world models predict future interfaces as text, structured states, pixels, renderable code, or latent features. Text and structured models predict local changes, complete states, or spatial sketches (Chae et al., 2024; Gu et al., 2025; Li et al., 2025; Cao et al., 2026; Wang et al., 2025; Xiao et al., 2026). Visual models generate future screen pixels or render a predicted text change (Rivard et al., 2026; Luo et al., 2026; Guan et al., 2026), while code-based models generate executable HTML or transition-grounded code updates (Koh et al., 2026; Zheng et al., 2026; Xu et al., 2026a). Other work predicts candidate-separating text or latent future-screen features (Li et al., 2026; Yang et al., 2026). MWM directly compares the four forms most relevant to WISH: local changes (delta), complete semantic states (full), diffusion images (diffusion), and renderable code (html) (Xu et al., 2026b).

World-model control. Most systems use one configured world model for candidate simulation, single- or multi-step lookahead, action correction, or safety screening (Chae et al., 2024; Gu et al., 2025; Cao et al., 2026; Fang et al., 2025; Mei et al., 2026; Shen et al., 2026; Chen et al., 2026; Yu et al., 2026). MWM studies how an entropy gate selects steps for self-reflection with a configured world model (Xu et al., 2026b). WM-R1 learns optional calls to one fixed model through reinforcement learning (Han & Qian, 2026). These methods use one configured world model for prediction. Related work on representation routing selects text, screenshots, or combined observations for a web task (Wei et al., 2026). WISH gives a frozen GUI agent control over future-state evidence by letting it decide, without training, whether to call a world model and which model each candidate needs. Supervised fine-tuning (SFT) and reinforcement learning (RL) update the agent using demonstrations or rewards (Xu et al., 2026b; Han & Qian, 2026). WISH supplies decision guidance and organises the information available at each stage of model use.

B LIMITATIONS

We evaluate task success with Qwen3-VL-8B and Qwen3-VL-32B-Instruct (Bai et al., 2025) on AndroidWorld (Rawles et al., 2025), AndroidLab (Xu et al., 2024), and MobileWorld (Kong et al., 2026). Our detailed analysis of autonomous decisions uses Qwen3-VL-8B on AndroidWorld, while MobileGym (Wu et al., 2026) supplies the teacher traces and world-model output analysis. WISH keeps the GUI agent and world models frozen throughout evaluation. In future work, we plan to combine WISH with training to further improve the GUI agent's autonomous decisions about whether to request world-model predictions and which world model to use.

C IMPLEMENTATION AND ADDITIONAL EXPERIMENTAL DETAILS

C.1 CONFIGURATION DEFINITIONS

Table 5: Complete-system AndroidWorld profiles with Qwen3-VL-8B under the fixed-time protocol. Each row is a separately sampled 116-task run. Reward > 0 counts positive-reward tasks, Complete counts reward $= 1$, and WM-step rate counts steps with at least one prediction request. These profiles compare complete configurations; Table 4 reports the targeted component tests.

Configuration	Reward > 0	Complete	Total reward	WM-step rate
Action selection with fixed rules (no WM)	47/116	47/116	47.0	0.0%
Agent chooses among world models	50/116	49/116	49.5	20.8%
Agent chooses among models with action repair	49/116	48/116	48.5	21.7%
Closed-loop WISH with joint proposal and call decision	49/116	48/116	48.5	15.7%
Closed-loop WISH with separate proposal and call decision	52/116	49/116	50.5	21.1%
WISH	**58/116**	**58/116**	**58.0**	21.7%

Table 6: Complete-system configurations from Table 5 under the shared frozen-agent, fixed-time protocol.

Configuration	Components
Action selection with fixed rules (no WM)	Checks action targets and parameters and flags repeated actions using the current GUI and recent actions, then selects a candidate without world-model predictions.
Agent chooses among world models	Uses the prediction decision, per-candidate model choice, and evidence rules for changing the first action.
Agent chooses among models with action repair	Uses the same model-use interface together with target-index, target-identity, and replacement-text repair from current-screen facts.
Closed-loop WISH with joint proposal and call decision	Uses the model-use interface, action repair, strict state–action memory, progress checkpoints, bounded recovery, filename checks, move/copy transaction state, and goal-based source recovery. One response contains both candidate proposal and the prediction decision.
Closed-loop WISH with separate proposal and call decision	Uses the same closed-loop components, with separate responses for candidate proposal and the prediction decision.
WISH	Uses the separate-decision closed-loop profile, varied proposal decoding, and bounded recovery from repeated navigation.

Each row reports a complete configuration evaluated in a separate run. Each run stores its resolved configuration, source hashes, per-step decisions, world-model requests, selected actions, and real execution outcomes.

C.2 IMPLEMENTATION RECORDS

The harness exposes one configuration object and one policy entry point. The reported configuration proposes up to three actions, chooses a main world model for each candidate, and permits new

prediction requests during the first 40 task steps. WISH requests predictions for the selected candidate–model pairs in parallel, and the html service renders its generated interface before action selection. The fixed playbook prompts reside in `gui_harness/prompts.py`. The same frozen GUI agent makes the policy decisions, while the teacher participates only during trace collection.

The implementation separates policy decisions from stage operations. The policy controls the prediction decision, per-candidate model choice, and candidate revision. The runner maintains per-task state, budgets, parallel requests, and trace records, while environment adapters translate observations and action formats. Stage functions execute the requested operation, and fixed recovery rules can open another bounded round through the same control loop.

We wrote the playbook prompts and checks by analysing recurring teacher explanations. At test time, WISH uses the fixed prompts and checks without retrieving teacher examples or updating model parameters.

Each step record contains the final candidate set, current-screen checks, prediction decision, per-candidate model assignments, generated outputs, selected action, total prediction count, and stage timings. Each task record stores the final environment result, while the run manifest stores the resolved configuration and source hashes.

C.3 Teacher Scoring and Trace Abstraction

Judgements and action choice. At each MobileGym step, Gemini 3.5 Flash evaluates every candidate separately with each world-model prediction. It also evaluates the first proposed action using the current screen without a prediction; we record this evaluation as no_wm. For each evaluation, the teacher returns `judgement`, `confidence`, and `reason`. For confidence $c \in [0, 1]$, the collection procedure assigns score $+c$ to a valid judgement and $-c$ to an invalid judgement. These scores express the teacher's support for an action under the supplied evidence.

For each world model, the collection procedure keeps the highest-scoring candidate, retaining the earlier candidate in proposal order when scores tie. It then compares the three model scores with the no_wm score and executes the action associated with the highest score. Ties prefer no_wm, then delta, full, and html, in that order. The collection procedure assigns failed predictions a fallback score of -1 without calling the teacher. The records retain all candidate scores and reasons, together with the selected action and model label. All world-model predictions precede this selection, including steps whose winning label is no_wm.

Teacher prompt. We use the following prompt for each judgement, organised by message and prediction input. Every request includes the current screenshot. Text predictions enter the user message, while a rendered html prediction supplies a second image when available.

> **System.** You are a reward scorer. Your sole task is to assess whether a candidate action on a phone helps achieve the user's overall goal. You do not execute or output actions. Use the goal, history, current screen, candidate action and its meaning, and the world model's prediction of the next screen when available. Judge whether the action is on the right track towards success and give a calibrated confidence.
>
> **User template.**
> Overall goal: *[goal]*
> History: *[history]*
> Candidate action: *[action and parameters]*; meaning: *[action description]*
> *[One of the following prediction blocks]*
>
> **No prediction.** Judge from the current screen alone whether executing this action is on the right track towards the goal.
>
> **Text prediction.** The world model *[model]* predicts the following next screen after execution: *[prediction]*. The prediction may be unfaithful or hallucinated; unreasonable or contradictory content usually indicates that the action fails.
>
> **Rendered prediction.** Image 2 shows the next screen predicted by the world model *[model]* after execution. The prediction may be unfaithful or hallucinated; unreasonable elements usually indicate that the action fails.
>
> **Required response.** Return only JSON with `judgement` set to `valid` or `invalid`, `confidence` set to a number between zero and one, and `reason` set to a brief explanation.

From reasons to decision guidance. We group recurring explanations by their decision conditions, as described in Section 3.1. Among the 3,671 no_wm wins, 3,359 (91.5%) arise from ties resolved by the priority rule above. We therefore use recurring reasons to guide WISH, while treating each model label as the result of the scoring and tie-breaking procedure.

When abstracting the guidance, we distinguish the information each reason cites from when the agent can assess that information. We compare reasons across world models for the same state and candidate, then identify whether they cite current-screen facts, a local change, several aspects of the next state, or a visual property. When a reason cites a predicted result, we express the information needed to assess the action as a condition for the agent. The condition uses the current goal, GUI, history, and candidates, so it can guide a request before that result exists. Conditions about the reliability of returned evidence belong to action selection. Table 7 maps paraphrased teacher reason patterns to WISH's decision guidance.

Table 7: How teacher reason patterns inform WISH guidance. Conditions that use current context guide checks and requests; conditions that need returned predictions guide action selection.

Reason pattern	Condition used by WISH	Stage
The required value or target is already visible.	Check whether current facts settle the choice before requesting a prediction.	Preparation; whether
The reason depends on the interface reached by an action.	Request predictions when plausible candidate actions have unresolved outcomes that could change the choice and depend on the current GUI.	Whether
The reason cites a field update or local transition.	Choose delta if that local change supplies the needed information.	Which
The reason assesses several fields or the resulting page together.	Choose full when the action needs assessment of the complete semantic state.	Which
The reason cites selection controls or the rendered interface mode.	Choose html when judging the action requires seeing the resulting interface.	Which
A predicted result conflicts with the goal or current-screen facts.	Check evidence against the default action and support for an executable alternative.	Action selection

C.4 Candidate Preparation Rules

Current-screen checks. WISH checks action types, required fields, target indices, and accessibility facts before model choice. When an action reason names one unique visible target, WISH can correct a missing or incorrect index without changing the intent. For replacement text in a nonempty field, it adds a clear-before-type flag unless the reason asks to append. It also detects duplicate execution keys, direction–reason conflicts, and conflicting field values. WISH removes invalid formats and uses softer warnings to rotate the list towards an unflagged candidate. If every candidate receives a warning, a full rotation restores the original order. The first surviving candidate is the initial default, although fixed rules, candidate addition, and recovery can later change the order.

Candidate completion variant. A separate append-only variant makes one extra proposal when the first response contains one or two actions. It adds new execution keys without replacing or reordering existing candidates. The final candidate count depends on the valid actions returned, and later stages may change their order.

Source-file action addition. World models can compare only actions present in A_t. WISH can therefore add one missing action for a file-move task that names a literal source file and source folder, provided that the current folder exactly matches the goal. The rule can select a unique extension filter, move through an observed name-sorted list, or long-press one unique visible exact filename. It uses the goal and current accessibility elements, without task names, app names, or retrieved examples. An unclear match leaves the proposal unchanged.

C.5 Prediction and Selection Checks

Model-choice responses and requests. The agent returns one model name per candidate in a single response, keeping the candidate order. A malformed model-choice list triggers one repair query. If

the repair still fails, parsing rules retain valid assignments and fill missing entries with delta, or with the first available world model. The runner creates natural-language action descriptions only after the prediction decision. The GUI agent decides whether to request one additional world-model prediction for the default action, keeping its main assignments unchanged. WISH generates all requested predictions in parallel and includes them in E_t, with at most $K_t + 1$ predictions per round before service retries.

Evidence and final action checks. The action-selection stage receives the current screenshot, exact action parameters, proposal reasons and confidence values, and accessibility facts for each target. Each world-model prediction remains attached to its candidate action. delta and full outputs enter as text, while html enters as a rendered image. Normalised substring matching checks that text cited by the agent occurs in the prediction attached to that candidate. For rendered outputs, WISH requires an attached image and a clear spatial description. The GUI agent interprets the visual evidence when comparing actions. These checks govern which evidence can support a change. A final schema check replaces an invalid selection with an executable alternative when one is available. Otherwise, WISH retains the selected action.

Rules for replacing the default. Evidence against the default can be a conflict with current facts that WISH confirms, a delta prediction stating that the default changes nothing, or a quoted prediction showing that the default leads to a wrong state. Support for an alternative can be current facts that WISH confirms or a quoted prediction showing progress towards the goal, and the alternative must be executable. Each quoted prediction must stay within its world model's scope: local changes for delta, the whole screen for full, and layout for html. Fixed task rules then verify the goal item before an irreversible confirmation, keep move and copy operations on track, require saving before completion, and stop repeated navigation and premature completion.

C.6 State Tracking and Recovery

State and action keys. A strict state key combines foreground accessibility structure with a quantised screenshot that excludes changing system bars, while the action key retains every execution parameter. Returning to the same state after an action increments the corresponding no-progress count; observing a changed state clears that count. The default threshold for confirmed lack of progress is two executions of the same action from the same state with both observations unchanged. The proposal prompt receives these records. Before model choice, WISH removes confirmed no-progress candidates if other candidates remain.

Text and file-operation records. For an explicit multi-item goal, WISH records an item as complete when real execution provides evidence that its action finished. Later proposal prompts include these completed items and require the agent to preserve them. For text entry, WISH stores the requested value and checks the real successor for an exact match before reporting verified input and visible commit controls. Move and copy operations use a transaction record that begins after the real GUI confirms selection of the exact goal file. The record persists while the agent opens the operation menu and navigates the destination picker, and WISH permits the final operation only at the requested destination. A related check compares filenames in real selection or confirmation dialogs with the goal and uses a visible cancel action after a mismatch. Local records track progress; the environment judges completion.

Recovery requests and fallbacks. Recovery handles proposals with no parseable candidates, candidate sets that fail the structural execution checks, and sets in which every candidate meets the no-progress threshold. When the first candidate meets this threshold, recovery can request alternatives even with other candidates remaining. An incomplete explicit goal can also trigger recovery if the agent proposes a terminal action. Each request supplies the trigger, goal, current GUI, compact interaction history, existing candidates, and allowed action format to the same frozen GUI agent. The default budget is three proposal samples per request, with one replacement action per sample. WISH rejects invalid or duplicate replacements and gives new directions priority when merging them with existing candidates, up to the configured candidate limit. The revised set enters the usual decision stages. If proposal parsing and recovery both fail, WISH returns a wait action. If recovery for an incomplete goal supplies no valid replacement, WISH uses an existing non-terminal candidate when available. Other failed repair attempts retain the available candidate set for the applicable selection and execution checks. Per-task action counts also allow one limited escape through an

untried non-terminal candidate. The complete-system profile disables recovery based solely on coarse state repetition or duplicate candidates, and disables the separate model-based terminal review.

C.7 Stage Inputs and Outputs

Table 8 records the inputs, outputs, and roles of the test-time stages. Recovery can return revised candidates to the decision stages before real execution. The same frozen GUI agent makes the prediction decision, per-candidate model choice, and prediction-based action comparison, while WISH applies the fixed checks around these decisions.

Table 8: Inputs, outputs, and control roles of the WISH stages. The prediction decision and per-candidate model choice form the central model-use interface.

Stage	Input → output	Role in the control loop
Candidate preparation	Current context and agent proposal → A_t	Supplies checked alternatives before model use.
Prediction decision	$(\bar{x}_t, A_t) \to z_t$	Requests prediction when the predicted outcomes of candidate actions could change the action choice.
Per-candidate model choice	$(\bar{x}_t, A_t, k) \to m_t^k$	Matches each candidate to the form of evidence it needs.
Requested prediction	Candidate–model pairs → E_t	Generates predictions for the main assignments and any additional request from the GUI agent.
Action selection under evidence rules	$(\bar{x}_t, A_t, E_t) \to \tilde{a}_t$	Requires evidence against the current choice and support for an alternative.
Execution checks	Current context and $\tilde{a}_t \to a_t$	Checks the tentative action before real execution on either path.
State update from real execution	$(\bar{x}_t, a_t, o_{t+1}) \to (h_{t+1}, \xi_{t+1})$	Tracks verified progress and filters repeated no-progress actions.
Recovery	Current context, candidate checks, and progress records → revised candidates	Requests alternative actions within a fixed budget, then returns to the decision stages.

C.8 Protocol for Paired Component Tests

In each component study, we run treatment and control on the same task set in two mirrored rounds. We inspect the first difference in executed actions and attribute a direct change only when both runs share the pre-action GUI and upstream input. A changed candidate set is the intended intervention in proposal studies; for later components, earlier proposal or environment divergence prevents attribution.

We report paired task success together with strict direct gains and losses. We use an exact two-sided McNemar test for order-preserving candidate completion.

Table 9: Full results of the paired component tests. Success counts positive-reward runs, and pairs include both mirrored rounds. Each row uses its own task set and control configuration. Direct gains and losses require the same task, pre-action GUI, and upstream input at the first action difference, with the component acting by that point. Appendix C.9 gives the corresponding cases.

Component added	Pairs	Success with	Success without	Direct gains	Direct losses
Separate three-candidate proposal	40	21	9	6	5
Order-preserving candidate completion	40	10	8	3	1
Action repair	38	16	11	2	0
Goal-based source-action recovery	18	13	10	1	0
State–action memory	50	22	16	0	0
Recovery after repeated lack of progress	22	17	15	0	0

C.9 Component Tests and Decision Traces

Action repair. The repair rule acts when current accessibility facts uniquely identify the intended target or replacement-text operation. Across 38 paired runs, 16 succeed with repair and 11 without it. The strict first-difference rule identifies two direct gains and no direct losses: one correction changes a target index, and the other replaces an existing value instead of appending. These cases support the use of current GUI facts before prediction.

Goal-based source-action recovery. Across 18 paired runs, success is 13/18 with recovery and 10/18 without it. The strict comparison finds one direct gain and no direct loss. Both runs in the gain case share the goal, GUI, and proposed long press. The treatment then locates the exact filename below the visible sorted range, selects it, and completes the move, whereas the control selects an approximate visible filename. In this case, matching the exact source filename leads to the intended file and a successful move.

Candidate coverage and order. The separate three-candidate proposal succeeds in 21/40 paired runs, compared with 9/40 for the joint proposal, with six direct gains and five direct losses. A separate coverage diagnostic finds one candidate on 36.6% of steps with the joint prompt and 0.5% with the separated design. One direct gain avoids premature completion in a multi-item deletion task. In one direct loss, both profiles propose the same three actions but reverse a tap and a long press. In this case, changing the first action changes the task result while candidate coverage stays the same. The append-only completion variant preserves proposal order and reaches 10/40 successes, compared with 8/40 without adding candidates (exact two-sided McNemar test, $p = 0.791$). The matched comparisons identify three direct gains and one direct loss.

Action selection after prediction. In the music-queue case, the goal requires five songs in a specified order. At step 2, the agent proposes scrolling up to find the first song, scrolling down, or clicking a menu. After world-model prediction, the selector chooses the click and explicitly identifies its target as the second requested song. The comparison run scrolls to seek the first song, but its candidate set contains three scrolling actions. The upstream inputs therefore differ at the first action difference. This example motivates checking the requested song order before changing the default action.

State–action memory. We add memory of actions that leave the current GUI unchanged to an action-repair controller and compare it with the same controller without memory. Across two mirrored rounds of the same 25 tasks, 22 runs with memory receive positive rewards, compared with 16 without it. The first-action analysis identifies no direct gains or losses attributable to memory.

Recovery from repeated lack of progress. This comparison adds candidate exclusion and renewed action proposal when real execution repeatedly leaves the same state unchanged. Both conditions use the same base controller, which already includes state–action memory. Across two mirrored rounds of 11 tasks, 17/22 runs with this recovery rule receive positive rewards, compared with 15/22 without it. The two task-level gains follow earlier differences in candidate proposals or GUI state, and the audit finds no direct gains or losses from the recovery rule. In this paired component test, the rule triggers four recovery rounds on three tasks across 301 logged steps. In the file-deletion case, it closes the unchanged "Open with" dialog, but the agent later deletes a similarly named file and fails the task.

C.10 World-Model Baseline Details

Table 10 gives task counts and total rewards behind the baseline results in Tables 1 and 2. Each run uses the same 116-task AndroidWorld suite and frozen Qwen3-VL-8B agent. The always-on candidate-selection pipeline proposes three actions, requests predictions for them from one fixed world model, and asks the GUI agent to select an action. The selector breaks exact ties through the completion order of parallel scores. Self-reflection first evaluates the initial action with its world-model prediction and can switch to a second candidate after an invalid judgement. The entropy gate controls this self-reflection step using three independently sampled actions at temperature 0.7 and an entropy threshold of 0.3.

The no-world-model run with Qwen3-VL-32B-Instruct completes 46/116 tasks, all with full reward. Its 1,779 logged steps contain no world-model requests.

The general delta and full predictions use the Gemini 3.5 Flash API, and the general image predictions use the GPT-image-2 API. The html comparator uses the locally reproduced Code2World check-

Table 10: Detailed AndroidWorld baseline results with Qwen3-VL-8B. Positive and complete counts use reward > 0 and reward $= 1$, respectively, with 116 tasks in every row. Missing or non-finite rewards count as unsuccessful.

Method	WM source	WM	Positive	Complete	Reward sum
No world model	—	—	37	37	37.0
Always-on WM (candidate selection)	Gemini 3.5 Flash	delta	41	41	41.0
	Gemini 3.5 Flash	full	32	32	32.0
	Code2World	html	33	33	33.0
	GPT-image-2	diffusion	34	32	33.0
	MWM weights	delta	42	42	42.0
	MWM weights	full	40	40	40.0
	MWM weights	html	38	36	37.0
	MWM weights	diffusion	25	25	25.0
MWM self-reflection (always-on)	MWM weights	delta	44	44	44.0
	MWM weights	full	46	45	45.5
	MWM weights	html	44	43	43.5
MWM self-reflection (entropy-gated)	MWM weights	delta	42	40	41.0
	MWM weights	full	45	45	45.0
	MWM weights	html	49	48	48.5

point. The GUI-specific checkpoints are MobileWorld-Delta-TEXT-8B, MobileWorld-TEXT-8B, MobileWorld-html-8B, and MobileWorld-Diffusion. These comparisons use each model's input format and prompting setup, so their differences reflect both the checkpoint and how it receives the task. The fixed-time harness profiles appear in Table 5.

Entropy-gated self-reflection requests world-model predictions on 386 of 1,769 steps for delta (21.8%), 364 of 1,810 for full (20.1%), and 399 of 1,727 for html (23.1%).

Each of the three GUI-specific world models solves tasks missed by the other two. Pairwise historical unions are 51/116 for delta with full, 48/116 for delta with html, and 49/116 for full with html. The three-model union is 55/116, 13 tasks beyond the strongest single model. Among these three world models, delta solves six tasks that neither full nor html solves; the corresponding counts are seven for full and four for html. These counts compare the same three runs used to compute the task unions above. The union counts tasks solved by at least one of the three separate single-world-model runs.

C.11 WISH Decision Traces

The following examples come from the AndroidWorld run with 50.0% task success analysed in Section 4.2. Step numbers start at one. The trace records the agent's call reason and primary model assignments separately from returned predictions and action selection.

Skipping prediction. In the Markor edit-and-rename task, step 2 offers a tap, a long press, and a scroll. The agent cites the visible target note and judges that tapping it is the clear next action for editing. It declines prediction and executes the tap.

Requesting prediction. In the Markor create-and-share task, step 3 offers the creation button, the "More" page, and the menu. The agent requests prediction because it wants to compare how these actions lead to creating a note. Its reason identifies the next interfaces to predict.

Choosing world models. At step 4 of the edit-and-rename task, the candidates are text entry, pressing Enter, and scrolling. The agent assigns delta, delta, and html, respectively, and gives the following reason before prediction:

> For candidate [0], 'input_text' directly modifies the note's content; 'delta' suffices to confirm the text insertion. For candidate [1], 'keyboard_enter' confirms the text is committed; 'delta' is sufficient as it only checks local state change. For candidate [2], 'scroll' requires visual confirmation of content positioning; 'html' is needed to verify what the screen looks like after scrolling.

These assignments come from the model-choice stage.

We compute the 21.7% call rate over all 1,606 logged steps, including any steps after the 40-step request limit. Across the run's 349 prediction steps, 334 have three primary assignments and 15 have two. These assignments use different primary world models on 208 steps (59.6%).

C.12 Prompt-Only Call Diagnostics

Table 11 records the prompt probes used to test whether tool access alone yields selective calls.

Table 11: Prompt-only call diagnostics. Each row reports its own trace subset and response measure.

Decision prompt	Observed response	Consequence
Direct request to use the tool when useful	Requests world models' predictions on 15/423 decisions (3.5%)	Few positive call decisions
Nine-item defect checklist	Flags 58/80 steps at 52% precision	Many false triggers
Outcome-certainty question	Reports uncertain outcomes on 222/265 decisions (83.8%)	Calls a WM on 229/277 steps (82.7%)
Actions, call decision, and model choice in one response	Selects no prediction on all four test steps	Repeats the same menu click
Joint proposal and call decision	Returns one candidate on 516/1,411 steps (36.6%)	Restricts candidate comparison
Separated proposal and call decision	Returns one candidate on 0.5% of evaluated steps	Restores candidate coverage

Direct call decision. The action prompt supplies the goal, action history, current screenshot, a labelled screenshot, and UI element descriptions. It asks for three alternative next actions, each with a reason, confidence, and action parameters. In the same response, the agent returns `use_world_model` and `world_model_reason`. We quote the decision-specific instructions verbatim from the English prompt below and omit the shared action-format instructions.

> You also have access to a **World Model tool**. When invoked, it takes your candidate actions and *simulates* what the screen would look like AFTER each action is performed, then a scorer compares those predicted next-states against the goal to help select the best candidate. The simulation costs extra time/compute, so you should call it only when it actually helps. YOU decide, every step, whether to use the World Model tool, by outputting a boolean field "use_world_model":
>
> Set it to true when the next state is hard to predict from the current screen alone, the candidate actions lead to meaningfully different outcomes, or you are genuinely uncertain which candidate is correct — i.e. seeing the simulated next-states would change your choice.
>
> Set it to false when the correct action is obvious / unambiguous, the candidates are essentially equivalent, or this is a terminal action (answer/status) — i.e. simulating would not change your choice. In that case candidate "1" is executed directly.
>
> Output a top-level boolean "use_world_model" (true/false), a short "world_model_reason" string explaining that decision, and exactly 3 candidate actions with keys "1", "2", "3" (strings).

A positive decision requests predictions from the WM configured for that run; a negative decision executes the first candidate. The four partial AndroidWorld runs use delta, full, html, and diffusion, respectively. Their logs contain 423 explicit Boolean decisions, comprising 408 negative and 15 positive responses. The denominator excludes 1,388 steps without an explicit Boolean response.

Outcome certainty. This probe first prepares candidate actions, then asks a separate question using the goal, recent history, current screenshot, and the candidates with their reasons. We use the following decision prompt.

> You are deciding whether it is worth using a world model to simulate the consequences before executing the first-choice action. **Strictly distinguish two kinds of certainty:**
>
> **Target certainty:** Am I certain that the first-choice action clicks or operates on the correct element on the screen?

> **Outcome certainty:** Am I certain what I will enter after execution, and that it **actually contains** what is needed to advance the goal? Or am I merely guessing from experience that it should be there, without having seen it?
>
> **Key point: target certainty does not imply outcome certainty.** Even if I know where to click, the outcome is uncertain whenever I am guessing whether the destination contains the target content.
>
> Output only JSON with `outcome_certain` set to true/false, `candidates_diverge` set to true/false, and `reason` set to an explanation.
>
> Set `outcome_certain=false` when the result is guessed or unverified, or the action has high risk and is irreversible. Set `candidates_diverge=true` when the candidates have clearly different outcomes and you are unsure which to choose.

The controller requests predictions when `outcome_certain` is false or `candidates_diverge` is true. In the saved AndroidWorld run, the agent returns the first condition on 222 of 265 decisions. Another seven decisions meet the second condition despite reporting a certain outcome, giving 229 steps with WM calls. Twelve further steps bypass this question because the run stops requesting predictions after 30 task steps. The call rate is therefore 229/277 (82.7%), while the uncertain-outcome rate is 222/265 (83.8%). These counts cover the saved partial run across four shards.

Actions and both WM decisions in one response. The MobileGym probe combines candidate proposal with both WM decisions in a single response. Its recorded fields include `think`, a `candidates` list, `use_world_model`, and `which_modality`. A positive call decision includes one model choice from delta, full, and html for the step; a negative decision leaves the model field null and executes the first candidate.

The four-step trace attempts to enable dark mode in Alipay. The agent opens the "My" page and then Settings. On the third and fourth steps, it clicks the same location for "Skin Centre", while the page remains at Settings. All four responses set `use_world_model=false` and `which_modality=null`; the trace contains no predictions and ends at the four-step limit without completing the task. The repeated click provides a concrete example of the agent declining predictions while making no progress.

C.13 Additional Harness Controls

Fixed controls require all relevant facts. Target repair requires an accessibility tree; source recovery requires one literal filename, its folder, and a matching current location. Transaction state starts only after the real GUI confirms that exact item. An unclear match preserves the proposal, while a failed prediction remains unavailable rather than becoming negative action evidence.

The strict state key combines foreground accessibility structure with a quantised screenshot to record exact state–action no-progress evidence. Counters reset per task and are never shared across parallel workers.

C.14 Repeated Full-System Runs

Table 3 reports the best run for each method in each evaluation setting. We repeat the complete WISH configuration with Qwen3-VL-8B five times on the same 116-task AndroidWorld suite. Table 12 reports the run-level mean, standard deviation, and observed range of completed tasks.

Table 12: Repeated WISH results with Qwen3-VL-8B over five 116-task AndroidWorld runs under the fixed-time protocol.

Metric	Mean ± s.d.	Observed range	Runs
Complete tasks	$(57 \pm 1)/116$	56–58/116	5