CUWORLDEVAL: LEARNING WORLDS FOR MULTI-TURN COMPUTER-USE AGENT EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating computer-use agents requires knowing whether they can complete tasks through interaction, not just match recorded actions. Offline action matching is inexpensive but cannot capture the consequences of alternative actions, while evaluation in real applications is costly to repeat. We introduce CUWorldEval, a learned environment that predicts screen descriptions and available controls after each action, allowing agents to attempt multi-turn tasks without executing the underlying applications. The key challenge is that plausible individual predictions do not guarantee reliable task evaluation. Errors can accumulate, change the agent's subsequent actions, and lead to task outcomes that differ from real software. We propose Multi-Turn On-Policy Distillation (Multi-Turn OPD), which combines error-conditioned teacher training with distillation on student-generated interaction histories. We first train a teacher to predict reference next states from the student's imperfect histories, using full reference-trajectory summaries to guide correction. We then freeze the teacher and distill it on fresh student rollouts under recorded actions. Each predicted state becomes part of the next turn's history, training the student to continue despite its own errors rather than only from accurate reference states. We evaluate the same agents on matched Word, Excel, and PowerPoint tasks in CUWorldEval and real software. For the two agents with offline baselines, CUWorldEval yields higher balanced accuracy, and it reaches up to 76% agreement with real-software task outcomes. These results support learned environments for interactive agent evaluation and show why their reliability must be assessed through complete task outcomes, not only individual predictions. Code is available at https://anonymous.4open.science/r/CUWorldEval-05EB.

1 INTRODUCTION

The goal of computer-use agent evaluation is to determine whether an agent can complete a task, not simply whether it chooses the expected action at a recorded step. This distinction matters in software such as Word, Excel, and PowerPoint, where an agent must observe the effects of its actions and decide how to continue. Offline benchmarks such as Mind2Web evaluate actions against recorded demonstrations (Deng et al., 2023), but cannot show what happens when an agent follows a different, potentially valid route. Interactive benchmarks such as WebArena and OSWorld instead execute agents in working environments and check task completion (Zhou et al., 2024; Xie et al., 2024). This captures the consequences of agent decisions, but requires environment setup, action execution, and output verification. Repeating these steps in virtual machines across Office tasks, agents, and trials motivates a less costly way to evaluate complete interactions. Windows Agent Arena addresses evaluation throughput through parallel virtual machines (Bonatti et al., 2024).

A learned environment offers a way to retain interaction without executing the underlying applications. WorldGym and WorldEval explore learned environments for robot-policy evaluation (Quevedo et al., 2026; Li et al., 2025), while ADWM studies evaluation of language agents (Liu et al., 2026) and SWE-World predicts execution feedback for software-engineering agents (Sun et al., 2026). We introduce **CUWorldEval** for multi-turn evaluation in Office applications (Figure 1). After each agent action, its world model predicts a textual screen description and the available controls. The agent uses this response to choose its next action, continuing until the run ends rather than being constrained to a reference trajectory. The world model acts as the environment, not as a planner or a training mechanism for the evaluated agent.

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

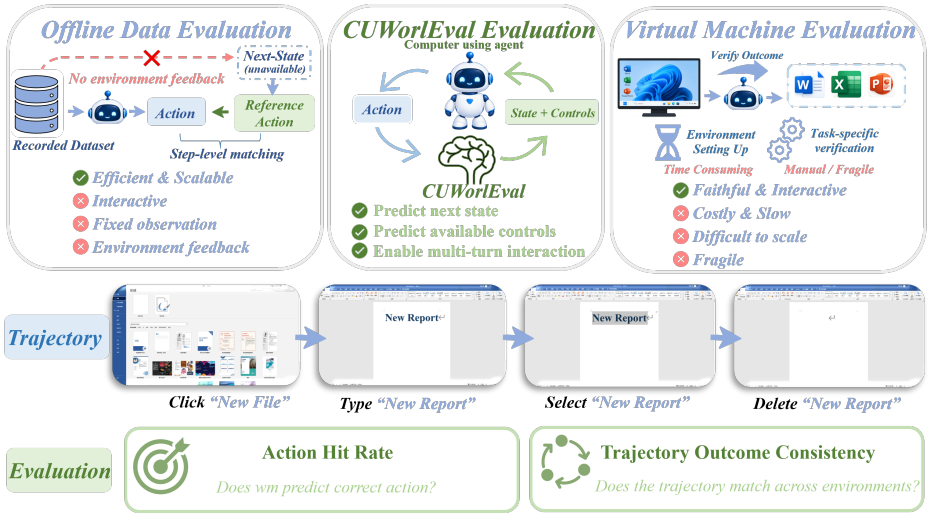


Figure 1: CUWorldEval predicts screen descriptions and available controls after agent actions. Multi-Turn OPD trains it on its own predicted histories with teacher supervision. Evaluation compares state predictions with recorded states and task outcomes with real-software runs.

The central question is whether this interaction is trustworthy: **does the learned environment agree with real software on whether an agent completes a task?** A plausible next screen is not enough. A missing control can block a capable agent, while a spurious one can make the task easier. Such errors accumulate because each predicted state affects the next action and prediction. One-step tests initialized from recorded states do not expose these effects. A useful evaluation environment must therefore be assessed by task outcomes as well as state similarity. This distribution shift resembles that addressed by DAgger and scheduled sampling (Ross et al., 2011; Bengio et al., 2015), but here histories contain model-predicted software states.

To address these errors, we propose **Multi-Turn On-Policy Distillation (Multi-Turn OPD)**. Response-level OPD trains on student-generated responses (Gu et al., 2024; Agarwal et al., 2024), but each recorded transition still begins from a reference history. Our method instead carries every predicted state into the next turn. We first train a teacher on multi-turn histories generated by a frozen SFT student, using reference next states as targets and full trajectory summaries to guide correction. We then freeze the teacher and distill it on fresh student rollouts under recorded actions. This prepares the teacher for student errors and trains the student to continue from imperfect predictions. Reference summaries are available only to the teacher during training. The design also builds on reference-conditioned self-distillation and privileged-information distillation (Zhao et al., 2026; Lopez-Paz et al., 2015).

We compare CUWorldEval and offline action matching against real-software execution. We first evaluate screen and control predictions against VM records under fixed actions in one-step and multi-turn settings. We then run the same agents on matched tasks in CUWorldEval and real software, allowing each run to follow the agent's decisions, and compare task outcomes. Across Word, Excel, and PowerPoint, CUWorldEval improves balanced accuracy over offline matching for both agents with offline baselines and reaches 76% agreement with real software. Multi-turn predictions are less reliable than one-step predictions, while Multi-Turn OPD improves outcome agreement over SFT alone. Remaining disagreement across agents shows that learned environments are useful evaluators but not yet substitutes for real software. The main contributions are:

- CUWorldEval allows agents to attempt multi-turn Office tasks in a learned environment, predicting screen descriptions and controls instead of executing the applications.
- We develop Multi-Turn OPD to train on histories containing the student's own prediction errors. A teacher is first trained to predict reference next states from these histories with access to trajectory summaries, then distilled into the student over fresh multi-turn rollouts.
- Experiments compare the same agents on matched tasks in the learned environment and real software. CUWorldEval exceeds offline action matching in balanced accuracy for both agents with offline baselines, while comparisons of one-step and multi-turn predictions show why state-prediction scores alone are insufficient for evaluating a learned environment.

2 Related Work

World models as policy evaluators. Learned world models increasingly evaluate robotic policies. WorldGym scores action-conditioned video rollouts with a vision–language model (Quevedo et al., 2026), while WorldEval uses latent-action-conditioned video to estimate real-robot performance (Li et al., 2025). dWorldEval uses discrete diffusion and a progress token (Li et al., 2026), and Veo-based simulation studies out-of-distribution and safety settings (Team et al., 2025). Tseng et al. (2025) compare policy values and rankings across video world models. CUWorldEval brings paired real/learned-world comparison to computer-using agents on Office tasks.

World models for language and software agents. Code2World predicts action-conditioned GUI states as renderable code (Zheng et al., 2026), complementing CUWM's state-transition representation (Guan et al., 2026). ADWM learns an autoregressive diffusion world model from behavior-policy trajectories and evaluates unseen LLM-agent policies without real-environment access (Liu et al., 2026); across four benchmarks, it reports 0.82 mean Spearman correlation with real-environment policy values. SWE-World uses an LLM surrogate to predict execution and test feedback for agent training and trajectory selection (Sun et al., 2026). CUWorldEval instead targets evaluator fidelity for Office agents through privilege-aware teacher preparation, multi-turn distillation, and state and task-outcome validation.

3 Method

CUWorldEval evaluates agents by letting them interact with predicted software states and checking whether their task outcomes agree with real execution. We first describe the shared interface that allows an agent to run in either environment (Section 3.1), then explain how Multi-Turn OPD trains the world model to continue from its own predictions (Section 3.2). The evaluation protocol and metrics are specified with the experimental setup in Section 4.1.

3.1 Evaluating Agents in a Learned Environment

For a task instruction x, both environments provide the same initial screenshot O_0 and initial state S_0. At each turn, the agent-facing state is $S_t = (d_t, c_t)$: a textual screen description d_t and a list of available controls c_t. The WM predicts each control's accessibility type and visible text, and the client assigns a numeric label before presenting the list to the agent. Thus, both environments expose the same label–text–type action contract. The agent receives this state and the interaction history, then selects a structured action such as clicking a control or typing a value.

In the virtual machine (VM), the action is executed in the real Office application. A screen observer describes the resulting screenshot, and the live accessibility tree supplies the controls. In the learned environment (WM), the world model predicts both outputs without executing the application. Thus, after the initial screenshot, agents in both environments act on textual descriptions and controls rather than newly supplied screenshots. Our comparison concerns real-software execution under this shared interface; it does not evaluate arbitrary screenshot-only agent interfaces. Appendix B specifies the state format, actions, and prompts.

Let $H_t^e = (x, O_0, S_0^e, A_0^e, \ldots, A_{t-1}^e, S_t^e)$ denote the history through the current state in environment e. The interaction loop is

$$A_t^e \sim \pi(\cdot \mid H_t^e), \qquad S_{t+1}^e \sim T_e(\cdot \mid H_t^e, A_t^e), \qquad e \in \{\mathrm{VM}, \mathrm{WM}\}, \tag{1}$$

where T_{VM} includes application execution and observation, while T_{WM} is the learned state predictor. These are transitions over agent-facing observations, not a claim that the textual state fully specifies the application's hidden state. Each loop produces a trajectory $\tau^e = (S_0^e, A_0^e, S_1^e, \ldots, A_{L_e-1}^e, S_{L_e}^e)$, where L_e is the number of actions in that run. At evaluation time, $T_{\mathrm{WM}}(\cdot \mid H, A) = p_S(\cdot \mid H, A)$ for the trained student defined below.

For evaluation, we hold the task, agent, prompt, and configuration fixed. A task-specific VM snapshot is restored before each real-software run, and the WM receives the matching initial observation and state. The two runs proceed independently: their actions and lengths may differ because their observations differ. We compare task outcomes rather than require the agent to follow the same trajectory. The world model therefore serves as the environment, without modifying the evaluated agent.

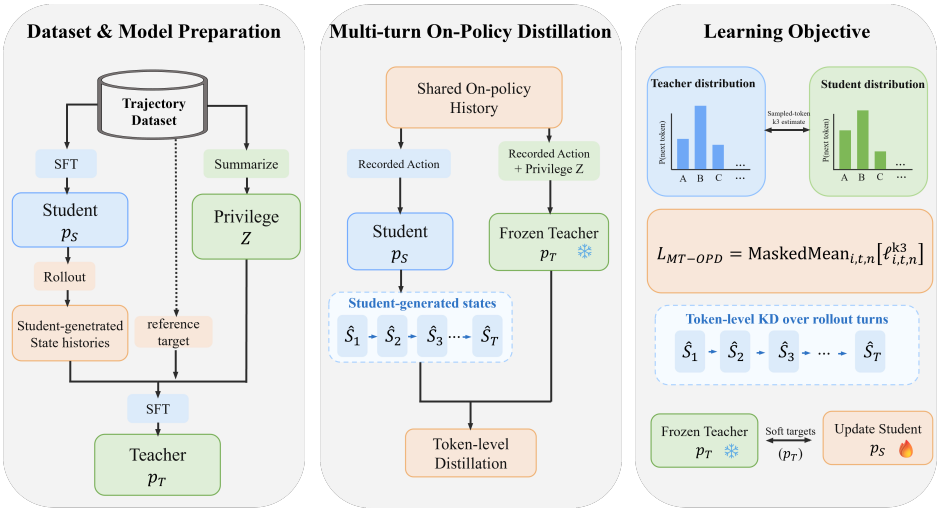


Figure 2: Multi-Turn OPD couples teacher preparation with student-generated multi-turn histories. We first train the teacher on the frozen SFT student's predicted histories, using reference summaries and next-state targets. We then freeze the teacher and distill it on fresh student rollouts, carrying each predicted state into the next turn. Student updates occur after collecting a complete batch.

3.2 Multi-Turn On-Policy Distillation

An inaccurate screen description or missing control can redirect the agent and affect all subsequent predictions. Standard supervised training on reference histories does not expose the world model to these accumulated errors. Our design addresses two linked problems: *which histories the student learns from*, and *how the teacher learns to supervise those histories*. We train a reference-guided teacher on the frozen student's actual prediction errors, then distill it on fresh multi-turn rollouts from the evolving student (Figure 2).

From response-level OPD to multi-turn state prediction. Consider response-level OPD applied separately to recorded transitions. The student generates a response from a fixed reference history, and the teacher supervises its generated token prefixes. However, the next training example still starts from a reference history: the student's predicted state does not determine what either model sees at the next turn. Sampling responses on-policy therefore does not, by itself, address accumulated state-prediction errors.

Multi-Turn OPD makes the history student-generated as well. Prior on-policy distillation methods supervise student-generated response prefixes (Gu et al., 2024; Agarwal et al., 2024), while OPSD additionally provides reference-conditioned teacher supervision (Zhao et al., 2026). In our setting, each predicted state is carried into later turns, so both the *within-response prefixes* and the *cross-turn histories* used for distillation come from the student, while actions remain fixed to the recorded sequence. Building on distillation with training-time privileged information (Lopez-Paz et al., 2015), we explicitly train the teacher on student-generated histories using reference summaries as privileged inputs and reference next states as supervised targets. The distinguishing design is this combination of error-conditioned teacher preparation and recursive student rollouts, not a new token-level divergence or merely a sum of independent one-step distillation losses. This comparison concerns response-level OPD applied to fixed reference histories, not every possible multi-turn distillation method.

Initializing the student. Each recorded trajectory contains $(x, O_0, A_{0:T-1}, S_{0:T})$. Here T is the recorded action count, S_t is a reference state, and $\hat{S}_t$ denotes a student prediction. We fine-tune the initial Qwen3.5 model p_{θ_0} on reference histories from the SFT split to obtain p_S. A separate OPD split supplies trajectories for multi-turn training (Section 4.1).

Error-conditioned teacher preparation. For each reference trajectory used to prepare the teacher, we aggregate descriptions of its step-wise progression into a complete summary

$$Z = F_{\text{sum}}(x, O_0, A_{0:T-1}, S_{0:T}). \tag{2}$$

We use an LLM to summarize the key points at each step of a reference trajectory, preserving their temporal order in a complete trajectory summary Z. The summary covers the entire reference trajectory and remains available to the teacher at every turn, including information about future reference steps. It provides privileged guidance for predicting reference next states from potentially inaccurate student-generated histories. The summary is not supplied to the student, and neither the teacher nor the summary is used during agent evaluation.

We freeze the SFT student and feed it the recorded actions in sequence, retaining its predicted states rather than replacing them with reference states. Starting from $\hat{S}_0 = S_0$, the rollout uses

$$\begin{aligned} \hat{H}_0 &= (x, O_0, S_0), \\ \hat{S}_{t+1} &\sim p_S(\cdot \mid \hat{H}_t, A_t), \\ \hat{H}_{t+1} &= \mathrm{Append}(\hat{H}_t, A_t, \hat{S}_{t+1}). \end{aligned} \tag{3}$$

Thus $\hat{H}_t$ includes the current predicted state, just as H_t^e includes the current state in Equation 1. We store training examples $(\hat{H}_t, A_t, Z, S_{t+1})$ in $\mathcal{R}$. The teacher p_T, initialized from p_{θ_0}, is then fine-tuned with the reference next state as its target:

$$\mathcal{L}_{\text{teacher}} = - \sum_{(\hat{H}_t, A_t, Z, S_{t+1}) \in \mathcal{R}} \log p_T(S_{t+1} \mid \hat{H}_t, A_t, Z). \tag{4}$$

State probabilities refer to the serialized description-and-control response. Only the teacher is updated in this stage. Its input preserves the student's accumulated errors, while its target and summary come from the reference trajectory. This explicitly trains reference-guided correction, rather than assuming a teacher trained only on clean histories can provide reliable supervision on erroneous ones. Importantly, the corrupted histories are produced by the student itself, not by manually perturbing reference states, and teacher supervision targets the reference next state rather than imitating the student's mistaken prediction.

Distilling on fresh multi-turn rollouts. We freeze the trained teacher and resume training the student. For each batch, the current student generates complete trajectories using Equation 3, with its parameters held fixed throughout collection. At each turn, both models condition on the same student history and recorded action; only the teacher additionally sees Z. The next history always contains the student's prediction, not the teacher's output or the reference next state. Unlike the fixed histories used to prepare the teacher, these rollouts are collected anew from the current student before each batch update. The teacher is trained once on the initial SFT student's histories and then held fixed; it is not retrained after each student update. This separates teacher preparation from the changing rollout distribution used for student distillation.

During fixed-action training, recorded actions are passed to the world model unchanged even when their targets are absent from the predicted control list. Such mismatches do not trigger a reset to reference states or automatically exclude the turn from training. If output sanitization repairs a response, the repaired response is retained in subsequent history, but the entire turn is excluded from the OPD loss. Per-turn generation budgets and supervision masks are detailed in Appendix A.4.

Learning objective. Let $\hat{S}_{t+1,<n}$ denote the first $n-1$ tokens of the serialized student response, and let M_{t+1} be its token count. For a student-generated token $y = \hat{S}_{t+1,n}$, denote its teacher and student probabilities by $p_T(y)$ and $p_S(y)$, conditioned on $(\hat{H}_t, A_t, Z, \hat{S}_{t+1,<n})$ and $(\hat{H}_t, A_t, \hat{S}_{t+1,<n})$, respectively. The implementation directly optimizes a clipped sampled-token `k3` loss rather than a full-vocabulary forward KL:

$$\begin{aligned} \delta_{t,n} &= \mathrm{clip}(\log p_T(y) - \log p_S(y), -c_\delta, c_\delta)\,, \\ \ell_{t,n}^{\text{opd}} &= \mathrm{clip}\big(e^{\delta_{t,n}} - \delta_{t,n} - 1, -c_\ell, c_\ell\big)\,. \end{aligned} \tag{5}$$

The clipping thresholds are specified in Appendix A.4. Teacher log-probabilities are treated as fixed targets. Gradients flow through the student log-probabilities at the collected tokens, not through token sampling.

For a batch $\mathcal{B}$ of student rollouts, indexed by i, let $m_{i,t,n}$ indicate valid generated response tokens. Injected action messages carry no loss, and repaired response turns are masked out. We normalize

the loss by the total number of valid response tokens:

$$\mathcal{L}_{\text{MT-OPD}}(\mathcal{B}) = \frac{\sum_{i\in\mathcal{B}}\sum_{t=0}^{T_i-1}\sum_{n=1}^{M_{i,t+1}} m_{i,t,n}\ell^{\text{opd}}_{i,t,n}}{\sum_{i\in\mathcal{B}}\sum_{t=0}^{T_i-1}\sum_{n=1}^{M_{i,t+1}} m_{i,t,n}}. \tag{6}$$

The sample index i was omitted in the preceding token-level equations for clarity. Token losses are evaluated at the collected student prefixes; student parameters are updated only after the full batch has been collected and its losses aggregated. There are no parameter updates between turns of a collected trajectory. The teacher remains fixed throughout distillation. At evaluation time, only the trained student supplies WM transitions; neither the teacher nor the reference summary is provided. Appendix A.4 and Figure 3 summarize the complete training procedure.

4 Experiments

We organize the experiments around three questions: **(1)** Does learned-world evaluation reflect real-software task outcomes more accurately than offline action matching (Section 4.2)? **(2)** Does one-step prediction quality reflect reliability over multiple steps (Section 4.3)? **(3)** Does Multi-Turn OPD improve evaluation beyond the SFT checkpoint and continued supervised training (Section 4.4)?

4.1 Experimental Setup and Evaluation Protocol

Data. We use GUI360 (Mu et al., 2025) to construct 10,861 SFT examples from 2,177 tasks. The OPD set retains 596 tasks from the SFT corpus and introduces 378 additional tasks, totaling 974 tasks. Evaluation uses 100 held-out Office tasks: 34 Word, 33 Excel, and 33 PowerPoint. Data and training details appear in Appendix A.

World models and agents. CUWorldEval is built on Qwen3.5-9B with SFT followed by Multi-Turn OPD; its intermediate checkpoint is denoted CUWorldEval (SFT only). We compare against open-weight models (Qwen3.5-9B, Qwen3.5-27B, and Qwen3-VL-32B), proprietary models (Grok-4.5, Gemini-3.5 Flash, and GPT-5.5), and Code2World (Zheng et al., 2026) in task-outcome evaluation. We evaluate three agents: GPT-5.5, Qwen3.5-27B, and Gemini-3.5 Flash. Interface and prompt details appear in Appendix B.

Offline action-matching baseline. The offline baseline predicts actions from recorded reference states without running an interactive environment. It labels a task successful when more than half of its predicted actions exactly match the recorded actions. These binary predictions are compared with the same VM outcome labels used to assess world models. This tests whether allowing the agent to interact provides a better estimate of task completion than matching a fixed trajectory.

Task-outcome protocol. Each agent solves the 100 test tasks with one rollout per task in each world model and five independent rollouts per task in the VM. Because the environments may lead the agent to different actions, trajectories are compared through their outcomes. GPT-5.2 judges every rollout's final state against the task's completion reference in five independent passes. This yields five judgments per task for each WM and 25 for the VM. Majority labels a WM task successful when at least three of its five judgments are positive. The VM reference label directly pools all 25 judgments, requiring at least 13 positive votes; this label is held fixed for all WM comparisons rather than recomputed for each judge pass. For Mean $\pm$ Std., each of the five WM judge passes supplies one label per task on the same fixed trajectories. We compute each metric over the 100 tasks against the fixed VM reference labels for each pass, then report the mean and sample standard deviation of the five metric values. Thus, the standard deviation reflects judge variability, not variation across independently generated WM rollouts. For the offline baseline, Majority compares its fixed task labels with the pooled VM reference. Its Mean $\pm$ Std. instead compares those fixed labels with five VM judge-pass label sets: within each pass, the five VM rollout judgments are majority-aggregated per task. The five resulting metric values measure VM-side judge variability, not repeated offline predictions. Appendix A.11 separately validates the task-completion judge against six human annotators.

Task-outcome metrics. Successes and failures need not be equally frequent in real-software runs, and their proportions can differ across agents. Overall agreement can therefore be dominated by the more frequent outcome, masking poor recognition of the less frequent one. We use balanced

accuracy (BA) as the primary metric to give equal weight to recognizing VM successes and VM failures. Taking the VM label as reference, TP and TN count tasks that succeed or fail in both environments; FP counts success only in the WM, and FN counts success only in the VM. For an agent evaluated on N tasks, we report

$$\mathrm{BA} = \frac{1}{2}\left(\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} + \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}\right), \qquad \text{Agreement} = \frac{\mathrm{TP}+\mathrm{TN}}{N}. \tag{7}$$

The two terms in BA are the recall of VM successes and the recall of VM failures. Raw agreement, called Consistency in the tables, reports the fraction of tasks with matching outcomes and is retained as a complementary measure. The positive/negative average is macro-F1: the mean of the F1 scores treating success and failure as the positive class in turn. The same definitions apply to offline predictions against VM labels. All methods use the same reference-based VM judgments, which remain subject to judge error.

State-prediction protocol. We feed the recorded action sequence of each test task to the world model and compare predicted states with the VM records. In the multi-turn setting, the world model conditions on its own previous predictions and carries its predicted control list forward, so errors can accumulate. In the single-turn setting, each prediction uses the reference history and controls. Both settings keep actions fixed, unlike task-outcome evaluation, in which the agent chooses its actions.

State-prediction metrics. Description consistency is the mean continuous semantic-consistency score against the recorded description, expressed on a 0–100 scale. Hit Rate is the fraction of eligible recorded actions whose target text appears in the available controls before execution; Text+Type additionally matches control type. Macro and Micro IoU compare predicted next-state controls with the next recorded control list, averaging per transition and pooling intersections and unions, respectively. The denominators differ: action-target coverage is measured over eligible recorded actions, whereas next-state overlap is measured over transitions.

4.2 Agreement with Real-Software Task Outcomes

We first test whether CUWorldEval gives more accurate task-outcome predictions than offline action matching. Table 1 reports the comparison for GPT-5.5 and Qwen3.5-27B.

Closer to VM outcomes than offline action matching. CUWorldEval achieves balanced accuracy of 74.40 with GPT-5.5 and 74.82 with Qwen3.5-27B, compared with 63.41 and 61.95 for offline action matching. The improvements are 10.99 and 12.87 percentage points. Raw task-outcome agreement rises from 61% to 71% and from 63% to 76%, respectively. Thus, for these agents, evaluating their interaction with a trained world model better reflects VM task outcomes than matching recorded actions.

Accounting for imbalanced task outcomes. The results illustrate why overall agreement should be considered alongside balanced accuracy. With GPT-5.5 as the agent, Gemini-3.5 Flash has higher raw agreement than CUWorldEval (73% versus 71%) but lower balanced accuracy (66.04 versus 74.40). CUWorldEval therefore performs better when recognition of VM successes and failures is weighted equally, despite agreeing on fewer tasks overall. It achieves the highest balanced accuracy for both agents in Table 1.

4.3 One-Step versus Multi-Turn Prediction

We next ask whether accurate one-step predictions are sufficient to assess the world model as an interactive evaluator. Recorded actions are held fixed in this experiment, isolating the effect of conditioning on predicted rather than reference histories.

Predicted histories expose errors hidden by one-step tests. GPT-5.5's description-consistency score falls from 60.92 in single-turn replay to 43.33 in multi-turn replay. CUWorldEval reaches 72.24 in the multi-turn setting and has the best Text+Type, Macro IoU, and Micro IoU among the models in Table 2, although its hit rate remains lower than GPT-5.5's (57.82 versus 60.79). Full single-turn results appear in Appendix C.

Table 1: WM–VM task-outcome agreement (%) for GPT-5.5 and Qwen3.5-27B. Per task: WM, 1 rollout × 5 judge passes; VM, 5 rollouts × 5 judge passes. Majority compares the five-vote WM label with the fixed 25-vote VM label. Mean ± Std. is over five judge passes: WM passes for WM rows and VM passes for gray offline baselines. Bold and underlining mark the best and second-best values per agent.

World model	Consistency		Balanced Acc.		Positive/Negative Avg.	
	Majority	Mean ± Std.	Majority	Mean ± Std.	Majority	Mean ± Std.
GPT-5.5 as Agent						
Offline Data Evaluation	61.00	59.60 ± 0.55	63.41	62.02 ± 0.58	60.52	59.04 ± 0.60
Code2World	44.00	43.80 ± 0.84	53.63	53.21 ± 0.97	42.53	42.43 ± 0.84
Qwen3.5-9B	57.00	56.60 ± 2.19	60.33	59.76 ± 2.18	56.79	56.36 ± 2.16
Qwen3.5-27B	68.00	67.00 ± 2.24	64.84	63.93 ± 2.68	64.84	63.86 ± 2.60
Qwen3-VL-32B	56.00	56.80 ± 1.30	55.60	56.62 ± 1.79	54.53	55.40 ± 1.51
Grok-4.5	71.00	70.20 ± 1.10	61.87	60.86 ± 1.56	61.84	60.55 ± 2.01
Gemini-3.5 Flash	**73.00**	**74.80** ± 0.84	66.04	68.22 ± 1.40	66.88	**69.24** ± 1.52
GPT-5.5	72.00	72.40 ± 1.34	63.96	64.40 ± 1.42	64.49	64.98 ± 1.65
CUWorldEval	71.00	69.00 ± 1.41	**74.40**	**72.33** ± 1.50	**70.64**	68.64 ± 1.41
Qwen3.5-27B as Agent						
Offline Data Evaluation	63.00	63.40 ± 1.95	61.95	62.28 ± 2.03	61.90	62.14 ± 2.03
Code2World	61.00	61.80 ± 0.84	55.04	55.65 ± 0.81	52.15	52.52 ± 0.95
Qwen3.5-9B	61.00	60.20 ± 2.39	56.16	55.33 ± 2.35	54.81	53.90 ± 2.53
Qwen3.5-27B	65.00	65.60 ± 1.52	64.76	65.04 ± 1.66	64.40	64.82 ± 1.60
Qwen3-VL-32B	66.00	66.00 ± 0.71	65.23	65.08 ± 0.75	65.11	65.01 ± 0.74
Grok-4.5	68.00	67.60 ± 0.89	71.39	70.98 ± 0.92	67.88	67.48 ± 0.90
Gemini-3.5 Flash	68.00	68.00 ± 1.22	69.16	69.24 ± 1.28	67.88	67.90 ± 1.23
GPT-5.5	68.00	67.80 ± 1.48	70.65	70.48 ± 1.67	67.99	67.77 ± 1.48
CUWorldEval	**76.00**	**76.40** ± 0.55	**74.82**	**75.16** ± 0.67	**75.00**	**75.37** ± 0.62

Table 2: Multi-turn offline trajectory replay on the 100 test tasks. Each prediction is conditioned on the model's own previous predictions. All values are on a 0–100 scale. Bold and underlining mark the best and second-best values per column.

World model	Desc. Cons.	Hit Rate	Text+Type	Macro IoU	Micro IoU
Qwen3.5-9B	24.02	40.20	37.62	55.41	54.88
Qwen3.5-27B	32.21	52.28	42.18	61.30	57.88
Qwen3-VL-32B	37.50	50.30	40.20	54.24	50.35
Gemini-3.5 Flash	28.86	54.46	45.74	49.20	48.96
GPT-5.5	43.33	**60.79**	50.50	57.53	56.68
CUWorldEval	**72.24**	57.82	**54.65**	**66.55**	**59.76**

Better one-step scores need not imply better task evaluation. The comparison between the SFT-only and OPD-trained models is particularly informative (Table 3). OPD slightly lowers single-turn description consistency, from 79.00 to 78.71, while improving multi-turn consistency from 70.93 to 72.24 and task-outcome balanced accuracy for all three agents in the ablation. One-step control scores also decrease, whereas four of the five multi-turn replay metrics improve. Thus the benefit of training cannot be assessed solely from clean-history prediction scores. We compare absolute multi-turn scores and task outcomes, not just the size of the single-to-multi-turn drop, which can shrink when one-step scores worsen.

Table 3: Prediction and evaluation quality need not improve together. Description consistency is on a 0–100 scale; task-outcome balanced accuracy (BA) is a percentage. Values summarize the existing replay and ablation results.

	Description Consistency		Task-Outcome BA		
World model	One-step	Multi-turn	GPT-5.5	Qwen3.5-27B	Gemini-3.5 Flash
CUWorldEval (SFT only)	**79.00**	70.93	66.70	64.96	57.65
CUWorldEval	78.71	**72.24**	**74.40**	**74.82**	**65.50**

4.4 Effect of Multi-Turn OPD

We compare the initial model, its SFT checkpoint, continued SFT, and the Multi-Turn OPD model with the teacher prepared as in Section 3.2. This distinguishes gains over the initial checkpoint from gains over an additional supervised-training stage. In OPD (SFT-student), the final SFT student checkpoint is frozen and used directly as the teacher, without the separate error-conditioned teacher preparation stage. OPD (SFT-teacher) instead uses the privilege-trained teacher described in Section 3.2.

Table 4: Training-stage ablations on Qwen3.5-9B (%). Top: multi-turn replay; bottom: Majority WM–VM task-outcome agreement (Cons.: consistency; Bal.: balanced accuracy; P/N: positive/negative average). Continued SFT uses the 974-task OPD corpus for three epochs. Bold and underlining mark the best and second-best values per column within each block.

Offline Trajectory Replay					
World model	Desc. Cons.	Hit Rate	Text+Type	Macro IoU	Micro IoU
Qwen3.5-9B	24.02	40.20	37.62	55.41	54.88
+ SFT	<u>70.93</u>	<u>56.24</u>	<u>52.67</u>	<u>65.07</u>	**60.42**
+ OPD (SFT-teacher)	**72.24**	**57.82**	**54.65**	**66.55**	<u>59.76</u>

Task-Outcome Agreement									
World model	GPT-5.5 as Agent			Qwen3.5-27B as Agent			Gemini-3.5 Flash as Agent		
	Cons.	Bal.	P/N	Cons.	Bal.	P/N	Cons.	Bal.	P/N
Qwen3.5-9B	57.00	60.33	56.79	61.00	56.16	54.81	48.00	51.69	36.86
+ SFT	67.00	66.70	65.48	<u>67.00</u>	<u>64.96</u>	<u>65.16</u>	56.00	57.65	54.84
+ SFT + Continued SFT	63.00	64.29	62.15	65.00	62.90	63.05	59.00	59.46	59.00
+ OPD (SFT-student)	<u>70.00</u>	<u>71.15</u>	<u>69.06</u>	58.00	52.50	50.00	<u>64.00</u>	<u>64.09</u>	<u>63.94</u>
+ OPD (SFT-teacher)	**71.00**	**74.40**	**70.64**	**76.00**	**74.82**	**75.00**	**65.00**	**65.50**	**65.00**

Beyond the SFT checkpoint. SFT accounts for most of the improvement in multi-turn description consistency, raising the score from 24.02 to 70.93. Multi-Turn OPD increases it to 72.24 and improves three of the four control metrics; Micro IoU decreases from 60.42 to 59.76. More importantly for evaluation, OPD improves task-outcome balanced accuracy over SFT by 7.70, 9.86, and 7.85 percentage points for GPT-5.5, Qwen3.5-27B, and Gemini-3.5 Flash, respectively. All three task-outcome metrics improve for these agents. This supports assessing the training stage by its task-level evaluation performance as well as by local state-prediction scores.

Continued-SFT control. Continued SFT on the 974-task OPD corpus for three epochs yields balanced accuracy of 64.29%, 62.90%, and 59.46% with GPT-5.5, Qwen3.5-27B, and Gemini-3.5 Flash, respectively. The trained-teacher OPD variant reaches 74.40%, 74.82%, and 65.50%, so the improvements over this continued-SFT control are 10.11, 11.92, and 6.04 percentage points. Additional supervised training on the same OPD corpus therefore does not reproduce the gains in this comparison. However, training budgets are not matched, and this control does not separately identify the contribution of recursive histories versus error-conditioned teacher preparation. Establishing those contributions requires matched-budget and component-wise controls.

5 Conclusion

We introduced CUWorldEval for evaluating computer-use agents in interactive learned environments. Its paired VM–WM protocol measures transition fidelity and task-outcome agreement, showing that one-step prediction alone cannot establish evaluator reliability. Multi-Turn OPD trains on student-generated histories to reduce compounding errors. Across Word, Excel, and PowerPoint, it improves multi-turn replay and outcome agreement over SFT, reaching 76% agreement with VM evaluation. Learned worlds remain agent-, task-, and criterion-dependent proxies that require validation against real execution. Broader task coverage, independent terminal-state checks, and bias analysis are needed to establish when such substitution is reliable.

AI USE STATEMENT

We used large language models to assist with language polishing and translation, organizing research materials, literature searches and citation formatting, preparing tables and figures, and reviewing experimental design and results. The use of language models as world models, agents, and judges is described in the method and experimental sections. The authors take responsibility for the final manuscript, including the accuracy of its claims, data, and references.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28, 2015.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

Yiming Guan, Rui Yu, John Zhang, Lu Wang, Chaoyun Zhang, Liqun Li, Bo Qiao, Si Qin, He Huang, Fangkai Yang, et al. Computer-using world model. *arXiv preprint arXiv:2602.17365*, 2026.

Yaxuan Li, Yichen Zhu, Junjie Wen, Chaomin Shen, and Yi Xu. Worldeval: World model as real-world robot policies evaluator. *arXiv preprint arXiv:2505.19017*, 2025.

Yaxuan Li, Zhongyi Zhou, Yefei Chen, Yaokai Xue, and Yichen Zhu. dworldeval: Scalable robotic policy evaluation via discrete diffusion world model. *arXiv preprint arXiv:2604.22152*, 2026.

Kaixuan Liu, Guojun Xiong, Weinan Zhang, and Shengpu Tang. Autoregressive diffusion world models for off-policy evaluation of llm agents. *arXiv preprint arXiv:2606.05558*, 2026.

David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. *arXiv preprint arXiv:1511.03643*, 2015.

Jian Mu, Chaoyun Zhang, Chiming Ni, Lu Wang, Bo Qiao, Kartik Mathur, Qianhui Wu, Yuhang Xie, Xiaojun Ma, Mengyu Zhou, et al. GUI-360°: A comprehensive dataset and benchmark for computer-using agents. *arXiv preprint arXiv:2511.04307*, 2025.

Julian Quevedo, Ansh Kumar Sharma, Yixiang Sun, Varad Suryavanshi, Percy Liang, and Sherry Yang. Worldgym: World model as an environment for policy evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 78932–78957, 2026.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Shuang Sun, Huatong Song, Lisheng Huang, Jinhao Jiang, Ran Le, Zhihao Lv, Zongchao Chen, Yiwen Hu, Wenyang Luo, Wayne Xin Zhao, et al. Swe-world: Building software engineering agents in docker-free environments. *arXiv preprint arXiv:2602.03419*, 2026.

Gemini Robotics Team, Krzysztof Choromanski, Coline Devin, Yilun Du, Debidatta Dwibedi, Ruiqi Gao, Abhishek Jindal, Thomas Kipf, Sean Kirmani, Isabel Leal, et al. Evaluating gemini robotics policies in a veo world simulator. *arXiv preprint arXiv:2512.10675*, 2025.

Wei-Cheng Tseng, Jinwei Gu, Qinsheng Zhang, Hanzi Mao, Ming-Yu Liu, Florian Shkurti, and Lin Yen-Chen. Scalable policy evaluation with video world models. *arXiv preprint arXiv:2511.11520*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models, 2026. *URL https://arxiv. org/abs/2601.18734*, 2, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

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

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

A Experimental Setup and Reproducibility

A.1 Data Sources and Splits

The SFT corpus is drawn from GUI360 (Mu et al., 2025), whose Office trajectories come from four instruction sources: Bing search queries, Microsoft 365 help articles, a question-answering benchmark, and WikiHow articles. Each trajectory is a sequence of transitions; a transition contains the pre-action screenshot, the accessibility tree, the recorded action and its target control, and the post-action screenshot. For every evaluation task we additionally store the natural-language request, a filtered initial control list given to both the agent and the world model, and a structured completion reference used by the task-completion judge (Appendix A.10).

The SFT corpus contains 2,177 tasks: 2,005 successful trajectories and 172 failed trajectories. Converting every transition into a multimodal ShareGPT training example produces 10,861 step-level samples. The domain distribution is 783 Excel, 747 PowerPoint, and 647 Word tasks.

The OPD corpus contains 974 tasks, comprising 886 successful and 88 failed trajectories with 2,887 recorded transitions. It is constructed by selecting a subset of the preceding corpus according to trajectory length and supplementing it with additional data. Specifically, 596 OPD tasks are also present in the 2,177-task SFT corpus, while the remaining 378 are not: 366 come from the `add886` source and 12 from `fail196`. Splits and overlaps are computed at the task level. The 100-task evaluation set is held out from the training data.

A.2 Supervised Fine-Tuning Setup

We initialize the student world model from Qwen3.5-9B and train it on the 10,861 step-level examples described above using LLaMA-Factory. We update the language-model parameters without low-rank adapters, while freezing the vision tower and multimodal projector. Training uses four NVIDIA A100 GPUs with BF16 computation and DeepSpeed ZeRO-3. A per-device batch size of one and 16 gradient-accumulation steps give an effective batch size of 64. Table 5 lists the training configuration.

Table 5: Supervised fine-tuning configuration of the Qwen3.5-9B world model. The SFT baseline is the final checkpoint after 850 optimizer updates, not a checkpoint selected by validation performance.

Parameter	Value
Backbone	Qwen3.5-9B
Trainable modules	Language model; vision tower and projector frozen
Training examples	10,861 transitions from 2,177 tasks
Training epochs / optimizer updates	5 / 850
Optimizer	Fused AdamW (PyTorch)
Peak learning rate	1×10^{-5}
Adam β_1, β_2, ϵ	0.9, 0.999, 10^{-8}
Learning-rate schedule	Cosine; 10% warmup (85 updates)
Per-device batch / gradient accumulation	1 / 16
Effective global batch	64 examples
Maximum sequence length	8,192 tokens
Chat template	`qwen3_5_nothink`
Maximum image pixels	262,144
Precision / distributed strategy	BF16 / DeepSpeed ZeRO-3
Gradient checkpointing	Enabled
Hardware	4× NVIDIA A100
Random seed	42
Recorded training time	17.02 hours

Empty-control supervision. An empty control list receives special loss masking. When an assistant target contains the literal block `<available_controls>` followed by an empty JSON list and the closing tag, tokens from the start of that block through the end of the assistant target are assigned the ignore index -100. The preceding state-description tokens remain supervised; the example itself is not discarded. This masking is a training-time rule, rather than a restriction that generated states must always contain controls.

Training completion and checkpoint. The recorded run completes five epochs with 170 optimizer updates per epoch. We use its final 850-update checkpoint for the SFT baseline and as the student initialization for the reported OPD run. The logged training duration is 61,261 seconds; no validation-best checkpoint is recorded. The archived model card records Transformers 5.6.0, PyTorch 2.12.0+cu130, Datasets 4.0.0, and Tokenizers 0.22.2.

A.3 Teacher Preparation and Supervision

The HF850 teacher is trained on 10,861 step-level examples constructed from the 2,177-task student self-rollout corpus described below and the corresponding ground-truth next states. Student-generated histories provide the input context, while the ground-truth next-state description and controls provide the supervised target. Its archived model card identifies this dataset as `opd_teacher_multiclear_latestonly`, and its trainer state records five completed epochs and 850 optimizer updates. Table 6 summarizes its training configuration. The resulting teacher weights are held fixed throughout OPD.

Table 6: Teacher-SFT configuration. The teacher is trained on student-generated histories paired with ground-truth targets and privileged trajectory summaries.

Parameter	Recorded value
Backbone family	Qwen3.5-9B
Learning rate	1×10^{-5}
Per-device batch size	1
Gradient-accumulation steps	16
Number of training GPUs	4
Effective global batch size	64 examples
Completed epochs / optimizer updates	5 / 850
Updates during OPD	None; teacher remains frozen

Archived student self-rollouts. The archived Qwen3.5-9B SFT self-rollout corpus contains 2,177 task files and 10,861 generated steps, verified by enumerating the files and their step records. The final integrity audit marks the corpus complete with no reported structural errors; this checks artifact completeness, not prediction correctness. Each task follows the recorded action sequence, starting from the original screenshot and initial controls. The strict-history protocol uses the student's generated descriptions and controls for subsequent states.

The corpus contains 2,116 tasks generated with this strict-history protocol and 61 preserved tasks from an earlier implementation. The latter use a different prompt-rewriting procedure, and two contain a truncated intermediate assistant output. They are retained as a separately identified legacy subset, not treated as protocol-identical strict rollouts. The strict run records 232 sanitized steps, with per-task metadata retaining the distinction between generated and repaired outputs.

Strict-history generation uses four independent single-GPU vLLM replicas, with up to 16 concurrent tasks per GPU. The configuration sets temperature to 1, top-k to 1, and top-p to 1, yielding greedy decoding through the top-k restriction. Thinking is disabled, each response is capped at 4,087 tokens, and generation stops after `</available_controls>`. The configured serving context limit is 262,144 tokens. These settings describe the offline corpus-collection run, not the subsequent OPD optimization or evaluation runs.

Constructing supervision from student histories. The HF850 teacher-training dataset pairs the archived student self-rollouts with the corresponding ground-truth transitions. For each reference step, it constructs an example containing the initial screenshot, task instruction, initial controls, and action sequence, interleaved with the student's previous state predictions. The final assistant message supplies the reference next-state description and post-action controls. The complete `multi-clear` trajectory summary is appended once to the system message and remains available throughout the example. It is teacher-only information: it is not included in the student's rollout input.

Only the final reference assistant response is supervised. Prompt tokens and historical student assistant messages are excluded from the teacher-SFT loss (`train_on_prompt=false`,

`mask_history=true`). The *latest-only* transformation preserves the descriptions of earlier student states but clears their control arrays, retaining the controls of the most recent historical state. This transformation does not replace the student's historical predictions with ground-truth states.

A.4 Multi-Turn OPD Configuration

The reported OPD run initializes the student from the final SFT checkpoint and uses a separate, frozen Qwen3.5-9B teacher trained with whole-trajectory privilege (Appendix A.3). The teacher checkpoint is the archived five-epoch, 850-update teacher, not the student checkpoint or a later teacher ablation. Each rollout recursively retains the student's generated history while replaying the recorded actions. The complete `multi-clear` trajectory summary is provided only to the teacher. Table 7 gives the resolved configuration of this run. The training manifest contains 974 unique tasks and 2,887 transitions; 55 repeated padding rows produce 1,029 task rows, or 49 batches of 21 per epoch. The trainer performs synchronous updates: after each update, refreshed student weights are used for the next rollout batch. Asynchronous rollout workers do not imply stale-policy training.

Multi-Turn OPD: training algorithm

Input: initial model p_{θ_0}; SFT and OPD trajectory splits.
Student initialization: $p_S \leftarrow \mathrm{SFT}(p_{\theta_0})$.
Teacher preparation: freeze p_S; set $\mathcal{R} \leftarrow \varnothing$.
for each teacher-preparation trajectory **do**
  Build Z using Eq. 2; $\hat{H}_0 = (x, O_0, S_0)$.
  for $t = 0, \ldots, T-1$ **do**
    Store $(\hat{H}_t, A_t, Z, S_{t+1})$ in $\mathcal{R}$.
    Generate $\hat{S}_{t+1} \sim p_S(\cdot \mid \hat{H}_t, A_t)$.
    $\hat{H}_{t+1} \leftarrow \mathrm{Append}(\hat{H}_t, A_t, \hat{S}_{t+1})$.
Initialize p_T from p_{θ_0}; train only p_T on $\mathcal{R}$ via Eq. 4.
Multi-turn distillation: freeze p_T; enable student updates.
repeat until the training budget is exhausted
  Select trajectories and summaries Z_i.
  Collect a fresh rollout batch $\mathcal{B}$ with p_S fixed.
  for each trajectory i **do**
    $\hat{H}_{i,0} = (x_i, O_{i,0}, S_{i,0})$.
    for $t = 0, \ldots, T_i - 1$ **do**
      Generate $\hat{S}_{i,t+1} \sim p_S(\cdot \mid \hat{H}_{i,t}, A_{i,t})$.
      Store history, action, summary, and response.
      $\hat{H}_{i,t+1} \leftarrow \mathrm{Append}(\hat{H}_{i,t}, A_{i,t}, \hat{S}_{i,t+1})$.
  Evaluate $\ell^{\mathrm{opd}}_{i,t,n}$ at stored prefixes via Eq. 5.
  Aggregate $\mathcal{L}_{\text{MT-OPD}}(\mathcal{B})$ via Eq. 6; update only p_S.
Return: trained student p_S for agent evaluation.

Figure 3: Multi-Turn OPD pseudocode. Reference next states prepare the teacher; teacher distributions supervise student distillation. Updates occur only after collecting a complete batch.

Numerical clipping. In Equation 5, we set $c_\delta = 20$ and $c_\ell = 10$. The teacher–student log-probability difference is clipped to $[-20, 20]$ before exponentiation, and the resulting token-level `k3` loss is clipped to $[-10, 10]$. These are numerical-stability settings of the implementation, not constants derived from the distillation objective.

Generation and supervision masks. The per-turn generation budget reserves nine tokens of the nominal 4,096-token limit for output sanitization. Recorded action messages carry zero loss weight. Normally generated response tokens carry unit weight; when the sanitizer repairs a response, that turn is masked out of the optimization loss. The masked token losses are normalized by the global number of valid response tokens. Dynamic optimization batch sizing is disabled; a full batch of 21 tasks across three student GPUs with microbatch size one corresponds to seven gradient-accumulation rounds. The implementation uses the sampled-token loss detailed in Section 3.2, with task rewards and policy-gradient terms disabled. Although forward computation and rollout use BF16, the training configuration retains FP32 parameters and gradient reduction.

Table 7: Multi-Turn OPD configuration. The full run completes 245 updates; the checkpoint used in the reported replay and closed-loop evaluations is update 147, at the end of epoch three.

Parameter	Value
Student initialization	Final Qwen3.5-9B SFT checkpoint (update 850)
Teacher	Frozen Qwen3.5-9B privilege-trained checkpoint
Training tasks	974
Global rollout / optimization minibatch	21 / 21 tasks
Microbatch per student GPU	1
Optimization passes per rollout batch	1
Student / teacher GPUs	3 / 1
Rollout workers	6
Optimizer	AdamW
Learning rate / schedule	2×10^{-6} / constant, no warmup
Adam β_1, β_2 / weight decay	0.9, 0.999 / 0.01
Gradient clipping	1.0
Log-probability-ratio clipping threshold c_δ	20
Token-loss clipping threshold c_ℓ	10
Forward / rollout precision	BF16
Rollouts per task	1
Temperature / top-k / top-p	1 / 1 / 1 (greedy through top-k)
Thinking / actor and rollout seed	Disabled / 42
Initial prompt / response budgets	8,192 / 24,576 tokens
Per-turn nominal / generation budgets	4,096 / 4,087 tokens
Student / teacher context limits	32,769 / 39,937 tokens
Training of reported checkpoint	3 epochs, 147 updates
Full archived run	5 epochs, 245 updates
Checkpoint interval	49 updates

Checkpoint selection. We use the checkpoint after three OPD epochs (147 updates, with 49 updates per epoch) for the reported offline replay and closed-loop evaluations. This checkpoint was selected on a small held-out validation set rather than on the 100-task test set. Although the archived run continued to five epochs and 245 updates, those later updates are not part of the checkpoint used for the reported results.

A.5 Offline Replay Implementation

The reported replay runs each cover the same 100 tasks and 524 recorded transitions, with one run per model and history setting. Both settings retain the initial screenshot and task instruction. Single-turn replay uses recorded actions and descriptions as history and the current recorded control list; multi-turn replay uses the same actions but generated descriptions and the previously predicted control list. History is serialized as action–description pairs, without repeating historical control arrays. The SFT baseline uses its final 850-update checkpoint; CUWorldEval uses the OPD checkpoint after 147 updates (epoch three).

For the reported CUWorldEval replay, the archived launch configuration uses BF16 inference with four-way tensor parallelism, a 32,768-token context limit, a 4,096-token response cap, and disabled thinking. Requests use the sampling configuration of temperature zero, top-p one, and seed 42. Four task workers are used, with a 3,600-second task timeout and one retry. Recorded actions are injected even when their controls are absent from the predicted state. An empty predicted control list is propagated without substitution from the reference state.

Description score. A GPT-5.5 text-only judge compares the predicted description with the recorded reference description on eight dimensions: application name, action, title bar, ribbon, main editing area, side panel, navigation, and status bar. The respective weights are $0.8, 1.4, 1.0, 1.1, 1.5, 0.8, 0.6, 0.8$, summing to eight; each dimension is assigned a score of 0, 0.5, or 1. A length penalty is subtracted from the weighted mean: no penalty is applied within the integer bounds corresponding to 75–125% of the reference token count, and a linear penalty based on relative distance outside these bounds is capped at 0.25. The final score is clipped to $[0, 1]$. To-

ken counts use the `cl100k_base` tokenizer, with a character-count fallback. An empty prediction scores zero; missing references or failed judge calls are excluded. All 524 transitions in the reported CUWorldEval run receive a score. Tables report the mean continuous score multiplied by 100, not the fraction exceeding a threshold. For example, its multi-turn mean is 72.24, whereas its separately recorded pass rate at threshold 0.75 is 52.48%.

Action coverage and control overlap. Hit Rate checks whether the recorded action target appears in the available controls immediately before that action: recorded controls at the initial step and the preceding predicted controls thereafter. Text matching trims and collapses whitespace and ignores case; Text+Type additionally matches a lowercased control type with non-alphanumeric characters removed. Steps with `select_text` or missing structured target metadata are excluded. The resulting denominator is 505 actions, including 93 initial-state checks and 100 final-action checks. CUWorldEval matches 292 targets by text and 276 by text and type, giving 57.82% and 54.65%, respectively.

IoU instead compares the controls predicted after an action with the recorded controls at the next step. It uses the 424 non-final transitions. Control sets are deduplicated by type–text pairs after whitespace normalization and case folding; unlike the coverage metric, punctuation in types is retained. Entries with both fields empty are discarded. Per-transition IoU is one when both sets are empty. Macro IoU averages these transition-level values, whereas Micro IoU divides the sum of intersection sizes by the sum of union sizes. Thus, coverage and IoU have different eligible-step sets and denominators.

A.6 VM Execution and Observation

AutoOffice provides the real-environment backend using VMware Workstation and a Windows 11 guest. The available VM configuration specifies two virtual CPUs and 4 GB RAM. A host-side controller manages the VM through `vmrun`, while an in-guest Flask service exposes observation, action, and application-setup endpoints. The service uses Windows UI Automation via `pywinauto` to enumerate controls, `pyautogui` for mouse and keyboard execution, and Office COM interfaces for application-specific setup. The exact Office build, display resolution, and locale are not recorded in the examined run metadata; we do not infer them from host settings.

Task initialization. The reset procedure restores the configured VM baseline, starts the guest, and waits for its service to become healthy. Task-specific initialization then closes the target application, waits one second, opens the task's template file, waits four seconds, and dismisses startup dialogs. Failure to close the application or open the template is treated as a setup error. Thus, a baseline snapshot and a task-specific template jointly define the starting state. For each task, all reported VM evaluations restore the same snapshot and use the same task template across agents and repeated runs. Neither the 29-task nor the 71-task subset changes its initialization between runs.

VM agent configurations. Table 8 summarizes the VM settings used for the reported evaluations, including Table 1. All evaluation agents use `final` prompt mode throughout the reported runs. The configuration uses `text_history`, observer `gpt-5.2-20251211`, snapshot restoration enabled, and continuous video recording disabled. The configured fallback step limit is 15, while the saved task records use an effective limit of twice the reference trajectory length.

Table 8: VM settings for the reported evaluations. All agents use `final` mode and the same snapshot for each task across five VM rollouts, covering both the 29-task and 71-task subsets. The WM client exposes the same action contract after assigning control labels.

Agent	Prompt mode	Snapshot across runs
GPT-5.5	`final`	Fixed per task
Qwen3.5-27B	`final`	Fixed per task
Gemini-3.5 Flash	`final`	Fixed per task

The `final` agent contract requests a numeric control label together with matching text and type, enforcing agreement among all three fields in both environments. The WM client supplies these labels from control-list order before the agent is called (Appendix B). The available VM history formatter retains the most recent eight action–description pairs and appends the current state; its saved requests preserve the resulting history text.

Observation and action execution. A single observation request captures a screenshot and accessibility controls back-to-back in the guest. The returned control labels refer to that observation's cached control list; they are execution handles, not persistent identifiers across states. A control-targeted click resolves the label within that observation and clicks the center of its screen rectangle. The current server caps accessibility enumeration at 500 controls with a 40-second traversal budget. Client timeout limits are 50 seconds for observation, 15 seconds for screenshot retrieval, 60 seconds for ordinary actions, and 155 seconds for opening a template. These are implementation limits, not measurements of average execution latency.

Textual observer and trace artifacts. In text-history mode, the post-action screenshot is converted into a textual state by a separate observer. An archived execution identifies this model as `gpt-5.2-20251211`; its saved prompt matches the task-aware template in Figure 6. The observer receives the current screenshot and task instruction, not a predicted next state. The implementation caps its response at 1,200 tokens, sets temperature zero and seed 42, and defaults to five attempts with exponential backoff capped at 16 seconds. These sampling and retry values are source-level defaults, rather than independently archived parameters for every run. Per-step traces preserve the before/after screenshots, control lists, action and execution result, description prompts and responses, and accumulated textual history. These artifacts support reconstruction even when continuous video recording is disabled.

A.7 Closed-Loop Inference Configuration

For the CUWorldEval checkpoint evaluated with the GPT-5.5 agent, the archived inference server uses vLLM 0.17.1, BF16 weights, two-way tensor parallelism, and a 32,768-token context limit. Thinking is disabled and the world-model response limit is 4,096 tokens. The image processor uses a maximum of 262,144 pixels and a minimum of 65,536 pixels, with one initial screenshot per prompt. The initial filtered accessibility list is loaded from the task record. Subsequent transitions use the generated description and control list; no teacher privilege is provided during evaluation.

The aligned run uses three concurrent task workers. A task is allowed one retry after its initial attempt, and its elapsed time is checked against a 1,200-second acceptance threshold after execution. Tasks that remain incomplete are rerun separately with a 7,200-second threshold, and their recovered trajectories replace only the corresponding failed records. The archived GPT-5.5–CUWorldEval run recovers one task this way. These execution retries are distinct from independent evaluation repeats and from repeated LLM judgments.

Termination and judge configuration. The rollout ends on `finish`, after twice the reference trajectory length, or when the WM predicts an empty control list. In the last case, the generated description is retained for judging rather than replaced with a reference state. An invalid control action is mapped to `wait(1)` and reuses the current state without calling the WM. The archived task-completion judge uses GPT-5.2 (`gpt-5.2-20251211`), prompt version `v0`, five judge passes, and one call per trajectory within each pass. One WM rollout per task yields five judgments, while five VM rollouts yield 25. The WM majority uses its five judgments, and the VM majority over all 25 judgments is fixed as the outcome reference. Its context-aware prompt includes the initial screenshot, action–description context, the completion reference, and the predicted final description; it does not provide the predicted control arrays to the judge.

A.8 Test-Set Statistics

Table 9 summarizes the 100 test tasks, which contain 524 ground-truth transitions (5.24 steps per task on average). The "Init. Controls" column is the size of the filtered initial control list that the agent and the world model receive; it excludes per-cell `DataItem` entries in Excel, which would otherwise dominate the list. Before filtering, the raw accessibility tree contains on average 374.5 controls per step in Excel (capped at 500), 77.5 in Word, and 59.0 in PowerPoint.

Table 10 breaks the 524 ground-truth actions down by function and by the accessibility type of the clicked control. Navigation clicks on ribbon tabs and menu entries are the largest category and are most frequent in PowerPoint, whereas Excel trajectories contain the most typing (formula and value entry) and the most cell selections. By trajectory length, 28 tasks require 2–3 steps, 31 require 4–5, 27 require 6–8, and 14 require 9–12. PowerPoint is the most long-horizon domain, with 20 of its 33 tasks requiring at least six steps.

Table 9: Statistics of the 100 test tasks. Steps are ground-truth trajectory lengths; Init. Controls is the size of the filtered initial control list. Sources: Bing search / Microsoft 365 / QA benchmark / WikiHow.

Domain	#Tasks	Avg. Steps	Min / Max	Init. Controls	Sources	Main Task Types
Word	34	4.76	2 / 12	73.30	23 / 3 / 4 / 4	formatting, text editing, insertion, review tools, file options
Excel	33	4.61	2 / 12	67.60	25 / 6 / 2 / 0	formulas, cell editing, cell formatting, data tools, workbook settings
PowerPoint	33	6.36	2 / 12	49.40	21 / 7 / 5 / 0	slide creation, layout and design, object insertion, view settings
All	100	5.24	2 / 12	63.50	69 / 16 / 11 / 4	–

Table 10: Distribution of ground-truth actions (%). Clicks are grouped by the control type of the target: Nav. = `TabItem`/`MenuItem`; Button = `Button`/`CheckBox`/`RadioButton`/`ComboBox`; List = `ListItem`; Cell/Editor = `DataItem`/`Edit`/`Document`. `select_text` is available only in Word.

Domain	#Actions	Nav. Click	Button Click	List Click	Cell/Editor Click	Type	Select Text	Scroll/Other
Word	162	27.20	29.60	11.10	4.90	12.30	11.10	3.70
Excel	152	21.70	26.30	4.60	18.40	28.90	–	0.00
PowerPoint	210	47.10	31.40	11.40	0.50	9.00	–	0.50
All	524	33.60	29.40	9.40	7.10	15.80	3.40	1.40

A.9 Example Tasks

Table 11 lists representative tasks from each domain together with their ground-truth action sequences.

Table 11: Example test tasks with their ground-truth length and action sequence.

Domain	Instruction	Steps	Ground-truth actions
Word	Insert a text watermark 'Confidential' in the Word document.	5	5 × click
Word	Bold the entries in the list in the Microsoft Word document.	8	4 × (select_text, click)
Word	Get suggestions for the misspelled word 'ageny' in the Microsoft Word document.	4	4 × click
Excel	Apply conditional formatting to highlight cells in the 'Age' column that are greater than 30.	6	4 × click, type, click
Excel	Subtract the value in cell B3 from the value in cell B2 and display the result in cell H4 using a formula.	4	2 × type, 2 × click
Excel	Remove duplicate values from the 'Country' column in the Excel spreadsheet.	8	8 × click
PowerPoint	Create a bubble chart on a new slide in the PowerPoint presentation.	8	8 × click
PowerPoint	Add a comment to Slide 1 in the PowerPoint presentation.	4	2 × click, type, click
PowerPoint	Hide background graphics on Slide 1 in PowerPoint for Windows.	9	9 × click

A.10 Task-Completion References

For each task, GPT-5.5 generates a structured completion reference from the request and the ground-truth trajectory. The reference consists of a `task_completion_state`, a list of `required_ui_points`, `acceptable_variations`, and `failure_signals`. The acceptable variations make the reference tolerant to valid alternative solutions. For example, for "Enable editing and input multiple items separated by a comma in a single cell," any worksheet cell and any item names are accepted as long as the comma-separated items remain in one cell.

Reference loading. The archived judging configuration requests the `new_generated` reference source. The evaluator uses the stored new reference when it is nonempty; otherwise it falls back to the recorded `train_format` reference, and then to the saved ground-truth description. The actual source is retained in each judgment record. Generation metadata preceding `task_completion_state:` are removed before prompting, so model names, file paths, and generation-status headers are not evaluation evidence. A requested source name alone does not guarantee that a revised reference was available for every task.

References used by the reported CUWorldEval runs. We inspect the saved task-completion judgments for CUWorldEval with GPT-5.5 and Qwen3.5-27B as agents. In each run, all 100 tasks (29 `minidata` and 71 `add_task`) record `new_generated` as the actual reference source; none falls back to `train_format`. SHA-256 hashes of the UTF-8 reference strings match for every task across its five judge passes and across the two agents. Thus, these two runs use identical task-wise reference text. This verifies reference consistency across the inspected runs, rather than attributing a particular reference-generation or correction procedure solely from its source label.

A.11 Human Validation of the Task-Completion Judge

Prior work documents both human agreement and systematic biases in LLM-based evaluation of conversational responses (Zheng et al., 2023); those findings do not establish correctness for Office task completion. In a separate annotation study, we validate individual task-completion judgments on a 29-task subset using three VM trajectories per task, giving 87 task–run pairs. This study assesses trajectory-level judging, not the 25-vote pooled VM reference labels used for agent outcome evaluation. Six human annotators judge task completion from the final-state text. For each pair, the human reference label is the majority of the valid human votes. The automatic label is the majority of five independent GPT-5.2 judge calls. Ten pairs have a tied human vote (3–3) and are excluded from the primary agreement calculation because they do not have a unique human-majority label. On the remaining 77 pairs, the final judging configuration agrees with humans on 71 pairs, yielding $71/77 = 92.2\%$ agreement.

The validation uses the same representation presented to the judge throughout our evaluation. VM terminal screenshots are converted into the textual state schema that also defines the WM training target, while the VM and WM paths share the completion-reference schema, judge prompt, and decision interface. This alignment controls modality and formatting differences and provides a common basis for comparing outcomes across environments.

Table 12 compares the evaluator-design variants. The final configuration uses the corrected structured completion reference generated from the task and final-state evidence, together with the task-aware state description. The unweighted agreement rate is the primary metric.

Table 12: Agreement between automatic task-completion judgments and human-majority labels. The primary analysis excludes 10 task–run pairs with tied human votes. "Weighted" assigns more weight to examples on which annotators agree more strongly. All rates are percentages. Bold and underlining mark the best and second-best agreement rates.

Configuration	Completion reference	State description	Agree / Total	Agreement	Weighted
Original	Human-written	Original	51 / 77	66.20	67.80
GPT-text reference	GPT-5.5, text evidence	Original	67 / 77	<u>87.00</u>	<u>88.20</u>
Image reference + task-aware description	GPT-5.5, text and image evidence	Task-aware	65 / 77	84.40	85.50
Final configuration	Corrected image-based reference	Task-aware	71 / 77	**92.20**	**95.00**

For the supplementary consensus-weighted analysis, a task–run pair i is assigned weight w_i according to its human-vote split: 1.0 for 6–0 or 5–0, 0.7 for 5–1 or 4–1, 0.5 for 4–2, and 0.3 for 3–2. The weighted agreement is

$$\text{Agreement}_{\text{weighted}} = \frac{\sum_i w_i \mathbb{1}[\hat{y}_i = y_i]}{\sum_i w_i}. \tag{8}$$

For the final configuration, the weighted numerator and denominator are 60.2 and 63.4, respectively, giving $60.2/63.4 = 95.0\%$. As a conservative sensitivity analysis, if the ten 3–3 ties are retained with weight 0.1 and always counted as disagreements, the score is $60.2/(63.4 + 10 \times 0.1) = 93.5\%$. This weighting is reported only as a secondary robustness analysis; the directly interpretable unweighted $71/77 = 92.2\%$ result is our primary human-validation statistic. Together, these results validate task-completion judging on the shared textual representation used by our evaluation pipeline. Applying the same judge, prompt, reference format, and state schema to every WM further supports controlled comparisons among models and training stages. Extending the human study to a broader range of generated terminal states would provide an additional validation dimension.

B Shared VM–WM Interface and Prompts

This section instantiates the shared interface of Section 3.1. In both environments, the agent receives the same four inputs—the task instruction, the initial screenshot O_0, a textual interaction history, and

the current control list—and emits one structured action. The environments differ only in how the next state is produced: the VM executes the action, describes the resulting screenshot, and reads the controls from the live accessibility tree, whereas the world model predicts both.

B.1 State Representation

A state $S_t = (d_t, c_t)$ consists of a textual description d_t and an agent-facing JSON control list c_t. The WM's raw control predictions contain accessibility type and visible text, without numeric labels or screen rectangles. Before presenting the list to the agent, the client assigns numeric labels according to the predicted list order. Each agent-facing entry therefore contains a label, text, and type in both environments. The agent copies the entry's `label` into its action's `control_label` field. Labels identify entries within the current observation, not persistent controls across states; their values need not match between a VM and a WM rollout. This formatting step does not add missing controls or consult reference future states. The interaction history uses the same format in both environments. Figure 4 illustrates the labeled agent-facing list and textual history.

Control List c_t

```
[{"label": 1,
  "control_type": "TabItem",
  "control_text": "Design"},
 {"label": 2,
  "control_type": "MenuItem",
  "control_text": "Watermark"},
 {"label": 3,
  "control_type": "Button",
  "control_text": "Page Color"},
 ...]
```

Interaction History

```
Execution History (previous steps in this task):
--- Step 0 ---
Action: {"function": "click", "args": {...},
         "status": "CONTINUE"}
Resulting State: <description d_1>
...
--- Current Step ---
Current State:
<description d_t>
```

Figure 4: Shared agent-facing state format. Left: a control list after label assignment (illustrative label values). Right: the textual interaction history.

B.2 Action Space

Actions are JSON tool calls with a `function`, an `args` object, and a `status` (`CONTINUE` or `FINISH`). The supported functions are `click(control_label, coordinate, button, double)`, `type(text)`, `key(keys)`, `scroll(x, y, amount)`, `wait(seconds)`, and `finish(reason)`; Word additionally supports `select_text(text)`. For an accessibility-control click, the `args` object must include a numeric `control_label` together with `control_text` and `control_type`, all copied from the same entry of the current control list. The same three-field contract is used in both environments; labels are assigned by the client rather than predicted by the WM. Coordinates are allowed only when the target is absent from the list. `key` is restricted to a single standalone key such as Enter, Tab, or Esc; keyboard shortcuts are disallowed so that every operation is grounded in a visible control. During agent-in-the-loop evaluation, an action that refers to a control absent from the current list is replaced by `wait(1)`, and the state is left unchanged. This validity rule is not applied to fixed-action replay or OPD training, where recorded reference actions are passed to the transition model unchanged. An online rollout ends when the agent issues `finish`, after twice the ground-truth number of steps, or when the WM predicts no available controls; in the last case its final description is retained for judging.

B.3 Prompts

Figures 5–8 reproduce the complete source templates for the text-and-control agent, task-aware VM observer, world model, and context-aware task-completion judge. Only line wrapping is changed; braces denote runtime substitutions, and doubled braces escape literal JSON braces in the Python templates. The observer text is additionally matched to a saved VM request. The archived evaluation configuration selects the context-aware judging template, version `v0`, and the non-strict, control-conditioned WM template; agent/WM source templates are versioned separately from the judge. Not every helper module was copied into the historical run snapshots, so source-template availability is distinguished from byte-for-byte verification of every past request.

The initial screenshot, instruction, history, and current controls populate the agent and WM contexts. The observer instead receives a current VM screenshot and task context. The WM response supplies

the description and control list parsed into d_{t+1}^{WM} and c_{t+1}^{WM}. Figure 5 gives the `final`-mode agent source templates implementing the shared label–text–type contract. In the reported evaluations, this contract is used for both VM observations and WM predictions after client-side label assignment. The unlabeled control format in Figure 7 is the WM generation target, not the format presented to the agent. For each evaluated agent, its prompt and output configuration are held fixed across the two environments.

Agent Prompt: Final-Mode Source Templates

```
You are an expert in Office Application automation and graphical user interfaces
with accessibility support.

You will be provided with an optional initial screenshot, textual state history,
current live accessibility controls, the task instruction, and supported actions.

Generate the single best next action to accomplish the task. Prefer a live
accessibility control over coordinates.

Click target contract:
- For every click on an accessibility control, copy all three fields from the same
  live Accessibility Information entry: numeric control_label, exact control_text,
  and exact control_type.
- control_label MUST be an integer, never a name such as "New Slide" or "Insert".
- Example: {"function":"click","args":{"control_label":55,"control_text":"New Slide",
  "control_type":"Button"},"status":"CONTINUE"}.
- Do not invent a target that is absent from Accessibility Information.

Keyboard restriction:
- The key action may press only a single standalone key such as Enter, Tab,
  Backspace, Delete, Esc, or an arrow key.
- Do NOT use keyboard shortcuts or modifier-key combinations (e.g. Ctrl+S, Ctrl+C,
  Ctrl+Shift+L, Alt+Tab). Click the corresponding ribbon button or menu item instead.

Underspecified tasks:
- If the instruction does not specify an exact target or content (e.g. which cell,
  which value), do NOT give up or finish without acting. Make a reasonable choice
  yourself and complete the task; any sensible target and non-empty content that
  satisfies the instruction is acceptable.
- Only finish without acting if the task is genuinely impossible in the current state.

Output a valid JSON array containing exactly one element. The element must contain
 a tool_call object with function, args, and status. Do not output extra text.

Strict final click contract:
- Every accessibility-control click MUST include control_label, control_text,
  and control_type.
- Copy all three values from the same current Accessibility Information entry.
- The runtime rejects the click if any field is missing or if the label's live text
  or type differs.
- Never combine a label from one entry with text or type from another entry.
```

```
Task instruction:
{instruction}

Screenshot Description / State History:
{screen_description}

Accessibility Information:
{a11y}

Supported actions:
{actions}

{image_note}

Return exactly one next action as JSON.
For clicks, include numeric control_label plus matching control_text and control_type
from one Accessibility Information entry.

FINAL MODE: A control click is valid only when control_label, control_text, and
control_type exactly identify the same live Accessibility Information entry.
```

Figure 5: Final-mode agent instruction and user source templates implementing the shared three-field click contract. The two blocks are concatenated into one user message, with the initial screenshot O_0 attached when available.

VM State Observer: Complete Task-Aware Prompt

```
You are a precise observer of Windows Office UI screenshots.
```

```
Your task is to describe the current VM screen as a precise textual UI state.

You are provided with exactly one screenshot of a Windows desktop running a
Microsoft Office application or related UI. The screenshot is the only
source of truth.

For additional context, the user is working on the following task:
{task_instruction}

Important:
- The task is provided only as context to help you focus your description.
  The screenshot remains the only source of truth.
- Describe the current screen only. Do not predict the next screen, and do
  not assume the task is complete.
- Pay particular attention to and describe in more detail the UI regions
  and content that are relevant to the task; for the rest of the screen,
  keep the description as usual.
- Describe only what is visible in the screenshot. Do not invent state
  that is not shown, even if the task implies it.
- If the application is Word, Excel, or PowerPoint, use terminology
  consistent with that app.
- All visible UI text should be enclosed in double quotes.

Write a single coherent paragraph in top-down order when applicable:
- Title Bar / window identity
- Ribbon / toolbar / menu state
- Main editing area or document canvas
- Sidebar / dialog / popout if visible
- Status bar or navigation area if relevant

Be concrete about positions and state, such as the active tab, selected
object, visible cell range, visible slide, open dialog, or active pane.
If the screenshot is unclear, say what is unclear without guessing.

Format your response exactly as follows:

<current_screen_description>
[Your current screen description here]
</current_screen_description>
```

Figure 6: Complete VM observer prompt. The upper block is the system message; the lower block accompanies a high-detail current screenshot in the user message. The returned description forms d_{t+1}^{VM}.

World Model: Complete Control-Conditioned Prompt

```
Your task is to predict and describe the likely appearance of the Next UI
Screenshot, AND predict the list of available UI controls in the next state.

You are acting as a World Model assistant for Office applications such as
{app_name}. Your output will be used for multi-step action planning: the
predicted UI description guides the next action choice, and the predicted
controls list enables subsequent action selection.

You are provided with:
- A visual reference for the task: for step 0, this is the current Office
  UI screenshot; for later steps, this is the initial Step 0 screenshot
  together with the trajectory history in the state context section
- The current state's available UI controls (accessibility information)
- A structured user action
- A function-level description of the GUI action and its arguments

Important:
- The Next UI Screenshot is not available; generate a plausible prediction
  of what it would look like.
- Use the provided visual reference, trajectory history when present, and
  current controls list to understand the current UI state.
- Many controls remain unchanged between steps; use the current controls
  as a strong prior for predicting the next state's controls.
- Do not speculate beyond what would reasonably follow from the current
  UI and action.

Reference Information:

Visual Reference / State Context:
{image}

Current Available Controls:
{current_ally}

Action:
{action}

GUI Action Description:
{gui_description}

Format your response exactly as follows:

<next_state_description>
[A single coherent paragraph describing the next state. Mention changed
UI regions and visible text in double quotes.]
</next_state_description>
<available_controls>
[A JSON array of controls. Each item should include control_type and
control_text.]
</available_controls>
```

Figure 7: Complete non-strict WM source template. The `image` placeholder contains task and state-context text; the initial screenshot is attached separately. The planning wording is preserved from the implementation rather than rewritten to match the paper's evaluation terminology.

Runtime substitutions. The agent's `actions` placeholder lists the function signatures in Appendix B, with `select_text` included only for Word. Its `image_note` indicates whether the initial screenshot is attached and directs the agent to the latest textual observation and current controls. The WM's `image` placeholder includes the task instruction and either "See the attached image." at step zero or the action–description history at later steps. The `current_ally` and `action` fields are serialized JSON. The function-level descriptions supplied to the WM through `gui_description` are reproduced below. Their text-and-type wording describes the target's semantics; it does not replace the agent's three-field output contract.

World-Model Action Descriptions

```
click: Click a visible UI control by its exact control_text and control_type,
  or use a coordinate only when no suitable control exists.
type: Type literal text into the focused control.
key: Press a single standalone key to enter or confirm input, such as Enter,
  Tab, Backspace, Delete, Esc, or an arrow key. Do NOT use keyboard shortcuts
  or modifier combinations (e.g. Ctrl+S, Ctrl+Shift+L, Alt+Tab, Win+D);
  accomplish such operations by clicking the corresponding ribbon button
  or menu item instead.
scroll: Scroll the current view at a screen position by a signed amount.
wait: Wait briefly for the UI to settle.
finish: Mark the task as completed with a short reason.
select_text: Select the requested text in the Word document.
```

Reference-based task-completion judge. The archived judge configuration selects the context-aware `v0` template in Figure 8. Its context contains recorded rollout actions and generated descriptions, with the initial screenshot attached when available. It does not include later real screenshots, predicted control arrays, or teacher privilege. The reference text is loaded and stripped of its generation header as described in Appendix A.10.

Task-Completion Judge: Complete v0 Template

```
You are an impartial LLM judge. Determine whether a predicted final UI state
matches the ground truth final UI state, indicating task completion.

Use the execution context to understand the task flow. Minor cosmetic
differences are acceptable; functional mismatches are not.

Output ONLY valid JSON:
{{
  "reason": "detailed explanation",
  "complete": "yes/no"
}}

Task Instruction:
{instruction}

{trajectory_context}

Ground Truth Final State:
<<<
{gt_state}
>>>

Predicted Final State:
<<<
{pred_state}
>>>
```

Figure 8: Context-aware task-completion judge, version `v0`, selected by the archived WM evaluation run. Braces are preserved in Python-template form.

B.4 Paired Rollout Example

Table 13 shows GPT-5.5 as the agent on the Word task "Insert a text watermark 'Confidential' in the Word document," illustrating one VM trajectory and the CUWorldEval trajectory. It is a trajectory-level illustration, not the full comparison of the five-vote WM label with the 25-vote VM reference. The agent chooses the same four actions in both environments, and both final states contain the diagonal "CONFIDENTIAL" watermark, so the task outcome is consistent.

C Single-Turn Offline Replay Results

Table 14 reports offline trajectory replay in the single-turn setting, where every prediction is conditioned on the ground-truth history and the ground-truth controls of the current state. This setting isolates one-step prediction quality and does not expose the model to its own errors.

Every world model is substantially more accurate in the single-turn setting than in the multi-turn setting of Table 2; for example, the description consistency of GPT-5.5 drops from 60.92% to 43.33%.

Table 13: Paired VM/WM rollout of the same agent on the same task (descriptions abridged). $|c|$ is the size of the control list returned after the action.

t	Agent action	VM next state	WM next state
0	click("Design", TabItem)	"Design" tab active; "Document Formatting" gallery; "Page Background" group with "Watermark", "Page Color", "Page Borders". $\|c\| = 108$	"Design" tab active; "Document Formatting" gallery with "Themes", "Colors", "Fonts", "Paragraph Spacing". $\|c\| = 76$
1	click("Watermark", MenuItem)	Watermark dropdown open: "CONFIDENTIAL 1", "CONFIDENTIAL 2", "DO NOT COPY 1", ..., "Custom Watermark...". $\|c\| = 130$	Watermark dropdown anchored under "Page Background" with a "Confidential" section of preview thumbnails. $\|c\| = 87$
2	click("CONFIDENTIAL 1", ListItem)	Dropdown closed; large light-gray diagonal "CONFIDENTIAL" watermark on the page. $\|c\| = 109$	Dropdown closed; light-gray diagonal "CONFIDENTIAL" watermark centered across the page. $\|c\| = 76$
3	finish	Final state shows the watermark.	Final state shows the watermark.

Table 14: Single-turn offline trajectory replay on the 100 test tasks. All values are percentages. Bold and underlining mark the best and second-best values per column; pale blue highlights CUWorldEval.

World model	Desc. Cons.	Hit Rate	Text+Type	Macro IoU	Micro IoU
Qwen3.5-9B	46.57	50.10	46.93	71.38	68.44
Qwen3.5-27B	62.04	60.99	53.86	76.47	72.34
Qwen3-VL-32B	61.95	59.21	52.28	75.88	72.01
Gemini-3.5 Flash	55.28	66.93	60.99	71.49	69.26
GPT-5.5	60.92	**71.29**	63.37	75.79	**72.83**
CUWorldEval (SFT only)	**79.00**	67.72	**64.55**	**77.59**	70.43
CUWorldEval	78.71	62.38	59.80	70.86	65.18

One-step metrics therefore overestimate how reliably a world model can sustain a closed-loop rollout.

Table 15 compares the two trained models' single-turn and multi-turn results. In the single-turn setting, the SFT-only model is the strongest, and CUWorldEval matches it in description consistency (78.71% vs. 79.00%) but is lower on the control-list metrics. This is consistent with the training objective: Multi-Turn OPD supervises the model on histories induced by its own rollouts rather than on ground-truth prefixes, so it does not optimize one-step prediction under clean histories. In exchange, CUWorldEval drops by only 4.3–6.5 points from single-turn to multi-turn replay, compared with 8.1–12.5 points for the SFT-only model, and it is better on four of the five multi-turn metrics (Table 4). Because an evaluator always operates on its own predicted histories, we regard multi-turn replay and agreement with VM evaluation as the relevant measures of evaluator quality.

Table 15: Single-turn (ST) versus multi-turn (MT) replay for the two trained world models. Δ is the multi-turn minus single-turn score (percentage points). Bold and underlining compare the models within each setting; for Δ, larger values indicate smaller drops. Pale blue highlights CUWorldEval.

World model		Desc. Cons.	Hit Rate	Text+Type	Macro IoU	Micro IoU
CUWorldEval (SFT only)	ST	**79.00**	**67.72**	**64.55**	**77.59**	**70.43**
	MT	70.93	56.24	52.67	65.07	**60.42**
	Δ	-8.07	-11.48	-11.88	-12.52	-10.01
CUWorldEval	ST	78.71	62.38	59.80	70.86	65.18
	MT	**72.24**	**57.82**	**54.65**	**66.55**	59.76
	Δ	**-6.47**	**-4.56**	**-5.15**	**-4.31**	**-5.42**

D LIMITATIONS

Our human validation of task-completion judging focuses on VM-executed trajectories represented in the same textual state schema used throughout WM training and evaluation. This design aligns the judge interface, prompt, reference format, and observation modality across environments and supports controlled comparisons among world models. Future work can broaden this validation with additional samples from diverse generated-state distributions and alternative evaluator configurations.