# Task-Adaptive Rubrics for GUI Reward Modeling

**Tao Xiong[1], Xavier Hu[1], Wenkai Wang[1], Qinzhuo Wu[2], Changqiao Wu[2], Pengzhi Gao[2], Wei Liu[2], Jian Luan [2], Shengyu Zhang[1,‡]**

[1]Zhejiang University, [2]MiLM Plus, Xiaomi Inc.

**Correspondence:** {xiongtao@zju.edu.cn,sy_zhang@zju.edu.cn}

## Abstract

Recent studies on GUI agents have increasingly focused on *outcome reward modeling*, which assigns outcome rewards by judging whether an executed trajectory satisfies the success criteria implied by the user instruction. Existing GUI reward verifiers, however, often under-specify how these criteria should be constructed for **each task instance**. Whether using generic rubric structures or implicit model reasoning, their judging criteria are not sufficiently task-adaptive: they can transfer checks across tasks, overlook concrete constraints in the current instruction, or become overly strict by enforcing unstated requirements. To address this limitation, we propose ADAPTRUBRIC, a Coarse-to-Fine Rubrics Framework that constructs task-adaptive judging criteria through a category-level coarse stage and an instance-level fine stage. ADAPTRUBRIC performs **category-level coarse rubric retrieval** by routing the instruction to a GUI task family and retrieving reusable task-family criteria, then conducts **instance-level fine rubric generation** to surface compact cues for concrete values, scopes, and constraints in the current instruction. Across offline reward evaluation and online reinforcement learning optimization, ADAPTRUBRIC consistently outperforms prior reward agents, improving F1 by 3.6 points over the baseline average under a matched image budget and yielding a 4.23-point task-success gain.

## 1 Introduction

Graphical User Interface (GUI) agents (Hu et al., 2025; Zhang et al., 2025; Liu et al., 2025a; Wang et al., 2024) have emerged as a promising paradigm for automating realistic digital devices. Enhancing GUI agents relies heavily on reliable outcome reward models (ORMs), which can filter high-value trajectories (Xia et al., 2025; Chen et al., 2025b; Xiong et al., 2025) from costly interactions, support benchmark evaluation and inference-time trajectory selection (Rawles et al., 2025; Xie et al., 2024), as well as provide reward signals (Xu et al., 2025, 2026; Li et al., 2026) for reinforcement learning.

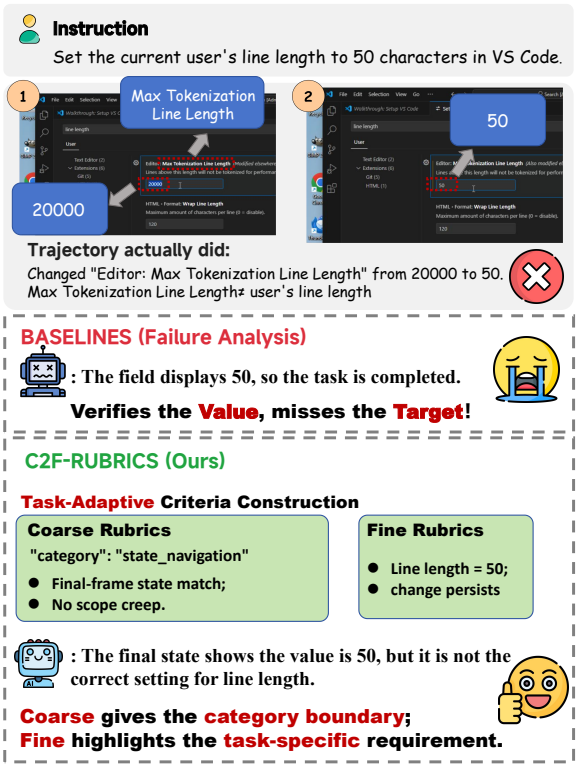



Figure 1: A motivating example for task-adaptive verification. The trajectory sets a related VS Code option to 50, but misses the requested line-length setting. While baselines verify only the surface value, ADAPTRUBRIC constructs coarse-to-fine criteria and correctly identifies the failure.

To determine whether a GUI trajectory succeeds, a reward verifier must first identify the success criteria (Gupta et al., 2025; Xie et al., 2026; Gunjal et al., 2025) implied by the instruction. These criteria specify the goal to be achieved, the instance-specific constraints to satisfy, and the visual evidence needed to support the final judgment. Exist-

‡Corresponding Author

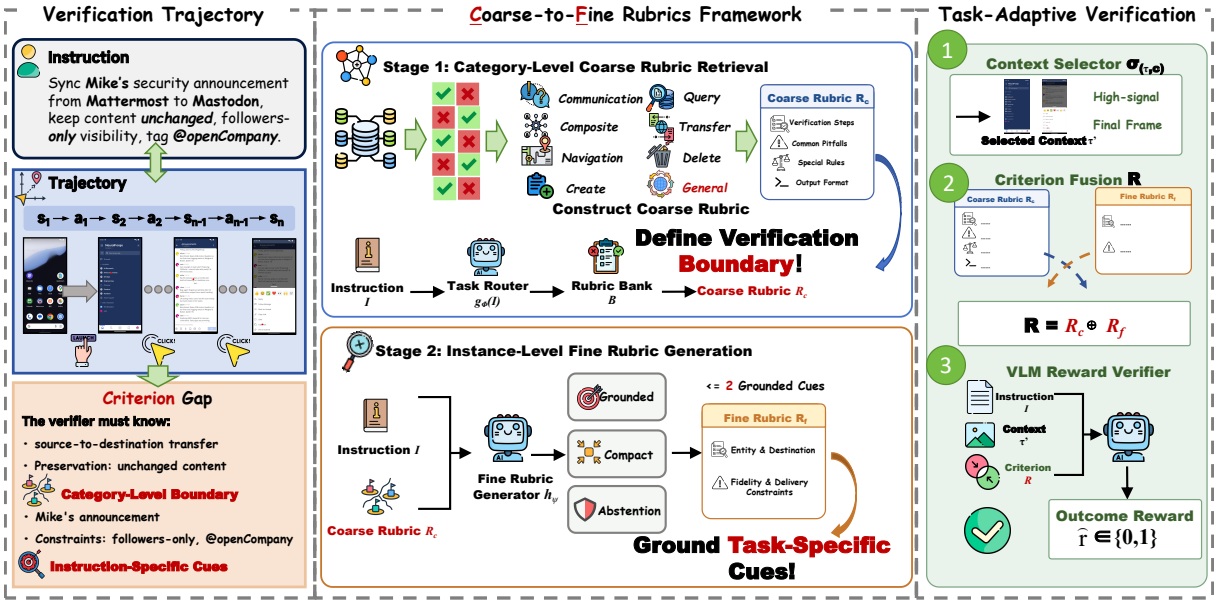



Figure 2: Overview of ADAPTRUBRIC. The left panel motivates task-adaptive verification with a concrete trajectory example and the resulting criterion gap. The middle panel illustrates **coarse-to-fine** rubric construction, where ADAPTRUBRIC retrieves a category-level coarse rubric from the rubric bank and augments it with compact instruction-specific cues. The right panel shows the final verification stage, where the fused criterion and selected trajectory context are passed to a VLM verifier for outcome reward prediction.

ing GUI reward verifiers construct or obtain these criteria in two common ways. Static-template methods (Yang et al., 2025; Qi et al., 2025; Lai et al., 2025; Wang et al., 2025a; Pan et al., 2024) encode the criteria in a fixed rubric structure, which makes verification stable but keeps the rubric generic. Such a rubric may ask whether the task is completed, but it does not adapt its checks to the current instruction's target object, required value, operation scope, or output format. Other methods (Li et al., 2026; Dai et al., 2026; Cui et al., 2026) leave the criteria to the model's implicit reasoning during verification. This gives the verifier more flexibility, but without clear verification boundaries for different task categories, the model may overlook required instruction details or introduce constraints beyond the intended scope of the task.

These limitations suggest that GUI reward verification requires task-adaptive criterion construction. The criteria should preserve verification boundaries for different task categories while adapting to the concrete requirements of each instruction instance. This allows the verifier to assess the trajectory against the task's actual requirements, rather than a generic template or a loosely defined set of implicit checks.

We propose ADAPTRUBRIC, a framework for task-adaptive criterion construction in GUI outcome reward modeling. ADAPTRUBRIC constructs explicit judging criteria through a category-level coarse stage and an instance-level fine stage. The coarse stage first routes each instruction to a GUI task category and retrieves a reusable rubric from a category-level rubric bank, where the rubric specifies verification steps, common pitfalls, and special rules for that category. The fine stage then uses the current instruction and the retrieved rubric to generate compact instance-level rubric cues, turning concrete requirements into task-specific checking items. Finally, ADAPTRUBRIC fuses both components into a single criterion supplied to the VLM verifier, so the reward model preserves category-level verification boundaries while adapting to the current instruction.

We evaluate ADAPTRUBRIC on both offline reward discrimination and online reinforcement learning. On the offline GUI reward benchmark, ADAPTRUBRIC achieves 86.7% accuracy and 86.6 F1, improving F1 by 3.6 points over the baseline average under a matched image budget. In online reinforcement learning experiments, using ADAPTRUBRIC as the reward verifier yields a 4.23-point absolute gain in task success rate and achieves the best result among compared reward agents. These results show that task-adaptive criterion construction improves trajectory-level reward judgment and

transfers to downstream GUI agent optimization. Our contributions are summarized as follows:

- We formulate task-adaptive criterion construction for GUI outcome reward modeling, where a verifier must derive explicit judging criteria from the user instruction before assigning reward.
- We introduce ADAPTRUBRIC, a coarse-to-fine rubric framework that combines category-level rubrics with compact instance-level rubrics into a single task-adaptive criterion for VLM-based reward verification.
- We evaluate ADAPTRUBRIC on offline reward discrimination and online reinforcement learning, achieving 86.7% accuracy and 86.6 F1 on the offline benchmark and a 4.23-point absolute gain in online task success rate.

## 2 Related Work

**GUI Agents for Task Automation.** The rapid progress of multimodal large language models (Singh et al., 2026; Anthropic, 2025; Team, 2026) has fundamentally transformed the landscape of autonomous GUI agents. Early GUI agents commonly relied on structured interface representations, such as DOM/HTML trees (Gur et al., 2018; Deng et al., 2023) for web pages and accessibility metadata for web or mobile interfaces (Li et al., 2024). With the emergence of large multimodal language models, later work increasingly shifted toward screenshot-based observations, often augmented with Set-of-Mark-style (Yang et al., 2023) visual annotations to expose clickable regions for visual grounding. To push their capability further, recent work post-trains these agents with supervised fine-tuning (Liu et al., 2025b; Hong et al., 2024) and offline reinforcement learning (Liu et al., 2025b, 2024). However, offline data limits how much an agent can explore beyond the trajectories it has already seen. Online reinforcement learning lets the agent interact with real environments, collect more diverse trajectories, and improve from a reward signal. To provide scalable training data and a reliable reward for both settings, outcome reward modeling is gaining increasing attention.

**Outcome Reward Modeling for GUI Agents.** Reliable outcome reward modeling is critical for GUI agents. A common approach is to use programmatic or rule-based evaluators (Rawles et al., 2025; Xie et al., 2024; Chen et al., 2025a) when the task can be deterministically checked, as in environments with executable assertions or handcrafted reward functions. Such evaluators are precise but costly to design and hard to scale to open-ended GUI tasks. Recent work therefore turns to model-based evaluators that estimate task success from screenshots, action histories, final states, or UI metadata. One line of work directly applies LLM-as-a-judge, feeding the trajectory and the instruction into a single model to obtain a binary or graded reward (Yang et al., 2025; Qi et al., 2025; Lai et al., 2025; Wang et al., 2025a; Pan et al., 2024). Another line improves verification reliability by decomposing the trajectory into milestones, adding verification steps, or introducing process-level rewards (Li et al., 2026; Zheng et al., 2026; Dai et al., 2026; Cui et al., 2026). These methods mainly change how evidence is collected, decomposed, or aggregated during verification. We focus on a complementary issue: before the verifier evaluates any evidence, it needs a task-adaptive judging criterion derived from the instruction. ADAPTRUBRIC addresses this by combining category-level rubrics with instance-level rubrics into an explicit criterion for GUI outcome reward verification.

## 3 Preliminary

A GUI agent interacts with an environment through a sequence of visual states and actions. Given a user instruction $\mathcal{I}$ and an initial screenshot $s_1$, the agent executes an action $a_t$ at each step and receives the next screenshot $s_{t+1}$ until termination. We denote the completed trajectory as

$$\tau = (s_1, a_1, s_2, a_2, \ldots, s_T, a_T, s_{T+1}), \quad (1)$$

where $s_t \in \mathcal{S}$ is a GUI screenshot, $a_t \in \mathcal{A}$ is the executed action, and $s_{T+1}$ is the terminal screenshot after the final action.

An outcome reward model (ORM) maps the instruction and trajectory to a binary success judgment,

$$\hat{r} = \mathcal{M}(\mathcal{I}, \tau) \in \{0, 1\}. \quad (2)$$

Task-adaptive outcome reward modeling requires an explicit judging criterion that specifies the success conditions for the current instruction. We denote this criterion by $\mathcal{R}$ and write

$$\hat{r} = f_\theta(\mathcal{I}, \tau', \mathcal{R}) \in \{0, 1\}, \quad (3)$$

where $f_\theta$ is the verifier, $\tau' \subseteq \tau$ is the selected trajectory context, and $\mathcal{R}$ specifies the success cri-

teria used to judge the trajectory. The goal of task-adaptive criterion construction is to derive $\mathcal{R}$ for the current instruction before assigning reward.

## 4 Method

Figure 2 gives a compact overview of ADAPTRUBRIC. As shown in the figure, Section 4.1 introduces category-level coarse rubric retrieval, Section 4.2 describes instance-level fine rubric generation, and Section 4.3 presents criterion fusion and reward verification.

### 4.1 Category-Level Coarse Rubric Retrieval

We construct the category-level rubric bank once before evaluation through an LLM-assisted induction procedure. Starting from a development trajectory pool $\mathcal{D}_{\text{dev}}$ collected from existing GUI agent benchmarks, we sample successful and failed trajectories and ask an LLM to summarize the verification dimensions that separate successful completions from failures. We then group these dimensions into broad GUI task categories. Each category rubric is refined on held-out trajectories, and whenever the rubric judgment disagrees with the trajectory label, we ask the LLM to revise it.

The resulting taxonomy $\mathcal{C}$ contains $K{=}8$ categories, namely `info_query`, `create_modify`, `delete_cleanup`, `communication`, `transfer`, `state_navigation`, `composite_workflow`, and `general`. The rubric bank is a fixed collection of category-level entries,

$$\mathcal{B} = \{E_c = (m_c, S_c)\}_{c \in \mathcal{C}}. \tag{4}$$

Each entry $E_c$ contains metadata $m_c$ and a set of natural-language rubric sections $S_c$. The metadata records the category label and prompt-assembly information, while the rubric sections specify verification steps, common pitfalls, special rules, and output format:

$$S_c = \{S_c^{\text{step}}, S_c^{\text{pitfall}}, S_c^{\text{rule}}, S_c^{\text{format}}\}, \tag{5}$$

During verification, ADAPTRUBRIC renders $S_c$ into the category-level coarse rubric $R_c$. The taxonomy and rubric dimensions are summarized in Appendix A. At inference time, a task router predicts the GUI task category from the instruction,

$$c = g_\phi(\mathcal{I}), \qquad c \in \mathcal{C}. \tag{6}$$

ADAPTRUBRIC then retrieves and renders the corresponding bank entry,

$$\begin{gathered} E_c = \text{Lookup}(\mathcal{B}, c), \quad S_c = \text{Sections}(E_c), \\ R_c = \text{Render}(S_c). \end{gathered} \tag{7}$$

We use top-1 lookup and if the router does not identify a specialized category, ADAPTRUBRIC falls back to the general entry $E_{\texttt{general}}$.

### 4.2 Instance-Level Fine Rubric Generation

The fine stage constructs an instance-level fine rubric for the current instruction. Given the instruction $\mathcal{I}$, the predicted category $c$, and the retrieved coarse rubric $R_c$, a rubric generator produces

$$R_f = h_\psi(\mathcal{I}, c, R_c), \qquad |R_f| \le 2, \tag{8}$$

where $R_f$ is a compact list of fine-grained rubric items. The generator receives the coarse rubric as context, but it is not asked to rewrite $R_c$ or fill its sections. Instead, it generates only additional instance-level checks that are grounded in the current instruction. The generator follows three constraints:

- **Instruction grounding.** Each fine rubric item must be supported by an explicit phrase, value, or constraint in the user instruction.
- **Compactness.** $R_f$ contains at most two items, so the fine rubric highlights only the most useful instance-level requirements rather than becoming another full checklist.
- **Abstention.** The generator may return $R_f = \varnothing$ when no reliable fine rubric item is needed.

These constraints keep the fine rubric compact and reduce the chance of adding requirements that are not specified by the instruction.

### 4.3 Criterion Fusion and Reward Verification

After obtaining the category-level coarse rubric and the instance-level fine rubric, ADAPTRUBRIC assembles the final judging criterion as

$$\mathcal{R} = \Phi(R_c, R_f) = R_c \oplus R_f. \tag{9}$$

The operator $\oplus$ keeps $R_c$ as the main body of the criterion and appends $R_f$ as a separated task-specific block. If the fine stage abstains, the final criterion reduces to the coarse rubric, $\Phi(R_c, \varnothing) = R_c$. The fused criterion therefore preserves the category-level verification boundary while adding the instruction-specific fine rubric when available.

Before verification, ADAPTRUBRIC selects a compact trajectory context from the full GUI trajectory,

$$\tau' = \sigma(\tau, c), \qquad |\tau'| \le M. \tag{10}$$

| Model | Ubuntu | | Mobile | | Windows | | macOS | | Web | | Overall | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | Acc | F1 | Acc | F1 | Acc | F1 | Acc | F1 | Acc | Prec | Rec | F1 |
| **ZeroGUI** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 83.8 | 84.0 | 74.5 | 76.2 | 80.3 | 75.6 | 90.9 | 78.8 | 81.1 | 83.2 | 82.0 | 83.2 | 80.0 | 81.6 |
| Qwen3-VL-8B-Instruct | 84.6 | 85.0 | 83.0 | 84.5 | 79.3 | 74.4 | 94.8 | 88.2 | 78.4 | 80.6 | 83.3 | 84.1 | 81.9 | 83.0 |
| Qwen3-VL-32B-Instruct | 84.6 | 85.2 | 83.5 | 85.0 | 80.3 | 75.6 | 96.1 | 90.3 | 82.1 | 83.5 | 84.1 | 84.6 | 83.1 | 83.9 |
| Qwen3-VL-235B-A22B-Instruct | 86.9 | 87.3 | 84.0 | 85.8 | 84.5 | 80.9 | 96.1 | 90.3 | 87.4 | 88.6 | 86.7 | 87.2 | 85.9 | 86.5 |
| Qwen3.5-122B-A10B | 85.8 | 86.5 | 82.4 | 83.7 | 83.6 | 80.2 | 96.1 | 90.9 | 80.0 | 82.7 | 84.8 | 84.1 | 85.6 | 84.8 |
| Qwen3.6-27B | 86.2 | 87.5 | 80.9 | 83.5 | 83.6 | 81.5 | 94.8 | 88.9 | 82.1 | 84.7 | 85.0 | 81.2 | 90.9 | 85.8 |
| Gemini 3 Flash | 88.5 | 89.0 | 80.3 | 80.6 | 87.8 | 85.7 | **97.4** | 93.8 | 87.9 | 88.3 | 87.7 | 89.2 | 85.7 | 87.4 |
| Gemini 3.1 Flash-Lite | 87.2 | 87.9 | 80.3 | 81.6 | 85.4 | 82.3 | 94.8 | 87.5 | 85.3 | 86.8 | 86.2 | 85.9 | 86.3 | 86.1 |
| *Mean* | 86.0 | 86.5 | 81.1 | 82.6 | 83.1 | 79.5 | 95.1 | 88.6 | 83.0 | 84.8 | 85.0 | 84.9 | 84.9 | 84.9 |
| **OS-Themis** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 72.6 | 71.4 | 79.3 | 78.9 | 75.1 | 68.3 | 84.4 | 57.1 | 80.0 | 81.9 | 75.5 | 79.5 | 68.3 | 73.5 |
| Qwen3-VL-8B-Instruct | 76.2 | 75.0 | 84.0 | 83.7 | 78.4 | 72.0 | 83.1 | 51.9 | 81.6 | 82.4 | 78.7 | 84.6 | 69.9 | 76.5 |
| Qwen3-VL-32B-Instruct | 77.1 | 74.6 | 81.9 | 80.7 | 76.5 | 65.3 | 90.9 | 77.4 | 83.7 | 81.9 | 79.3 | 91.6 | 64.1 | 75.5 |
| Qwen3-VL-235B-A22B-Instruct | 86.4 | 86.8 | **93.6** | 93.7 | 77.5 | 69.6 | 93.5 | 82.8 | 91.6 | 91.9 | 87.1 | 90.5 | 82.7 | 86.4 |
| Qwen3.5-122B-A10B | 85.8 | 85.8 | 88.3 | 88.2 | 79.8 | 73.0 | 88.3 | 66.7 | 79.0 | 75.6 | 84.5 | 92.0 | 75.3 | 82.8 |
| Qwen3.6-27B | 87.9 | 88.5 | **93.6** | **93.9** | **88.3** | **86.9** | 96.1 | 90.3 | 92.1 | 92.1 | 89.7 | 90.2 | 89.0 | 89.6 |
| Gemini 3 Flash | 81.5 | 81.5 | 84.0 | 84.2 | 86.9 | 84.4 | 88.3 | 71.0 | 80.5 | 77.3 | 82.9 | 88.1 | 75.9 | 81.5 |
| Gemini 3.1 Flash-Lite | 82.9 | 83.3 | 82.4 | 82.7 | 87.3 | 84.2 | 90.9 | 75.9 | 84.2 | 83.3 | 84.1 | 87.7 | 79.1 | 83.2 |
| *Mean* | 81.3 | 80.9 | 85.9 | 85.8 | 81.2 | 75.5 | 89.4 | 71.6 | 84.1 | 83.3 | 82.7 | 88.0 | 75.5 | 81.1 |
| **AdaptRubric** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 84.3 | 85.6 | 80.9 | 81.1 | 83.1 | 79.5 | **97.4** | **94.1** | 82.1 | 84.5 | 84.1 | 82.8 | 85.9 | 84.3 |
| Qwen3-VL-8B-Instruct | 83.9 | 85.1 | 83.5 | 84.6 | 81.7 | 79.1 | **97.4** | **94.1** | 81.6 | 83.3 | 84.0 | 82.6 | 85.9 | 84.2 |
| Qwen3-VL-32B-Instruct | 86.5 | 86.6 | 85.6 | 85.9 | 80.3 | 75.0 | 93.5 | 83.9 | 86.8 | 87.6 | 85.9 | 89.3 | 81.3 | 85.1 |
| Qwen3-VL-235B-A22B-Instruct | 86.1 | 86.7 | 87.8 | 88.9 | 84.5 | 81.4 | 94.8 | 87.5 | 88.9 | 90.0 | 86.9 | 87.0 | 86.7 | 86.8 |
| Qwen3.5-122B-A10B | 86.2 | 86.8 | 89.9 | 90.4 | 81.7 | 77.2 | 94.8 | 88.2 | 88.9 | 89.8 | 86.9 | 87.8 | 85.4 | 86.6 |
| Qwen3.6-27B | 87.3 | 88.5 | 91.0 | 91.6 | 84.5 | 82.5 | 93.5 | 85.7 | 91.6 | 92.4 | 88.3 | 85.4 | **92.1** | 88.7 |
| Gemini 3 Flash | **90.4** | **90.7** | 91.0 | 91.4 | 87.3 | 84.7 | 94.8 | 88.9 | **94.7** | **95.0** | **90.8** | **92.3** | 89.0 | **90.6** |
| Gemini 3.1 Flash-Lite | 86.4 | 87.1 | 83.0 | 83.3 | 85.0 | 81.6 | 94.8 | 87.5 | 88.9 | 89.7 | 86.5 | 87.2 | 85.4 | 86.3 |
| *Mean* | 86.4 | 87.1 | 86.6 | 87.2 | 83.5 | 80.1 | 95.1 | 88.7 | 87.9 | 89.0 | 86.7 | 86.8 | 86.5 | 86.6 |

Table 1: Offline reward discrimination results on OGRBench. Each method reports Accuracy (Acc) and F1-score (F1) for each platform; Overall columns include Acc, Precision (Prec), Recall (Rec), and F1. ZeroGUI uses the last-10 image budget. In each column, **bold** marks the best value across all $3\times8$ method-backbone cells, and underline marks the second-best (computed separately for the *Mean* rows).

The selector keeps screenshot-bearing steps that are useful for outcome judgment, including the initial and final states and frames around high-signal actions such as text entry, submission, or state-changing operations. The selected context, the instruction, and the fused criterion are then passed to the verifier,

$$\hat{r} = f_\theta(\mathcal{I}, \tau', \mathcal{R}) \in \{0, 1\}. \tag{11}$$

The verifier outputs a binary reward in a parsable format. Thus, AdaptRubric changes the judging criterion and trajectory context supplied to the verifier, while keeping the verifier architecture unchanged.

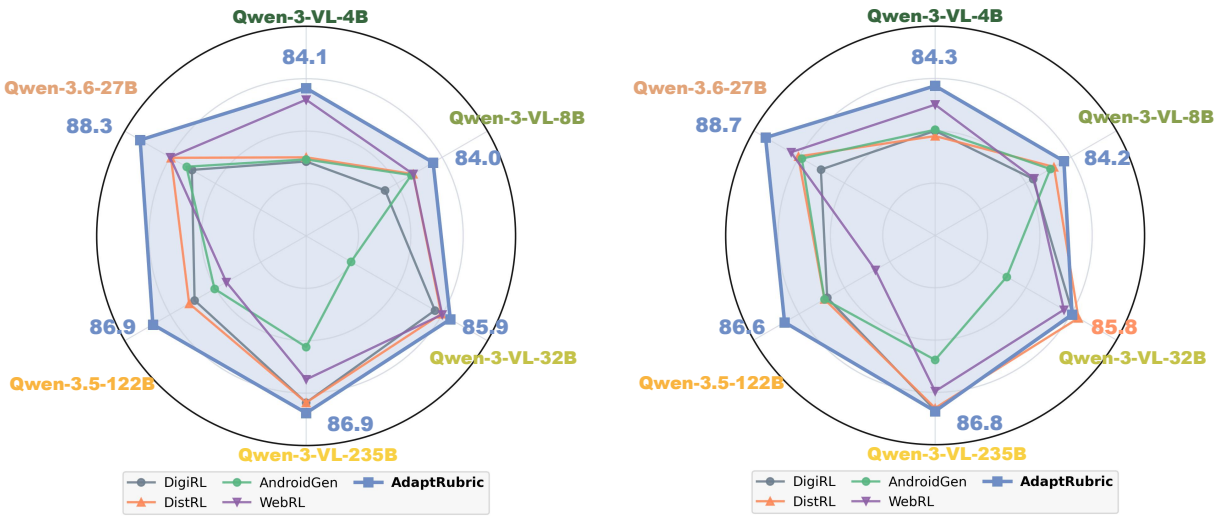
# 5 Experiment

## 5.1 Offline Evaluation

We evaluate offline GUI reward discrimination on OmniGUIRewardBench (OGRBench) (Li et al., 2026), a cross-platform benchmark for testing whether a reward verifier can correctly judge trajectory-level task success.

**Benchmark.** OGRBench contains 1,409 trajectories from five environments: OSWorld (Xie et al., 2024) for Ubuntu, AndroidWorld (Rawles et al., 2025) for mobile Android, (Bonatti et al., 2024) for Windows, macOSArena (Wang et al., 2025b) for macOS, and WebArena-Lite-v2 (Wang et al., 2025b; Zhou et al., 2023) for web tasks. The benchmark is nearly balanced overall, with 700 positive and 709 negative trajectories.

**Baselines.** We compare with six representative




(a) Overall Accuracy.

(b) Overall F1.

Figure 3: Cross-backbone radar comparison on OGRBench under the matched ten-screenshot budget. Each axis corresponds to one Qwen-family judge backbone, and each curve corresponds to one reward verifier.

GUI reward verification frameworks: DigiRL (Bai et al., 2024), DistRL (Wang et al., 2025a), AndroidGen (Lai et al., 2025), WebRL (Qi et al., 2025), ZeroGUI (Yang et al., 2025), and OS-Themis (Li et al., 2026). To align their trajectory inputs with AdaptRubric, all offline reward verifiers are evaluated under the same ten-screenshot budget. Appendix E details each baseline's original trajectory-context setting and our unified matched-budget instantiation, and Appendix D analyzes the effect of image budget.

For judge backbones, we evaluate open-source Qwen models from the Qwen3-VL (Bai et al., 2025), Qwen3.5 (Team, 2026), and Qwen3.6 (Qwen Team, 2026) series, as well as closed-source Gemini models (Team et al., 2025).

**Metrics.** For offline reward discrimination, we report Accuracy, Precision, Recall, and F1.

**Main Results.** Table 1 reports the main OGRBench comparison with ZeroGUI and OS-Themis across all eight judge backbones. In this setting, AdaptRubric achieves the **best average** performance, with 86.7% accuracy and 86.6 F1. Compared with the average of the two baselines under the main protocol, AdaptRubric improves the four overall metrics by 3.3 points on average, including 2.9 points in accuracy and 3.6 points in F1. The main gain comes from recall: the baseline average is 80.2%, while AdaptRubric raises recall to 86.5% with a comparable precision of 86.8%. This indicates that **task-adaptive criteria** help the verifier recover more successful trajectories while still controlling false positives. The improvement is also **consistent across platforms**, where AdaptRubric achieves the best average F1 on Ubuntu, Android, Windows, macOS, and Web tasks. These results support our core claim that combining category-level rubrics with instance-level fine rubrics yields more reliable trajectory-level reward judgment than generic terminal-state judging or implicit multi-step verification.

Figure 3 further compares AdaptRubric with four additional baselines, DigiRL, DistRL, AndroidGen, and WebRL, using the open-source Qwen-family judge backbones. This expanded comparison tests whether the same advantage holds against a broader set of reward-agent prompts under the matched ten-screenshot budget. AdaptRubric achieves the best accuracy on all backbones and the best F1 on five of six backbones, showing that the gain is not tied to a single judge scale. The detailed values in Appendix E show that the expanded-baseline mean F1 of AdaptRubric remains higher than all four additional baselines.

### 5.2 Online Reinforcement Learning

We further evaluate whether the offline reward advantage transfers to downstream GUI agent optimization.

**Setup.** We conduct online RL training using the ClawGUI (Tang et al., 2026) framework on the MobileWorld (Kong et al., 2025) environment. The policy backbone is MAI-UI-8B (Zhou et al., 2025),

and the policy is optimized with Group Relative Policy Optimization (GRPO) (Shao et al., 2024). For each task, GRPO samples four rollouts as a group, and the reward agent assigns an outcome reward to each completed trajectory. We keep the training protocol fixed and replace only the reward agent, comparing DigiRL, ZeroGUI, OS-Themis, and ADAPTRUBRIC. For ADAPTRUBRIC, the reward verifier uses Qwen3-VL-8B-Instruct and observes at most ten trajectory screenshots. All runs use a maximum episode length of 50 steps and train for 2 epochs; other hyperparameters are reported in Appendix C.

**Main Results.** Table 2 reports online reinforcement learning results when different reward agents provide training rewards for MAI-UI-8B. Without an external reward agent, MAI-UI-8B reaches 19.70% task success. Using ADAPTRUBRIC as the reward verifier increases success to 23.93%, a **4.23-point** absolute gain and the best result among compared reward agents. This suggests that the criterion constructed by ADAPTRUBRIC is not only better at offline trajectory discrimination, but also provides a more useful reward signal for policy improvement.

### 5.3 Reward-Guided Test-Time Scaling

We test whether reward verification helps select successful trajectories when multiple candidates are sampled for the same task.

**Setup.** We evaluate on 113 AndroidWorld tasks, with ten candidate trajectories sampled for each task. We use this offline candidate pool to simulate online test-time scaling, so that all verifiers select from the same trajectories and the dynamic execution environment remains controlled. To create a discriminative candidate pool, we mix eight rollouts from Qwen3-VL-8B-Instruct and two rollouts from Qwen3-VL-235B-A22B-Instruct. Details are provided in Appendix F. All reward verifiers share Qwen3-VL-32B-Instruct as the judge and consume the same per-task shuffled trajectory stream. We evaluate two selection protocols: **EarlyStop@N** scans the first $N$ trajectories sequentially and stops at the first one predicted as successful (online, low-latency setting), while **BestOfN@N** scores all $N$ candidates and returns the highest-scored one (offline, batch setting). All verifiers in this comparison output binary scores, so multiple candidates often share the same score under BestOfN. In this case, we always pick the last tied candidate as the final selection. Both gains are reported relative to Random selection.

| Backbone | Reward Agent | SR (%) | Δ |
|---|---|---|---|
| MAI-UI-8B | – | 19.70† | – |
| | DigiRL | 23.08 | +3.38 |
| | ZeroGUI | 22.22 | +2.52 |
| | OS-Themis | 22.22 | +2.52 |
| | **ADAPTRUBRIC** | **23.93 ± 0.85** | **+4.23** |

Table 2: Online reinforcement training results on MAI-UI-8B. We report the success rate (SR) after training with different reward agents. † denotes the result from Tang et al. (2026).

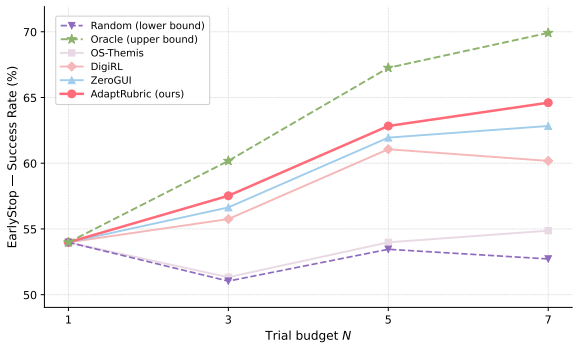
| Method | ES@7 | BoN@8 | Acc. | F1 | FPR |
|---|---|---|---|---|---|
| OS-Themis | +2.15 | +7.97 | 75.75 | 72.92 | **10.81** |
| DigiRL | +7.46 | +7.97 | 80.80 | 81.90 | 22.71 |
| ZeroGUI | +10.11 | +12.39 | 86.55 | 87.16 | 15.38 |
| ADAPTRUBRIC | **+11.88** | **+13.28** | **88.14** | **88.41** | 11.17 |

Table 3: Heterogeneous-pool offline reward-guided test-time scaling. ES@7 (EarlyStop) and BoN@8 (BestOfN) report success-rate gains over Random in percentage points. Acc., F1, and FPR (false-positive rate) are computed over trajectory-level reward predictions.

**Overall results.** Table 3 summarizes performance under the two selection protocols. ADAPTRUBRIC achieves the largest gains over Random, improving EarlyStop@7 by $+11.88$ points and BestOfN@8 by $+13.28$ points. It also obtains the strongest trajectory-level reward prediction results, with **88.14** accuracy and **88.41** F1. At the same time, its false-positive rate remains low at 11.17, close to the most conservative baseline OS-Themis.

**EarlyStop: low-latency selection.** Figure 4a reports EarlyStop SR@$N$ as the per-task budget grows from 1 to 7. ADAPTRUBRIC is the strongest method at every budget from $N{=}3$ onward. At $N{=}7$, it reaches $64.60\%$ task success, exceeding ZeroGUI by 1.77 points, DigiRL by 4.42 points, and OS-Themis by 9.73 points. The Oracle upper bound at the same budget is $69.91\%$, so ADAPTRUBRIC closes about $69\%$ of the gap between Random selection and the Oracle. Because EarlyStop commits to the first accepted trajectory, it favors verifiers that can avoid false-positive acceptances.

**BestOfN: batch selection.** Figure 4b reports BestOfN SR@$N$ for $N \in \{1, 2, 4, 6, 8\}$. ADAPTRUBRIC achieves the best result at $N{=}4$ with $64.60\%$ task success and at $N{=}8$ with $65.49\%$ task success, and it ties ZeroGUI at $N{=}6$. The only




(a) EarlyStop SR@$N$ ($N$=1,3,5,7).


(b) BestOfN SR@$N$ ($N$=1,2,4,6,8).

Figure 4: Test-time scaling results on the heterogeneous AndroidWorld pool of 113 tasks.

| Coarse | Fine | Setting | Acc. | ΔAcc. | F1 | ΔF1 |
|---|---|---|---|---|---|---|
| ✓ | ✓ | **Full (Coarse + Fine)** | **86.9** | — | **86.6** | — |
| ✓ | × | *w/o* Fine | 84.9 | −2.0 | 84.6 | −2.0 |
| × | ✓ | *w/o* Coarse | 85.2 | −1.7 | 84.6 | −2.0 |
| × | × | *w/o* Both (image-only) | 81.5 | −5.4 | 80.7 | −5.9 |

Table 4: **Ablation study of rubric construction on Qwen3.5-122B-A10B.** ΔAcc. and ΔF1 report the absolute drop relative to the full model.

| Method | Quality | | LLM Cost | | | | Runtime | |
|---|---|---|---|---|---|---|---|---|
| | Acc. | F1 | Calls | Calls/traj. | Tokens | Tok./traj. | Min. | s/traj. |
| OS-Themis | 78.7 | 76.5 | 21,490 | 15.25 | 243.75M | 173.0K | 88.8 | 3.78 |
| ZeroGUI | 83.3 | 83.0 | 5,608 | 3.98 | 100.15M | 71.1K | 51.7 | 2.20 |
| DigiRL | 78.7 | 80.8 | **1,402** | **1.00** | **23.46M** | **16.6K** | **34.2** | **1.46** |
| ADAPTRUBRIC | **84.0** | **84.2** | 4,220 | 3.00 | 26.98M | 19.1K | 36.6 | 1.56 |

Table 5: **Efficiency comparison on OGRBench using Qwen3-VL-8B-Instruct.** All methods use the same ten-screenshot trajectory budget. Lowest cost in each column is in bold.

weaker point is $N$=2, where binary reward outputs often produce ties and the fixed tie-breaking rule has a large effect on the selected trajectory. DigiRL decreases as the candidate set grows from $N$=4 to $N$=6, consistent with the difficulty of judging longer visual contexts. Overall, the batch setting amplifies the advantage of ADAPTRUBRIC, because reward errors accumulate when the verifier must compare many candidates.

### 5.4 Ablation Study

Table 4 isolates the contribution of the Coarse and Fine rubrics in ADAPTRUBRIC. The full model achieves **86.9** accuracy and **86.6** F1 on Qwen3.5-122B-A10B. Removing either component consistently weakens reward verification. Without the Fine rubric, F1 drops by **2.0** points, and without the Coarse rubric, F1 drops by **2.0** points as well. The largest decline appears when both rubrics are removed, where the verifier relies only on the instruction and trajectory screenshots and F1 falls by **5.9** points. This pattern shows that the two rubrics are complementary, since the Coarse rubric anchors the verifier to task-category boundaries and the Fine rubric adds instruction-specific success requirements.

### 5.5 Efficiency Analysis

Table 5 compares OGRBench quality and inference cost using Qwen3-VL-8B-Instruct. ADAPTRUBRIC achieves the best accuracy and F1, with **84.0** Acc. and **84.2** F1, while using substantially fewer calls, tokens, and runtime than OS-Themis and ZeroGUI. DigiRL is slightly cheaper, but its F1 remains **3.4** points lower than ADAPTRUBRIC. Overall, ADAPTRUBRIC provides the strongest quality–cost trade-off for repeated reward verification.

### 5.6 Case Study

Appendix G presents a representative failure case showing why task-adaptive criteria matter. The task asks the agent to add text to the top of a note, but the trajectory inserts the text below existing content. Baselines focus on whether the requested text appears and incorrectly mark the task as successful. ADAPTRUBRIC combines a coarse create/-modify rubric with fine placement cues, allowing the verifier to check both the edited item and the instruction-specific ordering constraint.

## 6 Conclusion

We presented ADAPTRUBRIC, a coarse-to-fine rubric framework for task-adaptive GUI outcome

reward modeling. The category-level Coarse rubric anchors verification to task-family success criteria, while the instance-level Fine rubric specifies the task-adaptive criteria needed for the current instruction. Across offline and online experiments, AdaptRubric consistently demonstrates the value of explicitly constructing task-adaptive criteria for success, supporting our core claim for reliable GUI reward modeling across GUI interaction settings.

## Limitations

AdaptRubric improves GUI reward verification by constructing task-adaptive rubrics, but it still has several scope boundaries. First, although our evaluation covers mobile, desktop, and web environments, it does not exhaust all possible applications, interface designs, or user instruction styles. Second, the coarse rubric bank is constructed offline and kept fixed during evaluation to ensure reproducibility and controlled comparison across methods; this makes the protocol stable, but it may not capture every newly emerging application or domain-specific workflow.

## References


Anthropic. 2025. Introducing claude 4. https://www.anthropic.com/news/claude-4. Accessed: 2026-05-25.

Hao Bai, Yifei Zhou, Mert Cemri, Jiayi Pan, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Preprint*, arXiv:2406.11896.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

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

Yuhan Chen, Yuxuan Liu, Long Zhang, Pengzhi Gao, Jian Luan, and Wei Liu. 2025a. Step: Success-rate-aware trajectory-efficient policy optimization. *Preprint*, arXiv:2511.13091.

Zhenfang Chen, Delin Chen, Rui Sun, Wenjun Liu, and Chuang Gan. 2025b. Scaling autonomous agents via automatic reward modeling and planning. *Preprint*, arXiv:2502.12130.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. 2026. Agentic reward modeling: Verifying gui agent via online proactive interaction. *Preprint*, arXiv:2602.00575.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuqing Yang, Yuanchun Li, Rui Tan, Mo Li, and Lili Qiu. 2026. Prore: A proactive reward system for gui agents via reasoner-actor collaboration. *Preprint*, arXiv:2509.21823.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Preprint*, arXiv:2306.06070.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *Preprint*, arXiv:2507.17746.

Taneesh Gupta, Shivam Shandilya, Xuchao Zhang, Rahul Madhavan, Supriyo Ghosh, Chetan Bansal, Huaxiu Yao, and Saravan Rajmohan. 2025. Carmo: Dynamic criteria generation for context-aware reward modelling. *Preprint*, arXiv:2410.21545.

Izzeddin Gur, Ulrich Rueckert, Aleksandra Faust, and Dilek Hakkani-Tur. 2018. Learning to navigate the web. *Preprint*, arXiv:1812.09195.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchen Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tieyong Zeng, Liang Wang, and 10 others. 2025. Os agents: A survey on mllm-based agents for general computing devices use. *Preprint*, arXiv:2508.04482.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. 2025. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments. *Preprint*, arXiv:2512.19432.

Hanyu Lai, Junjie Gao, Xiao Liu, Yifan Xu, Shudan Zhang, Yuxiao Dong, and Jie Tang. 2025. Androidgen: Building an android language agent under data scarcity. *Preprint*, arXiv:2504.19298.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on ui control agents. *Preprint*, arXiv:2406.03679.

Zehao Li, Zhenyu Wu, Yibo Zhao, Bowen Yang, Jingjing Xie, Zhaoyang Liu, Zhoumianze Liu, Kaiming Jin, Jianze Liang, Zonglin Li, Feng Wu, Bowen Zhou, Zun Wang, and Zichen Ding. 2026. Os-themis: A scalable critic framework for generalist gui rewards. *Preprint*, arXiv:2603.19191.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, and 11 others. 2024. Autoglm: Autonomous foundation agents for guis. *Preprint*, arXiv:2411.00820.

Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. 2025a. Infiguiagent: A multimodal generalist gui agent with native reasoning and reflection. *Preprint*, arXiv:2501.04575.

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

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. *Preprint*, arXiv:2404.06474.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. 2025. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *Preprint*, arXiv:2411.02337.

Qwen Team. 2026. Qwen3.6-27B: Flagship-level coding in a 27B dense model.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Fei Tang, Zhiqiong Lu, Boxuan Zhang, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026. Clawgui: A unified framework for training, evaluating, and deploying gui agents. *Preprint*, arXiv:2604.11784.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, and 1332 others. 2025. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Qwen Team. 2026. Qwen3.5: Accelerating productivity with native multimodal agents.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, and 1 others. 2024. Gui agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*.

Taiyi Wang, Zhihao Wu, Jianheng Liu, Jianye Hao, Jun Wang, and Kun Shao. 2025a. Distrl: An asynchronous distributed reinforcement learning framework for on-device control agents. *Preprint*, arXiv:2410.14803.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, Weiyun Wang, Xiangyu Zhao, Jixuan Chen, Haodong Duan, Tianbao Xie, Shiqian Su, Chenyu Yang, Yue Yu, Yuan Huang, and 8 others. 2025b. Mmbench-gui: Hierarchical multi-platform evaluation framework for gui agents. *arXiv preprint arXiv:2507.19478*.

Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2025. Agentrm: Enhancing agent generalization with reward modeling. *Preprint*, arXiv:2502.18407.

Lipeng Xie, Sen Huang, Zhuo Zhang, Anni Zou, Yunpeng Zhai, Dingchao Ren, Kezun Zhang, Haoyuan Hu, Boyin Liu, Haoran Chen, Zhaoyang Liu, and Bolin Ding. 2026. Auto-rubric: Learning from implicit weights to explicit rubrics for reward modeling. *Preprint*, arXiv:2510.17314.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu,

Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Preprint*, arXiv:2404.07972.

Tao Xiong, Xavier Hu, Yurun Chen, Yuhang Liu, Changqiao Wu, Pengzhi Gao, Wei Liu, Jian Luan, and Shengyu Zhang. 2025. Gui-pra: Process reward agent for gui tasks. *Preprint*, arXiv:2509.23263.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. 2026. Mobile-agent-v3.5: Multi-platform fundamental gui agents. *Preprint*, arXiv:2602.16855.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. 2025. Mobilerl: Online agentic reinforcement learning for mobile gui agents. *Preprint*, arXiv:2509.18119.

Chenyu Yang, Shiqian Su, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, Wenhai Wang, Yu Qiao, Xizhou Zhu, and Jifeng Dai. 2025. Zerogui: Automating online gui learning at zero human cost. *Preprint*, arXiv:2505.23762.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *Preprint*, arXiv:2310.11441.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. 2025. Instruction tuning for large language models: A survey. *Preprint*, arXiv:2308.10792.

Congmin Zheng, Xiaoyun Mo, Xinbei Ma, Qiqiang Lin, Yin Zhao, Jiachen Zhu, Xingyu Lou, Jun Wang, Zhaoxiang Wang, Weiwen Liu, Zhuosheng Zhang, Yong Yu, and Weinan Zhang. 2026. Adaptive milestone reward for gui agents. *Preprint*, arXiv:2602.11524.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, and Steven Hoi. 2025. Mai-ui technical report: Real-world centric foundation gui agents. *Preprint*, arXiv:2512.22047.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

# A Taxonomy and Rubric Bank

Table 6 lists the eight task families in our GUI taxonomy $\mathcal{C}$. For each family, we show its identifier, a brief description, and the key verification dimensions encoded in its static rubric $R_c$.

| Family | Description | Key Dimensions |
|---|---|---|
| `info_query` | Answer from screen | Format; content; evidence |
| `create_modify` | Create or edit | Explicit properties; completion; consistency |
| `delete_cleanup` | Remove or clear | Scope; target; completeness |
| `communication` | Send to recipient | Source; recipient; channel; confirmation |
| `transfer` | Move or copy | Source; action; destination; fidelity |
| `state_navigation` | Toggle or navigate | Final state; toggle; persistence |
| `composite_workflow` | Multiple goals | Per-family rubric; all subgoals succeed |
| `general` | Fallback | Task logic; action; final state |

Table 6: GUI task taxonomy used for coarse rubric retrieval.

# B Coarse Rubric Bank Construction

We construct the coarse rubric bank before evaluation and keep it fixed for all experiments. To build the development pool, we run `Qwen3.5-122B-A10B` on AndroidWorld and MobileWorld and collect complete trajectories with binary success labels from the environment evaluators. The pool contains 116 trajectories from each benchmark: AndroidWorld has 55 successful and 61 failed trajectories, while MobileWorld has 35 successful and 81 failed trajectories. MobileWorld originally contains 117 tasks in this collection, but one task fails during environment execution and therefore has no usable trajectory. We use `Claude Opus 4.6` as the offline rubric-construction model to summarize recurring success criteria and failure patterns, group them into GUI task families, and refine the category rubrics using disagreement cases. The resulting bank contains the eight entries shown in Table 6.

# C Online RL Training Details

Our online reinforcement learning experiments use ClawGUI-RL with GRPO on MobileWorld. Table 7 summarizes the main training hyperparameters. All compared reward agents use the same policy backbone and training protocol; only the reward verifier is changed.

| Setting | Acc | Prec | Rec | F1 |
|---|---|---|---|---|
| Last 2 screenshots | 79.1 | 84.9 | 70.7 | 76.8 |
| Last 10 screenshots | **85.0** | **84.9** | **84.9** | **84.9** |

Table 8: Effect of image budget on ZeroGUI. We report overall performance on OGRBench averaged across eight judge backbones.

| Item | Value |
|---|---|
| Training framework | ClawGUI-RL |
| Environment | MobileWorld |
| Policy backbone | MAI-UI-8B |
| Optimization algorithm | GRPO |
| Training epochs | 2 |
| Number of GPUs | 8 |
| Training batch size | 8 |
| Rollouts per task | 4 |
| History length | 3 |
| Maximum episode steps | 50 |
| Learning rate | $1\times10^{-6}$ |
| KL coefficient | 0.01 |
| Rollout temperature | 0.7 |
| Validation temperature | 0.4 |
| Maximum prompt length | 28,000 |
| Maximum response length | 512 |
| AdaptRubric verifier backbone | Qwen3-VL-8B-Instruct |
| Maximum screenshots for reward verification | 10 |

Table 7: Key hyperparameters for online RL training.

## D Image Budget Analysis

Trajectory context is an important input factor for GUI reward verification. The OS-Themis evaluation protocol runs ZeroGUI with the final two screenshots, while our main experiments use at most ten trajectory screenshots. To align the image budget, we also evaluate ZeroGUI with its final ten screenshots and use the ten-screenshot setting in subsequent experiments. Table 8 reports both settings. Increasing the image budget improves ZeroGUI mainly by raising recall, which makes our main comparison more conservative.

Table 9 further reports the additional baselines under their original trajectory-context settings. Most of these evaluators were designed around a final screenshot or a small final-state context, while ZeroGUI follows the two-screenshot setting used in prior evaluation. Comparing Table 9 with Table 11 shows that increasing the trajectory context substantially improves several baselines, which motivates the matched-budget setting used in the expanded comparison shown in Figure 3.

## E Additional Baselines under Matched Image Budget

This section provides the detailed numerical results behind the expanded matched-budget comparison summarized in Figure 3. Table 10 summarizes each baseline's original trajectory-context setting and our matched-budget instantiation. For a controlled comparison, we provide every baseline with the same ten-screenshot trajectory budget used by AdaptRubric and evaluate them with the same Qwen-family judge backbones used in Figure 3. Table 11 reports the full results, including ZeroGUI and OS-Themis for reference. The results show that AdaptRubric maintains the strongest mean overall F1 across the expanded baseline set, indicating that the advantage does not come from comparing only against ZeroGUI and OS-Themis.

## F Reward-Guided Test-Time Scaling Details

We evaluate reward-guided trajectory selection on AndroidWorld. The pool contains 113 tasks with complete rollout traces and ground-truth labels from the AndroidWorld evaluator. Instead of executing new rollouts separately for each verifier, we use this pre-collected trajectory pool to simulate online repeated attempts under a controlled environment. This ensures that different verifiers are compared on the same candidate trajectories rather than on different samples produced by a stochastic dynamic environment. For each task, we collect ten independent candidate trajectories: eight generated by Qwen3-VL-8B-Instruct and two generated by Qwen3-VL-235B-A22B-Instruct, all sampled with temperature 0.7.

**Why a heterogeneous pool.** Table 12 shows why we use a heterogeneous pool rather than a homogeneous single-policy pool. The 8B-only pool is strongly bimodal: $42.5\%$ of tasks fail on every trial and $38.9\%$ succeed on every trial, leaving only $18.6\%$ of tasks where any reward verifier can change the outcome. The 235B-only pool has a similar issue because it contains only two trajectories per task, yielding only $13.3\%$ mixed tasks. In these all-success or all-failure cases, the verifier choice is irrelevant and SR@$N$ differences are dominated by pool composition rather than reward quality. The heterogeneous pool raises the mixed-task fraction to $46.0\%$ and widens the Oracle-minus-Random headroom from about $+6$ points to $+20.00$ points. Relative to the 8B-only pool, it creates 31 additional mixed tasks, all caused by the added 235B trajectories changing an otherwise all-success or all-failure task into a task with both positive and negative candidates. This policy-level diversity makes the reward-guided selection setting informa-

| Model | Ubuntu | | Mobile | | Windows | | macOS | | Web | | Overall | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | Acc | F1 | Acc | F1 | Acc | F1 | Acc | F1 | Acc | Prec | Rec | F1 |
| **DigiRL** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 70.7 | 71.0 | 78.7 | 81.3 | 76.5 | 72.5 | 87.0 | 70.6 | 76.3 | 79.6 | 74.3 | 74.1 | 74.1 | 74.1 |
| Qwen3-VL-8B-Instruct | 73.3 | 73.8 | 73.4 | 76.9 | 77.9 | 73.7 | 88.3 | 71.0 | 74.2 | 77.8 | 74.9 | 74.7 | 75.0 | 74.8 |
| Qwen3-VL-32B-Instruct | 77.9 | 78.1 | 80.9 | 82.5 | 79.8 | 74.9 | 90.9 | 75.9 | 81.1 | 82.9 | 79.7 | 81.1 | 77.1 | 79.1 |
| Qwen3-VL-235B-A22B-Instruct | 77.1 | 76.5 | 78.7 | 80.4 | 78.4 | 72.3 | 90.9 | 75.9 | 81.1 | 82.5 | 78.8 | 81.9 | 73.6 | 77.5 |
| Qwen3.5-122B-A10B | 73.0 | 69.4 | 75.5 | 75.8 | 78.4 | 69.3 | 87.0 | 61.5 | 76.8 | 77.3 | 75.4 | 84.4 | 62.0 | 71.5 |
| Qwen3.6-27B | 73.1 | 71.0 | 76.6 | 78.0 | 77.0 | 69.6 | 92.2 | 78.6 | 78.4 | 79.6 | 75.9 | 81.3 | 67.0 | 73.5 |
| *Mean* | 74.2 | 73.3 | 77.3 | 79.1 | 78.0 | 72.0 | 89.4 | 72.2 | 78.0 | 80.0 | 76.5 | 79.6 | 71.5 | 75.1 |
| **DistRL** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 71.8 | 73.7 | 75.0 | 76.1 | 72.8 | 66.7 | 84.4 | 68.4 | 76.3 | 80.0 | 73.7 | 72.6 | 75.6 | 74.0 |
| Qwen3-VL-8B-Instruct | 79.1 | 80.1 | 75.0 | 77.3 | 78.9 | 74.6 | 89.6 | 75.0 | 73.2 | 77.3 | 78.3 | 77.4 | 79.6 | 78.5 |
| Qwen3-VL-32B-Instruct | 83.9 | 84.8 | 78.2 | 80.4 | 81.7 | 77.2 | 92.2 | 81.2 | 83.2 | 85.6 | 83.2 | 82.4 | 84.1 | 83.3 |
| Qwen3-VL-235B-A22B-Instruct | 80.6 | 81.0 | 81.9 | 83.7 | 79.8 | 74.3 | 90.9 | 78.8 | 80.0 | 82.4 | 81.1 | 81.8 | 79.7 | 80.8 |
| Qwen3.5-122B-A10B | 77.2 | 75.3 | 73.4 | 72.8 | 77.0 | 67.5 | 89.6 | 71.4 | 79.5 | 80.8 | 77.6 | 85.1 | 66.7 | 74.8 |
| Qwen3.6-27B | 80.4 | 80.2 | 77.7 | 77.4 | 77.9 | 69.3 | 88.3 | 71.0 | 78.4 | 79.6 | 79.8 | 84.6 | 72.7 | 78.2 |
| *Mean* | 78.8 | 79.2 | 76.9 | 78.0 | 78.0 | 71.6 | 89.2 | 74.3 | 78.4 | 81.0 | 79.0 | 80.6 | 76.4 | 78.3 |
| **AndroidGen** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 77.3 | 79.2 | 68.6 | 74.9 | 70.0 | 69.5 | 75.3 | 61.2 | 77.9 | 82.1 | 75.0 | 70.9 | 84.4 | 77.1 |
| Qwen3-VL-8B-Instruct | 76.4 | 75.5 | 75.5 | 78.3 | 70.4 | 65.2 | 85.7 | 70.3 | 79.5 | 82.8 | 76.3 | 77.2 | 74.1 | 75.7 |
| Qwen3-VL-32B-Instruct | 76.9 | 79.0 | 64.9 | 74.0 | 69.5 | 70.3 | 74.0 | 60.0 | 74.2 | 79.7 | 73.7 | 68.8 | 86.1 | 76.5 |
| Qwen3-VL-235B-A22B-Instruct | 78.1 | 79.6 | 71.8 | 77.1 | 73.2 | 67.8 | 84.4 | 71.4 | 80.5 | 82.5 | 77.2 | 75.1 | 81.0 | 77.9 |
| Qwen3.5-122B-A10B | 77.1 | 80.1 | 70.2 | 76.5 | 68.1 | 66.3 | 75.3 | 59.6 | 74.2 | 79.7 | 74.3 | 69.2 | 87.1 | 77.1 |
| Qwen3.6-27B | 82.6 | 83.0 | 77.1 | 81.2 | 74.2 | 68.2 | 85.7 | 66.7 | 75.8 | 79.5 | 79.8 | 79.1 | 80.9 | 79.9 |
| *Mean* | 78.1 | 79.4 | 71.4 | 77.0 | 70.9 | 67.9 | 80.1 | 64.9 | 77.0 | 81.0 | 76.0 | 73.4 | 82.3 | 77.4 |
| **WebRL** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 75.6 | 74.6 | 75.0 | 72.8 | 78.9 | 72.7 | 79.2 | 33.3 | 78.4 | 79.0 | 76.6 | 82.5 | 67.1 | 74.0 |
| Qwen3-VL-8B-Instruct | 76.8 | 75.1 | 79.8 | 79.6 | 70.9 | 59.2 | 80.5 | 34.8 | 82.1 | 83.3 | 77.2 | 84.1 | 66.7 | 74.4 |
| Qwen3-VL-32B-Instruct | 81.0 | 80.7 | 78.2 | 77.1 | 70.0 | 53.6 | 87.0 | 58.3 | 82.1 | 83.3 | 79.4 | 85.7 | 70.3 | 77.2 |
| Qwen3-VL-235B-A22B-Instruct | 80.3 | 82.0 | 74.5 | 76.2 | 77.9 | 73.1 | 84.4 | 62.5 | 81.1 | 83.8 | 79.5 | 77.7 | 82.3 | 79.9 |
| Qwen3.5-122B-A10B | 70.2 | 63.6 | 70.7 | 70.9 | 66.7 | 43.2 | 79.2 | 0.0 | 79.5 | 80.8 | 71.5 | 83.9 | 52.7 | 64.7 |
| Qwen3.6-27B | 83.9 | 85.0 | 78.7 | 81.8 | 81.2 | 76.2 | 81.8 | 53.3 | 80.0 | 82.4 | 82.2 | 80.9 | 84.0 | 82.4 |
| *Mean* | 78.0 | 76.8 | 76.1 | 76.4 | 74.3 | 63.0 | 82.0 | 40.4 | 80.5 | 82.1 | 77.7 | 82.5 | 70.5 | 75.4 |
| **ZeroGUI** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 72.3 | 67.8 | 75.5 | 75.8 | 76.5 | 67.5 | 87.0 | 58.3 | 80.5 | 81.2 | 75.3 | 85.1 | 61.0 | 71.0 |
| Qwen3-VL-8B-Instruct | 72.9 | 68.3 | 75.0 | 75.1 | 77.5 | 70.4 | 92.2 | 76.9 | 76.3 | 76.4 | 75.4 | 85.1 | 61.1 | 71.2 |
| Qwen3-VL-32B-Instruct | 74.9 | 72.1 | 83.0 | 84.2 | 80.8 | 73.5 | 88.3 | 64.0 | 80.5 | 81.4 | 78.4 | 86.1 | 67.3 | 75.5 |
| Qwen3-VL-235B-A22B-Instruct | 76.7 | 74.1 | 81.4 | 82.9 | 78.9 | 71.7 | 93.5 | 81.5 | 82.1 | 83.0 | 79.3 | 86.6 | 69.0 | 76.8 |
| Qwen3.5-122B-A10B | 85.8 | 86.5 | 82.4 | 83.7 | 83.6 | 80.2 | 96.1 | 90.9 | 80.0 | 82.7 | 84.8 | 84.1 | 85.6 | 84.8 |
| Qwen3.6-27B | 78.1 | 78.7 | 77.7 | 79.8 | 77.5 | 73.3 | 89.6 | 73.3 | 80.5 | 82.6 | 78.9 | 79.2 | 78.1 | 78.6 |
| Gemini 3 Flash | 80.3 | 79.6 | 77.1 | 76.8 | 82.2 | 78.2 | 90.9 | 75.9 | 84.7 | 85.0 | 81.3 | 86.7 | 73.7 | 79.7 |
| Gemini 3.1 Flash-Lite | 78.0 | 76.5 | 77.7 | 78.4 | 81.2 | 75.6 | 93.5 | 81.5 | 79.5 | 79.6 | 79.5 | 86.1 | 70.0 | 77.2 |
| *Mean* | 77.4 | 75.5 | 78.7 | 79.6 | 79.8 | 73.8 | 91.4 | 75.3 | 80.5 | 81.5 | 79.1 | 84.9 | 70.7 | 76.8 |

Table 9: Additional offline OGRBench results under each baseline's original trajectory-context setting. Most baselines use only the final screenshot or a small final-state context, while ZeroGUI follows the prior two-screenshot setting.

tive enough to distinguish verifiers.

**Per-task shuffling.** If the trial order followed the source model (`[8B×8, 235B×2]`), EarlyStop@$N$ for small $N$ would be dominated by the 8B success rate and a jump to large $N$ would mostly reflect the position of the stronger 235B rollouts rather than the verifier's judgment. To remove this position prior, we apply a fixed per-task shuffle (seed 42) so that every position $i \in [0, 9]$ contains a 235B trajectory with probability $\approx 20\%$. All reward methods score the same shuffled pool and do not receive the source model metadata. We use Qwen3-VL-32B-Instruct as the judge backbone for all compared reward methods.

For EarlyStop@N, the verifier scans candidates

| Baseline | Original trajectory context | Matched-budget instantiation | Verification style |
|---|---|---|---|
| ZeroGUI (Yang et al., 2025) | Final-state selection; the OS-Themis evaluation protocol uses the last two screenshots. | Use the final ten screenshots and the same four-vote majority setting as the main offline comparison. | Terminal-state judgment with voting. |
| DigiRL (Bai et al., 2024) | Autonomous evaluator prompt that judges task completion from visual evidence, typically centered on the observed screenshot/state. | Provide the same ten selected trajectory screenshots as visual evidence. | Autonomous trajectory evaluator. |
| DistRL (Wang et al., 2025a) | AUTO-EVALUATOR uses the task description with the last screenshot and recent action context. | Provide the same ten selected trajectory screenshots and the corresponding trajectory context. | Autonomous trajectory evaluator. |
| AndroidGen (Lai et al., 2025) | StepCritic-style evaluation uses the task, action history, and final screen to decompose required conditions and check completion. | Provide the same ten selected trajectory screenshots together with the action history. | Condition-wise task checking. |
| WebRL (Qi et al., 2025) | Outcome reward model uses the user intent, action history, and final web state. | Provide the same ten selected trajectory screenshots and trajectory history, replacing web-only state evidence with visual GUI evidence for cross-platform OGRBench. | Outcome reward modeling. |
| OS-Themis (Li et al., 2026) | Milestone-based multi-agent critic that selects and verifies outcome-critical trajectory evidence. | Use the same ten-screenshot trajectory budget as ADAPTRUBRIC for offline comparison. | Structured multi-step evidence verification. |

Table 10: Baseline context budgets and matched-budget instantiations for the expanded OGRBench comparison. All matched-budget runs use at most ten trajectory screenshots to align with ADAPTRUBRIC.

in order and stops at the first trajectory predicted as successful. If no candidate is predicted as successful within the budget, the protocol falls back to the last candidate in the scanned prefix. For BestOfN@N, the verifier scores all candidates in the prefix and selects a predicted-success trajectory; when binary rewards create ties, we use a deterministic last-candidate tie break. Random@N uniformly selects a candidate from the same prefix and serves as the lower reference, while Oracle@N succeeds whenever the prefix contains at least one truly successful trajectory. The main table reports method gains over Random in percentage points.

## G Detailed Case Study

Figure 5 shows a representative AndroidWorld case where task-adaptive criteria are crucial for avoiding a false positive reward. The user asks the agent to edit `note_SiFbv.txt` in Markor and add `Hello, World!` to the top of the note. The ground-truth label is failure. Although the agent opens the correct file and types the target phrase, the final screenshot shows that the original sentence, *"Don't forget to water the plants while I'm away."*, remains above the inserted text. Thus, the trajectory satisfies content presence but violates the positional requirement implied by "to the top of the note".

The baseline verifiers fail because they rely on a surface-level notion of task completion. ZeroGUI states that the final screenshot confirms `Hello, world!` is now the first line of the note and therefore assigns a successful reward. OS-Themis makes a similar error: its critical evidence claims that `Hello, world!` appears as the first line, followed by the original content. Both judgments are false positives. They verify that the requested text appears, but they do not correctly verify the spatial or textual ordering of the final note. In other words, they verify the content but miss the placement.

ADAPTRUBRIC avoids this error by constructing a task-adaptive criterion before making the reward judgment. The coarse stage routes the instruction to the `create_modify` category, whose rubric bounds verification to the final state of the edited item and the explicitly required properties. This prevents the verifier from judging the task using generic signals such as whether some edit was made. The fine stage then adds instruction-specific cues, including whether `Hello, World!` appears at the very top of `note_SiFbv.txt` before any other text. With the fused criterion, the verifier checks both the edited note and the top-placement requirement. It correctly observes that the final state contains the original sentence followed by `Hello, world!`, so the inserted text is not at the top of the note. The trajectory is therefore assigned a failure reward.

The single-component variants further illustrate why both levels are needed. The coarse-only veri-

| Model | Ubuntu | | Mobile | | Windows | | macOS | | Web | | Overall | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | F1 | Acc | F1 | Acc | F1 | Acc | F1 | Acc | F1 | Acc | Prec | Rec | F1 |
| **DigiRL** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 77.5 | 81.2 | 76.1 | 80.5 | 77.0 | 77.4 | 88.3 | 76.9 | 72.1 | 77.6 | 77.1 | 70.7 | 92.0 | 80.0 |
| Qwen3-VL-8B-Instruct | 80.0 | 82.8 | 76.6 | 80.4 | 79.8 | 78.8 | 89.6 | 78.9 | 70.0 | 76.2 | 78.7 | 73.2 | 90.1 | 80.8 |
| Qwen3-VL-32B-Instruct | 85.2 | 86.8 | 82.4 | 85.2 | 82.6 | 80.6 | 94.8 | 88.2 | 79.5 | 82.7 | 84.2 | 79.6 | 91.6 | 85.2 |
| Qwen3-VL-235B-A22B-Instruct | 87.9 | 89.1 | 80.9 | 83.9 | 84.0 | 82.3 | 94.8 | 87.5 | 82.1 | 84.5 | 85.9 | 81.8 | 92.3 | 86.7 |
| Qwen3.5-122B-A10B | 84.2 | 84.7 | 78.7 | 81.0 | 78.4 | 72.6 | 90.9 | 75.9 | 78.9 | 81.1 | 82.3 | 83.0 | 80.9 | 81.9 |
| Qwen3.6-27B | 84.2 | 85.1 | 79.3 | 81.7 | 77.9 | 71.9 | 94.8 | 87.5 | 80.0 | 82.2 | 82.6 | 82.1 | 83.1 | 82.6 |
| *Mean* | 83.2 | 85.0 | 79.0 | 82.1 | 80.0 | 77.3 | 92.2 | 82.5 | 77.1 | 80.7 | 81.8 | 78.4 | 88.3 | 82.9 |
| **DistRL** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 77.5 | 80.6 | 74.5 | 77.6 | 80.3 | 79.6 | 87.0 | 72.2 | 73.7 | 78.3 | 77.5 | 72.7 | 87.7 | 79.5 |
| Qwen3-VL-8B-Instruct | 83.8 | 85.5 | 79.3 | 81.7 | 78.9 | 78.3 | 90.9 | 81.1 | 75.8 | 80.0 | 81.8 | 77.1 | 90.0 | 83.1 |
| Qwen3-VL-32B-Instruct | 86.8 | 88.2 | 80.9 | 83.2 | 83.6 | 81.9 | 93.5 | 84.8 | 79.5 | 83.0 | 84.9 | 80.5 | 91.9 | 85.8 |
| Qwen3-VL-235B-A22B-Instruct | 87.3 | 88.5 | 81.4 | 84.0 | 82.6 | 80.0 | 94.8 | 88.2 | 84.7 | 86.9 | 85.9 | 82.4 | 91.0 | 86.5 |
| Qwen3.5-122B-A10B | 82.9 | 83.1 | 83.0 | 84.2 | 80.3 | 74.7 | 96.1 | 90.3 | 80.5 | 81.8 | 82.9 | 85.1 | 79.4 | 82.2 |
| Qwen3.6-27B | 87.0 | 88.1 | 79.3 | 80.6 | 82.6 | 79.1 | 93.5 | 85.7 | 81.1 | 82.7 | 84.9 | 83.3 | 87.0 | 85.1 |
| *Mean* | 84.2 | 85.7 | 79.7 | 81.9 | 81.4 | 78.9 | 92.6 | 83.7 | 79.2 | 82.1 | 83.0 | 80.2 | 87.8 | 83.7 |
| **AndroidGen** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 79.5 | 82.6 | 73.4 | 78.4 | 71.8 | 73.7 | 71.4 | 54.2 | 81.1 | 84.3 | 77.3 | 70.9 | 92.1 | 80.1 |
| Qwen3-VL-8B-Instruct | 84.6 | 85.7 | 77.7 | 81.3 | 73.2 | 71.9 | 87.0 | 75.0 | 81.1 | 84.6 | 81.6 | 77.7 | 88.4 | 82.7 |
| Qwen3-VL-32B-Instruct | 78.4 | 81.4 | 65.4 | 74.3 | 71.4 | 73.6 | 70.1 | 58.2 | 71.1 | 78.3 | 74.2 | 67.7 | 91.9 | 77.9 |
| Qwen3-VL-235B-A22B-Instruct | 82.5 | 84.3 | 71.3 | 76.7 | 76.5 | 74.5 | 89.6 | 80.0 | 83.2 | 85.6 | 80.6 | 76.0 | 88.9 | 81.9 |
| Qwen3.5-122B-A10B | 83.1 | 85.3 | 72.3 | 78.2 | 78.4 | 78.3 | 85.7 | 74.4 | 75.8 | 80.2 | 80.1 | 74.0 | 92.6 | 82.2 |
| Qwen3.6-27B | 86.5 | 88.1 | 81.4 | 84.2 | 75.6 | 75.5 | 94.8 | 88.9 | 75.8 | 80.5 | 83.2 | 77.4 | 93.4 | 84.7 |
| *Mean* | 82.4 | 84.6 | 73.6 | 78.9 | 74.5 | 74.6 | 83.1 | 71.8 | 78.0 | 82.2 | 79.5 | 74.0 | 91.2 | 81.6 |
| **WebRL** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 85.6 | 86.4 | 77.1 | 75.1 | 80.3 | 75.9 | 94.8 | 86.7 | 77.4 | 78.4 | 83.0 | 84.8 | 80.3 | 82.5 |
| Qwen3-VL-8B-Instruct | 82.9 | 83.2 | 81.4 | 81.7 | 75.1 | 66.7 | 87.0 | 61.5 | 83.2 | 84.9 | 81.8 | 84.4 | 77.6 | 80.9 |
| Qwen3-VL-32B-Instruct | 86.6 | 87.3 | 80.3 | 78.9 | 81.7 | 76.4 | 85.7 | 47.6 | 86.3 | 87.4 | 85.0 | 87.8 | 81.0 | 84.2 |
| Qwen3-VL-235B-A22B-Instruct | 86.0 | 87.7 | 78.2 | 80.4 | 82.2 | 80.6 | 90.9 | 81.1 | 78.9 | 82.6 | 83.7 | 78.7 | 92.1 | 84.9 |
| Qwen3.5-122B-A10B | 82.1 | 81.3 | 76.1 | 78.5 | 68.1 | 48.5 | 83.1 | 31.6 | 78.9 | 80.6 | 78.8 | 84.6 | 70.0 | 76.6 |
| Qwen3.6-27B | 86.1 | 87.8 | 82.4 | 85.1 | 82.6 | 80.6 | 92.2 | 82.4 | 82.6 | 85.1 | 85.0 | 80.2 | 92.6 | 85.9 |
| *Mean* | 84.9 | 85.6 | 79.2 | 80.0 | 78.3 | 71.5 | 89.0 | 65.1 | 81.2 | 83.2 | 82.9 | 83.4 | 82.3 | 82.5 |
| **ZeroGUI** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 83.8 | 84.0 | 74.5 | 76.2 | 80.3 | 75.6 | 90.9 | 78.8 | 81.1 | 83.2 | 82.0 | 83.2 | 80.0 | 81.6 |
| Qwen3-VL-8B-Instruct | 84.6 | 85.0 | 83.0 | 84.5 | 79.3 | 74.4 | 94.8 | 88.2 | 78.4 | 80.6 | 83.3 | 84.1 | 81.9 | 83.0 |
| Qwen3-VL-32B-Instruct | 84.6 | 85.2 | 83.5 | 85.0 | 80.3 | 75.6 | 96.1 | 90.3 | 82.1 | 83.5 | 84.1 | 84.6 | 83.1 | 83.9 |
| Qwen3-VL-235B-A22B-Instruct | 86.9 | 87.3 | 84.0 | 85.8 | 84.5 | 80.9 | 96.1 | 90.3 | 87.4 | 88.6 | 86.7 | 87.2 | 85.9 | 86.5 |
| Qwen3.5-122B-A10B | 85.8 | 86.5 | 82.4 | 83.7 | 83.6 | 80.2 | 96.1 | 90.9 | 80.0 | 82.7 | 84.8 | 84.1 | 85.6 | 84.8 |
| Qwen3.6-27B | 86.2 | 87.5 | 80.9 | 83.5 | 83.6 | 81.5 | 94.8 | 88.9 | 82.1 | 84.7 | 85.0 | 81.2 | 90.9 | 85.8 |
| *Mean* | 85.3 | 85.9 | 81.4 | 83.1 | 81.9 | 78.0 | 94.8 | 87.9 | 81.9 | 83.9 | 84.3 | 84.1 | 84.6 | 84.3 |
| **OS-Themis** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 72.6 | 71.4 | 79.3 | 78.9 | 75.1 | 68.3 | 84.4 | 57.1 | 80.0 | 81.9 | 75.5 | 79.5 | 68.3 | 73.5 |
| Qwen3-VL-8B-Instruct | 76.2 | 75.0 | 84.0 | 83.7 | 78.4 | 72.0 | 83.1 | 51.9 | 81.6 | 82.4 | 78.7 | 84.6 | 69.9 | 76.5 |
| Qwen3-VL-32B-Instruct | 77.1 | 74.6 | 81.9 | 80.7 | 76.5 | 65.3 | 90.9 | 77.4 | 83.7 | 81.9 | 79.3 | 91.6 | 64.1 | 75.5 |
| Qwen3-VL-235B-A22B-Instruct | 86.4 | 86.8 | 93.6 | 93.7 | 77.5 | 69.6 | 93.5 | 82.8 | 91.6 | 91.9 | 87.1 | 90.5 | 82.7 | 86.4 |
| Qwen3.5-122B-A10B | 85.8 | 85.8 | 88.3 | 88.2 | 79.8 | 73.0 | 88.3 | 66.7 | 79.0 | 75.6 | 84.5 | 92.0 | 75.3 | 82.8 |
| Qwen3.6-27B | 87.9 | 88.5 | 93.6 | 93.9 | 88.3 | 86.9 | 96.1 | 90.3 | 92.1 | 92.1 | 89.7 | 90.2 | 89.0 | 89.6 |
| *Mean* | 81.0 | 80.4 | 86.8 | 86.5 | 79.3 | 72.5 | 89.4 | 71.0 | 84.7 | 84.3 | 82.5 | 88.1 | 74.9 | 80.7 |
| **ADAPTRUBRIC** | | | | | | | | | | | | | | |
| Qwen3-VL-4B-Instruct | 84.3 | 85.6 | 80.9 | 81.1 | 83.1 | 79.5 | 97.4 | 94.1 | 82.1 | 84.5 | 84.1 | 82.8 | 85.9 | 84.3 |
| Qwen3-VL-8B-Instruct | 83.9 | 85.1 | 83.5 | 84.6 | 81.7 | 79.1 | 97.4 | 94.1 | 81.6 | 83.3 | 84.0 | 82.6 | 85.9 | 84.2 |
| Qwen3-VL-32B-Instruct | 86.5 | 86.6 | 85.6 | 85.9 | 80.3 | 75.0 | 93.5 | 83.9 | 86.8 | 87.6 | 85.9 | 89.3 | 81.3 | 85.1 |
| Qwen3-VL-235B-A22B-Instruct | 86.1 | 86.7 | 87.8 | 88.9 | 84.5 | 81.4 | 94.8 | 87.5 | 88.9 | 90.0 | 86.9 | 87.0 | 86.7 | 86.8 |
| Qwen3.5-122B-A10B | 86.2 | 86.8 | 89.9 | 90.4 | 81.7 | 77.2 | 94.8 | 88.2 | 88.9 | 89.8 | 86.9 | 87.8 | 85.4 | 86.6 |
| Qwen3.6-27B | 87.3 | 88.5 | 91.0 | 91.6 | 84.5 | 82.5 | 93.5 | 85.7 | 91.6 | 92.4 | 88.3 | 85.4 | 92.1 | 88.7 |
| *Mean* | 85.7 | 86.5 | 86.5 | 87.1 | 82.6 | 79.1 | 95.2 | 88.9 | 86.6 | 87.9 | 86.0 | 85.8 | 86.2 | 86.0 |

Table 11: Additional offline OGRBench results under the matched ten-screenshot trajectory budget. We include DigiRL, DistRL, AndroidGen, WebRL, ZeroGUI, OS-Themis, and ADAPTRUBRIC across multiple judge backbones.

fier predicts success because it recognizes that the note was modified and that the target text is visible, but the category-level rubric alone is too broad to enforce the exact top-of-note placement. The fine-only verifier receives the top-placement cue, but still misreads the final order and predicts success.

| Pool | Trials/task | All-fail | All-pass | Mixed | Trial SR | Oracle@$N$ | Headroom |
|---|---|---|---|---|---|---|---|
| 8B-only | 8 | 48 (42.5%) | 44 (38.9%) | 21 (18.6%) | 51.44% | 57.52% | +6.08 |
| 235B-only | 2 | 46 (40.7%) | 52 (46.0%) | 15 (13.3%) | 52.65% | 59.29% | +6.64 |
| 8B+235B | 10 | **32 (28.3%)** | **29 (25.7%)** | **52 (46.0%)** | 51.68% | **71.68%** | **+20.00** |

Table 12: Candidate-pool composition for reward-guided test-time scaling. Mixed tasks contain both successful and failed trajectories, and are the only tasks where reward-guided selection can change the final task success. Headroom is Oracle@$N$ minus Random@1.

Only the fused coarse-to-fine criterion combines a structured final-state check with the instruction-specific positional constraint, leading to the correct failure judgment.

# H PROMPTS

We provide the prompt-facing components used by ADAPTRUBRIC. The rubric bank is loaded once from the cold-start bank, the coarse stage routes each instruction to one of the eight rubric families and retrieves the corresponding bank entry, and the fine stage optionally generates instance-level rubric cues.

## H.1 Rubric Bank

The rubric bank $\mathcal{B}$ contains one cold-start `RubricSet` for each task family. Each entry stores natural-language sections, namely `verification_steps`, `common_pitfalls`, `special_rules`, and `output_format`. It also stores named `rubric_items` with criticality labels for structured ablations. The default verifier uses the natural-language sections; the structured `rubric_items` are kept for the legacy structured ablation.

```
RubricSet(
  id="RSET_INFO_QUERY",
  categories=["info_query"],
  description="Read screen content and return a value, count,
      list, or title.",
  sections={
    "verification_steps": [
      "Check explicit answer-format constraints, if any.",
      "Ground content correctness in screenshots from the
      trajectory.",
      "Check answer/submission evidence when the task
      requires an answer."
    ],
    "common_pitfalls": [
      "Correct content but wrong requested format.",
      "Answer hallucinated from memory rather than screen
      evidence.",
      "Wrong count or list item due to misreading visible
      content."
    ],
    "special_rules": [
      "A format constraint is hard only when stated by the
      instruction.",
      "The supporting evidence need not appear in the final
      frame.",
      "Semantic equivalence is acceptable when no format is
      specified."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
  },
  rubric_items=[
    ("R1", "Answer Format Match", "critical"),
    ("R2", "Content Correctness", "critical"),
    ("R3", "Answer Recorded", "default"),
    ("R4", "Screen-Grounded Source", "auxiliary")
  ]
)

RubricSet(
  id="RSET_CREATE_MODIFY",
  categories=["create_modify"],
  description="Create a new item or modify explicitly
      requested properties.",
  sections={
    "verification_steps": [
      "Collect only properties explicitly stated in the
      instruction.",
      "Check whether the required properties are reflected
      in the end state.",
      "Check that the edit is not left mid-flight."
    ],
    "common_pitfalls": [
      "Inventing filename, folder, toast, or UI-path
      requirements.",
      "Only a subset of explicitly required properties is
      applied.",
      "The final frame still shows an uncommitted editor or
      form."
    ],
    "special_rules": [
      "Any valid save or commit path counts unless a path is
       specified.",
      "Unmentioned properties are ignored."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
  },
  rubric_items=[
    ("R1", "Required Properties Set", "critical"),
    ("R2", "Save / Confirm Action Performed", "critical"),
    ("R3", "Final State Persisted", "critical"),
    ("R4", "No Unspecified Constraint Imposed", "default"),
    ("R5", "Form / Editor Closed After Save", "auxiliary")
  ]
)

RubricSet(
  id="RSET_DELETE_CLEANUP",
  categories=["delete_cleanup"],
  description="Remove items or clear state under specific or
       global scope.",
  sections={
    "verification_steps": [
      "Distinguish specific deletion from global cleanup.",
      "Check final-state absence or zero remaining matches.",

      "Check that any confirmation dialog has been committed
      .",
      "Guard against inferring absence from an unrelated
      screen."
    ],
    "common_pitfalls": [
      "Passing after one deletion when the instruction says
      all.",
      "Confirmation dialog still visible in the final frame
      .",
      "Deleting a similarly named but wrong item."
    ],
    "special_rules": [
      "Global cleanup requires visibly complete final
      evidence.",
      "Post-confirmation return to a list with the item gone
```

```
      is strong evidence."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
  },
  rubric_items=[
    ("R1", "Correct Target / Scope Identified", "critical"),
    ("R2", "Deletion Confirmed", "critical"),
    ("R3", "Final-State Absence / Count Verified", "critical
      "),
    ("R4", "Pre-Deletion Visibility", "auxiliary")
  ]
)

RubricSet(
  id="RSET_COMMUNICATION",
  categories=["communication"],
  description="Send, reply, forward, or share content to a
      recipient.",
  sections={
    "verification_steps": [
      "Check source content.",
      "Check recipient and channel.",
      "Check post-send evidence instead of treating a draft
      as sent."
    ],
    "common_pitfalls": [
      "Compose screen remains visible, so the message may be
       unsent.",
      "Wrong recipient or wrong channel.",
      "Visible sent bubble has different content."
    ],
    "special_rules": [
      "Thread view with a new bubble is sufficient for
      messaging apps.",
      "Literal message text must match up to minor
      whitespace differences."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
  },
  rubric_items=[
    ("R1", "Source Content Correctness", "critical"),
    ("R2", "Recipient Correctness", "critical"),
    ("R3", "Send Action Confirmation", "critical"),
    ("R4", "Channel / App Match", "default"),
    ("R5", "No Side Effects", "auxiliary")
  ]
)

RubricSet(
  id="RSET_TRANSFER",
  categories=["transfer"],
  description="Move, copy, import, export, clone, unzip, or
      save content from source to sink.",
  sections={
    "verification_steps": [
      "Identify source evidence.",
      "Check destination evidence.",
      "Check content fidelity at the sink.",
      "Detect partial or corrupted transfer."
    ],
    "common_pitfalls": [
      "Source read but destination left empty.",
      "Only a subset transferred when all was required.",
      "Manual retype introduces truncation or digit errors."
    ],
    "special_rules": [
      "Both source-side and sink-side evidence are required
      .",
      "Fidelity mismatch is a fail even if the transfer
      action succeeded."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
  },
  rubric_items=[
    ("R1", "Source Read Verified", "critical"),
    ("R2", "Destination Reached", "critical"),
    ("R3", "Content Fidelity", "critical"),
    ("R4", "Correct Transfer Mechanism", "default"),
    ("R5", "Completeness for all Tasks", "auxiliary")
  ]
)

RubricSet(
  id="RSET_STATE_NAV",
  categories=["state_navigation"],
  description="Navigate to a view, toggle a setting, or
      configure a preference.",
  sections={
    "verification_steps": [
      "Check the final-frame state.",
      "Check persistence of the requested state or view.",
      "Avoid requiring side effects not requested by the
      instruction."
    ],
    "common_pitfalls": [
      "Correct view was only visited mid-trajectory.",
      "Toggle was tapped but visual state did not change.",
      "Parent page reached instead of the requested sub-page
      ."
    ],
    "special_rules": [
      "The final frame is the primary anchor.",
      "Any UI path is acceptable unless a path is specified
      ."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
  },
  rubric_items=[
    ("R1", "Final Screen Match", "critical"),
    ("R2", "State Persisted", "critical"),
    ("R3", "Exact Sub-Page Reached", "default"),
    ("R4", "Toggle Visual State", "auxiliary")
  ]
)

RubricSet(
  id="RSET_COMPOSITE",
  categories=["composite_workflow"],
  description="Two or more independent sub-goals must all be
       completed.",
  sections={
    "verification_steps": [
      "Decompose the instruction into independent
      checkpoints.",
      "Check evidence for each sub-goal using its underlying
       family logic.",
      "Fail when any sub-goal is silently dropped.",
      "Treat ordering as required only when the instruction
      makes it required."
    ],
    "common_pitfalls": [
      "Judging only the first or most visible sub-goal.",
      "Over-decomposing implementation steps into false sub-
      goals."
    ],
    "special_rules": [
      "Composite success is an AND over sub-goals.",
      "Compositeness does not raise the per-sub-goal pass
      bar."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
  },
  rubric_items=[
    ("R1", "All Sub-goals Identified", "critical"),
    ("R2", "Each Sub-goal Evidenced", "critical"),
    ("R3", "No Silent Drop", "critical")
  ]
)

RubricSet(
  id="RSET_GENERAL",
  categories=["*"],
  description="Conservative fallback for tasks outside the
      specialized families.",
  sections={
    "verification_steps": [
      "Reconstruct the user's intended outcome.",
      "Check end-state evidence.",
      "Check action-visual consistency.",
      "Use conservative judgment under genuine ambiguity."
    ],
    "common_pitfalls": [
      "Intermediate success but final state drifts elsewhere
      .",
      "Action trace shows intent but no visual corroboration
      ."
    ],
    "special_rules": [
      "Prefer EVIDENCE_SUFFICIENT=no when evidence is
      unclear.",
      "Do not infer success from a confident-looking answer
      alone."
    ],
    "output_format": OUTPUT_FORMAT_HOLISTIC
```

```
  },
  rubric_items=[
    ("R1", "Task Goal Achieved", "critical"),
    ("R2", "Final State Consistent", "default"),
    ("R3", "Action-Visual Consistency", "auxiliary")
  ]
)

Shared meta rule appended to each family:
- Evidence anchors guide attention; they do not add
      requirements beyond the task instruction.
- If the final state clearly satisfies the user's intent but
       a specific anchor cannot be confirmed, prefer PASS
      unless the instruction made that anchor mandatory.
- Do not fail for properties the instruction did not state,
      such as filenames, folders, confirmation toasts,
      specific phrasing, or cosmetic details.
```

### H.2 Category-Level Coarse Rubric Retrieval

The coarse stage first routes the instruction to a rubric family. In the LLM-routing mode, the implementation uses the following classifier prompt and parses the single-line category output. The retrieved rubric is then selected deterministically: if the category is not `general`, the retriever returns the unique `RubricSet` whose `categories` field contains that category; otherwise it returns the wildcard `RSET_GENERAL`.

```
You are routing a GUI task to one of 8 rubric families.

TASK INSTRUCTION:
{instruction}

CATEGORIES (pick exactly one):
- info_query           : the agent must answer a question by
       reading screen content (a value, a count, a list, a
      title)
- create_modify        : the agent creates a NEW item or
      edits properties of an EXISTING item (note, event,
      file, drawing, form field, spreadsheet cell)
- delete_cleanup       : the agent removes items, uninstalls,
       clears data, gets rid of content
- communication        : the agent sends / replies /
      forwards / shares content to a recipient via a
      messaging channel
- transfer             : the agent moves content source ->
      sink (copy, paste, import, export, clone repo, unzip,
      save-as to a path)
- state_navigation     : the agent toggles a setting,
      configures a preference, changes theme/font/default
      app, navigates to a specific view
- composite_workflow   : the instruction lists TWO OR MORE
      INDEPENDENT sub-goals (different apps, different
      target objects) connected by "and then", "; then", ";".
       Sequential steps that all accomplish ONE goal are NOT
       composite.
- general              : the task genuinely does not fit any
       specific family above. Use ONLY as a last resort.

Decision rules:
1. If two or more categories seem to apply, pick the one
      matching the PRIMARY outcome the user cares about.
2. Prefer a specific family over general whenever possible.
3. composite_workflow requires independent sub-goals, not
      just multiple sub-steps of one operation.

OUTPUT EXACTLY ONE LINE (no other text, no markdown):
CATEGORY: <one of info_query | create_modify |
      delete_cleanup | communication | transfer |
      state_navigation | composite_workflow | general>
```

After retrieval, the selected category-level rubric sections are rendered into the verifier prompt. The implementation uses the following prompt skeleton; in the coarse-only setting, `{additional_attention}` is empty.

```
You are evaluating a GUI task.

TASK: {instruction}

COMPILED EVIDENCE:
- Agent's submitted answer: {agent_answer}
- Key text entered by agent: {typed_texts}
- Key actions performed: {key_actions}

ACTION TRACE:
{action_trace}

{verification_steps}

{common_pitfalls}

{special_rules}

{additional_attention}

Screenshots (in order) are shown below.

End your response with EXACTLY these three lines (do not add
       any text after):
SCORE: [0/1]
EVIDENCE_SUFFICIENT: [yes/no]
MISSING_EVIDENCE: [source_missing | sink_missing |
      final_state_ambiguous | answer_format_mismatch |
      global_count_unverified | subgoal_unverified | none]
```

### H.3 Instance-Level Fine Rubric Generation

The fine stage generates zero, one, or two instruction-specific rubric cues. It receives the instruction, the routed task family, and the critical items from the retrieved coarse rubric. The generator is explicitly allowed to abstain with `NO_HINT`; otherwise, the generated cues are injected into `{additional_attention}` in the verifier prompt above.

```
You are generating optional task-specific rubric cues for a
      GUI task verifier.

The verifier already has a coarse task-type rubric that
      covers standard verification
(action trace, final screenshot, task completion). Your job
      is to decide whether this
specific task needs additional fine-grained attention cues,
      and if so, generate them.

TASK INSTRUCTION:
{instruction}

TASK TYPE: {category}

COARSE RUBRIC ITEMS already applied (for reference -- do NOT
       paraphrase them):
{rubric_items_block}

YOUR JOB: Decide whether 0, 1, or 2 task-specific rubric
      cues would help the verifier.

OUTPUT NO_HINT when:
  - The coarse rubric already covers the task adequately.
  - Any cue you could write would merely restate the coarse
      rubric in task-specific words.
  - The task is straightforward and the main success
      criterion is obvious from the instruction.
  - You cannot generate a cue without introducing
      requirements NOT present in the instruction.

OUTPUT 1-2 CUES when the task has:
  - An explicit constraint that the coarse rubric might miss
       (a specific value, format,
    recipient, negative constraint like "do not send", exact
       date/time, recurrence rule).
  - A plausible partial-completion trap where the agent
      might stop one step early.
```

```
  - An answer-format requirement where the verifier might
      accept wrong structure.

RULES for valid cues:
  1. Must be grounded in an EXPLICIT phrase/value/constraint
       from the task instruction.
  2. Must distinguish plausible partial completion from
      actual success.
  3. Must NOT require "final screenshot" evidence unless the
       instruction explicitly
     asks for a visible final state. The verifier can use
      action trace, intermediate
     screenshots, and submitted answers as evidence.
  4. Must NOT use over-constraining language: avoid "visibly
      ", "exactly", "every/all"
     (unless the instruction itself uses these words), "
      saved/persisted" (unless the
     task is about saving), "final state must show".
  5. Must NOT add requirements absent from the instruction (
      e.g., "must remain in the
     app", "must show confirmation dialog", "must prove
      persistence").
  6. Should specify what evidence source is acceptable:
      action trace, any screenshot,
     submitted answer, or final screenshot.
  7. Use soft attention phrasing: "Pay attention to whether
      ...", "Check if ...",
     "The key distinction is ...", "Evidence of success
      includes ...".
     Do NOT start with "Confirm that ..." (too imperative).
  8. One sentence each, <= 40 words.

OUTPUT FORMAT (exactly one of these two forms, no other text
      ):

If no cue needed:
NO_HINT: <one-sentence reason>

If 1-2 cues:
CUE_1: <sentence>
CUE_2: <optional second sentence>
```

When cues are available, they are rendered as a soft task-specific attention block rather than as additional hard requirements.

```
TASK-SPECIFIC ATTENTION CUES (focus aids -- NOT additional
      pass/fail items):
The bullets below highlight aspects of THIS task that are
      worth attending to.
They guide where to look; they do NOT add pass/fail
      requirements beyond the
task instruction itself, and they do NOT override the META
      RULE above.
- {cue_1}
- {cue_2}

CUE USAGE -- REMINDERS THAT OVERRIDE EARLIER WORDING:
- A cue you cannot confirm in the screenshots is NOT, by
      itself, a failure.
  Base the verdict on whether the OVERALL final state
      satisfies the user's
  intent in the task instruction -- not on per-cue
      verifiability.
- Use EVIDENCE_SUFFICIENT=no only when the OVERALL verdict
      is uncertain,
  not when a single cue is unconfirmed but the rest of the
      evidence agrees.
- If the final state clearly satisfies the user's intent,
      prefer SCORE: 1
  even when one or more cues are visually ambiguous.
- A cue CLEARLY CONTRADICTED by visible evidence is still a
      strong FAIL signal.
```


Figure 5: Qualitative case study illustrating the importance of task-adaptive criteria. The task requires adding `Hello, World!` to the top of `note_SiFbv.txt` in Markor. The trajectory inserts the text, but places it below the original note content, yielding a failed task. Baseline verifiers focus on content presence and incorrectly judge the task as completed. AdaptRubric combines a coarse create/modify rubric with fine placement cues, enabling the verifier to check both the edited item and the instruction-specific requirement that no text appears above the inserted phrase.