

Benchmarking LLM Judges for Mobile Agent Evaluation

Ziqiang Wang^{1,2}, Li Gu^{1,2}, Zhixiang Chi³, Zhi Liu⁴,
Seyed Mehdi Ayyoubzadeh⁵, Yuanhao Yu⁵, Yang Wang^{1,2},

¹Mila – Québec AI Institute, ²Concordia University, ³University of Toronto,
⁴Shanghai University, ⁵McMaster University

Abstract

Mobile agent benchmarks increasingly rely on LLM-based judges to evaluate task completion, yet the reliability of these judges on mobile agent trajectories remains largely unexamined. We introduce MOBILEJUDGE BENCH, a benchmark for systematically evaluating LLM-as-judge methods on mobile agent trajectories. Our benchmark comprises 931 human-annotated trajectories spanning 6 mobile agent benchmarks, 4 agent models, and 68 apps. We evaluate 6 judge methods (five adapted from SPA-Bench, A3 with two modes, AndroidArena, and AgentRewardBench, plus a simple baseline we design) across multiple LLM backends. Our experiments reveal three key findings. First, a simple baseline judge with sampled screenshots is competitive with, and often exceeds, purpose-built methods, indicating that more elaborate judge pipelines do not consistently improve judge quality; among competitive methods, the LLM backbone is the primary driver. Second, benchmark quality metrics reliably predict real-world judge utility: they correlate with both agent ranking fidelity for evaluation and downstream performance when judges serve as reward signals for on-policy reinforcement learning. Third, failure analysis across two LLM backends uncovers qualitatively opposite failure profiles, one conservative and the other permissive, linked to the backbone’s precision-recall characteristics.

1 Introduction

Using LLMs to judge the outputs of other models has emerged as a scalable alternative to human evaluation across a wide range of AI tasks (Zheng et al., 2023; Gu et al., 2024). In the domain of autonomous mobile agents that operate smartphones on behalf of users, LLM-based judges play a dual role: they serve as *evaluators* in benchmarks that measure agent progress on realistic Android tasks (Rawles et al., 2025; Chen et al., 2025; Chai et al., 2025; Xing et al., 2024; Lee et al., 2024;

Xu et al., 2025b), and increasingly as *reward signals* for reinforcement learning (RL) training (Bai et al., 2024; Qi et al., 2025; Xu et al., 2025a). Judging mobile agent trajectories is particularly challenging: it requires interpreting multimodal evidence (sequences of screenshots, UI element trees, and executed actions) across long trajectories spanning diverse apps and interaction patterns. This makes judge quality critical not only for measuring progress but also for driving it.

While some benchmarks provide rule-based state checkers (Rawles et al., 2025; Xu et al., 2025b), these require per-task engineering and are difficult to scale to new tasks or apps. As a result, benchmarks increasingly adopt LLM-based judges, whether through coarse-to-fine screenshot matching (Chen et al., 2025), essential-state decomposition (Chai et al., 2025), or direct LLM judgment (Xing et al., 2024). However, the reliability of these LLM-based evaluators on mobile agent trajectories has not been systematically examined. Researchers select a benchmark, adopt its built-in judge, and report numbers without questioning the evaluation itself. This is problematic for two reasons. First, an unreliable judge produces noisy leaderboards: a judge that misclassifies even 10–15% of trajectories can shift agent rankings substantially (§5.2.1). Second, when judges serve as reward signals for RL training, errors in the reward directly corrupt the learning process (Gao et al., 2023; Huang et al., 2024). Yet neither the degree of unreliability nor which quality metrics (*e.g.*, accuracy, precision, recall) matter most for these downstream applications has been established for mobile agents.

In this work, we introduce MOBILEJUDGE BENCH, to our knowledge the first benchmark for evaluating LLM-as-judge methods on mobile agent trajectories (Figure 1). We collect 931 trajectories from 6 established mobile agent benchmarks, generated by 4 diverse agent models

across 68 apps, and obtain human expert annotations with multi-annotator redundancy. Using this benchmark, we evaluate 6 judge methods (five adapted from SPA-Bench, A3 with two modes, AndroidArena, and AgentRewardBench, plus a simple baseline we design for controlled ablation) across multiple LLM backends. Crucially, we go beyond measuring intrinsic judge accuracy: we validate that our benchmark metrics predict real-world judge utility for both agent evaluation and RL training.

Our contributions are as follows:

1. **A judge benchmark for mobile agents.** We construct a dataset of 931 human-annotated trajectories spanning 6 benchmarks, 4 agents, and 68 apps, together with a unified evaluation framework that standardizes judge assessment with classification and ranking metrics (§3).
2. **Systematic evaluation of judge methods.** We evaluate 6 judge methods across 5 LLM backends, revealing that no single method dominates and that judge accuracy depends substantially on both the method and the LLM backbone (§5.1).
3. **A simple baseline judge with ablation study.** We design a streamlined judge that is competitive with or exceeds purpose-built methods (up to 90.9% accuracy). Ablations over screenshot count, image resolution, UI meta-data, and agent reasoning show that screenshot count is the dominant design variable, while the other inputs have marginal impact (§4.2, §5.1).
4. **Benchmark metrics predict real-world judge utility.** We validate our benchmark through two downstream applications. For *evaluation*, meta-correlation analysis shows that judge quality metrics reliably predict agent ranking fidelity and success rate estimation accuracy. For *training*, on-policy RL experiments demonstrate that judge quality on our benchmark carries through to downstream agent performance (§5.2).
5. **Failure pattern analysis.** We analyze hard-core failure cases where nearly all judge methods err, revealing that different LLM backends produce qualitatively opposite failure profiles, one conservative (false-negative-heavy) and

the other permissive (false-positive-heavy), with distinct root cause taxonomies (§5.3).

2 Related work

LLM-as-a-judge. Using LLMs as surrogate evaluators was popularized by MT-Bench and Chatbot Arena (Zheng et al., 2023), where GPT-4 achieves over 80% agreement with human preferences on chat evaluation. Subsequent work has identified systematic biases in LLM judges, including position bias, verbosity bias, and self-preference (Gu et al., 2024; Li et al., 2024a), and studied judge reliability (Jung et al., 2024), contamination via preference leakage (Li et al., 2025), and fine-tuning dedicated judge models (Zhu et al., 2023; McAleese et al., 2024). Zhuge et al. (2024) extend the paradigm to Agent-as-a-Judge, equipping evaluators with agentic capabilities. This body of work focuses predominantly on evaluating text outputs (*e.g.*, chat responses, code). In the mobile agent domain, LLM judges must handle multi-step trajectories with interleaved screenshots, actions, and UI states, as seen in methods from SPA-Bench (Chen et al., 2025), A3 (Chai et al., 2025), and AndroidArena (Xing et al., 2024).

Mobile and GUI agent benchmarks. The mobile agent evaluation landscape is fragmented across benchmarks that differ in task design, environment, and evaluation method. AndroidWorld (Rawles et al., 2025) provides programmatic, state-based success checking; B-MoCA (Lee et al., 2024) tests generalization across device configurations with rule-based detectors; SPA-Bench (Chen et al., 2025) uses coarse-to-fine screenshot matching with MLLM verification; A3 (Chai et al., 2025) employs essential-state decomposition; AndroidArena (Xing et al., 2024) uses direct GPT-4 judgment; and AndroidLab (Xu et al., 2025b) verifies sub-goal completion via UI tree matching. On the desktop and web side, OSWorld (Xie et al., 2024), WebArena (Zhou et al., 2024), VisualWebArena (Koh et al., 2024), and Mind2Web (Deng et al., 2023) provide complementary evaluation environments. Large-scale datasets such as Android-in-the-Wild (Rawles et al., 2023) and AndroidControl (Li et al., 2024b) supply training data. While rule-based checkers are reliable by construction for their supported tasks, the LLM-based judges that many benchmarks rely on, from SPA-Bench to AndroidDaily (Sui et al., 2026), have not been systematically validated against human ground truth.

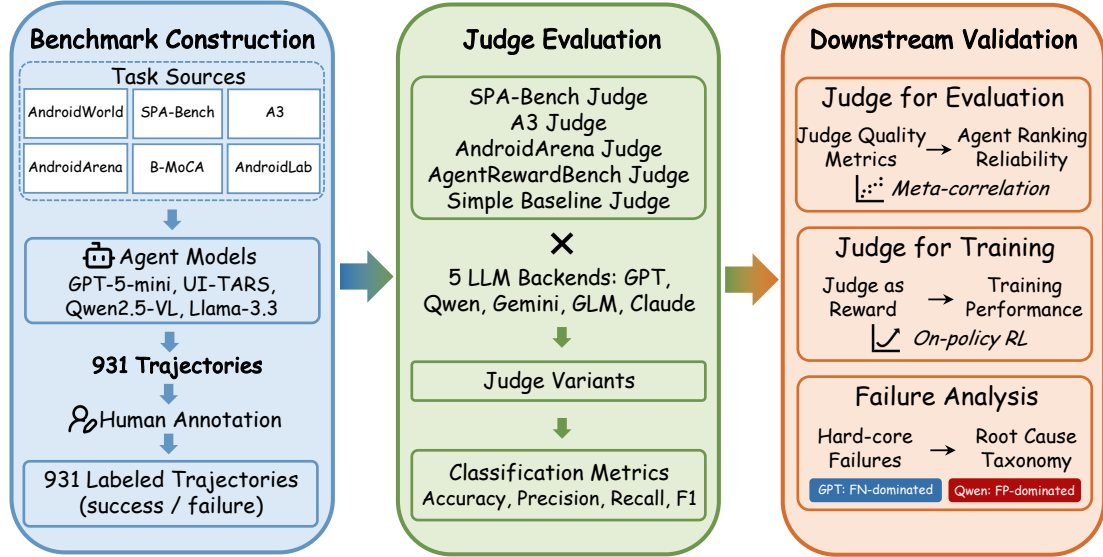


Figure 1: Overview of MOBILEJUDGE BENCH. We construct a judge benchmark from 931 human-annotated mobile agent trajectories spanning 6 benchmarks (left, §3), evaluate 6 LLM-as-judge methods across 5 backends (middle, §5.1), and validate that benchmark metrics predict judge utility for agent evaluation (§5.2.1) and on-policy training (§5.2.2), complemented by a root-cause failure analysis (§5.3) (right).

Automatic evaluation of agent trajectories.

AgentRewardBench (Lù et al., 2025) is the most directly related work: it benchmarks LLM judges on 1,302 web agent trajectories across 5 web benchmarks, evaluating judge methods including AER (Pan et al., 2024) and NNetNav (Murty et al., 2024). Concurrent work on mobile evaluation includes AutoEval (Sun et al., 2025), which decomposes tasks into expected success states for AndroidLab. WebJudge (Xue et al., 2025) trains a dedicated 7B judge model for web agent trajectories. More recent approaches move beyond passive screenshot inspection: VAGEN (Cui et al., 2026) and ProRe (Dai et al., 2025) introduce proactive verification, where the judge actively interacts with the environment to collect evidence, and AJ-Bench (Shi et al., 2026) evaluates such environment-aware judges on search, data-system, and desktop-GUI tasks. Our work differs from AgentRewardBench in three key respects: we focus on mobile agents with their distinct observation types and fragmented benchmark ecosystem; we integrate judge methods from 4 existing agent benchmarks into a unified evaluation; and we empirically link judge quality to downstream on-policy training performance, a connection not explored in existing agent judge benchmarks (Appendix B).

Reward modeling for agent training. The use of learned or model-based rewards for training originates from RLHF (Ouyang et al., 2022) and has

been extended through DPO (Rafailov et al., 2023) and RLAI (Bai et al., 2022). In the agent domain, DigiRL (Bai et al., 2024) demonstrates that a VLM evaluator can serve as the reward signal for RL training of device-control agents, achieving substantial improvements over supervised fine-tuning. WebRL (Qi et al., 2025) shows that reward model quality is critical for web agent training. For mobile agents specifically, MobileRL (Xu et al., 2025a) and MobileGUI-RL (Shi et al., 2025) apply online RL with various reward signals. RewardBench (Lambert et al., 2025) provides a general benchmark for reward models but does not cover agent-specific evaluation. Scaling laws for reward model overoptimization (Gao et al., 2023) establish that imperfect reward models lead to predictable performance degradation. Related to this line of work, we empirically measure whether judge quality metrics on a benchmark predict downstream agent training performance.

3 Judge benchmark construction

MOBILEJUDGE BENCH consists of 931 human-annotated mobile agent trajectories drawn from 6 established benchmarks, generated by 4 agent models across 289 unique tasks and 68 real Android apps. We describe the data collection, annotation process, and evaluation framework below.

Benchmark	Tasks	Traj.	Apps
SPA-Bench	69	220	38
AndroidWorld	52	205	10
A3	58	154	19
AndroidArena	51	151	14
B-MoCA	35	123	12
AndroidLab	24	78	7
Total	289	931	68

Table 1: Benchmark statistics. Tasks are unique task definitions; trajectories include up to 4 agent runs per task. Total app count excludes cross-app task categories in SPA-Bench; some apps appear in multiple benchmarks.

3.1 Task and trajectory collection

Task sources. We select tasks from 6 mobile agent benchmarks (Table 1) that collectively represent the major evaluation paradigms: programmatic state checking (AndroidWorld, B-MoCA), screenshot matching (SPA-Bench), essential-state decomposition (A3), direct LLM judgment (AndroidArena), and sub-goal matching (AndroidLab).

Agent models and trajectory format. We generate trajectories using 4 agents: GPT-5-mini and Qwen2.5-VL-72B (via M3A (Rawles et al., 2025)), UI-TARS-72B (Qin et al., 2025), and Llama-3.3-70B (Grattafiori et al., 2024) (via T3A). Each agent attempts every task, yielding up to 4 trajectories per task. Each trajectory is a sequence of (screenshot, action, UI tree, agent reasoning) tuples recorded at every step, stored in a unified format.

3.2 Human annotation

We recruit 9 graduate-student annotators to label each trajectory with a binary success judgment (success/failure). Each trajectory is independently annotated by 2–4 annotators using a custom annotation platform (Appendix A), which provides trajectory video playback, step-by-step screenshot inspection, and structured annotation forms. Annotators review the task instruction, screenshots, actions, and agent reasoning. Average pairwise agreement on the success label is 88.4%. For trajectories with annotator disagreement, annotators discuss and finalize the label. The final dataset is approximately balanced: 492 success (52.8%) and 439 failure (47.2%).

3.3 Evaluation framework

We design a unified evaluation pipeline that takes any judge method and any trajectory as input and produces standardized metrics. Given a set of judge predictions and human ground-truth labels, we re-

port accuracy, precision, recall, F1, and balanced accuracy at the trajectory level. A “positive” is a successful trajectory: precision measures how often the judge’s success predictions are correct, while recall measures how often truly successful trajectories are identified. Because the dataset is approximately balanced (53% positive), accuracy and balanced accuracy are close, but we report both for completeness.

4 Judge methods

We evaluate 6 LLM-as-judge methods on our benchmark. Five are adapted from existing agent benchmarks (SPA-Bench, A3 with two evaluation modes, AndroidArena, and AgentRewardBench) and one is a simple baseline we design for controlled ablation. All methods take a task instruction and agent trajectory as input and output a binary success/failure prediction with reasoning. They differ in how they represent the trajectory (screenshots vs. text, full sequence vs. final state) and how they structure the evaluation prompt.

4.1 Existing judge methods

SPA-Bench judge (Chen et al., 2025). The original SPA-Bench evaluation uses a coarse-to-fine pipeline: a coarse stage matches pre-annotated key components against the final screenshot, and a fine stage invokes an MLLM when the match is ambiguous. We use only the LLM-based fine stage, as the coarse stage requires task-specific key component annotations that are only available for SPA-Bench’s own tasks. The judge receives all screenshots with annotated action markers (red dots at tap locations, scroll indicators) overlaid on the images, together with detailed evaluation guidelines.

A3 judge (Chai et al., 2025). A3 provides two evaluation modes. *Final-state evaluation* presents only the last screenshot and its UI XML tree to the LLM, asking whether the task goal is achieved. *Essential-states evaluation* first decomposes the task instruction into a set of necessary sub-goals (essential states) via an LLM call, then evaluates whether each essential state is achieved by examining screenshots within a sliding window. We evaluate both modes; essential states are generated by the LLM rather than manually defined.

AndroidArena judge (Xing et al., 2024). This method formats the trajectory as a text-based chronological sequence of actions and structured UI element observations (extracted from accessibil-

ity trees). The formatted trajectory, together with the task instruction, is passed to an LLM for a direct success/failure judgment. The original method operates purely on text; we add a screenshot-based fallback for trajectories that lack text observations. This is the only method that primarily operates on text rather than images.

AgentRewardBench judge (Lù et al., 2025). Originally designed for web agent evaluation, this method evaluates trajectories along 4 dimensions via structured questions: (1) task success, (2) side effects, (3) action optimality, and (4) action looping. We adapt it for mobile agents by providing the last screenshot, the UI element list, and a step-by-step trajectory summary. The judge produces a reasoning trace followed by structured answers for each question; we use the task success answer as the final prediction.

4.2 Simple baseline judge

Existing methods entangle multiple design choices, making it difficult to isolate the effect of any single factor. We design a simple baseline judge with independently configurable components to enable controlled ablation. The judge receives the task instruction followed by a chronological step-by-step trajectory: each step includes the action taken, optionally the agent’s reasoning and visible UI elements, and a screenshot uniformly sampled from the full trajectory. The system prompt provides balanced evaluation guidelines: outcome-focused judgment, mid-trajectory success recognition, corrective action credit, and balanced framing that avoids overly strict conditions which tend to suppress recall. By varying screenshot count (3–192), UI metadata, agent reasoning, and image resolution independently, we isolate the effect of each factor (§5.1). The default configuration is 48 uniformly sampled screenshots at max long edge 600px, without UI metadata or agent reasoning.

5 Experiments

5.1 Judge evaluation

We evaluate all 6 judge methods (5 existing + our simple baseline) across 5 LLM backends: Qwen2.5-VL-72B (Bai et al., 2025), GPT-5-mini (OpenAI, 2025), Gemini 3 Flash (Google DeepMind, 2025), GLM-4.6V (GLM-V Team, 2025), and Claude Sonnet 4.5 (Anthropic, 2025). Table 2 reports accuracy on the full 931-trajectory benchmark.

Judge Method	Qwen	GPT	Gemini	GLM	Claude
AndroidArena	84.6	88.2	87.7	83.9	79.0
A3 (final state)	81.5	76.4	83.8	78.1	78.4
A3 (essential)	80.0	78.1	83.7	81.9	81.3
SPA-Bench	80.1	90.6	89.8	82.2	84.7
AgentRewardBench	84.9	82.8	89.3	<u>86.1</u>	<u>87.2</u>
Simple Baseline [†]	<u>85.7</u>	<u>90.8</u>	90.9	84.9	86.0

Table 2: Judge accuracy (%) across methods and LLM backends (Qwen2.5-VL-72B, GPT-5-mini, Gemini 3 Flash, GLM-4.6V, Claude Sonnet 4.5). Bold marks the best backend per method (row-wise); underline marks the best method per backend (column-wise). Full results in Appendix D. [†]Default configuration: 48 uniformly sampled screenshots at max long edge 600px; no UI metadata or agent reasoning; see ablation below.

Several findings emerge from the results. First, **no single method dominates**: the simple baseline is the strongest method with the Gemini, GPT-5-mini, and Qwen backends (90.9%, 90.8%, 85.7%), while AgentRewardBench is strongest with GLM and Claude (86.1% and 87.2%). Second, **the LLM backend substantially affects every method**: the same method can vary by 5.6–10.5pp across backends (*e.g.*, SPA-Bench ranges from 80.1% with Qwen to 90.6% with GPT). Third, **the simple baseline is competitive**: despite its minimal design, it is within 1.2pp of the strongest method even on the two backends it does not lead, so more elaborate pipelines do not consistently improve quality. A two-way variance decomposition over the 6×5 grid quantifies the two factors: method choice explains 49% of the accuracy variance and the backbone 21%, but the method share is driven by the two weakest methods, both purpose-built (the A3 modes). Excluding them as a sensitivity analysis reverses the shares (11% vs. 49%): among AndroidArena, SPA-Bench, AgentRewardBench, and the baseline, backbone choice dominates (Appendix D.4).

Ablation study. We ablate the simple baseline along two dimensions (Table 3). For *screenshot count vs. resolution*, at approximately equal cost, fewer higher-resolution screenshots outperform many low-resolution ones: 16 screenshots at 1/4 resolution achieves the best accuracy (91.2% for GPT, 86.5% for Qwen), while 192 at 1/64 degrades substantially (86.6%/82.3%). For *input components*, UI tree metadata has negligible impact (≤ 0.3 pp for Qwen), and agent reasoning provides only a modest gain for GPT (~ 1.7 pp). With enough screenshots, additional metadata is unnecessary.

Configuration	Qwen	GPT
<i>Screenshots $\times$ resolution (iso-cost, screenshots only)</i>		
16 screenshots @ 1/4 res	86.5	91.2
48 screenshots @ 1/16 res	85.6	90.6
192 screenshots @ 1/64 res	82.3	86.6
<i>Input components (3 screenshots, original resolution)</i>		
+ UI trees + reasoning	84.1	89.9
+ UI trees only	84.1	86.7
+ reasoning only	83.8	88.8
screenshots only	83.8	87.1

Table 3: Ablation study on simple baseline judge. Top: fewer higher-resolution screenshots outperform many low-resolution ones at equal cost. Bottom: UI metadata has minimal impact; agent reasoning provides a modest gain for GPT only.

5.2 Benchmark metrics predict real-world utility

A benchmark is only useful if its metrics predict how judges perform in practice. We validate MOBILEJUDGE BENCH through two downstream applications: using judges to *evaluate* agents (ranking and success-rate estimation) and using judges as *reward signals* for on-policy RL training.

5.2.1 Judge for evaluation: meta-correlation

For each of the 30 judge variants, we compute per-agent success rates (24 benchmark-model combinations) and compare against human ground truth via *ranking fidelity* (Spearman ρ) and *rate estimation error* (MAE). We then compute a *meta-correlation*: does higher judge quality predict more reliable agent evaluation? Details are in Appendix D.

Table 4 and Figure 2 present the results. F1 is the strongest predictor of ranking fidelity ($\rho_s = 0.90$, 95% CI [0.66, 0.92]), while balanced accuracy best predicts rate estimation ($\rho_s = -0.79$ [−0.92, −0.70]); the CIs for accuracy, F1, balanced accuracy, and recall all exclude zero, with recall the weakest of the four. Strikingly, precision has *no predictive power* for either metric (both of its CIs span zero); what matters is balanced classification, not precision alone. The highest-accuracy judge (Baseline/Gemini, 90.9%) closely tracks human success rates ($\rho = 0.97$, Figure 2c). The intervals are computed with a task-cluster bootstrap and corroborated by leave-one-out checks over backbones and methods, and by a sensitivity analysis of API nondeterminism (Appendix D.4).

5.2.2 Judge for training: on-policy RL

We train UI-TARS-7B-SFT (Qin et al., 2025) on the easy task set of AndroidWorld using GRPO with

Quality Metric	ρ_s w/ Ranking	ρ_s w/ MAE↓
Accuracy	0.89 [0.66, 0.92]	−0.77 [−0.92, −0.68]
Precision	0.00 [−0.23, 0.30]	−0.22 [−0.36, 0.05]
Recall	0.74 [0.43, 0.85]	−0.33 [−0.64, −0.16]
F1	0.90 [0.66, 0.92]	−0.72 [−0.89, −0.60]
Balanced Acc	0.87 [0.65, 0.91]	−0.79 [−0.92, −0.70]

Table 4: Meta-correlation across 30 judge variants. Each cell is the Spearman ρ_s between a judge quality metric and a reliability metric: agent ranking fidelity (ρ_s w/ Ranking) or success-rate estimation error (ρ_s w/ MAE, where negative means higher quality $\rightarrow$ lower error). Brackets give 95% CIs from a task-cluster bootstrap (tasks resampled with replacement within each source benchmark; 2,000 replicates; the 30 judge variants held fixed). Bold marks the strongest predictor per column.

4 reward configurations: the built-in rule-based checker, and our simple baseline judge with GPT-5-mini, GPT-5.2, and Qwen backends. All conditions share identical hyperparameters; only the reward source differs. Evaluation uses the ground-truth checker across 3 seeds (Appendix E).

Figure 3 shows that judge accuracy predicts training outcomes: rule-based (94.6% acc) $\rightarrow$ 54.6% best easy-set success rate, GPT-5-mini (92.2%) $\rightarrow$ 45.4%, GPT-5.2 (88.8%) $\rightarrow$ 42.6%, Qwen (88.8%) $\rightarrow$ 39.9%; the same ordering holds on the complete 116-task suite (36.8%, 30.2%, 27.9%, 26.4%; Appendix E). GPT-5.2 and Qwen have identical accuracy but opposite precision–recall profiles, and the higher-precision GPT-5.2 (precision 93.7% vs. 80.2%) reaches a 2.7pp higher easy-set success rate, consistent with false positives directly rewarding incorrect behavior (Huang et al., 2024). This comparison depends on the best-checkpoint convention: at the fixed final checkpoint the two conditions tie on the full task set (Appendix E). We therefore regard it as suggestive within our study rather than conclusive.

5.3 Failure analysis

Beyond aggregate metrics, we analyze cases where judges systematically fail, to understand *why* they fail and whether failure patterns depend on the LLM backbone.

5.3.1 Method

For each of the 5 judge methods adapted from existing benchmarks (this analysis excludes our simple baseline), we identify trajectories where the judge prediction disagrees with human ground truth. We run this analysis separately for two LLM back-

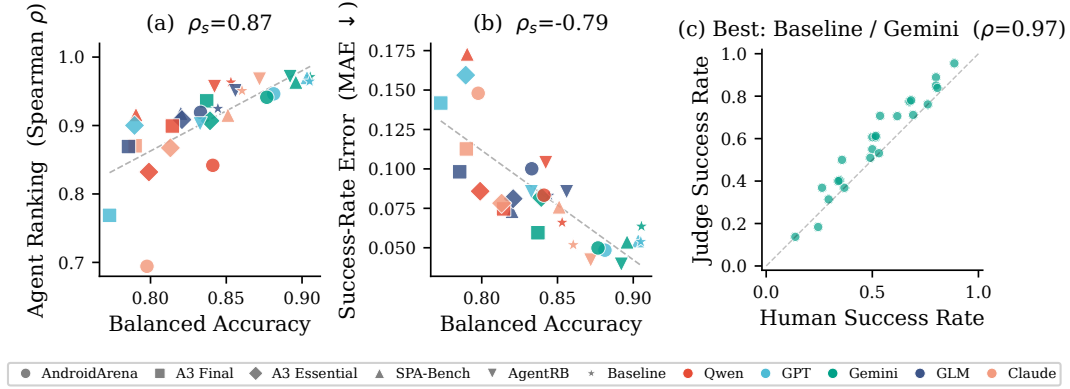


Figure 2: Judge quality predicts evaluation reliability (30 method $\times$ backend variants; marker shape=judge method, color=LLM backend). (a) Balanced accuracy correlates with agent ranking fidelity ($\rho_s=0.87$). (b) Balanced accuracy inversely correlates with success-rate estimation error. (c) The highest-accuracy judge (Baseline/Gemini, 90.9%) closely tracks human success rates across 24 agents ($\rho=0.97$).

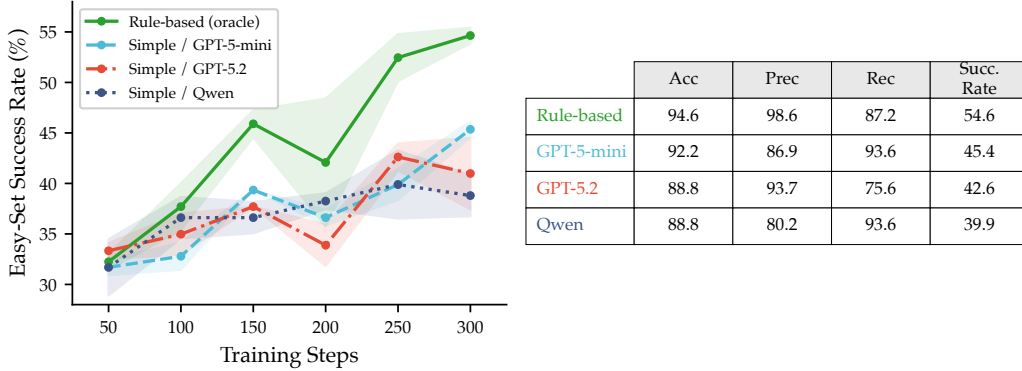


Figure 3: On-policy RL training curves with different judge reward signals (mean $\pm$ std over 3 seeds). Inset table: judge quality on the AndroidWorld subset (Acc/Prec/Rec %) and best easy-set success rate. Training performance tracks judge accuracy (rule-based > GPT-5-mini > {GPT-5.2, Qwen}); the equal-accuracy pair is ordered by precision under the best-checkpoint convention.

ends, GPT-5-mini and Qwen2.5-VL-72B, to compare failure profiles. We define *hard-core failures* as cases where at least 4 of the 5 methods err, indicating failures that are robust to judge design choices and likely reflect fundamental limitations. For each hard-core failure, we examine the task goal, ground-truth label, trajectory screenshots, and the reasoning produced by all 5 judges, then categorize the root cause into a taxonomy of error types (detailed in Appendix F).

5.3.2 Opposite failure profiles across backends

The failure profiles reveal a striking asymmetry. GPT-based judges produce 48 hard-core failures dominated by *false negatives* (30 FN, 18 FP): they are too conservative, failing to recognize successful trajectories. Qwen-based judges produce 78 hard-core failures dominated by *false positives* (71 FP, 7 FN): they are too permissive, accepting failed trajectories as successes. Only 22 cases appear in

both sets (17 false positives and 5 false negatives shared by both backends), suggesting that these represent genuinely ambiguous cases rather than backend-specific artifacts.

5.3.3 Root cause taxonomy

Figure 4 shows the root cause categories for each backend set, revealing qualitatively different failure modes. Representative examples for each category are in Appendix F.1 (Figures 8–9).

To assess the reliability of this classification, two raters independently categorized a random sample of 30 hard-core failure cases into the seven categories, agreeing on 29 of 30 (Cohen’s $\kappa = 0.957$; Appendix F.4).

GPT set (FN-dominated). The top failure mode is *last-frame anchoring* (12 cases): judges overly on the final screenshot and miss mid-trajectory completion evidence. *Unfamiliar success state* (9 cases) and *invisible state change* (7 cases) follow: judges either do not recognize valid success pat-

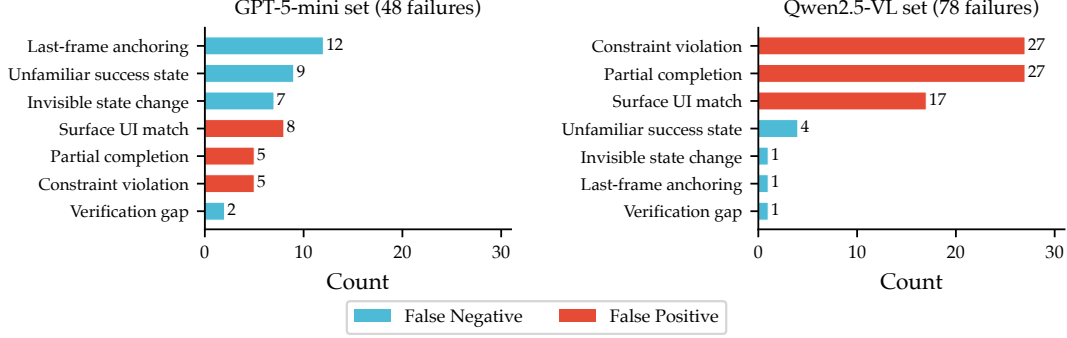


Figure 4: Root cause taxonomy for hard-core failures. The GPT set is dominated by false negatives (last-frame anchoring, unfamiliar success state), while the Qwen set is dominated by false positives (constraint violation, partial completion). Surface UI match is a shared weakness across both backends.

terns or miss changes not visually evident in screenshots. Among FPs, *surface UI match* (8 cases) leads: judges conclude success from superficial visual similarity without verifying the actual answer.

Qwen set (FP-dominated). *Constraint violation* and *partial completion* (27 cases each) dominate: judges miss task-specific constraints (*e.g.*, “nearest” parking lot) or accept incomplete results as success. *Surface UI match* (17 cases) is the third category. Representative examples are in Appendix F.1.

6 Discussion

Judges should be evaluated, not assumed reliable. Judge accuracy ranges from 76% to 91% depending on method and backend, and this variation shifts agent rankings by up to 13 positions and meaningfully affects training outcomes. Benchmarks should report judge reliability alongside agent performance.

Elaborate judge pipelines do not consistently outperform a simple baseline. Method choice, backbone choice, and their interaction all affect accuracy, but the added complexity of purpose-built methods is not reliably rewarded: the simple baseline is competitive with or better than every purpose-built method on every backend, and the weakest methods are purpose-built. Once those are excluded, backbone choice explains most of the remaining variance (Appendix D.4); among competitive methods, upgrading the backbone yields greater returns than engineering elaborate judge prompts.

Different applications demand different judge profiles. For *evaluation*, F1 and balanced accuracy best predict reliability; precision alone has no predictive power. For *training*, our matched-accuracy comparison suggests that false positives,

which directly corrupt the reward signal (Huang et al., 2024), are the more damaging error. These results suggest that balanced judges may be more suitable for leaderboards, while judges with strong false-positive control may be preferable as reward signals.

Failure modes are structurally addressable.

Last-frame anchoring can be mitigated by providing more screenshots, consistent with our ablation. *Surface UI match*, a shared weakness, motivates proactive verification approaches (Cui et al., 2026; Dai et al., 2025) where judges interact with the environment.

7 Conclusion

We introduced MOBILEJUDGE BENCH, a benchmark for evaluating LLM-as-judge methods on mobile agent trajectories. Through 931 human-annotated trajectories spanning 6 benchmarks, 4 agents, and 68 apps, we systematically evaluated 6 judge methods across 5 LLM backends. Our key findings are: (1) a simple baseline judge with sampled screenshots is competitive with, and often exceeds, purpose-built methods, indicating that more elaborate judge pipelines do not consistently improve judge quality; (2) benchmark quality metrics, particularly F1 and balanced accuracy, predict agent ranking reliability, and in our training study the judge with stronger false-positive control reached a higher best-checkpoint success rate; and (3) failure analysis reveals that different LLM backends produce qualitatively opposite failure profiles, with surface UI match as a shared weakness. We hope that MOBILEJUDGE BENCH encourages the community to treat judge evaluation as a first-class concern and provides a foundation for developing more reliable evaluation methods for mobile agents.

Limitations

Our benchmark evaluates binary success/failure judgments; finer-grained dimensions such as trajectory optimality, efficiency, and partial progress (Lù et al., 2025) are left to future work. Binary success is, however, the signal that the built-in checkers of current mobile-agent benchmarks produce and that mobile RL pipelines consume; it is therefore the natural first target for a judge benchmark.

Our training experiments are limited to the AndroidWorld environment and to a single judge method; the four reward sources are three LLM backbones of that judge plus a rule-based oracle. Training-time comparisons across judge methods are therefore narrower than the six-method offline study in §5.1.

Finally, the root-cause classification in our failure analysis involves subjective judgment in borderline cases, although a double-coded sample of 30 cases shows high inter-rater agreement (Appendix F.4).

Ethical Considerations

Judge unreliability is itself the primary risk this work concerns. LLM judges misclassify 9–24% of mobile-agent trajectories (§5.1); treating their outputs as ground truth can distort leaderboards (§5.2.1) and, when judges serve as reward signals, train agents on corrupted feedback (§5.2.2). MOBILEJUDGE BENCH is intended to mitigate this risk by quantifying it: we recommend reporting judge reliability alongside agent results and controlling false positives when judges are used as rewards.

Documenting systematic judge weaknesses (e.g., surface UI match, §5.3) could in principle be used to game LLM judges. We consider the transparency benefit to outweigh this risk: identifying failure modes is a prerequisite for building robust, state-verifying judges, which our analysis motivates. Relatedly, agents trained against permissive judges may appear successful while violating task constraints; our false-positive analysis (§5.2.2) is aimed at preventing exactly this outcome in practice.

All trajectories are collected in emulated Android environments; no real user data is involved and no agent action affects real services (Appendix A).

References

- Anthropic. 2025. Claude sonnet 4.5 system card. <https://www.anthropic.com/claude-sonnet-4-5-system-card>.
- 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. *Advances in Neural Information Processing Systems*, 37:12461–12495.
- Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. *Qwen2.5-v1 technical report*. *Preprint*, arXiv:2502.13923.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.
- Yuxiang Chai, Hanhao Li, Jiayu Zhang, Liang Liu, Guozhi Wang, Shuai Ren, Siyuan Huang, and Hongsheng Li. 2025. *A3: Android agent arena for mobile gui agents*. *Preprint*, arXiv:2501.01149.
- Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye HAO, Jun Wang, and Kun Shao. 2025. Spa-bench: A comprehensive benchmark for smartphone agent evaluation. In *The Thirteenth International Conference on Learning Representations*.
- Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. 2026. Agentic reward modeling: Verifying GUI agent via progressive trajectory-grounded interaction. *arXiv preprint arXiv:2602.00575*.
- Gaole Dai, Shiqi Jiang, Ting Cao, Yuqing Yang, Yuanchun Li, Rui Tan, Mo Li, and Lili Qiu. 2025. Prore: A proactive reward system for gui agents via reasoner-actor collaboration. *arXiv preprint arXiv:2509.21823*.
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.
- Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR.

- GLM-V Team. 2025. GLM-4.5V and GLM-4.1V-Thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*.
- Google DeepMind. 2025. Gemini 3 flash model card. <https://deepmind.google/models/model-cards/gemini-3-flash>.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.
- Sukai Huang, Shu-Wei Liu, Nir Lipovetzky, and Trevor Cohn. 2024. The dark side of rich rewards: Understanding and mitigating noise in vlm rewards. *arXiv preprint arXiv:2409.15922*.
- Jaehun Jung, Faeze Brahman, and Yejin Choi. 2024. Trust or escalate: Llm judges with provable guarantees for human agreement. *arXiv preprint arXiv:2407.18370*.
- Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.
- Nathan Lambert, Valentina Pyatkin, Jacob Morrison, Lester James Validad Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. 2025. Rewardbench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1755–1797.
- Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. 2024. Benchmarking mobile device control agents across diverse configurations. *arXiv preprint arXiv:2404.16660*.
- Dawei Li, Renliang Sun, Yue Huang, Ming Zhong, Bohan Jiang, Jiawei Han, Xiangliang Zhang, Wei Wang, and Huan Liu. 2025. Preference leakage: A contamination problem in llm-as-a-judge. *arXiv preprint arXiv:2502.01534*.
- Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024a. Llm-as-judges: a comprehensive survey on llm-based evaluation methods. *arXiv preprint arXiv:2412.05579*.
- Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyi Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024b. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154.
- Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J Pal, and Siva Reddy. 2025. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*.
- Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. 2024. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*.
- Shikhar Murty, Hao Zhu, Dzmitry Bahdanau, and Christopher D Manning. 2024. Nnetnav: Unsupervised learning of browser agents through environment interaction in the wild. *arXiv preprint arXiv:2410.02907*.
- OpenAI. 2025. GPT-5 system card. <https://openai.com/index/gpt-5-system-card/>.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.
- Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*.
- Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxia Dong. 2025. [WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning](#). In *The Thirteenth International Conference on Learning Representations*.
- Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.
- Christopher Rawles, Sarah Clinckemaele, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyi Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry,

- Divya Tyamagundlu, Timothy P Lillicrap, and Oriana Riva. 2025. [Androidworld: A dynamic benchmarking environment for autonomous agents](#). In *The Thirteenth International Conference on Learning Representations*.
- Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728.
- Guangming Sheng, Chi Zhang, Zilinfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*.
- Wentao Shi, Yu Wang, Yuyang Zhao, Yuxin Chen, Fuli Feng, Xueyuan Hao, Xi Su, Qi Gu, Hui Su, Xunliang Cai, and Xiangnan He. 2026. AJ-Bench: Benchmarking agent-as-a-judge for environment-aware evaluation. *arXiv preprint arXiv:2604.18240*.
- Yucheng Shi, Wenhao Yu, Zaitang Li, Yonglin Wang, Hongming Zhang, Ninghao Liu, Haitao Mi, and Dong Yu. 2025. Mobilegui-rl: Advancing mobile gui agent through reinforcement learning in online environment. *arXiv preprint arXiv:2507.05720*.
- Yifan Sui, Xin Huang, Hongbing Li, Fang Xu, Jiahe Lv, Haolong Yan, Yeqing Shen, Litao Liu, Zhimin Fan, Ziyang Meng, Jia Wang, Junbo Qi, Kaijun Tan, Zheng Ge, Xiangyu Zhang, Daxin Jiang, and Osamu Yoshie. 2026. AndroidDaily: A verifiable benchmark for mobile GUI agents on real-world closed-source applications. *arXiv preprint arXiv:2605.27761*.
- Jiahui Sun, Zhichao Hua, and Yubin Xia. 2025. Autoeval: A practical framework for autonomous evaluation of mobile agents. *arXiv preprint arXiv:2503.02403*.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.
- Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 2024. Understanding the weakness of large language model agents within a complex android environment. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6061–6072.
- Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. 2025a. Mobilerl: Online agentic reinforcement learning for mobile gui agents. *arXiv preprint arXiv:2509.18119*.
- Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2025b. Androidlab: Training and systematic benchmarking of android autonomous agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2166.
- Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.
- Yaowei Zheng, Juntong Lu, Shenzhi Wang, Zhangchi Feng, Dongdong Kuang, Yuwen Xiong, and Rishong Zhang. 2025. EasyR1: An efficient, scalable, multi-modality RL training framework. <https://github.com/hiyouga/EasyR1>.
- Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. [Webarena: A realistic web environment for building autonomous agents](#). In *The Twelfth International Conference on Learning Representations*.
- Lianghui Zhu, Xinggang Wang, and Xinlong Wang. 2023. Judgelm: Fine-tuned large language models are scalable judges. *arXiv preprint arXiv:2310.17631*.
- Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, et al. 2024. Agent-as-a-judge: Evaluate agents with agents. *arXiv preprint arXiv:2410.10934*.

A Benchmark details

Unified environment. We build a unified execution environment on top of the AndroidWorld codebase (Rawles et al., 2025), integrating apps and initialization logic from all 6 benchmarks. AndroidWorld tasks use a rooted system image without Google Play Store; other benchmarks use a separate system image with Google Play Store access (no root) and device backup/restore for task initialization. This reduces the maintenance burden of operating 6 separate codebases while preserving each benchmark’s original task semantics.

Licenses. Of the assets we build on, the AndroidWorld codebase and B-MoCA are released under the Apache-2.0 license, and SPA-Bench, A3, and AndroidLab under the MIT license, all permitting research use and redistribution. The AndroidArena repository specifies no license, and the AgentRewardBench code and dataset carry no standard license (the dataset is distributed under custom research terms of use, and we use no data from it): from AndroidArena we use task definitions, and from AgentRewardBench the judge-prompt design published in its paper, both for research evaluation with attribution. The trajectories, annotations, and evaluation code we release are our own artifacts.

Data provenance and privacy. All trajectories are collected on emulated Android devices; no real user accounts or personal data appear in screenshots, UI trees, actions, or agent reasoning. Screen content is routine app UI from the benchmarks’ controlled environments and contains no offensive material.

Annotators. The 9 annotators (§3.2) are graduate students at the authors’ institution, all proficient in English and experienced Android users. Annotation was conducted voluntarily as part of their funded research work, with the research use of the labels made clear; no additional task-specific payment was made. The study annotates machine-generated trajectories in emulated environments and involves no human-subjects data.

Annotation platform. We build a custom Streamlit-based annotation tool for trajectory-level labeling (Figure 5). The tool supports two modes: *batch mode*, where annotators navigate through benchmark→agent→trajectory hierarchies with automatic filtering of already-annotated items, and *single mode* for ad-hoc trajectory inspection. For

each trajectory, the platform provides: (1) a *trajectory video* synthesized from the screenshot sequence with configurable resolution, font size, and playback speed, enabling rapid overview of the full trajectory; (2) a *step viewer* showing before/after screenshots, the agent’s chain-of-thought reasoning, and the executed action for each step; and (3) an *annotation form* collecting labels for task achievability, success, trajectory optimality, side effects, repetition cycles, and free-text notes. Annotations are exported as structured JSON files mirroring the source directory hierarchy.

B Comparison with AgentRewardBench

AgentRewardBench (Lù et al., 2025) is the closest existing benchmark to ours: both evaluate LLM judges of agent trajectories against human annotations. The two benchmarks are complementary, differing in agent domain and in the validation questions they target. Table 5 provides a structured comparison; all AgentRewardBench entries are taken from Lù et al. (2025).

C Judge method details

The four existing judge methods (SPA-Bench, A3, AndroidArena, AgentRewardBench) use prompts adapted from their respective original papers. Below we present the prompt for our simple baseline judge, which is the only method we design from scratch.

Decoding and backend versions. All judges decode greedily (temperature 0), except the A3 judges, which follow their original implementation (temperature 0.7), and the GPT-5 family, whose API fixes sampling at the model default. Backend version strings: gpt-5-mini and gpt-5.2 via the OpenAI API; google/gemini-3-flash-preview, z-ai/glm-4.6v, and anthropic/claude-sonnet-4.5 via OpenRouter; Qwen2.5-VL-72B-Instruct served from a local vLLM endpoint.

C.1 Simple baseline judge prompt

System prompt:

```
You are an expert evaluator for
mobile agent task completion.
Your role is to determine
whether a given task has been
successfully completed based
on the provided trajectory
evidence.
```

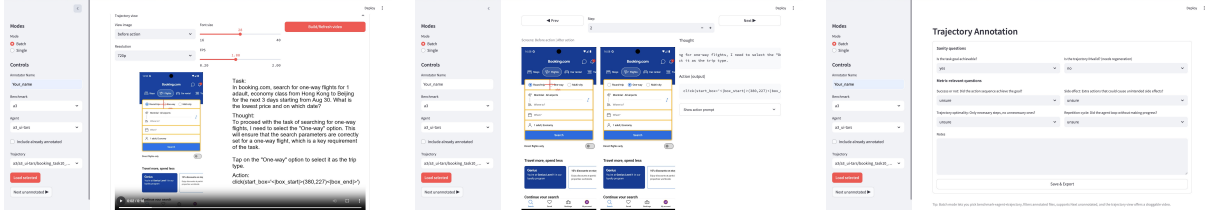


Figure 5: Annotation platform. Left: trajectory video view with task description, agent reasoning, and playback controls. Center: step-by-step viewer showing before/after screenshots and action details. Right: annotation form collecting task achievability, success, optimality, side effects, repetition cycles, and free-text notes.

Evaluation Guidelines:

1. **Outcome-Focused:** Judge based on whether the primary objective was achieved, not the path taken.
2. **Use All Provided Evidence:** Base your judgment on all available information -- screenshots, actions, agent reasoning, and UI element lists. Do not assume information beyond what is provided.
3. **Mid-Trajectory Success:** The task may be completed in an intermediate step rather than the final one. Additional actions after completion do not invalidate success, unless they explicitly undo it.
4. **Corrective Actions:** If the agent made mistakes but corrected them and achieved the goal, that counts as success.
5. **UI State Indicators:** Pay attention to visual indicators like selected tabs, checkboxes, highlighted items, and confirmation messages.

Be balanced in your judgment -- avoid being overly strict (missing true successes) or overly lenient (accepting failures).

User prompt header:

```
## Task
{task}

## Agent Trajectory
The following shows the
agent's execution trajectory
in chronological order.
Each step's screenshot is
placed immediately after its
description.
```

Step template (with all optional components enabled):

```
### Step {step_num}
Action: {action}
Agent Reasoning: {reasoning}
UI Elements: {ui_elements}
[Screenshot attached]
```

User prompt footer:

```
## Your Judgment
Based on the trajectory above,
determine if the task was
successfully completed.
```

```
Reply in the following format:
Result: <1 for success, 0 for failure>
Confidence: <high/medium/low>
Reason: <brief explanation in 1-2 sentences>
```

The step template varies depending on ablation configuration: agent reasoning and UI elements are independently toggled on/off. Screenshots are uniformly sampled from the trajectory (always including the first and last frames) and resized to a configurable resolution before encoding.

C.2 Uniform vs. event-based screenshot sampling

A possible concern with uniform screenshot sampling is that it may miss transient evidence (*e.g.*, a momentary confirmation toast). We therefore implemented an event-based sampler that prioritizes frames associated with informative actions (text entry, app switches, navigation, trajectory-terminating actions, and transitions between action types) and compared it with uniform sampling using the simple baseline judge (GPT-5-mini backbone) with all other settings identical. At the operating budget of 48 frames, only 1.3% of trajectories (12/931) exceed the budget, so the two strategies select identical frames for the remaining 98.7% by construction; on the 12 differing trajectories, the event-based sampler judges 11 correctly and the uniform sampler 10. Because so few trajectories are subsampled at 48 frames, we also ran a

	AgentRewardBench (Lù et al., 2025)	MOBILEJUDGE BENCH (ours)
Domain	Web browsing: WebArena, VisualWebArena, AssistantBench, WorkArena, WorkArena++	Mobile (Android): SPA-Bench, AndroidWorld, A3, AndroidArena, B-MoCA, AndroidLab; 68 real apps
Trajectory observations	Browser screenshots, DOM, and accessibility tree; judges evaluate the final browser state (screenshot or accessibility tree), for some judges together with the agent’s thought and action sequence	Per-step (screenshot, action, UI tree, agent reasoning) tuples; judge methods span full-sequence and final-state designs, image and text inputs
Judges evaluated	3 judge designs (AER, NNetNav, and a simplified judge), instantiated as 12 judges over 4 LLM backbones	6 judge methods (adapted from SPA-Bench, A3 with two modes, AndroidArena, and AgentRewardBench, plus a simple baseline), fully crossed with 5 LLM backbones: 30 judges
Human annotation	Each trajectory reviewed by an expert (success, side effects, repetitiveness); 89.3% success agreement measured on one agent–benchmark subset	Each trajectory independently labeled by 2–4 of 9 annotators; 88.4% pairwise success agreement; disagreements resolved by discussion
Judge validation	Per-trajectory precision, recall, and F1 against expert labels; precision emphasized as the primary metric, motivated by rejection fine-tuning and reward modeling; expert-vs-automatic success-rate comparison across agents	Per-trajectory accuracy, balanced accuracy, precision, recall, F1; meta-level validation linking judge metrics to agent-ranking fidelity and success-rate estimation error over 24 agents, with task-cluster bootstrap CIs
Downstream RL	Motivates judge use for fine-tuning and RL rewards; no training experiment	On-policy GRPO training with judge-based rewards (4 reward configurations, 3 seeds), linking judge metrics to training outcomes
Failure analysis	Four judge-error categories with examples	Seven-category root-cause taxonomy of hard-core failures, contrasted across two backbones; double-coded sample: $\kappa = 0.957$

Table 5: Structured comparison of AgentRewardBench (Lù et al., 2025) and MOBILEJUDGE BENCH. The two benchmarks cover different agent domains and are complementary; MOBILEJUDGE BENCH additionally validates judge metrics against downstream evaluation and training utility.

stress test at a tight 8-frame budget, where 61.9% of trajectories are subsampled. Accuracy is 89.6% (uniform) vs. 89.3% (event-based); the two samplers disagree on 6.1% of trajectories (57/931), and the disagreements split nearly evenly (30 uniform-correct vs. 27 event-correct). Uniform sampling therefore does not appear to limit judge quality at the trajectory lengths and input budgets we evaluate.

D Full evaluation results

D.1 Evaluation metric definitions

Given a set of trajectories with human ground-truth labels and judge predictions, we compute the following metrics.

Trajectory-level classification. Let TP, FP, TN, FN denote the counts of true positives, false positives, true negatives, and false negatives, where a “positive” is a successful trajectory. We compute:

- Accuracy = $(TP + TN) / (TP + TN + FP + FN)$
- Precision = $TP / (TP + FP)$

- Recall = $TP / (TP + FN)$
- F1 = $2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall})$
- Balanced Acc = $\frac{1}{2} \left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP} \right)$

Precision measures how often the judge’s success predictions are correct; recall measures how often truly successful trajectories are identified. Our dataset is approximately balanced (53% positive), so accuracy and balanced accuracy are close, but we report both for completeness.

Agent-level reliability. For each agent a (defined as a benchmark–model pair, e.g., android_world/gpt-5-mini), we compute the success rate under human labels (h_a) and under judge predictions (j_a). We then measure reliability across $N=24$ agents via two complementary metrics:

- **Ranking fidelity:** Spearman rank correlation ρ_s between the vectors $(h_1, \dots, h_N)$ and $(j_1, \dots, j_N)$. High ρ_s means the judge preserves the relative ordering of agents.

- **Rate estimation error:** Mean absolute error $MAE = \frac{1}{N} \sum_a |h_a - j_a|$. Low MAE means the judge accurately estimates absolute success rates per agent.

Meta-correlation. To assess whether benchmark quality predicts downstream evaluation reliability, we compute Spearman ρ_s between a quality metric vector (one value per judge variant, *e.g.*, accuracy) and a reliability metric vector (one value per judge variant, *e.g.*, agent-level ρ_s), across all 30 judge variants.

D.2 Agent-level data points

Table 6 lists the 24 agents (benchmark–model combinations) used for computing ranking fidelity and rate estimation error. SPA-Bench single-app and cross-app trajectories are merged into one agent per model.

Agent	Model	Traj.	Rate
arena / gpt-5-mini	gpt-5-mini	44	0.886
bmoca / gpt-5-mini	gpt-5-mini	31	0.806
a3 / gpt-5-mini	gpt-5-mini	45	0.800
android-lab / gpt-5-mini	gpt-5-mini	20	0.800
spa / gpt-5-mini	gpt-5-mini	67	0.761
android_world / gpt-5-mini	gpt-5-mini	52	0.692
arena / ui-tars	ui-tars	41	0.683
a3 / ui-tars	ui-tars	40	0.675
bmoca / ui-tars	ui-tars	31	0.516
android-lab / ui-tars	ui-tars	20	0.500
spa / ui-tars	ui-tars	53	0.491
android_world / ui-tars	ui-tars	50	0.340
arena / llama	llama	34	0.618
bmoca / llama	llama	28	0.500
a3 / llama	llama	28	0.357
android_world / llama	llama	52	0.346
spa / llama	llama	51	0.294
android-lab / llama	llama	19	0.263
a3 / qwen	qwen	41	0.537
arena / qwen	qwen	32	0.531
bmoca / qwen	qwen	33	0.515
android-lab / qwen	qwen	19	0.368
spa / qwen	qwen	49	0.245
android_world / qwen	qwen	51	0.137

Table 6: The 24 agent-level data points used for computing ranking fidelity and rate estimation error. Rate is the human-annotated success rate. Agents are grouped by model and sorted by success rate within each group.

D.3 Full classification results

Table 7 presents the complete evaluation results for all 30 judge variants, including precision, recall, F1, balanced accuracy, and agent-level reliability metrics.

D.4 Uncertainty quantification and variance decomposition

Task-cluster bootstrap. The 95% CIs in Table 4 treat the 30 judge variants as fixed and quantify uncertainty from the benchmark sample. Each replicate resamples tasks with replacement within each of the 6 source benchmarks, where a sampled task carries all of its 1–4 agent trajectories (289 tasks and 931 trajectories in the original sample); we then recompute every judge’s quality metrics, the per-agent reliability metrics, and the cross-judge meta-correlation (2,000 replicates; percentile intervals). Figure 6 visualizes the intervals. The conclusions are robust to leaving out any single backbone or judge method: F1 vs. ranking fidelity stays within $[0.87, 0.95]$ across all leave-one-out configurations, and precision remains non-predictive ($\rho_s \in [-0.15, 0.13]$). They are also robust to API nondeterminism. We ran each backbone (with the simple baseline judge) three times at temperature 0 on a stratified 100-trajectory subset, propagated the measured per-backbone standard deviation (0–4pp) into each judge’s quality scores as Gaussian perturbations, and recomputed the meta-correlation over 1,000 iterations. This analysis addresses provider-side nondeterminism and is distinct from the sampling uncertainty quantified by the bootstrap.

Method vs. backbone variance decomposition. Table 8 decomposes the total sum of squares of the 6×5 method $\times$ backbone quality grid into the method main effect, the backbone main effect, and the residual (interaction), with task-cluster bootstrap CIs. On the full grid the method share dominates, but it is driven by the two A3 modes, the weakest methods. Excluding these two (leaving AndroidArena, SPA-Bench, AgentRewardBench, and the baseline) reverses the accuracy shares to 0.11 (method) vs. 0.49 (backbone). Because this subset is selected by performance, we report the exclusion as a sensitivity analysis rather than the primary result.

E Training experiment details

Base model and algorithm. We use UI-TARS-7B-SFT as the base policy model and train with GRPO (Group Relative Policy Optimization) on $2 \times H100$ 80GB GPUs, using the EasyR1 framework (Zheng et al., 2025) built on verl (Sheng et al., 2024). Training runs in 16 parallel Docker containers, each executing one AndroidWorld task instance at a time.

Judge Method	Backend	Acc	Prec	Rec	F1	BAcc	Ag. ρ	Ag.MAE
Baseline	Gemini	90.9	87.6	96.3	91.8	90.5	0.97	0.064
Baseline	GPT	90.8	88.3	95.1	91.6	90.5	0.96	0.054
SPA-Bench	GPT	90.6	87.8	95.5	91.5	90.3	0.97	0.054
SPA-Bench	Gemini	89.8	88.5	92.9	90.7	89.6	0.96	0.054
AgentRewardBench	Gemini	89.3	88.6	91.7	90.1	89.2	0.97	0.040
AndroidArena	GPT	88.2	89.0	88.6	88.8	88.1	0.95	0.049
AndroidArena	Gemini	87.7	88.4	88.4	88.4	87.7	0.94	0.050
AgentRewardBench	Claude	87.2	88.3	87.4	87.8	87.2	0.97	0.042
AgentRewardBench	GLM	86.1	82.0	94.5	87.8	85.6	0.95	0.086
Baseline	Claude	86.0	87.4	86.0	86.7	86.0	0.95	0.052
Baseline	Qwen	85.7	82.6	92.5	87.2	85.3	0.96	0.066
Baseline	GLM	84.9	81.0	93.3	86.7	84.5	0.92	0.083
AgentRewardBench	Qwen	84.9	79.9	95.3	86.9	84.2	0.96	0.104
SPA-Bench	Claude	84.7	91.5	78.5	84.5	85.1	0.92	0.076
AndroidArena	Qwen	84.6	81.8	91.3	86.3	84.1	0.84	0.083
AndroidArena	GLM	83.9	79.7	93.5	86.1	83.3	0.92	0.100
A3 (final)	Gemini	83.8	84.5	84.7	84.6	83.7	0.94	0.059
A3 (essential)	Gemini	83.7	88.5	79.5	83.7	83.9	0.91	0.082
AgentRewardBench	GPT	82.8	90.7	75.2	82.2	83.3	0.90	0.086
SPA-Bench	GLM	82.2	82.0	85.0	83.4	82.0	0.92	0.073
A3 (essential)	GLM	81.9	85.2	79.7	82.3	82.1	0.91	0.081
A3 (final)	Qwen	81.5	82.3	82.9	82.6	81.4	0.90	0.075
A3 (essential)	Claude	81.3	83.0	81.3	82.1	81.3	0.87	0.078
SPA-Bench	Qwen	80.1	73.3	98.2	83.9	79.0	0.92	0.173
A3 (essential)	Qwen	80.0	80.5	82.1	81.3	79.9	0.83	0.086
AndroidArena	Claude	79.0	91.1	66.9	77.1	79.8	0.69	0.148
A3 (final)	Claude	78.4	87.8	68.7	77.1	79.0	0.87	0.113
A3 (final)	GLM	78.1	85.5	70.5	77.3	78.5	0.87	0.098
A3 (essential)	GPT	78.1	92.3	63.8	75.5	78.9	0.90	0.159
A3 (final)	GPT	76.4	90.5	61.9	73.5	77.3	0.77	0.142

Table 7: Full evaluation results for all 30 judge variants, sorted by accuracy. Acc/Prec/Rec/F1/BAcc are trajectory-level percentages. Ag. ρ is the agent-level Spearman rank correlation with human rankings. Ag.MAE is the mean absolute error of per-agent success rate estimates.

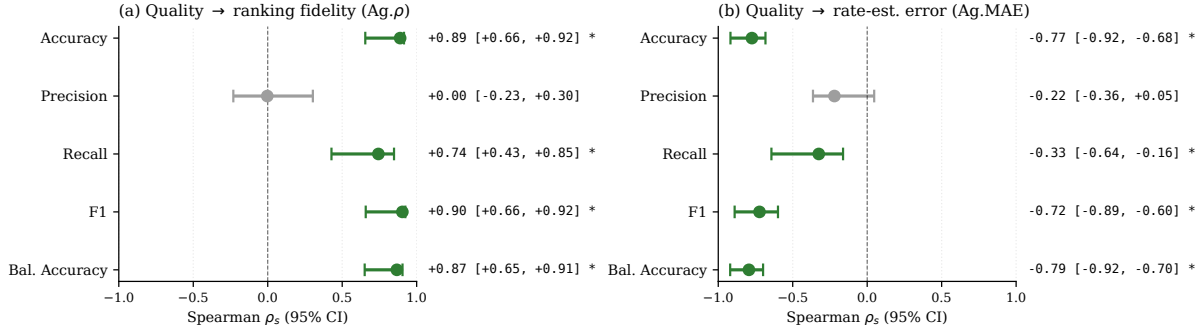


Figure 6: Task-cluster bootstrap 95% CIs for the meta-correlation across the 30 judge variants. Green: interval excludes zero; grey: interval spans zero. Precision is non-predictive for both reliability metrics.

Training data. The training set is the easy task subset from AndroidWorld: 553 task instances across 38 parameterizable task templates and 14 apps. Each template is instantiated with different random seeds (2–16 seeds per template). The remaining 23 easy templates that produce identical instances regardless of seed are excluded from training but included in evaluation.

Hyperparameters. All conditions share identical hyperparameters (Table 9); only the reward source differs. Each training step processes 2 task instances with 8 rollouts each; the resulting 16 rollouts form one update batch. One episode iterates through all 553 instances (276 steps). Training pro-

ceeds for up to 300 steps with checkpoints every 50 steps.

Compute. Each training run takes roughly two days on the 2×H100 setup (≈ 100 GPU-hours per condition, ≈ 400 GPU-hours across the four conditions), excluding checkpoint evaluation. On the judge-evaluation side, the 6×5 grid of Table 2 comprises $\approx 28k$ trajectory-level judge evaluations, and our full logs across ablations and configuration sweeps total $\approx 57k$ evaluations, each involving one or more LLM calls.

Reward conditions. The rule-based condition uses AndroidWorld’s built-in Python verification scripts that programmatically check task comple-

Metric	Method	Backbone	Residual
Accuracy	0.49 [0.37, 0.59]	0.21 [0.13, 0.30]	0.30 [0.25, 0.39]
F1	0.52 [0.43, 0.60]	0.16 [0.11, 0.22]	0.32 [0.27, 0.40]
Bal. Acc	0.46 [0.33, 0.56]	0.24 [0.15, 0.34]	0.30 [0.24, 0.39]
<i>Competitive 4×5 subgrid (excluding the two A3 modes)</i>			
Accuracy	0.11 [0.04, 0.20]	0.49 [0.34, 0.62]	0.40 [0.29, 0.54]
F1	0.12 [0.06, 0.21]	0.38 [0.26, 0.51]	0.49 [0.38, 0.61]
Bal. Acc	0.10 [0.04, 0.19]	0.54 [0.39, 0.66]	0.37 [0.26, 0.50]

Table 8: Share of total variance (η^2) across the 6×5 judge grid attributable to the method and backbone main effects and the residual (interaction), with task-cluster bootstrap 95% CIs. The lower block repeats the decomposition on the competitive 4×5 subgrid that excludes the two A3 modes; the same task draws are used for both grids.

Parameter	Value
Base model	UI-TARS-7B-SFT
RL algorithm	GRPO
Optimizer	AdamW (bf16)
Learning rate	1×10^{-6}
LR warmup	5% of steps (linear)
KL coefficient	0.05
Update epochs	1
Clip ratio	[0.2, 0.3] (asymmetric)
Max gradient norm	1.0
Rollouts per task	8
Tasks per step	2
Max agent steps	20
Sampling temperature	1.0 (train), 0 (eval)

Table 9: Training hyperparameters for on-policy RL experiments.

tion by inspecting device state. The LLM judge conditions use our simple baseline judge (48 uniformly sampled screenshots, max long edge 600px, no UI metadata or agent reasoning) with three backends: GPT-5-mini, GPT-5.2, and Qwen2.5-VL-72B. All produce binary rewards (1.0 for success, 0.0 for failure). The rule-based condition’s reward coincides with the evaluation signal below, so its result is best read as an upper bound.

Evaluation protocol. All conditions are evaluated using the ground-truth rule-based checker (not the LLM judge), ensuring fair comparison. Evaluation covers all 116 AndroidWorld task templates across 3 seeds ($s \in \{7, 30, 1234\}$), with greedy decoding (temperature = 0). Success rate is computed as the fraction of successful tasks, averaged over 3 seeds (348 evaluations per checkpoint). *Best checkpoint* refers to the checkpoint with the highest seed-averaged success rate for a condition.

Full task-set results. While Figure 3 reports the easy-set success rate, we also evaluate the same checkpoints on the complete 116-task suite. Taking the best success rate per condition (the same convention as for the easy set), the ordering is unchanged: rule-based 36.8%, GPT-5-mini 30.2%, GPT-5.2 27.9%, Qwen 26.4%. At the fixed final checkpoint (step 300), GPT-5.2 and Qwen tie at 25.9%; the precision comparison in §5.2.2 therefore holds under the best-checkpoint convention but not at a fixed step. Table 10 lists both conventions for every condition.

Reward signal	Best	Final (step 300)
Rule-based	36.8	36.8
GPT-5-mini judge	30.2	30.2
GPT-5.2 judge	27.9	25.9
Qwen judge	26.4	25.9

Table 10: Success rate (%) on the complete 116-task AndroidWorld suite, averaged over 3 seeds: best checkpoint per condition vs. the final (step-300) checkpoint.

Per-seed training curves. Figure 7 shows the per-seed breakdown.

F Failure analysis details

F.1 Failure case examples

Figures 8 and 9 show representative examples for each of the 7 failure categories, with a key trajectory screenshot and analysis.

F.2 Failure identification

For each LLM backend (GPT-5-mini and Qwen2.5-VL), we run the 5 existing judge methods (excluding our simple baseline) on the 931 benchmark trajectories and identify cases where the judge prediction disagrees with human ground truth. We then compute the intersection of failures across methods: a trajectory is a *hard-core failure* if at least 4 out of 5 methods produce the wrong prediction. This threshold ensures that the failures are not method-specific artifacts but reflect cases that are fundamentally difficult for the given LLM backbone.

F.3 Root cause classification

For each hard-core failure case, we compile a structured dossier containing: (1) the task instruction, (2) the human ground-truth label and the error direction (FP or FN), (3) the judge predictions and

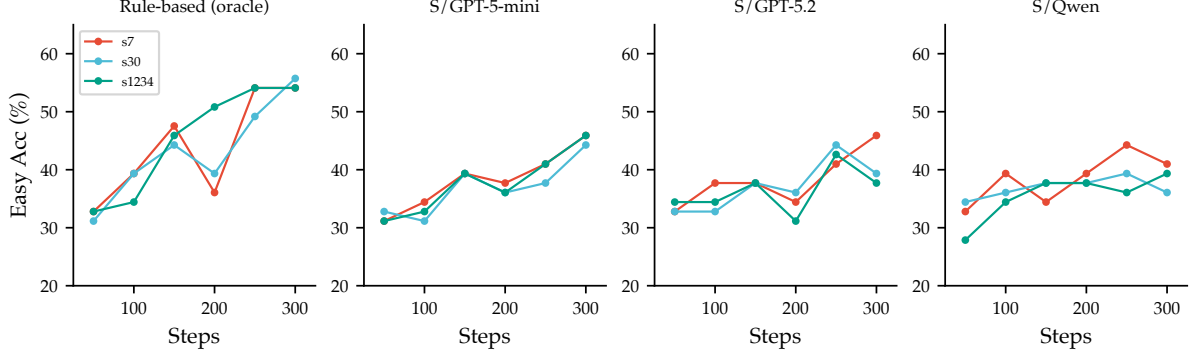


Figure 7: Per-seed easy-set accuracy for each reward condition. Seed variance is moderate; the ranking Rule-based > GPT-5-mini > GPT-5.2 > Qwen is consistent across most seeds and steps.

reasoning from all 5 methods, and (4) key trajectory screenshots. We then classify each case into one of the following predefined error categories, based on examining the trajectory evidence and judge reasoning:

False negative categories (judges miss true success):

- **Last-frame anchoring:** The judge over-relies on the final screenshot and misses evidence of task completion from earlier steps.
- **Invisible state change:** The task was completed but the change is not visually evident (*e.g.*, a setting toggled internally, an item deleted from a database).
- **Verification gap:** The trajectory lacks explicit visual confirmation for irreversible actions (*e.g.*, delete, post, toggle) despite a consistent final state.
- **Unfamiliar success state:** The judge applies overly strict criteria misaligned with the benchmark’s definition of success.

False positive categories (judges accept true failures):

- **Surface UI match:** The judge concludes success based on superficial visual similarity (*e.g.*, a screen that looks like the target but is not).
- **Partial completion:** Only a subset of the task requirements is met; the judge overlooks the remaining components.
- **Constraint violation:** The judge misses task-specific constraints (*e.g.*, “nearest,” “cheapest,” a specific date or quantity).

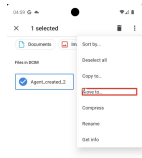
Using these categories, we classify 48 hard-core failures for the GPT-5-mini set and 78 for the Qwen set. The full distributions are shown in Figure 4.

F.4 Taxonomy reliability

Two raters independently categorized a random sample of 30 hard-core failure cases (10 false negatives, 20 false positives) into the seven root-cause categories, given the task instruction, the ground-truth outcome, the trajectory, the judges’ reasoning, and the written category definitions. The raters agreed on 29 of 30 cases (96.7%; Cohen’s $\kappa = 0.957$; false-negative subset 10/10, $\kappa = 1.000$; false-positive subset 19/20, $\kappa = 0.924$). The single disagreement is a borderline case for which two categories plausibly apply (surface UI match vs. constraint violation). We attribute the high agreement to the categories being defined in terms of concrete, observable behaviors.

G Use of AI Assistants

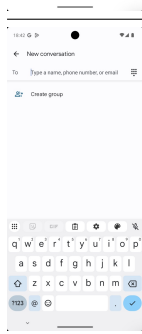
We used AI Assistants for writing assistance: editing prose for clarity and consistency of register, and suggesting rewrites of individual passages. The authors reviewed every suggestion and applied them selectively. Literature search was performed by the authors without AI assistance. All research ideas, experimental design, analyses, results, and claims are the authors’ own; the authors verified all AI-suggested text and take full responsibility for the content of this paper.



FN: Last-frame anchoring (GPT: 12, Qwen: 1)

Task: “Go to the DCIM folder in internal storage. Create a subfolder named Agent_created.” **GT:** Success **Pred:** Failure

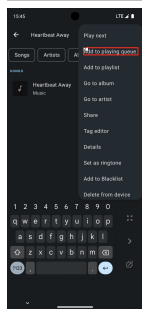
The agent created the subfolder in an earlier step, but then navigated away. The final screenshot shows a different location, so judges anchored to the final frame miss the completion evidence.



FN: Unfamiliar success state (GPT: 9, Qwen: 4)

Task: “Start chatting in message.” **GT:** Success **Pred:** Failure

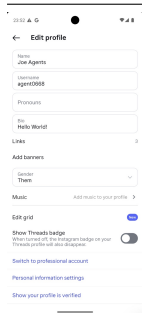
The agent reaches the “New conversation” screen with the recipient field active. Judges require a sent message, but the benchmark defines success as simply entering the chat composition flow.



FN: Invisible state change (GPT: 7, Qwen: 1)

Task: “Add songs to the playing queue: Through the Storm, Hidden Paths, Forever Young, ...” **GT:** Success **Pred:** Failure

The agent added songs via transient overflow-menu actions (“Add to queue”), but never opened the queue screen. No screenshot shows the resulting queue contents.

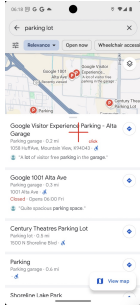


FN: Verification gap (GPT: 2, Qwen: 1)

Task: “Edit profile: add link, change gender to Custom, switch to private account.” **GT:** Success **Pred:** Failure

Link and gender changes are evidenced by toasts in earlier steps, but the privacy toggle is never visually confirmed—no frame shows “Private account” being enabled.

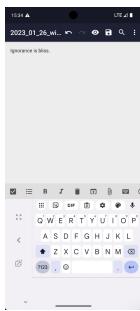
Figure 8: Representative false negative failure examples (4 categories). Judges incorrectly predict failure despite task success.



FP: Constraint violation (GPT: 5, Qwen: 27)

Task: “Find the nearest parking lot.” **GT:** Failure **Pred:** Success

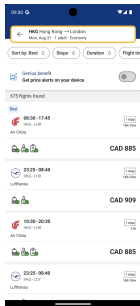
Judges accept opening any parking lot result from a Google Maps search. The task requires the *nearest* one, but no judge verifies whether the selected result has the smallest distance.



FP: Partial completion (GPT: 5, Qwen: 27)

Task: “Create a note in Markor named 2023_01_26_wise_yacht.md with text: Ignorance is bliss.” **GT:** Failure **Pred:** Success

Judges see the filename and text in the editor and conclude success. However, the note was never saved—Markor shows an unsaved draft, and the file does not exist on the filesystem.



FP: Surface UI match (GPT: 8, Qwen: 17)

Task: “Search for one-way flights from Hong Kong to London on Aug 31. Which is the cheapest?”

GT: Failure **Pred:** Success

Judges see a flight results page with matching headers (HKG→London, Aug 31, Economy) and assume success. The agent never identified the cheapest flight—the visual similarity was sufficient to fool all judges.

Figure 9: Representative false positive failure examples (3 categories). Judges incorrectly predict success despite task failure.