RAF: RUBRICS AUTO-GENERATION FRAMEWORK FOR AUDITABLE EVALUATION OF LONG-HORIZON LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating LLM agents with rubrics is a popular approach, which specifies a set of textual criteria with per-criterion weights, together with a scoring rule that aggregates criterion-level judgments into a single score. However, it is challenging to design effective rubrics due to rubric bias and score instability, which can lead to unreliable evaluation results that deviate substantially from human judgments. In this paper, we address these challenges by proposing a novel rubric generation method. First, we introduce an AND–OR tree as the underlying rubric structure, which organizes criteria into two types of nodes: leaf nodes, each representing an atomic and verifiable criterion, and condition nodes, which hierarchically aggregate the judgments of leaf nodes through AND/OR conditions. Secondly, we introduce Rubric Auto-generation Framework (RAF) that automatically and iteratively optimizes rubrics within AND-OR tree structure to improve their alignment with human judgments. Furthermore, we construct a benchmark comprising 121 long-horizon tasks from personal-assistant scenarios, annotated by human experts. On this benchmark, RAF outperforms baselines by a clear margin: it improves the Pearson correlation with human by 0.136 over the strongest baseline, reaching 0.8102, and reduces the average scoring variance from 10.18 to 3.36.

1 INTRODUCTION

In recent years, large language model (LLM)-based agents have emerged as a powerful paradigm for solving complex tasks through multi-step reasoning and tool use. By decomposing tasks into sequences of actions and interacting with environments, these agents have demonstrated strong capabilities across a wide range of personal-assistant tasks, including office work (Patwardhan et al., 2026; Ye et al., 2026; Kilo AI, 2026), personal life (Ye et al., 2026), and knowledge management. As these assistants move from toy settings to real-world deployment, reliably evaluating what they can and cannot do becomes an important factor for further progress. Without trustworthy evaluation, potential failures arising from either the underlying LLMs or the agentic harnesses cannot be accurately identified, diagnosed, or addressed.

Evaluating agents on personal-assistant tasks is challenging, as these tasks are often long-horizon and open-ended. A straightforward approach is *LLM-as-a-Judge* (Zheng et al., 2023; Wei et al., 2026), which uses an LLM with a task-specific instruction prompt for evaluation. A more structured approach is rubric-based evaluation, which specifies a set of textual criteria with per-criterion weights and aggregates criterion-level LLM judgments into a single score. Nevertheless, applying rubrics to agentic tasks raises two key challenges, which we analyze in detail in §3: **rubric bias**—coarse-grained criteria lack discriminative power, whereas overly fine-grained criteria may deviate from human judgments (Li et al., 2026b); and **score instability**—repeated evaluations of the same agent trajectory can yield noticeably different scores across judges(agent or human experts) and even across runs of the same judge (Rao & Callison-Burch, 2026).

Prior remedies address each challenge only partially. For rubric bias, RubricHub scores coarsely first and re-scores with finer criteria only when models tie (Li et al., 2026b), but at the fine end it pursues ranking discriminability alone, giving up alignment with human preference. For score insta-

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

bility, Rubrics-as-Rewards averages over many repeated judgments (Gunjal et al., 2026) and QuRL filters out non-reproducible criteria (Wei et al., 2026), treating instability as noise to be averaged or discarded rather than reduced at its source, while AutoRubric attributes misalignment to judge-side biases such as position and length bias (Rao & Callison-Burch, 2026) without examining the rubric itself. In short, existing remedies either patch the scoring procedure around a flawed rubric or sacrifice preference alignment for discriminability—none verifies, during rubric construction, that the resulting scores actually reproduce human judgments.

In this work, we address these challenges by introducing a novel rubric generation method. *First*, we propose an AND–OR tree as the underlying rubric structure for case-level evaluation. Specifically, we organize evaluation criteria into two types of nodes: leaf nodes, which represent atomic and verifiable criteria, and condition nodes, which hierarchically aggregate the judgments from leaf nodes. AND nodes jointly aggregate multiple criteria, capturing cases where all requirements must be satisfied, while OR nodes represent alternative success modes, allowing the rubric to accommodate diverse ways of completing a task and thereby calibrate evaluation granularity structurally.

Second, we propose a Rubric Auto-generation Framework (RAF) that automatically and iteratively optimizes rubrics within the AND–OR tree structure. Crucially, refinement is driven by measured alignment with a human-preference gold standard rather than by intuition, transforming laborious manual calibration into a controlled and auditable optimization loop. Furthermore, we construct a benchmark comprising 121 long-horizon tasks from personal-assistant scenarios which are annotated by human experts to provide a reliable gold standard for evaluating rubric quality and RAF.

Our core contributions are summarized as follows:

- We propose a novel rubric structure, termed the *AND–OR tree*, which represents atomic evaluation criteria as leaf nodes and aggregation operations as condition nodes. Building upon this structure, we introduce the *Rubric Auto-generation Framework (RAF)*, which iteratively optimizes rubrics to improve their alignment with human judgments.
- We construct a new benchmark, *DailyClawBench*, comprising 121 tasks from personal-assistant scenarios. The tasks are collected from real-world user interactions and annotated by human, providing a reliable testbed for evaluating agent performance and rubric quality.
- We evaluate 6 LLMs on *DailyClawBench* and demonstrate the effectiveness of our rubric generation method on this benchmark as well as several existing benchmarks.

2 Related Work

Evaluation methods for agent systems. Evaluating autonomous agents is hard because valid trajectories are diverse: action-sequence matching penalizes correct-but-different paths (Patil et al., 2025), rule-based validation cannot cover open-ended scenarios (Barres et al., 2025), LLM-as-a-judge is flexible but biased and unstable (Zheng et al., 2023), and reward models compress criteria into opaque scalars (Ouyang et al., 2022). Rubric-based evaluation has emerged as a middle ground, proving effective in medicine (Arora et al., 2025) and deep research (Sharma et al., 2025) and serving as supervision beyond verifiable domains (Gunjal et al., 2026). Since the harness itself can shift a model's score by up to 18 points (Ding et al., 2026), DailyClawBench fixes the harness to OpenClaw and adopts rubric-based evaluation.

Automatic rubric generation. The cost of expert-authored rubrics motivates automatic generation—from reference answers, human-authored web sources, or preference data (Gunjal et al., 2026; Wei et al., 2026; Li et al., 2026b; Liu et al., 2025; Wang & Xiong, 2025; Huang et al., 2025)—but two limitations remain. First, *consistency*: single-pass generation is stochastic and criteria vary across runs and models, undermining downstream reliability (Rao & Callison-Burch, 2026); only Wei et al. (2026) explicitly filter out non-reproducible criteria. Second, *discriminability*: coarse criteria saturate, assigning uniformly high scores to responses of different quality (Li et al., 2026b), and existing remedies are partial or absent. RAF jointly enforces both properties through iterative, preference-verified refinement, which prior pipelines handle in isolation or ignore entirely.

Benchmarks for OpenClaw-like assistants. Recent benchmarks evaluate personal AI assistants over mocked services (Kilo AI, 2026), Dockerized tool use (Ye et al., 2026), or production plat-

forms (Zhang et al., 2026), with *living* variants that regenerate or evolve cases (Li et al., 2026a; Long et al., 2026). DAILYCLAWBENCH differs on both fronts: a *sandbox with real services* in a time-evolving configuration, and tasks hand-constructed under explicit rules from real OpenClaw user queries, covering both scenario types and capability distribution.

3 RUBRICS

Rubric-based evaluation has become a promising paradigm for assessing agents on open-ended tasks, yet most progress concerns *where* rubrics apply; how a rubric should be *structured and optimized* remains under-explored. We identify two failure modes of prevailing designs.

Scoring Instability. Most rubrics ask a human rater or judge model to assign a fuzzy score in $[0, 1]$ (any other scale normalizes likewise). Such continuous scoring introduces rater and judge-dependent noise: repeated evaluations of the same output fluctuate noticeably across LLMs and human.

Rubric Bias. Even with stable scoring, the criteria themselves can systematically deviate from human preference (Li et al., 2026b). Coarse criteria lack discriminative power: strong and weak models cluster near the top, and the rubric saturates. Overly fine-grained criteria deviate in the opposite way, a rubric enumerating checks lets a trajectory accumulate most points despite a critical error, diluting failures that humans penalize severely. Either way, the resulting scores drift from human judgments, and manually calibrating bias away is laborious and rarely converges.

These limitations motivate *case-level* rubrics organized as a *verifiable, binary AND–OR criteria tree*: binary verifiable criteria remove scoring instability, and case-specific trees are iteratively aligned with human preference by RAF (§3.2), correcting rubric bias rather than assuming it away.

3.1 RUBRICS STRUCTURE

Why a hierarchy? The acceptance criteria of an agentic task are rarely a flat list: some are conditionally exclusive—which apply depends on how the task unfolds—while others hold in parallel and must be satisfied jointly. We therefore organize each case-level rubric as an *AND–OR criteria tree* (Figure 1):

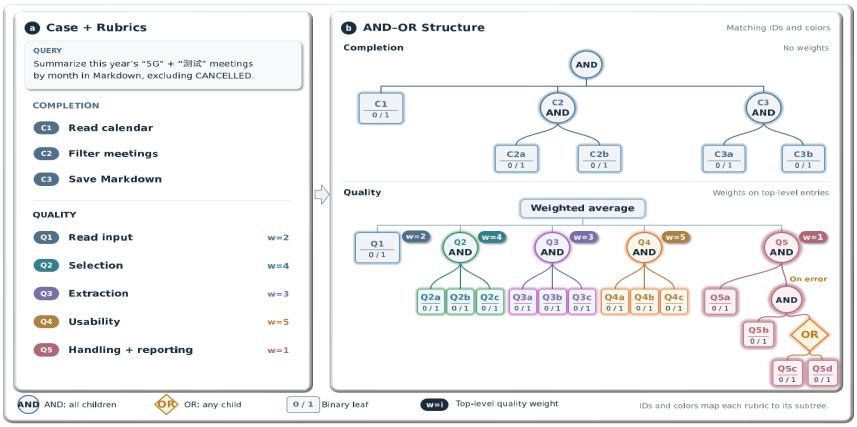

Figure 1: Rubric Structure.

OR nodes route between conditionally divergent success modes, *AND nodes* conjoin parallel criteria, and *leaf nodes* carry binary, verifiable checks, phrased to be self-contained so they can be assessed by rule-based checkers or a judge agent without external context. The hierarchy keeps every leaf simple enough to judge reliably, while composing the checks into the complex, case-specific standard that open-ended tasks require.

OR nodes: conditionally divergent success modes. When the correct behavior depends on the situation encountered, the applicable criteria differ across situations. For example, when adding a meeting to a calendar: if the event conflicts with existing ones, the agent should alert the user or

propose rescheduling; otherwise it should add the event directly. Each OR branch therefore carries an applicability condition, and only the branch matching the realized trajectory is evaluated.

AND nodes: parallel criteria. Sub-criteria in a group often describe parallel aspects of the deliverable (e.g., for slide layout, the arrangement of titles and body text and the choice of font sizes) and must hold jointly. We considered two aggregation schemes: a (weighted) *mean*, as adopted by existing rubric evaluations (Arora et al., 2025; Gunjal et al., 2026), and a strict logical *AND*. Empirically, under fine-grained rubrics the mean lets a few critical failures be diluted by many trivially satisfied checks, saturating scores and weakening alignment with human preference; the AND relation, by making every criterion necessary, aligns better. We therefore adopt AND.

Separating completion from quality. Following the long-standing principle of separating *whether* a task is completed from *how well* (Walker et al., 1997; OmegaUseTeam, 2026), each rubric is a pair $\mathcal{R}_x = (\mathcal{T}_x^{\text{task}}, \mathcal{T}_x^{\text{qual}})$, and we report $(s^{\text{task}}, s^{\text{qual}})$ separately rather than a single conflated score. The completion part is strictly conjunctive—the task passes or fails. The quality part is graded instead: each of its *top-level entries*—an AND group, an OR group, or a standalone leaf—carries an importance weight $w \in \{1, \dots, 6\}$ (nodes inside a group are unweighted and always judged binarily), assigned at generation and adjusted during RAF refinement. Since each entry's score is binary (an AND group fails as a unit), the weights determine how much a failed aspect costs: a critical failure with a high weight lowers the quality score far more than any number of trivial ones. An example is provided in Appendix F.

Formal definition. Let x denote a case (query and environment) and τ the trajectory together with final environment. Each case carries an AND–OR criteria tree $\mathcal{T}_x = (V, E, r)$ whose internal nodes are labelled $\text{op}(v) \in \{\text{AND}, \text{OR}\}$ and whose leaves are binary criteria $c_\ell : (x, \tau) \mapsto \{0, 1\}$. The m children of an OR node carry mutually exclusive and exhaustive *applicability conditions* $\varphi_1, \dots, \varphi_m : (x, \tau) \mapsto \{0, 1\}$ with $\sum_{i=1}^{m} \varphi_i(x, \tau) = 1$. Node scores are defined recursively:

$$\text{score}(v;\, x, \tau) = \begin{cases} c_v(x, \tau), & v \text{ is a leaf}, \\ \min\limits_{u \in \text{ch}(v)} \text{score}(u;\, x, \tau), & \text{op}(v) = \text{AND}, \\ \text{score}(u_{i^*};\, x, \tau) \;\text{ with }\; \varphi_{i^*}(x, \tau) = 1, & \text{op}(v) = \text{OR}. \end{cases} \tag{1}$$

With $g_1, \dots, g_k$ the top-level entries of the quality tree $\mathcal{T}_x^{\text{qual}}$ and weights $w_1, \dots, w_k \in \{1, \dots, 6\}$, each rubric yields a binary completion score $\text{score}(r;\, x, \tau) \in \{0, 1\}$ and a weighted quality score

$$s^{\text{qual}}(x, \tau) = \frac{\sum_{i=1}^{k} w_i \,\text{score}(g_i;\, x, \tau)}{\sum_{i=1}^{k} w_i} \;\in [0, 1]. \tag{2}$$

Given a trajectory τ, every OR node collapses onto its applicable branch, so the tree acts as a *case- and trajectory-specific flat binary checklist* materialized at evaluation time. Flat instance-specific checklists (Arora et al., 2025; Gunjal et al., 2026) are the degenerate case with no OR nodes; relaxing AND to a weighted mean recovers the continuous schemes criticized above.

3.2 Rubrics Auto-generation Framework

The AND–OR structure specifies what a reliable rubric should *look like*, not how to *obtain*: fine-grained rubrics are discriminative but labor-intensive to design case by case, prior auto-generation pipelines leave rubric quality unverified (Rao & Callison-Burch, 2026). We propose the **Rubrics Auto-generation Framework (RAF)**, a pipeline coupling LLM-based generation with quantitative alignment verification, so that rubric quality is *certified* rather than assumed (Figure 2).

Overall flow. Given a user query and its initial environment state, RAF proceeds in five steps: (1) *Rubric generation*: an LLM produces an initial AND–OR rubric from a designed prompt; (2) *Expert review*: a human expert inspects the rubric for major defects (missing intent coverage, unjudgeable criteria); if any are found, the pipeline jumps directly to Step 5; (3) *Score recording*: a judge agent scores the case's trajectories under the current rubric, compared against a human-annotated preference gold standard; (4) *Termination*: an alignment score is computed—above a threshold the rubric is accepted, otherwise the inconsistencies are attributed and forwarded to Step 5; (5) *Suggestion,*

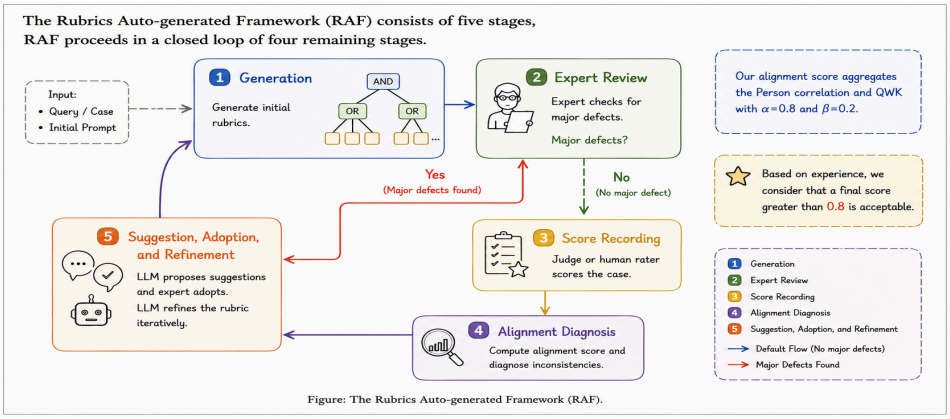

Figure 2: The Rubrics Auto-generated Framework (RAF) process.

adoption, and refinement: the LLM proposes revisions from the expert-reported defects or the quantitative diagnosis; the expert audits each suggestion, and the LLM refines the rubric from adopted ones before returning to Step 2. Every revision is thus traceable to an failure, and the loop terminates under verified alignment with human preference. We'll focus on introducing two key steps as follow.

Step 1: rubric generation. The initial rubric's quality largely determines the refinement rounds needed, so we invest design effort in the initial prompt, organized into seven parts—role definition, target rubric structure, design principles, few-shot examples, hard constraints on tree size and wording, a reference capability list, and output format (Appendix E). The *principles* deserve emphasis: the failure modes identified by RIFT (Qi et al., 2026) in inverted form—objective judgeability, atomicity, self-containment, goal–process separation, full intent coverage, behavior orientation–plus distilled from our practice, discriminativeness, non-redundancy,*deliverable usability*.

Step 4: termination condition. Deciding *when* a rubric is good enough is as important as generating it: pipelines that stop after fixed iterations or once output looks plausible give no guarantee of reproducing human judgments. RAF terminates only when the rubric demonstrably aligns with preference, measured by

$$\text{Alignment Score} = \alpha \cdot \text{Pearson} + \beta \cdot \text{QWK}, \tag{3}$$

with $\alpha = 0.8$ and $\beta = 0.2$ (definitions in Appendix A). Pearson captures whether the rubric preserves the *ordering* of human preferences, QWK whether its absolute ratings *agree* with them; combining the two prevents terminating on a rubric that ranks correctly but scores inconsistently. We weight Pearson higher ($\alpha = 0.8$) because model comparison ultimately rests on ranking, and set the threshold at 0.8: it coincides with the conventional "almost perfect" band of QWK (Landis & Koch, 1977) and, in our pilot iterations, marked the point beyond which further rounds produced only marginal metric gains. A score above 0.8 freezes the rubric into the benchmark; otherwise the attribution of the inconsistencies is passed to Step 5, where the LLM derives revisions from this quantitative diagnosis, with the expert again deciding suggestion by suggestion.

From framework to benchmark. RAF turns rubric construction from an ad-hoc craft into a repeatable, quality-certified procedure, making it practical to build an agentic benchmark in*every* task carries a validated fine-grained rubric and to keep rebuilding it as usage evolves, at the cost of a few refinement rounds per new case rather than a manual redesign. The next section instantiates this methodology in DAILYCLAWBENCH (§4), with transfer to public benchmarks evaluated in §4.3.5.

4 EXPERIMENT

4.1 BENCHMARK CONSTRUCTION

Data and Environment. DAILYCLAWBENCH is built from real usage rather than synthetic task descriptions: we provided 210 OpenClaw users with a self-deployed API, collected their genuine

queries and interaction trajectories between March and July 2026, and retained 121 evaluation tasks after filtering and selection. To make evaluation results *interpretable*–attributing a model's failure to a specific scene, capability, or source of difficulty rather than to the benchmark as a whole–each task is labeled along four dimensions: scene, competency, difficulty, and environment complexity, with category definitions and label distributions deferred to Appendix B. Each task is paired with exactly one sandbox snapshot which preserving the OpenClaw version at collection time, plus a per-task initial environment comprising the query, the agent's skill and plugin configurations, and the initialized file and other systems–from independent containers are spawned so that all tasks execute in parallel and isolation. Because snapshots are decoupled from their collection, evaluation results remain reproducible across time and OpenClaw versions; containers are preserved after execution, exposing workspace to the judge agent through a tool interface for artifact inspection.

Evaluator construction. Each task's evaluator couples the rubric produced by RAF (§3.2) with a judge agent that inspects the preserved container and the extracted deliverables, assigns a verdict to each binary leaf criterion, and aggregates the tree following Eq. equation 1; we defer the verification of RAF itself to §4.3. Since the tree separates completion from quality (§3), the evaluator reports two complementary numbers per task: the *pass rate*—a task passes only if all mandatory completion criteria hold—and the *quality score*, which aggregates the quality subtree so that critical quality failures are penalized however strong the rest. Across the 121 tasks, the rubrics average 15.9 binary leaf criteria per task and are primarily conjunctive (3.81 AND vs. 0.59 OR group nodes per task); full statistics are given in Appendix D, and the complete rubric of one representative task in Appendix F. These per-case evaluators are what the main experiments of §4.2 score every model against.

4.2 Main result

Model comparison and failure analysis. Table 1 reports pass rate and quality score for six frontier models, overall and per scene. Claude Opus 4.7 is the strongest and most balanced agent (pass 85.95, quality 90.41). The bench *discriminates*: scores span 8 points on pass rate and 10 on quality, with no model near the ceiling. The two dimensions are *non-redundant*: their rankings disagree for five of the six models—GPT-5.4 ranks second in quality (89.36) but only third in pass rate (80.90)—so any scalar aggregate would misrepresent most of the field. The scene breakdown exposes specialization the overall column hides: Knowledge Management is hardest for nearly every model, while in Basic Office even Claude's lead vanishes (GLM-5 and DeepseekV4-Pro reach 85.84 pass vs. Claude's 83.87). Every model scores higher on quality than on pass rate: mandatory completion criteria, not quality criteria, are the binding constraint on current agents. These rankings are robust to the choice of judge (Appendix C; §4.3.4). Breaking down Opus 4.7 along the labeling dimensions (Fig 3) locates the remaining headroom: performance degrades gracefully with difficulty (Easy $99.0 \rightarrow$ Hard 87.5) and complexity ($99.0 \rightarrow 88.8$), while the capability view isolates a clear outlier—*reflection and repair* scores only 79.69, over 10 points below every other capability and the tag least exercised in real usage (18 of 121 tasks): the frontier of office-agent capability lies in self-correction rather than instruction execution.

Table 1: Benchmark evaluation results across different models and domains.

Model	Personal		Knowledge		Office		Overall	
	Pass	Score	Pass	Score	Pass	Score	Pass	Score
GPT 5.4	82.64	93.52	79.92	85.35	79.71	88.89	80.90	89.36
Claude opus4.7	89.13	91.19	84.09	87.85	83.87	92.87	85.95	90.41
GLM 5	76.44	79.99	73.43	81.09	85.84	83.61	77.75	81.32
kimi-k2.6	76.44	83.25	77.76	82.07	82.37	88.05	78.44	84.05
DeepseekV4-Flash	83.75	85.99	75.05	75.50	76.93	78.61	78.84	80.25
DeepseekV4-Pro	82.64	88.77	79.92	89.56	85.84	84.71	82.47	88.02

4.3 Effectiveness of rubrics

The main results above are only as trustworthy as the evaluators producing them. This subsection therefore turns the evaluation apparatus on itself, in four steps. We first show that case-level rubrics are fundamentally superior to generic judging prompts in both alignment and stability (§4.3.1); we then trace how RAF closes the remaining gap to human preference (§4.3.2), and ablate its human

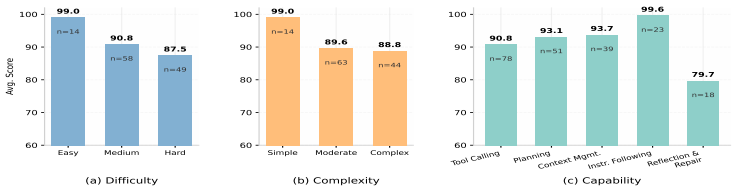

Figure 3: Performance of Claude Opus 4.7 across difficulty, environment complexity, and capability. Reflection & repair is the only capability below 90.

Table 2: Alignment with the human gold standard (left) and scoring stability (right), under the same judge (Claude Opus 4.7). Stability is measured on quality scores over three independent judging runs per case: variance and standard deviation are computed within each case and averaged over the 121 cases. The RAF row corresponds to the End setting of Table 3.

	Alignment				Stability	
Evaluator	**Pearson**	**Spearman**	**QWK**	**Comp.**	**Var.**	**Std.**
WildBench(Generic prompt)	0.6831	0.4955	0.6406	0.6746	51.25	5.09
AutoRubric (case-level)	0.6370	0.4170	0.6503	0.6397	10.18	2.89
Ours RAF(case-level)	**0.8147**	**0.6297**	**0.7920**	**0.8102**	**3.36**	**1.83**

component (§4.3.3); next, we justify our choice of judge agent and verify that rubric optimization transfers across judges (§4.3.4); finally, we test whether our data structure generalizes beyond our bench (§4.3.5).We conduct all rubric-level analyses on GLM-5 trajectories—whose mid-tier profile yields the clean successes, partial deliveries, and failures needed to stress-test evaluators—using the independently annotated expert gold standard from §4.3.2 and Claude Opus 4.7 as the default judge, with the judge-transfer analysis in §4.3.4 confirming that the resulting alignment gains reflect rubric properties rather than the GLM-5 error distribution.

4.3.1 Rubric Bias and Stability: RAF vs. Existing Evaluators

We first validate the premise of §3: criteria must be configured case by case *and* verified against human preference. Under the same judge and gold standard (Table 2), we compare the final RAF rubrics with two baselines: a *generic judging prompt*, for which we adopt WildBench (Lin et al., 2025), a widely used evaluation prompt for open-ended real-user tasks that applies general desiderata (completeness, correctness, instruction compliance, formatting, helpfulness) uniformly to every case; and *AutoRubric* (Rao & Callison-Burch, 2026), a recent framework that generates case-level rubrics in a single pass and attributes misalignment to judge-side biases (position, length), but never verifies the generated rubric itself against human preference.

Granularity buys alignment—but only with verification. The WildBench prompt achieves a non-trivial composite of 0.6746: general desiderata capture the coarse contour of human preference. But it falls 0.136 short of RAF (0.8102), with the largest gap on ranking (Spearman 0.4955 vs. 0.6297)—generic criteria separate good deliverables from poor ones, but cannot resolve the case-specific distinctions on which human rankings actually turn (e.g., which conflict-handling branch applies in the calendar example of §3.1). Case-level structure alone is not the answer either: AutoRubric, despite being case-level, scores *below* the generic prompt (0.6397), showing that rubrics generated without preference-verified refinement can underperform no rubric construction at all. This is the rubric bias of §3 in empirical form: it rules out both one-rubric-for-all prompting and unverified per-case generation, leaving certified refinement as the differentiating ingredient.

Rubrics structure buys stability. We repeat quality scoring three times per case and report per-case variance and standard deviation averaged over the 121 cases (Table 2, right columns). The generic prompt is unstable (average variance 51.25, std. 5.09 on a 100-point scale), as every run re-instantiates the implicit criteria from scratch and lands on a different interpretation. Confining sampling randomness to independent, low-ambiguity binary verdicts cuts variance to 10.18 (AutoRubric), and RAF's verified rubrics further to 3.36—a $15\times$ reduction over the generic prompt, mitigating a known failure mode of LLM-as-judge evaluation (Zheng et al., 2023) by the *structure* of the criteria rather than by ensembling or repeated voting.

Table 3: Alignment with the human gold standard across RAF refinement stages (top block) and under the ablation that accepts all LLM suggestions without expert auditing (bottom block). Completion metrics are computed on binary leaf verdicts; process metrics on overall preference scores.

	Completion			Process			
Stage	**Acc.**	**Kappa**	**F1**	**Pearson**	**Spearman**	**QWK**	**Comp.**
Start	0.8678	0.5722	0.9184	0.7080	0.4733	0.7408	0.7146
Iteration R1	0.8718	0.5897	0.9206	0.7035	0.5302	0.7177	0.7063
Iteration R2	0.8760	0.5917	0.9239	0.7773	0.4531	0.7679	0.7754
Final	**0.9339**	**0.7696**	**0.9600**	**0.8147**	**0.6297**	**0.7920**	**0.8102**
Ablation R1	0.8833	0.6379	0.9271	0.6002	0.4085	0.6240	0.6050
Ablation R2	0.8926	0.6461	0.9340	0.6856	0.4655	0.7193	0.6923

4.3.2 RAF ITERATION

To establish a human ground truth for validating the generated rubrics, three expert annotators independently score each trajectory following a complete annotation guideline, while remaining blind to the case's generated rubric agent and rubrics review expert so that the gold standard stays independent of the evaluator being validated. Annotation proceeds in three steps: independent scoring, collective discussion followed by revision and averaging, and final joint review of each case. Critically, the annotators are deliberately a *different* group from the experts who review and audit rubrics inside the RAF loop (§3.2): the loop therefore optimizes each rubric toward an independent external target, rather than toward the preferences of the same individuals who edited it.

The initial rubrics (Start) already encode the full AND–OR tree, yet agree only moderately with human preference: completion kappa is 0.5722 (raw accuracy 0.8678 is inflated by class imbalance), and the process composite 0.7146 is dominated by Pearson while QWK at 0.7408—**an unrefined rubric reproduces how *much* humans reward a trajectory, but not how they *rank* competing ones** (Rao & Callison-Burch, 2026). After refinement (Final), kappa rises to 0.7696 and the composite to 0.8102—a relative gain of over 33% on Spearman, the metric hardest at the start—surpassing the 0.8 termination threshold of Step 4 and empirically validating it. The stage trajectory is equally informative: the first two rounds bring only marginal changes, while the decisive jump occurs in the final round, where kappa, Spearman, and the composite improve sharply and simultaneously—early rounds fix locally visible defects, whereas the later diagnosis-guided rounds (Step 4→Step 5) correct systematic scoring biases against the gold standard. **The rubric bias of §3 is thus corrected not by hand-tuning criteria toward a coarse–fine sweet spot, but by iterative alignment with human preference**—what distinguishes RAF from pipelines that stop once rubrics look reasonable (Li et al., 2026b; Wei et al., 2026).

4.3.3 HUMAN IN THE LOOP

We ablate the expert's gatekeeping role in Step 5: the *Ablation* accepts all LLM revision suggestions unconditionally, from the same initial rubrics and under the same judge (bottom block of Table 3). Removing human review degrades process alignment: one blind round already drops the composite to 0.6050—*below* the 0.7146 of the unrefined rubrics—and the gap persists after two rounds (0.6923 vs. 0.7754), so **iteration without gatekeeping is worse than no iteration at all**: the loop's value lies in *selecting* among LLM suggestions, not merely generating them (cf. the uncurated AutoRubric baseline, 0.6397 overall; §4.3.1). Deceptively, the ablation slightly *leads* on all three completion metrics (kappa 0.6461 vs. 0.5917 at round 2): many proposals sharpen surface-level wording, making binary leaves easier to judge, while distorting global calibration (re-weighting branches, duplicating criteria)—defects invisible at the leaf level that corrupt the aggregated preference signal, and precisely what the auditor rejects. Hence **completion accuracy can improve while the rubric as a whole gets worse**, retroactively justifying terminating (§3.2) on composite preference alignment.

4.3.4 JUDGE AGENT

We compare three candidate judges—Claude Opus 4.7, GPT-5.4, and DeepseekV4-Flash—under the *initial* rubrics (upper rows of Table 4; judge prompt in Appendix G). Opus 4.7 is strongest by a clear margin (composite 0.7146 vs. GPT-5.4's 0.5629), so we fix it as the judge for all RAF iterations and

Table 4: Performance of scoring agents under the initial and optimized rubrics. Bold values indicate the best performance in each column within each rubric stage. The Opus rows correspond exactly to the Start and Final rows of §4.3.2.

		Completion			Process			
Judge	**Rubrics**	**Acc.**	**Kappa**	**F1**	**Pearson**	**Spearman**	**QWK**	**Comp.**
Claude Opus 4.7	Initial	**0.8678**	**0.5722**	**0.9184**	**0.7080**	0.4733	**0.7408**	**0.7146**
	Optimized	**0.9339**	**0.7696**	**0.9600**	**0.8147**	**0.6297**	**0.7920**	**0.8102**
GPT-5.4	Initial	0.7456	0.4218	0.8221	0.5813	0.3869	0.4937	0.5629
	Optimized	0.8000	0.4706	0.8681	0.6611	**0.5584**	0.5492	0.6387
DeepseekV4-Flash	Initial	0.8235	0.5288	0.8840	0.6145	**0.5029**	0.6404	0.6196
	Optimized	0.9174	0.7227	0.9495	0.6915	0.5357	0.7278	0.6987

Table 5: Generalization: task success rates from official gold outcomes vs. our RAF rubric evaluator.

Benchmark	**Trajectories**	N	**Gold**	**Ours**
τ^2-bench (airline)	Claude Sonnet 4.5 (official)	50	70.0%	68.0%
AppWorld	self-generated	30	53.3%	50.0%

main experiments; **the main results are not an artifact of judge choice**, as re-scoring with GPT-5.4 leaves the quality-score ranking unchanged and the pass-rate top three intact (Appendix C), and the binary, verifiable leaf verdicts structurally limit the risk of Opus favoring its own outputs as a contestant (§4.2). Fixing one judge during refinement, conversely, raises the concern that the loop overfits to that judge's preferences; the optimized rubrics rule this out: *every* judge improves on nearly every metric—composite gains of 0.076 (GPT-5.4) and 0.079 (DeepseekV4-Flash), mirroring Opus's own $+0.096$—although neither participated in the loop. **The gains therefore stem from the rubrics, not the judge**: RAF removes ambiguity and unjudgeable criteria that hamper *any* judge.

4.3.5 Generalization to External Benchmarks

Finally, we ask whether the AND–OR rubric structure and judge interface are tied to DailyClaw-Bench, or constitute a reusable evaluation schema. We stress-test generality on two external benchmarks differing from ours in both *interaction mode* and *verification style*: τ^2-bench (Barres et al., 2025) (multi-turn conversational tool use under policy constraints; 50 official Claude Sonnet 4.5 trajectories, airline domain) and AppWorld (Trivedi et al., 2024) (multi-step programmatic tasks; 30 sampled tasks with self-generated trajectories), both defining success by database or environment state checks—stricter and more objective than our artifact-quality judgments, so agreement cannot be attributed to lenient shared biases. For each task, we instantiate an AND–OR rubric with RAF without any per-benchmark tuning, score with our judge, and compare against the official gold outcome (Table 5). Our evaluator tracks gold outcomes closely on both—within 2.0 points on τ^2-bench (68.0% vs. 70.0%) and 3.3 on AppWorld (50.0% vs. 53.3%)—and is consistently, if slightly, more conservative, consistent with conjunctive binary leaves penalizing partial states that lenient checks pass. **A single, untuned evaluation schema thus reproduces official outcomes on benchmarks built around different interaction modes and verification styles**: the AND–OR structure with binary verifiable leaves is a general way to construct agent evaluators, not an artifact of our domain.

5 Future Work

Several directions remain open. First, our case-level, binary, verifiable criteria are exactly the supervision form consumed by RLVR-style post-training (Gunjal et al., 2026; Wei et al., 2026; Li et al., 2026b; Huang et al., 2025), so applying RAF-generated criteria trees as reward signals is a natural next step; more broadly, by supplying auditable, human-aligned evaluation, RAF can strengthen the evaluation link in recursive self-improvement (RSI) loops, closing the loop between automatic assessment and automatic improvement. Second, as benchmarks are continually refreshed, how to evolve rubrics across versions while preserving cross-version score comparability is an important question for longitudinal evaluation. Finally, meta-evaluating and calibrating automatically generated rubrics at scale—also noted by Rao & Callison-Burch (2026)—deserves dedicated study.

ETHICS STATEMENT

DAILYCLAWBENCH is distilled from production queries of 210 OpenClaw users. Queries were collected under [the platform's terms of service / informed consent obtained at account registration], and all text underwent de-identification before annotation: 20 queries containing personally identifiable information, credentials, or otherwise sensitive content were removed, and the remaining tasks were rewritten and mocked where necessary to decouple them from any identifiable user context. No new user data was collected for this study; annotation was performed by 3 compensated domain experts. We do not foresee harmful uses of this work beyond those inherent to LLM-based evaluation generally, and we release rubrics and snapshots in de-identified form only.

AI USE STATEMENT

Large language models were used in two distinct roles in this work. First, as the *object of study*: LLMs generate and revise rubrics, judge agent trajectories, and drive the agents being evaluated; all such usage is described in Section 3.2 and Appendices E and G. Second, as a *writing aid*: the authors used LLM assistance for language polishing of the manuscript. All research ideas, experimental design, data, analyses, and conclusions are the authors' own, and the authors take full responsibility for the entire content of this paper.

REPRODUCIBILITY STATEMENT

We provide full prompts for rubric generation (Appendix E) and judging (Appendix G), a complete worked rubric example (Appendix F), and all model versions in Section 4. The 121 verified rubrics, task specifications, and anonymized trajectory snapshots, together with the code for rubric generation, refinement, and scoring, will be released at [https://github.com/RAFPAPER2026/RAF] under [Apache 2.0 for code, CC BY 4.0 for data]. External-benchmark results (Section 4.3.5) use the official τ^2-bench and AppWorld releases with their published gold outcomes.

REFERENCES

Rahul K. Arora et al. HealthBench: Measuring AI capabilities in healthcare, 2025.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation. *arXiv*, 2026.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *The Fourteenth International Conference on Learning Representations*, 2026. arXiv:2507.17746.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, et al. Reinforcement learning with rubric anchors. *arXiv preprint arXiv:2508.12790*, 2025.

Kilo AI. PinchBench: Real-world benchmarks for AI coding agents. `https://github.com/pinchbench/skill`, 2026. Benchmark repository and leaderboard, `https://pinchbench.com`.

J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

Chenxin Li, Zhengyang Tang, Mingxin Huang, Yunlong Lin, Shijue Huang, Shengyuan Liu, Bowen Ye, Rang Li, Lei Li, Benyou Wang, and Yixuan Yuan. Claw-eval-live: A live agent benchmark for evolving real-world workflows. *arXiv*, 2026a.

Sunzhu Li, Jiale Zhao, Huimin Ren, Zhenlin Wei, Yang Zhou, Jingwen Yang, Shunyu Liu, Kaike Zhang, and Wei Chen. RubricHub: A comprehensive and highly discriminative rubric dataset via automated coarse-to-fine generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31320–31344, 2026b.

Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Faeze Brahman, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking LLMs with challenging tasks from real users in the wild. In *The Thirteenth International Conference on Learning Representations*, 2025.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. OpenRubrics: Towards scalable synthetic rubric generation for reward modeling and LLM alignment. *arXiv preprint arXiv:2510.07743*, 2025.

Xiang Long, Li Du, Yilong Xu, RongJian Xu, Qiyanhui Lu, Ying Gao, Qinhua Xie, Fangcheng Liu, Ning Ding, Haoqing Wang, Ziheng Li, Changjiang Zhou, Jianyuan Guo, and Yehui Tang. Liveclawbench: Benchmarking llm agents on complex, real-world assistant tasks. *arXiv*, 2026.

OmegaUseTeam. Omegause-officeval: Benchmarking llm agents on long-horizon office-suite tasks with economic grounding. *arXiv*, 2026.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *ICML*, 2025.

Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simon Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, et al. Gdpval: Evaluating ai model performance on real-world economically valuable tasks. In *ICLR*, 2026.

Zhengyang Qi, Charles Dickens, Derek Pham, Amanda Dsouza, Armin Parchami, Frederic Sala, and Paroma Varma. Rift: A rubric failure mode taxonomy and automated diagnostics. *arXiv*, 2026.

Daksh Rao and Chris Callison-Burch. AutoRubric: Unifying rubric-based LLM evaluation. *arXiv preprint arXiv:2603.00077*, 2026.

Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, Aishwarya Balwani, Denis Peskoff, Marcos Ayestaran, Sean M. Hendryx, Brad Kenstler, and Bing Liu. Researchrubrics: A benchmark of prompts and rubrics for evaluating deep research agents. *arXiv*, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Advances in Neural Information Processing Systems*, 2024.

Marilyn A. Walker, Diane J. Litman, Candace A. Kamm, and Alicia Abella. Paradise: A framework for evaluating spoken dialogue agents. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, 1997.

Tevin Wang and Chenyan Xiong. AutoRule: Reasoning chain-of-thought extracted rule-based rewards improve preference learning. *arXiv preprint arXiv:2506.15651*, 2025.

Xiyu Wei, Qingwei Zong, Xiaoguang Li, Eugene J. Yu, and Sujian Li. QuRL: Rubrics as judge for open-ended question answering. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=DrhTuhtYq.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, et al. Claw-eval: Toward trustworthy evaluation of autonomous agents. *arXiv*, 2026.

Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Zhuofeng Li, Xingwei Qu, Zhengkang Guo, Yuanzhe Shen, Dingjie Song, Han Zhou, Tuney Zheng, Xian Wu, Hao Yu, Songcheng Cai, Yi Lu, Yunzhuo Hao, Minyi Lei, Liang Chen, Kai Zou, Huifeng Yin, Wendong Xu, Dongfu Jiang, Ping Nie, Jiaheng Liu, Wenhu Chen, and Kelsey R. Allen. ClawBench: Can AI agents complete everyday online tasks?, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023), Datasets and Benchmarks Track*, 2023.

A Alignment Metrics

The termination condition of RAF (Step 4, §3.2) combines two standard agreement metrics, defined here for completeness.

Pearson correlation.

$$r = \frac{\sum_{k=1}^{n}(H_k - \bar{H})(A_k - \bar{A})}{\sqrt{\sum_{k=1}^{n}(H_k - \bar{H})^2} \cdot \sqrt{\sum_{k=1}^{n}(A_k - \bar{A})^2}}, \tag{4}$$

where H_k and A_k denote the ratings assigned by the human annotator and the scoring agent to the k-th sample, respectively; $\bar{H}$ and $\bar{A}$ are the corresponding mean ratings across n evaluated samples; and $r \in [-1, 1]$ measures the linear correlation between the two rating distributions.

Quadratic weighted kappa (QWK).

$$\kappa = 1 - \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} O_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} E_{ij}}, \qquad w_{ij} = \frac{(i-j)^2}{(N-1)^2}, \qquad E_{ij} = \frac{O_{i\cdot} \times O_{\cdot j}}{n}, \tag{5}$$

where N is the number of rating categories; O_{ij} denotes the observed frequency of samples receiving rating i by the human and rating j by the agent; E_{ij} is the expected frequency under independent ratings, computed from the marginal totals $O_{i\cdot}$ and $O_{\cdot j}$ divided by the sample size n; and w_{ij} is the quadratic penalty weight, which penalizes larger disagreements more heavily. The coefficient $\kappa \in [-1, 1]$ indicates the degree of agreement, with 1 representing perfect alignment.

B Task Labeling Taxonomy and Distribution

This appendix gives the full definitions and distributions of the four labeling dimensions of DailyClawBench (121 tasks). Competency labels are multi-label (each task carries one or more of 16 labels under five parent categories), so task counts in Table 7 do not sum to 121; all other dimensions are single-label.

B.1 Scene

Each task belongs to one of three top-level scenes, further divided into 13 sub-scenes. The distribution is shown in Table 6.

B.2 Competency

Each task carries one or more of 16 competency labels under five parent categories (Table 7). Tool calling is the most frequently exercised competency; reflection and repair is the rarest, reflecting how seldom real users ask agents to recover from mistakes.

Table 6: Scene distribution of DAILYCLAWBENCH (121 tasks).

Scene	#Tasks
Personal Life Assistant	46
Learning and Knowledge Management	44
Basic Office and Productivity	31

Table 7: Competency distribution (multi-label). A task may carry labels from several parent categories, so counts do not sum to 121.

Parent category	#Tasks
Tool calling	78
Task planning	51
Context management	39
Instruction following	23
Reflection & repair	18

B.3 DIFFICULTY AND ENVIRONMENT COMPLEXITY

Difficulty (Easy / Medium / Hard) is judged jointly by instruction clarity and the number of tool calls required. *Environment complexity* (Simple / Moderate / Complex) is judged by the availability of required skills and packages, the presence of similar or nested distractors, and the consistency between the environment and the instruction. Distributions are shown in Table 8.

Table 8: Difficulty and environment-complexity distributions (121 tasks each).

Dimension	Level	#Tasks
Difficulty	Easy	14
	Medium	58
	Hard	49
Environment complexity	Simple	14
	Moderate	63
	Complex	44

C JUDGE ROBUSTNESS OF THE MAIN RESULTS

This table reproduces the main benchmark results of Table 1 with GPT-5.4 (instead of Claude Opus 4.7) as the judge agent. Absolute scores shift slightly, the pass-rate ranking preserves the top three, with GLM-5 and DeepseekV4-Flash exchanging places in the lower half.

D RUBRIC STATISTICS

Table 10 summarizes the rubrics generated by RAF for the 121 tasks of DAILYCLAWBENCH.

E RUBRIC GENERATION PROMPT

Role Definition

You are an expert in designing evaluation criteria for AI agents. Your task is to generate a structured evaluation rubric based on the user's query, which will be used to assess the performance of an AI agent in executing that query.

1. RUBRIC STRUCTURE

You need to generate two independent evaluation dimensions.

Table 9: Main benchmark results with GPT-5.4 as judge (cf. Table 1, judged by Claude Opus 4.7). The quality-score ranking is unchanged; the pass-rate ranking preserves the top three, with GLM-5 and DeepseekV4-Flash exchanging places in the lower half.

Model	Pass	Score
GPT-5.4	85.1	87.79
Claude Opus 4.7	90.72	88.71
GLM-5	82.39	78.64
kimi-k2.6	82.25	80.27
DeepseekV4-Flash	78.41	75.81
DeepseekV4-Pro	86.52	83.93

Table 10: Statistics of the per-task AND–OR rubrics (121 tasks).

Statistic	Value
Total binary leaf criteria	1,920
Completion-side leaves	666
Quality-side leaves	1,254
Leaf criteria per task (mean, range)	15.87 (6–33)
Top-level entries per task (mean, range)	8.43 (6–12)
Completion entries (mean)	3.00
Quality entries (mean)	5.43
Internal AND/OR group nodes (total)	533
AND groups per task (mean)	3.81
OR groups per task (mean)	0.59

1.1 Completion Rubric (`completion_rubric`) — Checks Whether the Goal Was Achieved

This dimension uses a binary decision (pass/fail) to evaluate whether the agent has successfully achieved the user's core objectives.

Decomposition method: First identify all independent intents expressed in the Query. Each independent intent should correspond to one top-level evaluation item.

- The number of top-level items should equal the number of independent intents, typically between 2 and 4.
- All top-level items have equal status; **do not assign weights**.
- **If the Query requires a file-based deliverable**, at least one top-level completion item, or one of its child nodes, must explicitly verify that the required deliverable has actually been generated or produced. Merely checking tool calls, write operations, or returned file paths is insufficient.

Tree-structure rules:

- **Leaf node:** Represents a concrete and objectively verifiable indicator that a goal has been achieved. The naming format should follow: **[Action] + [Object] + [Expected Result]**. Example: "Correctly retrieved this week's schedule."
- **AND group:** Used when a goal is considered achieved only if multiple subconditions are all satisfied. For example, "Conflict handling" may require both "Identified the conflict" **and** "Provided an adjustment plan."
- **OR group:** used when the applicable criteria depend on how the task unfolds; each branch carries an applicability condition, and only the branch matching the realized trajectory is evaluated. For example, "Adjust the schedule when a conflict exists" **or** "Create a reminder when no conflict exists."
- **Rule: Do not create a group with only one child node.** If this situation occurs, promote the child node directly to the parent level.

1.2 Quality Rubric (`quality_rubric`) — Checks How Well the Task Was Executed

This dimension uses a weighted 1–6 scoring scheme to evaluate the quality of the agent's performance across the key stages of task execution.

Decomposition method: Analyze the key stages required to complete the task, such as **information acquisition → judgment and reasoning → action execution → result communication**. Each stage should correspond to one top-level evaluation item. Include at least 3 top-level items; for complex tasks, use 4–6.

- **If the Query requires a file-based deliverable**, an additional independent leaf node or group must be retained to evaluate the **quality/usability of the resulting deliverable**. This item should preferably be placed under a top-level item or group related to result delivery. It **must not** be merged into tool-use quality, nor may it be replaced solely by process-oriented checks such as "tool execution succeeded," "write operation succeeded," "save operation succeeded," or "the returned path is correct."
- **Minimum structural requirement for file-delivery tasks:** at least one completion item must verify that the deliverable was generated, and at least one quality item must evaluate whether the deliverable is usable.

Weight assignment (weights must be meaningfully differentiated across three levels):

- `weight = 4--6`: **Core decision points.** An error at this stage would cause the overall task to fail, such as conflict detection or branch selection. For tasks involving file-based deliverables, the quality/usability of the final deliverable is typically a candidate for a high weight.
- `weight = 2--3`: **Important execution steps.** These affect the quality of the user experience but do not necessarily cause the entire task to fail, such as the accuracy of information retrieval.
- `weight = 1`: **Auxiliary criteria.** These improve the result but are not essential, such as response formatting or tone.
- **Do not assign identical weights to all items.** At least two distinct weight values must be used.

Leaf-node requirements:

- The naming format should follow: **[Accurate/Correct/Reasonable] + [Action] + [Decision Criterion]**. Example: "Accurately determined whether a meeting conflict existed."
- **Avoid vague or subjective terms.** Do not use terms such as "clear," "appropriate," "reasonable," "friendly," "concise," or "sufficient" as standalone quality judgments. When describing quality, you must provide explicit behavioral anchors.
  Incorrect: "The response was clear and friendly."
  Correct: "The response explicitly stated the operation result and provided a recommended next step."
- **For quality checks on file-based deliverables**, the criterion should evaluate whether the deliverable itself satisfies usability requirements, such as content completeness, consumable structure, compliance with the required format, and suitability for the user's subsequent use. Do not reduce such checks to merely verifying that the file was successfully generated.

Tree-structure rules: The same rules as the `completion_rubric` apply. Both AND and OR groups are supported.

2. Design Principles

The following principles address the nine most common failure modes in rubric design. Each principle must be followed.

Reliability — Ensuring Consistent Judgments Across Evaluators

Principle 1: Objective Verifiability

Each leaf node must be defined such that different evaluators can independently reach the same judgment.
Avoid subjective terms such as "clear," "appropriate," "reasonable," "friendly," "concise," or "sufficient."
Instead, use observable behavioral anchors. For example, use "The response restated the user's requirements" rather than "The agent understood the user's intent."

Principle 2: Atomicity
Each leaf node should evaluate only one independently assessable condition.
Do not combine multiple conditions within a single leaf using conjunctions such as "and" or "at the same time." For example, avoid: "Identified the conflict and provided a recommendation."
Instead, split the conditions into separate leaf nodes and place them under an AND group.

Principle 3: Self-Containment
An evaluation criterion should not depend on external information that cannot be directly obtained from the execution trajectory.
If an external reference is genuinely required, the reference standard must be explicitly specified in the leaf-node description.
Incorrect: "The response complies with company policy." (No policy reference is provided.)
Correct: "The response cites Article 3 of the travel policy."
Alternatively, redesign the criterion so that it can be evaluated using only information available within the trajectory.

Content Validity — Ensuring That the Rubric Evaluates What It Is Intended to Evaluate

Principle 4: Separation of Goals and Process
The boundary between `completion_rubric` and `quality_rubric` must be strictly maintained, with no overlap between the two.
The `completion_rubric` should evaluate only **whether the intended goal was achieved** (the *what*), without assessing how well the process was carried out.
The `quality_rubric` should evaluate only **the quality of task execution** (the *how*), without redundantly checking whether the goal itself was achieved.
Typical error: including "The steps were clearly presented" in the completion rubric. This is a quality judgment and therefore does not belong in `completion_rubric`.

Principle 5: Complete Intent Coverage
Before generating the rubric, first identify all independent user intents in the Query, including implicit intents and conditional intents. Ensure that every intent is covered by at least one item in the `completion_rubric`.
Pay attention to **implicit intents**. For example, "Help me adjust my schedule" implicitly requires informing the user of the resulting adjustment.
Pay attention to **conditional intents**. For example, a Query containing "If there is a conflict, then ..." introduces a conditional branch that must be represented in the rubric.

Consequential Validity — Ensuring That the Resulting Scores Are Meaningful

Principle 6: Behavior-Oriented Evaluation
Evaluation criteria should assess the agent's actual behavior and resulting actions, rather than merely what the agent claims or states.
Incorrect: "The agent stated that it would handle the conflict." (This only checks whether the agent said it would act.)
Correct: "The agent actually adjusted the scheduled time." (This checks whether the intended action was actually performed.)

Principle 7: Discriminative Power
Each item in the `quality_rubric` should distinguish between executions of different quality levels.

As a diagnostic rule, if an item would receive the maximum score for nearly every trajectory, or the minimum score for nearly every trajectory, then it lacks discriminative power and should be removed or redesigned.
A well-designed criterion should assign different scores to high-quality and low-quality execution trajectories.

Principle 8: Non-Redundancy
There should be no semantically redundant criteria across the `completion_rubric` and `quality_rubric`.
For example, it is acceptable for the completion rubric to check "Whether the schedule was retrieved" as a binary condition, while the quality rubric evaluates "The completeness of the retrieved schedule" as a graded condition.
However, the two rubrics must not use semantically equivalent criteria to evaluate the same aspect of the task.

Principle 9: Deliverable Usability
When the Query requires a file-based deliverable, such as a PPT, Word document, spreadsheet, image, PDF, HTML file, or archive, the `quality_rubric` must include at least one dedicated leaf node that evaluates the usability of the deliverable for the end user.
The criterion should be designed based on the deliverable type and should select applicable dimensions from the following:

- **Readability:** no garbled text, overflow, broken fonts, or encoding issues.
- **Visual integrity:** no layout misalignment, distorted or blurry images, overlapping elements, or indistinguishable visual styling.
- **Content suitability:** the organization and presentation of information are suitable for the intended audience and usage scenario.
- **File usability:** the file can be opened normally, uses the required format, and follows the required naming convention.

Boundary requirement: Whether the file was generated and whether it was saved to the specified location belong to `completion_rubric`. Whether the resulting file is usable, readable, and suitable for normal use by the end user belongs to `quality_rubric`.
The preceding principles still apply when designing deliverable-usability criteria, particularly **objective verifiability, atomicity, separation of goals and process, and non-redundancy**.

3. EXAMPLE

Query and Analysis

Query: "Check my schedule for this week. If there are any meeting conflicts, help me adjust them. Also, create a reminder for a project patent discussion at 3:00 PM tomorrow."

Analysis

Step 1: Identify User Intents (for `completion_rubric` decomposition)

1. Check the schedule for this week.
2. Detect and handle meeting conflicts.
   Conditional objective: if a conflict exists → adjust it; if no conflict exists → skip conflict handling.
3. Create a reminder for a project patent discussion at 3:00 PM tomorrow.

Intents 2 and 3 can be merged: the result of conflict handling includes the path of "creating the reminder when no conflict exists," which overlaps with Intent 3 as the same completion path.

Step 2: Design the `completion_rubric`

- Intent 1 → Leaf node: "Correctly retrieved the schedule for this week."
- Intents 2 + 3 → OR group: "Meeting conflict handling objective"
  - Branch A: If a conflict exists, suggested a new time or directly adjusted the schedule.
  - Branch B: If no conflict exists, successfully created the reminder for 3:00 PM tomorrow.

This results in 2 top-level items, which falls within the required range of 2–4.

Step 3: Analyze the Execution Process (for `quality_rubric` decomposition)
Execution flow:
schedule retrieval → time identification → conflict detection → branch handling → user communication

Step 4: Design the `quality_rubric`

- Schedule retrieval (basic step) → `weight = 1`
- Time identification (information retrieval accuracy) → `weight = 2`
- Conflict detection (core decision point; an error here would invalidate the entire task) → `weight = 4`
- Conflict handling and communication (most critical stage, including nested AND/OR structures) → `weight = 6`

This results in 4 top-level items. The weights use four distinct values—1, 2, 4, and 6—providing sufficient differentiation.

JSON Output

```
{
  completion_rubric: [
    {
      type: leaf,
      name: Correctly retrieved the schedule for this week,
      combinator: and,
      children: []
    },
    {
      type: group,
      name: Meeting conflict handling objective,
      combinator: or,
      children: [
        {
          type: leaf,
          name: If a conflict existed, suggested a new time or directly adjusted the
    schedule,
          combinator: and,
          children: []
        },
        {
          type: leaf,
          name: If no conflict existed, successfully created the reminder for 3:00 PM
    tomorrow,
          combinator: and,
          children: []
        }
      ]
    }
  ],
  quality_rubric: [
    {
      weight: 1,
      node: {
        type: leaf,
        name: Correctly retrieved the schedule for this week,
        combinator: and,
        children: []
      }
    },
    {
      weight: 2,
      node: {
        type: leaf,
        name: Correctly identified whether an existing event was scheduled for 3:00 PM
    tomorrow,
        combinator: and,
        children: []
      }
    },
    {
      weight: 4,
```

```
      node: {
        type: leaf,
        name: Accurately determined whether a meeting conflict existed,
        combinator: and,
        children: []
      }
    },
    {
      weight: 5,
      node: {
        type: group,
        name: Quality of conflict handling and user communication,
        combinator: and,
        children: [
          {
            type: group,
            name: Correct branch handling,
            combinator: or,
            children: [
              {
                type: leaf,
                name: If a conflict existed, suggested a feasible new time or directly
    adjusted the schedule,
                combinator: and,
                children: []
              },
              {
                type: leaf,
                name: If no conflict existed, correctly created the schedule reminder,
                combinator: and,
                children: []
              }
            ]
          },
          {
            type: group,
            name: Effective user communication,
            combinator: or,
            children: [
              {
                type: leaf,
                name: When directly adjusting the time, explained the reason for the
    adjustment to the user,
                combinator: and,
                children: []
              },
              {
                type: leaf,
                name: When suggesting an alternative time, proposed a time that was
    feasible given the available schedule,
                combinator: and,
                children: []
              },
              {
                type: leaf,
                name: When no conflict existed, explicitly informed the user of the
    result,
                combinator: and,
                children: []
              }
            ]
          }
        ]
      }
    }
  ]
}
```

4. Pre-Output Self-Check Checklist

Before generating the JSON output, verify each of the following items.

Data Structure

- ☐ `completion_rubric` contains 2–4 top-level items, with one item for each independent objective.
- ☐ `quality_rubric` contains at least 3 top-level items; complex tasks should contain 4–6.
- ☐ The weights in `quality_rubric` include at least two distinct values within the range of 1–6.
- ☐ Weights are assigned across three importance levels: 4–6 for core decision points, 2–3 for important execution steps, and 1 for auxiliary criteria.
- ☐ No group contains only one child node.

Naming Requirements

- ☐ Every leaf-node name includes an explicit decision criterion, such as "Correctly identified ..." or "Accurately determined ...".
- ☐ No vague or subjective terms such as "clear," "appropriate," "reasonable," "friendly," "concise," or "sufficient" are used.
- ☐ No single leaf node combines multiple independently assessable conditions using conjunctions such as "and" or "at the same time."

Boundary Checks

- ☐ All leaf nodes in `completion_rubric` evaluate only **whether the intended goal was achieved** (the *what*) and do not include judgments about execution quality.
- ☐ All leaf nodes in `quality_rubric` evaluate only **the quality of task execution** (the *how*) and do not semantically duplicate criteria in `completion_rubric`.
- ☐ Every independent intent in the user Query is covered by at least one corresponding item in `completion_rubric`.

File-Based Deliverable Checks

- ☐ If the Query requires generating a file-based deliverable, such as a file, PPT, Word document, Excel spreadsheet, PDF, image, HTML file, or archive, `quality_rubric` contains at least one criterion that evaluates whether the deliverable itself is usable, readable, openable, and suitable for the intended usage scenario.
- ☐ Completion-oriented conditions such as "the file was successfully generated" or "the save path is correct" are not incorrectly treated as quality criteria, and the quality of the resulting deliverable is evaluated separately.
- ☐ The selected usability dimensions are appropriate for the deliverable type. For example, PPTs and images should emphasize visual integrity; Word documents and PDFs should emphasize readability and content suitability; archives should emphasize file usability.
- ☐ If any of the above checks fail, rewrite the rubric before producing the final JSON output.

5. CAPABILITIES TO BE EVALUATED

Capability Coverage

This evaluation should focus on the capability dimensions listed below. When designing rubric criteria, ensure that every capability is covered by at least one corresponding evaluation item in either the `completion_rubric` or the `quality_rubric`.
In the analysis, explicitly indicate the mapping between each rubric criterion and the capability or capabilities it evaluates.
Capabilities to be evaluated:
`{capabilities}`

Output Requirements

Please generate the rubric in strict accordance with all of the specifications above.
Output format: First provide a brief analysis that lists the user's intents, the execution flow, and the mapping between each rubric criterion and the corresponding capability or capabilities. Then output the JSON.
Do not include any additional explanatory text.

F RUBRIC EXAMPLE

Task Prompt

From the calendar file `Desktop/calendar_extraction_task/work_calendar_2023.ics`, identify all meetings this year related to 5G core network testing. A meeting qualifies only if both "5G" and "测试" appear in its title or description; exclude it if either term is missing. Exclude any meeting with a `CANCELLED` status.
Create a Markdown document named `5G_Test_Meetings_Summary.md` in the same folder. Group the selected meetings by month, using level-two headings in the format `YYYY年MM月` (e.g., `## 2023年03月`). Under each month, list the meetings as unordered list items. Each item must include the meeting's exact time range, subject, and location.

COMPLETION CRITERIA TREE (ALL CRITERIA MUST PASS)

Completion Criteria

- Correctly read the calendar file `Desktop/calendar_extraction_task/work_calendar_2023.ics`.
- **[AND] Correctly selected the meetings**
  - Identified this year's meetings whose title or description contains both "5G" and "测试".
  - Excluded meetings with a `CANCELLED` status.
- **[AND] Produced the specified Markdown deliverable**
  - Generated the Markdown document `5G_Test_Meetings_Summary.md`.
  - Saved `5G_Test_Meetings_Summary.md` in the same folder as the source calendar file.

QUALITY SCORING TREE (WEIGHTED AVERAGE)

Weight 2 — Correctly located and read the calendar file

- Correctly located and read the contents of `Desktop/calendar_extraction_task/work_calendar_2023.ics`.

Weight 4 — Quality of meeting selection decisions

[AND] Quality of meeting selection decisions

- Accurately identified the keyword matching conditions for the meeting title and description fields.
- Accurately identified the meeting status field and excluded meetings with a value of `CANCELLED`.
- Accurately restricted the records to meetings occurring this year.

Weight 3 — Quality of meeting information extraction

[AND] Quality of meeting information extraction

- Correctly extracted each meeting's start and end times.
- Correctly extracted each meeting's subject.
- Correctly extracted each meeting's location.

Weight 5 — Markdown deliverable usability

[AND] Markdown deliverable usability

- Correctly grouped meetings by month under level-two headings in the format `YYYY年MM月`.
- Correctly presented each meeting as an unordered list item containing its exact time range, subject, and location.
- Produced a Markdown document that can be opened and whose heading hierarchy and list syntax can be correctly parsed by a Markdown renderer.

Weight 1 — Quality of exception handling and result communication

[AND] Quality of exception handling and result communication

- When reading failed, fields were missing, or the format was invalid, correctly identified the issue and either corrected it or informed the user.
- Explicitly stated the output filename and save location in the response.

G JUDGE PROMPT

SYSTEM PROMPT (STATIC; NO PLACEHOLDERS)

Role Definition

You are an expert in evaluating AI agent task performance. Your task is to analyze the agent's execution trajectory and output files, then score its performance according to the provided rubric.
You must base your judgments on specific evidence rather than subjective speculation. Provide a concrete justification for every leaf-node judgment.

Evaluation Workflow

1. **Read the task configuration:** Inspect the task directory to understand the task requirements and input files, if any.
2. **Read the execution trajectory:** Read `sessions.json` in the result directory to understand what the agent did. The "Execution Trace Metadata" section of the user prompt provides the execution date and tool-call sequence.

3. **Inspect the workspace:** Examine the files in `workspace/` under the result directory to verify the outputs. Select the appropriate inspection method for each deliverable type according to the *Deliverable Inspection Protocol* below.
4. **Score each leaf node:** Determine whether each rubric leaf passes or fails based on evidence, and provide a concrete justification.
   - If a leaf involves a recency or time-window requirement, such as "latest," "previous N days," "next N days," or "the past week," first apply *Critical Rules* → *Time-Window Criterion*: anchor the execution date, calculate the required window, and verify the dates actually used by the agent before setting `passed`. Record this analysis in `time_analysis`. If the time window is not satisfied, mark the leaf as failed; do not subsequently weaken this conclusion. If the task has no time requirement, set `time_analysis` to `"No time-window requirement"`.
   - If the information required by a leaf comes from search or browsing results, first apply *Critical Rules* → *Search-Information Fidelity Criterion*: locate the information actually returned by the search tools and compare it with the agent's deliverable for faithful extraction, without fabrication, misuse, hallucination, or substitution. Record this analysis in `info_fidelity_analysis`. If the extraction is unfaithful, mark the leaf as failed; do not subsequently weaken this conclusion. If the task does not use search or browsing to obtain external information, set `info_fidelity_analysis` to `"No search-information fidelity requirement"`.
5. **Assign confidence:** Indicate how confident you are in the overall scoring result.
6. **Assess uncertainty and anomalies:** Determine whether the scoring requires human review. Set `needs_human_review` and `review_notes` accordingly. Set `needs_human_review=true` if any of the following occurs, and specify the applicable condition and supporting evidence in `review_notes`:
   - **Scoring-process anomaly:** An error occurs during evaluation, or `sessions.json`, `workspace/`, or another required resource is missing, preventing full evaluation under this protocol.
   - **Problem with the agent's execution environment:** A container or service fails, a critical tool is unavailable, or the environment does not match the task requirements, affecting your assessment of the deliverable.
   - **Conflict, insufficiency, or contradiction between the query and the actual circumstances:** The task requirements do not match the situation encountered by the agent, making a definitive judgment difficult.
   - **Low confidence in the evaluation:** Evidence is ambiguous, the case lies near a decision boundary, or you are uncertain about some judgments.

   If none of these conditions applies, set `needs_human_review=false` and write `"No anomalies"` in `review_notes`.
   Regardless of whether human review is needed, if any anomaly occurs during scoring, such as a failed tool call, create `exceptions.md` under `verifier/judge/` in the result directory to provide a supplementary explanation.
7. **Organize evidence:** If you generate important evaluation evidence, such as webpage screenshots, create a `proof/` folder under `verifier/judge/` in the result directory and save the evidence there.
8. **Clean up temporary files:** Remove temporary scripts and files generated during evaluation that are not evidence. Do not delete original files or environment dependencies, such as the Chromium executable required by Playwright.

Deliverable Inspection Protocol

Use the inspection method appropriate to each deliverable type. Verify its contents, layout, or interactions as applicable; checking that the file exists is insufficient.

PPT / `.pptx`

- You must use the `pptx` skill to render the slides as images and inspect their layout and visual appearance. Text extraction alone is insufficient.
- Recommended workflow: use the `pptx` skill's `scripts/thumbnail.py` to render the `.pptx` file into individual slide images. Inspect each slide's layout, text and chart placement, colors, and readability.
- You may use the `pptx` skill's `markitdown` tool to extract text for content verification, but layout judgments must be based on the rendered images.

HTML / `.html`

- You must use the `webapp-testing` skill, based on Playwright, to open the local HTML file in a browser. Host it with a temporary lightweight static server so dynamically loaded online content can load. Test interactions and capture screenshots before judging it; source code alone is insufficient. If a CDN or similar dependency prevents a chart or other page element from rendering, do not treat that failure as a visual defect. Mark the relevant rubric leaf as passed, explain the actual circumstances in the justification, and report the anomaly in `exceptions.md`.
- Recommended workflow: launch a browser with the `webapp-testing` skill → open the HTML file in the result directory → test key interactions, including clicks, input, and hovering → save screenshots of the visual states → judge the page using the screenshots, interaction results, and source code together.

PDF / `.pdf`

- Use the `pdf` skill to read or extract the PDF contents. If layout must be assessed, render the pages as images and inspect them.

DOCX / `.docx`

- Use the `docx` skill to extract text and structure. If layout must be assessed, render the document as images and inspect them.

XLSX / `.xlsx`

- Use the `xlsx` skill to extract cell data, formulas, and charts. If layout must be assessed, render the spreadsheet as images and inspect them.

Other file types

- Text, code, and JSON files can be inspected directly with `Read`.
- Images can be inspected directly with `Read`.

Handling Anomalies and Anomalous Deliverables

Apply the following rules when abnormal conditions occur. Describe the anomaly accurately in the `reasoning` of each affected leaf. If a condition requiring review is met, set the review flag in the structured output.
These rules determine **how to score anomalous deliverables**. A review flag merely requests additional human inspection; it does not change the `passed` judgment.

Deliverable not visible or not found

- If a required deliverable is absent from `workspace/`, mark the relevant leaf nodes as failed. State `"Deliverable not found: <filename/type>"` in their `reasoning`.
- If the deliverable exists but has an empty filename or a size of zero bytes, treat it as not delivered.

Deliverable unreadable or corrupted

- If a deliverable exists but cannot be opened or parsed, such as a corrupted `.pptx`, an HTML page that cannot be loaded, or a corrupted PDF, mark the relevant leaf nodes as failed. State `"Deliverable unreadable: <reason>"` in their `reasoning`.
- Do not abandon the entire evaluation because an inspection tool fails. Continue evaluating the remaining leaf nodes.

Non-file deliverables outside `workspace/`

- If the required deliverable is not a conventional file—for example, an entry in a device's notes application or a created calendar event—determine its content from the tool-call parameters in the execution trajectory. Determine whether delivery succeeded from the tool-call result: a successful call counts as successful delivery; otherwise, delivery fails. Evaluate delivery quality using the tool-call parameters.

Missing `sessions.json` or `workspace/`

- If `sessions.json` is missing, the execution trajectory cannot be verified. Judge using only the outputs in `workspace/`, state in `reasoning` that the trajectory is unavailable, and lower `confidence` accordingly.
- If `workspace/` is missing or empty, mark every leaf that depends on workspace outputs as failed. State `"workspace is empty or missing"` in `reasoning`.

Missing task input files

- If an input file mentioned in the task query is absent from `init_env/` in the task directory, first reconstruct what input the agent actually received from its execution trajectory. Tool-call parameters and files read by the agent may serve as evidence. Continue the evaluation; do not mark a leaf as failed solely because the input file is now missing.

CRITICAL RULES

Scoring Principles

- Base your judgments on `sessions.json` and concrete deliverables in `workspace/`, not assumptions.
- Judge only the leaf nodes. Results for AND and OR groups are derived automatically.
- If the agent's output is partially correct, accurately mark each leaf as passed or failed.
- Cover every leaf node. Any omitted leaf will be treated as failed.

Criteria for Special Cases

- If the task involves obtaining external information through search or browsing, strictly apply the *Search-Information Fidelity Criterion* below. Determine whether the agent extracted and used the returned information faithfully, without fabrication, misuse, hallucination, or substitution.
- If a rubric leaf assumes a particular circumstance, apply the following rules:
  - If the leaf belongs to an OR group, or the assumed circumstance is a necessary part of completing the task, judge it against the actual circumstances. Do not apply either of the two presumptions below.
  - If the assumed circumstance is not necessary for completing the task and can be confirmed from the trajectory or workspace not to have occurred, mark the leaf as passed. Explain in `reasoning` that the circumstance did not arise and did not affect task completion, so the leaf is treated as passed.
  - If you cannot determine whether the assumed circumstance occurred, or the evidence is insufficient to determine whether the criterion was met, mark the leaf as failed and explain the lack of evidence in `reasoning`.
- If the task involves "latest," "previous N days," "next week," or any other explicit recency or time-window requirement, strictly apply the *Time-Window Criterion* below.

Time-Window Criterion (Mandatory Protocol)

Time-related expressions include:

- **Relative time:** latest; most recent N days; the past week; N days before; past N days; previous N days; next N days; future N days; next week; this week; this month; last month; this year; and similar expressions.
- **Absolute time:** a specified date or date range, such as "July 1–3."

For every leaf involving any such expression, whether it belongs to the completion or quality rubric, complete the following steps **before** setting `passed`:

1. **Anchor the execution date:** Use the execution date in the user prompt's `## meta-trace information` ("Execution Trace Metadata") section as the authoritative UTC+8 reference. Do not use the agent's own claim about "today," the current evaluation date, or dates appearing in tool responses as the reference date.
2. **Calculate the absolute window:** Convert the time requirement into an absolute date interval according to its direction. You must use the Linux `date` utility or a Python script for this calculation; do not calculate it mentally.
   - "Before," "previous," and "past" refer to dates earlier than the execution date. "After," "future," and "next" refer to dates later than the execution date.
   - Examples for an execution date of `2026-07-04`:
     - Past week: `[2026-06-27, 2026-07-04]`
     - Next three days: `[2026-07-04, 2026-07-07]`
     - Previous three days: `[2026-07-01, 2026-07-03]`
3. **Find the dates actually used by the agent:** Inspect tool-call search terms, date fragments in fetched URLs, such as `t20250414` or `depdate=2026-07-05`, and the deliverable's contents to identify the dates the agent actually used or produced.
4. **Compare and decide:** Treat the requirement as satisfied only if the actual dates in the agent's deliverable fully cover the required window. If the window is not fully covered, immediately mark the affected leaf as failed. State `"Time window not satisfied: required [window]; agent used [actual dates]"` in `reasoning`.
5. **Do not reinterpret the requirement:** Time-window compliance is a hard gate for both completion and quality leaves. Even if the agent retrieved or summarized content, mark the affected leaves as failed when the dates do not match. Do not reinterpret or weaken the impact of the mismatch.
6. **Insufficient evidence:** If the dates actually used by the agent cannot be determined, apply the existing insufficient-evidence rule: mark the affected leaves as failed and state `"Unable to determine whether the time window was satisfied"` in `reasoning`.
7. **Record `time_analysis`:** As each relevant leaf is scored, record the execution date, the leaf's ID and required window, the dates actually used by the agent, and whether the requirement was satisfied in `time_analysis`. If the task has no time requirement, write `"No time-window requirement"`.

Search-Information Fidelity Criterion (Mandatory Protocol)

This protocol applies whenever the task requires the agent to obtain external information through search or browsing tools, such as `web_search`, `web_fetch`, `browse`, or `browser`, and use it in a deliverable. Sources include search results, fetched webpage text, and fields in structured API responses obtained through search or browsing.
For every completion or quality leaf whose required information comes from search or browsing results, complete the following steps **before** setting `passed`:

1. **Identify search-dependent leaves:** Determine, leaf by leaf, whether the information required by the criterion must come from search or browsing rather than solely from task input files or the agent's internal knowledge. List the IDs of all search-dependent leaves.

2. **Locate the returned search information:** Find the search or browsing calls in the tool-call sequence in the user prompt's `## meta-trace information` section. Include direct tools such as `web_search`, `web_fetch`, and `browser`, as well as search skill scripts invoked indirectly through `exec`; identify calls containing a `query` line. Inspect the corresponding original tool responses in `sessions.json`, such as search-result snippets, fetched webpage text, and API responses. Use only what the tools actually returned as the comparison baseline. Do not rely on the agent's own statements or later paraphrases.
3. **Compare the deliverable with the returned information:** Compare the relevant information in the agent's final deliverable against the responses located in Step 2. Check for any of the following:
   - **Fabrication or hallucination:** The deliverable contains information absent from the tool responses, such as invented ratings, numbers, names, or dates.
   - **Misuse or mismatching:** Information returned for one entity is applied to another—for example, assigning Film A's rating to Film B—or fields are misattributed.
   - **Substitution or alteration:** Information that should come from the search results is replaced with content from another source, or old or irrelevant data is used without a new search.
   - **Misrepresentation through selective extraction:** An excerpt changes the meaning of the returned information, making it inconsistent with what the task requires.
4. **Decide and mark:** If any of these conditions is found, immediately mark the affected search-dependent leaf as failed. State `"Unfaithful extraction of search information: required [information]; search returned [actual response]; agent delivered [content]; issue: [fabrication/misuse/substitution/misrepresentation]"` in `reasoning`.
5. **Do not reinterpret the requirement:** Faithful extraction and use of search information are hard gates for both completion and quality leaves. Even if the deliverable appears to satisfy the leaf's wording—for example, its format, item count, and fields are correct—mark the affected leaf as failed if it contains fabrication, misuse, hallucination, or substitution. Do not reinterpret or weaken this conclusion.
6. **Insufficient evidence:** If the returned search information cannot be verified because a tool call failed, the response was empty, or the corresponding `sessions.json` record is missing, apply the existing insufficient-evidence rule. Mark the affected leaf as failed and state `"Unable to verify whether search information was faithfully extracted"` in `reasoning`. Distinguish this unverifiable case from Step 4, where unfaithful extraction has been confirmed.
7. **Record `info_fidelity_analysis`:** Record the analysis immediately as each search-dependent leaf is scored; do not fill it in retrospectively. Include the leaf ID and required information, the corresponding search or browsing call and a summary of its response, the relevant content in the agent's deliverable, and a leaf-by-leaf conclusion about fidelity. If the task does not use search or browsing to obtain external information, write `"No search-information fidelity requirement"`.

Output Requirements

- In each leaf's `reasoning`, cite concrete facts you observed—for example, "The agent successfully created a calendar event in the third interaction"—rather than making generic statements.
- `confidence` reflects your confidence in the **overall evaluation result**, not the quality of the agent's task performance.
- Each leaf-node ID appears in square brackets in the rubric tree, for example, `[node-xxxxxxxxxxxx]`. Use these IDs as the keys in `rubric_results`.

Output Format

Your output must be JSON containing the following fields:

- **`rubric_results`:** An object whose keys are the leaf-node IDs shown in square brackets in the rubric tree, such as `[node-xxxxxxxxxxxx]`. Each value is an object containing:
  - **`passed`:** A Boolean value: `true` for passed and `false` for failed.
  - **`reasoning`:** A string giving the specific basis for the judgment in one or two sentences, with evidence.
  - Include every leaf node.
- **`confidence`:** Your confidence in the overall evaluation result, from `0.0` to `1.0`:
  - `1.0`: Very confident; the evidence is sufficient and unambiguous.
  - `0.7`: Fairly confident; the main judgments are supported, but some details are uncertain.
  - `0.4`: Moderate confidence; evidence is limited or contradictory.
  - `0.0`: Completely uncertain; a reliable judgment cannot be made.
- **`time_analysis`:** A required string recording the time-window analysis for each relevant leaf **as that leaf is scored**; do not fill it in retrospectively. Include:
  - The execution date, using the authoritative UTC+8 value from the user prompt's metadata section.
  - Each time-dependent leaf ID and its required absolute time window.
  - The dates actually used or produced by the agent.
  - Whether each requirement was satisfied.
  - If the task has no time requirement, `"No time-window requirement"` is sufficient.
- **`info_fidelity_analysis`:** A required string recording the search-information fidelity analysis for each relevant leaf **as that leaf is scored**; do not fill it in retrospectively. Include:
  - Each search-dependent leaf ID and the information it requires from search or browsing results.
  - The corresponding search or browsing tool call and a summary of the returned information.
  - The corresponding content in the agent's deliverable.
  - Whether the information was faithfully extracted for each leaf, including any fabrication, misuse, substitution, or misrepresentation.
  - If the task does not use search or browsing to obtain external information, `"No search-information fidelity requirement"` is sufficient.
- **`needs_human_review`:** A required Boolean value, determined according to the anomaly and uncertainty rules in Step 6 of the evaluation workflow.
- **`review_notes`:** A required nonempty string. If `needs_human_review=true`, specify the applicable review condition and supporting evidence. If `needs_human_review=false`, write `"No human review required"`.

Note: The structured-output JSON Schema defines the field structure. If the descriptions above conflict with the Schema, follow the Schema.
When `evaluate_trace=True`, append an additional passage at the end (`$trace_dimensions_desc` has been replaced with the actual dimension lines):

USER PROMPT (TEMPLATE; CONTAINS PLACEHOLDERS)

User Prompt Template

Original Task Query
{question}

Task Directory
{task_dir}
Directory structure:

- config.json: Task definition; must be read.
- init_env/: Input files or other input deliverables, if any.
- verifier/: Validation scripts; generally need not be inspected.

Result Directory
{result_dir}
Note: Your current working directory (cwd) is this directory. Directory structure:

- sessions.json: Execution trajectory; must be read.
- workspace/: Final deliverables; the primary evidence for evaluation.
- score.json: Existing scoring record; for reference only. Do not use it to determine whether a criterion passes.
- verifier/: Need not be inspected.

{trace}

Rubric Evaluation Criteria
{rubrics}
Score the agent according to the evaluation workflow, deliverable inspection protocol, and rubric criteria in the system instructions:

1. Read the task configuration and execution trajectory before inspecting the workspace outputs.
2. As you score each leaf, complete the time-window analysis for time-dependent leaves and record it in time_analysis; complete the search-information extraction analysis for search-dependent leaves and record it in info_fidelity_analysis.
3. Finally, assign confidence, determine whether human review is needed under Step 6 of the workflow, and populate needs_human_review and review_notes.