# RECURSIVE LLM-AS-A-JUDGE FOR LONG-HORIZON WEB-AGENT TRAJECTORIES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Recent advances in large language models (LLMs) have enabled web agents to perform increasingly complex, long-horizon tasks. Reliable evaluation of long-horizon web-agent trajectories requires effective evidence acquisition and joint reasoning over collected evidence. However, evidence acquisition is challenging when trajectories exceed the judge's context window, as exhaustive inspection is inefficient when evidence is sparse and fixed selection cannot adapt to uncertainties uncovered during evaluation. Moreover, evidence aggregation must then account for both cross-window dependencies and each rubric's logical structure. To address these challenges, we introduce RECURSIVEWEBJUDGE, an LLM-as-a-judge framework inspired by recursive reasoning that couples adaptive evidence acquisition with rubric-specific aggregation. The framework first partitions trajectories into context-compatible windows and decomposes rubrics into verifiable requirements. For each selected window, the judge records local assessments and factual notes for these requirements in a shared evidence pool, using accumulated evidence and unresolved requirements to guide subsequent window selection and stopping. It then reasons over the accumulated evidence across windows to derive a final verdict, applying AND when all requirements must hold and ANY when a single qualifying instance suffices. We evaluate RECURSIVEWEBJUDGE on AgentRewardBench, a benchmark of expert-annotated multimodal web-agent trajectories, using 3B, 7B, and 11B judge backbones to assess task success and non-progressing repetition. It improves average macro-F1 across task success and repetition detection by up to 26.6% over the strongest baseline, with its 7B judge surpassing full-trajectory Claude Code evaluation. Code is available at `https://anonymous.4open.science/r/recursive_judge-91BF`.

## 1 INTRODUCTION

As LLM agents become more capable, they are increasingly applied to complex, long-horizon tasks. In web browsing, benchmarks such as Odysseys feature realistic multi-site tasks that require sustained interaction and produce extended trajectories (Jang et al., 2026). Evaluating web agents involves assessing whether they fulfill the user's goal and identifying undesirable behaviors such as non-progressing repetition (Lù et al., 2025). When such assessments cannot be performed programmatically, language and vision-language models are commonly used as judges to evaluate agents from their recorded trajectories (He et al., 2024; Xue et al., 2025). However, as trajectories lengthen grows, reliable assessment requires identifying evidence distributed across many interaction steps and integrating it against multiple criteria. This raises a central research question: *How can LLM judges identify and aggregate rubric-relevant evidence for reliable evaluation across long-horizon web-agent trajectories?*

Efficiently locating rubric-relevant evidence is difficult in long-horizon trajectories. Long-horizon trajectories may exceed an LLM judge's context window, making single-call evaluation of the full trajectory infeasible. Retaining only the most recent observations reduces the input length but can discard relevant evidence from earlier steps (He et al., 2024). Sliding-window and chunk-based methods broaden coverage by evaluating the trajectory across multiple calls (He et al., 2025; Kale et al., 2025). However, exhaustively inspecting every segment is inefficient when relevant evidence is sparse. Recent work therefore preselects observations based on their relevance to the task or evaluation criteria (Xue et al., 2025; Rosset et al., 2026). Yet when selection is fixed before verification,



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

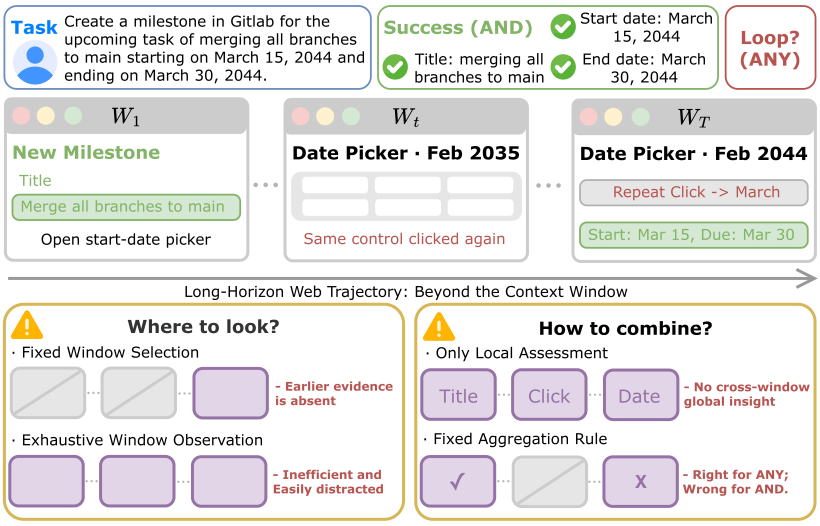



Figure 1: **Challenges in judging long-horizon web-agent trajectories.** In the illustrated GitLab milestone task, the agent eventually satisfies all title and date requirements but also exhibits a non-progressing repetition loop. Task success is conjunctive (AND), whereas loop detection is existential (ANY). Because the trajectory exceeds the judge's context window, fixed window selection may omit relevant earlier evidence, while exhaustive inspection is inefficient when evidence is sparse (left). Even after evidence is collected, independent local assessments cannot capture cross-window dependencies, and a fixed aggregation rule cannot accommodate both logical structures (right).

it cannot respond to uncertainties uncovered during assessment. This motivates an adaptive process in which unresolved rubric requirements guide which trajectory segment to inspect next.

Beyond identifying relevant evidence, reliable evaluation requires rubric-specific aggregation. Evaluation rubrics vary in logical structure, with some requiring all conditions to hold and others being satisfied by a single qualifying instance. Task-completion rubrics are often conjunctive, requiring all task-specific conditions to hold (Xue et al., 2025; Rosset et al., 2026), whereas behavioral rubrics such as non-progressing repetition are existential, requiring only one qualifying instance (Lù et al., 2025). This motivates an adaptive aggregation framework that accommodates the varying logical requirements of different rubrics.

To address these challenges, we introduce RECURSIVEWEBJUDGE, which couples adaptive evidence acquisition with rubric-specific aggregation. The framework builds on recursive reasoning, which decomposes complex problems and combines partial results (Zhang et al., 2025). It first partitions each trajectory into context-compatible windows and decomposes each rubric into verifiable requirements for subsequent evidence acquisition and aggregation. Evidence acquisition draws on reasoning-guided retrieval in multi-step question answering (Trivedi et al., 2023). The judge iteratively selects which trajectory window to inspect based on the rubric and the evidence accumulated so far. After each inspection, the resulting local assessment and factual note are consolidated into an entry in a shared evidence pool. The process stops once the collected evidence is sufficient to determine a verdict, thereby avoiding exhaustive inspection of the trajectory. The framework then proceeds to rubric-specific aggregation, which derives the final verdict from the collected evidence according to the rubric's logical structure. Many evaluation rubrics can be represented using conjunctive (AND) or existential (ANY) logic. RECURSIVEWEBJUDGE applies AND when all decomposed requirements must hold and ANY when a single qualifying instance suffices. Together, these mechanisms enable reliable evaluation of long-horizon web-agent trajectories by effectively identifying rubric-relevant evidence and aggregating it according to each rubric's logical structure.

Our contributions can be summarized as follows:

- We introduce RECURSIVEWEBJUDGE, a framework for reliable evaluation of long-horizon web-agent trajectories. It combines *trajectory windowing* and *rubric decomposition* to provide a structured foundation for subsequent evidence collection and aggregation.

- We develop *adaptive evidence acquisition* that uses accumulated evidence and unresolved rubric requirements to guide window selection and stopping. We further design *rubric-specific aggregation* that reasons over the collected evidence using AND or ANY logic according to each rubric's logical structure.
- We evaluate RECURSIVEWEBJUDGE on AgentRewardBench using 3B, 7B, and 11B judge backbones across different model families. Our method achieves the highest average macro-F1 across task success and repetition detection in all three backbone comparisons, with up to 26.6% relative improvement over the strongest baseline. The 7B configuration also surpasses full-trajectory Claude Code evaluation on this measure. Ablations show that adaptive evidence acquisition improves over endpoint-only evaluation and approaches exhaustive performance with fewer window inspections.

## 2 RELATED WORK

**Web Agent Evaluation.** Recent work uses language and vision-language models to evaluate web agents from their interaction histories, outputs, and supporting evidence. A growing line of work improves these judges by organizing evaluation around task requirements and the evidence needed to assess them. WebVoyager assesses task completion using task instructions, agent responses, and recent screenshots (He et al., 2024), while WebJudge identifies task key points and selects relevant screenshots (Xue et al., 2025). Universal Verifier combines explicit rubrics with criterion-conditioned screenshot selection and holistic assessment (Rosset et al., 2026), while Mind2Web 2 uses hierarchical rubrics and source-grounded verification to assess answer correctness and attribution (Gou et al., 2025). However, these approaches do not explicitly couple evidence acquisition guided by accumulated evidence and unresolved requirements with aggregation tailored to the logical structures of different rubrics.

**Long-Horizon Agent Evaluation.** As LLM agents become capable of sustained interaction, benchmarks increasingly evaluate them on long-horizon workflows. Such workflows often involve open-ended outputs and interaction-dependent requirements that are difficult to assess with programmatic or final-state checks alone, motivating model-based evaluation. Odysseys targets realistic multi-site web tasks (Jang et al., 2026), while GTA introduces compositional tasks requiring extended web interaction (Huang et al., 2026). Related evaluation settings extend to research and consumer services. DeepResearch Bench evaluates open-ended research tasks involving information gathering and synthesis (Du et al., 2025), while VitaBench examines agents' ability to satisfy multiple goals and evolving user requirements through sustained interaction (He et al., 2025). These settings often admit multiple valid solutions, requiring evaluation of output quality and requirement satisfaction across the interaction. Yet, REFLECT shows that LLM judges still struggle to detect process- and outcome-level failures in research agents (Wang et al., 2026), highlighting the need for more reliable judges to assess long-horizon agent performance.

**Recursive Decomposition and Inference.** Recent work has applied recursive reasoning to complex tasks by decomposing them into smaller subproblems and combining their solutions. In formal theorem proving, Hilbert recursively decomposes difficult theorems into subgoals and assembles their proofs under formal verification (Varambally et al., 2025). For long-context question answering and code repository understanding, Recursive Language Models externalize the input and invoke language models on programmatically selected subproblems (Zhang et al., 2025). Related ideas extend to combinatorial reasoning: Recursive Models solve subtasks in isolated contexts and return their results to the parent computation (Yang et al., 2026). At the agent level, Recursive Agent Optimization learns when and how to delegate subtasks to recursive instances of the same agent, with applications to deep research, document processing, and crafting tasks (Gandhi et al., 2026). These approaches provide a flexible foundation for decomposing evaluation requirements and adapting evidence acquisition and aggregation to different rubrics.

## 3 PRELIMINARIES

**Web Trajectory Definition.** We represent a web-agent trajectory for a task instruction $g$ as an alternating sequence of observations, rationales, and actions,

$$\mathcal{T} = \big(o_1, h_1, a_1, \ldots, o_{T-1}, h_{T-1}, a_{T-1}, o_T\big), \tag{1}$$

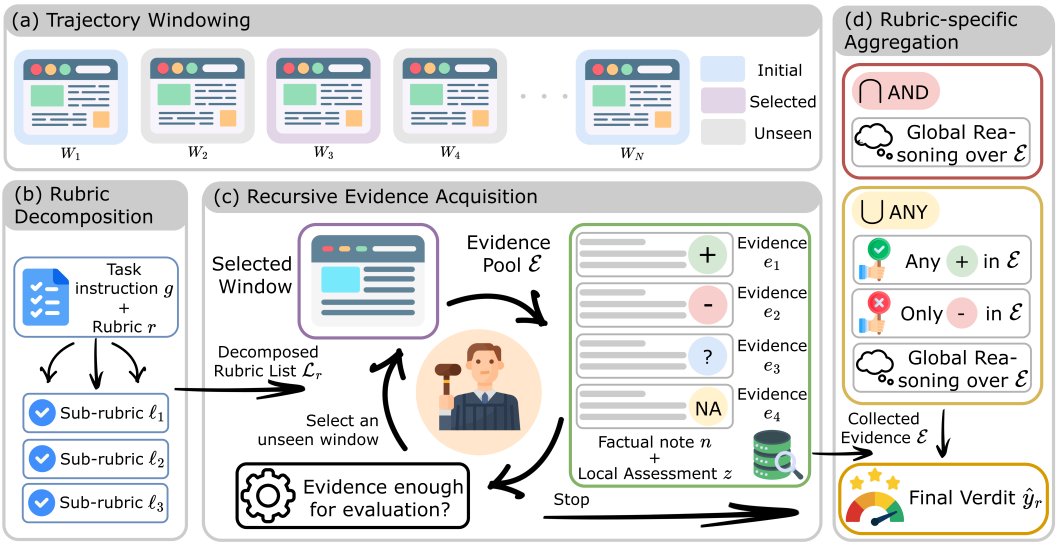



Figure 2: **Overview of RecursiveWebJudge for long-horizon web-agent evaluation.** **(a)** The trajectory is partitioned into contiguous windows, with the first and last inspected initially. **(b)** The rubric is decomposed into verifiable sub-rubrics that guide evidence collection. **(c)** The judge collects local assessments and factual notes in a shared evidence pool $\mathcal{E}$, using the accumulated evidence to select an unseen window or stop inspection. **(d)** Rubric-specific aggregation produces a trajectory-level verdict under the original rubric. AND uses global reasoning across the collected evidence to determine whether all requirements hold. ANY applies deterministic rules to relevant assessments, excluding NA, and resolves the remaining cases through global reasoning over $\mathcal{E}$.

where $o_t$ is a browser observation consisting of a screenshot and the available textual representations of the page, $a_t$ is a webpage-executable action, $h_t$ is the agent's recorded rationale for $a_t$, and $T$ is the final step. We write $x_t = (o_t, h_t, a_t)$ for the record of step $t < T$, and $x_T = (o_T)$ for the terminal observation.

**Problem Setup.** We consider web-agent trajectory evaluation as: given $g$, a trajectory $\mathcal{T}$, and an evaluation rubric $r$, a multimodal LLM judge $J$ predicts whether the trajectory satisfies the rubric,

$$\hat{y}_r = J(g, \mathcal{T}, r) \in \{0, 1\}, \tag{2}$$

where $\hat{y}_r = 1$ indicates satisfaction and $\hat{y}_r = 0$ otherwise. Long-horizon trajectories make this prediction difficult: $\mathcal{T}$ may exceed the context capacity of $J$, and the evidence relevant to a given rubric is typically sparse and scattered across the trajectory.

## 4 Method

We introduce RecursiveWebJudge, an LLM-as-a-judge framework for reliable evaluation of long-horizon web-agent trajectories. Inspired by recursive reasoning (Zhang et al., 2025), the framework decomposes the evaluation problem into manageable units that support targeted evidence acquisition and subsequent integration. The framework proceeds in four stages: (i) *trajectory windowing* partitions a long trajectory into contiguous windows that can be inspected separately; (ii) *rubric decomposition* splits a rubric into verifiable sub-requirements; (iii) *adaptive evidence acquisition* adaptively inspects windows to collect evidence for these sub-requirements; and (iv) *rubric-specific aggregation* synthesizes the collected evidence into a final judgment. Figure 2 illustrates the framework, and Algorithm 1 states the full procedure.

### 4.1 Trajectory Windowing

Presenting an entire long trajectory in a single call often exceeds the context window; even when it fits, redundant interactions can dilute evidence relevant to a given rubric. We therefore partition the step records into $N = \lceil T/w \rceil$ contiguous, non-overlapping windows $\mathcal{W} = (W_1, \ldots, W_N)$ of at

most $w$ steps each, and let the judge inspect one window per call. Each call presents the actions and rationales in the window alongside the raw multimodal observations. The corresponding prompt is shown in Table 3.

### 4.2 RUBRIC DECOMPOSITION

A rubric may contain multiple requirements whose evidence is distributed across windows, which makes a single holistic judgment difficult to ground. Conditioned on $g$, we prompt the judge (Table 4) to decompose $r$ into a set of verifiable sub-rubrics,

$$\mathcal{L} = \mathsf{Decompose}(g, r) = \{\ell_1, \ldots, \ell_M\}, \tag{3}$$

and set $\mathcal{L} = \{r\}$ if no separable requirements are identified. Sub-rubrics guide evidence collection but are not scored independently. Each window is examined for evidence relevant to all sub-rubrics in $\mathcal{L}$, while the final verdict is determined using the original rubric $r$. Decomposition makes the rubric's evidence explicit, allowing unresolved requirements to guide subsequent window selection.

### 4.3 ADAPTIVE EVIDENCE ACQUISITION

Evidence acquisition alternates between observing a window and deciding whether and where to inspect next based on the accumulated evidence. Let $\mathcal{E}$ denote the evidence pool accumulated so far, shared across all sub-rubrics, and $\mathcal{U}$ the indices of windows not yet inspected.

**Window Observation.** A single call assesses window $W_i$ against all $M$ sub-rubrics and returns one evidence record per sub-rubric using the prompt in Table 5,

$$e_i = \mathsf{Observe}(g, \mathcal{L}, W_i) = \big\{(i, j, z_{ij}, n_{ij})\big\}_{j=1}^{M}, \tag{4}$$

where $z_{ij} \in \{+, -, ?, \mathrm{NA}\}$ is a local assessment of $\ell_j$ on $W_i$ and $n_{ij}$ is a factual note. The labels `satisfies` $(+)$ and `not_satisfies` $(-)$ indicate that the window provides sufficient evidence to establish that $\ell_j$ holds or fails; `unclear` $(?)$ indicates relevant but insufficient or ambiguous evidence; and `not_relevant` $(\mathrm{NA})$ indicates no relevant evidence. The note $n_{ij}$ summarizes the observed actions together with any goal-relevant outcome, such as displayed values, returned answers, and state changes. These notes support cross-window reasoning about whether later actions preserve or undo earlier progress.

**Adaptive Inspection.** We first inspect $W_1$ and $W_N$ in separate calls and initialize $\mathcal{E} = e_1, e_N$. These windows establish the initial task context and observed end state, which guide the selection of intermediate windows. After each update to $\mathcal{E}$, the judge reviews the accumulated evidence and uses the control prompt shown in Table 6 to select an unseen window or terminate:

$$\mathsf{Control}(g, r, \mathcal{L}, \mathcal{E}, \mathcal{U}) \in \mathcal{U} \cup \{\mathsf{STOP}\}. \tag{5}$$

Upon selecting $W_i$, we obtain $e_i$ via Eq. 4, update $\mathcal{E} \leftarrow \mathcal{E} \cup e_i$, and remove $i$ from $\mathcal{U}$. The cycle repeats until the judge returns STOP, until $\mathcal{U} = \varnothing$. Conditioning window selection on the accumulated evidence can concentrate observation on windows that may affect the final verdict.

### 4.4 RUBRIC-SPECIFIC AGGREGATION

Evaluation rubrics impose different conditions for satisfaction. Task success, for example, may require all requested outcomes to be achieved, whereas non-progressing repetition can be established by a single qualifying instance. We therefore assign each rubric an aggregation operator, AND or ANY, depending on the conjuctive or existential nature of the rubric. The judge then derives the final verdict from the local assessments and factual notes stored in $\mathcal{E}$.

**AND Aggregation.** For AND rubrics, we prompt the judge (Table 7) to integrate the collected assessments and notes and determine whether all requirements hold:

$$\hat{y}_r = \mathsf{Decide}_{\mathrm{AND}}(g, r, \mathcal{E}). \tag{6}$$

The final judgment uses evidence from all inspected windows, including intermediate progress, failures, and subsequent recovery.

**ANY Aggregation.** An ANY rubric holds if a qualifying instance occurs anywhere. Let $\mathcal{Z} = \{z_{ij} \in \mathcal{E} \mid z_{ij} \neq \text{NA}\}$ denote the relevant local assessments. We exclude the NA evidence to prevents irrelevant windows from biasing aggregation toward a negative verdict. A single positive assessment is sufficient, and a negative verdict requires relevant evidence that is uniformly negative. Therefore, we first apply a deterministic rule and defer only the residual cases to the judge:

$$\hat{y}_r = \begin{cases} 1, & + \in \mathcal{Z}, \\ 0, & \mathcal{Z} = \{-\}, \\ \mathsf{Decide}_{\text{ANY}}(g, r, \mathcal{E}), & \text{otherwise}, \end{cases} \tag{7}$$

where $\mathcal{Z} = \{-\}$ holds only if $\mathcal{Z}$ is non-empty and contains no other label. If the relevant evidence remains inconclusive, the judge uses the full evidence pool and the aggregation prompt shown in Table 8 to determine the final verdict.

## 5 EXPERIMENT SETUPS

**Evaluation Datasets.** We evaluate RECURSIVEWEBJUDGE on AgentRewardBench (Lù et al., 2025), the only public web-agent benchmark we identified with expert annotations for both success and non-progressing repetition. Its expert annotations cover both task success and non-progressing repetition, enabling evaluation of outcome-based and behavioral judgments. The benchmark comprises 1,302 trajectories generated by web agents executing tasks in WebArena, VisualWebArena, AssistantBench, WorkArena, and WorkArena++. To study long-horizon evaluation, we select 813 trajectories with at least 15 steps and use the benchmark's expert annotations as ground truth.

**Evaluation Rubrics and Metrics.** We evaluate two benchmark rubrics that instantiate the aggregation rules in Section 4.4. *Success* asks whether the agent's action sequence achieves the task goal and uses AND, since all requested outcomes must be achieved. *Repetition* asks whether the agent cycles through actions without making progress toward the goal and uses ANY, since one qualifying cycle suffices. The positive rates are 10.8% for Success and 80.0% for Repetition. We exclude *Side Effect* due to its low positive rate (6.5%) and ambiguous definition. Given the opposing class imbalances, we report macro-F1 as the primary metric. COMB is the unweighted mean of the two rubric-level macro-F1 scores.

**Baselines.** We compare against five representative model-based evaluators that differ in how they select and aggregate evidence. All judges use the same backbone, rubric text, verdict labels, and output format, while all window-based methods use the same window size.

- **WebVoyager** (He et al., 2024) judges the task and final response from the last window.
- **WebJudge** (Xue et al., 2025) extracts task-completion key points, retains screenshots above a relevance threshold, and judges them with the full action history.
- **Universal Verifier** (Rosset et al., 2026) generates task-specific criteria, selects up to five screenshots per criterion, and holistically evaluates the extracted evidence.
- **Mind2Web 2** (Gou et al., 2025) evaluates final answers and cited sources using task-specific rubric trees that aggregate leaf-level checks.
- **TRACE** (Mittapalli et al., 2026) triages the full trajectory, adaptively investigates suspect windows while retaining cross-window evidence, and synthesizes the findings into a final verdict.
- **Claude Code (Full Trajectory)** (Anthropic, 2025) evaluates the complete trajectory against the task instruction and evaluation rubric, serving as a frontier-model reference.

**Implementation Details.** We evaluate Qwen2.5-VL-3B-Instruct, Qwen2.5-VL-7B-Instruct, and Llama-3.2-11B-Vision-Instruct, spanning two model families and three parameter scales (Bai et al., 2025; Meta, 2024). All models are served with vLLM through OpenAI-compatible endpoints with a 32K-token context window. Images are capped at $768 \times 768$ pixels. All window-based methods use non-overlapping windows of $w = 5$ steps, with each call receiving one window of actions, rationales, and multimodal observations. We use greedy decoding with temperature 0 and report means over three seeds, $\{0, 1, 2\}$. Variance persists because vLLM inference is not bitwise-deterministic (floating-point non-associativity under continuous batching can flip near-tied tokens), so each seed is effectively an independent run.

Table 1: **Overall performance on AgentRewardBench.** mF1 denotes macro-F1. COMB is the unweighted mean of the two rubric-level mF1 scores. Results are reported as mean±std over 3 random seeds. Within each model block, **bold** and underlining indicate the best and second-best results in each column, respectively.

| Method | Succ. mF1 | Rep. mF1 | COMB |
|---|---|---|---|
| *Claude Code (Opus 4.8)* | | | |
| Full Trajectory | 68.4 ±0.8 | 50.7 ±1.2 | 59.6 ±0.9 |
| *Llama-3.2-11B-Vision-Instruct* | | | |
| WebVoyager | 54.8 ±0.6 | 24.2 ±0.5 | 39.5 ±0.4 |
| WebJudge | 50.0 ±0.4 | 22.8 ±0.2 | 36.4 ±0.3 |
| Universal Verifier | 45.7 ±1.5 | 22.7 ±0.8 | 34.2 ±0.8 |
| TRACE | 36.2 ±0.5 | 19.3 ±0.8 | 27.7 ±0.6 |
| Mind2Web 2 | **56.7** ±0.6 | 47.9 ±0.3 | 52.3 ±0.3 |
| RECURSIVEWEBJUDGE (Ours) | 54.8 ±0.8 | **50.4** ±1.3 | **52.6** ±0.8 |
| *Qwen2.5-VL-3B-Instruct* | | | |
| WebVoyager | 54.7 ±0.8 | 23.3 ±0.7 | 39.0 ±0.7 |
| WebJudge | 32.6 ±0.1 | 20.1 ±0.2 | 26.3 ±0.2 |
| Universal Verifier | 48.3 ±1.0 | 24.2 ±1.6 | 36.3 ±0.4 |
| TRACE | 39.6 ±0.5 | 23.7 ±1.3 | 31.6 ±0.8 |
| Mind2Web 2 | 54.2 ±0.3 | 25.3 ±0.1 | 39.8 ±0.1 |
| RECURSIVEWEBJUDGE (Ours) | **57.4** ±1.2 | **37.7** ±0.5 | **47.5** ±0.8 |
| *Qwen2.5-VL-7B-Instruct* | | | |
| WebVoyager | 65.7 ±0.5 | 19.6 ±0.2 | 42.7 ±0.2 |
| WebJudge | 63.1 ±0.7 | 17.4 ±0.2 | 40.2 ±0.4 |
| Universal Verifier | 67.1 ±2.4 | 17.2 ±0.2 | 42.1 ±1.1 |
| TRACE | 62.1 ±0.6 | 19.1 ±0.3 | 40.6 ±0.3 |
| Mind2Web 2 | 61.1 ±1.0 | 39.6 ±0.4 | 50.4 ±0.4 |
| RECURSIVEWEBJUDGE (Ours) | **68.1** ±0.5 | **59.5** ±2.1 | **63.8** ±0.8 |

# 6 EXPERIMENTS

## 6.1 MAIN RESULTS

We compare RECURSIVEWEBJUDGE with five model-based evaluators on long-horizon trajectories using three judge backbones. Table 1 reports macro-F1 for Success and Repetition and their average, COMB, alongside a full-trajectory evaluation reference using Claude Code (Opus 4.8). Case studies are provided in Appendix C.4.

**RecursiveWebJudge achieves the highest COMB across all three backbones.** Our method obtains COMB scores of 47.5, 63.8, and 52.6 with the 3B, 7B, and 11B backbones, surpassing the strongest baseline for each backbone by 7.7, 13.4, and 0.3 points, respectively. The largest gain occurs with Qwen2.5-VL-7B-Instruct, where COMB increases from 50.4 to 63.8, corresponding to a 26.6% relative improvement.

**The largest and most consistent gains occur in detecting non-progressing repetition.** Repetition macro-F1 exceeds the strongest baseline by 12.4, 19.9, and 2.5 points at 3B, 7B, and 11B, respectively. For Success, our method improves macro-F1 by 2.7 and 1.0 points on the two Qwen backbones and achieves joint second-best performance on Llama-11B, with a 1.9-point gap to the best result. The stronger gains in repetition detection support the framework's effectiveness in evaluating global behavioral patterns across long-horizon trajectories.

**The 7B configuration exceeds the full-trajectory Claude Code reference in COMB.** RECURSIVEWEBJUDGE achieves a COMB score of 63.8 with the 7B backbone, compared with 59.6 for Claude Code, a gain of 4.2 points. The comparison highlights the competitiveness of the 7B configuration, particularly for behavioral evaluation.

## 6.2 ABLATION STUDY

We examine the roles of adaptive evidence acquisition and rubric-specific aggregation through three ablation variants. First+Last inspects only the initial and final windows, whereas Exhaustive inspects all windows without early stopping. AggFlip swaps the aggregation rules to assess the importance of matching aggregation to each rubric's logical structure. Table 2 summarizes the results.

**Adaptive inspection consistently outperforms endpoint-only inspection.** Relative to First+Last, RECURSIVEWEBJUDGE improves COMB by 6.0, 5.9, and 6.8 points on 11B, 3B, and 7B judges,

Table 2: **Ablation on RECURSIVEWEBJUDGE.** For *adaptive evidence acquisition*, First+Last inspects only the initial and final windows, whereas Exhaustive inspects all windows without early stopping. For *rubric-specific aggregation*, AggFlip swaps the aggregation rules, applying ANY to Success and AND to Repetition. We report macro-F1 (mF1), false positive rate (FPR), and false negative rate (FNR). COMB is the unweighted mean of the two mF1 scores. (seed 0)

| | Success | | | Repetition | | | |
|---|---|---|---|---|---|---|---|
| Variant | mF1 ↑ | FPR ↓ | FNR ↓ | mF1 ↑ | FPR ↓ | FNR ↓ | COMB ↑ |
| *Llama-3.2-11B-Vision-Instruct* | | | | | | | |
| RECURSIVEWEBJUDGE | 54.7 | 10.8 | 79.3 | 50.4 | 64.4 | 32.0 | 52.6 |
| First+Last | 51.0 | 8.4 | 89.7 | 42.3 | 31.2 | 62.3 | 46.6 |
| Exhaustive | 54.4 | 10.5 | 80.5 | 52.5 | 61.3 | 30.3 | 53.5 |
| AggFlip | 34.2 | 61.5 | 50.0 | 41.0 | 28.6 | 65.2 | 37.6 |
| *Qwen2.5-VL-3B-Instruct* | | | | | | | |
| RECURSIVEWEBJUDGE | 58.7 | 9.0 | 73.6 | 38.1 | 15.6 | 73.1 | 48.4 |
| First+Last | 56.5 | 9.8 | 77.0 | 28.6 | 11.2 | 85.9 | 42.5 |
| Exhaustive | 60.2 | 6.6 | 74.7 | 53.1 | 40.0 | 41.4 | 56.6 |
| AggFlip | 24.6 | 76.9 | 47.3 | 23.3 | 14.3 | 91.3 | 24.0 |
| *Qwen2.5-VL-7B-Instruct* | | | | | | | |
| RECURSIVEWEBJUDGE | 68.6 | 8.3 | 51.7 | 57.3 | 70.0 | 15.8 | 62.9 |
| First+Last | 63.8 | 8.4 | 63.2 | 48.5 | 36.9 | 50.4 | 56.1 |
| Exhaustive | 65.6 | 9.0 | 57.5 | 60.7 | 76.2 | 5.3 | 63.1 |
| AggFlip | 40.9 | 50.0 | 50.0 | 23.3 | 14.3 | 91.3 | 32.1 |

with macro-F1 gains on both rubrics across all backbones. The gains are larger on Repetition (+8.1 to +9.5) than on Success (+2.2 to +4.8), consistent with repetitive behavior manifesting in intermediate interactions that endpoint windows fail to capture. On Success, the improvement stems mainly from lower FNR (3.4–11.5 points), while FPR changes by at most 2.4 points, indicating that intermediate evidence helps recognize completions not evident from the endpoints alone.

**Adaptive inspection reduces inspection cost, with the accuracy gap concentrated on Repetition.** On Qwen-7B, RECURSIVEWEBJUDGE skips 9.4% of intermediate windows under Success and 18.8% under Repetition across the evaluation set, yet its COMB (62.9) recovers 97% of the gain of Exhaustive (63.1) over First+Last (56.1). On Success, it matches or exceeds Exhaustive on Llama-11B and Qwen-7B (54.7 vs. 54.4; 68.6 vs. 65.6) and remains within 1.5 points on Qwen-3B, suggesting that inspecting all windows yields little additional benefit for judging task completion. On Repetition, by contrast, Exhaustive is consistently better (by 2.1, 15.0, and 3.4 points on Llama-11B, Qwen-3B, and Qwen-7B), with lower FNR on every backbone, most notably on Qwen-3B (41.4 vs. 73.1).

**Aggregation rules must match the logical structure of each rubric.** Swapping the aggregation rules (AggFlip) causes the largest degradation among all variants, reducing COMB by 15.0, 24.4, and 30.8 points on each judges. The resulting errors shift toward the failure mode implied by each rule. Applying ANY to Success raises FPR by 50.7, 67.9, and 41.7 points, since a single window judged successful suffices to declare completion; applying AND to Repetition raises FNR by 33.2, 18.2, and 75.5 points, since loops are often confined to a portion of the trajectory yet must appear in every window to be flagged.

## 6.3 ADDITIONAL STUDIES

### 6.3.1 PERFORMANCE ACROSS TRAJECTORY LENGTHS

We group trajectories into three token-length bins: $< 39$K, 39–65K, and $\geq 65$K to investigate the performance across trajectory lengths. The results are reported in Fig. 3.

**Our method is less sensitive to trajectory length than most baselines.** From the shortest to the longest bin, RECURSIVEWEBJUDGE changes little in most settings. However, several baselines degrade substantially, most notably Trace and WebJudge on Llama-11B Success. Its clearest declines occur on Qwen-7B Success, where most baselines degrade even more, and on Qwen-3B Repetition, where the decline is comparable to those of Mind2Web 2 and Trace.

**Our method leads consistently on the longest trajectories.** In the $\geq$65K bin, RECURSIVEWEBJUDGE achieves the highest macro-F1 on all three backbones and both rubrics. Mind2Web 2 and WebVoyager fall behind at $\geq$65K on Success compared to our method, which is consistent with their greater sensitivity to length. On Repetition, our method leads in every bin on both Qwen backbones, while on Llama-11B, Mind2Web 2 is nearly matches our method in the longest.

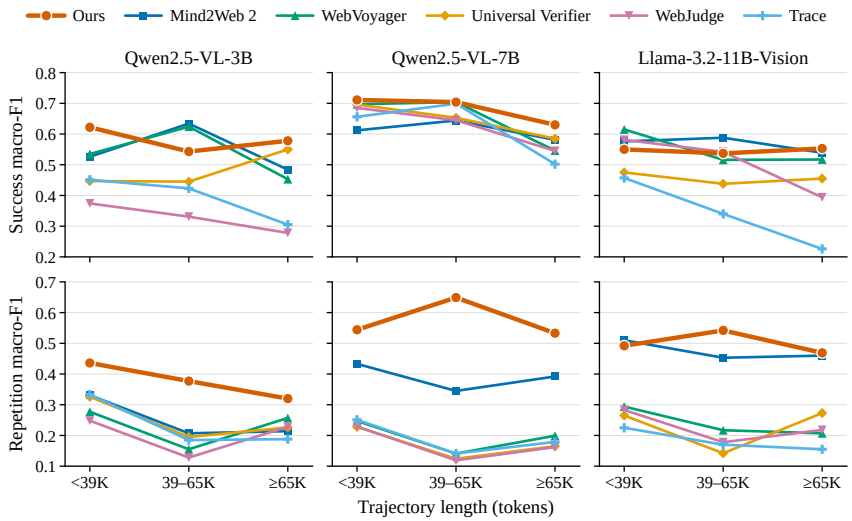



Figure 3: **Performance across trajectory lengths.** Macro-F1 for success judgment (top) and repetition detection (bottom) by trajectory length; trajectories are grouped into three bins by token length (<39K, 39–65K, ≥65K), and all methods within a column share the same backbone.

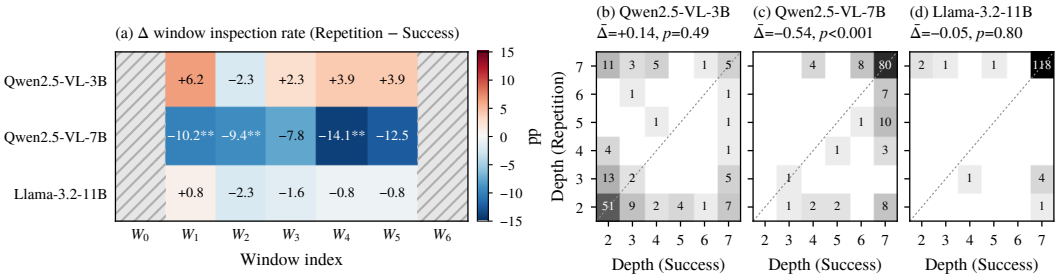



Figure 4: **Rubric-dependent inspection on normalized seven-window trajectories.** (a) Per-window inspection rate difference (pp); hatched endpoints are always inspected. Stars: exact McNemar tests with Holm correction over 15 comparisons ($^{*}p < 0.05$, $^{**}p < 0.01$). (b–d) Joint distribution of number of inspected windows; cells below the diagonal indicate fewer windows for Repetition. $\bar{\Delta}$: mean depth difference; $p$: two-sided Wilcoxon signed-rank test.

### 6.3.2 Rubric-Dependent Window Selection

We compare inspection behavior under the Success and Repetition rubrics across normalized window trajectories (Figure 4). The experiment details are in Appendix C.2.

**The rubric shapes window selection, but the effect is backbone-specific.** With Qwen-7B, all five intermediate windows are inspected less often under Repetition than under Success, with decreases of 7.8–14.1 percentage points, three of which remain significant after Holm correction. In contrast, no significant depth difference is detected for Qwen-3B ($p = 0.49$) or Llama-11B ($p = 0.80$), for opposite reasons: Llama-11B inspects all seven windows under both rubrics on 92.2% of trajectories, leaving little room for rubric-dependent selection, whereas Qwen-3B frequently stops early but shifts its depth between rubrics in both directions.

## 7 Conclusion

In this work, we introduced RecursiveWebJudge, a framework inspired by recursive reasoning that combines adaptive evidence acquisition with rubric-specific aggregation for long-horizon web-agent evaluation. On AgentRewardBench, it improves average macro-F1 across task success and repetition detection by up to 26.6% over the strongest baseline using the same backbone. Its 7B judge also surpasses full-trajectory Claude Code evaluation on this metric. Ablations support the importance of matching aggregation to rubric logic and show that adaptive inspection can approach exhaustive performance while examining fewer windows. In the future, we hope to assess generalization on human-annotated web-agent evaluation benchmarks with longer trajectories and more complex cross-step dependencies.

### AI USE STATEMENT

In this work, we used Claude Code to assist with implementing our method and developing auxiliary scripts. We used ChatGPT to obtain feedback on experimental design, identify and summarize relevant literature, organize references, and improve the clarity and readability of the manuscript. Claude Code was also evaluated as an experimental baseline. The authors checked the related work, reviewed and executed the AI-assisted code, and reviewed the manuscript text. We take responsibility for the final content of this work, including all text, claims, code, and other artifacts produced with the assistance of generative AI.

### ETHICS STATEMENT

This work aims to improve the reliability of automated evaluation for long-horizon web-agent trajectories. All experiments are conducted offline using existing trajectories from the publicly available AgentRewardBench dataset (Lù et al., 2025). We do not recruit human participants or collect new user data. Automated judgments may nevertheless miss relevant evidence or reflect biases in the underlying models and benchmark data. These judgments should therefore be interpreted within the evaluated settings, with human oversight retained for consequential applications. We plan to release our code, prompts, and experimental configurations to support transparency and reproducibility.

## REPRODUCIBILITY STATEMENT

To support reproducibility, we describe RECURSIVEWEBJUDGE in detail in Section 4, provide the algorithm in Appendix A, and include all prompts in Appendix B. Section 5 documents the evaluation datasets and data processing procedures, implementation details and experimental configurations, and evaluation rubrics and metric calculations. We plan to provide an anonymous GitHub repository containing our implementation to facilitate reproduction of the reported results.

## REFERENCES

Anthropic. Claude Code. GitHub repository, 2025. URL https://github.com/anthropics/claude-code.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025. URL https://arxiv.org/abs/2502.13923.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents. *CoRR*, abs/2506.11763, 2025.

Apurva Gandhi, Satyaki Chakraborty, Xiangjun Wang, Aviral Kumar, and Graham Neubig. Recursive agent optimization. *CoRR*, abs/2605.06639, 2026.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jimenez Gutierrez, Yiheng Shu, Chan Hee Song, Jiaman Wu, Shijie Chen, Hanane Nour Moussa, Tianshu Zhang, Jian Xie, Yifei Li, Tianci Xue, Zeyi Liao, Kai Zhang, Boyuan Zheng, Zhaowei Cai, Viktor Rozgic, Morteza Ziyadi, Huan Sun, and Yu Su. Mind2web 2: Evaluating agentic search with agent-as-a-judge. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting*

*of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 6864–6890. Association for Computational Linguistics, 2024.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. Vitabench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. *CoRR*, abs/2509.26490, 2025.

Tenghao Huang, Kung-Hsiang Huang, Prafulla Kumar Choubey, Yilun Zhou, Muhao Chen, Jonathan May, and Chien-Sheng Wu. GTA: generating long-horizon tasks for web agents at scale. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 18805–18820. Association for Computational Linguistics, 2026.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks. *CoRR*, abs/2604.24964, 2026.

Neil Kale, Chen Bo Calvin Zhang, Kevin Zhu, Ankit Aich, Paula Rodriguez, Scale Red Team, Christina Q. Knight, and Zifan Wang. Reliable weak-to-strong monitoring of LLM agents. *CoRR*, abs/2508.19461, 2025.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stanczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *CoRR*, abs/2504.08942, 2025.

Meta. Llama 3.2 Vision Model Card. Model card, 2024. URL `https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD_VISION.md`.

Vijitha Mittapalli, Shreyaa Jayant Dani, Satya Srujana Pilli, Snigdha Ansu, Mohammadreza Teymoorianfard, Franck Dernoncourt, Hongjie Chen, Yu Wang, Ryan A. Rossi, and Nesreen K. Ahmed. TRACE: trajectory reasoning through adaptive cross-step evidence aggregation for LLM agents. *CoRR*, abs/2606.07054, 2026.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. The art of building verifiers for computer use agents. *CoRR*, abs/2604.06240, 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pp. 10014–10037. Association for Computational Linguistics, 2023.

Sumanth Varambally, Thomas Voice, Yanchao Sun, Zhifeng Chen, Rose Yu, and Ke Ye. Hilbert: Recursively building formal proofs with informal reasoning. *CoRR*, abs/2509.22819, 2025.

Leyao Wang, Yanan He, Peng Chen, Asaf Yehudai, Yixin Liu, Rex Ying, Michal Shmueli-Scheuer, and Arman Cohan. Time to REFLECT: can we trust LLM judges for evidence-based research agents? *CoRR*, abs/2605.19196, 2026.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *CoRR*, abs/2504.01382, 2025.

Chenxiao Yang, Nathan Srebro, and Zhiyuan Li. Recursive models for long-horizon reasoning. *CoRR*, abs/2603.02112, 2026.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *CoRR*, abs/2512.24601, 2025.

# A RECURSIVEWEBJUDGE ALGORITHM

Algorithm 1 formalizes how local evidence supports both adaptive inspection and trajectory-level judgment. The shared evidence pool connects these two processes: accumulated evidence guide subsequent window selection and provide the context for final aggregation. Retaining factual notes alongside local assessments allows the judge to relate evidence across windows, including intermediate failures and subsequent recovery. Sub-rubrics organize evidence collection, while the original rubric determines how the collected findings should be interpreted. This design separates local observations from the final verdict, allowing evidence distributed across the trajectory to be assessed jointly.

**Algorithm 1** RECURSIVEWEBJUDGE

**Input:** Task instruction $g$; trajectory of $T$ steps with step records $x_t = (o_t, h_t, a_t)$ for $t < T$ and $x_T = (o_T)$; rubric $r$ with aggregation type $\gamma_r \in \{\text{AND}, \text{ANY}\}$
**Hyperparameters:** window size $w$
**Output:** Trajectory-level verdict $\hat{y}_r \in \{0, 1\}$

*Stage 1: Trajectory windowing (Sec. 4.1)*

$N \leftarrow \lceil T/w \rceil$
$W_i \leftarrow \big(x_{(i-1)w+1}, \ldots, x_{\min(iw,T)}\big)$ for $i = 1, \ldots, N$ ▷ *contiguous, non-overlapping*
$\mathcal{W} = (W_1, \ldots, W_N)$

*Stage 2: Rubric decomposition (Sec. 4.2)*

$\mathcal{L} \leftarrow \mathsf{Decompose}(g, r) = \{\ell_1, \ldots, \ell_M\}$ ▷ *Eq. equation 3; $\mathcal{L} \leftarrow \{r\}$ if inseparable*

*Stage 3: Adaptive evidence acquisition (Sec. 4.3)*

$e_1 \leftarrow \mathsf{Observe}(g, \mathcal{L}, W_1)$ ▷ *Eq. equation 4; initial task context*
**if** $N > 1$ **then** $e_N \leftarrow \mathsf{Observe}(g, \mathcal{L}, W_N)$ ▷ *observed end state*
$\mathcal{E} \leftarrow \{e_i \mid i \in \{1, N\}\}$; $\mathcal{U} \leftarrow \{1, \ldots, N\} \setminus \{1, N\}$; $k \leftarrow |\mathcal{E}|$
**while** $\mathcal{U} \neq \varnothing$ **do**
  $s \leftarrow \mathsf{Control}(g, r, \mathcal{L}, \mathcal{E}, \mathcal{U})$ ▷ *Eq. equation 5; $s \in \mathcal{U} \cup \{\mathsf{STOP}\}$*
  **if** $s = \mathsf{STOP}$ **then break**
  $e_s \leftarrow \mathsf{Observe}(g, \mathcal{L}, W_s)$
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{e_s\}$; $\mathcal{U} \leftarrow \mathcal{U} \setminus \{s\}$; $k \leftarrow k + 1$

*Stage 4: Rubric-specific aggregation (Sec. 4.4)*

**if** $\gamma_r = \text{AND}$ **then**
  $\hat{y}_r \leftarrow \mathsf{Decide}_{\text{AND}}(g, r, \mathcal{E})$ ▷ *Eq. equation 6; all requirements must hold*
**else**
  $\mathcal{Z} \leftarrow \big\{z_{ij} \mid (i, j, z_{ij}, n_{ij}) \in \mathcal{E},\ z_{ij} \neq \text{NA}\big\}$ ▷ *drop* NA *evidence*
  **if** $+ \in \mathcal{Z}$ **then**
    $\hat{y}_r \leftarrow 1$ ▷ *Eq. equation 7; one qualifying instance suffices*
  **else if** $\mathcal{Z} = \{-\}$ **then**
    $\hat{y}_r \leftarrow 0$ ▷ *non-empty and uniformly negative*
  **else**
    $\hat{y}_r \leftarrow \mathsf{Decide}_{\text{ANY}}(g, r, \mathcal{E})$ ▷ *residual case deferred to the judge*
**return** $\hat{y}_r$

# B PROMPT USAGE

We provide the prompt templates used at each stage of RECURSIVEWEBJUDGE, along with the evaluation prompt for the Claude Code full-trajectory baseline. Notation follows Algorithm 1, and placeholders in **{...}** are filled at run time.

Table 3: **Window presentation (user message) for window $W_i$.** Each item lists one sub-rubric $\ell_j \in \mathcal{L}$ with its two answer values. Each step $x_t = (o_t, h_t, a_t)$ in $W_i$ is shown as a text label (URL, action $a_t$, rationale $h_t$), immediately followed by its raw screenshot (observation $o_t$); the final step $x_T = (o_T)$ carries no action or rationale.

*user*

Rubric item(s) (all from one question):
- id=**{lid}** (answer: **{v1}** — **{v2}** — unclear — n/a): **{**$\ell_j$**}**

== window **{**$i$**}** (steps **{first}**-**{last}**) ==
[Step **{**$t$**}**] URL: **{**$\text{url}_t$**}**
  Agent observed this screen, then took action: **{**$a_t$**}**
  Reasoning: **{**$h_t$**}**
<screenshot of step $t$>
... (repeated for each step in $W_i$)

Table 4: **Prompt for** Decompose$(g, r)$ **(Stage 2).** Decides whether rubric $r$ is checked whole ($\mathcal{L} = \{r\}$) or split into $M$ same-type sub-rubrics $\mathcal{L} = \{\ell_1, \ldots, \ell_M\}$ with $2 \le M \le M_{\max}$.

*system*

You are preparing to judge a web agent's trajectory on ONE question. Before looking at evidence, decide whether this question is best checked as a single whole, or whether it naturally breaks into several smaller, independently-verifiable sub-requirements that must each be checked. Split ONLY when the question genuinely contains separable parts, and keep the SAME kind of check: if the question asks whether the task succeeded, the parts are sub-goals that must ALL hold; if it asks whether the agent looped without progress, the parts are specific repeated patterns any of which would count. If it is already a single indivisible check, do not split.

Reason in 1-2 sentences, THEN on a new line output ONLY a JSON list: an empty list `[]` to keep it whole, or a list of 2-**{**$M_{\max}$**}** short sub-requirement strings to split into, e.g. `["filtered to the right category", "sorted by price ascending"]`.

*user*

User goal: **{**$g$**}**
Question (answer: **{hint}**): **{**$r$**}**

Table 5: **Prompt for** $\mathsf{Observe}(g, \mathcal{L}, W_i)$ **(Stage 3).** For each sub-rubric $\ell_j \in \mathcal{L}$, the judge emits a local assessment $z_{ij}$ (`label`) and a factual note $n_{ij}$ (`note`), which together form the evidence record $e_i$. The first listed value, the second listed value, `unclear`, and `n/a` correspond to $z_{ij} = +$, $-$, unclear, and NA, respectively. The system prompt is shared by both rubrics; for the success rubric, the note addendum is appended.

*system (shared by both rubrics)*

You are shown ONE window of a web agent's trajectory: for each step its screenshot, the action taken, and the agent's reasoning; plus one or more rubric items. Each item is a question with exactly two possible answers, listed with the item. Using ONLY what THIS window shows, answer each item's question – holding a HIGH BAR before committing to either decisive answer.

Think first, then answer:
1) REASONING – for EACH item, reason step by step about what this window concretely shows: cite the specific action(s), screen state, or the agent's final answer, and weigh it against the item's question. If this window contains the agent's FINAL answer / end state, judge it decisively.
2) ANSWER – only AFTER reasoning, choose exactly ONE value for each item:
- the item's FIRST listed value – ONLY if THIS window DECISIVELY establishes it: the item's condition is actually, verifiably true here (the outcome the question asks about is really present AND correct, or the described behavior really occurs) – NOT merely that the agent produced some output, clicked around, or appeared to be trying.
- the item's SECOND listed value – ONLY if this window gives clear evidence AGAINST it.
- "unclear" – on-topic but NOT conclusive on its own (plausible / partial, or the outcome is produced but you cannot verify from this window that it is correct). When torn between a decisive value and unclear, choose unclear.
- "n/a" – nothing in this window bears on the item.

Judge ONLY from this window; never fabricate a decisive answer on weak or merely-plausible evidence – use "unclear" instead.
Write your reasoning first (free text). THEN, on a new line, output ONLY a JSON object mapping each item id to {`"label": "<one listed value / unclear / n/a>", "note": "<=25-word note citing the concrete action/screen"`}, e.g. {`"item0": {"label": "not_satisfies", "note": "final answer names the wrong city"`}}.

*success-only note addendum (appended to the shared system prompt)*

NOTE CONTENT: the 'note' must capture BOTH (a) the ACTION taken in this window and (b) its RESULT – the goal-relevant outcome/state it produced. Do not stop at the action alone. If the goal asks for a value or answer (a price, a name, a count, a date, a link), REPORT the value/answer the screen now shows (or 'no result / not shown' if none). If the goal asks to complete or change something, report the resulting state (e.g. confirmation shown / value unchanged / error / unconfirmed). Write it as `'<action>; <resulting goal-relevant state or value>'` – e.g. 'searched DHL price; page shows $42.50 for 1-week delivery' or 'clicked Save; price field still shows old value, no confirmation'. Keep it NEUTRAL and factual (attribute=value, action + resulting state); do NOT judge whether it satisfies the goal and do NOT use 'successfully', 'completed', 'matches the goal', or 'as required'. The 'label' still carries your judgment; the note is neutral evidence for the DECIDE step.

Table 6: **Prompt for** $\mathsf{Control}(g, r, \mathcal{L}, \mathcal{E}, \mathcal{U})$ **(Stage 3).** Given the evidence pool $\mathcal{E}$ (per-window findings) and the unseen windows $\mathcal{U}$ (menu), the judge returns either a window $s \in \mathcal{U}$ to inspect next or STOP.

*system*

You are gathering evidence from a web agent's trajectory to later answer one question about it. It is split into numbered windows (a few consecutive steps each); you have per-window findings so far and a menu of UNSEEN windows. Decide the next move. Keep looking while the question could still be answered differently by evidence you have not seen – consider whether it might depend on parts of the trajectory you have not yet inspected. Stop once the question could be answered and more windows would not change the answer.

Reason in 1-2 sentences, then end with EXACTLY one of:
`<action>inspect</action><window>N</window>` (N = an UNSEEN window number from the menu)
`<action>stop</action>`

Table 7: **Prompt for AND aggregation** $\mathsf{Decide}_{\mathrm{AND}}(g, r, \mathcal{E})$ **(Stage 4, success).** The judge reasons over the local assessments $z_{ij}$ and notes $n_{ij}$ in $\mathcal{E}$, ordered by window index, to decide whether the full goal $g$ was completed by the end; `Successful` and `Unsuccessful` map to $\hat{y}_r = 1$ and 0. Entries with $z_{ij} = \mathrm{NA}$ are excluded from $\mathcal{E}$ before this call.

*system*

You give the FINAL success verdict for a web-agent trajectory by REASONING over per-window findings gathered by an earlier inspection. You are NOT shown the trajectory itself – only these findings: each is one window's answer plus a short note describing what that window's screen/action showed, listed in window order (earliest first, latest last).

Decide whether, by the END of the trajectory, the user's FULL goal was actually accomplished. Reason over the findings as a sequence, not window-by-window in isolation:
- A finding that only shows a SUB-STEP – navigating to or reaching a page, opening/loading something, filling a field, running a search, doing ONE of several required items, or an action whose result is not confirmed – is NOT completion.
- Count the goal as achieved ONLY if the findings together show the required final outcome/answer was actually produced (and, as far as the notes reveal, correct) with EVERY required part done.
- Do not be swayed by a single note that merely says 'successfully <did an action>'; ask whether the WHOLE task was finished by the end. If later windows show the agent still navigating/searching after an apparent success, it was NOT done.
Write 1-3 sentences of reasoning, THEN on a new line output ONLY a JSON object mapping each id to "Successful" or "Unsuccessful", e.g. {"success0": "Successful"}.

Table 8: **Prompt for ANY aggregation** $\mathsf{Decide}_{\mathrm{ANY}}(g, r, \mathcal{E})$ **(Stage 4, repetition).** Invoked only in the residual case of Algorithm 1, i.e., when the non-NA assessments $\mathcal{Z}$ contain no $+$ and are not uniformly $-$; `Yes` and `No` map to $\hat{y}_r = 1$ and 0.

*system*

You give the FINAL verdict for each rubric item listed, using the per-window answers+notes gathered for it (ignore items' 'n/a' windows; weigh 'unclear' partial evidence together with any decisive answers).

Think first (1-2 sentences per id over its evidence), THEN on a new line output ONLY a JSON object mapping each id to its verdict, using that item's OWN two verdict values. A success-type item's verdict is "Successful" ONLY if the evidence, taken together, shows the goal's REQUIRED OUTCOME was actually achieved AND correct (not merely that the agent produced some answer or took plausible steps); otherwise "Unsuccessful". A repetition-type item's verdict is "Yes" ONLY if the evidence shows a non-progressing loop actually occurred, else "No". Example: {"item0": "Successful"}.

Table 9: **Prompt for Claude Code full-trajectory evaluation (system message).** Each rubric is evaluated in a separate call. The shared system prompt is followed by the corresponding pair of verdict tags.

*system (shared by both rubrics)*

You are judging a web agent's trajectory on ONE question. You are shown the user's goal and evidence gathered from the trajectory (observations, actions, and/or findings). Reason step by step about what the evidence actually shows and weigh it against the question, then commit to a verdict.

End your response with EXACTLY one of these tags:

*success tags (appended to the shared system prompt)*

```
<success>Successful</success>
<success>Unsuccessful</success>
```

*repetition tags (appended to the shared system prompt)*

```
<repetition>Yes</repetition>
<repetition>No</repetition>
```

# C Additional Experimental Details

## C.1 Baseline Adaptations

All baselines are re-implemented on a shared harness so that they differ *only* in how trajectory evidence is selected; verdict emission, the judged backbone, and overflow handling are held constant. We describe these as *adapted* implementations wherever they depart from the published evaluators.

**Shared protocol.** Each step exposes the pre-action observation (screenshot and pruned accessibility tree), the action, the agent reasoning, and the page URL; a "final answer" is surfaced from the terminal `send_msg_to_user`/`stop` action (its arguments and reasoning), and an empty placeholder is passed when none exists. Every evaluator routes its final decision through one *neutral, single-rubric* verdict call per rubric: given the goal, the rubric question, and the evaluator's assembled evidence, the model reasons and commits, ending with exactly one tag (`<success>Successful|Unsuccessful</success>` or `<repetition>Yes|No</repetition>`). The Success and Repetition questions are identical across all methods; sabotage-, task-completion-, or final-answer-specific wording from the source evaluators is confined to the evidence-gathering stage and never alters the question. Verdicts are read by tag extraction and normalization; an unparsable response yields a null verdict that is excluded from scoring (counted as coverage loss). Because a unified multimodal request may exceed the served context, a budget guard is applied *after* each method's native selection but *before* the call: it (i) lowers every image's resolution down a fixed pixel ladder, (ii) drops the oldest images while retaining the most recent and enforcing the per-request image cap, and (iii) truncates the longest text items (typically accessibility trees) from the tail; screenshot labels are preserved when their image is dropped so evidence stays aligned.

**WebVoyager (He et al., 2024).** An *adapted* last-window evaluator. The final window is rendered as interleaved (observation, action) pairs with their screenshots and prepended with the agent's final answer. The same tail serves both rubrics (the tail is rubric-agnostic), and each rubric is judged in its own verdict call.

**WebJudge (Xue et al., 2025).** Two stages with *unified* relevance selection. Each screenshot-bearing step is scored once for joint relevance to both questions on a 1–5 scale (parsed as the first digit following the `Score` marker); steps scoring $\geq 3$ are retained as key evidence. Each rubric is then judged over the retained (observation, action) pairs. A single score per step is shared by both rubrics, in contrast to Universal Verifier below.

**Universal Verifier (Rosset et al., 2026).** An *adapted* MMRubricAgent: rather than generating task-specific criteria, the two fixed rubric questions are used as the criteria. Each screenshot is scored 1–5 *separately* per criterion (emitted as `<success>N</success>` and `<repetition>N</repetition>` and parsed by tag), and each criterion keeps its own top-$K$ frames ($K$=5, positive scores only), restored to trajectory order. Each criterion is then judged over its own selected frames, so evidence selection is rubric-specific.

**Mind2Web 2 (Gou et al., 2025).** An *adapted* decompose-then-verify evaluator. Since trajectories carry no final report or cited sources, the citation-grounded verifier is replaced by trajectory verification. An Extractor call decomposes the goal from a compact text-only action overview into a two-branch JSON rubric tree (parsed from the first JSON object; at most five leaves per branch): Success leaves are sub-requirements that must *all* hold, Repetition leaves are looping patterns that count if *any* occurs. If extraction yields no leaves, a canonical fallback leaf is used per branch. Each leaf is then verified independently by one call over an evenly-spaced sample of at most six screenshot-bearing (observation, action) pairs (the terminal frame is always included), emitting `<verdict>Yes|No</verdict>`. Verdicts aggregate bottom-up: Success is the conjunction over Success leaves, Repetition the disjunction over Repetition leaves.

**TRACE (Mittapalli et al., 2026).** We adapt TRACE to each evaluation rubric while retaining its triage–inspect–judge architecture. For each rubric, a single triage pass receives the user goal, the rubric, and a compact text-only overview of the full trajectory. It extracts task constraints and identifies prioritized investigation regions, including local events and patterns involving non-adjacent steps. Each windows serve as evidence-retrieval units, while cross-step analyses may reference observations from different windows. Within each selected region, a controller repeatedly chooses

among `InspectStep`, `CompareSteps`, `CheckRubric`, and `AnalyzeDecisions`, conditioned on accumulated findings. The analysis actions use the relevant accessibility trees, actions, and screenshots to produce step-cited evidence. Findings are retained across regions and inform subsequent investigation.

**Claude Code (Anthropic, 2025).** As a long-context reference point, we include a judge that forgoes all evidence selection and simply places the *entire* trajectory into a single frontier-model context. Every step is serialized in order as an (observation, action) record — its URL, executed action, agent reasoning, and full pruned accessibility tree — interleaved with that step's screenshot, and prefixed with the agent's final response; the two rubrics are then each answered in one call through the same neutral, single-tag verdict prompt used by all baselines, so only the evidence (here, everything) differs. Backed by Claude Opus 4.8 with a 1M-token context, this baseline tests whether raw long-context capacity can substitute for structured, cross-window evidence gathering. It does not: while strong on short trajectories, its accuracy degrades monotonically as trajectory length grows, whereas our recursive judge — which retrieves and reasons over evidence window by window — stays roughly flat and overtakes it on the longer buckets. This contrast is the core motivation for treating long-horizon judging as a cross-context evidence problem rather than a context-length problem.

### C.2 Experiment Details

**Rubric-Dependent Window Selection**

We analyze window selection on 128 trajectories sampled uniformly across lengths from the evaluation set, using a single run per backbone and rubric. The same trajectories are used in all settings, so comparisons between rubrics are paired. Since trajectories contain different numbers of windows, we map each window $W_i$ ($i = 1, \ldots, N$) to its relative position $p_i = (i-1)/(N-1) \in [0, 1]$ and assign it to one of seven equal-width bins $B_k$, where $k = \min(\lfloor 7p_i \rfloor, 6)$. A bin is counted as inspected if any window within it is inspected, and the inspection depth of a trajectory is the number of inspected bins, ranging from 2 to 7.

Since $W_1$ and $W_N$ are always inspected (Algorithm 1), we exclude $B_0$ and $B_6$ from the per-bin comparison, leaving five intermediate bins that reflect the choices of Control. For each backbone and intermediate bin, we compare the paired binary inspection indicators under the two rubrics with an exact McNemar test and apply Holm correction over all 15 comparisons (3 backbones $\times$ 5 bins). For depth, we report the mean paired difference $\bar{\Delta}$ (Repetition $-$ Success) over all trajectories and test it with a two-sided Wilcoxon signed-rank test, discarding zero differences. The skip rates reported in Section 2 are the fraction of intermediate bins left uninspected, averaged over trajectories.

### C.3 Detailed Results

Table 10 reports positive-class precision, recall, and F1 alongside macro-F1 for all methods and backbones. Most baselines rarely flag repetition: except for Mind2Web 2 on Llama-11B and Qwen-7B, their Repetition recall stays below 10%, whereas RecursiveWebJudge reaches 68.0, 26.9, and 85.2 on Llama-11B, Qwen-3B, and Qwen-7B, respectively. On Success, RecursiveWebJudge is more conservative than most baselines, trading recall for precision and attaining the highest precision on both Qwen backbones.

Table 10: **Detailed results.** P, R, and F1 are computed on the positive class (successful or repetitive trajectories), and mF1 averages F1 over both classes; COMB is the unweighted mean of the two mF1 scores. All values are on a 0–100 scale. Bold marks the best mF1 and COMB within each backbone. Claude Opus 4.8, which judges the full trajectory with a 1M-token context, is reported as a reference and excluded from bolding.

| | Success | | | | Repetition | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Method | P | R | F1 | mF1 | P | R | F1 | mF1 | COMB |
| *Claude Opus 4.8 (1M context, full trajectory)* | | | | | | | | | |
| Full-trajectory judge | 50.0 | 37.5 | 42.9 | 68.4 | 80.4 | 89.5 | 84.7 | 50.7 | 59.6 |
| *Llama-3.2-11B-Vision-Instruct* | | | | | | | | | |
| WebVoyager | 18.9 | 61.2 | 28.9 | 54.8 | 77.3 | 8.9 | 16.0 | 24.2 | 39.5 |
| WebJudge | 15.9 | 67.8 | 25.7 | 50.0 | 77.1 | 6.8 | 12.5 | 22.8 | 36.4 |
| Universal Verifier | 11.4 | 43.1 | 18.1 | 45.7 | 71.7 | 7.2 | 13.0 | 22.7 | 34.2 |
| TRACE | 13.9 | 89.2 | 24.0 | 36.2 | 77.8 | 3.9 | 7.5 | 19.3 | 27.7 |
| Mind2Web 2 | 20.7 | 34.5 | 25.9 | **56.7** | 80.3 | 61.1 | 69.4 | 47.9 | 52.3 |
| RECURSIVEWEBJUDGE | 18.8 | 20.7 | 19.7 | 54.8 | 81.0 | 68.0 | 73.9 | **50.4** | **52.6** |
| *Qwen2.5-VL-3B-Instruct* | | | | | | | | | |
| WebVoyager | 23.0 | 84.9 | 36.2 | 54.7 | 72.4 | 8.9 | 15.8 | 23.3 | 39.0 |
| WebJudge | 14.3 | 93.3 | 24.8 | 32.6 | 69.3 | 4.4 | 8.3 | 20.1 | 26.3 |
| Universal Verifier | 15.9 | 55.1 | 24.6 | 48.3 | 79.6 | 8.8 | 15.8 | 24.2 | 36.3 |
| TRACE | 13.6 | 75.5 | 23.0 | 39.6 | 88.1 | 7.5 | 13.8 | 23.7 | 31.6 |
| Mind2Web 2 | 17.5 | 37.5 | 23.9 | 54.2 | 84.3 | 9.7 | 17.4 | 25.3 | 39.8 |
| RECURSIVEWEBJUDGE | 23.8 | 24.5 | 24.1 | **57.4** | 85.8 | 26.9 | 41.0 | **37.7** | **47.5** |
| *Qwen2.5-VL-7B-Instruct* | | | | | | | | | |
| WebVoyager | 31.9 | 64.7 | 42.7 | 65.7 | 84.9 | 3.0 | 5.8 | 19.6 | 42.7 |
| WebJudge | 29.0 | 73.6 | 41.7 | 63.1 | 88.9 | 0.7 | 1.4 | 17.4 | 40.2 |
| Universal Verifier | 37.9 | 47.5 | 42.2 | 67.1 | 95.2 | 0.6 | 1.2 | 17.2 | 42.1 |
| TRACE | 27.0 | 78.5 | 40.2 | 62.1 | 87.6 | 2.6 | 5.1 | 19.1 | 40.6 |
| Mind2Web 2 | 27.6 | 38.7 | 32.2 | 61.1 | 89.4 | 28.3 | 43.0 | 39.6 | 50.4 |
| RECURSIVEWEBJUDGE | 40.7 | 46.7 | 43.5 | **68.1** | 83.8 | 85.2 | 84.5 | **59.5** | **63.8** |

### C.4 CASE STUDIES

We present six cases that illustrate when adaptive inspection helps and where it fails (Table 11). All cases use Qwen2.5-VL-7B-Instruct with seed 0 and window size $w = 5$, and reference labels are expert annotations (majority vote where multiple annotations exist). Each trajectory contains 30 action steps followed by a final observation, yielding seven windows in which $W_7$ contains only the final observation. Windows are indexed from 1, while steps keep the zero-based identifiers of the trajectory logs (`s00`–`s30`). All quoted notes and rationales are verbatim model outputs.

Table 11: **Overview of case studies.** Each verdict answers the rubric question: whether the agent completed the task (Success) or looped without making progress (Repetition). Verdicts that disagree with the expert label are shown in *italics*; numbers in parentheses give the windows inspected by our method out of seven. ‡Correct verdict reached through a flawed rationale. VWA denotes VisualWebArena.

| Case | Task | Rubric | Expert | First+Last | Ours | Exhaustive |
|---|---|---|---|---|---|---|
| 1 | WebArena 654 | Repetition | Yes | *No* | Yes (5) | Yes |
| 2 | VWA 277 | Repetition | Yes | *No* | Yes (7) | Yes |
| 3 | WebArena 690 | Success | Yes | *No* | Yes (7) | *No* |
| 4 | WorkArena hardware | Success | No | No | No (7) | No |
| | | Repetition | Yes | Yes | Yes (3) | Yes |
| 5 | VWA 27 | Repetition | No | *Yes* | *Yes* (7) | *Yes* |
| 6 | WebArena 654 | Success | No | *Yes* | No‡ (6) | *Yes* |

### C.4.1 WHERE ADAPTIVE INSPECTION HELPS

**Case 1: An idle loop between the endpoints (WebArena 654).** The agent is asked to fill in a refund request on a contact form without submitting it. It completes the form in `s00–s02` and then issues 27 consecutive `noop()` actions (`s03–s29`); experts label the trajectory as repetitive. Both endpoint windows look benign: $W_1$ covers the form filling ("Agent filled out the form and decided not to submit it.") and $W_7$ shows the completed form. Both are therefore labeled negative, and First+Last returns No because uniformly negative evidence yields a negative verdict. Our method continues to the intermediate windows, where $W_2$–$W_4$ are labeled positive (e.g., "Agent repeatedly observed the same contact form page without taking any action."). It then stops, judging that the remaining windows "do not seem necessary to confirm the pattern," and returns Yes after five of seven windows, in agreement with Exhaustive. Because the loop occupies only the middle of the trajectory, detecting it requires evidence beyond the endpoints.

**Case 2: A cycle recognizable only on recurrence (VisualWebArena 277).** The agent must find the cheapest yellow video projector and post its name to a forum. It cycles nearly four times through the same three pages, namely the electronics page, the Video Projectors listing, and a product page (`s02–s12`), and later scrolls the forum page up and down without posting (`s18–s29`); experts label the trajectory as repetitive. $W_1$ contains only the first pass through the cycle, which is indistinguishable from normal navigation, and $W_7$ contains only the final forum page; First+Last labels them negative and unclear, respectively, and returns No. Our method continues to $W_2$, where the cycle recurs ("Agent clicked back to main electronics page after selecting video projectors."), and to $W_4$–$W_6$, which cover the repeated scrolling (e.g., "Repetitive scrolling without visible goal"), and returns Yes, as does Exhaustive. Unlike the idle loop in Case 1, this cycle consists of individually plausible actions and becomes recognizable only once a pass recurs.

**Case 3: Corroborating an incomplete endpoint observation (WebArena 690).** The agent must send the shop owner a message requesting a promised coupon. It submits the message at `s02`, which triggers a confirmation banner, but then repeatedly refills the form and clicks the submit button for the rest of the episode (e.g., 12 consecutive `click('1504')` at `s04–s15`); experts label the trajectory as both successful and repetitive. $W_1$ records the confirmation ("Form submission successful; confirmation message displayed"), whereas the note for $W_7$ records only that the optional Phone Number field is empty, overlooking the confirmation banner that remains visible on the final screen. With only these two findings, First+Last treats the empty field as a possibly unmet requirement and returns Unsuccessful. Our method additionally observes blocked resubmissions in $W_2$ and $W_3$ ("Form submission not possible due to required field 'What's on your mind?' not filled.") and the confirmation again in $W_4$ and $W_6$. Reasoning over this ordered sequence, it concludes that the goal was "ultimately achieved" despite the failed attempts. Evidence repeated across windows thus compensates for an incomplete observation at an endpoint. Exhaustive inspection observes the same windows yet returns Unsuccessful, indicating that aggregating such mixed evidence remains sensitive even under full coverage.

**Case 4: Rubric-dependent inspection depth (WorkArena hardware sorting).** The agent must open the Hardware Assets list and sort it, but clicks the same "All" menu button 30 times in a row (`s00–s29`) without reaching the module; experts label the trajectory as unsuccessful and repetitive. Under Repetition, $W_1$ and $W_7$ are already labeled positive; the judge inspects $W_2$, finds "consistent lack of progress," and stops after three windows, four fewer than Exhaustive. Under Success, every inspected window through $W_5$ is labeled negative, yet the judge keeps inspecting "to see if the agent eventually reaches the intended module," covering all seven windows before returning Unsuccessful. The Control prompt is identical for both rubrics, so the difference in depth arises from the questions themselves: a single loop settles Repetition, whereas an early failure does not rule out later completion. All three variants are correct here, so this case concerns inspection cost rather than accuracy; it is the per-trajectory counterpart of the rubric-dependent selection analyzed in Section 6.3.2.

### C.4.2 WHERE IT FAILS

**Case 5: Misreading an unchanged screen (VisualWebArena 27).** The agent must count the RV listings posted on a given date that show the vehicle interior. It clicks through the listings on the results page, mostly on distinct targets (e.g., `[151]`, `[145]`, `[166]`), while the URL stays

fixed and the rendered page appears unchanged; consistent with the distinct targets, experts do not label the trajectory as repetitive. The judge nevertheless labels $W_1$ and $W_3$–$W_5$ positive. For $W_4$, which contains clicks on five different targets (`s15`–`s19`), its note reads "Agent repeatedly clicks the same element without any visible change." First+Last and Exhaustive make the same error, so the failure stems from interpreting evidence rather than acquiring it. Because a single positive window determines a Repetition verdict, such misreadings translate directly into false positives, consistent with the high Repetition FPR in Table 2. Distinguishing systematic browsing from looping therefore requires relating actions to their targets rather than relying on unchanged screens alone.

**Case 6: A correct verdict with a flawed rationale (WebArena 654).** Under Success, experts label the trajectory from Case 1 as unsuccessful. Its message claims that "the product SKU is included" but never states the SKU, leaving a task requirement unmet. None of the variants detects this omission, as the notes describe the form as "filled out correctly," and First+Last and Exhaustive accordingly return Successful. Our method returns Unsuccessful after six windows, matching the expert label, but for the wrong reason: its final reasoning assumes that the goal was "to fill out a contact form and submit it" and treats the instructed non-submission as a failure. Agreement at the verdict level can therefore conceal errors in both perception and goal interpretation, and verifying fine-grained content such as identifiers remains a limitation shared by all variants.