DiagEval: Trajectory-Conditioned Diagnosis for Reliable Software Evaluation with GUI Agents

Sirui Hong[1]* **Zhijie Liu**[1]* **Tengfei Li**[1] **Wei Tao**[2] **Yifan Wu**[3] **Chenglin Wu**[1]†
[1]DeepWisdom [2]Independent Researcher [3]The Hong Kong University of Science and Technology (Guangzhou)

Abstract

Evaluating LLM-generated interactive software requires execution in addition to static analysis. The key difficulty is that correctness is a graph-level reachability property over latent UI state-transition graphs, whereas a GUI evaluator observes only a single execution trajectory. A failed rollout therefore rules out only one realized path, leaving failure attribution ambiguous between evaluator-side execution error and genuine software defect. We present DIAGEVAL, a trajectory-conditioned diagnostic evaluation protocol for post-failure GUI-agent evaluation of interactive software. Rather than blindly retrying from scratch, DIAGEVAL reuses the failed trajectory to choose targeted diagnostic probes and aggregates their outcomes into an internal attribution signal. The latent-graph view motivates the diagnostic problem; DIAGEVAL does not reconstruct the graph or estimate calibrated posterior probabilities. We evaluate DIAGEVAL on WebDevJudge-Unit and RealDevBench across multiple GUI-agent evaluators and LLM backbones. On false-negative cases, DIAGEVAL recovers 45.6–62.1% of failures that were initially misattributed to software defects, outperforming retry-based baselines with 34.4–160.0% relative gains. On the full evaluation sets, this recovery improves accuracy from 69.9% to 78.3% on WebDevJudge-Unit and from 65.0% to 81.6% on RealDevBench. Controlled component ablations attribute the main verdict improvements to trajectory-conditioned targeted probing, while EIG ordering and attribution-based stopping primarily reduce diagnostic probe cost. We additionally report induced TN flips and independent audits to characterize the reliability limits of post-failure verdict revision. Our code is available at `https://github.com/scutGit/DiagEval`.

1 Introduction

As LLM-generated software grows from isolated code snippets to full-stack applications with complex UIs, the evaluation problem changes with it: static analysis is no longer enough, and correctness must be verified through interactive execution [Bian et al., 2025, Lu et al., 2025c, Xiao et al., 2026, Li et al., 2026, Kong et al., 2026, Peng et al., 2026]. GUI agents are a natural choice for this role. However, they introduce a new reliability risk: evaluator-side failures can become false evidence against the software itself. This is not a corner case. Recent works [Li et al., 2026, Lu et al., 2025c] on automated web-development evaluation show that agentic evaluators can incorrectly label feasible tasks as infeasible due to their own operational failures rather than actual shortcomings in the implementation. In parallel, exploratory GUI testing work identifies an execution-bias attribution problem, where software-side defects are misidentified as agent-side execution slips [Gao et al., 2026]. This ambiguity is further amplified by agent-environment misalignment: as noted by Liu et al. [2025], an agent's expected outcome may diverge from the environment's actual state transition. For example, a button that remains unchanged after a click may indicate a broken interface element, a misgrounded action, or a view that failed to expose the relevant state transition.

*Equal contribution.
†Corresponding author: `alexanderwu@deepwisdom.ai`.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

At its core, the challenge is *failure attribution under limited trajectory evidence*: a failed GUI-agent rollout records only one agent–environment interaction trace, in which actions, observations, DOM states, logs, and reasoning traces are entangled. These signals are diagnostically ambiguous because the same failed transition can arise from different sources: the agent may ground or execute an action incorrectly, the environment may expose only a partial observation of the relevant program state, or the software may genuinely block all valid progress. This ambiguity is structural rather than incidental, because software correctness is a graph-level property while an evaluator observes only a path-local slice of it. A test should be considered successful if at least one valid execution path reaches the target state in the software's latent state-transition graph. By contrast, a failed rollout rules out only the sampled path and does not prove that the goal state is unreachable. We refer to this structural under-determination as the *single-trajectory identifiability gap*. After observing a single failed trajectory, the latent cause of failure is therefore structurally underdetermined. The failure may reflect evaluator-side under-exploration or execution error (AGENTFAIL), or it may indicate a genuine system defect (ENVFAIL). Without probing alternative paths or selectively acquiring additional evidence, these hypotheses remain observationally confounded.

Existing agent frameworks and reliability mechanisms improve GUI-agent execution in different ways, but they do not directly address post-failure attribution. ReAct-style agents [Yao et al., 2023] interleave reasoning and acting to gather evidence, but they do not explicitly distinguish execution unreliability from genuine software failure after a failed interaction. Repetition- and feedback-based strategies such as naive retry, Best-of-N sampling, and self-correction can reduce transient errors by adding more attempts or feedback, but they often fail when the same agent-side mistake recurs across runs or is reinforced by correlated self-critique [Shinn et al., 2023, Madaan et al., 2023, Huang et al., 2024]. More broadly, robustness methods acknowledge environment noise and uncertainty, but aim to harden policies against it rather than attribute a particular failure [Smirnova et al., 2019]. Post hoc attribution and active information acquisition offer related tools for diagnosing completed traces or selecting informative actions, but they are not designed to turn a failed GUI-agent rollout into online diagnostic probes for software evaluation [Zhang et al., 2025a, Veiga and Renoux, 2023, Chen et al., 2026]. What is missing is a diagnostic stance for GUI-agent evaluation: after a failure, the evaluator should use the failed trajectory to decide which uncertainty sources to probe next and how the resulting evidence should be attributed to evaluator-side error versus genuine software defect.

To address this, we propose DIAGEVAL, which formulates post-failure software evaluation as active diagnosis under attribution uncertainty. Rather than retrying from scratch, DIAGEVAL reuses the executed trajectory to identify the dominant source of uncertainty, interacts with the environment through targeted probes to disambiguate uncertainty, and integrates the resulting outcomes into an internal attribution signal. In this way, post-failure evaluation becomes a trajectory-conditioned diagnostic process rather than undifferentiated re-execution. Across two interactive software benchmarks, DIAGEVAL improves recovery and accuracy over retry-based baselines, and its diagnostic mechanism transfers across GUI-agent frameworks, supporting post-failure diagnosis as an evaluator-side reliability mechanism. Our contributions are as follows:

1. **Problem formulation.** We identify post-failure attribution under limited trajectory evidence as a distinct reliability problem in interactive software evaluation. We show that single-trajectory verdicts are structurally insufficient because graph-level correctness cannot be determined from path-local observation alone.
2. **Trajectory-conditioned diagnostic protocol.** We propose DIAGEVAL, which decomposes a failed trajectory into typed sources of uncertainty, prioritizes candidate diagnostic probes via a structured information-value ranking, and integrates branch-typed outcomes into an internal attribution signal through an update rule.
3. **Reliability gains across benchmarks.** On WebDevJudge-Unit (WDJ-U) [Li et al., 2026] and RealDevBench (RDB) [Bian et al., 2025], DIAGEVAL recovers 45.6–62.1% of false negatives, compared with 17.5–46.2% under retry-based baselines. On the full evaluation sets, accuracy rises from 69.9% to 78.3% on WDJ-U and from 65.0% to 81.6% on RDB.
4. **Reliability–cost frontier and transfer.** DIAGEVAL improves reliability without scaling the GUI-execution backbone or specializing to a GUI-agent framework. With `gemini-3-flash-preview` as the executing GUI agent, it reaches 76.3%/78.6% on WDJ-U/RDB, exceeding AppEvalPilot run end-to-end on `claude-opus-4-6` at less than half its per-case cost; it also transfers to UI-TARS without retuning, with $+14.2$–$+22.1$ accuracy points.

2 Related Work

2.1 Evaluation for Software Engineering

Software-engineering evaluation has progressed from function- and repository-level code benchmarks [Chen et al., 2021, Zhuo et al., 2024, Jain et al., 2025, Zhang et al., 2024, Ding et al., 2023, Liu et al., 2024, Jimenez et al., 2024, Miserendino et al., 2025] to agentic, interactive settings [Zhuge et al., 2024, Chan et al., 2024, Zhou et al., 2024, Xie et al., 2024, Rawles et al., 2025]. Most relevant to us, WebDevJudge and RealDevBench [Li et al., 2026, Bian et al., 2025] adopt execution-based evaluation in dynamic environments, yet still treat a failed rollout as a terminal verdict. We instead study post-failure attribution: whether a failure reflects a true software defect or evaluator-side under-exploration.

2.2 LLM-as-a-Judge

LLM-as-a-Judge [Zheng et al., 2023] evaluates static outputs, and Agent-as-a-Judge [Zhuge et al., 2024] extends this paradigm to intermediate trajectories. Existing efforts to improve reliability, e.g., debiasing, hallucination reduction, judge finetuning, rubric prompting, and multi-judge ensembling [Ye et al., 2024, Li et al., 2025, Chen et al., 2024, Thakur et al., 2025, Xu et al., 2023, Wei et al., 2025, Chan et al., 2023, Zhu et al., 2025], often operate on fixed evidence. This assumption is inadequate for interactive software evaluation, where a failed trajectory exposes only a path-local slice of graph-level correctness. Recent Computer-Using Agent verification methods also improve judgment within the fixed-trajectory setting [Rosset et al., 2026]. AJ-Bench explicitly studies judges that interact with environments to acquire evidence [Shi et al., 2026]. Within this broader setting, we focus on targeted diagnosis after a failed GUI rollout to revise the software verdict.

2.3 GUI Agents

GUI agents have progressed from metadata-based systems to fully visual agents that can autonomously interact with software environments [Cheng et al., 2024, Wu et al., 2024, Xu et al., 2024, Gou et al., 2024], enabled by multimodal LLMs such as Claude 3.5 Sonnet [Anthropic, 2024] and Gemini 2.5 [DeepMind, 2025]. However, as evaluators, GUI agents remain unreliable under partial observability, asynchronous rendering, and agent-induced execution errors, including hallucination, UI mislocalization, and inconsistent behavior in dynamic environments [Jin et al., 2026, Li et al., 2026]. Frameworks such as WebGen-Bench [Lu et al., 2025c] and UXAgent [Lu et al., 2025a] exploit GUI agents for large-scale interactive evaluation, but their judgments are still trajectory-dependent and lack attribution capability: they cannot reliably distinguish software defects from evaluator-side failures. We address this limitation by formulating evaluator failure as a trajectory-level diagnosis problem and using cross-trajectory evidence to distinguish software defects from agent-side execution misses.

2.4 Failure Attribution

Recent work studies failure attribution from agent execution trajectories. Who&When localizes the responsible agent and step in multi-agent failure logs [Zhang et al., 2025b]. AgenTracer uses replay and fault injection to construct training data for a learned failure tracer [Zhang et al., 2025a], while TraceElephant provides full traces and reproducible environments for attribution evaluation [Chen et al., 2026]. Our focus is agent-versus-environment failure attribution in GUI evaluation, using active trajectory-conditioned probing and EIG-based probe ranking.

3 Preliminaries and Problem Formulation

3.1 Graph-Based Evaluation as Reachability Inference

We model the software under test as a latent state-transition graph $\mathcal{G} = (\mathcal{S}, \mathcal{E})$, where each edge represents an action-labeled executable UI transition between latent interface states. A test case $T = (I, O)$ is successful if some reachable state satisfies the target output condition. We use $V_{\text{case}} \in \{0, 1\}$ to denote this ground-truth success label. Software correctness is therefore a graph-level reachability property. The evaluator, however, does not observe $\mathcal{G}$ directly. Instead, it interacts with the software through a partially observed trajectory $\tau = (o_0, a_0, \ldots, o_N)$, where each o_t is an observation and each a_t is an executed action. Based on this trajectory, the evaluator outputs a binary verdict $\hat{y} \in \{0, 1\}$ indicating whether the test case is judged successful. The evaluation objective is therefore to maximize $\mathbb{P}(\hat{y} = V_{\text{case}})$. The key mismatch is structural: the evaluator observes

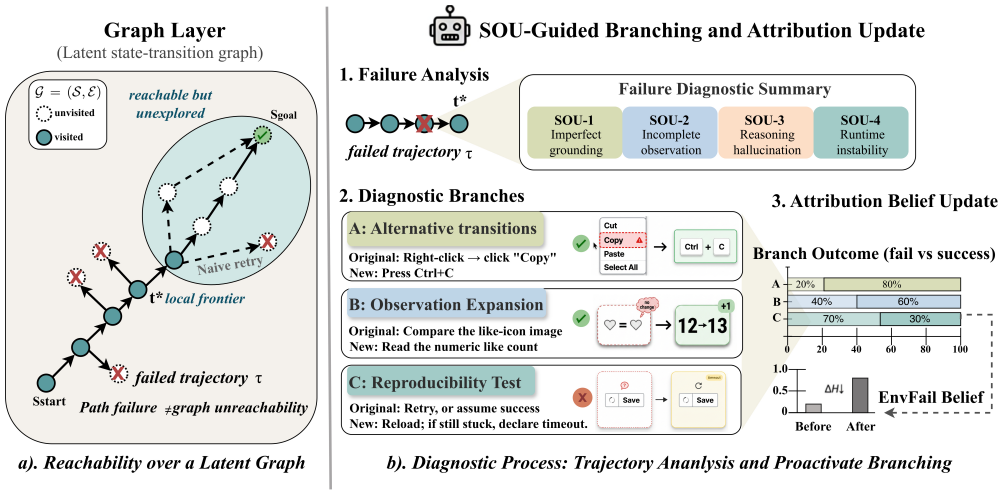


Figure 1: **Overview of DIAGEVAL.** Given a failed rollout τ, DIAGEVAL parses a failure diagnostic summary (FDS), dispatches SOU-guided diagnostic branches, and integrates evidence across multiple branch trajectories to refine an internal attribution score over $Z \in \{\text{AGENTFAIL}, \text{ENVFAIL}\}$.

only the realized execution path, whereas correctness depends on whether some valid path exists in the underlying reachable graph. As shown in Figure 1(a), a failed trajectory τ rules out only the realized path, not the reachability of s_{goal}. As a result, a failed trajectory does not establish that the target state is unreachable, making negative verdicts particularly difficult to interpret in GUI-based software evaluation.

3.2 Sources of Uncertainty and the Attribution Problem

When a GUI agent fails to complete a task, the latent failure cause $Z \in \{\text{AGENTFAIL}, \text{ENVFAIL}\}$ is not directly observable. AGENTFAIL indicates that $V_{\text{case}} = 1$, i.e., a valid execution path exists in $\mathcal{G}$, but the evaluator fails to discover or verify it. ENVFAIL indicates that $V_{\text{case}} = 0$, i.e., the target state is genuinely unreachable under the intended task semantics. The evaluator must infer Z from indirect evidence carried by the executed trajectory. Each rollout reveals only a noisy, partial local view of the latent transition graph, shaped by perception, reasoning, and runtime distortions. We identify four systematic *sources of uncertainty* (SOU) that challenge attribution:

(SOU 1) Imperfect grounding (perception errors). The agent may fail to correctly locate or interact with the intended UI elements [Gou et al., 2024, Lee et al., 2025, Chen et al., 2025a, Yang et al., 2025] due to visual ambiguity, dynamic layouts, or subtle affordance changes. Such errors can make an evaluator-side failure appear externally similar to an environment-side blockage.

(SOU 2) Incomplete observation (partial observability). The agent's view captures only the currently visible UI state, missing hidden elements such as collapsed menus, off-screen content, or latent interface regions [Kaynak et al., 2025, Garousi et al., 2024]. As a result, the evaluator may conclude failure even though valid interactions remain undiscovered.

(SOU 3) Reasoning hallucination (inference errors). LLM-based evaluators may prematurely conclude task completion, over-interpret weak UI signals, or rationalize failed interactions as environmental faults [Anthropic Engineering Team, 2025, Zhang et al., 2025c, Chen et al., 2025b, Lu et al., 2025b, Ye et al., 2025]. This introduces cognitive distortion on top of already partial evidence.

(SOU 4) Runtime instability (execution errors). Transient environment conditions such as network timeouts, nondeterministic backend latency, or flaky UI state may cause non-reproducible failures, affecting action validity and outcome stability.

These heterogeneous, context-dependent SOUs cannot be reliably disambiguated from a single trajectory alone. Different SOUs imply different diagnostic responses: SOU-1/2/3 require testing alternative paths, while SOU-4 needs reproducibility checks. This makes attribution inherently diagnosis-dependent.

3.3 The Single-Trajectory Identifiability Gap

This ambiguity is structural. Given a failed trajectory τ, the post-failure evaluation problem is to maintain evidence about $Z \in \{\text{AGENTFAIL}, \text{ENVFAIL}\}$ from partial trajectory observations. This

is particularly challenging for negative verdicts, since a single observed trajectory reveals only one realized path, while failure may arise either from an environment-side defect or from the evaluator's inability to discover an alternative feasible path. We refer to this underdetermination as the *single-trajectory identifiability gap*, which motivates post-failure evaluation as a sequential diagnosis problem requiring active evidence acquisition.

4 Method: DiagEval

After an initial FAIL, DIAGEVAL performs four steps: summarize the uncertain part of the failed rollout, resume near the localized point, try a small number of targeted alternative interactions, and revise the verdict using observed state evidence (Figure 1(b)). The latent-graph view motivates the diagnostic problem; DiagEval does not reconstruct the latent state-transition graph. EIG orders selected probes, while the internal attribution score supports early stopping and diagnostic ranking. The complete loop is given in Appendix C.

4.1 Failure Summary and Restart Selection

Fork-Node Localization. Given an initial failed trajectory $\tau_1 = (o_0, a_0, \ldots, o_T)$, DIAGEVAL seeks a *fork node* t^*, the restart point from which targeted re-exploration is expected to most effectively reduce attribution uncertainty over Z. Intuitively, the fork node is not simply the most anomalous step, but the earliest actionable point at which alternative continuations may still separate AGENTFAIL from ENVFAIL. Conceptually, a useful fork node balances *goal proximity* to a feasible continuation and *attribution informativeness* for reducing uncertainty over Z:

$$t^* \in \arg\max_{t \in \mathcal{T}_{\text{cand}}} \widehat{\text{Prog}}(t) \cdot \widehat{\Delta H}(t). \tag{1}$$

where $\mathcal{T}_{\text{cand}} \subseteq \{0, \ldots, T-1\}$ denotes the candidate restart steps in τ_1, $\widehat{\text{Prog}}(t) \in [0,1]$ measures estimated proximity to a plausible continuation, and $\widehat{\Delta H}(t) \geq 0$ measures expected reduction in attribution uncertainty. Because $\mathcal{G}$ and the true attribution distribution over Z are not observable from a single failed trajectory, DIAGEVAL treats Eq. 1 as an idealized selection criterion rather than a closed-form objective. It approximates this criterion with a reflective LLM-based judge $\mathcal{J}$ using a compressed diagnostic summary $\mathcal{R}_1$ derived from the failed GUI-agent trajectory τ_1. The summary aggregates the executed trajectory context, agent action explanations, and final failure evidence such as screenshots, DOM state, and judge output. The judge $\mathcal{J}$ analyzes $\mathcal{R}_1$ and predicts $(t^*, h^*, \delta^*) \leftarrow \mathcal{J}(\mathcal{R}_1, \texttt{task})$, where t^* is the selected restart index, h^* is the dominant SOU, and δ^* is a diagnostic explanation. In effect, $\mathcal{J}$ implicitly performs both candidate ranking and step selection over $\mathcal{T}_{\text{cand}}$ from trajectory-level evidence.

Failure Diagnostic Summary (FDS). Based on the fork-node reasoning, DIAGEVAL constructs a Failure Diagnostic Summary (FDS): $\mathcal{F}_0 = (t^*, h^*, D_{t^*}, \mathcal{C}_{t^*})$, where $D_{t^*} = (\delta^*, \mathcal{A}_{\text{cand}})$ provides the failure explanation and candidate policies, and $\mathcal{C}_{t^*}$ denotes the recoverable environment context associated with t^*, used to restore the local UI state through prefix replay or context reconstruction before executing diagnostic probes. The FDS serves as the structured interface between failure parsing and downstream diagnostic probing. Implementation details are provided in Appendix F.1.

4.2 Diagnostic Probe Generation

Given the diagnostic state $\mathcal{F}_k = (t_k^*, h_k, D_{t_k^*}, \mathcal{C}_{t_k^*})$, DIAGEVAL initializes probe types at the fork node t_k^* and instantiates them as a candidate branch set $\mathcal{P}_k$. Here, a *branch* denotes a detailed executable plan instantiated from a probe type. The dominant SOU h_k guides this routing; details are provided in Appendices F.2 and F.3. Figure 2 illustrates why diagnostic branches should be guided by the FDS rather than by blind retry: an unguided retry may reproduce the same visible state, whereas an FDS-guided branch redirects exploration toward more informative transitions.

Type A (Alternative transitions). Type-A probes realize the same subgoal through alternative actions, modalities, or routes. They test whether the original failure was caused by misexecution rather than genuine unreachability, and are primarily used when $h_k \in \{1, 3\}$.

Type B (Observation Expansion). Type-B probes expand the visible frontier of the interface through scrolling, navigation, menu expansion, or state revelation. They test whether the original failure stemmed from incomplete observation, and are primarily used when $h_k = 2$.

Type C (Reproducibility Test). Type-C probes perform controlled repetitions under the same or minimally perturbed conditions to test whether the failure is stable. They are used when $h_k \in \{3, 4\}$, especially for runtime instability and ambiguous reasoning failures.

Each branch $b \in \mathcal{P}_k$ contains: (i) a concrete action sequence, and (ii) an expected observation pattern that would differentially support AGENTFAIL or ENVFAIL. These probe types are not only behaviorally distinct, but also attributionally asymmetric, which we formalize as branch-typed likelihoods in Sec. 4.3.

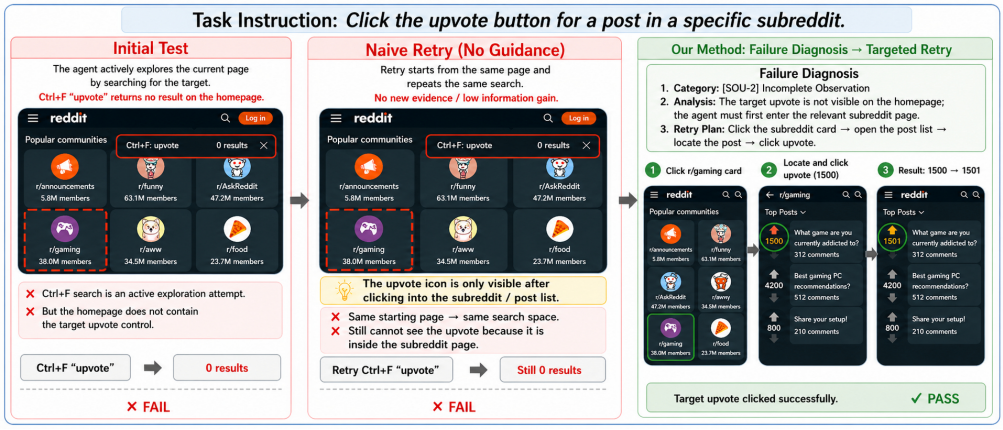


Figure 2: **Comparison of post-failure retry strategies** on *"Click the upvote button for a post in a specific subreddit."* The initial test (left) and a naive retry (middle) both fail along essentially the same homepage trajectory. DIAGEVAL (right) uses the FDS to identify SOU-2 (*Incomplete Observation*), generates a targeted retry plan, and verifies success (`1500` → `1501`).

4.3 Information-Gain Branch Ranking

Before execution in round k, DIAGEVAL orders candidates in $\mathcal{P}_k$ by a structured information-value principle: branches are preferred when their possible outcomes are expected to better separate AGENTFAIL from ENVFAIL. We instantiate this principle with a heuristic **Expected Information Gain** (EIG) score based on the branch-typed update rule in Sec. 4.4. Here, $p^{(k)}$ denotes DIAGEVAL's current preference toward ENVFAIL at diagnosis round k. To simulate outcomes, DIAGEVAL uses likelihood parameters rather than calibrated probabilities. For a branch b of type $d(b)$, w_b scores recovery under AGENTFAIL, $\beta_{d(b)}$ scores success that can still occur under ENVFAIL, and $\gamma_{d(b)}$ scores failure compatibility under AGENTFAIL. We set $\gamma_A > \gamma_B > \gamma_C$, making Type C failures stronger evidence for ENVFAIL. Formal parameter definitions are provided in Appendix F.4. The EIG score of branch b is then

$$\mathrm{EIG}(b; \mathcal{F}_k) = H_{\mathrm{bin}}(p^{(k)}) - \mathbb{E}_{o \sim P(\cdot|b)} \left[H_{\mathrm{bin}}(p_{b,o}^{(k+1)}) \right], \tag{2}$$

where $H_{\mathrm{bin}}(p^{(k)})$ is the attribution uncertainty before executing any branch in round k, and $p_{b,o}^{(k+1)}$ is the hypothetical attribution signal obtained if branch b produced outcome o. The expectation is over $o \in \{\texttt{verified_success}, \texttt{fail}\}$ under the outcome scores defined above. Consequently, branches expected to produce the largest reduction in attribution uncertainty are ranked first, while lower-gain branches are deferred, making the limited branch budget focus on more informative probes.

4.4 Attribution Score Update and Stopping

After EIG ranks the candidate set $\mathcal{P}_k$, DIAGEVAL executes branches in descending EIG order. Let $p^{(k,j-1)}$ and $p^{(k,j)}$ denote the attribution signal before and after executing the j-th branch in round k. For branch $b_{k,j}$, let $d_{k,j} = d(b_{k,j})$, $w_{k,j} = w_{b_{k,j}}$, and $o_{k,j}$ be its observed outcome. We update the attribution score using a Bayes-style heuristic rule:

$$p^{(k,j)} = \begin{cases} \dfrac{p^{(k,j-1)}}{p^{(k,j-1)} + (1 - p^{(k,j-1)})\gamma_{d_{k,j}}}, & o_{k,j} = \texttt{fail}, \\ \dfrac{\beta_{d_{k,j}}\, p^{(k,j-1)}}{w_{k,j}(1 - p^{(k,j-1)}) + \beta_{d_{k,j}}\, p^{(k,j-1)}}, & o_{k,j} = \texttt{verified_success}. \end{cases} \tag{3}$$

Here, `verified_success` denotes the online evaluator's judgment that the original task's specified interaction and expected result have been satisfied. The stopping rule reflects outcome asymmetry: `verified_success` immediately stops diagnosis with verdict PASS (AGENTFAIL) on the basis of the evaluator's success signal. Failures are accumulated, and if the updated signal satisfies $p^{(k,j)} \geq \tau_{\text{env}}$, DIAGEVAL stops early with verdict FAIL (ENVFAIL); otherwise, diagnosis continues with the updated attribution score until budget exhaustion. The attribution score therefore controls failed-probe evidence accumulation and early stopping, but does not independently produce a PASS verdict.

5 Experiments

5.1 Experimental Setup

Datasets and Metrics. We evaluate on two open-source web-development benchmarks: RealDevBench (RDB) [Bian et al., 2025] with 429 annotated executable cases and WebDevJudge-Unit (WDJ-U) [Li et al., 2026] with 502 unit-level tasks with ground-truth pass/fail labels. We report end-to-end accuracy and average cost per case (USD). For cases with ground-truth PASS but an initial FAIL verdict, we report the number and rate of recovered false negatives (FNs). For cases with ground-truth FAIL and an initially correct negative verdict, we count true-negative flips (TN flips), i.e., incorrect reversals to PASS. We define *net corrections* as recovered FNs minus TN flips. These quantities characterize diagnostic gains and erroneous verdict reversals under the same initial evaluation cohort. Appendix D provides dataset details, and Appendix E describes the RDB annotation protocol.

Baselines. We compare AppEvalPilot [Bian et al., 2025], WebVoyager [He et al., 2024], UI-TARS Agent [Qin et al., 2025][3], and model-native CUA. We report model-native CUA for Qwen3.5-35B-A3B, Claude Opus 4.6, and Gemini 3 Flash Preview [Qwen Team, 2026, Anthropic, 2026a, Google DeepMind, 2025], and evaluate GUI-agent frameworks with Claude Opus 4.6 or Gemini 3 Flash Preview as shown in Table 1. We set the GUI-agent step budget *max_iters* = 15 and 20 on WDJ-U and RDB, respectively.

Diagnostic Settings. We use AppEvalPilot as the GUI-agent evaluation framework, invoking DIAGEVAL's online diagnosis procedure from Sec. 4 only when an initial FAIL verdict is returned. We evaluate two diagnostic budgets: DIAGEVAL ($\times 1$) runs one diagnosis round, and DIAGEVAL ($\times 2$) adds a second round for cases that remain unresolved after the first. Each round k instantiates a candidate pool $\mathcal{P}_k$ of $N = 5$ branches and executes the top $K = 3$ under EIG ranking. Diagnosis starts from a case-specific attribution score $p^{(0)}$ produced by the supervisor during FDS construction from the initial failed trajectory, with a fallback of 0.1 when no valid score is returned. Diagnosis terminates upon `verified_success` or once $p^{(k,j)} \geq \tau_{\text{env}} = 0.7$. We set $(w_0, \beta_0) = (0.60, 0.20)$ and $(\gamma_A, \gamma_B, \gamma_C) = (0.60, 0.50, 0.40)$, and instantiate the diagnostic judge $\mathcal{J}$ with Claude Sonnet 4.6 [Anthropic, 2026b]. Per-case costs include the AppEvalPilot rollout and $\mathcal{J}$ calls for FDS construction and branch selection. Appendix H.5 provides an auxiliary analysis of LLM-call composition on the FN diagnostic subset. The parameter-setting protocol and sensitivity analyses are reported in Appendix H.6.

Controlled Ablation Protocol. For the WDJ-U component ablation, all configurations are evaluated on the same frozen initial cohort of 502 cases, including 114 FNs and 186 TNs. The initial verdicts and trajectories are held fixed. We vary FDS and fork-point reuse, SOU-guided targeted branching, EIG ordering, and attribution-based early stopping. The branch-based configurations use the same maximum branch budget. Unless otherwise specified, ablation costs are reported per initially false-negative diagnostic case rather than per full-set case.

5.2 Experimental Results

(i) Reliability through targeted recovery. Table 1 presents the end-to-end results. With Gemini 3 Flash Preview as the GUI-execution backbone, DIAGEVAL ($\times 1$) achieves 76.3% accuracy on WDJ-U and 78.6% on RDB, compared with 69.9% and 65.0% for AppEvalPilot. A second diagnostic round improves accuracy to 78.3% and 81.6%, respectively. On WDJ-U, the first diagnostic round recovers 52 of 114 initial FNs while flipping 20 of 186 initial TNs, yielding 32 net corrections; two rounds cumulatively recover 68 FNs and flip 26 TNs, yielding 42 net corrections. On RDB, FN

[3]We use the UI-TARS Agent implementation released with the WebDevJudge open-source repository.

Table 1: Main results on WDJ-U and RDB. `Acc.` is end-to-end accuracy, Δ is the absolute change in percentage points relative to AppEvalPilot with the same model, and FN/TN report false-negative and true-negative counts. Changes are computed before rounding. Costs are reported in USD per case. $*$ indicates that the AppEvalPilot-relative change is not computed. Arrows indicate metric direction; best values are bolded, and DIAGEVAL rows are highlighted.

Model	Framework	Acc. ↑	Δ ↑	FN ↓	TN ↑	$/case ↓
(a) WDJ-U (502 cases)						
Qwen3.5-35B-A3B	Model CUA	46.2%	$*$	258	**211**	0.080
Claude Opus 4.6	Model CUA	70.9%	-3.2	97	174	0.321
	AppEvalPilot	74.1%	—	106	199	0.404
	UI-TARS	74.5%	$+0.4$	103	198	0.362
	WebVoyager	72.9%	-1.2	106	193	0.190
Gemini 3 Flash Preview	Model CUA	56.6%	-13.3	160	165	0.040
	AppEvalPilot	69.9%	—	114	186	0.075
	UI-TARS	54.8%	-15.1	143	139	0.044
	WebVoyager	68.9%	-1.0	102	169	**0.035**
	DIAGEVAL ($\times 1$)	76.3%	$+6.4$	62	166	0.122
	DIAGEVAL ($\times 2$)	**78.3%**	$\mathbf{+8.4}$	**46**	160	0.154
(b) RDB (429 cases)						
Qwen3.5-35B-A3B	Model CUA	42.2%	$*$	238	**81**	0.090
Claude Opus 4.6	Model CUA	69.0%	-0.9	94	52	0.346
	AppEvalPilot	69.9%	—	104	66	0.385
	UI-TARS	71.3%	$+1.4$	97	65	0.188
	WebVoyager	70.4%	$+0.5$	89	53	0.189
Gemini 3 Flash Preview	Model CUA	62.2%	-2.8	107	36	**0.019**
	AppEvalPilot	65.0%	—	132	73	0.112
	UI-TARS	52.9%	-12.1	181	70	0.020
	WebVoyager	68.3%	$+3.3$	107	62	0.025
	DIAGEVAL ($\times 1$)	78.6%	$+13.5$	70	69	0.183
	DIAGEVAL ($\times 2$)	**81.6%**	$\mathbf{+16.6}$	**50**	62	0.234

counts decrease from 132 to 70 after one round and to 50 after two rounds, while the initially correct TN counts decrease from 73 to 69 and 62, respectively. Thus, the accuracy improvements reflect substantial FN recovery despite nonzero erroneous TN reversals, rather than unqualified recovery of all initially failed evaluations.

(ii) Accuracy–cost tradeoff. On WDJ-U, DIAGEVAL ($\times 1$) with Gemini 3 Flash Preview reaches 76.3% accuracy at \$0.122 per full-set case, compared with 74.1% accuracy and \$0.404 per case for AppEvalPilot using Claude Opus 4.6. On RDB, the corresponding DIAGEVAL setting reaches 78.6% at \$0.183 per case, versus 69.9% at \$0.385 per case for the Claude Opus 4.6 AppEvalPilot configuration. These comparisons characterize accuracy–cost trade-offs under the evaluated configurations; because both the execution backbone and evaluation pipeline differ, they do not isolate a causal effect of the attribution score or imply universal cross-backbone cost superiority.

(iii) Attribution evidence from SOU-guided probes. Figure 3 breaks down where the recovered cases come from. SOU-2 contributes the most to FN recovery, mainly through Type B visibility branches, and achieves the highest recovery on both WDJ-U and RDB (75.0% and 60.3%). SOU-1 and SOU-3 mainly trigger Type A alternative-transition branches, suggesting evaluator-side grounding or plan-realization errors. SOU-4 behaves differently: reproducibility-oriented branches produce a TN flip of 0.0% on RDB and 5.8% on WDJ-U within the SOU-4 subset. Overall, the accuracy gain comes from structured diagnostic evidence and informative counter-evidence, rather than from indiscriminately flipping failed verdicts.

5.3 Ablation Study

Diagnostic-Mechanism Ablation. To isolate what drives verdict correction, we compare five configurations on the same frozen WDJ-U initial-evaluation cohort (N=502; 114 initial FNs and 186 initial TNs), under the ×1 diagnostic budget. Naive Retry reruns the task; NR+FDS/Fork adds failure diagnosis and fork-point resumption; Targeted Branch-only additionally executes SOU-guided typed probes in the default order, stopping on verified success or after budget exhaustion; Branch+EIG changes only the probe order; and Full DIAGEVAL additionally enables attribution-score updates and early termination with a FAIL verdict. The three branch-based arms reuse the same FDS, fork point, candidate pool, selected probes, executor, and maximum budget. Table 2 reports full-set accuracy, FN recovery, TN flips, net corrections, average probes, and cost per initially false-negative case.

Table 2 shows that targeted branching is the main source of verdict improvement. Adding SOU-guided typed probes to NR+FDS/Fork raises full-set accuracy by 6.2 points and recovers 30 addi-

Table 2: Controlled component ablation on WDJ-U ($N = 502$, $\times 1$). All arms share the frozen initial evaluation cohort (114 FNs and 186 TNs). *FN rec.*: recovered FNs; *TN flips*: initially correct negative verdicts changed to PASS; *Net*: FN rec. minus TN flips. Average probes and USD cost are measured per originally false-negative diagnostic case, not per full-set case.

Method	Acc. (%)	FN rec.	TN flips	Net	Avg. probes	Cost/FN case
Naive Retry	69.92	20	20	0	1.00	0.059
NR+FDS/Fork	70.32	23	21	+2	1.00	0.053
Targeted Branch-only	**76.49**	**53**	20	**+33**	3.00	0.153
Branch+EIG	**76.49**	**53**	20	**+33**	2.63	0.134
Full DIAGEVAL	76.29	52	20	+32	**2.38**	**0.122**

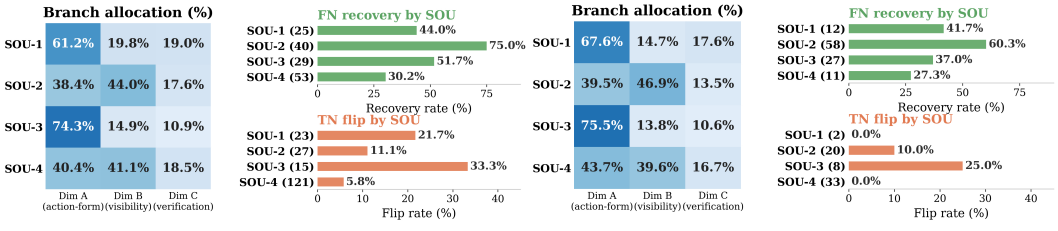
tional FNs with one fewer TN flip. EIG preserves identical verdicts while reducing average probes from 3.00 to 2.63 and cost from 0.153 to 0.134. Full DIAGEVAL further reduces probes and cost to 2.38 and 0.122, but early stopping recovers one fewer FN and lowers accuracy by 0.20 points. Thus, targeted probing explains the accuracy gain in this ablation, while EIG and Eq. 3 contribute ordering and stopping efficiency. Complementary FN-subset results on RDB and under the $\times 2$ diagnostic budget are reported in Table 8, showing consistent recovery gains across both benchmarks and budgets.

Bilateral Attribution Score: Magnitude and Quality. Table 3 evaluates whether the terminal attribution score p_{end} ranks initially missed FNs and retained TNs in their respective ground-truth attribution directions. ROC-AUC is 0.620/0.720 at $\times 1$ and 0.761/0.763 at $\times 2$ on RDB/WDJ-U, respectively. The relatively modest RDB $\times 1$ result indicates that this ranking signal varies across settings. We therefore interpret p_{end} as an uncalibrated diagnostic ranking and stopping signal, rather than an independent correctness oracle or a calibrated estimate of the true failure cause.

Table 3: **Score-based bilateral attribution by DIAGEVAL.** Resolved cases are FN examples recovered to *pass* or TN examples retained as *fail*; missed cases are the corresponding complements. We report the mean score assigned to the ground-truth attribution direction and the ROC-AUC of $P(\text{env})$ for ranking TN over FN cases.

	FN: GT= AGENTFAIL		TN: GT= ENVFAIL		
Setting	$\bar{P}(\text{agent})\,\vert_{\text{res}} \uparrow$	$\bar{P}(\text{agent})\,\vert_{\text{miss}} \downarrow$	$\bar{P}(\text{env})\,\vert_{\text{res}} \uparrow$	$\bar{P}(\text{env})\,\vert_{\text{miss}} \downarrow$	**ROC-AUC** $\uparrow$
RealDevBench (RDB)					
DIAGEVAL ($\times 1$)	0.670	**0.126**	**0.805**	0.430	0.620
DIAGEVAL ($\times 2$)	**0.672**	0.208	0.803	**0.374**	**0.761**
WebDevJudge-Unit (WDJ-U)					
DIAGEVAL ($\times 1$)	**0.659**	**0.135**	**0.873**	**0.402**	0.720
DIAGEVAL ($\times 2$)	0.657	0.148	0.864	0.431	**0.763**

Efficiency Analysis of Branch Ordering. We further assess EIG-based diagnostic efficiency by comparing branch ordering methods. As detailed in Appendix H.4, EIG ordering achieves the highest first-branch success rate (66%) and the lowest probe cost (1.48 executed probes per recoverable case), demonstrating a clear efficiency advantage over the other ordering methods. Additionally, we examine full-pipeline run-to-run variability (Appendix H.2), sensitivity to supervisor choice (Appendix H.7), and relaxed evidence-scoring assumptions (Table 14) to assess their impact on diagnostic accuracy and efficiency.



(a) WDJ-U. (b) RDB.

Figure 3: **SOU-typed branch allocation and diagnostic outcomes.** Each subfigure corresponds to one benchmark. The left panel shows the distribution of Type A/B/C probes within each SOU, and the right panels report FN recovery for Z=AGENTFAIL and TN flip for confirmed Z=ENVFAIL cases.

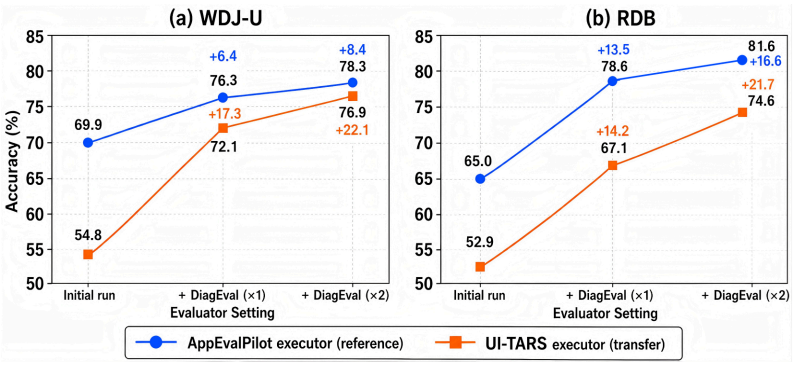


Figure 4: **Cross-framework transfer results.** Δ denotes the absolute accuracy gain in percentage points over the corresponding single-pass baseline of the same GUI-agent framework.

5.4 Cross-Framework Transfer

We evaluate whether the proposed diagnostic mechanism transfers across GUI-agent frameworks, with details provided in Appendix H.3. We use UI-TARS as the GUI-agent evaluation framework and invoke DIAGEVAL's online diagnosis procedure when UI-TARS returns an initial FAIL verdict. As shown in Figure 4, DIAGEVAL achieves maximum absolute end-to-end accuracy improvements of 22.1 and 21.7 percentage points on WDJ-U and RDB, respectively. These gains indicate that DIAGEVAL operates as a framework-agnostic evaluator-side mechanism, rather than a prompt-level enhancement tied to a specific GUI-agent framework.

6 Conclusion and Limitations

DIAGEVAL shows that post-failure GUI evaluation is better framed as trajectory-conditioned diagnosis than as repeated retry. By reusing prior trajectories, launching targeted probes, and aggregating cross-trajectory evidence, it recovers 45.6–62.1% of false negatives (vs. 17.5–46.2% for retry), yields mean entropy reduction of $\Delta H = 0.27$–0.41 over the binary attribution score under the fixed update model, and lifts full-set accuracy from 69.9% to 78.3% on WebDevJudge-Unit and from 65.0% to 81.6% on RealDevBench. More broadly, when correctness is graph-level but evidence is path-local, reliability depends on adaptive evidence acquisition rather than attempt multiplicity alone.

Limitations. DIAGEVAL relies on hand-tuned likelihood hyperparameters, prompt-implemented branch ranking, and a binary attribution space, so the resulting score is an internal diagnostic signal rather than a calibrated posterior. The fail-update rule also assumes $\mathbb{P}(\text{fail} \mid \text{ENVFAIL}, d) = 1$ for every probe type, which may be too strong. The score accumulates probe evidence using a conditional-independence approximation. Our empirical evaluation focuses on web applications across two benchmarks and the tested executor/supervisor configurations; empirical support for applicability to non-web GUI domains remains limited. The diagnostic loop further depends on an external LLM supervisor $\mathcal{J}$ for FDS construction and branch generation (see §5.1), so failures or biases of $\mathcal{J}$ can propagate into both probe selection and the resulting attribution signal. The online success verifier can also mistake related functionality or visually similar outcomes for task completion, particularly when the task constrains the interaction modality or requires a persistent post-condition (Appendix H.8). Future work should strengthen verification of task-specific constraints, learn calibrated likelihoods, reduce prompt dependence in branch selection, study cross-supervisor robustness, and extend attribution beyond the binary split.

Acknowledgments and Disclosure of Funding

Funding. This work received no third-party funding or in-kind support.

Competing interests. The authors declare no competing interests.

References

Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, 2024. Accessed on March 28, 2025.

Anthropic. Claude Opus 4.6, February 2026a. URL https://www.anthropic.com/news/claude-opus-4-6. Anthropic model announcement and system-card reference.

Anthropic. Claude Sonnet 4.6 system card, February 2026b. URL https://www.anthropic.com/claude-sonnet-4-6-system-card. System card for Claude Sonnet 4.6.

Anthropic Engineering Team. Effective harnesses for long-running agents. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents, 2025. Accessed: 2025-12.

Yutong Bian, Xianhao Lin, Yupeng Xie, Tianyang Liu, Mingchen Zhuge, Siyuan Lu, Haoming Tang, Jinlin Wang, Jiayi Zhang, Jiaqi Chen, Xiangru Tang, Yongxin Ni, Sirui Hong, and Chenglin Wu. You don't know until you click: Automated gui testing for production-ready software evaluation, 2025. URL https://arxiv.org/abs/2508.14104.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate, 2023. URL https://arxiv.org/abs/2308.07201.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. MLLM-as-a-judge: Assessing multimodal LLM-as-a-judge with vision-language benchmark. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=dbFEFHAD79.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. *arXiv preprint arXiv:2604.22708*, 2026.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. GUICourse: From general vision language model to versatile GUI agent. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21936–21959, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1065. URL https://aclanthology.org/2025.acl-long.1065/.

Yihan Chen, Benfeng Xu, Xiaorui Wang, Yongdong Zhang, and Zhendong Mao. Training llm-based agents with synthetic self-reflected trajectories and partial masking, 2025b. URL https://arxiv.org/abs/2505.20023.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024.

Google DeepMind. Gemini 2.5 computer use model, 2025. URL https://blog.google/technology/google-deepmind/gemini-computer-use-model.

Yangruibo Ding, Zijian Wang, Wasi Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, et al. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. *Advances in Neural Information Processing Systems*, 36:46701–46723, 2023.

Yifei Gao, Jiang Wu, Xiaoyi Chen, Yifan Yang, Zhe Cui, Tianyi Ma, Jiaming Zhang, and Jitao Sang. Guitester: Enabling gui agents for exploratory defect discovery, 2026. URL https://arxiv.org/abs/2601.04500.

Vahid Garousi, Nithin Joy, Zafar Jafarov, Alper Buğra Keleş, Sevde Değirmenci, Ece Özdemir, and Ryan Zarringhalami. Ai-powered software testing tools: A systematic review and empirical assessment of their features and limitations, 2024. URL https://arxiv.org/abs/2409.00411.

Google DeepMind. Gemini 3 Flash model card, December 2025. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Flash-Model-Card.pdf. Model card for Gemini 3 Flash; evaluated via Gemini 3 Flash Preview API.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024. URL https://arxiv.org/abs/2410.05243.

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. *arXiv preprint arXiv:2401.13919*, 2024.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Chao Jin, Wenkui Yang, Hao Sun, Yuqi Liao, Qianyi Jiang, Kai Zhou, Jie Cao, Ran He, and Huaibo Huang. Halluclear: Diagnosing, evaluating and mitigating hallucinations in gui agents. *arXiv preprint arXiv:2604.17284*, 2026.

Ergün Batuhan Kaynak, Mayasah Lami, Sahand Moslemi, and Anil Koyuncu. Llmshot: Reducing snapshot testing maintenance via llms. In *2025 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 827–832, 2025. doi: 10.1109/ICSME64153.2025.00087.

Fanheng Kong, Jingyuan Zhang, Yang Yue, Chenxi Sun, Yang Tian, Shi Feng, Xiaocui Yang, Daling Wang, Yu Tian, Jun Du, Wenchong Zeng, Han Li, and Kun Gai. Webtestbench: Evaluating computer-use agents towards end-to-end automated web testing. *arXiv preprint arXiv:2603.25226*, 2026.

Hyunseok Lee, Jeonghoon Kim, Beomjun Kim, Jihoon Tack, Chansong Jo, Jaehong Lee, Cheonbok Park, Sookyo In, Jinwoo Shin, and Kang Min Yoo. Reguide: Data efficient gui grounding via spatial reasoning and search, 2025. URL https://arxiv.org/abs/2505.15259.

Chunyang Li, Yilun Zheng, Xinting Huang, Tianqing Fang, Jiahao Xu, Lihui Chen, Yangqiu Song, and Han Hu. WebDevJudge: Evaluating (M)LLMs as critiques for web development quality. In *Proceedings of the International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=CCSPm6V5EF.

Songze Li, Chuokun Xu, Jiaying Wang, Xueluan Gong, Chen Chen, Jirui Zhang, Jun Wang, Kwok-Yan Lam, and Shouling Ji. Llms cannot reliably judge (yet?): A comprehensive assessment on the robustness of llm-as-a-judge, 2025. URL https://arxiv.org/abs/2506.09443.

Kaiming Liu, Xuanyu Lei, Ziyue Wang, Peng Li, and Yang Liu. Agent-environment alignment via automated interface generation. *arXiv preprint arXiv:2505.21055*, 2025.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=pPjZIOuQuF.

Yuxuan Lu, Bingsheng Yao, Hansu Gu, Jing Huang, Zheshen Jessie Wang, Yang Li, Jiri Gesi, Qi He, Toby Jia-Jun Li, and Dakuo Wang. Uxagent: An llm agent-based usability testing framework for web design. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, CHI EA '25, pages 1–12. Association for Computing Machinery, 2025a. doi: 10.1145/3706599.3719729. URL https://doi.org/10.1145/3706599.3719729.

Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan, and Hongsheng Li. Webgen-agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning, 2025b. URL https://arxiv.org/abs/2509.22644.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025c. URL https://arxiv.org/abs/2505.03733.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering? *arXiv preprint arXiv:2502.12115*, 2025.

Zhiyuan Peng, Wei Tao, Xin Yin, Chenhao Ying, Yuan Luo, and Yiwen Guo. PlayCoder: Making llm-generated gui code playable, 2026. URL https://arxiv.org/abs/2604.19742.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *Proceedings of the International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/01a83bc2f2732a58e6aa731e659e7101-Abstract-Conference.html.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. The art of building verifiers for computer use agents. *arXiv preprint arXiv:2604.06240*, 2026. doi: 10.48550/arXiv.2604.06240. URL https://arxiv.org/abs/2604.06240.

Wentao Shi, Yu Wang, Yuyang Zhao, Yuxin Chen, Fuli Feng, Xueyuan Hao, Xi Su, Qi Gu, Hui Su, Xunliang Cai, and Xiangnan He. AJ-bench: Benchmarking agent-as-a-judge for environment-aware evaluation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 25371–25413. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1269. URL https://aclanthology.org/2026.findings-acl.1269/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Elena Smirnova, Elvis Dohmatob, and Jérémie Mary. Distributionally robust reinforcement learning, 2019. URL https://arxiv.org/abs/1902.08708.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. Judging the judges: Evaluating alignment and vulnerabilities in LLMs-as-judges. In Ofir Arviv, Miruna Clinciu, Kaustubh Dhole, Rotem Dror, Sebastian Gehrmann, Eliya Habba, Itay Itzhak, Simon Mille, Yotam Perlitz, Enrico Santus, João Sedoc, Michal Shmueli Scheuer, Gabriel Stanovsky, and Oyvind Tafjord, editors, *Proceedings of the Fourth Workshop on Generation, Evaluation and Metrics (GEM²)*, pages 404–430, Vienna, Austria and virtual meeting, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-261-9. URL https://aclanthology.org/2025.gem-1.33/.

Tiago Veiga and Jennifer Renoux. From reactive to active sensing: A survey on information gathering in decision-theoretic planning. *ACM Computing Surveys*, 55(13s):1–22, 2023.

Tianjun Wei, Wei Wen, Ruizhi Qiao, Xing Sun, and Jianghong Ma. Rocketeval: Efficient automated llm evaluation via grading checklist, 2025. URL https://arxiv.org/abs/2503.05142.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

Bang Xiao, Lingjie Jiang, Shaohan Huang, Tengchao Lv, Yupan Huang, Xun Wu, Lei Cui, and Furu Wei. Code aesthetics with agentic reward feedback. In *Proceedings of the International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Q87kwGI6bx.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://openreview.net/forum?id=tN61DTr4Ed.

Wenda Xu, Danqing Wang, Liangming Pan, Zhenqiao Song, Markus Freitag, William Wang, and Lei Li. INSTRUCTSCORE: Towards explainable text generation evaluation with automatic feedback. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5967–5994, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.365. URL https://aclanthology.org/2023.emnlp-main.365/.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. Aria-UI: Visual grounding for GUI instructions. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22418–22433, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1152. URL https://aclanthology.org/2025.findings-acl.1152/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. Justice or prejudice? quantifying biases in llm-as-a-judge, 2024. URL https://arxiv.org/abs/2410.02736.

Naimeng Ye, Xiao Yu, Ruize Xu, Tianyi Peng, and Zhou Yu. Ai agents for web testing: A case study in the wild, 2025. URL https://arxiv.org/abs/2509.05197.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems?, 2025a. URL https://arxiv.org/abs/2509.03312.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, and Qingyun Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 76583–76599. PMLR, 2025b. URL https://proceedings.mlr.press/v267/zhang25cq.html.

Shudan Zhang, Hanlin Zhao, Xiao Liu, Qinkai Zheng, Zehan Qi, Xiaotao Gu, Yuxiao Dong, and Jie Tang. Naturalcodebench: Examining coding performance mismatch on humaneval and natural user queries. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 7907–7928, 2024.

Weichen Zhang, Yiyou Sun, Pohao Huang, Jiayue Pu, Heyue Lin, and Dawn Song. Mirage-bench: Llm agent is hallucinating and where to find them, 2025c. URL https://arxiv.org/abs/2507.21017.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges, 2025. URL https://arxiv.org/abs/2310.17631.

Mingchen Zhuge, Changsheng Zhao, Dylan Ashley, Wenyi Wang, Dmitrii Khizbullin, Yunyang Xiong, Zechun Liu, Ernie Chang, Raghuraman Krishnamoorthi, Yuandong Tian, Yangyang Shi, Vikas Chandra, and Jürgen Schmidhuber. Agent-as-a-judge: Evaluate agents with agents, 2024. URL https://arxiv.org/abs/2410.10934.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

A Broader Impact

DIAGEVAL improves the reliability of automated software evaluation by separating evaluator-side errors from genuine defects. However, because GUI agents process screenshots, DOM states, and interactive workflows, this approach risks exposing sensitive user data. Furthermore, enhanced UI exploration capabilities could be maliciously misused to probe software vulnerabilities. To mitigate these risks, deployments must strictly use sandboxed environments, exclude sensitive data, ensure fully auditable diagnostic traces, and restrict application to authorized testing settings.

B Detailed SOU and Diagnostic Operators

Table 4: Detailed sources of uncertainty (SOU) and diagnostic operators. This table expands the corruption mode and observable cue for each SOU.

SOU	Corruption	Cue	Op.
Imperfect grounding	Wrong target or actuation	A blocked edge may still be reachable	A
Incomplete observation	Hidden viewport, tab, or state	A continuation may lie off-screen or off-state	B
Reasoning hallucination	Misread UI semantics	A reported blockage may be spurious	A/C
Runtime instability	Non-deterministic execution	A failure may be transient	C

C Sequential Diagnostic Algorithm

Overall diagnostic procedure. The SOU operators in Table 4 define the evidence channels used by DIAGEVAL. Algorithm 1 summarizes how these operators are invoked in the sequential diagnostic loop. After an initial failed evaluation, DIAGEVAL constructs an FDS, generates and selects diagnostic branches, orders them by EIG score, and updates the attribution score after each executed probe. Diagnosis terminates when a branch verifies success, when the attribution signal exceeds τ_{env}, or when the diagnostic budget is exhausted.

Algorithm 1: DIAGEVAL Sequential Diagnostic Evaluation

Input: `task`; evaluator $\mathcal{E}$; rounds R; candidates N_{cand}; budget K; threshold τ_{env}.
Output: final verdict $\hat{y}$ and terminal attribution score p.

$(\tau_1, \hat{y}_1) \leftarrow \mathcal{E}(\texttt{task})$; if $\hat{y}_1 = \text{PASS}$, return $\hat{y} = \text{PASS}$
Initialize $\mathcal{H} \leftarrow \emptyset$
for $k = 0, \ldots, R-1$ **do**
  $\mathcal{F}_k = (t_k^*, h_k, D_{t_k^*}, \mathcal{C}_{t_k^*}) \leftarrow \Phi_{\text{FDS}}(\tau_1, \mathcal{H})$
  if $k = 0$ **then** $p_{\text{cur}} \leftarrow \text{InitScore}(\mathcal{F}_k)$ **end if**; $p^{(k,0)} \leftarrow p_{\text{cur}}$
  $\mathcal{P}_k \leftarrow \mathcal{G}(\mathcal{F}_k, N_{\text{cand}})$
  $\widetilde{\mathcal{P}}_k \leftarrow \text{TopK}_{K,\, b \in \mathcal{P}_k} \mathcal{J}_{\text{sel}}(b; \mathcal{F}_k, \mathcal{H})$
  $\pi_k \leftarrow \text{argsort}_{b \in \widetilde{\mathcal{P}}_k}[-\text{EIG}(b; \mathcal{F}_k, p^{(k,0)})]$
  for $j = 1, \ldots, |\pi_k|$ **do**
    $b_{k,j} \leftarrow \pi_k[j], \quad d_{k,j} \leftarrow d(b_{k,j})$
    $(\tau_{k,j}^{\text{br}}, o_{k,j}) \leftarrow \text{Exec}(b_{k,j}, \mathcal{C}_{t_k^*})$
    $p^{(k,j)} \leftarrow \text{Update}(p^{(k,j-1)}, o_{k,j}, b_{k,j}); \quad p_{\text{cur}} \leftarrow p^{(k,j)}$
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(b_{k,j}, d_{k,j}, o_{k,j}, \tau_{k,j}^{\text{br}}, p^{(k,j)})\}$
    if $o_{k,j} = \texttt{verified_success}$ **then return** $\hat{y} = \text{PASS}$
    if $p^{(k,j)} \geq \tau_{\text{env}}$ **then return** $\hat{y} = \text{FAIL}$
  end for
end for
return $\hat{y} = \text{FAIL},\ p_{\text{cur}}$

D Datasets and Metrics

We evaluate on two open-source web-development benchmarks: RealDevBench (RDB) [Bian et al., 2025] and WebDevJudge-Unit (WDJ-U) [Li et al., 2026]. RDB is a repository-level benchmark. For this dataset, we use the commercial software-generation system Atoms[4] to generate code because it provides direct run-and-deploy functionality, allowing us to collect deployed software environments without preparing offline containers. The raw RDB annotation pool contains 662 generated cases with valid paired annotator scores after excluding construction-failure records. From this pool, we remove cases whose deployed environments fail completely and form two disjoint subsets: a 41-case pilot set used only for lightweight sanity checks of empirical parameters, and a 429-case main test set used for all reported RDB results. The 41-case pilot set also provides the dimension-prior counts later summarized in Tables 6 and 7; it is not used in the reported RDB test results. WDJ-U contains 502 unit-level tasks with ground-truth pass/fail labels.

For both benchmarks, accuracy is the fraction of cases whose final evaluator verdict matches the ground-truth label, and average cost per case is the total API cost divided by the number of evaluated cases. The FN subset contains cases with ground-truth PASS but initial verdict FAIL; FN recovery rate is the fraction of this subset corrected to final PASS under the matched diagnostic setting. For diagnostic analyses, TN flip rate measures the fraction of initially correct FAIL verdicts on ground-truth FAIL cases that are changed to PASS, and entropy reduction ΔH is computed from the internal attribution signal before and after diagnostic probing.

E RealDevBench Annotation Protocol

For the RealDevBench portion, we manually annotate each generated software project against the task-specific test requirements to determine whether the environment-level execution is genuinely successful. Each case is independently reviewed by two test engineers, after which an algorithm reviewer performs calibration. Cases with inconsistent judgments are re-executed and adjudicated in a second round. This process yields the environment-level pass/fail labels for the RDB pilot and main-test subsets, with only the 429-case main test set used in the reported RDB results.

Inter-rater agreement. The first annotator (A) and the second annotator (B) independently assign a binary score to each case: 0 for failure and 1 for pass. After excluding records with construction failures, 662 cases receive valid scores from both annotators. The two annotators agree on 611 cases, yielding an observed agreement of $P_o = 0.923$. Cohen's κ is 0.734, with an expected chance agreement of $P_e = 0.710$, indicating moderate-to-substantial agreement under the standard interpretation of κ values. The marginal score distributions are also similar: annotator A assigns 0 to 17.37% of cases and 1 to 82.63%, while annotator B assigns 0 to 17.82% and 1 to 82.18%. Table 5 summarizes the confusion matrix.

Table 5: Inter-rater agreement between the first annotator (A) and second annotator (B) after filtering construction failures. Scores are binary: 0 denotes failure and 1 denotes pass.

	B = 0	**B = 1**
A = 0	91	24
A = 1	27	520

The total disagreement rate is 7.7% (51/662). Among the disagreements, A is stricter in 24 cases (A=0, B=1), while B is stricter in 27 cases (A=1, B=0), suggesting that the two annotators have broadly balanced strictness.

F Prompt Templates

This section provides the core prompt templates used by DIAGEVAL's supervisor module. All prompts produce structured JSON output for downstream processing. Full verbatim prompts are available in the supplementary code.

[4] `https://atoms.dev/`

F.1 Failure Diagnostic Summary (FDS) Prompt

The FDS prompt takes a compressed failed trajectory and produces the initial diagnostic record.

Failure Diagnostic Summary Prompt

```
You are a test replay supervisor.  A GUI agent just finished its first-round test run and the
result is FAILURE. You are given the compressed trajectory.
Analyze the trajectory and answer three questions in order:
1.  **Why did it fail?** - Is it the agent's fault (wrong strategy, wrong clicks) or the
environment's fault (app broken, page not loading, UI non-functional)?
2.  **Should we retry?** - Will a second attempt with a different strategy likely succeed, or is
retrying pointless?
3.  **If retry, from which step?** - Which step is the last known-good state to restart from, and
what should the agent do differently?
## How to read the trajectory
- **"action"** = the raw pyautogui command executed (click, scroll, key press, etc.).
- **"summary"** = what the agent INTENDED to do.  This is NOT confirmation the action succeeded.
- You MUST use the **final result evidence** to retroactively judge earlier steps.  If evidence
says "all elements were unresponsive", then earlier clicks on those elements also failed - even if
summaries sound optimistic.
- A click + optimistic summary != success.  Only mark a step as successful if there is
corroborating evidence.
## Failure type definitions
- **agent**:  The application works (at least partially), but the agent chose wrong actions,
clicked wrong elements, used a bad strategy, or gave up too early.  Retry with a better approach is
likely to help.
- **env**:  The application itself is broken - page didn't load, JS not running, all UI elements
non-responsive, blank/black screen, or critical infrastructure failure.  Retrying the same test is
unlikely to help without fixing the environment.
- **ambiguous**:  Cannot determine clearly; some elements worked but the failure pattern doesn't
clearly point to agent or environment.
## Task under test
{task_desc}
## Trajectory
last_completed_iter:  {last_completed_iter}
### Step-by-step timeline (action + summary pairs):
{timeline_text}
### Final result:
{result_history_text}
## Required output
Return ONLY a single JSON object (no markdown fences):
{
  "failure_type":  "<agent | env | ambiguous>",
  "fail_reason":  "<2-3 sentences:  why did the test fail?  What went wrong?>",
  "should_retry":  <true | false>,
  "retry_reason":  "<1-2 sentences:  why retry will/won't help>",
  "restart_from_iter":  <int, 0-based step index to restart from; set to 0 if should_retry is false
or if no step succeeded>,
  "restart_explanation":  "<2-3 sentences:  why restart from this step, and what should the agent
do differently>"
}
## Examples
Agent failure (retry worthwhile):
{
  "failure_type":  "agent",
  "fail_reason":  "The agent successfully loaded the page and opened the date picker, but then
repeatedly clicked the same start-date field without ever selecting an end date.  The date range
was never completed, so the display info never updated.",
  "should_retry":  true,
  "retry_reason":  "The application is functional.  The agent just needs a better strategy for
selecting both start and end dates.",
  "restart_from_iter":  4,
  "restart_explanation":  "Step 4 successfully opened the start-date picker (confirmed by the date
value changing in step 5).  Restart here and immediately select a start date, then switch to the
end-date field - do not re-click the start-date field."
}
Environment failure (retry not worthwhile):
{
  "failure_type":  "env",
  "fail_reason":  "No UI element responded to any interaction throughout the entire session.
Buttons, dropdowns, and scroll all failed.  The page appears to have loaded as a static render
without JavaScript execution.",
  "should_retry":  false,
  "retry_reason":  "The application is completely non-functional.  Retrying with the same
environment will produce the same result.",
  "restart_from_iter":  0,
```

```
  "restart_explanation":  "No step produced any verifiable progress.  If the environment is fixed,
start from scratch."
}
Ambiguous failure:
{
  "failure_type":  "ambiguous",
  "fail_reason":  "The agent managed to interact with some elements (dropdown worked, scroll
worked) but the core Upload File button was consistently unresponsive across 6 attempts at
different coordinates.",
  "should_retry":  true,
  "retry_reason":  "Some UI elements work, so the app may be partially functional.  A different
upload approach (drag-and-drop, keyboard shortcut) might succeed.",
  "restart_from_iter":  2,
  "restart_explanation":  "Step 2 confirmed the page is partially interactive (dropdown responded).
Restart here and try file upload via keyboard shortcut (Ctrl+O) or drag-and-drop instead of
clicking the button."
}
## Your answer (JSON only):
```

Operational representation of the uncertainty profile. The SOU taxonomy in the main text defines conceptual evidence channels; we do not assume that each failure belongs strictly to only one SOU. In implementation, the uncertainty profile h_k is represented as an operational routing profile rather than as a four-dimensional probability vector over SOUs. Concretely, we write $h_k = (c_k, w_k)$, where c_k is a discrete failure category inferred from the FDS failure reason, and w_k is a normalized distribution over the three diagnostic branch dimensions $\mathcal{D} = \{A, B, C\}$. The FDS first produces structured fields such as `failure_type`, `fail_reason`, `restart_from_iter`, and `restart_explanation`.

F.2 Failure Category Classification Prompt

After the FDS is produced, a second inference pass expands the SOU-driven probing strategy into three probe types and five concrete subcategories, each corresponding to a specific diagnostic objective and intervention pattern (as mentioned in §4.2). Each category is anchored by 2–3 concrete examples in the prompt (e.g., for `wrong_target`: "The agent clicked the preset text label instead of the actual radio button circle, so the resize was never applied"). The classified category maps to empirical per-dimension success priors (Table 6) for branch budget allocation.

Failure Category Classification Prompt

```
Classify the agent's failure reason into exactly ONE category.
## Categories
- **insufficient_exploration**:  The agent gave up too early or failed to navigate/scroll
enough.  It drew premature conclusions without fully exploring the page or available UI.
- **wrong_strategy**:  The agent located the correct elements but used the wrong technique,
algorithm, or execution approach (wrong key timing, wrong game strategy, wrong sequence).
- **wrong_target**:  The agent interacted with the wrong element, wrong coordinates, or
misidentified what a UI component does.  The element itself may be correct but the agent
pointed at the wrong thing.
- **env_boundary**:  The feature may not be implemented, the app has a broken component,
or the UI is non-responsive regardless of what the agent tries.
- **unknown**:  The failure does not clearly fit any of the above categories.
## Examples
[insufficient_exploration]
"The agent only pressed 'pagedown' twice and then concluded no carousel exists.  It never took a
screenshot to confirm the page state."
"The agent completed the survey but could not find dimension percentage scores on the results page.
It did not scroll through the full results page."
"The agent scrolled through the timeline but did not explore all UI controls or buttons that might
reveal comparison photos."
[wrong_strategy]
"The agent moved the paddle briefly (0.5 seconds) and then the ball was lost.  It never implemented
continuous paddle control to keep the ball alive."
"The agent repeatedly anchored on card (0,0) paired with every other card - an invalid memory match
strategy since both cards need to be different."
"The agent performed hard drops without attempting to clear lines, so the LINES counter remained at
0."
[wrong_target]
"The agent repeatedly attempted to fill form fields using incorrect coordinates, resulting in
validation errors."
```

```
"The agent clicked the preset text label instead of the actual radio button circle, so the resize
was never applied."
"The agent attempted to drag a locked (correctly-placed) puzzle piece instead of testing an
unlocked piece."
[env_boundary]
"Clicking a tag on a note card navigated to the detail page instead of filtering - the tag
filtering feature may not be implemented."
"The agent waited 105 seconds but no inactivity prompt appeared - the timeout threshold may be
longer than tested or not implemented."
"Every time a shape property value was committed, the selected shape disappeared from the canvas -
likely an application bug."
## Failure reason to classify
{fail_reason}
Return JSON only, no markdown:
{"category":  "insufficient_exploration | wrong_strategy | wrong_target | env_boundary | unknown"}
```

Table 6: Empirical per-dimension success counts by failure category, used for SOU-driven dimension allocation. Source: 103 branches across the 41-case RDB pilot set.

Failure category	Dim A	Dim B	Dim C
`insufficient_exploration`	1	7	0
`wrong_strategy`	3	0	0
`wrong_target`	2	0	0
`env_boundary`	2	1	0
`unknown`	0	0	0

Source of dimension priors. The counts in Table 6 are derived from the 41-case RDB pilot set, consisting of 41 agent-fail cases and 103 executed diagnostic branches. The table reports only successful-branch counts; failed executed branches are part of the 103-branch source pool but do not contribute to the cells. We use these counts as lightweight routing priors for SOU-guided branch-dimension allocation. The 429-case RDB main test set and WDJ-U are held out from this pilot source and have zero case-level overlap with it.

To check whether these pilot-derived priors reflect stable diagnostic structure, we further compare the dominant successful dimension on held-out evaluation logs in Table 7. For the categories with clear pilot signal, the dominant successful dimension is consistent across the pilot, the RDB main-test FN set, and the WebDevJudge-Unit FN set: `wrong_strategy`, `wrong_target`, and `env_boundary` are all dominated by Dimension A. Thus, the pilot counts serve as coarse dimension-level routing priors rather than case-specific statistics. We apply Laplace smoothing before converting the counts into allocation weights, so every diagnostic dimension retains non-zero probability.

Table 7: **Held-out consistency of pilot-derived dimension priors.** Each entry reports the dominant successful dimension followed by the raw successful-branch counts along Dimensions A/B/C. For example, A $(3, 0, 0)$ means that Dimension A is the dominant dimension, with 3 successful branches in Dimension A, 0 in Dimension B, and 0 in Dimension C.

Failure category	Pilot	RDB Main Test	WDJ-U
`wrong_strategy`	A $(3, 0, 0)$	A $(4, 1, 0)$	A $(20, 0, 1)$
`wrong_target`	A $(2, 0, 0)$	A $(5, 0, 1)$	A $(7, 1, 0)$
`env_boundary`	A $(2, 1, 0)$	A $(4, 1, 0)$	A $(9, 2, 0)$

F.3 Branch Generation Prompt

The branch generation prompt (used in §4.2) instructs the executor agent to produce N candidate plans across the three operator dimensions. The prompt provides:

- The task description and failure analysis from the FDS.
- The trajectory tail (last steps before failure) for context.
- The current screenshot at the fork-point state.
- Per-dimension guidance and minimum count constraints (e.g., "Dimension A minimum: 3, Dimension B minimum: 1, Dimension C minimum: 1").

Each generated plan is a JSON object with fields: `dimension/type` (A/B/C), `title` (short identifier), `plan` (step-by-step concrete strategy), and `reason` (why this plan addresses the identified failure). The dimension guidance for Type A is dynamically generated from the FDS (`restart_explanation`); Type B and C use fixed templates targeting viewport expansion and interactability probing respectively.

Trajectory Reuse and Diagnostic History. Each executed probe contributes a trajectory fragment consisting of the resumed context, the action sequence executed under the probe, the resulting UI observations, and the terminal outcome. We append the probe type, realized trajectory fragment, and success/failure outcome to the diagnostic history H_k. When resuming from an intervention point, the executor replays or re-establishes the prefix needed to recover the local UI context around t_k^*, and launches the new probe from that context rather than restarting the entire case.

F.4 Information-Value Branch Ranking Prompt

Branch ranking is conditioned on the current Failure Diagnostic Summary (FDS) $\mathcal{F}_k$ together with local execution context, including the visible UI state at t_k^* and prior probe outcomes in $\mathcal{H}_k$, rather than unconstrained free-form text alone. The branch selection module (§4.3) instructs the supervisor to rank and select the top-K executable branches from the candidate pool by approximating the EIG ranking score in Eq. 2. Each candidate branch is evaluated with respect to both attribution relevance and executability. The prompt first applies four validity checks:

1. **Root-cause fit**: does the branch directly address the diagnosed failure mode or dominant SOU?
2. **Feasibility**: can the branch be executed from the current checkpoint and visible interface state?
3. **Non-redundancy**: does the branch avoid trivially repeating actions that already failed without introducing new evidence?
4. **Specificity**: is the branch a concrete, executable action plan rather than a vague strategy?

Branches that fail these checks are removed from $\mathcal{A}_{\text{valid}}(\mathcal{S}_k)$. For each remaining branch, the supervisor estimates the branch outcome under the two latent attribution hypotheses: whether the branch is likely to succeed if the original failure was AGENTFAIL, and whether the same outcome would remain likely if the environment is genuinely blocked. Operationally, the prompt asks for structured fields:

- `valid`: whether the branch passes the feasibility, specificity, and non-repetition checks.
- `p_success_agent`: estimated likelihood of success under AGENTFAIL.
- `p_success_env`: estimated likelihood of an observed success signal under ENVFAIL.
- `eig_rationale`: why the success/failure outcomes would or would not separate the two attribution hypotheses.
- `rank`: the final order among valid branches by expected attribution-entropy reduction.

These numeric likelihoods are not treated as globally calibrated probabilities. Instead, they instantiate the local world model used for branch prioritization at selection time. Formally, for a branch b of type $d(b)$, we use

$$\begin{aligned} w_b &= P(\texttt{verified_success} \mid \textsc{AgentFail}, b), \\ \beta_{d(b)} &= P(\texttt{verified_success} \mid \textsc{EnvFail}, b), \\ \gamma_{d(b)} &= P(\texttt{fail} \mid \textsc{AgentFail}, b), \end{aligned}$$

where w_b captures recovery under evaluator-side failure, $\beta_{d(b)}$ captures success that can still occur under environment-side failure, and $\gamma_{d(b)}$ captures failure compatibility under evaluator-side failure. The actual observed outcome is incorporated afterward through the branch-typed score update in Eq. 3. Therefore, selection-time information value acts as a planning signal for branch ordering, while the realized entropy reduction is determined only after branch execution.

G Case Study and Examples

G.1 Diagnostic Pipeline Case Example

We walk through a complete DIAGEVAL diagnostic cycle on a concrete case from RealDevBench to illustrate how the components from §4 interact in practice.

Task. "Check saturation adjustment functionality" on a film-effect image processing website.

Failed trajectory (Round 1). The agent attempted to interact with a saturation slider before loading any image into the editor. Clicks at coordinates (960, 692) on the "Choose File" button did not trigger a file dialog. After repeated failed clicks and three action-error retries, the agent reported failure without trying alternative approaches. The trajectory tail:

```
Step 2:  click(960, 692) -- Click ''Choose File'' to upload image
Step 3:  click(1850, 442) -- Click saturation slider
Step 4:  click(960, 692) -- Retry ''Choose File'' button
Steps 5-7:  [action error, retrying] -- Report failure
```

FDS output.

> **failure_type**: agent **failure_category**: `wrong_target`
> **fail_reason**: "The agent attempted to interact with a saturation slider before any image was loaded into the editor, and the file upload button clicks at (960, 692) did not successfully open a file dialog or load an image. The agent never established a valid starting state (an image in the editor) before trying to test the saturation adjustment feature."
> **should_retry**: true **initial** p_{env}: 0.10
> **fork-point** (t_0^*): step 0 (complete restart)
> **restart_explanation**: "No step produced verifiable progress, so restart from step 0. The agent should first take a screenshot to understand the current UI state, then locate and click the correct file upload button, confirm an image is loaded in the editor, and only then interact with the saturation slider."

The low initial $p_{\text{env}} = 0.10$ reflects the supervisor's assessment that this is likely an agent-side grounding error rather than a broken application.

Dimension allocation. Given `failure_category = wrong_target`, the empirical priors from Table 6 assign weights w_A=0.6, w_B=0.2, w_C=0.2, concentrating the branch budget on alternative transitions (Type A), because `wrong_target` failures are most likely resolved by finding the correct interactive element.

Generated branches (5 candidates).

- **A-0**: *OCR-based button targeting.* Use visual text recognition to find the "Upload"/"Open"/"Choose File" button instead of hardcoded coordinates; click the detected bounding box center.
- **A-1**: *Parent container interaction.* Click the larger drop-zone container surrounding the upload area with a 50px offset from the previously failed point.
- **A-2**: *Alternative image source.* Look for a "Sample Image" gallery or "File" menu in the navigation bar to load an image without using the upload button.
- **B-3**: *Viewport expansion.* Scroll down 500px to check if the upload button or saturation slider is partially off-screen.
- **C-4**: *Interactability probe.* Hover over (960, 692) to check cursor change, confirming whether the element is actually interactive before clicking.

Selected branches (K=3). The supervisor selected A-0, A-2, A-1 in priority order, all of which are Type A, consistent with the `wrong_target` SOU hypothesis. Under the local world model, these branches have high EIG score because success under any of them would strongly support AGENTFAIL: the application would be shown reachable once the upload step is grounded correctly. Their failure would also provide useful negative evidence because three distinct action-form realizations would have failed from the same fork point. The B and C candidates were deprioritized because viewport expansion and hover-only interactability checks are less likely to separate AGENTFAIL from ENVFAIL for this diagnosed grounding error.

This example illustrates the full diagnostic pipeline: the FDS produces a structured failure analysis with SOU hypothesis, branch generation proposes candidate probes over diagnostic dimensions, information-value selection prioritizes probes expected to maximally shift the attribution signal, and executed outcomes update the internal attribution score.

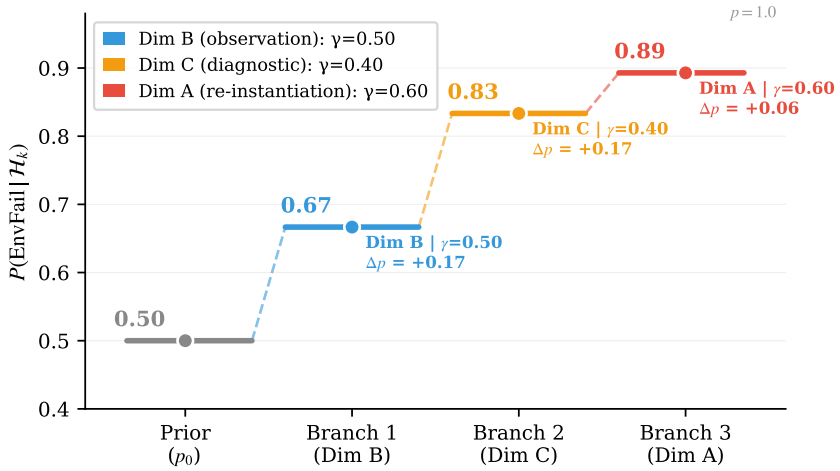


Figure 5: **Attribution-score update across diagnostic branches in the running example.** Each branch yields the same binary outcome (fail), but produces a different update magnitude due to branch-typed likelihoods γ_d, consistent with Eq. 3.

Intervention-point execution details. Operationally, resuming from t_k^* means recovering the local UI context needed for the next diagnostic action, not teleporting to an exact simulator state. When available, the system replays the verified trajectory prefix up to t_k^*; otherwise it reconstructs the local context by navigating back to the same page / panel configuration before executing the probe. This design is sufficient for diagnosis because probes are defined over the local neighborhood of t_k^*, but it can be imperfect when the environment contains hidden session state or asynchronous effects that are difficult to reproduce exactly.

G.2 Illustrative Running Example

Running example. This is a fixed three-probe illustration. With the default $\tau_{\text{env}} = 0.70$, online diagnosis would stop after the second failure; the third update only illustrates the scoring rule. Consider the test point *"verify that clicking the timeline sets the video end point"* in a VideoClipper application. The initial evaluation fails because the file-upload button is non-responsive—the evaluator cannot reach the timeline editor. DIAGEVAL classifies this as `env_boundary` and allocates branch budget with emphasis on Dimension A.

For this illustration, we use an initial attribution score $p_{\text{ENV}}^{(0)} = 0.50$ and consider three probes spanning all three evidence channels:

1. **Branch 1 (Dim B, observation probe):** scroll the page to search for a pre-loaded sample video or alternative entry point. *Fails.* $p_{\text{ENV}}^{(1)} = 0.50 \xrightarrow{\gamma_B=0.50} 0.67$ $(\Delta p = +0.17)$.
2. **Branch 2 (Dim C, diagnostic probe):** Tab-cycle through focusable elements to test whether the upload button accepts keyboard activation. *Fails.* $p_{\text{ENV}}^{(2)} = 0.67 \xrightarrow{\gamma_C=0.40} 0.83$ $(\Delta p = +0.17)$—this probe uses the smallest γ value, making failure hardest to reconcile with evaluator-side error and therefore providing the strongest evidence toward environment-side blockage.
3. **Branch 3 (Dim A, re-instantiation probe):** hover and double-click the upload button with alternative timing. *Fails.* $p_{\text{ENV}}^{(3)} = 0.83 \xrightarrow{\gamma_A=0.60} 0.89$ $(\Delta p = +0.06)$—the smallest shift, because γ_A=0.60 is the largest likelihood, making a failed re-instantiation most compatible with evaluator miss. The diminishing increment also reflects score saturation: as p_{ENV} approaches 1, each additional failure contributes less marginal evidence.

After the three illustrative updates, the internal attribution score $p_{\text{ENV}} \approx 0.89$ strongly favors environment-side blockage (Figure 5). All three branches produce the same binary outcome (fail), yet the branch-typed observation model converts them into different evidence strengths via the dimension-specific likelihoods γ_d. This is the core mechanism that enables structured attribution evidence: uniform retry, which treats every failure identically, cannot distinguish these cases.

G.3 Additional Case Studies

Case 3: Trajectory anchoring (OfficeEmployeeTaskAllocator) **Observed mismatch and resume-only behavior.** Baseline sees dashboard showing 6 tasks but "All Tasks" list shows only 4. NR+IE ($\times 1$, $\times 2$) resumes and repeats the same observation three times, anchored to "count rows in All Tasks list."

DIAGEVAL branch and outcome. NR ($\times 2$) discovers an alternative via status filters: Not Started (1) + In Progress (3) + Completed (2) = 6.

Takeaway. Resume context without branching locked the agent into a failed verification strategy.

Case 4: Multi-step architectural reasoning (Professional Portfolio) **Observed failure and resume-only behavior.** Baseline tries Download button, Ctrl+S, More actions menu, right-click, and no download is triggered. NR ($\times 2$) repeats the same button-clicking pattern with no memory of what was tried.

DIAGEVAL branch and outcome. DIAGEVAL ($\times 2$) identifies that direct button clicks are ineffective and generates a Dim A branch: open "Download Resume" $\rightarrow$ PDF viewer loads $\rightarrow$ click download icon within viewer $\rightarrow$ save dialog triggered.

Takeaway. This two-step approach requires understanding the application's document viewing architecture.

Case 5: Environment pre-conditioning (CSS Animation Playground) **Observed failure.** No method succeeds in $\times 1$ because the animation completes before the pause button can be tested.

DIAGEVAL branch and outcome. DIAGEVAL ($\times 2$) diagnoses the temporal root cause and generates a Dim B branch: set "Iteration Count" to Infinite *first*, then click pause. The animation stops and the button icon changes.

Takeaway. The environment is functional but requires a specific interaction ordering.

H Additional Experiments

H.1 Multi-Round Recovery and Diagnostic Ablations

Experimental setting. We extend the recovery ablation to two benchmarks and two diagnostic budgets to examine whether additional diagnostic rounds improve recovery beyond repeated execution. We use the same initially false-negative cohorts as the main evaluation: 114 cases on WDJ-U and 132 on RDB. For each dataset, we compare (i) naive retry (NR), which repeats execution without diagnosis; (ii) NR+IE, which introduces the Failure Diagnostic Summary (FDS) and fork-point resumption but not SOU-guided targeted branching; and (iii) the full DIAGEVAL diagnostic procedure. The $\times 1$ and $\times 2$ settings allow one and two diagnostic rounds, respectively, with the same benchmark-specific execution backbone and evaluation setup. We measure FN recovery rate, cost per originally false-negative case, and entropy reduction ΔH in the internal attribution score. This FN-subset analysis complements, but does not replace, the TN-inclusive end-to-end component ablation in Table 2.

Results and analysis. With one diagnostic round, full DIAGEVAL recovers 45.6% of WDJ-U FNs, compared with 17.5% for NR and 20.2% for NR+IE; on RDB the corresponding rates are 47.0%, 32.6%, and 37.1%. Increasing the diagnostic budget to two rounds raises full-method recovery to 59.6% on WDJ-U and 62.1% on RDB, surpassing both NR (28.1%/46.2%) and NR+IE (30.7%/43.2%). Relative to its own one-round setting, the full method gains 14.0 and 15.2 percentage points of FN recovery on WDJ-U and RDB, respectively. This additional coverage requires more diagnosis: FN-subset cost increases from \$0.122 to \$0.163 on WDJ-U and from \$0.183 to \$0.234 on RDB. The score entropy reduction also rises from 0.274 to 0.410 on WDJ-U and from 0.305 to 0.325 on RDB. Thus, the two-round setting offers a recovery–cost trade-off on both benchmarks, and the gains over retry-based controls are not explained by additional attempts alone. These results characterize recovery within initially missed cases; TN flips and overall accuracy are evaluated separately in the main experiments.

Table 8: Multi-round FN recovery and diagnostic ablation on two benchmarks. *FDS*: failure diagnosis; *Fork*: fork-point resume; *Branch+Att.*: SOU-guided branching with attribution score update. *Rec.%*: FN recovery rate; *Cost*: USD per case; ΔH: entropy reduction in the internal attribution score from a neutral prior (higher indicates stronger attribution evidence). Results are grouped by the one-round and two-round diagnostic budgets; boldface highlights full-method recovery and entropy reduction.

	Components			RDB (n=132)			WDJ-U (n=114)		
Method	FDS	Fork	Branch+Att.	Rec.% ↑	Cost ↓	ΔH ↑	Rec.% ↑	Cost ↓	ΔH ↑
DIAGEVAL w/o Diagnose (NR, ×1)	–	–	–	32.6	0.030	0.001	17.5	0.059	0.046
DIAGEVAL w/o Branch (NR+IE, ×1)	✓	✓	–	37.1	0.018	0.024	20.2	0.053	0.063
DIAGEVAL (×1)	✓	✓	✓	**47.0**	0.183	**0.305**	**45.6**	0.122	**0.274**
DIAGEVAL w/o Diagnose (NR, ×2)	–	–	–	46.2	0.057	0.012	28.1	0.115	0.120
DIAGEVAL w/o Branch (NR+IE, ×2)	✓	✓	–	43.2	0.045	0.024	30.7	0.130	0.141
DIAGEVAL (×2)	✓	✓	✓	**62.1**	0.234	**0.325**	**59.6**	0.163	**0.410**

Table 9: Three independent TN-inclusive full-cohort WDJ-U runs ($N = 502$ per run), with the initial evaluation regenerated in each run. FN/TN denotes the initial false-negative and true-negative counts; Rec./Flip denotes recovered FNs and flipped TNs. The ×2 counts are cumulative over both diagnostic rounds. Accuracy is reported in percent. The final row reports mean accuracy ± sample standard deviation across runs.

	Initial evaluation		DIAGEVAL ×1		DIAGEVAL ×2	
Run	FN/TN	Acc.	Rec./Flip	Acc.	Rec./Flip	Acc.
1	115/186	69.72	50/18	76.10	65/23	78.09
2	115/184	69.32	51/19	75.70	66/25	77.49
3	113/188	70.52	51/17	77.29	67/22	79.48
Avg.	–	69.85 ± 0.61	–	76.36 ± 0.83	–	78.35 ± 1.02

H.2 End-to-End Run-to-Run Variability

We conduct three independent TN-inclusive full-cohort WDJ-U runs, regenerating the initial evaluation in each run and diagnosing both the resulting FNs and TNs. Table 9 reports the individual runs and the mean ± sample standard deviation. Mean initial, ×1, and ×2 accuracies are $69.85 \pm 0.61\%$, $76.36 \pm 0.83\%$, and $78.35 \pm 1.02\%$, respectively.

H.3 Cross-Framework Transfer

We further evaluate whether DIAGEVAL can improve a different GUI-agent framework without retuning. Specifically, DIAGEVAL is developed with AppEvalPilot as the source framework, and we transfer the same diagnostic procedure to UI-TARS as the target framework. The diagnostic components, including failure parsing, branch generation, branch selection, probing, and attribution-score update, are kept unchanged; only the underlying GUI-agent framework is replaced.

Figure 4 reports absolute accuracy changes in percentage points over each framework's own single-pass baseline. On WDJ-U, UI-TARS obtains 54.8% accuracy under single-pass evaluation. Adding DIAGEVAL raises accuracy to 72.1% under the ×1 setting and 76.9% under the ×2 setting, corresponding to absolute gains of +17.3 and +22.1 percentage points. On RDB, UI-TARS improves from 52.9% to 67.1% and 74.6%, giving absolute gains of +14.2 and +21.7 percentage points. These results show that DIAGEVAL provides substantial absolute accuracy improvements beyond the framework on which it was developed, indicating that the diagnostic procedure transfers across GUI-agent frameworks rather than relying on AppEvalPilot-specific behavior.

To test whether the gains of DIAGEVAL can be explained by repeated sampling alone, we compare it against a diagnosis-free *naive retry* baseline on the same WDJ-U originally false-negative subset used in the main evaluation (114 cases). For a fair comparison, naive retry uses the same backbone (Gemini 3 Flash Preview) and the same AppEvalPilot framework as DIAGEVAL, so that any difference reflects the diagnostic mechanism rather than backbone capacity. Naive retry reruns each task from scratch, without failure parsing, branch construction, or attribution-score updates. We consider three naive-retry baselines: *Retry-1* (a single retry), *Majority-3* (majority vote over three attempts),

and *Best-of-3*, which counts a case as recovered if any attempt succeeds. Best-of-3 is therefore an optimistic sampling-only baseline.

Table 10: **Recovery on WDJ-U originally false-negative cases.** All methods are evaluated on the same 114 cases. Retry-1 denotes a single naive retry; Majority-3 denotes majority vote over three attempts; Best-of-3 counts a case as recovered if any attempt succeeds.

Method	Recovered / Recovery Rate
Naive retry (Retry-1)	$20/114$ (17.5%)
Naive retry (Majority-3)	$16/114$ (14.0%)
Naive retry (Best-of-3)	$33/114$ (28.9%)
DIAGEVAL ($\times 1$)	$52/114$ (45.6%)
DIAGEVAL ($\times 2$)	$68/114$ (59.6%)

Table 10 shows that DIAGEVAL's gains cannot be attributed to repeated sampling alone. Under naive retry, recovery remains limited on false negatives: one retry recovers 17.5%, Majority-3 recovers 14.0%, and Best-of-3 recovers 28.9%. The lower Majority-3 result also suggests that majority aggregation can introduce additional voting noise when retry outcomes are unstable across runs. Even against the optimistic Best-of-3 baseline, DIAGEVAL remains substantially stronger, reaching 45.6% recovery after one diagnostic round and 59.6% after two. This shows that DIAGEVAL gains its advantage by identifying attribution-informative sources of uncertainty and converting them into targeted probes that help separate AGENTFAIL from ENVFAIL, thereby reducing attribution uncertainty, rather than by retrying more times alone.

H.4 Branch Prioritization

We evaluate whether belief-conditioned branch prioritization improves diagnostic efficiency. The analysis is conducted on the WebDevJudge-Unit FN set. Among the 114 FN cases, 50 are successfully recovered by diagnostic probing, and these 50 recovered cases form the evaluation set for branch ordering. This evaluation is conditional on branch selection: for each case, the LLM-selected branch set already contains at least one branch that can recover the task. We therefore keep the selected branch set fixed and vary only the execution order, so that any performance difference reflects the efficiency of ordering rather than the ability to select a successful branch.

Table 11: **Ordering performance on recoverable cases.** Different execution orders are compared with the selected branch set fixed. `First-branch success` denotes the fraction of cases recovered by the first executed branch.

Ordering strategy	First-branch success	Avg. probes / case	Total probes	Saving vs. random
Random ordering	34% (17/50)	2.00	100	–
LLM-default ordering	58% (29/50)	1.60	80	20%
EIG ordering	**66% (33/50)**	**1.48**	**74**	**26%**

Compared methods. We compare three ordering strategies: (i) **Random ordering**, which executes the selected branches in random order; (ii) **LLM-default ordering**, the order produced by the LLM judge and used as the default execution order in DIAGEVAL; (iii) **EIG ordering**, our belief-conditioned ordering strategy that ranks branches by EIG score over the attribution score.

As shown in Table 11, EIG ordering achieves the highest first-branch success rate and the lowest probe cost. Relative to LLM-default ordering, it improves first-branch success from 58% to 66% and reduces the total number of probes from 80 to 74. Compared with random ordering, EIG nearly doubles the first-branch success rate from 34% to 66% and saves 26% of probe executions. These results suggest that, once the LLM judge has selected semantically plausible diagnostic branches, EIG-based ordering can more effectively place recoverable branches earlier in the execution sequence.

H.5 Model-Call Composition in FN Retry

Figure 6 reports the Gemini/Claude call ratio per case across the FN retry pipeline. This ratio reflects the division of labor in DIAGEVAL: Gemini is used for the GUI-agent execution loop, while Claude is used for supervisor-side diagnosis, retry-plan generation, and checkpoint analysis. In the first run, both benchmarks show a high ratio (≈18–19), since each case mainly invokes Gemini for the agent rollout and only requires lightweight Claude-side supervision. After Retry ×1, the ratio decreases on both WDJ-U and RDB, reflecting additional Claude calls introduced by failure verification, diagnostic planning, and restart-point analysis.

Gemini calls remain relatively stable across stages (≈11–13 per case), while the ratio varies mainly with Claude-side supervision. At Retry ×2, fewer Claude calls raise the ratio on WDJ-U, whereas RDB retains greater supervisor involvement. Thus, the figure complements the cost analysis by showing that changes in call composition mainly reflect diagnostic supervision.

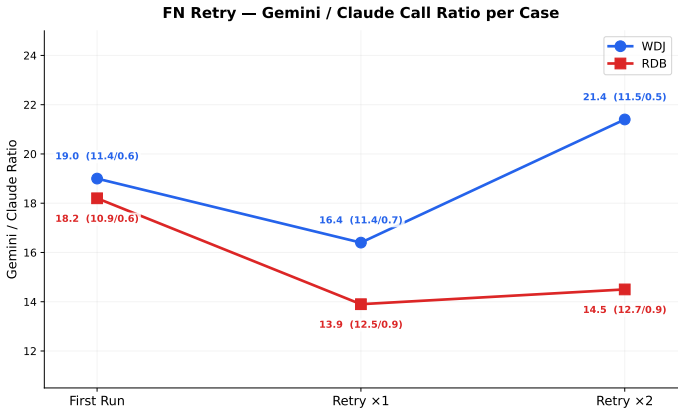


Figure 6: **Gemini / Claude call ratio per case across the FN retry pipeline.** The ratio is computed as the average number of Gemini agent calls divided by the average number of Claude supervisor calls per case. Parenthetical values report the rounded underlying averages (Gemini avg. / Claude avg.). Gemini calls remain relatively stable across stages, while most variation in the ratio comes from Claude-side diagnostic supervision.

H.6 Sensitivity to Hyperparameters

Parameter-setting protocol. The default update and stopping parameters are empirical scoring constants rather than fitted probabilities. We select $(w_0, \beta_0) = (0.60, 0.20)$, $(\gamma_A, \gamma_B, \gamma_C) = (0.60, 0.50, 0.40)$, and $\tau_{\text{env}} = 0.70$ through lightweight sanity checks on the 41-case RDB pilot set, while holding out the 429-case RDB main test set for final reporting. Table 6 provides the pilot-derived routing sanity check for branch-dimension allocation, whereas Tables 12 and 13 test whether the reported trends are stable around the default scoring constants.

Table 12: **Sensitivity to likelihood parameters for DIAGEVAL (×1).** Results are reported in terms of ΔH under the branch-typed score update. The row $w_0 = \beta_0 = 0.50$ is a *non-informative reference*, in which success/failure outcomes alone provide no directional preference between AGENTFAIL and ENVFAIL; any remaining attribution signal therefore comes from the branch-typed probing structure. The default setting $(w_0, \beta_0) = (0.60, 0.20)$ is used in the main experiments.

w_0 / β_0	RDB	WDJ-U
$w_0 = \beta_0 = 0.50$ (non-inform. ref.)	0.269	0.254
$w_0 = 0.50$, $\beta_0 = 0.10$	0.365	0.361
$w_0 = 0.60$, $\beta_0 = 0.20$ (default)	**0.305**	**0.274**
$w_0 = 0.70$, $\beta_0 = 0.20$	0.321	0.315
$w_0 = 0.80$, $\beta_0 = 0.10$	0.428	0.429
$w_0 = 0.90$, $\beta_0 = 0.05$	0.531	0.536

Table 13: **Sensitivity of the ENVFAIL stopping threshold τ_{env}.** We vary the early-stopping threshold τ_{env} for the accumulated ENVFAIL belief and report its effect on diagnostic efficiency and recoverable-case safety. `Saved branches` denotes the number and percentage of diagnostic branches skipped by threshold-based early stopping. `Correct RC recovery` counts recoverable cases that are still successfully recovered, while `RC truncation error` counts recoverable cases that are incorrectly stopped before recovery.

Threshold τ_{env}	Saved branches	Correct RC recovery	RC truncation error
0.65	83 (43.9%)	37/50	13/50
0.70	**67 (35.4%)**	**50/50**	**0/50**
0.75	40 (21.2%)	50/50	0/50
0.80	20 (10.6%)	50/50	0/50
0.85	1 (0.5%)	50/50	0/50
0.90	0 (0.0%)	50/50	0/50

Table 14: WDJ-U $\times 1$ sensitivity to failure likelihood q and log-evidence tempering λ. Costs are reported per originally false-negative diagnostic case.

q	λ	Acc.	FN recovered	TN flips	Probes	Cost	AUC
1.0	1.0	76.29%	52	20	2.38	.122	.720
0.9	1.0	76.49%	53	20	2.45	.125	.719
1.0	0.5	76.49%	53	20	2.56	.130	.715
0.9	0.5	76.49%	53	20	2.61	.133	.713

Sensitivity to Likelihood Parameters We test whether the attribution signal depends strongly on a particular choice of likelihood parameters (Table 12). To align with the main method, we parameterize the branch-typed update by a base pair (w_0, β_0), where w_0 denotes the nominal probability of observing `verified_success` under AGENTFAIL, and β_0 denotes the nominal probability of observing success under ENVFAIL. Branch-specific terms w_b and $\beta_{d(b)}$ are instantiated from this base pair together with branch type.

Across settings, DIAGEVAL remains stable and retains non-trivial entropy reduction. Even at $w_0 = \beta_0 = 0.50$, where success/failure outcomes alone carry no attribution preference, DIAGEVAL still achieves $\Delta H \approx 0.25$–0.27. This indicates that attribution signal does not come only from raw success/failure outcomes, but also from the branch-typed probing structure itself. Increasing the separation between w_0 and β_0 yields larger ΔH, but also makes the update more sensitive to the assumed likelihood asymmetry. We therefore use $(w_0, \beta_0) = (0.60, 0.20)$ as the default setting in the main experiments.

Controlled $q \times \lambda$ evidence replay. To test the assumptions used in accumulating failure evidence, we conduct a 2×2 fixed-order replay on the frozen WDJ-U cohort ($N = 502$). We hold the initial evaluations, candidate and selected probes, EIG order, recorded probe outcomes, success-verification rule, and stopping threshold $\tau_{\text{env}} = 0.70$ fixed.

The default failure update in Eq. 3 corresponds to $q = \lambda = 1$. We vary the assumed ENVFAIL failure likelihood used in the heuristic update, $q \in \{1.0, 0.9\}$, and a log-evidence tempering coefficient $\lambda \in \{1.0, 0.5\}$. For a failed probe of type d, the log-odds increment becomes $\lambda \log(q/\gamma_d)$. Lowering q relaxes the default deterministic-failure assumption by weakening the evidence that a failed probe provides for ENVFAIL. Setting $\lambda = 0.5$ halves each log-evidence increment, testing sensitivity to potential overcounting of correlated probes. Attribution scores and stopping decisions are recomputed during replay, so the number of executed probes and cost may change even though the probe pool and its ordering are held fixed. The immediate-PASS rule remains unchanged. Table 14 reports the four configurations and their verdict/efficiency outcomes.

H.7 Supervisor Sensitivity

We conduct a matched repair-stage comparison on the same $N = 200$ WDJ-U cases and shared initial trajectories (69.6% accuracy), changing only the supervisor used for diagnosis and retry.

Table 15: Matched repair-stage supervisor comparison on WDJ-U ($N = 200$). All rows share the same initial trajectories and initial accuracy of 69.6%. Matched accuracies are accompanied by the corresponding three-run sample standard deviations.

Repair-stage supervisor	$\times 1$	$\times 2$	$\times 2$ **gain**
Claude Sonnet 4.6	$76.6 \pm 0.58\%$	$78.6 \pm 0.76\%$	+9.0 pt
GPT-5.4	$76.0 \pm 0.50\%$	$78.0 \pm 0.29\%$	+8.4 pt
DeepSeek-v4-pro	$75.7 \pm 0.29\%$	$77.5 \pm 0.29\%$	+7.9 pt

As shown in Table 15, all three supervisors improve the shared initial accuracy by 7.9–9.0 points, indicating that DIAGEVAL is not sensitive to the choice of supervisor.

In a separate three-run end-to-end evaluation on the 200-case subset, Qwen3.5-35B-A3B improves from 67.7% initially to 69.3% and 70.1% at $\times 1$ and $\times 2$, respectively, corresponding to a 2.4-point gain. We report it as complementary sensitivity evidence rather than as part of the matched one-factor comparison.

H.8 Independent Verification of Verdict Reversals

Screenshot audit. We first checked whether screenshots supported the revised PASS verdicts. Two human raters independently reviewed all 94 WDJ-U $\times 2$ reversals (68 recovered FNs and 26 TN flips), seeing only the original test description, expected result, and chronological post-action screenshots. Packages hid method/supervisor identities, FN/TN group, original/final verdicts, agent-reported results and evidence, reasoning, and attribution scores.

Raters cited the decisive frame and assigned one of three labels: *valid*, the expected result was visibly achieved, with persistence where required; *erroneous*, visible evidence contradicted success; or *insufficient*, the screenshots established neither success nor failure, including missing evidence of the required interaction or postcondition. Treating V as screenshot-supported and E+I as unverified, the raters agreed on 77/94 cases (81.9%, $\kappa = 0.520$): 62 jointly supported, 15 jointly unverified, and 17 disagreements (Table 16). Three-way agreement was 74.5% ($\kappa = 0.358$). 7 of 24 three-way disagreements concerned whether missing evidence contradicted success or was merely insufficient.

This post-hoc audit measures screenshot support rather than live task feasibility: apparently successful states may not establish that the required interaction occurred.

To assess whether screenshot-based judgments were supported by actual application behavior, we complemented the screenshot audit with live-app testing against the original task requirements and expected results.

Live-app adjudication of TN flips. A third method-blind human re-executed all 26 WDJ-U $\times 2$ TN-flip tasks from a clean state, three times per task. Of these 26 cases, 10 achieved the expected result in all three trials, whereas 16 were confirmed negative under this protocol. By comparison, screenshot raters A and B had labeled 15 and 18 of the 26 cases valid, respectively, indicating that screenshots can over-certify success. Among the 10 live-confirmed successes, 7 had also been labeled valid by both screenshot raters; the remaining 3 lacked unanimous screenshot support. Official benchmark labels remain unchanged in the primary results; all 26 are still counted as TN flips when computing end-to-end accuracy.

Random live-app audit of recovered FNs. To assess whether screenshot over-certification also affected recovered FNs, we drew a fixed-seed simple random sample of 25 from all 68 recovered FNs on WDJ-U $\times 2$. The sample contained 15 cases that both screenshot raters had labeled valid and 10 with at least one erroneous or insufficient judgment. A method-blind live-app rater tested each application three times and judged every trial against its original requirement and expected result.

Of these 25 cases, 24 succeeded in all three trials and 1 failed in all three (Table 17). Within this sample, 16/16 cases labeled valid by Rater A and 19/20 labeled valid by Rater B achieved the expected result in all three live trials. The single 0/3 case was `web_100_4`. Its expected result required both that typed text appear and that focusing the field change its `box-shadow` from inset to outset. The text appeared correctly, but the live-app rater found no required shadow change, so the case was not counted as successful. Screenshot Rater A had labeled it insufficient, while Rater B labeled it valid.

Table 16: Independent screenshot audit of WDJ-U $\times 2$ verdict reversals. V/E/I denote valid, erroneous, and insufficient. Valid/decidable is V/(V+E), excluding insufficient cases. Binary agreement and Cohen's κ compare V against E+I.

Group	n	**Rater A V/E/I**	**Rater B V/E/I**	**Valid/decidable A/B (%)**	**Binary agreement**	**Binary κ**
Recovered FNs	68	53/6/9	55/0/13	89.8/100.0	85.3%	0.551
TN flips	26	15/2/9	18/2/6	88.2/90.0	73.1%	0.428
Overall	**94**	**68/8/18**	**73/2/19**	**89.5/97.3**	**81.9%**	**0.520**

Table 17: Case counts by successful live trials on WDJ-U $\times 2$. The FN sample contains 25 of 68 recovered cases; the TN subset contains only the 16 previously confirmed negative cases, not all 26 TN flips.

Audit group	n	**0/3**	**1/3**	**2/3**	**3/3**
Random recovered-FN sample	25	1	0	0	24
Previously negative TN flips	16	16	0	0	0

Repeated trials and the success-verification boundary. To examine whether TN flips reflected occasional genuine success from unstable applications, we analyzed the three-trial outcomes for the 16 TN-flip cases previously confirmed negative. All 16 failed in every trial; none succeeded once or twice (Table 17). Thus, no intermittent success was observed in this subset under the three-trial protocol.

Case-level analysis identified a challenging verification boundary: an application may provide related functionality or a visually similar outcome while failing the task's specified interaction. For example, in `web_23_1`, clicking opened a similar menu, but the required hover-triggered behavior remained unavailable. During multi-probe diagnosis, distinguishing these outcomes requires retaining the original interaction constraints while integrating evidence from alternative exploratory branches.

The live-app audits support task feasibility in most sampled recovered FNs, but do not independently validate the original agent trajectories. They also highlight the importance of retaining the original interaction constraints and required persistent postconditions across diagnostic branches. Three-trial tests cannot rule out intermittent behavior in other cases or over longer runs.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: See abstract and introduction

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in the conclusion section.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide relevant information in experiment section.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide relevant link for the data and code.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See experiment section.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: See Appendix H.2 for the mean and standard deviation across repeated runs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experiments do not train new models. We report the LLM backbones, diagnostic budgets, per-case API costs, and call-composition analysis.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This paper has conducted in the paper conform.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We have discussed both positive and negative impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We do not have data or models that have a high risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have properly credited the creators or original owners of assets used in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This paper does not introduce new assets like models or data.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This paper does not include crowdsourcing experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification:We describe the usage of LLMs in the method and experiment sections.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.