APEX: ADAPTIVE PROGRESSIVE EXAMINATION OF AGENTIC WORKFLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM)-based agentic workflows have shown substantial gains over single agents on complex tasks, but they are harder to investigate when they underperform, since a weakness can lie in any aspect of any of their units: the agents, the communication strategy among them, the memory they share, and the tools through which they act. Existing failure attribution analyzes the trajectory of a run that has already failed and returns a single responsible agent and step, so nothing is learned until a failure has occurred, failures are attributed to agents alone, and the aspect in which the responsible unit is weak remains unidentified. We formulate *unit-level weakness localization*, which identifies the weak units of a workflow and the aspects in which they are weak before they fail in deployment. We present (1) APEX, which adaptively and progressively explores a workflow's execution trajectories, running probes that target the assertions of a dynamically generated rubric tree about which it is statistically least confident; (2) AXE, an execution engine that records what every unit was given and changed, and can perturb any unit in isolation; and (3) AXEBench, a labeled benchmark of perturbed workflows generated by AXE. On AXEBench, APEX correctly localizes 90.5% of injected weaknesses in agents, against 57.1% for the state of the art, and 51.5% across all units of the workflow, against 14.5%, as prior methods can only target agents; it identifies the weak aspect in 79.0% of the weaknesses it localizes. Guided by APEX, an LLM instructed to repair the workflow recovers more than twice as much of the lost performance as it does without guidance, and human experts validate both the verdicts APEX relies on and the benchmark it is evaluated on. Code and data are publicly available.[1]

1 INTRODUCTION

Large Language Model (LLM)-based *agentic workflows* have shown substantial improvement in solving complex tasks compared to single agents (Hong et al., 2024; Qian et al., 2024), and various frameworks now provide the foundation to build them (Wu et al., 2024; Li et al., 2023; Fourney et al., 2024). However, an agentic workflow is composed of more than its agents: it also comprises the communication strategy among them, the memory they share, and the tools through which they act, and each of these units can contribute to its success or failure (Huang et al., 2025; Fu et al., 2026; Orogat et al., 2027). These complexities make agentic workflows hard to investigate when they underperform, as the weakness can lie in any aspect of any unit, and a substantial share of failures originate in how the agents are organized rather than in how well their LLMs reason (Cemri et al., 2025; Qi et al., 2026).

Prior work has largely turned to *failure attribution*: given the execution trajectory of a run that has already failed, identify the agent and the step at which the failure was decided (Zhang et al., 2025b; 2026; Ma et al., 2026a; Zhu et al., 2026; Wang et al., 2026; Xia et al., 2026; Qin et al., 2024). These works share two properties: (1) they output a single agent and a single step, and (2) they are post hoc. Property (1) is not always accurate, since the communication strategy, the memory and the tools are units at which the workflow can equally be weak (Huang et al., 2025; Sun et al., 2025). Moreover, the same unit can be weak in several respects that call for different remedies, such as a tool that fails to respond and a tool that responds inconsistently, so an actionable diagnosis must identify the weak unit together with the *aspect* in which it is weak, which an (agent, step) pointer does not express and

[1] https://anonymous.4open.science/r/APEX-5FBC

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

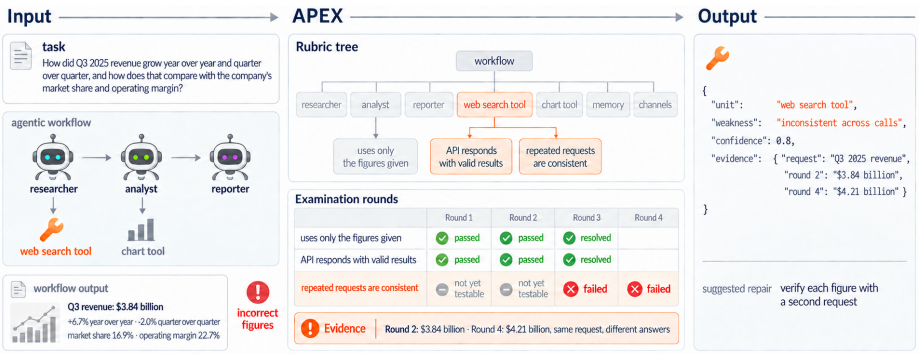

Figure 1: APEX examines an agentic workflow whose report contains wrong figures. The assertion that repeated search requests are consistent cannot be tested in a single run, and fails once the same request returns two different figures; APEX reports the search tool, the aspect of it that is weak, and the evidence behind that verdict.

repair methods built on it cannot exploit (Ma et al., 2026b). Property (2) limits what can be learned in the first place: a workflow induces an effectively unbounded space of execution trajectories, so attribution from an observed failure is confined to the weaknesses that have already surfaced.

As a running example, consider the market-analysis workflow of Figure 1, where a researcher collects data through a web search tool, an analyst computes and plots statistics with a chart tool, and a reporter writes the final report. The search tool does not return the same numbers for the same request, yet every call is answered plausibly, so each run appears healthy and the weakness is revealed only by comparing successive runs. Given the failed run, the failure attribution methods of Zhang et al. (2025b) blame the reporter or the analyst agent, whereas APEX identifies the search tool and the aspect in which it is weak, its inconsistency across repeated requests.

Therefore, we go one step further and study *unit-level weakness localization*: given a workflow, identify its weak units together with the aspects in which they are weak. Unlike failure attribution (Zhang et al., 2025b; 2026), no failed trajectory is given, so we discover weaknesses by adaptive progressive examination, which runs only the most informative executions of an effectively unbounded space. APEX, the framework we propose, builds a rubric tree of assertions about the units of the workflow, runs the workflow on probes that target the assertions it is statistically least confident about, and judges each assertion against what the units actually did, following rubric-based and LLM-as-a-judge evaluation (Starace et al., 2025; Zhuge et al., 2025; Zheng et al., 2023; Gu et al., 2026). Unlike the static rubrics written by hand for each task in prior evaluation (Starace et al., 2025), the rubric tree is generated from the units of the workflow and grows where needed, and every assertion, such as "the researcher agent collects every requested figure", belongs to a unit and carries a belief about whether it holds. Probes are ranked by rank-discounted entropy, as in rank-biased precision (Moffat & Zobel, 2008), and selected by maximum marginal relevance (Carbonell & Goldstein, 1998), adapting these information retrieval measures from ranking documents to selecting executions. In Figure 1, an inconsistency observed in round 3 lowers APEX's confidence in "repeated requests are consistent", and round 4 confirms that it does not hold. The output is a ranked set of units, each with the assertion it is held responsible for, the observations behind it, and a statistically derived confidence score, rather than a pointer to a single step, at the cost of more tokens than post-hoc methods that read a single failed trajectory.

This requires an execution environment, AXE, that runs the workflow under the examiner's control, records what each unit was given and changed, and can perturb a single unit in isolation. Since existing benchmarks consist of recorded trajectories that cannot be re-executed and label only the responsible agent and step (Zhang et al., 2025b), we also use AXE to generate AXEBench by generator-based construction (Chang et al., 2025) and fault injection (Huang et al., 2025), with the weak unit and aspect known by construction.

On AXEBench, APEX correctly localizes 90.5% of injected weaknesses in agents, against 57.1% for the state of the art, and 51.5% across all units, against 14.5%, as prior methods can only target agents. APEX outperforms the baselines even when given as many executions, and identifies the correct aspect in 79.0% of the cases it localizes. An LLM instructed to repair the workflow fixes 56.3% of the injected weaknesses when guided by APEX, against 12.5% without it. Our contributions are:

- We formulate *unit-level weakness localization* in agentic workflows: identifying their weak units and the aspects in which they are weak.
- We propose APEX, a framework that executes the workflow on the probes that most reduce its uncertainty about assertions organized in a dynamic rubric tree, localizing weak units and aspects with evidence.
- We build AXE, an execution engine that compiles an agentic workflow with a human in the loop, records what each unit was given and changed, and can perturb any unit at a controllable severity.
- We release AXEBench, 6,332 executable, perturbed workflow instances, built on tasks from two established benchmarks (Zhu et al., 2025; Chen et al., 2021) and labeled by construction with their weak unit and its weak aspect, for evaluating agentic workflows beyond their agents.

2 Related Work

Agentic workflows and their evaluation. Agentic workflows compose role-specialized LLM agents that coordinate through a communication protocol and act through tools (Hong et al., 2024; Qian et al., 2024; Wu et al., 2024; Li et al., 2023; Fourney et al., 2024), building on single-agent designs (Yao et al., 2023; Shinn et al., 2023), and their performance depends on the architecture as much as on the models (Huang et al., 2025; Fu et al., 2026; Orogat et al., 2027), which is why we treat communication, memory and tools as units that can be weak in their own right. Benchmarks measure whether such systems complete tasks (Zhu et al., 2025; Sun et al., 2025; Ma et al., 2024; Xu et al., 2025; Mialon et al., 2024), and PaperBench grades attempts against hierarchical rubrics written for each task (Starace et al., 2025). They report whether a system succeeded, not which part is responsible, whereas we generate the rubric around the workflow's units.

LLM-as-a-judge and examination-based evaluation. LLM judges commonly grade model outputs (Zheng et al., 2023; Gu et al., 2026), and Agent-as-a-Judge extends them to intermediate agent steps (Zhuge et al., 2025). Guinet et al. (2024) generate task-specific exams and keep the most informative via item response theory, and PQR refines queries to elicit failures (Lu et al., 2026). Both build test inputs actively, as APEX does, but assess the whole system, not its units and aspects.

Failure attribution and fault injection. Who&When identifies the agent and the step responsible for a failed run (Zhang et al., 2025b); later methods train attributors (Zhang et al., 2026; Zhu et al., 2026), localize errors to trajectory spans (Wang et al., 2026), analyze the interaction graph (Xia et al., 2026), or validate attributions by re-execution, observing that single-step attribution is often ill-posed (Ma et al., 2026a), and related studies diagnose failure modes (Cemri et al., 2025; Kim et al., 2026; Qi et al., 2026), borrowing the term localization from software engineering (Wong et al., 2016; Qin et al., 2024). Fault injection measures the resilience of topologies (Huang et al., 2025) or trains attributors (Zhang et al., 2026), and FlowFixer repairs workflows from a diagnosis (Ma et al., 2026b). All these works rely on a failed trajectory or a known fault, and none localizes which units of a workflow are weak, and in what respect, before a failure is observed; no existing benchmark supports evaluating such localization, which motivates APEX and AXEBench (Appendix L).

3 Problem Formulation

Agentic workflows. Following prior work that models agentic systems as computational graphs (Zhuge et al., 2024; Zhang et al., 2025a), we represent an agentic workflow as a tuple $W = (G, K, M, \sigma)$. The graph $G = (A, E)$ has the LLM-based agents A as its nodes and a channel $a \to a' \in E$ for each directed communication between two agents, which together define the communication strategy. K is the set of tools, each agent a acting through a permitted subset $K_a \subseteq K$, M is the shared memory, and σ is the schedule that determines which agent acts at each step. Aligning with the components agentic workflow frameworks declare explicitly (Wu et al., 2024; Li et al., 2023; Hong et al., 2024), we define the *units* of the workflow, to which a weakness can be attributed, as

$$\mathcal{U}(W) \;=\; A \cup E \cup \{M\} \cup K. \tag{1}$$

In our running example (Figure 1), $A = \{a_{\text{researcher}}, a_{\text{analyst}}, a_{\text{reporter}}\}$, $K = \{k_{\text{web_search}}, k_{\text{chart}}\}$, $E = \{a_{\text{researcher}} \to a_{\text{analyst}}, a_{\text{analyst}} \to a_{\text{reporter}}\}$, and M holds the retrieved numbers and charts.

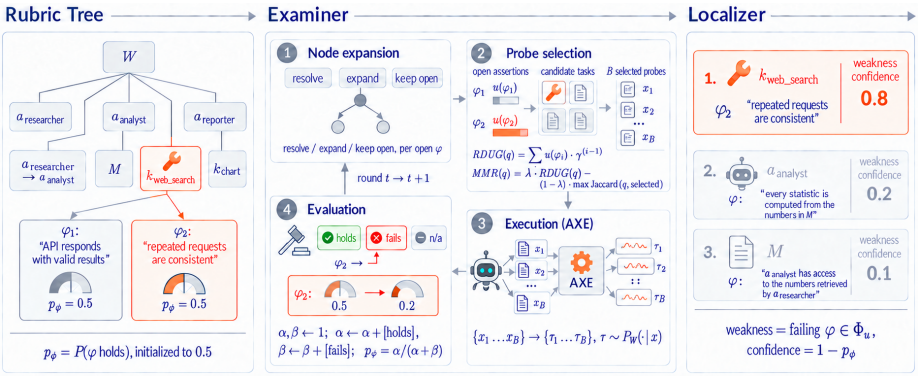

Figure 2: APEX in three stages. The *rubric tree* turns the workflow specification into per-unit assertions. The *examiner* then runs in rounds: it expands the open assertions, selects the B probes whose verdicts would most reduce the uncertainty about them, runs them through AXE, and updates each belief. The *localizer* returns the weak units, ranked, each with the assertion it fails.

Executing W on an input task x produces an *execution trajectory* $\tau = (s_1, \ldots, s_n)$, where each step $s_i = (u_i, c_i, o_i, \delta_i)$ records the unit u_i that acted, the context c_i it was given, its output o_i, and the change δ_i it made to the shared memory M. Since the agents are stochastic, W induces a distribution $\tau \sim P_W(\cdot \mid x)$ over an effectively unbounded space (full notation in Appendix A).

Aspects and assertions. An *aspect* of a unit $u \in \mathcal{U}(W)$ is a testable property of its behavior, and its corresponding *assertion* ϕ is a Boolean predicate that states it; on a trajectory τ, ϕ receives one of three verdicts: it holds, fails, or is not applicable (n/a). Each assertion belongs to a single unit, Φ_u denotes the assertions of u, and without loss of generality every assertion states a desirable property.

Unit weaknesses. A *weakness* of u is an assertion $\phi \in \Phi_u$ that fails with non-negligible probability,

$$\Pr_{x \sim \mathcal{D},\ \tau \sim P_W(\cdot \mid x)} \left[\phi(\tau) = \mathsf{fail} \;\middle|\; \phi(\tau) \neq \mathsf{n/a} \right] \;>\; \epsilon, \tag{2}$$

where $\mathcal{D}$ is the distribution of input tasks the workflow is meant to serve; u is then weak in the aspect that assertion ϕ states, as $k_{\mathsf{web_search}}$ is in "repeated requests are consistent".

Unit-level weakness localization. Given a workflow W, the task is to identify (1) the set of weak units $U^* \subseteq \mathcal{U}(W)$, and (2) for each $u \in U^*$, its weaknesses $\Phi_u^* \subseteq \Phi_u$, whereas failure attribution (Zhang et al., 2025b; 2026) returns an (agent, step) pointer into one failed trajectory.

Objective. With no failed trajectory to analyze, evidence must be gathered by executing W. We examine W in rounds, each executing it on B *probes* (input tasks) and observing the assertions' verdicts. With $h = (U^*, \{\Phi_u^*\})$ the true unit-level weaknesses of W and $\mathcal{E}_t$ the evidence gathered by round t, the goal is to select the next B probes expected to reduce the uncertainty about h most,

$$X_{t+1} \;=\; \arg \max_{X \subseteq \mathcal{X},\ |X| = B} \Big\{ \mathbb{H}\big(h \mid \mathcal{E}_t\big) \;-\; \mathbb{E}_{V_X} \Big[\mathbb{H}\big(h \mid \mathcal{E}_t, (X, V_X)\big) \Big] \Big\}, \tag{3}$$

where $\mathcal{X}$ is the task space, V_X the verdicts on X, and $\mathbb{H}$ denotes entropy; Eq. (3) thus maximizes the expected information gain about h, as in Bayesian active learning with noisy observations (Golovin et al., 2010) and budgeted probe selection (Cheng et al., 2010). As this gain needs unobserved verdicts, APEX (Section 4) actively probes where its uncertainty about the assertions is highest.

4 APEX

APEX takes the specification of a workflow W and returns its weak units, ranked, each with the assertions it fails, gathering evidence by selecting the tasks W executes (Figure 2).

4.1 Rubric Tree

The rubric is a rooted tree R over the nodes $V = \{W\} \cup \mathcal{U}(W) \cup \Phi$, with $\Phi = \bigcup_u \Phi_u$, that APEX builds from the specification to examine different aspects of the workflow (Figure 2, left), where

the workflow W is the root, with edges toward its units, every unit toward its own assertions, and every assertion toward the child assertions that refine it. Its first level comprises the units of Eq. (1), derived from the specification rather than proposed by an LLM, which makes attribution structural: each assertion and its verdict belong to one unit, which makes R a tree rather than a directed acyclic graph. An LLM writes each unit's Φ_u from the responsibilities the specification gives it, each assertion weighted by its importance w_ϕ, and each leaf of R holds a $\mathrm{Beta}(\alpha, \beta)$ posterior over the probability that its assertion holds, initialized at $\alpha = \beta = 1$, so $p_\phi = 0.5$.

4.2 Adaptive Progressive Examiner

The examiner runs in rounds of four steps (Figure 2, centre), until every assertion is resolved or the total probe budget is spent, each round building on what the earlier ones observed.

Node expansion (step 1). The examiner reads each open assertion's verdicts and evidence, and either (1) resolves it, when they agree conclusively, (2) expands it, when they disagree because it bundles distinct behaviors, into two or three narrower children that inherit its unit and cover its meaning, or (3) keeps it open. The tree thus deepens where evidence conflicts, as when "$a_{\mathsf{researcher}}$'s retrieved numbers reach a_{analyst}" is split into which numbers arrive and whether they arrive unchanged.

Probe selection (step 2). A *probe* is an input task the workflow is executed on. Eq. (3) favors the probes whose verdicts would most reduce the uncertainty about h; as those verdicts are known only once a probe has run, APEX ranks candidates by the uncertainty they stand to resolve, the current entropy of the assertions they target. The candidates are a fixed set of served tasks, or a pool an LLM designs each round from the workflow's own task specification. Each candidate q comes with the assertions $T_q = (\phi_1, \ldots, \phi_{|T_q|})$ it targets, ordered by how likely each is to receive a verdict rather than n/a, and we score q by its *relevance-discounted uncertainty gain* (RDUG), which discounts that order geometrically, as in rank-biased precision (Moffat & Zobel, 2008). The B probes of the round are then selected greedily under *maximum marginal relevance* (MMR) (Carbonell & Goldstein, 1998),

$$\mathsf{RDUG}(q) = \sum_{i=1}^{|T_q|} u(\phi_i)\, \gamma^{\,i-1}, \qquad \gamma \in (0, 1], \tag{4}$$

$$\mathsf{MMR}(q) = \lambda \cdot \mathsf{RDUG}(q) \;-\; (1-\lambda) \cdot \max_{q' \in S} \mathsf{Jaccard}(T_q, T_{q'}), \tag{5}$$

with S starting empty and extended with the highest-scoring candidate until it holds B probes, and the Jaccard term penalizing overlap in targeted assertions, to ensure probe diversity within a round.

Evaluation (steps 3 and 4). AXE (Section 5) executes and records the selected probes, and a judge returns a verdict on every assertion a probe targeted, with its evidence. An n/a verdict leaves the belief unchanged, while every holds or fails verdict v moves the belief of its assertion,

$$\alpha \leftarrow \alpha + \mathbf{1}[v = \mathsf{holds}], \quad \beta \leftarrow \beta + \mathbf{1}[v = \mathsf{fails}], \quad p_\phi = \frac{\alpha}{\alpha + \beta}, \quad u(\phi) = \mathbb{H}(p_\phi), \tag{6}$$

so p_ϕ tracks the proportion of agreeing verdicts, its uncertainty $u(\phi)$, captured by entropy, decreases as evidence accumulates, and the next round leans toward the assertions whose $u(\phi)$ remains high.

4.3 Localizer

The localizer turns the beliefs into APEX's ranked output (Figure 2, right). With Φ_u^+ the probed assertions of u and $s_\phi = \pm 1$ as an assertion states a desirable property or not, the failure score of u is

$$F(u) \;=\; \frac{1}{\sum_{\phi \in \Phi_u^+} w_\phi} \sum_{\phi \in \Phi_u} w_\phi \cdot \min\big(0,\; s_\phi\,(p_\phi - 0.5)\big) \;\le\; 0. \tag{7}$$

Keeping the negative terms alone stops a unit weak in one aspect from being excused by those it passes, and normalizing by what was probed leaves a unit no probe reached neither blamed nor cleared. From the scores, assertions and evidence, the examiner returns the units it holds responsible, ranked, each with its failing assertions and a confidence of $1 - p_\phi$. A correct decision needs both the scores, which locate where evidence turns negative, and the observations, which show why: a broken channel leaves the downstream agent failing its own assertions, and only the observations separate cause from consequence. *APEX-rule*, our ablation method, ranks units only by $F(u)$ (Section 6.2).

5 AXE AND AXEBENCH

AXE. AXE is the execution engine of APEX (Appendix B), standing between the examiner and the workflow, which lets APEX examine an arbitrary workflow through the single interface $\mathsf{Exec}(W, x) \to \tau$. It compiles a workflow, given in the form of Section 3 or described in natural language, resolving the structural choices a description leaves open, such as which agents may write, with the user in the loop. Each turn of an execution gives one unit its context and lets it act only through the tools it is permitted, and AXE records the turn's *footprint*: the context, the message, the tool calls, and the change to the shared memory as a diff. Tracing both what a unit states and what it changes lets a verdict land on the right unit: an agent whose message contains the analysis but whose turn left shared memory untouched has not delivered it. AXE can also perturb any named unit, applying the fault to its raw output before any other unit, tool or judge observes it (Huang et al., 2025), so the weakness is guaranteed rather than requested from a model that may decline to misbehave: agents by hollowing what they produce, tools by discarding their actions, channels by failing to deliver one unit's work to another, and memory by dropping or truncating what was written to it, at a severity set by how often a unit misbehaves and how much of its output it damages.

AXEBench. AXEBench contains 6,332 instances (Appendix E) built on tasks from two established benchmarks, the 100 coding tasks of MultiAgentBench (Zhu et al., 2025) and the 164 problems of HumanEval (Chen et al., 2021), each run through the three-agent development workflow of MultiAgentBench and through a four-agent chain that AXE compiles from a natural-language description, with twelve perturbations per task: 2,636 weak agents, 1,584 weak channels, 1,056 weak tools and 1,056 weak memories, plus instances at different levels of weakness severity (Section 6.3). To generate an instance, a workflow runs on a task, AXE perturbs one of its units, and the perturbed version runs on the same task; the instance holds both trajectories and is labeled with the perturbed unit and the aspect of it the fault damages. An instance is discarded unless the perturbation changed the recorded behavior, and a unit already weak in the healthy run is never chosen as a target. Because the fault defines each label and these filters ensure the label is correct, instances can be extended at scale without human annotation, and since each also stores its perturbed run, post-hoc methods receive exactly the evidence they were designed for, whereas existing failure-attribution datasets (Zhang et al., 2025b) cannot be re-executed. Five annotators experienced with agentic workflows validated a stratified sample of 40 instances with the labels hidden: every instance is correctly identified as perturbed, and in 77.5% of them the unit identified as weak matches the label (Appendix F).

6 EXPERIMENTS

6.1 EXPERIMENTAL SETUP

Workflow and instances. The main results are obtained on AXEBench instances, each perturbed in one unit: 165 of the three-agent workflow on MultiAgentBench tasks (42 agents and 41 each of channels, memory and tools) and 80 of the four-agent chain on HumanEval (20 of each), both balanced over unit kinds and sized to keep examination cost tractable (Section 6.5). Beyond coding, Appendix H evaluates 84 question-answering instances on HotpotQA (Yang et al., 2018).

Models and budget. Every examined workflow runs on `gpt-4o-mini` and APEX and all baselines on `gpt-5-mini`, so the comparison holds both models fixed and varies only the method; Appendix I studies other examiner models and topologies. An APEX examination is capped at five rounds of $B = 2$ probes, with $\gamma = 0.5$ and $\lambda = 0.8$; prompts and details are in Appendices C and D.

Baselines. We compare against the three failure-attribution methods of Zhang et al. (2025b), each returning a single agent and step from the recorded perturbed run, the evidence it was designed for: (1) *all-at-once*, which reads the trajectory in one call; (2) *step-by-step*, which walks it one step at a time; and (3) *binary search*, which repeatedly halves it. We also compare against (4) *random*, which names a unit at random, and (5) *APEX-rule*, whose localizer uses only the failure scores of Eq. (7). To further investigate APEX's advantage over the baselines, we add two that execute the workflow repeatedly: (6) *APEX-random*, which differs from APEX only in selecting its probes at random, and (7) *all-at-once×10*, which reads ten executions of the perturbed workflow instead of one.

Metrics. As APEX returns a ranked set of weak units and the baselines output a single one, we compare methods by top-1 accuracy, with Wilson 95% intervals: a method is correct when the unit

Table 1: Unit-level weakness localization, as the share of instances whose weak unit is named first, on (a) MultiAgentBench and (b) HumanEval instances, with Wilson 95% intervals. Prior methods cannot name any unit other than an agent, and those cells are shown with a dash.

	(a) MultiAgentBench, three-agent workflow					(b) HumanEval, four-agent chain				
	Overall	Agent	Chan.	Mem.	Tool	**Overall**	Agent	Chan.	Mem.	Tool
APEX	**51.5** [43.9, 59.0]	**90.5**	**53.7**	**51.2**	**9.8**	**56.2** [45.3, 66.6]	**95.0**	**85.0**	**40.0**	5.0
APEX-rule	24.2 [18.3, 31.3]	61.9	34.1	0.0	0.0	31.2 [22.2, 42.1]	75.0	50.0	0.0	0.0
APEX-random	34.5 [27.7, 42.1]	81.0	24.4	29.3	2.4	37.5 [27.7, 48.5]	85.0	40.0	25.0	0.0
All-at-once	14.5 [10.0, 20.7]	57.1	—	—	—	17.5 [10.7, 27.3]	70.0	—	—	—
All-at-once×10	18.2 [13.0, 24.8]	71.4	—	—	—	21.2 [13.7, 31.4]	85.0	—	—	—
Step-by-step	12.7 [8.5, 18.7]	50.0	—	—	—	16.2 [9.7, 25.8]	65.0	—	—	—
Binary search	11.5 [7.5, 17.3]	45.2	—	—	—	23.8 [15.8, 34.1]	**95.0**	—	—	—
Random	8.5 [5.1, 13.7]	4.8	4.9	17.1	7.3	13.8 [7.9, 23.0]	20.0	10.0	15.0	**10.0**

it names first is the perturbed one. On the instances a method localizes correctly, we further measure its *aspect accuracy*: each perturbation carries a plain statement of the aspect it damages (Section 5), and an LLM judge decides whether the assertion the method flagged describes that aspect. The verdicts APEX accumulates also come from an LLM judge; on a stratified sample of 50, it returns the majority verdict of five human experts on 82.0% (Cohen's $\kappa = 0.719$), and it agrees with them on every n/a, which decides whether a probe moves a belief (Appendix G).

6.2 Localization accuracy

Table 1 reports localization accuracy. APEX names the weak unit first in 51.5% of instances, against 14.5% for the strongest prior method. This improvement comes from two separate sources. (1) On agents, the only unit kind every method can target, APEX reaches 90.5% against 57.1%, close to the 53.5% these methods achieve on their original benchmark (Zhang et al., 2025b); progressive examination of a workflow is thus more effective than reading its recorded failure, even when both choose from the same candidate units. (2) On the other three unit kinds, the prior methods score zero, as they only target agents and cannot name a channel, memory or tool.

Repairing a weak unit also requires knowing its weak aspect (Section 6.6); among correct localizations, APEX identifies it in 79.0%. Human experts confirm the aspect judge on 86.0% of 50 cases (Appendix G.5). Deciding on the scores alone (APEX-rule) halves accuracy to 24.2% and drops memory and tools to zero, supporting Section 4.3: observations separate cause from consequence.

APEX-random and all-at-once×10 show that APEX's advantage over the baselines does not come from executing more. APEX-random reaches 34.5%, and the gap to APEX is widest on channels, 53.7% against 24.4%. All-at-once×10 reaches 71.4% on agents, up from 57.1%, still below APEX's 90.5%, and cannot name any other unit kind. The same holds on HumanEval, where APEX reaches 56.2% against 37.5% for APEX-random and 23.8% for the best prior method, with 75.0% aspect accuracy (Table 1b). Beyond coding, on HotpotQA, APEX reaches 51.2% against 13.1% for the best prior method and 71.4% against 52.4% on agents, and 86.0% aspect accuracy (Appendix H).

APEX performs weakest on tools, at 9.8% on MultiAgentBench, slightly higher than random, as a faulty tool makes its corresponding agent look responsible. Most of APEX's misidentifications come from three such confusions: broken tools identified as agents (68.3%), memory faults identified as agents (41.5%), and broken channels identified as memory (31.7%). Each pair leaves near-identical evidence in a single run, and since every assertion belongs to a single unit, no single verdict can tell them apart; assertions that relate two units are a natural extension of the rubric tree (Appendix K).

6.3 Perturbation severity analysis

We repeat the examination with the agent and memory faults injected at four strengths, from *faint*, where the unit misbehaves on a quarter of its turns and damages a tenth of its output, up to *severe*. Results are averaged over five runs for consistency. Figure 3 shows that APEX scales with the severity of the weakness: on agents it rises from 33.3% to 90.5% as the fault is triggered more often, while the prior methods do not improve, and step-by-step declines from 66.7% to 50.0%. Similarly,

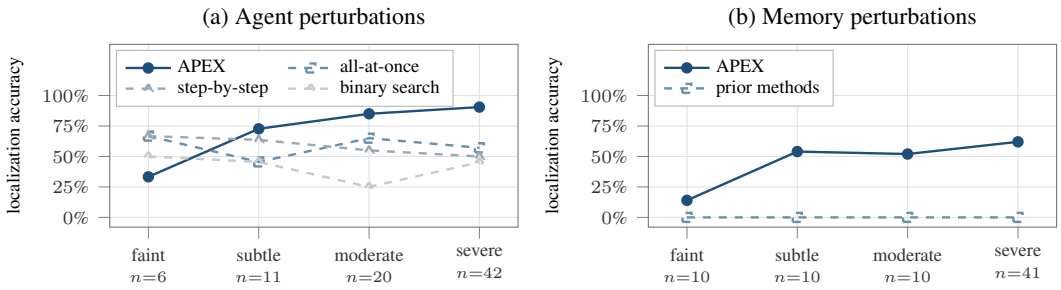

Figure 3: Accuracy under (a) agent and (b) memory perturbations of increasing strength, averaged over five runs. A fainter fault often leaves the run unchanged, so fewer instances pass the filter.

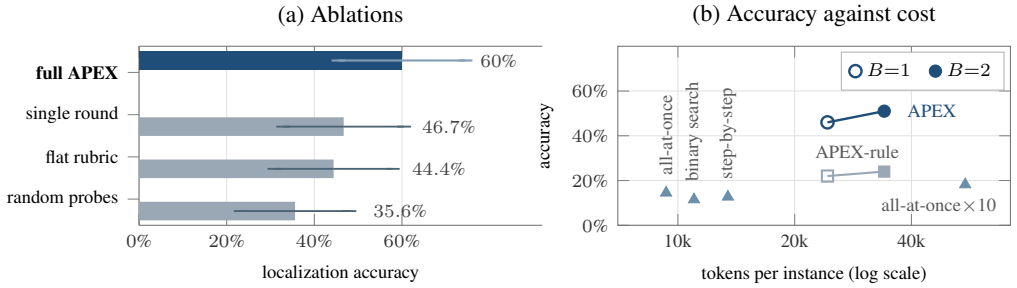

Figure 4: (a) Removing one component of the examiner at a time, with Wilson 95% intervals. (b) Accuracy against tokens per instance, on a log scale, as the probe budget B grows.

memory weaknesses only become visible once the fault is triggered often enough: APEX rises from 14% at the faintest setting to 62% at the severe one, while the prior methods stay at zero (Figure 3b).

At the faintest setting APEX falls below the baselines on agents: an instance enters AXEBench only if the perturbation changed the recorded run, so a post-hoc method always receives a trajectory in which the fault occurred, whereas APEX must trigger a rare fault itself. The label-correctness filter thus favors post-hoc methods where a weakness rarely occurs, an asymmetry absent at full severity.

6.4 Ablation Study

We remove one component of the APEX examiner at a time on a balanced 45-instance subset of the main set, keeping the probe budget and the localizer fixed. *Single round* spends the whole probe budget at once, so no round sees what an earlier one observed; *flat rubric* never expands an assertion; and *random probes* selects from the candidate pool uniformly instead of by Eq. (5).

Removing any single component costs accuracy (Figure 4a), which drops from 60.0% to 46.7%, 44.4% and 35.6%, the largest drop coming from random probes; since the intervals overlap, the order among the three is not conclusive, but each component improves the accuracy of the APEX examiner. Additionally, on the full main set, the accuracy of APEX-random drops significantly compared to APEX (Section 6.2), showing the effectiveness of the probe selection component. The key factor is selecting probes where the uncertainty about the assertions is highest, which is how APEX works toward Eq. (3) without knowing the verdicts in advance.

6.5 Examination Cost

Figure 4(b) places accuracy against cost in tokens per instance rather than currency, to avoid dependence on one model's price. The post-hoc methods spend 9.3–13.5k tokens and reach 11.5–14.5%, while APEX spends 24.3k at $B = 1$ and 34.0k at $B = 2$, reaching 46% and 51%. (1) *APEX vs. the post-hoc baselines*: APEX spends two to four times as many tokens for about four times the accuracy. However, spending more tokens on reading does not close that gap by itself: all-at-once×10 reads ten trajectories, spends more tokens than APEX, but reaches only 18.2%. (2) *Within APEX*: increasing B raises cost and accuracy, a tunable trade-off, while APEX-rule stays below 25%.

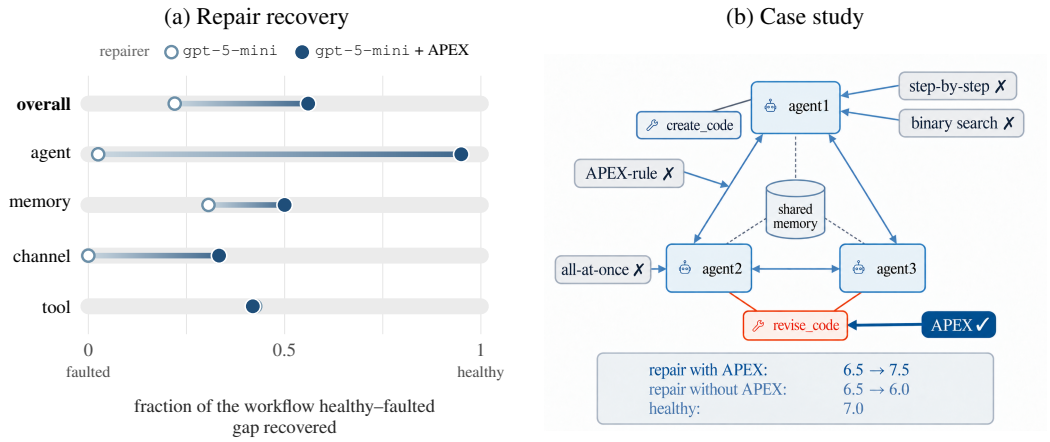

Figure 5: (a) The ratio of the healthy–faulted performance gap the `gpt-5-mini` repairer recovers, with and without APEX's output, where 0 is the perturbed workflow and 1 the original one. (b) One instance with a broken `revise_code` tool: the unit each method names and the score after repair.

6.6 Downstream application: workflow repair

The main goal of weakness localization is to repair the workflow. We execute every instance under four conditions, its solution scored 0 to 10 by the same LLM grader (`gpt-4o-mini`): the original workflow (*healthy*), the perturbed one (*faulted*), and the perturbed one repaired by `gpt-5-mini` from APEX's diagnosis (*guided*) or without it (*unguided*). Both repairs may change one unit and are only instructed that the workflow underperforms, without naming the unit. We report the *recovery rate*, the share of the healthy–faulted gap a repair closes, where the fault lowered the score.

In Figure 5(a), APEX's output makes the LLM repairer more effective: an APEX-guided repair closes 0.56 of the gap against 0.22 unguided, removes the weakness on 56.3% against 12.5%, and restores healthy performance on 55.6% against 27.8%. The margin follows localization accuracy: widest on agents, where a guided repair recovers 0.95 of the gap and an unguided one almost none, it narrows on memory and channels and closes on tools.

Case study. Figure 5(b) shows a broken `revise_code` tool, the unit kind where APEX is weakest. The prior methods each blame an agent and APEX-rule the `agent1–agent2` channel. APEX never probed the tool's assertions, yet reaches it through the other units: `agent2` passes its own assertions while its turn leaves the solution unchanged, the commit `agent1` makes through `create_code` is visible, and memory passes its assertions, so only a broken revise tool explains all three. Guided by APEX, the LLM repairer restores `revise_code`, raising the score from 6.5 to 7.5, while the unguided one rewrites `agent1`'s prompt, lowering it to 6.0 (Appendix J).

7 Conclusion

We formulated unit-level weakness localization in agentic workflows, which identifies not only which unit of a workflow is weak but also in which aspect, before it surfaces as a failure in deployment. APEX addresses this task by progressively examining the workflow rather than reading a failed trajectory. On AXEBench, it localizes weaknesses across all four unit kinds where prior methods only target agents, outperforms them on agents, 90.5% against 57.1%, improves with weakness severity, and helps an LLM repairing a workflow recover over twice as much lost performance.

Limitations and future work. (1) Our evaluation centers on coding, along with question answering (Appendix H); nothing in APEX is specific to coding, and as AXE compiles workflows from natural language, other domains follow naturally. (2) The weaknesses are injected, which makes every label exact, and perturbing several units at once would test compound faults. (3) APEX needs an executable workflow, which also enables pre-deployment examination, and costs more than post-hoc methods, though its probe budget trades cost for accuracy (Section 6.5). (4) APEX confuses units leaving the same evidence (Section 6.2), which cross-unit assertions could separate.

AI USE STATEMENT

LLM assistance was used solely for polishing the manuscript and language enhancement. All intellectual contributions, scientific claims, and conclusions in this paper are entirely the authors' own, and the authors take full responsibility for the final content.

ETHICS STATEMENT

This work studies weakness localization in LLM-based agentic workflows using automatically generated labels. All workflows and tasks come from public academic benchmarks. Five annotators with experience in agentic workflows labelled benchmark instances and judge verdicts for validation; they labelled only system outputs, and no personal data was collected from them or used anywhere in this work. The perturbation machinery is intended for examining one's own workflows before deployment; we see no ethical risk in its release, and the generated resources are for research use only.

REPRODUCIBILITY STATEMENT

We release our source code, AXEBench, and the prompts used at every stage through an anonymous link.[2] Model versions and configurations are given in Section 6.1; the generation procedure for AXEBench, including the filters applied to every instance, is described in Section 5.

REFERENCES

Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 335–336, 1998.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025.

Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, Siddharth Patki, Ishita Prasad, Xavier Puig, Akshara Rai, Ram Ramrakhya, Daniel Tran, Joanne Truong, John M. Turner, Eric Undersander, and Tsung-Yen Yang. PARTNR: A benchmark for planning and reasoning in embodied multi-agent tasks. In *International Conference on Learning Representations (ICLR)*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Reynold Cheng, Eric Lo, Xuan S. Yang, Ming-Hay Luk, Xiang Li, and Xike Xie. Explore or exploit? Effective strategies for disambiguating large databases. *Proceedings of the VLDB Endowment*, 3 (1–2):815–825, 2010.

Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-One: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

Yuhang Fu, Ruishan Fang, Jiaqi Shao, Huiyu Zheng, Zhengtao Zhu, Bing Luo, and Tao Lin. Do more agents help? controlled and protocol-aligned evaluation of LLM agent workflows. *arXiv preprint arXiv:2606.05670*, 2026.

[2] `https://anonymous.4open.science/r/APEX-5FBC`

Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal Bayesian active learning with noisy observations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2010.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Zhouchi Lin, Bowen Zhang, Lionel Ni, Wen Gao, Yuanzhuo Wang, and Jian Guo. A survey on LLM-as-a-judge. *The Innovation*, 7 (6):101253, 2026. doi: 10.1016/j.xinn.2025.101253.

Gauthier Guinet, Behrooz Omidvar-Tehrani, Anoop Deoras, and Laurent Callot. Automated evaluation of retrieval-augmented language models with task-specific exam generation. In *International Conference on Machine Learning (ICML)*, 2024.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

Jen-tse Huang, Jiaxu Zhou, Tailin Jin, Xuhui Zhou, Zixi Chen, Wenxuan Wang, Youliang Yuan, Michael R. Lyu, and Maarten Sap. On the resilience of LLM-based multi-agent collaboration with faulty agents. In *International Conference on Machine Learning (ICML)*, 2025.

Myeongsoo Kim, Dingmin Wang, Siwei Cui, Farima Farmahinifarahani, Terry Yue Zhuo, Shweta Garg, Baishakhi Ray, Rajdeep Mukherjee, and Varun Kumar. Coherence collapse: Diagnosing why code agents fail after reaching the right code. *arXiv preprint arXiv:2603.24631*, 2026.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yunan Lu, Luigi Liu, Omar Yahia, Arpit Sharma, and Zhou Yu. PQR: A framework to generate diverse and realistic user queries that elicit QA agent failures. *arXiv preprint arXiv:2605.16551*, 2026.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. DoVer: Intervention-driven auto debugging for LLM multi-agent systems. In *International Conference on Learning Representations (ICLR)*, 2026a.

Xuyan Ma, Yawen Wang, Junjie Wang, Xiaofei Xie, Boyu Wu, Mingyang Li, Dandan Wang, and Qing Wang. Diagnosis-driven automatic repair for agentic workflow via symbolic inference. *arXiv preprint arXiv:2607.02882*, 2026b.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations (ICLR)*, 2024.

Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems*, 27(1):1–27, 2008.

Abdelghny Orogat, Ana Rostam, and Essam Mansour. Architectural design, not only model intelligence, governs multi-agent LLM performance. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2027. To appear.

Shihao Qi, Jie Ma, Rui Xing, Wei Guo, Xiao Huang, Zhitao Gao, Jianhao Deng, Jun Liu, Lingling Zhang, Bifan Wei, Boqian Yang, Pinghui Wang, Jianwen Sun, Jing Tao, Yaqiang Wu, Hui Liu, Yu Yao, and Tongliang Liu. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in LLM-based multi-agent systems. *arXiv preprint arXiv:2605.14892*, 2026.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Yihao Qin, Shangwen Wang, Yiling Lou, Jinhao Dong, Kaixin Wang, Xiaoling Li, and Xiaoguang Mao. AgentFL: Scaling LLM-based fault localization to project-level context. *arXiv preprint arXiv:2403.16362*, 2024.

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

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *International Conference on Machine Learning (ICML)*, 2025.

Haochen Sun, Shuwen Zhang, Lujie Niu, Lei Ren, Hao Xu, Hao Fu, Fangkun Zhao, Caixia Yuan, and Xiaojie Wang. Collab-Overcooked: Benchmarking and evaluating large language models as collaborative agents. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

Jiaming Wang, Ziteng Feng, Jiangtao Wu, Ruihao Li, Qianqian Xie, Yuxiang Ren, He Zhu, Xueming Han, Fanyu Meng, Junlan Feng, and Jiaheng Liu. Where do deep-research agents go wrong? span-level error localization in agent trajectories. *arXiv preprint arXiv:2606.02060*, 2026.

W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*, 2024.

Yufei Xia, Anjun Gao, Yueyang Quan, Zhuqing Liu, and Minghong Fang. Who broke the system? failure localization in LLM-based multi-agent systems. In *Conference on Language Modeling (COLM)*, 2026.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

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

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? In *International Conference on Learning Representations (ICLR)*, 2026.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations (ICLR)*, 2025a.

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 *International Conference on Machine Learning (ICML)*, 2025b.

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

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 8580–8622, 2025.

Taiyu Zhu, Yifan Wu, Weilin Jin, Ying Li, and Gang Huang. StepFinder: A temporal semantic framework for failure attribution in multi-agent systems. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2026.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *International Conference on Machine Learning (ICML)*, pp. 62743–62767, 2024.

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. In *International Conference on Machine Learning (ICML)*, pp. 80569–80611, 2025.

A NOTATION

Table 2 collects the notation of Sections 3 and 4.

Table 2: Notation.

Symbol	Meaning
$W = (G, K, M, \sigma)$	agentic workflow: graph, tools, shared memory, execution schedule
$G = (A, E)$	agents A and the directed channels E between them
K, $K_a \subseteq K$	tools, and the tools agent a is permitted to use
$\mathcal{U}(W) = A \cup E \cup \{M\} \cup K$	units of the workflow (Eq. 1)
x, $\mathcal{D}$, $\mathcal{X}$	input task, the distribution of tasks served, the task space
$\tau = (s_1, \ldots, s_n)$	execution trajectory, with $s_i = (u_i, c_i, o_i, \delta_i)$
$P_W(\cdot \mid x)$	distribution of trajectories of W on task x
ϕ, Φ_u, Φ	assertion, the assertions of unit u, all assertions
holds / fails / n/a	the three verdicts an assertion receives on a trajectory
ϵ	threshold on the failure probability that defines a weakness (Eq. 2)
U^*, Φ_u^*	weak units, and the weaknesses of unit u
$h = (U^*, \{\Phi_u^*\})$	the true unit-level weaknesses of W
$\mathcal{E}_t$	evidence gathered by round t
B, X_{t+1}, V_X	probes per round, the probes of round $t{+}1$, their verdicts
$\mathbb{H}$	entropy
R, V	rubric tree and its nodes, $V = \{W\} \cup \mathcal{U}(W) \cup \Phi$
w_ϕ	relative weight of an assertion within its unit
$\mathrm{Beta}(\alpha, \beta)$, p_ϕ	belief over whether ϕ holds, and its mean $\alpha/(\alpha+\beta)$
$u(\phi) = \mathbb{H}(p_\phi)$	uncertainty of an assertion
q, $T_q = (\phi_1, \ldots, \phi_{\lvert T_q \rvert})$	candidate probe and the assertions it targets, by relevance
γ, λ	rank discount of RDUG (Eq. 4) and trade-off of MMR (Eq. 5)
S	probes selected so far within a round
Φ_u^+, s_ϕ	probed assertions of u, and the polarity of ϕ (± 1)
$F(u)$	failure score of unit u (Eq. 7)

B AXE: Workflow Compilation and Perturbation

Figure 6 summarizes AXE. A workflow is either given in the form of Section 3 or described in natural language, in which case the compiler maps the description to that form. The compiled workflow is validated before it runs: every unit has a distinct identity, the graph is connected with an entry and a terminal unit, the units together cover the stated goal, and every tool permission refers to a tool that exists. When the description leaves a structural choice open, such as how many agents there are, which of them may write to the shared memory, or when execution stops, the compiler asks the user rather than guessing, so that the compiled units are the ones the user intends. The four-agent chain of Table 1(b) and the question-answering workflow of Appendix H were both compiled this way from a short description.

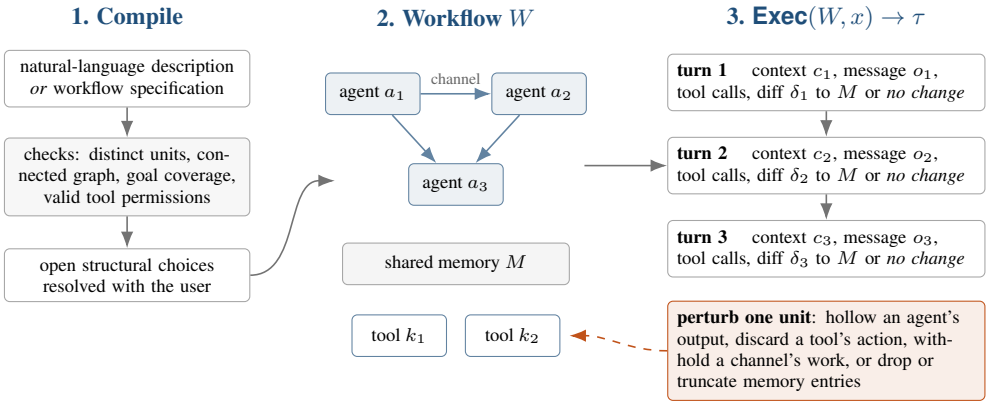

Figure 6: AXE compiles a workflow, executes it on a task while recording a footprint for every turn, and can perturb any single unit.

Execution and footprints. Each turn gives one unit its context, which contains the task and the entries of the shared memory it is allowed to read, takes a single model call, and lets the unit act only through its permitted tools. The footprint of the turn records the context, the message, the tool calls with their results, and the change the turn made to the shared memory as a diff, or an explicit note that nothing changed. Diffs are capped at 3,000 characters in the trajectory the judge reads. Because every change to the shared memory is attributed to the turn that made it, a verdict about what a unit delivered is decided by its diff rather than by what its message claims.

Perturbations. A perturbation is applied to a unit's raw output before any other unit, tool or judge observes it, following the output-injection protocol of Huang et al. (2025), so the weakness is present by design rather than requested from a model. Table 3 lists the four kinds. Agent and memory perturbations have a severity, set by two parameters: the share of a unit's turns that are affected, and how much of an affected output is damaged. Table 4 gives the four levels used in Section 6.3; the main results use the severe level. Tool and channel perturbations have no severity: a tool either applies its action or does not, and a channel either delivers or withholds.

Table 3: The four kinds of perturbation AXE applies.

Unit	Perturbation
Agent	hollows the agent's output: a share of the functions it writes keep their signature but lose their body (coding), or a share of what it states is replaced by placeholders (question answering)
Tool	the tool is revoked from every agent that holds it, so the actions submitted to it are discarded
Channel	the downstream agent no longer receives the upstream agent's entries in its context
Memory	a share of the entries written to the shared memory is dropped, or truncated to a quarter of their length, before later units read them

Table 4: Severity levels for agent and memory perturbations. For agents, the first column is the share of turns affected and the second the share of the output damaged in an affected turn; for memory, the second column is the share of entries dropped or truncated.

Level	turns affected	share damaged
faint	25%	10%
subtle	50%	20%
moderate	100%	40%
severe	100%	60%

For each task, the three-agent workflow of MultiAgentBench receives twelve perturbations: five on its agents, one on each of its two tools, two on its memory (drop and truncate), and one on each of its three channels. The other workflows receive the same twelve, mapped onto their own units.

C PROMPTS

APEX uses six prompts, one per step of the examination, and the evaluation uses two more. Each prompt consists of a system message, reproduced below, and a user message that supplies the workflow specification, the current assertions with their beliefs, verdicts and evidence, and the required JSON output format. All prompts are released with the code.

Rubric construction (Section 4). Writes the assertions Φ_u of every unit, each with a weight w_ϕ and a polarity.

```
You are an expert evaluator of LLM-based agentic workflows specializing in diagnostic rubric
     construction.

You are given a workflow W = (G, T, C) and the fixed list of its evaluatable units -- the
agents, communication channels, memory, and tools that a weakness could be localized to.

For each unit you write a SMALL, PRECISE set of assertions about that unit's behaviour.
Be conservative: every assertion is a claim that will cost real executions to investigate,
so only write assertions about behaviours that genuinely warrant investigation -- the unit's
actual responsibilities, observable directly in an execution trace of THIS workflow.
```

Node expansion (step 1). Resolves, expands or keeps each open assertion.

```
You are an expert evaluator steering the examination of an LLM-based agentic workflow.

For each open assertion you see the verdicts it has received so far and the judge's evidence
     , and you
decide what the examination should do with it next: RESOLVE it (the evidence is conclusive
     -- further
probing would not change the answer), EXPAND it into sharper child assertions (it bundles
     distinct
behaviours, which is why the verdicts disagree), or KEEP PROBING it (it is a single
     behaviour whose
evidence is still thin).

You are conservative on both sides: you do not resolve an assertion on weak or contradictory
evidence, and you do not split an assertion that is already one precise behaviour.
```

Probe selection (step 2). Designs the candidate probes, each with the assertions T_q it targets in order of relevance; RDUG and MMR then select B of them.

```
You are an expert exam designer for evaluating LLM-based agentic workflows.

You design targeted task instances that, when executed by the workflow under examination,
produce execution traces that reveal evidence for evaluation assertions.

Crucially, every task you generate must be a CONCRETE INSTANCE of the workflow's own task
specification (T) -- the same kind and domain of task the workflow is built to handle. You
probe the workflow within its intended task family; you never invent tasks outside it.

Each task is built around a primary assertion but naturally surfaces evidence for others too
     .
You must list assertions in DESCENDING RELEVANCE ORDER -- the first assertion is the one
this task most directly probes, the last is the weakest connection.
```

Assertion judge (steps 3 and 4). Reads a trajectory and returns holds, fails or n/a for every targeted assertion, with the evidence for it. Its system message instructs it to ground every verdict in the trajectory; to judge what a unit did, as recorded in its diff, rather than what its message claims; to judge each unit by its own turn rather than by the final solution; to separate the quality of a unit's work from whether that work reached its destination, which belongs to the tool or channel involved; and to return n/a when the trajectory contains no evidence either way.

Localizer (Section 4.3). Reads the failure scores, the assertions and the judge's evidence, and returns the weak units, ranked.

```
You are diagnosing an agentic workflow. You are given every evaluatable unit of the
workflow -- its agents, the channels along which their work is handed on, the shared memory,
      and
the tools through which they act -- together with the assertions tested about each unit, how
      often
each held, and the evidence the judge recorded.

Identify which unit or units are weak. Reason about what kind of unit each piece of evidence
implicates:
- an AGENT is weak when the work it produces is itself poor, missing or hollow;
- a TOOL is weak when content submitted to it does not reach the shared solution intact;
- a CHANNEL is weak when one unit's work never reaches the unit that needed it, so that unit
  proceeds in ignorance of it;
- MEMORY is weak when what was stored is not what is later available.

A unit that merely suffered the consequences of another unit's weakness is not itself weak:
     prefer
the unit that explains the others. Read the observations, not only the counts: when several
     units
fail, ask which failure came first and would account for the rest (a channel that withheld
     work
explains a downstream agent redoing it; a tool whose calls never took effect explains an
     agent whose
work never landed; memory that dropped entries explains agents that lacked earlier work). In
particular:
- if an agent's own output is complete but its turn left the shared solution unchanged, or
     its work
  never reached the solution, the weak unit is the TOOL it acts through, not the agent;
- if a unit lacks the contributions of SEVERAL earlier units, or entries arrive cut short,
     the weak
  unit is the shared MEMORY; if exactly one pair of units fails to see each other's work
       while the
  other units do see it, the weak unit is the CHANNEL between that pair;
- an agent is weak when the work in its own output is itself hollow, wrong or missing.
Always name at least one unit: the one the evidence implicates most. Rank the units you
     implicate, most likely first, and include a
unit only if the evidence supports it.
```

Aspect judge (Section 6.1). Decides whether a flagged assertion describes the aspect a perturbation damages (validated in Appendix G.5).

```
You are checking whether a diagnosis identified the right KIND of weakness.

You are given the weakness that was actually introduced into one unit of an agentic workflow
     , and
the assertion a diagnostic method flagged as failing for that unit. Decide whether the
     assertion
describes the same weakness.

Judge the substance, not the wording. The assertion need not use the same words; it needs to
      be
about the same defect. In particular:
- describing work as hollow, placeholder or unimplemented matches a weakness about hollow
     output;
- describing work as not reaching or not being applied to the shared solution matches a
     weakness
  about work that never lands, and is NOT the same as the work itself being poor;
- describing a unit as unaware of, ignoring or duplicating another unit's work matches a
     weakness
  about work not travelling between them;
- describing context as lost, truncated or not preserved matches a weakness about memory.

Two assertions can both be true of a run and still describe different weaknesses. Only a
     match if
the assertion picks out the SAME defect.
```

Repairer (Section 6.6). Proposes a change to a single unit, from the diagnosis in the guided condition and from the workflow alone in the unguided one.

```
You repair LLM-based agentic workflows.

You are given a workflow -- its agents with their instructions and tool permissions, the
     channels
between them, the shared memory, and the tools they act through -- and a diagnosis naming
     the units
believed to be weak, with the evidence behind that belief.

Propose the smallest change that would fix the named units. Rewrite an agent's instructions
     only to
address the specific weakness described; do not rewrite the workflow's purpose, do not add
     new
agents, and do not touch units the diagnosis does not name. A good repair is one another
     engineer
would recognise as addressing exactly the problem reported.

Choose the action by the kind of unit you repair:
- an AGENT (e.g. agent2): "rewrite_prompt", giving its full replacement instructions;
- a CHANNEL between two agents (an id such as "agent1<->agent2"): "restore_channel", with
     that id
  as the unit, so work flows between the two agents again;
- the shared MEMORY ("memory"): "reset_memory";
- a TOOL (e.g. a tool id such as create_code): "grant_tool", naming the agents whose role
     needs it.
```

D Implementation and Experimental Details

Models. The examined workflows run on `gpt-4o-mini`. APEX, the assertion judge, the localizer, all baselines, the aspect judge and the repairer run on `gpt-5-mini`; the grader of Section 6.6 runs on `gpt-4o-mini`. Appendix I replaces the examiner with `gpt-4o-mini`.

Hyperparameters. Each round designs a pool of four candidate probes and selects $B = 2$ of them, with $\gamma = 0.5$ in RDUG and $\lambda = 0.8$ in MMR. An examination runs at most five rounds, so at most ten probes, and ends earlier when every assertion is resolved. An assertion is expanded at most three times along a path of the rubric tree. Every belief starts at $\alpha = \beta = 1$. The cost study of Section 6.5 varies B and keeps everything else fixed.

Baselines. The three methods of Zhang et al. (2025b) are re-implemented on the same model as APEX and receive the recorded trajectory of the perturbed run, formatted as the footprints of Appendix B. All-at-once$\times 10$ receives ten such trajectories of the same instance in one prompt. APEX-random uses APEX's rubric, judge, localizer and budget, and draws its probes uniformly from the candidate pool.

Harness. Instances are examined in parallel, each in its own process state. Every model call is retried up to four times on a transient error, and results are checkpointed per instance, so an interrupted run resumes without repeating finished instances. Token usage is recorded per call.

Execution. Experiments ran on a workstation with an Intel Core i9-11900K (8 cores, 16 threads) and 128 GB of memory, running Windows Server.

E AXEBench Composition

Table 5 breaks AXEBench down by workflow and task source. The two coding task sources are crossed with the two coding workflows, and each task receives the twelve perturbations of Appendix B. An instance is kept only if its perturbation changed the recorded run, which removes a few candidates. The severity instances of Section 6.3 and the question-answering instances of Appendix H are listed separately.

F AXEBench Validation Protocol

We sampled 40 instances from the main benchmark, 10 per unit kind, and showed each with its task, the workflow's units and the recorded perturbed trajectory, withholding the perturbed unit, the

Table 5: Composition of AXEBench.

Workflow	Tasks	Agents	Channels	Tools	Memory	Total
three-agent (MultiAgentBench)	MultiAgentBench, 100	500	300	200	200	1,200
three-agent (MultiAgentBench)	HumanEval, 164	820	492	328	328	1,968
four-agent chain	HumanEval, 164	816	492	328	328	1,964
four-agent chain	MultiAgentBench, 100	500	300	200	200	1,200
Total		2,636	1,584	1,056	1,056	6,332
Severity levels faint to moderate		37			30	67
Question answering (HotpotQA, 12 questions)		58	36	24	24	142

aspect, the perturbation and the note recording that it changed the run. The same five annotators as in Appendix G answered two questions per instance, whether the instance is perturbed at all and which unit is weak, with the labels hidden from them: they labelled independently, then observed one another's labels, discussed and voted again. Their final answers are aggregated by majority vote, with no ties. Table 6 reports the result. Of the five memory instances the annotators did not confirm, three truncate entries and two drop them.

Table 6: Human validation of AXEBench labels on 40 instances, by majority vote over five annotators after discussion.

	value
Instance correctly identified as perturbed	100%
Unit identified as weak matches the label	77.5% (70.0% before discussion)
agents	100%
channels	90%
tools	70%
memory	50%
Fleiss' κ on the weak unit, before / after discussion	0.685 / 0.932

G Assertion Judge Validation Protocol

This appendix gives the protocol behind Section 6.1 and the full agreement results.

G.1 Sampling

We re-ran APEX on four AXEBench instances, one per unit kind, with every judgment recorded: the assertion, the probe the verdict came from, the trajectory the judge read, and the verdict and evidence it returned. That gave 238 judgments, 112 holds, 68 fails and 58 n/a. We sampled 50 of them, stratified over unit kind and over the verdict the judge returned, so that the rarer verdicts are represented well enough to be scored. The stratification makes the sample unrepresentative of how often each verdict occurs, which is why we also report agreement re-weighted to the $112/68/58$ mix (Section G.3).

The sheet is blind. It shows the assertion, the unit it belongs to, the probe task, the expected behavior and the recorded trajectory, and it withholds the judge's verdict and evidence until a label has been entered. The judge reads a trajectory whose recorded diffs are capped at AXE's limit of $3{,}000$ characters; the sheet shows $8{,}000$ characters per step, so an annotator never labels from less evidence than the judge had.

G.2 Annotators and instructions

Five annotators with experience building and debugging agentic workflows labelled the sheet independently. They then observed one another's labels and reasoning, without the judge's verdict, discussed the items and voted again. All five labelled all 50 items and none was skipped. The instruction given was to decide, from the trajectory alone, whether the assertion holds (the trajectory shows it is true in this run), fails (the trajectory shows it is false), or is n/a (the run never gave

that unit an occasion to display the behavior, so the trajectory cannot settle it). The final labels are aggregated by majority vote, which was unambiguous on every item: no item produced a tie, and no item's majority label changed between the two rounds.

G.3 Results

Table 7 reports agreement. Among the annotators, Fleiss' κ is 0.815 before discussion and 0.877 after, both in the range usually read as almost perfect, which is what licenses the majority vote as a reference. Against the final majority vote the judge scores 82.0% with Cohen's $\kappa = 0.719$, and 81.6% once re-weighted to the natural verdict mix. Figure 7 breaks the result down by verdict, which is the more informative view: the judge is right on every n/a, its errors are concentrated in holds, where it accepts six assertions the annotators reject, and no disagreement anywhere involves n/a.

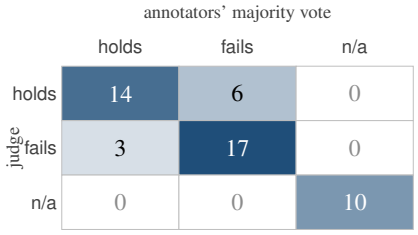

Figure 7: The judge's verdict against the annotators' final majority vote, over the 50 sampled judgments. The n/a row and column are empty off the diagonal.

Table 7: Judge agreement with the annotators' final majority vote, over the 50 sampled judgments.

	agreement	
Overall	$41/50 = 82.0\%$	Cohen's $\kappa = 0.719$
Re-weighted to the $112/68/58$ mix	81.6%	
Among annotators, before / after discussion		Fleiss' $\kappa = 0.815$ / 0.877
Agent assertions	$9/10 = 90.0\%$	
Memory assertions	$13/16 = 81.2\%$	
Channel assertions	$8/10 = 80.0\%$	
Tool assertions	$11/14 = 78.6\%$	
Judge returned holds	$14/20 = 70.0\%$	
Judge returned fails	$17/20 = 85.0\%$	
Judge returned n/a	$10/10 = 100.0\%$	

G.4 The nine disagreements

Six of the nine are assertions the judge accepts and the annotators reject, and all six ask for a judgment of degree rather than of fact. Four are of the form *when one of two agents acts after the other, its work builds on that contribution rather than ignoring, duplicating or redoing it*: an agent had the earlier contribution in context and rewrote most of it, which the judge read as building on it and the annotators read as redoing it. The other two ask whether a contribution is *a complete, working implementation with no placeholder bodies*, where the threshold between an incomplete implementation and a stub is not sharp.

The remaining three run the other way. Two are the same conjunctive assertion, that an edit *replaces a previously missing or stubbed requirement visible in shared memory*, which the judge failed because nothing had yet landed in the shared solution while the annotators read the edit itself as satisfying the assertion. The last requires a top-level module docstring, which the judge failed because a comment preceded it.

None of the nine is a disagreement about n/a. The separation between an assertion a probe could not settle and one it settled negatively, which is the distinction that decides whether a verdict moves a belief in Eq. (6), is the part of the judge's behavior the annotators confirm exactly.

G.5 THE ASPECT JUDGE

Aspect accuracy (Section 6.1) is decided by a second LLM judge, which compares the assertion a method flagged, together with what the diagnosis found about it, against the plain statement of the aspect the perturbation damages. We validated it on 50 of the 81 cases it scored on the main set, sampled in proportion to its decision and to unit kind: 40 matches and 10 non-matches, over 23 agent, 14 channel and 13 memory cases (no correctly localized tool instance has a flagged assertion). Five annotators received exactly the judge's input and instructions, with its decision hidden, and labelled every item independently in a single round. They agree at Fleiss' $\kappa = 0.870$, unanimously on 43 of the 50 items, and the judge returns their majority label on 43 of the 50, 86.0% (Cohen's $\kappa = 0.653$).

The judge agrees on every channel and memory item and on all ten non-matches. All seven disagreements are the same agent assertion, which asks `agent1` to lay out a scaffold of function stubs for the task's requirements; the diagnosis found that it does not hold, and the judge read that failure as the hollow output the perturbation introduced, while the annotators did not, since a scaffold is itself made of stubs. The judge is therefore lenient on this case: by the annotators' labels, 33 of its 40 matches hold, which would place the aspect accuracy of APEX at about 65% rather than 79.0%.

H GENERALIZATION BEYOND CODING

To examine a workflow outside coding, we extend AXEBench to multi-hop question answering on HotpotQA (Yang et al., 2018), in its distractor setting, where each question comes with ten documents, only some of which support the answer. AXE compiles a four-agent chain from a natural-language description: a *retriever* selects the supporting sentences and records them with the `record_evidence` tool, a *reasoner* resolves the intermediate entity and then the answer, a *verifier* checks every reasoning step against the recorded evidence, and an *answerer* submits a short answer with the `submit_answer` tool. The workflow thus has ten units: four agents, three channels, the memory and two tools. It is perturbed by the same twelve perturbations as the coding workflows, except that an agent's output is hollowed by replacing part of what it states with placeholders rather than by emptying function bodies. Of the 144 candidate instances generated on twelve questions, 142 changed the recorded run and were kept, and every method is evaluated on 84 of them, 21 of each unit kind. Table 8 reports the results.

Table 8: Unit-level weakness localization on 84 instances of the four-agent question-answering workflow on HotpotQA, as in Table 1. Brackets give Wilson 95% intervals.

	Overall	Agent	Chan.	Mem.	Tool
APEX	**51.2** [40.7, 61.6]	**71.4**	**85.7**	**42.9**	**4.8**
APEX-rule	32.1 [23.1, 42.7]	47.6	76.2	0.0	**4.8**
All-at-once	13.1 [7.5, 21.9]	52.4	—	—	—
Step-by-step	4.8 [1.9, 11.6]	19.0	—	—	—
Binary search	3.6 [1.2, 10.0]	14.3	—	—	—
Random	11.9 [6.6, 20.5]	9.5	14.3	19.0	**4.8**

The results follow the coding workflows. APEX leads overall, 51.2% against 13.1% for the strongest prior method, and on agents, 71.4% against 52.4%; the prior methods remain limited to agents; deciding on the scores alone (APEX-rule) loses memory; and tools remain the hardest unit kind. Among the instances APEX localizes correctly, it identifies the right aspect in 86.0% (37/43).

I ROBUSTNESS ACROSS WORKFLOWS, TOPOLOGIES AND EXAMINER MODELS

Table 1(b) and Appendix H vary the workflow and the task source. This appendix varies the two remaining factors on the three-agent workflow: its topology and the model that examines it.

Topology. The same three agents are wired as the mesh of the main results, as a chain, and as a star, and each arm is perturbed with the same perturbations, with the channel perturbations drawn from

the channels that exist in that arm. Each arm has 24 instances, six per unit kind, at the severe level, and receives its own rubric and the same probe budget. Table 9 reports the result. The intervals of the three arms overlap, so APEX localizes weaknesses comparably across topologies. Agents are localized well in every arm and tools remain the hardest unit kind, as in the main results. Channels are where the arms differ most, and the difference follows the rubric: for the same eight units, the examiner writes 13 assertions for the star and 8 for the chain, one per unit. With a single assertion per channel, the chain's examination does not separate a channel from the agent downstream of it, while the star localizes all six of its channel instances.

Examiner model. The same 24 mesh instances are examined with `gpt-4o-mini` in place of `gpt-5-mini` for the whole examiner, including the rubric, the probes, the judge and the localizer, while the examined workflow is unchanged. Localization is statistically indistinguishable, 41.7% against 45.8%, and so is aspect accuracy, $9/10$ against $9/11$, at a quarter of the cost per instance.

Table 9: Robustness to topology and to the examiner model, on 24 instances per arm (six per unit kind). Brackets give Wilson 95% intervals; aspect accuracy is over the correctly localized instances.

Topology	Examiner	Overall	Agent	Chan.	Mem.	Tool	Aspect	Cost / inst.
mesh	`gpt-5-mini`	45.8 [27.9, 64.9]	83.3	66.7	33.3	0.0	9/11	$0.133
chain	`gpt-5-mini`	29.2 [14.9, 49.2]	66.7	0.0	50.0	0.0	7/7	$0.103
star	`gpt-5-mini`	54.2 [35.1, 72.1]	100.0	100.0	0.0	16.7	9/12	$0.118
mesh	`gpt-4o-mini`	41.7 [24.5, 61.2]	66.7	50.0	0.0	50.0	9/10	$0.031

J Case Study Details

The instance of Figure 5(b) is a coding task of MultiAgentBench, a collaborative schedule-management application, run by the three-agent workflow with its `revise_code` tool revoked from both agents that hold it. `agent1` writes the initial solution through `create_code`; `agent2` and `agent3` can change it only through `revise_code`.

What each method names. All-at-once names `agent2`; step-by-step and binary search name `agent1`. Each reads a trajectory in which the later agents describe changes that never appear in the solution, and each attributes the gap to an agent.

What APEX observed. The examination ran five rounds of two probes. Table 10 lists the failure score of every unit. Neither `revise_code` assertion received a verdict, because no probe produced a turn in which the tool could be seen to apply an edit, so its failure score is zero. The most negative score belongs to the `agent1`–`agent2` channel, which is why APEX-rule names it. The localizer reads the observations behind the scores instead. `agent2`'s assertions hold, with the judge's evidence that "`agent2`'s turn left the shared solution unchanged; no previously present functions were removed or altered by `agent2`"; `agent1`'s assertions hold, with its `create_code` commit visible in the footprint; and the memory's assertions hold, with every entry available to later units. An agent that produced its work and a memory that preserved it, with nothing ever revised, leave the revise tool as the only unit that explains the run, and APEX ranks `revise_code` first.

Table 10: Failure scores $F(u)$ of the case-study instance, with the number of each unit's assertions that received a verdict.

Unit	$F(u)$	probed assertions
`agent1`–`agent2` channel	-0.50	$1/3$
`agent2`–`agent3` channel	-0.25	$2/3$
`agent3`	-0.10	$3/3$
`agent1`, `agent2`, `agent1`–`agent3` channel	0.00	all or most
memory, `create_code`	0.00	$2/2$
`revise_code`	0.00	$0/2$

Repair. The healthy workflow scores 7.0 on this task and the perturbed one 6.5. Given APEX's diagnosis, the repairer restores `revise_code` and the score rises to 7.5; without it, the repairer rewrites `agent1`'s instructions and the score falls to 6.0.

K Failure Analysis

Figure 8 crosses the true unit kind of each of the 165 main instances with the kind of unit APEX identified. The diagonal counts instances where APEX identified a unit of the right kind, which need not be the right unit: of the 9 tool instances on the diagonal, 5 identify the other tool. Off the diagonal, three cells hold most of the errors Section 6.2 discusses, tools identified as agents, memory identified as agents, and channels identified as memory. The figure omits one channel instance on which the examination named no unit.

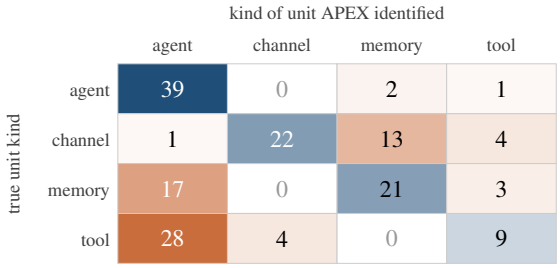

Figure 8: True unit kind against the kind of unit APEX identified, over the 165 main instances. Blue marks the right kind, orange the wrong one, and shading follows the count.

L Extended Related Work

Section 2 summarizes related work; this appendix gives it in full.

LLM-based agentic workflows. Agentic workflows compose role-specialized LLM agents that coordinate through a communication protocol and act through tools (Hong et al., 2024; Qian et al., 2024; Wu et al., 2024; Li et al., 2023; Fourney et al., 2024), building on single-agent designs (Yao et al., 2023; Shinn et al., 2023). Their performance depends on the architecture as much as on the models (Huang et al., 2025; Fu et al., 2026; Orogat et al., 2027), which is why we treat communication, memory and tools as units that can be weak in their own right.

Evaluation of agentic systems. Benchmarks measure how well agentic systems complete tasks in collaborative and competitive settings (Zhu et al., 2025; Sun et al., 2025), over multi-turn interaction (Ma et al., 2024), and on realistic professional and assistant tasks (Xu et al., 2025; Mialon et al., 2024), and PaperBench grades attempts against hierarchical rubrics written for each task (Starace et al., 2025). They report whether a system succeeded rather than which part is responsible, with fine-grained measures annotated by hand; we generate the rubric dynamically around the workflow's units and validate the verdicts judged against it on a human-labeled sample (Section 6).

LLM-as-a-judge and examination-based evaluation. Grading model outputs with an LLM is now common practice (Zheng et al., 2023; Gu et al., 2026), and Agent-as-a-Judge extends it to the intermediate steps of agentic systems (Zhuge et al., 2025). Closest to our notion of examination, Guinet et al. (2024) generate task-specific exams and use item response theory to retain the most informative questions, and PQR refines user queries to elicit agent failures (Lu et al., 2026). Both construct test inputs actively, as APEX does, but assess the system as a whole rather than its units and aspects.

Failure attribution in multi-agent systems. Who&When introduced the task of identifying the agent and the step responsible for a failed run (Zhang et al., 2025b); later methods train dedicated attributors (Zhang et al., 2026; Zhu et al., 2026), localize errors to trajectory spans (Wang et al., 2026) or analyze the interaction graph (Xia et al., 2026), related studies diagnose recurring failure modes (Cemri et al., 2025; Kim et al., 2026), and the area has been surveyed (Qi et al., 2026). DoVer validates attributions by intervening and re-executing, and observes that single-step attribution is often ill-posed (Ma et al., 2026a). All of these approaches start from a failed trajectory and return a

pointer into it. The term localization comes from software fault localization (Wong et al., 2016), to which LLM-based agents have also been applied (Qin et al., 2024).

Fault injection in multi-agent systems. Huang et al. (2025) inject errors into agents with AutoTransform and AutoInject to measure the resilience of different topologies, AgenTracer injects faults and replays trajectories counterfactually to train its attributor (Zhang et al., 2026), and FlowFixer repairs workflows from a failure diagnosis (Ma et al., 2026b). However, none of these works localizes which units of a workflow are weak, and in what respect, before a failure is observed, and no existing benchmark supports evaluating such localization. This gap motivates APEX and AXEBench: AXE extends AutoInject-style perturbation to channels, memory and tools, generating AXEBench, a labeled benchmark of executable, perturbed workflows, following generator-based benchmarks (Chang et al., 2025).