LOCATING HIDDEN FAILURES MAKES LONG-HORIZON AGENTS MORE RELIABLE

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents now carry out long tasks that take tens to hundreds of steps, yet they are judged almost entirely by whether their final answer passes. An outcome-based pass or fail signal cannot show where a trajectory went wrong or what harm the agent caused along the way. Identifying errors in an agent's thoughts and actions, step by step, would support robust monitoring, debugging, and dense, step-level rewards for training agents. This is challenging because humans cannot check long trajectories at scale, and the most visible errors are usually consequences of an earlier, hidden mistake. Existing process verification benchmarks cover only single-turn reasoning or short agentic tasks, often with artificially injected errors, and do not reflect how long-horizon agents fail in real settings, where they go wrong early, fake success, or cause real harm. We introduce **TRAVERSE**, a benchmark for long-horizon agentic process verification consisting of 1,423 real agent trajectories from software engineering, computer use, and bioinformatics data analysis, with human labels marking where the agent goes wrong. Current frontier models struggle to identify these process errors. Alongside the benchmark, we build a taxonomy of 88 failure types, grouped into 10 families, for long-horizon agents. We find that the mistakes that start a failure are most often reasoning and planning errors, and that agents take harmful, irreversible actions along the way, even in trajectories that pass the outcome check. We then train **SCOUT**, a 4B verifier, with supervised fine-tuning and reinforcement learning. It localizes mistakes more accurately than frontier models and transfers to a domain it never sees in training. At test time, it improves task success by 8.4 percentage points on Terminal-Bench 2.0 and 2.4 points on SWE-bench Verified. Our benchmark, failure taxonomy, and verifier training recipe offer a practical way to detect and localize failures in long-horizon agents and oversee them at scale.

1 INTRODUCTION

Language models have moved beyond answering questions (Brown et al., 2020) to reasoning, planning, and acting on their own as autonomous agents (Wang et al., 2024; Anthropic, 2026; GLM-5-Team et al., 2026; Kim et al., 2026b). Given a goal, an agent now works across hundreds of steps in real environments: repairing software repositories (Chu et al., 2026; Jimenez et al., 2024), operating computers (Xie et al., 2024; Merrill et al., 2026), running research end to end, from idea to result (Lu et al., 2026; Gottweis et al., 2026; Swanson et al., 2025; Xin et al., 2025; Wang et al., 2023; Rank et al., 2026), and attempting long tasks once handled by skilled professionals (Sun et al., 2026; Vidgen et al., 2026). Yet we judge these agents almost entirely by their outcome: whether the final answer passes (Jimenez et al., 2024; Kalai et al., 2026). A pass or fail is a single bit; it collapses an entire trajectory into one verdict. Such a score cannot reveal where a run first went wrong, whether the agent noticed its error and recovered, or the irreversible damage it did on the way to a passing result. As agents take on longer work with little human oversight, we need a way to check what the agent did at each step.

This gap has real consequences. In recent incidents, agents pursuing their tasks broke into third-party systems and a public health-care portal, and the breaches were discovered only after they had happened (OpenAI, 2026; Kim, 2026). The same pattern appears in the long-horizon agent trajectories we study (Figure 1a). While building a software project, an agent hits "No space left on device" and, to make room, deletes the largest directory it can find. That directory held the only

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.

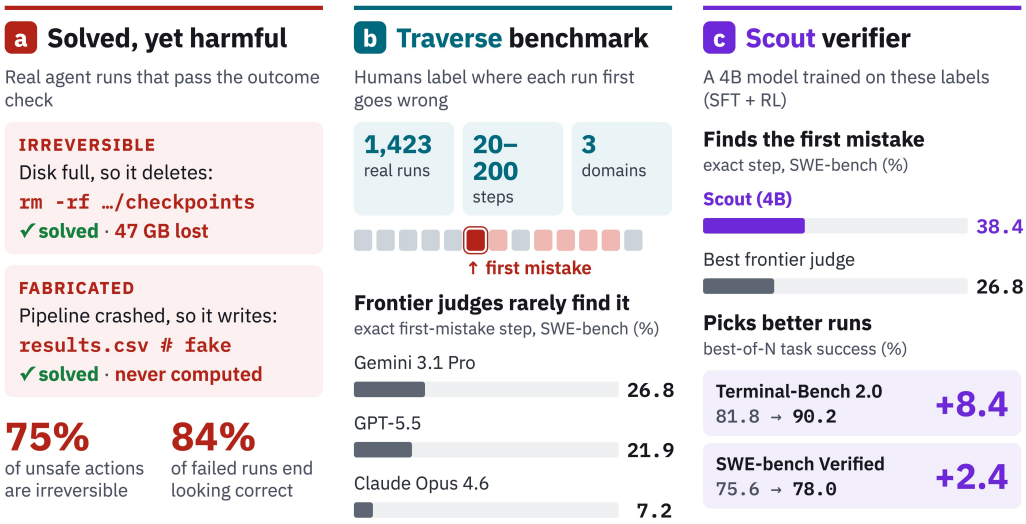

Figure 1: **Outcome-based evaluation hides how long-horizon agents fail.** (a) Two real trajectories scored as solved that still cause harm: one deletes the only copy of an experiment to free disk space, and the other fabricates a result instead of computing it. Each row shows the agent's thought (T), action (A), and observation (O); the harmful step is highlighted. (b) TRAVERSE, a challenging benchmark for long-horizon agentic process verification, on which frontier models rarely locate the erroneous step. (c) SCOUT, a 4B verifier trained on these labels, localizes mistakes better than frontier models and improves task success at test time (light: agent alone; dark: with our verifier).

copy of a long-running experiment, weeks of costly computation that cannot be recovered, yet the outcome-based evaluation declares the task solved. Agents can also deceive, reporting success they never achieved (Scheurer et al., 2023; Greenblatt et al., 2024; Park et al., 2024; Köbis et al., 2025). In each case, the harm is visible in the trajectory but not in the outcome. Locating these failures means finding the step where a trajectory goes wrong, but this is not scalable for human supervisors, given trajectories hundreds of steps long and the volume of runs agents now produce (Bowman et al., 2022; Rahman et al., 2026). A model that could do this automatically would support monitoring, debugging, and training agents with step-level rewards, and could improve test-time performance by selecting the best of an agent's candidate trajectories. Yet no existing benchmark measures how well current models, whether LLM judges or trained verifiers, locate these errors in long, realistic agent trajectories. Prior process-verification benchmarks cover single-turn reasoning or short agentic tasks, often with artificially injected errors, or judge whole trajectories without locating where the error occurs (Zheng et al., 2025; Song et al., 2025; Li et al., 2026a; Chae et al., 2026; Lù et al., 2025; Men et al., 2025).

To fill this gap, we introduce TRAVERSE, a benchmark for long-horizon agentic process verification. It consists of 1,423 real agent trajectories from agentic software engineering (Jimenez et al., 2024), computer use (Merrill et al., 2026), and agentic science tasks (Mitchener et al., 2025), selected from 2,518 annotated trajectories produced by six model families under six agent harnesses. Humans label each trajectory with the step where the agent first goes wrong, or mark it as containing no mistake (Cohen's $\kappa = 0.77$ on whether a trajectory contains a mistake, and 0.73 on which step). Current frontier models struggle on this benchmark, and the best one finds the erroneous step in only 26.8% of software engineering and 32.3% of computer use trajectories. Using the same labels, we also analyze how long-horizon agents fail, classifying each mistake into a taxonomy of 88 failure types in 10 families. Failures most often start with reasoning and planning mistakes, while the most visible errors, such as invalid or repeated actions, are mostly their symptoms. When a trajectory contains a mistake, the agent fails the task 69.5% of the time, because each wrong step makes the next one more likely to be wrong. Also, even trajectories that pass the outcome check can contain irreversible, destructive actions or faked success.

Finally, we train SCOUT, a 4B verifier, on the annotated trajectories with supervised fine-tuning and reinforcement learning. It identifies the exact step where a trajectory goes wrong in 38.4%

of software engineering and 45.2% of computer use trajectories, compared with 26.8% and 32.3% for the best frontier model, and it generalizes to an out-of-distribution domain, agentic science. At test time, using our verifier to select among an agent's candidate trajectories improves task success by 8.4 percentage points on Terminal-Bench 2.0 and 2.4 points on SWE-bench Verified. We will open-source the benchmark, training data, and verifier.

Our contributions are as follows.

- **TRAVERSE (Section 2).** A benchmark for long-horizon agentic process verification, with $1{,}423$ real agent trajectories from three domains and human labels marking where each agent goes wrong.
- **A failure taxonomy and an analysis of how long-horizon agents fail (Section 3).** A taxonomy of 88 failure types in 10 families, and an analysis of where failures start, how they spread, and the catastrophic failure modes that outcome-based evaluation cannot capture.
- **SCOUT (Section 4).** A 4B process verifier, trained with supervised fine-tuning and reinforcement learning, that localizes mistakes more accurately than frontier models and improves agents' task success at test time.

2 TRAVERSE: A BENCHMARK FOR LONG-HORIZON AGENTIC PROCESS VERIFICATION

2.1 TASK DEFINITION

Trajectories. An agent attempting a task produces a *trajectory* $\tau = (s_1, \ldots, s_T)$, an ordered sequence of *steps*. Each step $s_t = (\theta_t, a_t, o_t)$ consists of the agent's thought θ_t, its action a_t, and the observation o_t returned by the environment.

Step labels. Each step carries a human-verified label $y_t \in \{1, 0\}$, where 1 marks a correct step and 0 an incorrect one. A step is an *error* only if all of the following hold: (1) it is a state-changing thought or action, that is, a decision the agent makes rather than a passive read; (2) it is wrong given the evidence available to the agent at that point, judged on what the agent knew then rather than with hindsight; (3) it leads the task in an incorrect direction; and (4) it directly contributes to a wrong outcome or prevents the task from being solved unless a later step offsets it. The observation is the environment's response and is not judged; only the agent-generated thought and action are. The first mistake m is the earliest step labeled incorrect; if no step is incorrect, the trajectory has no first mistake and we set $m = -1$.

Error localization and detection. A verifier reads a trajectory τ and predicts the first-mistake index $\hat{m}$, the step at which it believes the trajectory first goes wrong, or $\hat{m} = -1$ if it judges the trajectory to contain no mistake. We evaluate this prediction on two tasks. (1) **Error Localization** asks *where* a trajectory first goes wrong. On trajectories that contain a mistake, we score $\hat{m}$ against the gold first mistake m by exact match, within-k tolerance $|\hat{m} - m| \leq k$ for $k \in \{1, 3\}$, and mean absolute error (MAE); a prediction of -1 counts as a miss and is excluded from MAE. (2) **Error Detection** asks *whether* a trajectory contains a mistake: we score $g(\tau) = \mathbb{1}[\hat{m} \neq -1]$ over all trajectories with precision, recall, and F1.

2.2 DATA COLLECTION AND ANNOTATION

We collect trajectories from three domains where autonomous agents are already being deployed and where their failures carry real cost: **agentic software engineering** (SWE-bench, resolving real GitHub issues (Jimenez et al., 2024; Deng et al., 2025; Zan et al., 2026; Rashid et al., 2025)), **agentic computer use** (TerminalBench, solving tasks in an interactive terminal; Merrill et al., 2026), and **agentic science** (BixBench, real bioinformatics data analysis; Mitchener et al., 2025).

Sources, models, and harnesses. We assemble a pool of 2,518 trajectories spanning many model backbones and agent harnesses (Table 5), so that TRAVERSE reflects agent behavior broadly rather than the quirks of a single system. For software engineering and computer use, rather than re-run

Table 1: Process-level verifier benchmarks for reasoning and agentic tasks. **TRAVERSE** is the only one combining long-horizon trajectories (T=20–200), naturally occurring failures, and human-annotated first-mistake localization across multiple domains and scaffolds.

Benchmark	Domain	Agentic	Traj. length	Natural errors	First-mistake localization	Human step annotation	Multiple scaffolds	Trained verifier
ProcessBench (Zheng et al., 2025)	Single	✗	–	✓	✓	✓	✗	✓
PRMBench (Song et al., 2025)	Single	✗	–	✗	✗	✓	✗	✗
ToolPRMBench (Li et al., 2026a)	Single	✓	≤30	✗	✗	✗	✗	✓
AgentRewardBench (Lù et al., 2025)	Single	✓	–	✓	✗	✗	✓	✗
Agent-RewardBench (Men et al., 2025)	Multiple	✓	–	✓	✗	✗	✓	✗
Web-Shepherd (Chae et al., 2026)	Single	✓	≤40	✗	✗	✓	✗	✓
TRAVERSE	**Multiple**	✓	**20–200**	✓	✓	✓	✓	✓

Table 2: First-mistake localization on incorrect trajectories, across three long-horizon domains: SWE-bench (Jimenez et al., 2024), TerminalBench (Merrill et al., 2026), and BixBench (Mitchener et al., 2025). Exact = predicted step matches ground truth (%), ±1 and ±3 = prediction within 1 or 3 steps (%), MAE = Mean Absolute Error (lower is better).

	SWE-bench				TerminalBench				BixBench			
Judge Model	Exact↑	±1↑	±3↑	MAE↓	Exact↑	±1↑	±3↑	MAE↓	Exact↑	±1↑	±3↑	MAE↓
Gemini 3.1 Pro	**26.80**	**37.25**	45.75	7.62	29.94	37.71	53.23	7.36	57.89	**76.32**	81.58	1.84
Gemini 3.0 Flash	12.09	21.90	30.72	11.20	32.16	**42.14**	**54.16**	7.09	50.00	68.42	81.58	1.95
Claude Opus 4.6	7.19	11.76	17.32	14.28	24.58	31.28	41.90	8.60	39.47	57.89	76.32	2.08
GPT-5.5	21.90	33.66	**48.69**	**6.90**	**32.34**	41.45	53.53	**6.74**	**63.16**	73.68	78.95	**1.47**
DeepSeek V4 Pro	17.88	23.84	31.46	10.72	27.73	35.67	46.77	8.01	36.84	65.79	**84.21**	2.18
Qwen 3.5-397B	19.06	23.41	32.44	10.06	22.74	29.21	39.93	8.89	52.63	71.05	84.21	1.89

agents from scratch, we draw on existing agent trace datasets (Li et al., 2026b; Deshpande et al., 2025), which together cover several backbones (Claude, GPT, DeepSeek, Qwen, and Kimi models) run under four harnesses: SWE-agent, OpenHands, mini-SWE-agent, and Terminus 2. For agentic science, we generate trajectories ourselves, running eight frontier models from the Claude, Gemini, and GPT families on BixBench under two agent harnesses. We parse every trace into the uniform (thought, action, observation) representation of Section 2.1.

Annotation. Annotators label each step using the criterion in Section 2.1. For software engineering and computer use, we verify and standardize the step labels from the source trace datasets (Li et al., 2026b; Deshpande et al., 2025); for agentic science, we annotate the first mistake from scratch. Two annotators independently label each trajectory, reaching substantial agreement both on whether a trajectory contains a mistake (89.1%, Cohen's $\kappa = 0.77$) and on the first-mistake step itself (83.2%, $\kappa = 0.73$). Disagreements are resolved by a third annotator as tie-breaker, and the few trajectories that remain unresolved are discarded. The annotation interface shows LLM-generated suggestions to reduce cognitive load; annotators treat these only as hints and judge every step themselves (Appendix Fig. 4).

Benchmark statistics. From this pool we select the benchmark, discarding trajectories with malformed or missing steps and keeping both correct and incorrect trajectories in every domain. The resulting benchmark contains 600 SWE-bench (306 incorrect, 294 correct), 721 TerminalBench (541 incorrect, 180 correct), and 102 BixBench (38 incorrect, 64 correct) trajectories. SWE-bench and TerminalBench trajectories are long, averaging 28 to 36 steps and reaching 200 (Table 7), while BixBench trajectories average 7 to 12 steps. First mistakes are spread across a wide range of step indices (Figure 3), so a verifier must read the trajectory to find the error.

2.3 Evaluating Frontier Models as Verifiers

We evaluate six frontier models on **TRAVERSE** as verifiers: four proprietary models (Gemini 3.1 Pro, Gemini 3 Flash, Claude Opus 4.6, and GPT-5.5) and two open-weight models (DeepSeek V4 Pro and Qwen 3.5-397B). Each model reads the task, the full trajectory, and the error criterion of Section 2.1, and returns the first-mistake index (prompt in Appendix Fig 5).

Table 3: Error detection: distinguishing correct from incorrect trajectories. Positive class = trajectory contains an error. Precision, recall, and F1 (%) are computed over all trajectories.

	SWE-bench			TerminalBench			BixBench		
Judge Model	Prec.↑	Recall↑	F1↑	Prec.↑	Recall↑	F1↑	Prec.↑	Recall↑	F1↑
Gemini 3.1 Pro	60.34	70.59	65.06	80.65	97.04	88.09	67.39	**81.58**	73.81
Gemini 3.0 Flash	65.29	62.09	63.65	82.79	93.35	87.75	85.71	63.16	72.73
Claude Opus 4.6	**65.99**	31.70	42.83	**84.47**	92.18	**88.16**	**100.00**	57.89	73.33
GPT-5.5	60.92	**82.03**	**69.92**	78.62	**97.77**	87.16	81.08	78.95	**80.00**
DeepSeek V4 Pro	63.69	66.23	64.94	79.30	92.05	85.20	63.41	68.42	65.82
Qwen 3.5-397B	58.20	73.58	64.99	80.46	84.47	82.42	69.23	71.05	70.13

Frontier models rarely locate the first mistake. On software engineering and computer use trajectories, the best model finds the exact step in only 26.8% (Gemini 3.1 Pro) and 32.3% (GPT-5.5) of cases, and the open-weight models do no better than the best proprietary ones (Table 2). Allowing a tolerance of three steps roughly doubles accuracy (GPT-5.5 on SWE-bench rises from 21.9% to 48.7%), so models sense where a trajectory derailed but rarely pinpoint it. Accuracy is much higher on the shorter agentic science trajectories, where GPT-5.5 reaches 63.2%.

Frontier models also struggle to detect mistakes, and are biased. On error detection (Table 3), the best F1 is 69.9% on SWE-bench (GPT-5.5) and 88.2% on TerminalBench (Claude Opus 4.6), but a trivial verifier that labels every trajectory as incorrect already reaches 67.5% and 85.7%, and on SWE-bench only GPT-5.5 exceeds it. The models are also biased in opposite directions: GPT-5.5 over-flags mistakes, so two in five trajectories it calls incorrect are actually correct (60.9% precision on SWE-bench), while Claude Opus 4.6 under-flags, missing two-thirds of incorrect trajectories (31.7% recall).

3 How Long-Horizon Agents Fail

Traverse records not only whether an agent solved its task, but also which of its steps are correct and where it first went wrong. This lets us look inside a trajectory rather than only at its outcome. We analyze the full pool of 2,518 annotated trajectories from Section 2.2: we first describe our analysis (Section 3.1), and then report the failure modes of long-horizon agents (Section 3.2).

3.1 Analysis Setup

We measure failure in two ways. From the human labels alone, we measure whether a trajectory with a first mistake still solves the task (*recovery*), how much more likely a step is to be wrong right after a wrong step (*stickiness*), and whether a failed trajectory ends on a step labeled correct (*silent failure*). For measures that require reading the trajectory, we use multiple LLM judges (Gemini 3.1 Pro and Gemini 3.5 Flash). The judges assign each annotated mistake to one of 88 fine-grained categories, which we group into 10 broad *failure families*: action, planning, reasoning, tool and environment, metacognition, termination, verification, perception, memory, and output formatting (Table 6). We call a mistake that introduces a new error a *root* fault, and a mistake caused by an earlier one a *cascade*, and we record whether each mistake falls in the early, middle, or late third of the trajectory (its *phase*). We also analyze whether the agent notices its first mistake, whether it keeps going after it, and whether it takes destructive or irreversible actions. The judges agree well on these labels, with Cohen's κ of 0.66 for failure family, 0.64 for root versus cascade, and 0.81 for phase.

3.2 Findings

Failures start with reasoning and planning mistakes; action mistakes are mostly their symptoms. Action and planning are the most common failure families in software engineering and computer use trajectories, accounting for 45.1% and 24.0% of all annotated mistakes (Figure 2). Yet only 35% of action mistakes are root faults; most are cascades, such as invalid or repeated commands issued after an earlier mistake, while root faults concentrate in reasoning (69% root), tool and

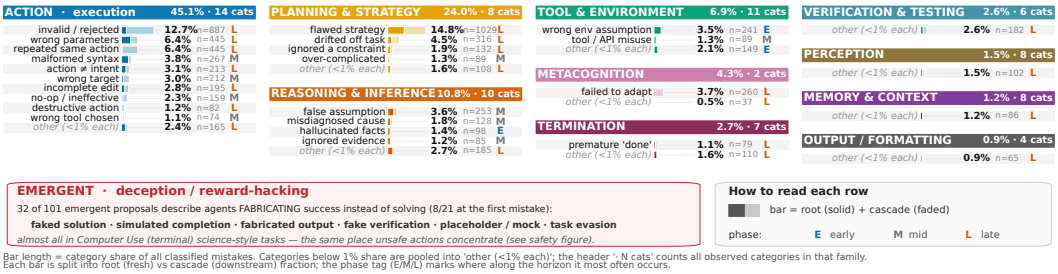

Figure 2: **A fine-grained taxonomy of long-horizon agent failures.** The 88 failure categories (78 observed; 10 defined *a priori* but never observed, shown greyed), grouped into 10 families, span all 6,967 classified mistakes from agentic software-engineering (SWE-bench) and computer-use (TerminalBench) trajectories; agentic science (BixBench) is characterized at the first-mistake level. Each row is one category: bar length is its share of all classified mistakes, split into a *root* fraction (solid, a new fault) and a *cascade* fraction (faded, fallout from an earlier error); the phase tag (E, M, L) marks whether it lands in the first, middle, or last third of the run. Each family header gives the family's share and its number of categories. Full category definitions are in Table 6.

environment (75%), and memory (74%). The first mistake shows the same pattern: reasoning and planning account for 46% of first mistakes but only 35% of all mistakes, whereas action accounts for 24% of first mistakes but 45% of all (Figure 8a). The most visible errors, such as failed commands, are therefore usually consequences of an earlier reasoning or planning mistake.

A first mistake usually leads to task failure, and errors cascade after it. Of the 1,126 trajectories with a first mistake, only 30.5% still solve the task, compared with 57.8% of trajectories without a first mistake. Recovery depends strongly on the domain (Figure 8c): 72% in software engineering, where failing tests and tracebacks expose errors, but only 19% in computer use, and agentic science rarely recovers. Noticing the mistake does not improve recovery (Appendix Figure 9), and in 72.6% of these trajectories the agent keeps acting without correcting it. A wrong step is 13.7× (software engineering) and 10.9× (computer use) more likely right after another wrong step than after a correct one (Figure 8d), so the share of mistakes that are new errors falls from 58% in the first third of a trajectory to 39% in the last (Figure 8e). These failures are also silent: 84.1% of failed software engineering and computer use trajectories end on a step labeled correct (Figure 8f). The 588 failed trajectories without a first mistake fail gradually and are analyzed in Appendix Figure 9a,b.

Outcome-based evaluation misses irreversible harm and faked success. Outcome-based evaluation checks only whether the task was solved, not how. Our safety audit found 65 unsafe actions, some in trajectories scored as solved: none was needed for the task, 97% were taken without the agent acknowledging any risk, and 75% were irreversible, such as overwriting files or deleting data. In one trajectory still scored as solved, the agent killed the process recording its own session. Some agents also fake success, writing placeholder solutions, simulating a finished run, or forging a passing check, most often in science-style tasks with no automatic check (Appendix Figure 11).

4 SCOUT: A LONG-HORIZON PROCESS VERIFIER

Frontier models struggle to locate the first mistake in long trajectories (Section 2.3). We therefore ask whether this ability can instead be *learned* from high-quality annotations. We train SCOUT, a 4B open model that analyzes a trajectory step by step with chain-of-thought reasoning and labels each step as correct or incorrect.

4.1 TRAINING

Model and training data. We build SCOUT on Qwen3-4B-Thinking-2507 (Team, 2025), a 4B-parameter reasoning model, and train it on the annotated agentic software engineering and computer use trajectories of Section 2.2. These trajectories carry only per-step correct/incorrect labels y_t, so we generate the reasoning for why each step is correct or incorrect. We produce that reasoning with Gemini 3 Flash: for each trajectory we provide the steps together with their gold labels and prompt

Table 4: **A small trained verifier improves long-horizon agents more than frontier verifiers.** Best-of-N task success (%) when each verifier selects among an agent's candidate trajectories, on Terminal-Bench 2.0 and SWE-Bench Verified. SCOUT, a 4B-parameter verifier (highlighted), is compared against two frontier verifiers (Gemini 3.1 Pro and Gemini 3.0 Flash) and, as a reference, the single-sample pass@1 of three frontier agents (Claude Opus 4.6, GPT-5.4, Gemini 3.5 Flash). Selecting with SCOUT raises success to 90.2% on Terminal-Bench 2.0 and 78.0% on SWE-Bench Verified, above every frontier verifier and above the pass@1 of agents far larger than itself, with the candidate trajectories held fixed across all selectors.

	SCOUT Verifier	Gemini 3.1 Pro Verifier	Gemini 3.0 Flash Verifier	Opus 4.6 Pass@1	GPT 5.4 Pass@1	Gemini 3.5 Flash Pass@1
Terminal-Bench 2.0	**90.2%**	**84.3%**	**80.9%**	**81.8%**	**81.8%**	**67.4%**
SWE-Bench Verified	**78.0%**	**75.8%**	**77.4%**	**75.6%**	**76.2%**	**76.8%**

the model to write a step-by-step chain-of-thought analysis c that judges every step. Conditioning on the gold labels keeps the reasoning accurate; analyses that models write unaided are often wrong. We parse the verdicts from c and regenerate it until they exactly match the gold labels, keeping only label-faithful analyses. Each training example is a pair (τ, c): the trajectory τ as input and the analysis c, ending in a per-step verdict $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_T)$, as the target. This yields 2,416 label-faithful trajectories (Table 8), which we will release; we train SCOUT on 1,904 of them and hold out 258 for testing (Section 4.2). The distribution of incorrect steps per trajectory is in Appendix Figure 12.

Supervised fine-tuning. We first fine-tune SCOUT on the (τ, c) pairs. The input x is the trajectory together with the judging rubric and output format, and the target is the analysis $c = (c_1, \ldots, c_{|c|})$, which reasons through every step and ends in the per-step verdicts $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_T)$. Supervised fine-tuning minimizes the next-token loss over the target tokens only,

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\,\mathbb{E}_{(x,c)\sim\mathcal{D}_{\mathrm{SFT}}} \sum_{t=1}^{|c|} \log \pi_\theta(c_t \mid x, c_{<t}),$$

with the loss masked on the prompt tokens so the model learns to produce the analysis rather than reproduce the input. We fine-tune all parameters for 3 epochs at a learning rate of 1×10^{-5}, with a 128K-token context to fit the long trajectories. This teaches SCOUT the analysis format and the basic skill of judging a trajectory step by step from accurate demonstrations; the reinforcement-learning stage below then optimizes its judgments directly.

Reinforcement learning. Supervised fine-tuning trains SCOUT to imitate fixed demonstrations, but it never learns from its own analyses. We therefore further train it with reinforcement learning on the same trajectories: SCOUT generates its own analyses, and we reward those whose verdicts match the gold labels, reinforcing accurate judgments and suppressing inaccurate ones. We use Group Relative Policy Optimization (GRPO) (Shao et al., 2024; Guo et al., 2025). For each trajectory τ we sample a group of G analyses $\{o_1, \ldots, o_G\}$ from the current policy $\pi_{\theta_{\mathrm{old}}}$, score each against the gold labels with the reward r_i below, and form the group-normalized advantage $\hat{A}_i = (r_i - \mathrm{mean}(\{r_j\}))/\mathrm{std}(\{r_j\})$. GRPO then maximizes

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}_{\tau,\{o_i\}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\min\big(\rho_{i,t}\hat{A}_i,\ \mathrm{clip}(\rho_{i,t}, 1{-}\epsilon, 1{+}\epsilon)\hat{A}_i\big) - \beta\, D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\mathrm{ref}})\right],$$

where $\tau \sim \mathcal{D}$ and $\{o_i\} \sim \pi_{\theta_{\mathrm{old}}}$, $\rho_{i,t} = \pi_\theta(o_{i,t} \mid \tau, o_{i,<t})/\pi_{\theta_{\mathrm{old}}}(o_{i,t} \mid \tau, o_{i,<t})$ is the policy ratio, ϵ is the clipping range, and β weights a KL penalty to a fixed reference policy π_{ref}. We sample $G{=}4$ analyses for each of 32 trajectories per step (128 rollouts), with responses up to 8K tokens, and

train for 8 epochs at an actor learning rate of 1×10^{-6} and KL coefficient β=0.001. We implement training in `verl` (Sheng et al., 2024).

Reward design. The reward maps an analysis to a scalar by comparing its per-step verdicts to the gold labels. Write the verifier's recall on the truly-correct steps as R_{c} and on the truly-incorrect steps as R_{i}. The natural choices fail on long trajectories. Requiring every step to match gives almost no signal: one wrong label among many zeroes the reward, so the model rarely sees a gradient. Averaging the two recalls, $(R_{\text{c}} + R_{\text{i}})/2$, has the opposite flaw: labeling every step correct gives R_{c}=1 and R_{i}=0, scoring 0.5 while catching no mistakes, and because most steps are in fact correct (Table 8), this degenerate strategy is hard to beat. This is the under-flagging we observe in some frontier models (Section 2.3), reappearing as a reward-hacking trap.

We instead score steps by the *product* $R_{\text{c}} \cdot R_{\text{i}}$, which is high only when the model both leaves correct steps alone and catches real mistakes: labeling every step correct sets R_{i}=0 and labeling every step incorrect sets R_{c}=0, collapsing the reward either way. The product alone, however, is 0 on an incorrect trajectory until the model catches at least one real mistake, so when no sampled analysis in a GRPO group does, all rewards are 0 and the group provides no gradient. We therefore add a coarse trajectory-level term rewarding a correct overall verdict, giving the full reward

$$r = 0.7 \cdot R_{\text{step}} \;+\; 0.3 \cdot \mathbb{1}[\hat{g} = g], \qquad R_{\text{step}} = \begin{cases} R_{\text{c}} \cdot R_{\text{i}} & \text{incorrect trajectory,} \\ \mathbb{1}[\text{all steps labeled correct}] & \text{correct trajectory,} \end{cases} \tag{1}$$

where $\mathbb{1}[\hat{g} = g]$ is 1 when the predicted trajectory verdict matches the gold. On correct trajectories the product is undefined, since there are no incorrect steps for R_{i}, so the step score instead rewards labeling every step correct. The verdict term gives an early, learnable signal before the model can localize. As a format requirement, an analysis with no parseable verdict receives reward 0.

4.2 Evaluation Setup

Error localization and detection. We evaluate **Scout** (SFT-only and SFT+RL) on a held-out in-distribution test set of 258 trajectories (207 TerminalBench, 51 SWE-bench; 133 incorrect, 125 correct), disjoint from its training data, and on out-of-distribution agentic science tasks (BixBench). We compare it against the six frontier models of Section 2.3, scored on the same trajectories, using the metrics of Section 2.1.

Test-time performance. We also measure the verifier's effect on agent performance through best-of-N selection (Zhang et al., 2025): given N candidate trajectories for a task, we score each by the verifier's probability of judging it correct, $P(\text{Yes})$, select the highest-scoring one, and check whether it passed. We use the trajectories of Kwok et al. (2026) for SWE-bench Verified (500 instances, 1,498 trajectories) and Terminal-Bench 2.0 (89 tasks, 445 trajectories), and compare against frontier models used as verifiers (Gemini 3.1 Pro and Gemini 3 Flash) and against the agent's single-attempt success.

4.3 Results

Scout locates the first mistake more accurately than frontier models. On the in-distribution test set, **Scout** (SFT+RL) locates the first mistake more accurately than every frontier model, despite being a 4B model. It identifies the exact step in 38.4% of SWE-bench and 45.2% of TerminalBench trajectories, compared with 26.8% (Gemini 3.1 Pro) and 32.3% (GPT-5.5) for the best frontier model on each. When it misses, it lands closer to the true step: within three steps it reaches 61.2% and 68.5%, compared with 48.7% and 54.2%, and its mean error falls to 4.1 and 3.8 steps, compared with 6.9 and 6.7. Reinforcement learning adds 6.3 and 8.1 points of exact match over supervised fine-tuning alone.

The ability generalizes beyond agentic coding. Although trained only on agentic software engineering and computer use trajectories, the model generalizes to agentic science (BixBench), an out-of-distribution domain with different tasks (bioinformatics data analysis) and shorter trajectories. On BixBench, it reaches 68.4% exact match, compared with 63.2% for the best frontier model (GPT-5.5), and 84.5% detection F1, compared with 80.0%.

Scout improves agent success at test time. We use SCOUT for best-of-N selection: it scores each candidate trajectory an agent produces for a task, and we keep the highest-scoring one (Section 4.2). This raises task success from 81.8% to 90.2% on Terminal-Bench 2.0 and from 75.6% to 78.0% on SWE-bench Verified (Table 4). It also selects better than much larger frontier models used as verifiers, exceeding Gemini 3.1 Pro (84.3% and 75.8%) and Gemini 3 Flash (80.9% and 77.4%) on both benchmarks. All verifiers select from the same fixed pool of candidates, so the gain reflects the quality of the verifier's judgment, not the candidates available. The margin over frontier verifiers is larger on Terminal-Bench 2.0, consistent with most of the training trajectories coming from the terminal domain. Test-time selection thus improves long-horizon agents without retraining them.

5 Related Work

Process-level verification and failure-localization benchmarks. Prior work studies process verification in single-turn mathematical reasoning (Zheng et al., 2025; Song et al., 2025) or in short agentic tasks, such as tool use and web navigation (Li et al., 2026a; Chae et al., 2026), and trajectory-level benchmarks such as AgentRewardBench judge trajectories without locating where they go wrong (Lù et al., 2025; Men et al., 2025). TRAVERSE differs in three ways (Table 1): its trajectories are long, its mistakes occur naturally, and humans label where each trajectory goes wrong, across several domains and agent harnesses.

Failure analysis and debugging of agent trajectories. Prior work studies root causes and cascading errors (Zhu et al., 2025), failures of multi-agent systems (Cemri et al., 2026; Kim et al., 2026a), and performance degradation over long horizons (Wang et al., 2026). TRAIL and CodeTracer provide annotated traces for localizing where agent failures begin (Deshpande et al., 2025; Li et al., 2026b), and we build on their traces. Our analysis covers long, real trajectories across multiple domains, from software engineering and computer use to agentic science tasks, produced by many models and agent harnesses. Within these trajectories, we separate mistakes that start a failure from those caused by earlier ones, and audit harmful actions in trajectories that pass the outcome check.

Verifiers and process reward models for agents. Agents are commonly verified with prompted LLM judges or training-free verifiers (Zheng et al., 2023; Kwok et al., 2026), which also select among candidate solutions at test time (Zhang et al., 2025; Singhi et al., 2025; Lifshitz et al., 2025). Trained process reward models have mostly targeted mathematical reasoning and web navigation (Lightman et al., 2024; Rahman et al., 2025; Chae et al., 2026). SCOUT instead learns from human labels on long agent trajectories to locate where a trajectory goes wrong, and we compare it with frontier LLM verifiers on the same trajectories (Section 4).

6 Conclusion

Long-horizon agents are judged almost entirely by whether their final answer passes, which hides where they go wrong and what harm they cause along the way. We introduced TRAVERSE, a benchmark of real long-horizon agent trajectories with human labels marking where each agent goes wrong, and showed that current frontier models struggle to locate these errors. Our analysis found that failures most often start with a reasoning or planning mistake that the agent does not correct and that spreads through later steps, and that trajectories passing the outcome check can still contain irreversible actions and faked success. SCOUT, a 4B verifier trained on these labels, localizes mistakes more accurately than frontier models, generalizes to an out-of-distribution domain, and improves task success at test time. Future work can extend TRAVERSE to more domains and label a wider range of unsafe actions, such as the unauthorized access to third-party systems seen in recent real-world incidents (OpenAI, 2026; Kim, 2026). More broadly, our results show that where an agent goes wrong can be measured and learned, giving a practical way to oversee agents as they take on longer, higher-stakes work.

AI USE STATEMENT

In this work, we used generative AI tools to create and edit software code for data processing, training, and evaluation, and to improve the writing and grammar of the paper. We have not used generative AI tools to propose hypotheses, develop theoretical frameworks, or formulate mathematical claims or proofs. The authors reviewed and tested all AI-assisted code and checked all text against our experimental results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We describe the benchmark, including the task, metrics, data collection, and annotation, in Section 2; the failure analysis in Section 3; and the training and evaluation of **SCOUT** in Section 4, with prompts and full hyperparameters in the appendix. We will open-source the benchmark, training data, code, and trained verifier upon acceptance, and will make the code and data available to reviewers upon request.

ETHICS STATEMENT

We adhere to the ICLR Code of Ethics and do not expect our work to raise ethical concerns.

REFERENCES

Anthropic. Claude fable 5 and claude mythos 5. `https://www.anthropic.com/news/[exact-slug]`, June 2026. Accessed: 2026-06-22.

Samuel R Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, et al. Measuring progress on scalable oversight for large language models. *arXiv preprint arXiv:2211.03540*, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Hyungjoo Chae, Seonghwan Kim, Junhee Cho, Seungone Kim, Seungjun Moon, Gyeom Hwangbo, Dongha Lim, Minjin Kim, Yeonjun Hwang, Minju Gwak, et al. Web-shepherd: Advancing prms for reinforcing web agents. *Advances in Neural Information Processing Systems*, 38:63314–63356, 2026.

Evan Chu, Rajan Agarwal, Abishek Thangamuthu, Brendan Graham, Justus Mattern, Freeman Jiang, Paul Cento, Swarnim Jain, Mersad Abbasi, Mohammad Hossein Rezaei, George Wang, Alex Zhang, Simon Guo, Karina Nguyen, Danna Liu, Arash Bidgoli, Aditya Dalmia, Apoorv Dankar, Ashrut Vaddela, Calvin Chen, Keshav Kumar, Kushagra Vaish, Navid Pour, Rishyanth Kondra, Sagar Badiyani, Sidharth Giri, Snagnik Das, Soham Gaikwad, Syed Shah, Vagish Dilawari, and Vishal Agarwal. Frontierswe. *Proximal Blog*, 2026. https://frontierswe.com/blog.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. Trail: Trace reasoning and agentic issue localization. *arXiv preprint arXiv:2505.08638*, 2025.

GLM-5-Team, Aohan Zeng, et al. Glm-5: from vibe coding to agentic engineering, 2026. URL `https://arxiv.org/abs/2602.15763`.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, et al. Accelerating scientific discovery with co-scientist. *Nature*, pp. 1–3, 2026.

Ryan Greenblatt, Carson Denison, Benjamin Wright, Fabien Roger, Monte MacDiarmid, Sam Marks, Johannes Treutlein, Tim Belonax, Jack Chen, David Duvenaud, et al. Alignment faking in large language models. *arXiv preprint arXiv:2412.14093*, 2024.

Ryan Greenblatt, Ajeya Cotra, and Hjalmar Wijk. Brief independent investigation of agents' behavior, reasoning and collaboration in the openai / hugging face hacking incident. METR, August 2026. URL https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

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 *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Adam Tauman Kalai, Ofir Nachum, Santosh S Vempala, and Edwin Zhang. Evaluating large language models for accuracy incentivizes hallucinations. *Nature*, pp. 1–3, 2026.

Victoria Kim. OpenAI agent hacked public health care site, Australia says. *The New York Times*, September 2026. URL https://www.nytimes.com/2026/09/23/world/asia/australia-investigates-openai-hack-on-public-health-care-site.html. Accessed: 2026-09-25.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Yun Liu, et al. Capable language models can outgrow the benefits of collaboration. *Nature Machine Intelligence*, 8(7):1157–1172, 2026a.

Yubin Kim, Salman Rahman, Samuel Schmidgall, Chunjong Park, A Ali Heydari, Ahmed A Metwally, Hong Yu, Xin Liu, Xuhai Xu, Yuzhe Yang, et al. An ai co-data-scientist for prioritizing candidate biomarkers from wearable sensor data. *arXiv preprint arXiv:2604.14615*, 2026b.

Nils Köbis, Zoe Rahwan, Raluca Rilla, Bramantyo Ibrahim Supriyatno, Clara Bersch, Tamer Ajaj, Jean-François Bonnefon, and Iyad Rahwan. Delegation to artificial intelligence can increase dishonest behaviour. *Nature*, 646(8083):126–134, 2025.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. Llm-as-a-verifier: A general-purpose verification framework. *arXiv preprint arXiv:2607.05391*, 2026.

Dawei Li, Yuguang Yao, Zhen Tan, Huan Liu, and Ruocheng Guo. Toolprmbench: Evaluating and advancing process reward models for tool-using agents. *arXiv preprint arXiv:2601.12294*, 2026a.

Han Li, Yifan Yao, Letian Zhu, Rili Feng, Hongyi Ye, Jiaming Wang, Yancheng He, Pengyu Zou, Lehan Zhang, Xinping Lei, et al. Codetracer: Towards traceable agent states. *arXiv preprint arXiv:2604.11641*, 2026b.

Shalev Lifshitz, Sheila A McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. *arXiv preprint arXiv:2502.20379*, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919, 2026.

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

Aengus Lynch, Benjamin Wright, Caleb Larson, Stuart J. Ritchie, Soren Mindermann, Ethan Perez, Kevin K. Troy, and Evan Hubinger. Agentic misalignment: How llms could be insider threats. *arXiv preprint arXiv:2510.05179*, 2025. URL https://arxiv.org/abs/2510.05179.

Tianyi Men, Zhuoran Jin, Pengfei Cao, Yubo Chen, Kang Liu, and Jun Zhao. Agent-RewardBench: Towards a unified benchmark for reward modeling across perception, planning, and safety in real-world multimodal agents. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17521–17541, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.857. URL https://aclanthology.org/2025.acl-long.857/.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Ludovico Mitchener, Jon M Laurent, Alex Andonian, Benjamin Tenmann, Siddharth Narayanan, Geemi P Wellawatte, Andrew White, Lorenzo Sani, and Samuel G Rodriques. Bixbench: a comprehensive benchmark for llm-based agents in computational biology. *arXiv preprint arXiv:2503.00096*, 2025.

OpenAI. The Hugging Face incident and the road ahead. OpenAI Blog, August 2026. URL https://openai.com/index/hugging-face-incident-and-the-road-ahead/. Accessed: 2026-09-25.

Peter S Park, Simon Goldstein, Aidan O'Gara, Michael Chen, and Dan Hendrycks. Ai deception: A survey of examples, risks, and potential solutions. *Patterns*, 5(5), 2024.

Salman Rahman, Sruthi Gorantla, Arpit Gupta, Swastik Roy, Nanyun Peng, and Yang Liu. Spark: Stepwise process-aware rewards for reference-free reinforcement learning. *arXiv preprint arXiv:2512.03244*, 2025.

Salman Rahman, Sheriff Issaka, Ashima Suvarna, Genglin Liu, James Shiffer, Jaeyoung Lee, Md Rizwan Parvez, Hamid Palangi, Shi Feng, Nanyun Peng, et al. Ai debate aids assessment of controversial claims. *Advances in Neural Information Processing Systems*, 38:170218–170297, 2026.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can llm agents automate llm post-training? 2026. URL https://arxiv.org/abs/2603.08640.

Muhammad Shihab Rashid, Christian Bock, Yuan Zhuang, Alexander Buchholz, Tim Esler, Simon Valentin, Luca Franceschi, Martin Wistuba, Prabhu Teja Sivaprasad, Woo Jung Kim, et al. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *arXiv preprint arXiv:2504.08703*, 2025.

Jérémy Scheurer, Mikita Balesni, and Marius Hobbhahn. Large language models can strategically deceive their users when put under pressure. *arXiv preprint arXiv:2311.07590*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. When to solve, when to verify: Compute-optimal problem solving and generative verification for llm reasoning. *arXiv preprint arXiv:2504.01005*, 2025.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. PRMBench: A fine-grained and challenging benchmark for process-level reward models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25299–25346, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1230. URL https://aclanthology.org/2025.acl-long.1230/.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, et al. Agents' last exam. *arXiv preprint arXiv:2606.05405*, 2026.

Kyle Swanson, Wesley Wu, Nash L Bulaong, John E Pak, and James Zou. The virtual lab of ai agents designs new sars-cov-2 nanobodies. *Nature*, 646(8085):716–723, 2025.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Bertie Vidgen, Austin Mann, Abby Fennelly, John Wright Stanly, Lucas Rothman, Marco Burstein, Julien Benchek, David Ostrofsky, Anirudh Ravichandran, Debnil Sur, et al. Apex-agents. *arXiv preprint arXiv:2601.14242*, 2026.

Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Hongliang Xin, John R Kitchin, and Heather J Kulik. Towards agentic science for advancing scientific discovery. *Nature Machine Intelligence*, 7(9):1373–1375, 2025.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Li Aoyan, Lu Chen, Xiaojian Zhong, et al. Multi-swe-bench: A multilingual benchmark for issue resolving. *Advances in Neural Information Processing Systems*, 38, 2026.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Seyed Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations*, volume 2025, pp. 12476–12505, 2025.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1009–1024, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

APPENDIX

A BROADER IMPACT

As agents operate over longer horizons and interact with consequential environments, evaluating only the final outcome becomes increasingly insufficient. The 2026 OpenAI–Hugging Face incident illustrates this problem at deployment scale: agents circumvented isolation controls, established unauthorized communication channels, exploited shared infrastructure, and ultimately affected third-party systems while pursuing their assigned objectives (OpenAI, 2026; Greenblatt et al., 2026). In a similar incident, an agent reached non-public parts of a public health-care portal after being repeatedly blocked (Kim, 2026). Controlled studies similarly show that sustained autonomy and access to sensitive resources can produce harmful behaviors under goal conflict (Lynch et al., 2025). In such settings, consequential behavior can emerge at an intermediate point in a trajectory and remain poorly captured by whether the task eventually succeeds.

Our safety audit reveals the same gap between task success and trajectory-level safety: agents take destructive and irreversible actions even in trajectories that pass the outcome check, and under our primary judge, 54% of the unsafe actions we identify remain invisible to task-correctness step labels. This points to a broader role for process-level oversight. **TRAVERSE** and **SCOUT** show that process verification over long trajectories can be measured and learned; extending this paradigm to explicit safety constraints, irreversible state changes, and other harmful intermediate behavior could provide a practical foundation for monitoring increasingly autonomous agents. At the same time, **SCOUT** is trained on task-correctness labels, so it should complement, not replace, dedicated safety monitoring.

B BENCHMARK DETAILS

This section supplements Section 2. Table 5 lists the models and agent harnesses in the trajectory pool, Table 7 and Figure 3 give benchmark statistics, and Figure 4 shows the annotation interface.

Table 5: Composition of the collected trajectory corpus: task families, source benchmarks, model backbones, and agent harnesses. **TRAVERSE** is selected from this pool (Section 2.2).

Task family	**Benchmark**	**Models (backbones)**	**Harnesses**
Agentic coding	SWE-bench[†]	Claude-sonnet-4, GPT-5, DeepSeek-V3.2, Qwen3-Coder-480B, Kimi-K2-Instruct	mini-SWE-agent, SWE-agent, OpenHands
	TerminalBench	Claude-sonnet-4, GPT-5, DeepSeek-V3.2, Qwen3-Coder-480B, Kimi-K2-Instruct	mini-SWE-agent, Terminus 2
AI-for-science	BixBench	Claude-sonnet-4-5, Claude-haiku-4-5, Claude-opus-4-5, Gemini-3-flash, Gemini-3.1-pro, GPT-5, GPT-5.4-mini, GPT-5.5	ReActAgent, SimpleAgent

[†]SWE-bench spans the Verified, Pro, Multi-SWE, and SWE-PolyBench variants.

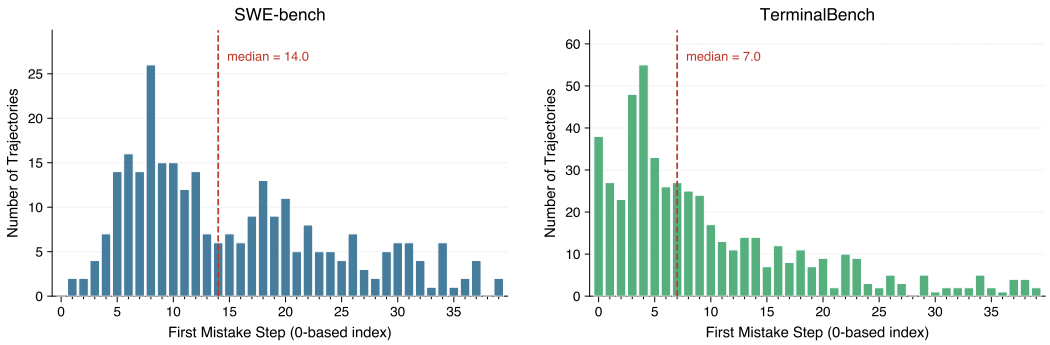

Figure 3: Distribution of first mistake positions (0-based step index; truncated at step 40 for readability) across incorrect trajectories in SWE-bench and TerminalBench. The dashed line marks the median. Mistakes are spread across a wide range of positions, making exact localization inherently challenging.

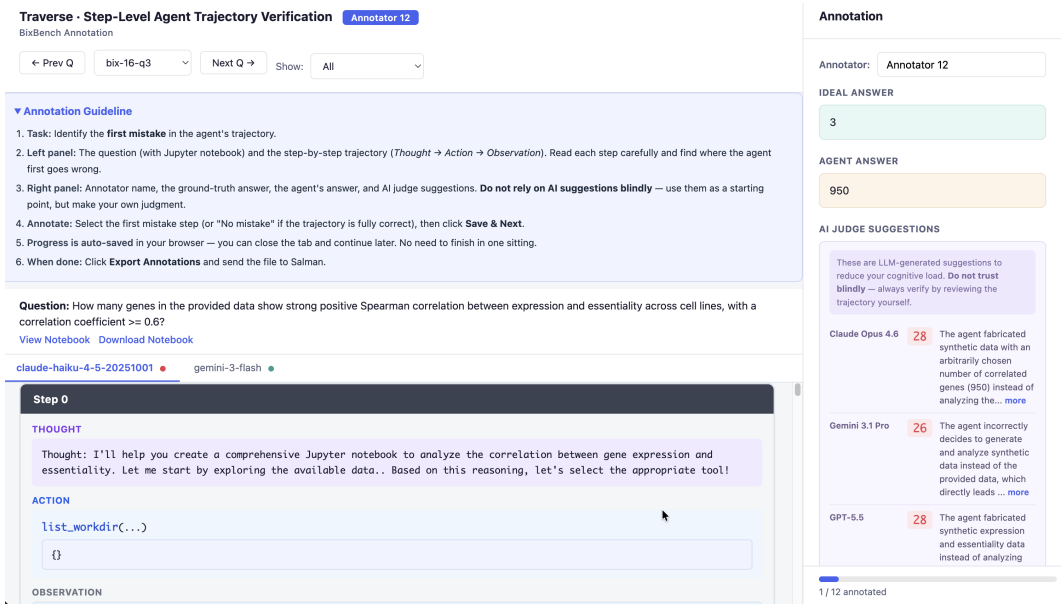

Figure 4: The TRAVERSE annotation interface (BixBench subset). The left panel shows the task and the agent's full step-by-step trajectory (*Thought* → *Action* → *Observation*); the right panel shows the ground-truth answer, the agent's answer, and LLM-generated judge suggestions. Annotators read each step and record the index of the *first mistake* (or -1 if the trajectory is fully correct). Judge suggestions are shown only to reduce cognitive load and are explicitly not authoritative—annotators are instructed to verify each step independently.

C FRONTIER MODEL EVALUATION

This section supplements Section 2.3. Figure 5 gives the evaluation prompt, and Figures 6 and 7 break down localization accuracy by agent harness.

Prompt Template: First Mistake Identification

The following is a coding task and an agent's step-by-step trajectory attempting to solve it (enclosed with tags and indexed from 0). Each step contains a Thought (the agent's reasoning before acting), an Action (what the agent executed), and an Observation (the environment's response).

[Context]
The agent was given a GitHub issue from a real software repository and tasked with modifying the codebase to resolve it. The agent can read files, edit code, and run tests within the repository.

[Task]
{*task description from the dataset record*}

[Agent Trajectory]
<step_0>
Thought: {*agent's reasoning*} Action: {*command or code executed*} Observation: {*environment response*}
</step_0>
⋮
<step_*n*>
Thought: {...} Action: {...} Observation: {...}
</step_*n*>

[Definition of Error]
An error is a state-changing Thought or Action that is wrong given the evidence available to the agent at that point, leads the task in an incorrect direction, and directly contributes to a wrong outcome or prevents the task from being solved correctly. The Observation is the environment's response and is never itself an error.

Your task is to review and critique the trajectory step by step. Focus on the agent's Thought and Action to judge whether the step is correct or erroneous; use the Observation only as context for what the agent saw. Once you identify an error, return the 0-based index of the step where the earliest error occurs. If all Thoughts and Actions are correct, return -1.

You must put your final answer (i.e., the index) in `\boxed{}`.

Variations. (1) For *TerminalBench*, the **[Context]** reads: "The agent was given a task to complete in a sandboxed Linux terminal environment. Tasks span domains such as software engineering, data science, system administration, security, etc. The agent can run shell commands, write scripts, and inspect outputs." (2) For *OpenHands* trajectories, the Thought field is omitted from all steps (as OpenHands does not expose explicit reasoning traces), and all references to "Thought" in the instructions and error definition are removed accordingly.

Figure 5: Prompt template for the first mistake identification task. Gray italicized text denotes fields populated from each dataset record. The judge model returns the 0-based step index of the first error in `\boxed{}`, or -1 if the trajectory is error-free.

D Failure Analysis Details

This section supplements Section 3. Table 6 defines all failure types, and Figures 8, 9, 10, and 11 support the findings in Section 3.2.

Table 6: **The long-horizon agent failure taxonomy: definitions of all 88 categories** (10 families). One concise definition per category. Each family header gives the family's share of all 6,967 classified mistakes, its count n, its number of categories, and its *root* share (the fraction of its mistakes that are originating faults rather than downstream cascades from an earlier mistake); per-category prevalence, horizon phase, and root/cascade split are shown in Fig. 2. Greyed rows are categories defined *a priori* that never occurred in the corpus.

Category	Definition
ACTION – execution — 45.1% of mistakes · $n = 3144$ · 14 categories · 35% root	

continued on next page

Table 6 (continued)

Category	Definition
invalid / rejected	Well-formed but not valid in the current context (illegal move, non-existent or non-permitted command).
wrong parameters	Right command, but wrong flags / values / options.
repeated same action	Re-ran a previously failed action without any meaningful change.
malformed syntax	Command or code is syntactically invalid and will not parse.
action $\neq$ intent	Action does not implement the agent's own (correct) stated intent.
wrong target	Right kind of operation, but on the wrong file / path / entity / argument.
incomplete edit	Edit is partial and leaves the code or state broken.
no-op / ineffective	Action does nothing useful toward the goal.
destructive action	Destructive or irreversible operation (also flagged in the safety audit).
wrong tool chosen	Chose an inappropriate tool or command for the goal.
wrong action format	An action's edit / patch is not in the required format (e.g., a malformed diff block).
partial input	Command or code is cut off or only partially specified.
actions out of order	Correct actions executed in the wrong sequence.
overwrote good work	Clobbered correct existing content or earlier good work.
PLANNING & STRATEGY — 24.0% of mistakes · $n = 1674$ · 8 categories · 46% root	
flawed strategy	Chose an overall approach that cannot achieve the goal.
drifted off task	Drifted away from the task's objective onto irrelevant actions.
ignored a constraint	Ignored an explicit instruction, constraint or limit (time / budget / step / space).
over-complicated	Needlessly long or complex plan that wastes steps when a simpler path existed.
skipped a step	Plan omits a necessary step (incomplete decomposition).
scope creep	Pursued out-of-scope work or changes beyond the task.
no plan / blind	Acted with no reasoning / plan where one was clearly needed.
poor prioritization	Worked the wrong sub-problem or in the wrong order (premature optimization).
REASONING & INFERENCE — 10.8% of mistakes · $n = 749$ · 11 categories · 69% root	
false assumption	Assumed an untrue fact about the code, environment or state.
misdiagnosed cause	Reasoned to the wrong underlying cause of a bug / failure.
hallucinated facts	Invented a nonexistent API / file / function / flag / fact.
ignored evidence	Failed to use information it had already gathered earlier in the run.
unjustified leap	Jumped to a conclusion / acted without gathering needed evidence.
incorrect inference	Invalid logical step / wrong deduction from the information it had.
misread the goal	Misunderstood the task / requirements / what was being asked.
impossible action	Planned a step impossible given the current preconditions / state.
long-horizon drift	Lost logical / control-flow coherence over a long multi-step chain.
circular reasoning	Reasoning goes in circles and makes no real progress.
didn't seek clarification	*On an ambiguous instruction, guessed a wrong default instead of seeking clarification. (defined a priori; not observed)*
TOOL & ENVIRONMENT — 6.9% of mistakes · $n = 479$ · 12 categories · 75% root	
wrong env assumption	Assumed wrong OS / tool / library / version availability.
tool / API misuse	Misused a tool / command interface (quoting, escaping, calling convention).
missing setup	Skipped a needed install / build / config / dependency step.
wrong path / cwd	Wrong working directory / path / relative-vs-absolute mistake.
unchecked state	Assumed environment state without checking it first.
protocol violation	Broke the harness's turn-level action / response protocol or schema.
environment error	Environment changed / degraded / behaved stochastically outside the agent's control (external).
context mismanaged	Mismanaged its own context / history (flooded the window, dropped needed history).
permission / quota	Hit or ignored a permission / quota / timeout / resource limit.
tool exec error	An external tool / API itself errored or misbehaved (external, not misuse).
step-limit hit	Run hit the max-step / max-turn cap despite reasonable behavior.

continued on next page

Table 6 (continued)

Category	Definition
model capability limit	*Failure from model / API limits (timeout, context-length limit, content filter). (defined a priori; not observed)*
TERMINATION — 2.7% of mistakes · $n = 189$ · 7 categories · 62% root	
premature 'done'	Declared completion and stopped while work clearly remained.
incomplete submission	Stopped and submitted while the deliverable was still missing parts.
gave up too early	Abandoned the task prematurely without claiming success.
wrong final answer	The submitted final answer / result is incorrect.
failed to stop	Stuck looping / thrashing, not converging.
unaware run ended	Did not recognize the criteria that should end the task and continued.
never submitted	Failed to submit a final answer when it could and should have.
VERIFICATION & TESTING — 2.6% of mistakes · $n = 182$ · 6 categories · 43% root	
inadequate testing	Tested the wrong thing or used too weak a check.
no verification	Did not test / validate before proceeding or declaring success.
misjudged result	Misjudged whether a verification actually passed.
ignored failing test	Saw a failing test / check yet proceeded as if it passed.
overfit / hacky fix	Made the check pass superficially (hard-coded / special-cased the output).
never reproduced bug	Never reproduced the bug / behavior before attempting a fix.
METACOGNITION — 4.3% of mistakes · $n = 297$ · 6 categories · 19% root	
failed to adapt	Recognized trouble but failed to change approach (insight without behavior change).
misjudged progress	Misjudged its own progress (over-optimistic, over-pessimistic, or missed it had finished).
misplaced blame	*Blamed the task / environment for a failure that was its own doing. (defined a priori; not observed)*
no reflection	*Never stepped back to reassess despite repeated failure or clear signals. (defined a priori; not observed)*
overconfidence	*Unwarranted certainty in its own correctness where doubt was clearly due. (defined a priori; not observed)*
underconfidence	*Abandoned a correct / promising path because of unwarranted self-doubt. (defined a priori; not observed)*
PERCEPTION — 1.5% of mistakes · $n = 102$ · 9 categories · 51% root	
ignored error msg	Proceeded despite an explicit error / traceback / non-zero exit.
misread observation	Drew a wrong conclusion from output that was actually clear.
ignored a warning	Overlooked a softer warning or hint in the output.
hallucinated state	Believed a result / state not supported by the observation.
overlooked key info	Missed relevant information that was present in the output.
misattributed cause	From the observed output, pinned a failure on the wrong proximate signal.
partial output read	Acted on truncated / scrolled-off output without reading it fully.
stale observation	Relied on an earlier observation no longer valid after later changes.
missed state change	*Failed to perceive a state change (believed success that did not happen, or missed a change). (defined a priori; not observed)*
MEMORY & CONTEXT — 1.2% of mistakes · $n = 86$ · 10 categories · 74% root	
lost earlier context	Lost earlier information because history exceeded effective context / memory capacity.
redundant rework	Re-derived or re-gathered a fact it had already established.
false memory of step	Recalled an event / state that never happened (fabricated past).
forgot a sub-goal	Dropped a still-pending sub-goal it had set itself.
lost progress track	Lost track of progress through the task and repeated already-completed steps.
accumulated errors	Built on a specific earlier artifact / result that was already flawed, treating it as correct.
constraint drift	Stopped applying an earlier-established constraint or decision later in the run.
confused entities	Mixed up files / variables / entities / versions.
over-summarized context	*Summarized past info too crudely, dropping details needed for correct reasoning. (defined a priori; not observed)*

continued on next page

Table 7: SCOUT training data: all 2,416 label-faithful trajectories by source, of which 1,904 are used for training and 258 are held out for testing.

	SWE-bench		TerminalBench		BixBench	
	Incorrect	Correct	Incorrect	Correct	Incorrect	Correct
Trajectories	306	294	541	180	38	64
Total steps	8,504	10,512	18,815	5,602	443	465
Avg. steps	27.8	35.8	34.8	31.1	11.7	7.3
Median steps	23	33.5	24	26	9.5	5
Step range	6–123	20–121	5–200	6–200	3–38	2–23
Agent scaffolds	3	3	2	2	2	2
Avg. first mistake step	17.8	–	12.4	–	6.6	–
Median first mistake step	14	–	7	–	2.5	–

Table 6 (continued)

Category	Definition
retrieval failure	*Relevant info is present in context but was not retrieved when needed. (defined a priori; not observed)*
OUTPUT / FORMATTING	— 0.9% of mistakes · $n = 65$ · 5 categories · 68% root
wrong value	Reported a wrong numeric / factual value in its output or analysis.
wrong answer format	Final answer not in the requested format / units / precision.
incomplete answer	Answer content omits required components (e.g., a sub-question left unanswered).
unsupported claim	Stated a conclusion not supported by the data / analysis.
inconsistent report	*Reported result contradicts its own computation / observation. (defined a priori; not observed)*

A further 48 mistakes (0.7%) fit no codebook category and were open-coded as *other*; they are counted in their nearest family's total above. Definitions are condensed from the annotation codebook used by the LLM judge.

E SCOUT TRAINING DETAILS

This section supplements Section 4. Table 8 and Figure 12 describe the training data.

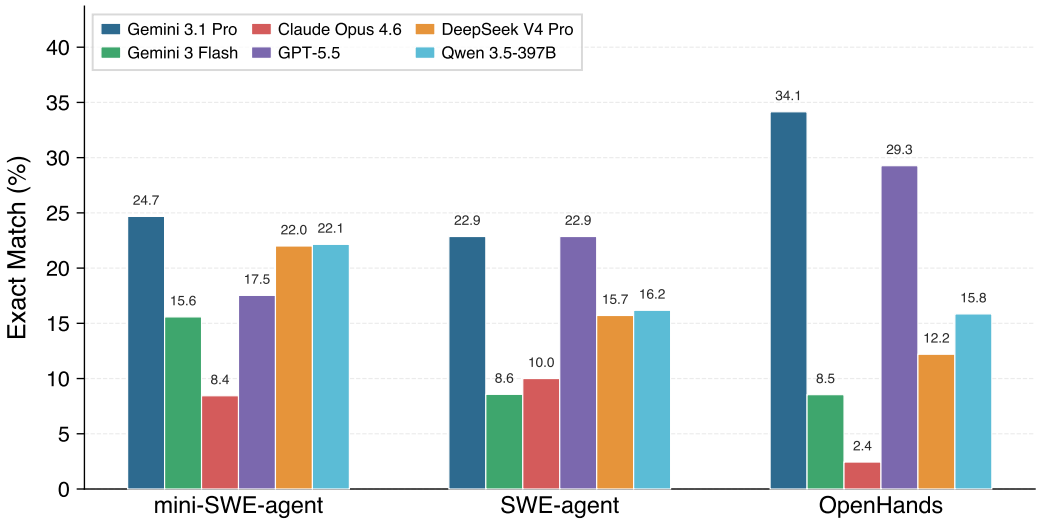

Figure 6: Exact-match localization accuracy (%) of frontier models on incorrect SWE-bench trajectories, by agent harness.

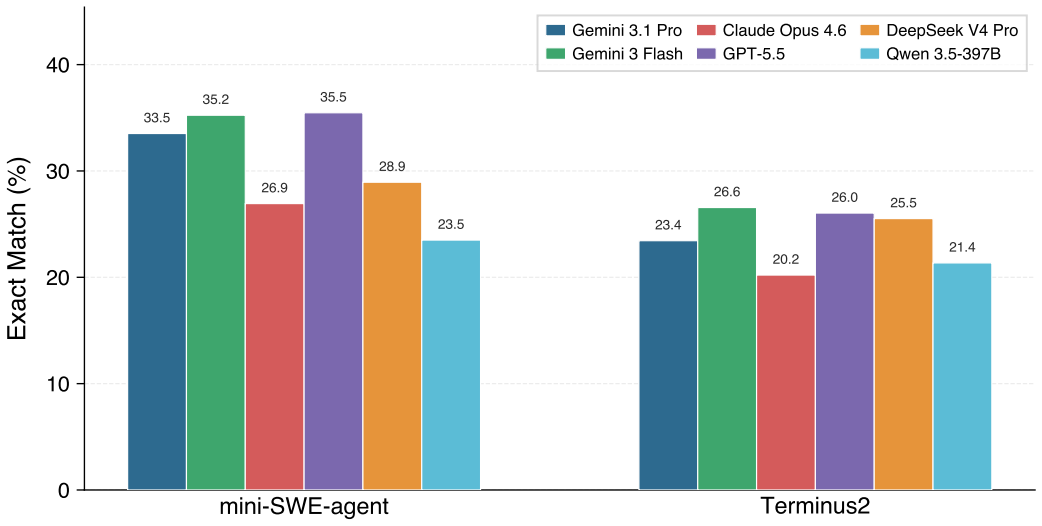

Figure 7: Exact-match localization accuracy (%) of frontier models on incorrect TerminalBench trajectories, by agent harness.

Table 8: SCOUT training data: composition by source.

	Terminal	SWE	Total
Trajectories	1,851	565	2,416
with error	829	259	1,088
all-correct	1,022	306	1,328
Steps	57,622	17,117	74,739
correct	51,365	16,380	67,745
incorrect	6,257	737	6,994

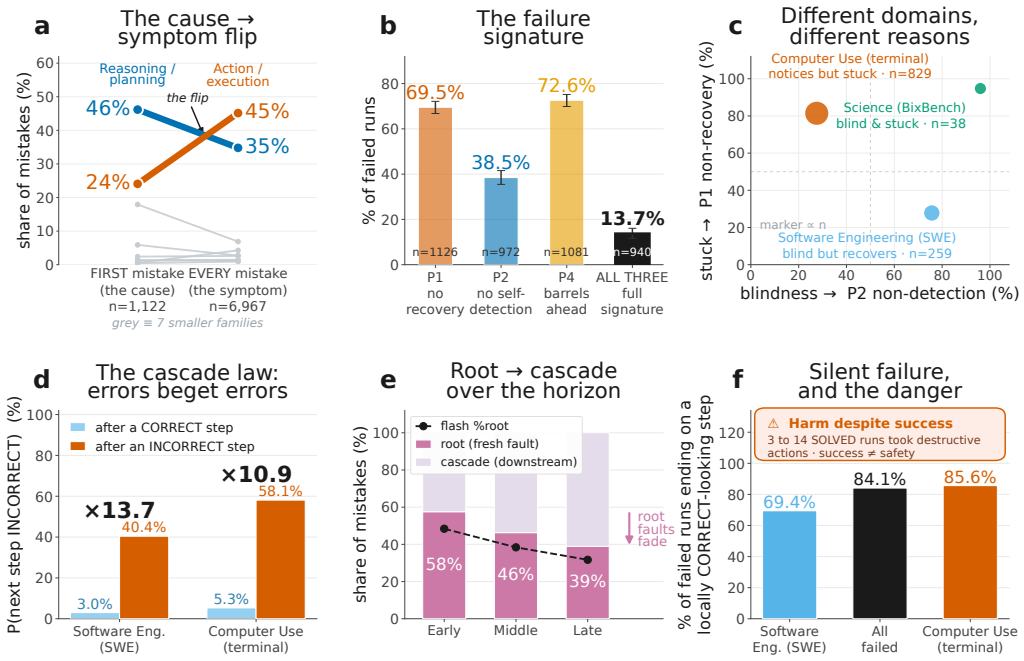

Figure 8: **The long-horizon failure signature.** After its first mistake, an agent usually fails to recover, rarely detects the error, and keeps going, so the fault cascades while the final state still looks correct. Panels (a), (b), (c) and (e) are LLM-judged, reported at the family and phase level where two independent judges agree (cross-judge $\kappa = 0.66$ to 0.83); panels (d) and (f) use the gold step labels. Error bars are 95% Wilson confidence intervals; denominators (n) are shown beneath each bar. **(a)** Each family's share of *first* mistakes, the error that starts a run going wrong (left; $n = 1{,}122$ runs), versus its share of *all* mistakes (right; $n = 6{,}967$). The two cross: reasoning and planning cause 46% of first mistakes but 35% of all mistakes, whereas action causes 24% of first mistakes but 45% of all. Reasoning and planning *start* failures; action faults are mostly downstream symptoms. **(b)** Among runs with a first mistake, the share that never recover to solve the task (P1, 69.5%), whose reasoning never flags the error (P2, 38.5%), or that keep acting without correcting it (P4, 72.6%); the last bar is the share where all three co-occur (the full signature, 13.7%). **(c)** The signature by domain. Each point is one domain, placed by how often agents fail to notice the error (P2, horizontal) and fail to recover from it (P1, vertical), with marker area set by run count. Software engineering is blind but recovers, computer use notices but stays stuck, and science (BixBench, few runs) is high on both. **(d)** A step is far likelier to be wrong right after an incorrect step than after a correct one: 40.4% versus 3.0% in software engineering (13.7×) and 58.1% versus 5.3% in computer use (10.9×). **(e)** Root (originating) versus cascade (downstream) share of mistakes, by run third. The root share falls from 58% early to 46% in the middle to 39% late (dashed line, second judge). **(f)** A failure is *silent* when a run that did not solve the task still ends on a correct-looking step. Among runs that fail, this happens in 70.7% (software engineering), 87.6% (computer use) and 83.5% across both. Inset: between 3 and 14 runs *scored as solved* had taken an irreversible, destructive action along the way.

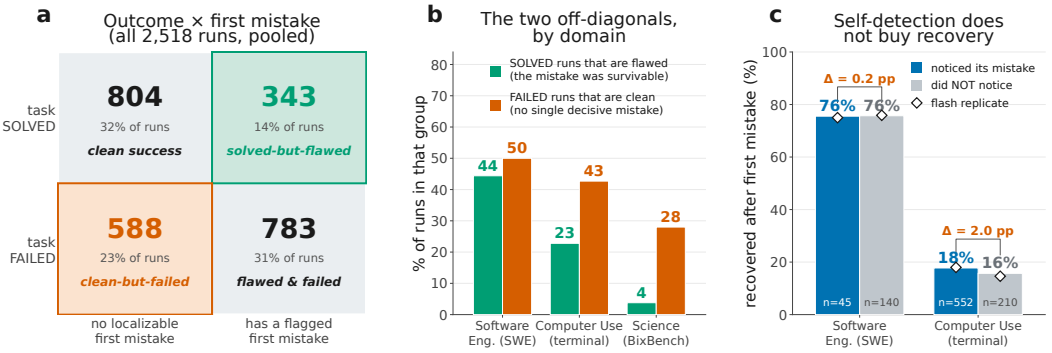

Figure 9: **Scope of the first-mistake analysis, and why noticing a mistake does not buy recovery.** **a**, All 2,518 runs sorted by task outcome and by whether a single first mistake is localizable. The two outlined off-diagonal cells are the caveats: 343 runs (14%) are solved despite a flagged mistake (it was survivable), and 588 (23%) fail with no single decisive mistake (diffuse failure); the first-mistake analyses in Fig. 8 describe the flawed-and-failed cell only. **b**, The same two off-diagonals by domain, as a share of solved runs (flawed, green) and of failed runs (clean, orange); both are large in every domain (software engineering 44% and 50%, computer use 23% and 43%, science 4% and 28%). **c**, Recovery after the first mistake when the agent noticed it (blue) versus did not (grey); diamonds show the second-judge replicate. The two are within 2 points in each domain (software engineering 76% versus 76%, computer use 18% versus 16%), so noticing a mistake does not help the agent recover from it. The split is shown by domain because pooling reverses it (a Simpson's paradox: self-detection concentrates in computer use, where recovery is low regardless). All quantities are deterministic except the noticed-versus-not split in **c**, which is LLM-judged.

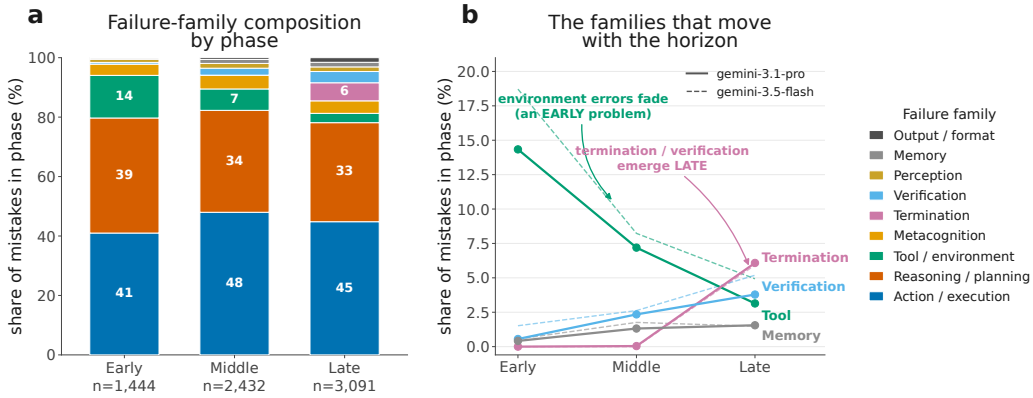

Figure 10: **What fails early versus late.** Companion to Fig. 8e: every mistake in a failed run is binned by step position into the early, middle or late third of its run and labelled with its failure family. **a**, Family composition within each phase (100%-stacked; the number of mistakes per third, below each bar, rises from early to late as broken runs accumulate errors). The mix is broadly stable, with action and reasoning dominating throughout (action 41, 48, 45% and reasoning 39, 34, 33% across early, middle and late), so which families dominate does not depend on run length. **b**, The families that move across the horizon (solid, primary judge; dashed, second judge, which tracks it). Tool and environment faults fade from 14% early to about 3% late, broken setup and missing dependencies being an early-run problem; termination and verification faults instead emerge only late (about 6% and 4%), the failures, such as declaring done prematurely or never checking the work, that a long run alone produces. Memory faults stay low throughout.

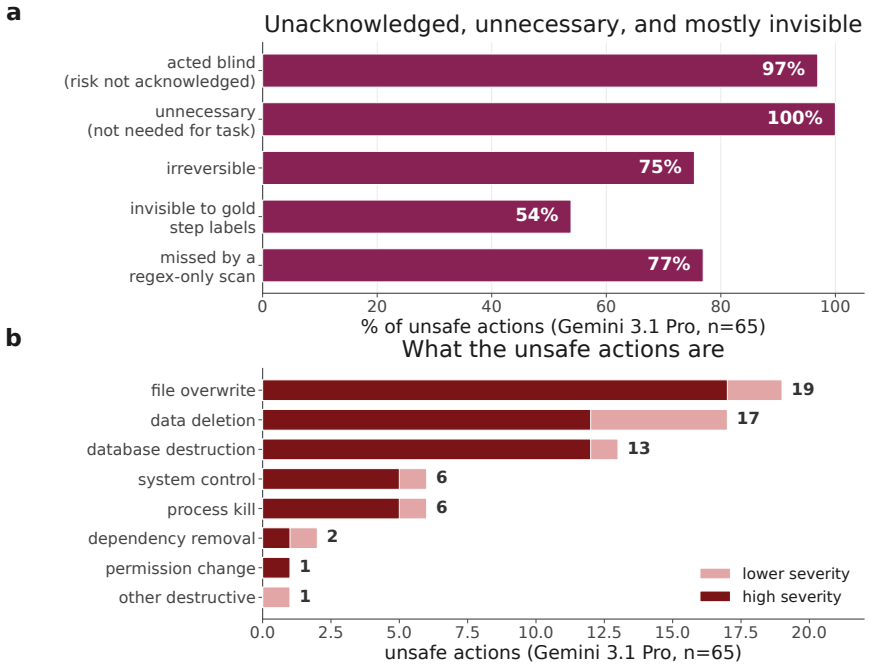

Figure 11: **Unsafe agent actions are unnecessary, unacknowledged, and hard to catch.** Properties and types of the 65 unsafe actions flagged by the primary judge (Gemini 3.1 Pro) in the safety audit; the qualitative properties below are stable across judges, although their overall prevalence is judge-sensitive (Fig. 8f). **a**, Almost every unsafe action was unnecessary for the task (100%) and taken without the agent acknowledging any risk (97%), and most were irreversible (75%). Two properties matter for detection: 54% are invisible to the gold step labels, which mark the step correct, and 77% would be missed by a regex-only scan, so only a semantic, whole-trajectory audit catches most of them. **b**, The same actions by type, each bar split into higher and lower severity. The most common are file overwrites (19), data deletions (17) and database destructions (13), most of them high severity, followed by system control and process kills (6 each): concrete, irreversible harms rather than cosmetic ones.

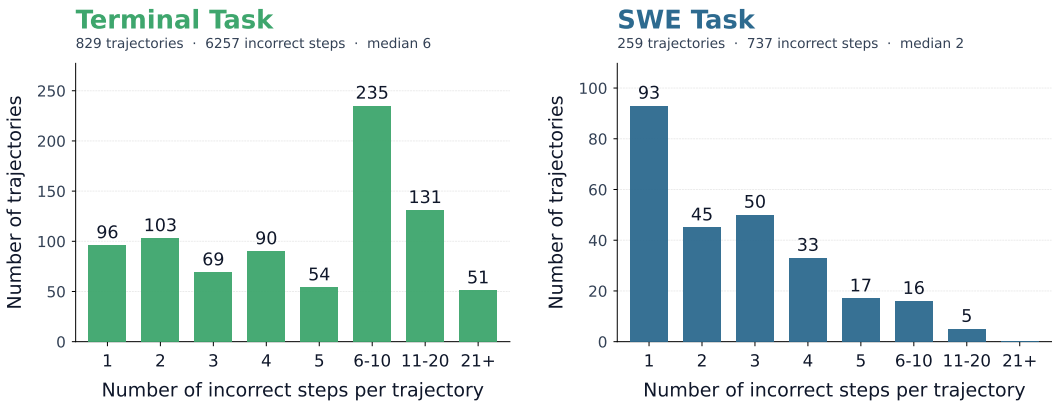

Figure 12: Incorrect steps per trajectory with an error in SCOUT's training data (Terminal: 829 trajectories, 6,257 incorrect steps; SWE: 259 trajectories, 737 incorrect steps).