# TRACEGRAPH: Shared Decision Landscapes for Diagnosing and Improving Agent Trajectories

Anonymous ACL submission


## Abstract

Agent benchmarks usually reduce an entire interaction trajectory to a pass rate or reward score. Such scalars rank systems, but they obscure what process capabilities a benchmark exercises and where a model fails. We introduce TRACEGRAPH, a graph-based framework that turns released multi-model agent trajectories into *shared decision landscapes*. For each task, TRACEGRAPH builds a graph over observable action–observation states from pooled rollouts before model identity is introduced. It then overlays outcome-informed productive cores and trap regions, and summarizes each rollout with three events: Access, Trap exposure, and Repair. Across trajectories spanning five benchmark splits, TRACEGRAPH profiles reveal navigation differences hidden by aggregate scores and show that splits differ in whether they reward avoiding traps or recovering from them. The same TRACEGRAPH landscape also motivates a trap-aware recovery pipeline for SWE-BENCH: a runtime detector fires on states matching historical trap regions, then lightweight continuation policies are evaluated from the same prefix. On fired states, the best pooled single-factor policy raises official resolved rate from 40.4% to 43.5% on the per-provider fired subset and from 41.0% to 44.8% on common-fired instances, with provider-specific active components. Overall, TRACEGRAPH provides a process vocabulary for asking what agent benchmarks test, where models diverge on a shared landscape, and how failure regions can guide downstream improvement.


## 1 Introduction

Language models first made intermediate computation visible through chain-of-thought prompting and self-consistency sampling (Wei et al., 2022; Wang et al., 2023). Agent benchmarks extend this trace to interleaved thoughts, tool calls, and environment feedback, as in ReAct and Toolformer-style tool use (Yao et al., 2023b; Schick et al., 2023). Yet most reports still collapse each rollout to one scalar: a pass/fail label, reward score, or leaderboard average. Such scores rank systems, but they do not reveal what process a benchmark rewards.

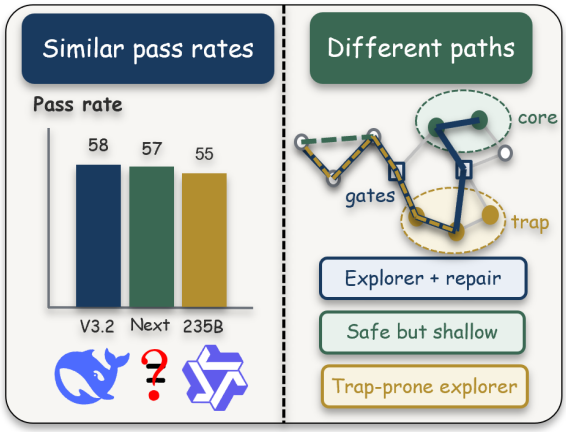


Figure 1: Pass rates can hide process diversity. **Left**: three models score within 3 points on the same benchmark. **Right**: on a shared decision landscape built from pooled rollouts, they follow distinct paths: broad exploration with repair, shallow trap avoidance, or high-access navigation with more trap exposure.

The missing information is often visible in the trajectory itself. On SWE-BENCH (Jimenez et al., 2024), a failed rollout may never inspect the right file, or it may localize the bug and then fail to repair a bad patch. On tool-use benchmarks, one failure may violate a state constraint while another follows a slow but recoverable path. These are different process failures, not merely different degrees of difficulty.

We introduce TRACEGRAPH to recover this process structure from released multi-model trajectory datasets. For each task, TRACEGRAPH pools all model rollouts and builds a shared graph over observable action–observation signatures. Model identity is excluded from graph construction and

added only afterward, so each model becomes traffic on the same task-specific terrain. TRACEGRAPH then overlays outcome-informed productive cores and trap regions, and summarizes each rollout with three events: reaching a core (*Access*), entering a trap (*Trap exposure*), and returning from a trap to a core (*Repair*). The name reflects the workflow: traces become graphs, and models are compared by how they move through the resulting landscape. These roles use terminal outcomes by design, so TRACEGRAPH is descriptive rather than a blind success predictor: given a common map, what process does each model supply, and what process does each benchmark demand?

A shared landscape is useful not only for retrospective diagnosis, but also for deciding when a live agent trajectory has entered a historically risky region. If trap regions correspond to repeatable trajectory states, an online detector built from those states should identify moments where a small recovery policy can help. We therefore instantiate a TRACEGRAPH-guided trap-aware prefix-fork pipeline on the benchmark with the clearest positive Repair demand. An agent proceeds until a trap detector fires; matched continuations then resume from the same state while varying a conservative diagnosis note and a temperature bump. The goal is not to claim a universal recovery mechanism, but to show that graph-derived traps can guide a downstream policy that improves task resolution.

**Our contributions can be summarized as follows.**

1. **TRACEGRAPH for shared decision landscapes.** We construct model-agnostic task graphs from pooled rollouts, then overlay outcome-informed core and trap regions after graph construction.

2. **A compact process profile.** Across a five-split trajectory release spanning 7,329 records, 427 tasks, and five models, Access, Trap exposure, and Repair separate model navigation styles and benchmark demands that aggregate scores collapse.

3. **A TRACEGRAPH-guided recovery pipeline.** Because SWE-BENCH shows the clearest positive Repair demand, we turn graph-derived traps into runtime triggers and test two single-factor recovery policies on all 500 SWE-BENCH Verified instances. The best pooled policy improves resolved rate by $+3.1$ points ($p$=0.016) on the per-provider fired subset and by $+3.8$ points ($p$=0.045) on common-fired instances, while the active component differs by provider.

## 2 Related Work

**Outcome-centered evaluation.** Frameworks such as HELM and the LM Evaluation Harness emphasize coverage, transparency, and reproducibility across many tasks and metrics (Liang et al., 2023; Gao et al., 2023). Agent benchmarks bring this apparatus to interactive settings, including software repair, tool use, and general agent environments (Jimenez et al., 2024; Liu et al., 2024; Yao et al., 2025). They are indispensable for tracking progress, but their headline artifacts usually summarize final success. We reuse the released logs for a complementary purpose: describing the process demands that separate high- and low-outcome trajectories.

**Process traces, failure patterns, and intervention.** Chain-of-thought and self-consistency showed that sampled traces expose alternative reasoning paths before answers are aggregated (Wei et al., 2022; Wang et al., 2023). ReAct, Toolformer, WebShop, and ALFWorld extended this process view to tool use and environment interaction (Yao et al., 2023b; Schick et al., 2023; Yao et al., 2022; Shridhar et al., 2021). Recent intervention work also treats long-trace failures as actionable: Trap-Aware Adaptive Restart identifies "thinking traps" and restarts from earlier reasoning prefixes (Chen et al., 2026b). We study an analogous but distinct setting: tool-using software agents, where trap states are observable action–observation signatures and continuations resume from a live benchmark state.

**Graphs as inference programs and measurement objects.** Tree-of-Thoughts and Graph-of-Thoughts use graph structure during inference to guide search (Yao et al., 2023a; Besta et al., 2024). Other work uses graphs after generation to cluster chain-of-thought units, analyze trace topology, and model software-agent processes over exploration, localization, patching, validation, and loops (Xiong et al., 2025; Tan et al., 2025; Liu et al., 2026). A related methodological precedent is SLICEGRAPH, a post-hoc atlas of same-answer chain-of-thought route isomerism over hidden activation slices (Chen et al., 2026a). TRACEGRAPH adopts this graph-as-measurement perspective, but shifts the ob-

ject to observable agent states, multi-model task graphs, benchmark-level process demand, and trap-triggered recovery on SWE-BENCH.

## 3 TRACEGRAPH: Building Shared Decision Landscapes

TRACEGRAPH takes as input a set of agent rollouts and returns a shared decision landscape with three readable rollout events. For a task $t$, let $\mathcal{R}_t = \bigcup_m \mathcal{R}_{t,m}$ be all rollouts from all available models. Because the released data contain only a few passes per model per task, TRACEGRAPH builds one landscape from $\mathcal{R}_t$ and compares models only after that landscape is fixed. This separation is central to the framework: graph construction defines the terrain, while model and benchmark analyses are overlays on that terrain.

### 3.1 Stage 1: Building the Shared Landscape

Each action–observation step $i$ is encoded as a sparse set of observable keys $K_i$: tool and action type, command class, observation pattern, file cue, temporal phase, and a pre-specified search-interaction family for SEARCH. Pairwise similarity is IDF-weighted Jaccard overlap over these key sets,

$$\mathrm{sim}(i,j) = \frac{\sum_{k \in K_i \cap K_j} \mathrm{idf}(k)}{\sum_{k \in K_i \cup K_j} \mathrm{idf}(k)}, \qquad (1)$$

so rare and more informative keys like a specific traceback pattern at a specific file weigh more than generic ones like a bare "read" action. We convert this to a distance $d(i,j) = 1 - \mathrm{sim}(i,j)$; among mutual-$k$NN pairs, the graph edge weight is $w_{ij} = \exp(-d(i,j)/\sigma)$ with fixed scale $\sigma$. For each task we build this mutual-$k$NN graph over all pooled steps, then decompose it into biconnected components (BCCs) and articulation points. The BCC decomposition is the structural foundation of the landscape: each BCC groups steps that share dense mutual similarity, and the *articulation points* (or *gates*) between BCCs mark strategy or phase transitions—places where a rollout can branch into qualitatively different continuations. A separate annotation pass confirms that 82 of 100 graph-identified articulation crossings correspond to a human-judged strategy or phase change (4.6× enrichment over random consecutive steps; see §4.2). Model identity is never part of the signature, distance, or graph construction. The exact key alphabet and fixed graph hyperparameters are in Appendix C.1 and Appendix 3.

### 3.2 Stage 2: Outcome Overlays

On top of the fixed BCC structure, we add two outcome-informed overlays. Let $G_B = (\mathcal{B}_t, E_B)$ be the block quotient graph, whose nodes are BCC blocks. For a block $b$, let $\mathcal{R}_b = \{r : b \in V_r\}$ be the rollouts that visit it and let $y_r \in [0,1]$ be the terminal outcome (binary for all benchmarks except the max-normalized MCPBENCH reward). We assign each block a Laplace-smoothed excess reward,

$$\hat{\mu}_b = \frac{\sum_{r \in \mathcal{R}_b} y_r + \frac{1}{2}}{|\mathcal{R}_b| + 1}, \qquad s_b = \hat{\mu}_b - \bar{y}_t, \qquad (2)$$

where $\bar{y}_t$ is the mean outcome over rollouts of task $t$. To avoid treating each block independently, we diffuse this centered reward over the quotient graph:

$$v^{(\ell+1)} = \alpha P_B^\top v^{(\ell)} + (1-\alpha)s, \qquad (3)$$

where $P_B$ is the row-normalized weighted adjacency matrix of $G_B$ and $v^{(0)} = s$. After fixed-step diffusion and $L_\infty$ normalization, the productive core and trap masks are the high positive and low negative quantiles,

$$C_t = \{b : v_b > 0 \ \wedge \ v_b \geq Q^+_{.75}\},$$
$$B_t = \{b : v_b < 0 \ \wedge \ v_b \leq Q^-_{.25}\}, \qquad (4)$$

where $Q^+_{.75}$ is the 75th percentile among positive field values and $Q^-_{.25}$ is the 25th percentile among negative field values. These overlays are descriptive: they locate where high- and low-outcome traffic concentrates, rather than predicting outcomes for unseen runs. Thus $C_t$ and $B_t$ are not semantic labels like "good action" or "bad action"; a trap block can still contain exploratory traffic (see §4.2).

### 3.3 Stage 3: Rollout Events

The final TRACEGRAPH representation uses only three rollout events, all defined on core/trap overlays. For a rollout $r$, let $V_r$ be its visited block set and $b_{r,1:T_r}$ its compact block sequence. *Access* asks whether $V_r$ intersects $C_t$. *Trap exposure* asks whether $V_r$ intersects $B_t$. *Repair* asks whether the sequence visits a trap and later reaches a core.

$$A_r = \mathbf{1}[V_r \cap C_t \neq \emptyset],$$
$$E_r = \mathbf{1}[V_r \cap B_t \neq \emptyset],$$
$$R_r = \mathbf{1}[\exists s < u : b_{r,s} \in B_t, \ b_{r,u} \in C_t]. \qquad (5)$$

For binary-reward benchmarks, the rollout outcome is the recorded binary success/reward field; for MCPBENCH, it is the per-task max-normalized continuous reward. Appendix B.2 gives the full definitions.

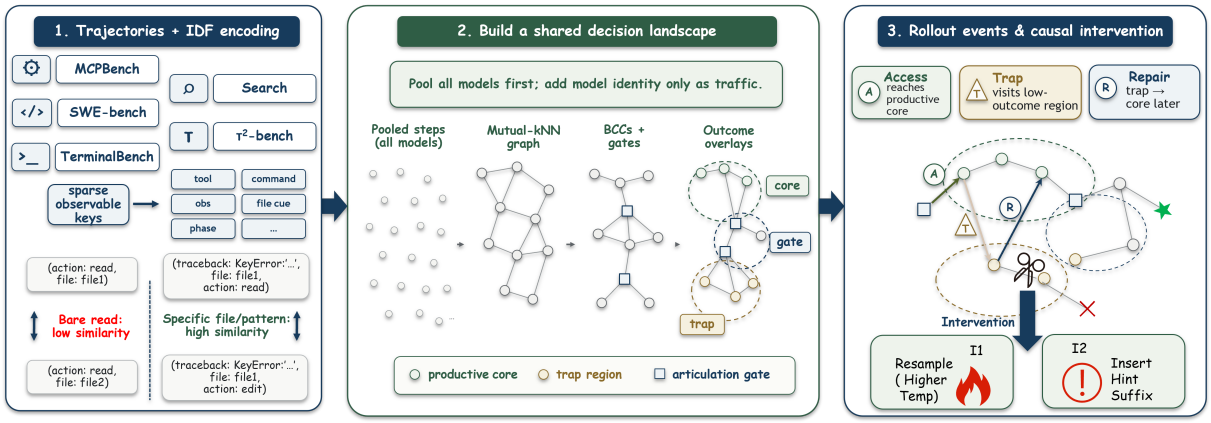


Figure 2: Overview of TRACEGRAPH. **Stage 1**: action–observation steps from five benchmark splits are encoded into sparse observable keys and compared via IDF-weighted Jaccard similarity. **Stage 2**: all model rollouts are pooled into a mutual-$k$NN graph, decomposed into BCCs and articulation gates, then annotated with outcome-informed overlays (productive cores, trap regions). **Stage 3**: three rollout events—Access, Trap, and Repair—are read from each model's path. The same trap overlay later seeds a runtime trigger for a lightweight SWE-BENCH recovery pipeline.

### 3.4 Supply and Demand Profiles

TRACEGRAPH turns the three events into two descriptive summaries. A model's *supply* is the task-centered mean of its rollout events,

$$S_{m,a} = \mathbb{E}_t\left[a_{m,t} - \mathbb{E}_{m' \in M_t} a_{m',t}\right], \qquad (6)$$

where $a \in \{A, E, R\}$ and $a_{m,t}$ is the average event value for model $m$ on task $t$. A benchmark's *demand* is the reward-weighted contrast between high- and low-outcome traffic,

$$D_{b,a} = \frac{\sum_{r \in b} y_r a_r}{\sum_{r \in b} y_r} - \frac{\sum_{r \in b}(1 - y_r) a_r}{\sum_{r \in b}(1 - y_r)}. \qquad (7)$$

For binary rewards this is simply the mean event value among resolved rollouts minus the mean event value among failed rollouts. For MCP-BENCH it is the continuous analogue using normalized reward. We use these profiles for interpretation, not as a score-prediction model.

## 4 Graph Analysis with TRACEGRAPH

### 4.1 Data and Setup

| Split | Records | Models | Process focus |
|---|---|---|---|
| MCPBENCH | 899 | 5 | MCP tool orchestration |
| SEARCH | 3,270 | 5 | Web search and answer synthesis |
| SWE-BENCH | 747 | 5 | Repository editing and tests |
| $\tau^2$-BENCH | 984 | 5 | API use under policies |
| TERMINALBENCH | 1,429 | 5 | Command-line tasks |

Table 1: Trajectory records in the five retained release splits before graph-validity filtering. Each task is attempted by five models with up to roughly four passes per model, with some missing or filtered attempts.

We analyze the gated cx-cmu agent trajectory release on Hugging Face (Chenyan Xiong Research Group at CMU, 2026). The analyzed splits include tool orchestration, web-search answer synthesis, software repair, API-policy interaction, and command-line tasks from MCPBENCH (Wang et al., 2025), the SEARCH split of the cx-cmu trajectory release (Chenyan Xiong Research Group at CMU, 2026), SWE-BENCH (Jimenez et al., 2024), $\tau^2$-BENCH (Barres et al., 2025), and TERMINAL-BENCH (Merrill et al., 2026). The public dataset card describes several benchmark splits and multiple passes from the same five models on the same tasks; it records SEARCH as a split label with web-search trajectories for answering complex questions, including `browsecomp`, `webvoyager`, and `mind2web` domains (Chenyan Xiong Research Group at CMU, 2026). We therefore use SEARCH as the release split name rather than as a standalone benchmark title, while noting that the release tool inventory is reconstructed from the General-AgentBench source tree (Chenyan Xiong Research Group at CMU, 2026; Li et al., 2026). We use the five splits with rich tool or environment interaction and exclude the short math split, leaving a retained release subset of 7,329 trajectory records on 427 tasks before graph-validity filtering. The analyzed trajectories are English-language task interactions with code, tool, or API observations. The models are DeepSeek-R1, DeepSeek-V3.2, Gemini-2.5-Flash, Qwen3-235B, and Qwen3-Next.

For the four binary-reward splits, $y_r$ is the

recorded binary success/reward flag: for example, SWE-BENCH uses resolved-style issue success, whereas SEARCH uses the binary `score` recorded in the release. For MCPBENCH, which reports a continuous multi-criteria score, we use per-task max-normalized reward. This avoids introducing a binary threshold for a split that is near-saturated under a positive-reward rule. For SEARCH, we add pre-specified search-interaction keys for URL domain, query novelty, and cumulative source diversity; exact extraction rules are in Appendix C.1.

### 4.2 Do TRACEGRAPH Landscapes Expose Interpretable Agent States?

We first check whether a TRACEGRAPH landscape is a usable measurement scaffold. The evidence is deliberately modest: it asks whether the graph units are interpretable enough to support analysis, not whether every block has a perfect semantic label.

**Block semantics.** Two annotators scored 400 blinded BCC pairs, split evenly between within-BCC and across-BCC pairs. After dropping unsure labels, within-BCC pairs were marked similar 84.8% of the time (162/191), compared with 40.2% for across-BCC pairs (72/179; Fisher exact OR=8.30, $p = 1.4 \times 10^{-19}$). Per-benchmark gaps range from +89 percentage points on SWE-BENCH to +25 percentage points on TERMINAL-BENCH; SEARCH has a +29 point gap under the pre-enrichment signature. This supports reading BCCs as coarse shared agent states (Figure 3a).

**Articulations and regions.** A second annotation pass scored 100 graph-identified articulation crossings. Eighty-two show at least a minor strategy or phase change, a 4.6× enrichment over random consecutive steps (Figure 3b). We use this as a graph-construction sanity check, not as a main supply–demand axis. A third pass scored 50 core blocks and 50 trap blocks. Core blocks are 2.1× more likely than trap blocks to be judged productive progress (34% vs. 16%, Fisher $p = 0.032$), and region-consistent extreme labels—productive core blocks and looping/dead-end trap blocks—outnumber inconsistent extremes 35 to 11. However, 36% of trap blocks are judged exploration, so a trap is best understood as a low-outcome region, not necessarily useless behavior (Figure 3c).

**Why a shared graph?** Pooled graphs produce an average of 16.1 non-trivial blocks across 408 valid tasks, enough to compare several models as traffic on the same terrain. Per-model graphs are much smaller because each model contributes only a few rollouts per task. A representation ablation further shows that the full multi-channel signature has the broadest valid-task coverage; the ablation is kept in Appendix B.4 because it is a sanity check rather than the semantic core of the paper.

### 4.3 What Process Profiles Do Models Exhibit?

Once the landscape is fixed, model identity is traffic over the same terrain. Figure 4a shows model supply as a heatmap; exact values with bootstrap CIs are in Appendix Table 5. Positive values mean the model exhibits that event more often than the average model on the same tasks; for Trap exposure, lower values mean more avoidance.

The profiles give model descriptions rather than a new ranking. **DeepSeek-V3.2** is the clearest high-access, high-repair model: its Access and Repair CIs are both positive, while Trap is nearly neutral. **Qwen3-Next** is low-access and low-repair, with lower Trap supply; this looks less like robust recovery and more like conservative, shallow navigation. **Qwen3-235B** shows mildly elevated Access and marginally elevated Trap exposure, suggesting a less conservative exploration style, but its Repair interval overlaps zero. **DeepSeek-R1** is near-balanced, and **Gemini-2.5-Flash** has lower Access and Repair supply in this corpus.

### 4.4 What Process Demands Do Benchmarks Impose?

We next ask which events separate high-outcome from low-outcome traffic inside each benchmark. Figure 4b shows the demand heatmap; exact values with bootstrap CIs are in Appendix Table 6. On binary splits this is resolved minus failed traffic; on MCPBENCH it is the continuous reward-weighted analogue.

Access is broadly shared: every split has a positive Access demand with CIs excluding zero. The more diagnostic differences are Trap and Repair.

**Trap-averse benchmarks.** $\tau^2$-BENCH has the most negative Trap demand ($-0.401$), while MCPBENCH ($-0.166$) and TERMINALBENCH ($-0.124$) are also trap-averse with intervals below zero. High-outcome rollouts in these splits avoid low-outcome regions, which matches settings where state updates, protocol mistakes, or invalid tool choices can be difficult to undo. MCPBENCH also has a clearly positive Access de-

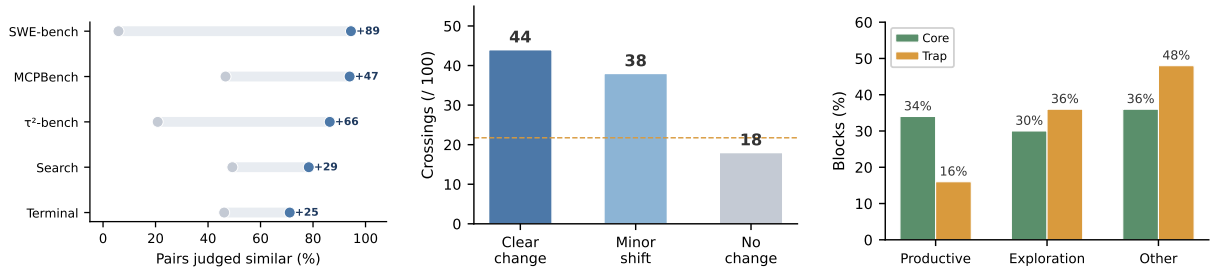


Figure 3: Human annotation checks for graph-derived states. **(a)** Within-BCC step pairs are judged similar far more often than across-BCC pairs on every benchmark (overall 84.8% vs. 40.2% after dropping unsure labels; Fisher OR = 8.30). **(b)** Of 100 graph-identified articulation crossings, 82 show a strategy or phase change (4.6× the random baseline, dashed line). **(c)** Core blocks are annotated as productive progress 2.1× more often than trap blocks (34% vs. 16%); 36% of trap blocks are exploration rather than dead ends.

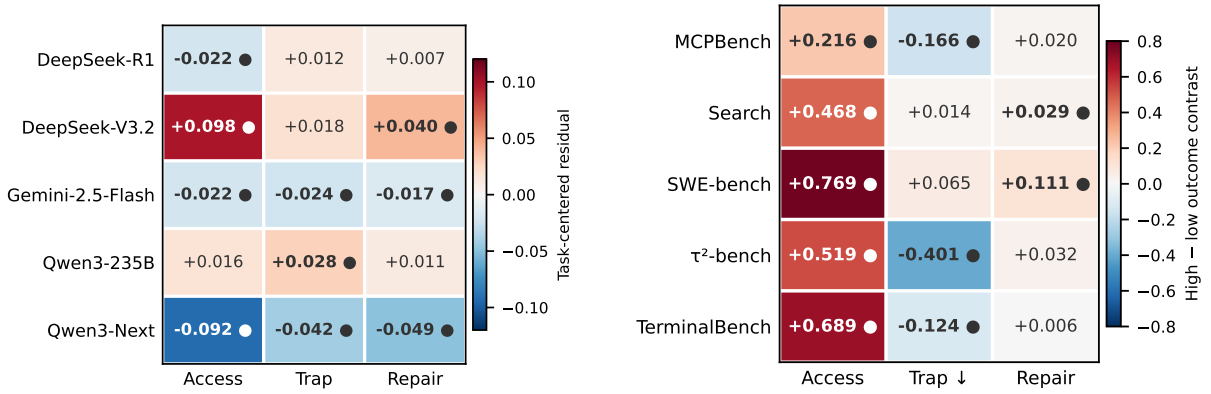


Figure 4: Supply–demand profile heatmaps. **(a)** Model supply: task-centered event residuals; DeepSeek-V3.2 is distinctly high-Access/high-Repair, while Qwen3-Next is low on both. **(b)** Benchmark demand: resolved-minus-failed contrasts (or the reward-weighted analogue on MCPBENCH); Access is universally positive, $\tau^2$-BENCH is strongly Trap-averse, MCPBENCH/TERMINALBENCH are also negative on Trap, and SWE-BENCH has by far the strongest positive Repair demand. Filled markers (●) denote 95% bootstrap CIs excluding zero.

mand (+0.216), suggesting that successful tool-orchestration trajectories must both reach productive regions and avoid drifting into low-outcome states. This explains why lower-Trap supply can be useful even when a model is otherwise low-access.

**Repair-heavy software tasks.** SWE-BENCH has the strongest Repair demand (+0.111) and the strongest Access demand (+0.769), both with positive intervals. Successful repository-editing trajectories must both reach productive code regions and recover after failed tests, wrong files, or bad edits. This makes SWE-BENCH the natural setting for the TRACEGRAPH-guided recovery pipeline in §5: the intended lever is not trap avoidance, but recovery after a trap-like state. By contrast, the negative Trap demand on $\tau^2$-BENCH and MCPBENCH suggests a different design target, namely preventing risky state updates before they occur, so we do not use a repair-style diagnosis policy as the pipeline test for those splits.

**Search as access-and-repair.** SEARCH is not trap-averse in this signature, but it has positive Access demand (+0.468) and a smaller positive Repair demand (+0.029). Under the evidence-quality keys, high-outcome search rollouts are distinguished mainly by reaching productive evidence states, with occasional recovery from unproductive query paths.

Together, the supply and demand profiles make the central point: benchmarks are not simply harder or easier versions of the same task. They reward different process events, and models align with those demands in different ways. Figure 4 visualises both matrices as heatmaps; filled markers denote bootstrap CIs excluding zero. Figure 5 concretizes this on a single SWE-BENCH task, showing how three models navigate the same shared landscape with visibly different strategies.

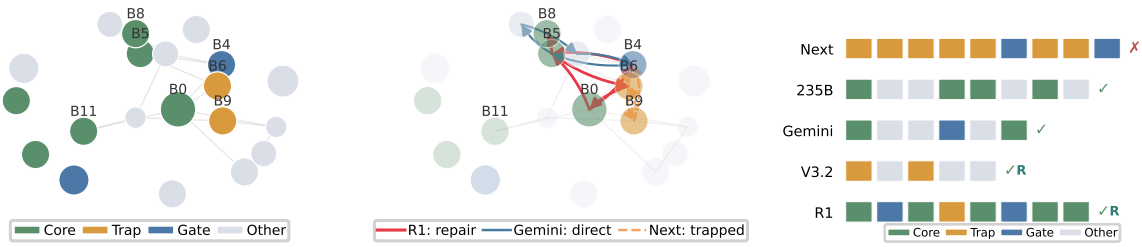


Figure 5: A shared landscape on SWE-BENCH task `django-9296`. **(a)** Block quotient graph: nodes are BCC blocks (green=core, amber=trap, blue=articulation landmark, gray=other). **(b)** Three trajectories as arrows: DeepSeek-R1 repairs from B6 back to core; Gemini navigates directly; Qwen3-Next (dashed) cycles in B9↔B6 without escaping. **(c)** Block ribbons per model; ✓/× marks resolution, **R** marks Repair.

## 5 TRACEGRAPH-Guided Trap-Aware Recovery

The TRACEGRAPH analysis asks where high- and low-outcome traffic concentrates on a shared task map. We next test whether that map can guide a small recovery pipeline: act only when a live rollout reaches a state resembling historical traps, and use only information already visible in the agent's context. Because §4.4 shows that SWE-BENCH most strongly demands recovery after trap exposure, we instantiate trap-aware recovery on SWE-BENCH and measure official resolved rate after trap-triggered continuations. The goal is not a universal repair algorithm, but a test of whether graph-derived traps identify states where a lightweight, non-oracular continuation change helps.

### 5.1 From Trap Regions to Runtime Triggers

TRACEGRAPH converts the trap overlay into a runtime detector by storing canonicalized key sets from trap-side states, such as `ACTION:edit`, `OBS:error`, `TOOL:<name>`, and local file-path cues. Generic phase and non-tool keys are stripped so the detector matches recurring failure contexts rather than full traces. A second library comes from high-committor non-trap states. At a live step $s$, the detector computes the candidate key set $q_s$ and scores

$$\mathrm{sim}_{\mathrm{trap}}(s) = \max_{u \in \mathcal{T}} J_{\mathrm{idf}}(q_s, u),$$
$$\mathrm{sim}_{\mathrm{core}}(s) = \max_{u \in \mathcal{N}} J_{\mathrm{idf}}(q_s, u), \qquad (8)$$

where $\mathcal{T}$ and $\mathcal{N}$ are the trap and non-trap libraries. A trigger fires when trap similarity clears a fixed threshold and exceeds core similarity by a margin (Appendix Table 7). For SWE-BENCH, the current step must also carry a local file-path cue, and the candidate action must be an edit or submit action. These local checks keep the trigger tied to observable graph signatures while reducing fires on generic exploration.

### 5.2 Prefix-Fork Recovery Policy

When the detector fires, we snapshot the docker workspace and conversation prefix, then continue from that identical state under three policies. The no-intervention *Baseline* continues at $T$=0.6 with no added note. *Hot* raises temperature to $T$=0.9 with top-$p$=0.95. *Note* keeps $T$=0.6 and appends a conservative diagnosis note drawn only from the agent's recent log: the last action, observation signal, implicated file or target keys, detector confidence, and a family-level diagnosis. The note asks the agent to re-read the failing evidence, localize narrowly, make one minimal change, and revise or discard the edit if the evidence does not support it. No arm receives a gold patch, verified-test list, or oracle information.

This prefix-fork design compares the same trap state with no pipeline action, extra sampling diversity, or evidence-grounded localization. We report official SWE-BENCH resolved rate in Table 2; Appendix B.10 provides paired uncertainty estimates.

### 5.3 Downstream Accuracy Improves at Trap-Triggered States

The experiment uses the full 500-instance SWE-BENCH Verified pool at seed 11, spanning 12 repositories and three continuation providers not used in the observational graph analysis: Qwen3.6-35B-A3B, GLM-5.1, and DeepSeek-V4-Pro. Table 2 reports each provider's complete fired subset and a common-fired subset of 96 instances where all providers fire and complete all three arms.

TRACEGRAPH-triggered recovery improves downstream resolution on fired states. In the per-provider fired subset, pooled Hot raises resolved rate from 40.4% to 43.5% (+3.1 pp,

| Provider | $n_{\text{fired}}$ | Baseline ($T$=0.6, no note) | Hot ($T$=0.9, no note) | Note ($T$=0.6, note) |
|---|---|---|---|---|
| *Per-provider fired subset (each provider's own complete set)* | | | | |
| Qwen3.6-35B-A3B | 214 | 29.4 | 30.8 (+1.4) | **34.1** (+4.7) |
| GLM-5.1 | 196 | 48.0 | **52.6** (+4.6) | 49.5 (+1.5) |
| DeepSeek-V4-Pro | 206 | 44.7 | **48.1** (+3.4) | 45.6 (+1.0) |
| Pooled (3-prov, $n$=616 cells) | 616 | 40.4 | **43.5** (+3.1) | 42.9 (+2.4) |
| *Common-fired subset (same 96 instances, all three providers)* | | | | |
| Qwen3.6-35B-A3B | 96 | 31.2 | 31.2 | **34.4** (+3.1) |
| GLM-5.1 | 96 | 50.0 | **56.2** (+6.2) | 52.1 (+2.1) |
| DeepSeek-V4-Pro | 96 | 41.7 | **46.9** (+5.2) | 44.8 (+3.1) |
| Pooled (3-prov, $n$=288 cells) | 288 | 41.0 | **44.8** (+3.8) | 43.8 (+2.8) |

Table 2: TRACEGRAPH-guided trap-aware recovery on the full 500-instance SWE-BENCH Verified pool. Cells report official resolved rate (%); parentheses show paired gain over Baseline. Gains are computed from unrounded paired outcomes, so they may differ by 0.1 pp from subtracting displayed rounded rates. Best non-Baseline cells are bold-shaded. Appendix B.10 reports uncertainty; Appendix B.11 gives cases.

$p$=0.016), and pooled Note reaches 42.9% (+2.4 pp, $p$=0.064). The common-fired subset shows the same pattern: Hot increases pooled resolved rate from 41.0% to 44.8% (+3.8 pp, $p$=0.045). Thus a trap region discovered from shared offline trajectories can become a runtime trigger that improves a downstream metric.

The active recovery route is provider-specific: Qwen3.6-35B-A3B benefits most from the diagnosis note (+4.7 pp, $p$=0.036), while GLM-5.1 (+4.6 pp, $p$=0.037) and DeepSeek-V4-Pro (+3.4 pp) benefit most from the temperature bump.

Appendix B.10 reports the full component statistics and the conservative diagnosis template; Appendix B.11 gives case studies showing how the same trigger state can lead to different provider-specific recoveries.

## 6 Discussion and Conclusion

TRACEGRAPH is neither another scalar score nor a blind success predictor. Its outcome-informed core and trap regions are descriptive: they show where successful and failed traffic concentrates on a shared map. Projecting a rollout into this map turns a pass-or-fail label into a process reading over Access, Trap exposure, and Repair. Thus, models with similar aggregate scores may still occupy different regions, avoid different dead ends, or recover at different moments. In our analysis, DeepSeek-V3.2 is the clearest high-access, high-repair model, Qwen3-Next is conservative and shallow, and other models lie between these extremes (§4.3). Benchmarks emphasize different events: $\tau^2$-BENCH is strongly trap-averse, MCPBENCH and TERMINAL-BENCH are negative on Trap, SWE-BENCH adds Repair demand, and SEARCH mainly rewards productive evidence access (§4.4). These contrasts are not a new leaderboard, but coordinates for interpreting whether a task suite tests exploration, avoidance, recovery, or their mixture. The central claim is that benchmark difficulty reflects mixed process demands rather than a single hidden factor.

The recovery experiment shows that this map can guide a small downstream intervention without becoming a full controller. Graph-derived traps trigger prefix-fork continuations only when a live SWE-BENCH rollout enters a historical failure region, keeping the intervention sparse and interpretable. On fired states from the 500-instance SWE-BENCH Verified pool, the best pooled single-factor policy raises resolved rate by +3.1 points on the per-provider fired subset ($p$=0.016) and by +3.8 points on common-fired instances ($p$=0.045). The useful cue is provider-specific: Qwen3.6-35B-A3B benefits most from an evidence-grounded note, while GLM-5.1 and DeepSeek-V4-Pro benefit most from a temperature bump. Together, these results separate when a trajectory enters a risky region from how a continuation model escapes it.

Shared process landscapes make benchmark comparisons more legible, expose local events behind global scores, and identify states where lightweight interventions can be tested. They offer a vocabulary for asking where systems go, where they get stuck, and how they recover.

## Limitations

The TRACEGRAPH landscape roles are outcome-informed, so the framework is an analytical instrument for explaining where high- and low-outcome traffic concentrates, not a blind success predictor. The source data contains roughly four passes per model per task across five observational models; shared graphs mitigate sparsity for landscape construction but not for cell-level inference. For MCP-BENCH, a per-task max-normalized continuous reward replaces binarization but treats the per-task maximum as the local ceiling, which it may not be. The symbolic signature captures tool, command, observation, file, and phase cues, but may miss fine-grained natural-language semantics, especially for SEARCH. The TRACEGRAPH-guided recovery pipeline applies only to a structurally selected SWE-BENCH fired subset and three continuation providers, so its gains should be read as local downstream improvements rather than as a universal agent-repair result.

## References


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

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Kang Chen, Junjie Nian, Yixin Cao, and Yugang Jiang. 2026a. SliceGraph: Mapping process isomers in multi-run chain-of-thought reasoning. *Preprint*, arXiv:2605.14619.

Kang Chen, Fan Yu, Junjie Nian, Shihan Zhao, Zhuoka Feng, Zijun Yao, Heng Wang, Minshen Yu, and Yixin Cao. 2026b. Thinking traps in long chain-of-thought: A measurable study and trap-aware adaptive restart. *Preprint*, arXiv:2601.11940.

Chenyan Xiong Research Group at CMU. 2026. Agent trajectories dataset: Processing and format documentation. Hugging Face dataset.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2023. A framework for few-shot language model evaluation. Zenodo. Version v0.4.0.

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

Xiaochuan Li, Ryan Ming, Pranav Setlur, Abhijay Paladugu, Andy Tang, Hao Kang, Shuai Shao, Rong Jin, and Chenyan Xiong. 2026. Benchmark test-time scaling of general LLM agents. *Preprint*, arXiv:2602.18998.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D. Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, and 31 others. 2023. HELM: Holistic evaluation of language models. *Transactions on Machine Learning Research*. Featured Certification, Expert Certification.

Shuyang Liu, Yang Chen, Rahul Krishna, Saurabh Sinha, Jatin Ganhotra, and Reyhan Jabbarvand. 2026. Process-centric analysis of agentic software systems. *Preprint*, arXiv:2512.02393.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024. AgentBench: Evaluating LLMs as agents. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, and 66 others. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *Preprint*, arXiv:2601.11868.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *Proceedings of the International Conference on Learning Representations*.

Xue Wen Tan, Nathaniel Tan, Galen Lee, and Stanley Kok. 2025. The shape of reasoning: Topological analysis of reasoning traces in large language models. *Preprint*, arXiv:2510.20665.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the Eleventh International Conference on Learning Representations*.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. 2025. MCP-bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. *Preprint*, arXiv:2508.20453.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Zhen Xiong, Yujun Cai, Zhecheng Li, and Yiwei Wang. 2025. Mapping the minds of LLMs: A graph-based analysis of reasoning LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17751–17763.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. 2025. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. In *Proceedings of the Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.

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

## A Responsible Research Statement

**Potential risks.** TRACEGRAPH is an analysis and evaluation framework, not a deployed agent-control system. The main risks are over-interpreting outcome-informed graph roles as blind predictors of success, overfitting recovery policies to benchmark-specific traps, or using automated software-editing agents without human review. We mitigate these risks by presenting the recovery pipeline as a local benchmark intervention on fired SWE-BENCH states, using no gold patches or verified-test lists in the recovery note, reporting provider-specific behavior, and avoiding any deployment recommendation.

**Artifacts, licenses, and intended use.** We use access-controlled or provider-served research artifacts: the gated cx-cmu agent trajectory release on Hugging Face, the underlying benchmark suites and evaluation harnesses, and the three continuation models cited in §4.1 and §5. The public card for the cx-cmu release gates file access behind contact sharing, and the public README does not list a separate dataset license or a detailed citation block. We therefore cite the dataset card directly, treat the trajectories as access-controlled research material, and use them only for the analysis reported in this paper. For the recovery study, GLM-5.1 and DeepSeek-V4-Pro are accessed through provider APIs under provider terms, while Qwen3.6-35B-A3B is run locally from Qwen-family weights whose public model card lists the Apache-2.0 license. We do not redistribute raw trajectories, benchmark instances, source repositories, model weights, or API outputs. Our intended use is research evaluation and process analysis of agent benchmarks, and any future sharing of derived artifacts would need to respect the access conditions of the source artifacts.

**Privacy, content, and data handling.** We do not collect new personal data, infer demographic attributes, or conduct user profiling. The source artifacts consist of benchmark tasks, model-generated traces, tool outputs, repository paths, issue statements, and synthetic or benchmark-specific environment records. We inspected the schema and representative examples for fields that directly name or identify people, and the analysis converts textual fields into canonicalized symbolic keys such as tool type, command class, observation pattern, file cue, and phase cue. The paper reports aggregate statistics and illustrative benchmark snippets rather than raw private user conversations.

**Computation and reproducibility.** We do not train or fine-tune models. Graph construction, annotation analysis, bootstrapping, and table generation use custom scripts over fixed symbolic signatures and fixed hyperparameters reported in §3 and Appendix 3. The recovery experiment uses two API-served continuation providers (GLM-5.1 and

DeepSeek-V4-Pro) and one locally hosted continuation model (Qwen3.6-35B-A3B). Provider-side infrastructure for the API models is not observable to us, so we report model/provider names, data sizes, fired subsets, temperatures, top-$p$ values, confidence intervals, and permutation tests rather than provider GPU hours. The locally hosted Qwen3.6-35B-A3B runs on a single NVIDIA H20 GPU, and the Qwen intervention sweep consumed approximately ten GPU-hours in total.

**Human annotation.** The graph-interpretability sanity checks use two internal annotators rather than crowdworkers or recruited human subjects. The annotators were given task-level trajectory fragments and asked to judge within- vs. across-block similarity, articulation phase changes, and whether core/trap blocks correspond to productive progress or low-outcome exploration. No external participants were recruited or paid, no annotator personal data were collected, and the annotation is used only as a sanity check for graph interpretability. The full instruction text is in Appendix B.6.

**AI assistance.** AI assistants were used for language polishing, LaTeX editing, patch preparation, and submission-checklist drafting. The authors supplied the research ideas, experimental design, data analysis, and claims, and manually verified all citations, numerical results, and final text. No AI assistant is treated as an author.

# B Additional TRACEGRAPH Details

## B.1 Hyperparameters

All graph and diffusion hyperparameters are fixed across benchmarks.

| Parameter | Value |
|---|---|
| Mutual-$k$NN $k$ | 6 |
| Edge-weight RBF scale $\sigma$ | 0.35 |
| Diffusion teleport $\alpha$ | 0.65 |
| Diffusion steps | 24 |
| Core quantile (top, $C_t$) | 0.75 |
| Trap quantile (bottom of negative, $B_t$) | 0.25 |
| Bootstrap resamples | 2,000 |

Table 3: Fixed hyperparameters used by TRACEGRAPH.

## B.2 Rollout-Event Definitions

The three rollout events in §3.3 are defined on the role overlays $(C_t, B_t)$. For a rollout $r$ with compact primary-block sequence $b_{r,1:T_r}$ and visit set $V_r = \{b_{r,s}\}$:

$$A_r = \mathbf{1}[V_r \cap C_t \neq \emptyset], \tag{9}$$

$$E_r = \mathbf{1}[V_r \cap B_t \neq \emptyset], \tag{10}$$

$$R_r = \mathbf{1}[\exists s < u : b_{r,s} \in B_t,\ b_{r,u} \in C_t]. \tag{11}$$

For binary benchmarks $y_r \in \{0, 1\}$; for MCP-BENCH we use the per-task max-normalized continuous reward $y_r = \text{reward}_r / \max_{r' \in \mathcal{R}_t} \text{reward}_{r'} \in [0, 1]$.

The model supply $S_{m,a}$ is the average over tasks of the task-centered cell residual:

$$\begin{aligned} a_{m,t} &= \mathbb{E}_{r \in \mathcal{R}_{t,m}}[a_r], \\ \tilde{a}_{m,t} &= a_{m,t} - \frac{1}{|M_t|} \sum_{m' \in M_t} a_{m',t}, \\ S_{m,a} &= \mathbb{E}_t[\tilde{a}_{m,t}], \end{aligned} \tag{12}$$

where $a \in \{A, E, R\}$.

The benchmark demand $D_{b,a}$ is the reward-weighted contrast over rollouts:

$$\begin{aligned} D_{b,a} &= \frac{\sum_{r \in b} w_r a_r}{\sum_{r \in b} w_r} - \frac{\sum_{r \in b}(1 - w_r) a_r}{\sum_{r \in b}(1 - w_r)}, \\ w_r &= y_r. \end{aligned} \tag{13}$$

For binary $y_r$, $w_r$ is $0/1$ so Eq. 13 reduces exactly to the mean-over-resolved minus mean-over-failed contrast.

## B.3 Axis Sign Sanity, Rollout Events

The three events are descriptive coordinates, not predictors, but they should still align with success in the expected direction across the corpus: high-outcome traffic should have larger $A$ and $R$ and smaller $E$ than low-outcome traffic. Pooling all rollouts on the four binary splits and computing the unweighted contrast $\mathbb{E}_{y=1}[a_r] - \mathbb{E}_{y=0}[a_r]$ with 2,000-bootstrap 95% CIs, all three events have CIs that exclude zero in the expected direction (Table 4). Including MCPBENCH under the continuous treatment gives the same qualitative pattern: positive Access, negative Trap, and small Repair.

## B.4 Signature Ablation

We rebuild the pipeline on 150 tasks under seven representation conditions. The full signature has the largest model-separation statistic among non-degenerate representations and the broadest valid-task coverage; observation-only and no-phase variants remain useful, tool-only lowers separation while preserving broad coverage, and file-only,

| Event | $\mathbb{E}_{y=1} - \mathbb{E}_{y=0}$ | 95% CI |
|---|---|---|
| Access ($A_r$) | $+0.572$ | $[+0.544, +0.600]$ |
| Trap exposure ($E_r$) | $-0.064$ | $[-0.091, -0.036]$ |
| Repair ($R_r$) | $+0.040$ | $[+0.023, +0.057]$ |
| All three sign-consistent | | Yes |

Table 4: Rollout-event sign sanity on the four binary benchmarks (4,686 graph-valid binary rollout-event rows; 2,000-bootstrap 95% CIs). All three main events shift in the expected direction at $p < 0.05$.

action-only, and random reassignment collapse coverage, with action-only yielding no stable separation estimate (Figure 6).

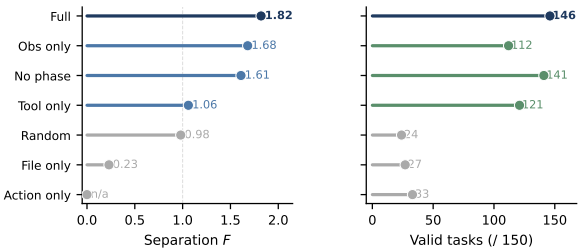


Figure 6: Signature ablation on 150 tasks. Full signatures give the strongest model separation ($F = 1.82$) and widest valid-task coverage ($146/150$); random, file-only, and action-only collapse coverage, and action-only yields no stable separation estimate.

### B.5 Score Matrix and Reward Semantics

The four binary-reward splits use the resolved/failed flag of each rollout. For MCP-BENCH, the dataset provides a continuous multi-criteria reward; the main text uses the per-task max-normalized continuous reward $y_r = \text{reward}_r / \max_{r' \in \mathcal{R}_t} \text{reward}_{r'}$ throughout the reward field and benchmark-demand calculations. A simple positive-reward threshold is near-saturated ($\sim 97\%$ resolved across models), so we use the normalized continuous treatment instead; no threshold sweep is reported as a main result because the analysis avoids binarization altogether.

### B.6 Annotation Instructions

The graph-interpretability sanity checks used two internal annotators. The annotators were shown de-identified task id, benchmark name, two or three consecutive trajectory fragments, and the graph-derived relation being checked; model identity and terminal outcome were hidden. The instruction text was:

> Read only the displayed action, observation, tool, file, and phase cues. For a pair of steps, label them *similar* if they represent the same local subtask, evidence-gathering mode, tool-use mode, or edit/debugging phase, even if the surface text differs; otherwise label them *different*. For an articulation crossing, label whether the second step marks a clear strategy or phase change relative to the first. For a block, label the fragment as productive progress, exploration without convergence, repeated or stuck behavior, or unclear. Do not infer model identity, do not use terminal success, and mark unclear when the displayed evidence is insufficient. This annotation is for aggregate method validation only and should not identify any person or judge any human participant.

No screenshots or additional risk disclaimers were used because the task was an internal, non-interactive annotation of already released agent traces. The directional sanity-check $p$-values reported in §4.2 use one-sided alternatives matched to the stated hypotheses: within-BCC similarity $>$ across-BCC similarity, and core productive-progress rate $>$ trap productive-progress rate.

### B.7 Illustrative Annotation Examples

We show representative trajectory fragments from each of the three annotation tasks in §4.2. Tool arguments and observations are lightly trimmed for space.

**BCC pair: within-block (similar).** SWE-BENCH, task `sympy-16886`. Two steps from the same BCC block; the annotator labelled them *similar*.

**A** `view crypto/crypto.py:1500-1550` → Morse-code definitions
**B** `view crypto/crypto.py:500-600` → Vigenère helpers

Both steps view different sections of the same file, reflecting a single "inspect source" strategy. The graph merges them because their key sets share the high-IDF key `FILE_PATH:crypto/crypto.py`.

**BCC pair: across-block (different).** SWE-BENCH, task `pytest-5773`. Two steps from different BCC blocks; the annotator labelled them *different*.

**A** `bash: git show de6f.. | grep "python_files"` → commit diff
**B** `str_replace` on `_pytest/python.py` → "replaced"

Step A inspects git history; Step B edits source code. Their key sets differ on tool, action type, command class, and observation pattern, so they land in separate blocks.

**Articulation crossing: clear phase change.** TERMINALBENCH, task `processing-pipeline`. Two steps immediately before and after an articulation point; the annotator labelled the crossing a *clear phase change*.

**Before** `read_file collect_data.sh` → shell script body
**After** `read_file run_pipeline.sh` → orchestration calling `collect`, `process`, `report`

The rollout shifts from examining an individual component to the top-level orchestration wrapper. The articulation point captures this strategy transition.

**Core block: productive progress.** MCPBENCH, task `mcpbench_10`. Three consecutive steps from a block labelled *core* by the diffusion field; the annotator judged the block *productive progress*.

**1** `MCP_sum([10.3, 5.4, 8.2])` → `23.9`
**2** `MCP_sum([52, 29, 43])` → `124`
**3** `MCP_sum([52.2, 29.1, 43.2])` → `124.5`

The rollout makes consistent, goal-directed MCP tool calls. All observed visitors attain the task-local maximum normalized reward, placing the block firmly in the productive core.

**Trap block: low-outcome exploration.** SWE-BENCH, task `django-14580`. Three steps from a block labelled *trap*; the annotator judged the block *exploration without convergence*.

**1** `view serializer.py:270-280` → `TypeSerializer` def
**2** `view serializer.py:170-190` → `functools` imports
**3** `view serializer.py:400-450` → (empty—file ends)

The rollout reads scattered sections of the same file without converging on a fix. Block purity is 0.125 (most visitors fail), placing it in the trap region. Notably, 36% of annotated trap blocks are labelled exploration rather than dead-end, consistent with the main-text observation that traps are low-outcome regions, not necessarily useless behavior.

### B.8 Robustness of Supply and Demand Profiles

We report task-bootstrap CIs in the main text and additionally sweep the graph-role hyperparameters. The qualitative signs used in the paper are stable under core-quantile, trap-quantile, Laplace-smoothing, diffusion, and mutual-$k$NN perturbations. For MCPBENCH under the continuous treatment, Access remains positive and Trap remains negative across the same perturbations, while Repair stays small. Trap-averse conclusions for MCPBENCH and $\tau^2$-BENCH are stable across trap thresholds, and SWE-BENCH remains the split with the strongest positive Repair demand across the baseline sweep. The only borderline sign in the sweeps is TERMINALBENCH Trap under the largest $k$NN setting, which is why the main text treats TerminalBench as mildly rather than strongly trap-averse. As an external replication check, an independently ingested TerminalBench trajectory set reproduces the three-axis demand shape (Access positive, Trap negative, Repair near zero). Figure 7 shows the baseline demand values alongside the sweep ranges; all signs remain stable except the borderline TERMINALBENCH Trap under the largest $k$NN setting.

### B.9 Full Supply and Demand Tables

Tables 5 and 6 give the exact numerical values underlying Figure 4. Table 5 reports model supply as task-centered residuals with task-bootstrap confidence intervals, while Table 6 reports benchmark demand as resolved-minus-failed contrasts (or the reward-weighted analogue for MCPBENCH); positive Access and Repair indicate that high-outcome traffic uses those events more often, whereas negative Trap indicates stronger avoidance of low-outcome regions.

### B.10 Additional Trap-Aware Recovery Details

**Recovery arms.** The three prefix-fork arms defined in §5.2 separate two single-factor recovery routes. *Baseline* continues from the trigger snapshot at the baseline temperature $T$=0.6 with no inserted note. *Hot* switches the continuation to $T$=0.9, top-$p$=0.95 with no note, isolating the effect of increased sampling diversity. *Note* appends the diagnosis note at $T$=0.6, isolating the effect of evidence-grounded localization.

**Conservative diagnosis note.** The trap-side note is a fixed template populated at fire time. The oper-

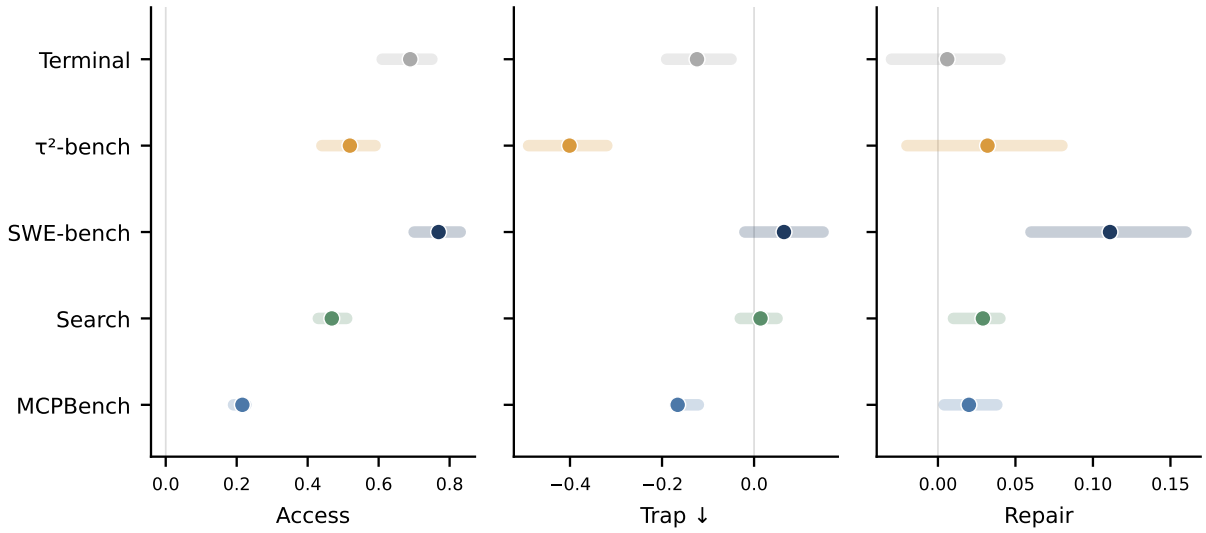


Figure 7: Demand-sign robustness across hyperparameter sweeps. Dots show baseline values; shaded bars show the range across core-quantile, trap-quantile, Laplace-smoothing, diffusion, and mutual-$k$NN perturbations. All qualitative signs are stable except the borderline TERMINALBENCH Trap sign under the largest mutual-$k$NN setting.

| Model | Access | Trap | Repair |
|---|---|---|---|
| DeepSeek-R1 | $-.022$ $[-.043, -.001]$ | $+.012$ $[-.010, +.035]$ | $+.007$ $[-.007, +.020]$ |
| DeepSeek-V3.2 | $+.098$ $[+.077, +.118]$ | $+.018$ $[-.003, +.038]$ | $+.040$ $[+.027, +.052]$ |
| Gemini-2.5-Flash | $-.022$ $[-.040, -.004]$ | $-.024$ $[-.041, -.006]$ | $-.017$ $[-.025, -.009]$ |
| Qwen3-235B | $+.016$ $[-.002, +.035]$ | $+.028$ $[+.009, +.047]$ | $+.011$ $[-.000, +.022]$ |
| Qwen3-Next | $-.092$ $[-.116, -.067]$ | $-.042$ $[-.067, -.017]$ | $-.049$ $[-.058, -.039]$ |

Table 5: Model supply profiles with 2,000 task-bootstrap 95% CIs. Values are task-centered event residuals; because graph-valid task support differs by model, columns need not sum to zero exactly.

ative content is: re-read the failing test, traceback, or last database observation; localize to the smallest directly relevant function or read-only check; make one minimal change consistent with the evidence; and revise or discard the current edit/action if the evidence does not support it. The slots are populated only from the agent's own per-step log: last command/action, last observation signal, implicated file or target keys, detector confidence, and a six-way family-level trap diagnosis. No gold patch, verified-test list, or other oracle signal is used.

**Tested instances.** The main TRACEGRAPH-guided recovery experiment uses the full 500-instance SWE-BENCH Verified pool at seed 11, covering all 12 SWE-BENCH repositories that appear in the cx-cmu trajectory release: `django` (231), `sympy` (75), `sphinx-doc` (44), `scikit-learn` (32), `matplotlib` (34), `pytest-dev` (19), `astropy` (22), `pydata` (22), `pylint-dev` (10), `psf` (8), `mwaskom` (2), `pallets` (1). The fired-subset analysis in the upper block of Table 2 uses each provider's own complete three-arm set ($n$=196 for GLM-5.1, 206 for DeepSeek-V4-Pro, 214 for Qwen3.6-35B-A3B); the common-fired analysis in the lower block restricts to the 96 instances on which all three providers fire and produce a complete three-arm set.

**Paired contrast statistics.** Table 8 reports the full set of paired contrasts that summarise Table 2; the main-text intervention $p$-values refer to these same tests. All confidence intervals use the cluster bootstrap with 10,000 resamples clustered by instance id; $p$-values are one-sided paired-sign permutation against the alternative $\Delta > 0$.

### B.11 Trap-Aware Intervention Case Studies

**Illustrative intervention cases.** The following three cases separate a positive xarray example, a positive Django example, and a regression-style failure mode.

**Case 1: `pydata__xarray-4094`.** The same trigger snapshot drives three qualitatively different recovery patterns across providers (Table 9). The detector fires while the agent is already

| Benchmark | Access | Trap | Repair |
|---|---|---|---|
| MCPBENCH | $+.216$ $[+.182, +.249]$ | $-.166$ $[-.199, -.130]$ | $+.020$ $[-.004, +.042]$ |
| SEARCH | $+.468$ $[+.423, +.508]$ | $+.014$ $[-.026, +.054]$ | $+.029$ $[+.007, +.051]$ |
| SWE-BENCH | $+.769$ $[+.713, +.822]$ | $+.065$ $[-.009, +.148]$ | $+.111$ $[+.049, +.175]$ |
| $\tau^2$-BENCH | $+.519$ $[+.456, +.579]$ | $-.401$ $[-.467, -.333]$ | $+.032$ $[-.014, +.083]$ |
| TERMINALBENCH | $+.689$ $[+.637, +.738]$ | $-.124$ $[-.177, -.067]$ | $+.006$ $[-.024, +.038]$ |

Table 6: Benchmark demand profiles with 2,000-bootstrap 95% CIs. Negative Trap means high-outcome traffic avoids low-outcome regions.

| Detector setting | Value |
|---|---|
| Trap similarity threshold | 0.35 |
| Trap–core margin | 0.03 |
| Warmup steps | 2 |
| Cooldown steps | 4 |
| Max triggers per rollout | 1 |
| Allowed intents | edit, submit |
| Observation-key gate | none |
| File-locality gate | require a `FILE_PATH:*` cue |
| Rollout step budget | 30 total steps per arm |

Table 7: Fixed detector and continuation settings used in the 500-instance SWE-BENCH recovery study across all three providers.

reading `xarray/core/dataarray.py`, with trigger keys `ACTION:edit`, `CMD:sed`, `FILE_PATH:xarray/core/dataarray.py`, and an earlier `ValueError` reproduction context. Across providers, the raw assistant traces converge on the same local diagnosis: after selecting one variable from the stacked array, the stacked coordinate remains as a scalar coordinate, so later merging the per-variable arrays back into a dataset creates a coordinate conflict. The divergence appears one refinement later. Qwen3.6-35B-A3B under *Baseline* and *Hot* spends the post-trigger budget on reproduction and environment repair (missing `numpy`, then a `numpy` version mismatch) and never submits a patch. Under *Note*, it eventually reaches the right local fix idea and edits the key line to drop the stacked coordinate after `squeeze(drop=True)`, but the first version uses `drop_vars(dim)` and then fails on the broader multi-dimensional round-trip test because the coordinate is not always present. DeepSeek-V4-Pro shows a milder version of the same phenomenon: *Baseline* reaches the right diagnosis but commits the brittle `drop(dim)` variant, whereas either *Hot* or *Note* is enough to move it to the safer `drop_vars(dim, errors="ignore")` form. GLM-5.1 is the easiest case: once triggered, both non-Baseline arms jump directly to that safe one-line fix.

**Case 2: `django__django-13128`.** The trigger fires later in the rollout, after the agent has already narrowed the bug to `django/db/models/expressions.py` and is inspecting `CombinedExpression` with an `OBS:error` context (Table 10). The trap here is not a wrong first diagnosis but a low-yield validation loop. In the GLM-5.1 *Baseline* trace, the model states the correct diagnosis almost immediately in its own scratch reasoning: subtracting two temporal fields is being inferred as `DateTimeField` when it should produce `DurationField`. But the baseline rollout then burns budget on environment setup and brittle ad-hoc probes, repeatedly re-deriving the same explanation without committing a final patch. *Hot* and *Note* both shorten that loop. The successful GLM arms add a narrow `_resolve_output_field()` override directly to `CombinedExpression`, then fix one subtle bug exposed by a direct probe: comparing field *instances* with `==` is too strict, so the final patch compares field types / internal types and returns `fields.DurationField()` only for temporal subtraction. The *Note* arm is especially illustrative: after trigger it stays anchored to the smallest relevant region—`CombinedExpression` and the temporal-subtraction tests—rather than relaxing the global mixed-type logic elsewhere in `BaseExpression`. DeepSeek-V4-Pro shows the complementary provider-specific pattern on the same instance. Only *Hot* reaches a submitted fix; the *Note* arm remains in the expression-tree / test-harness analysis loop and ends with an empty submission, matching the aggregate result that this provider benefits mainly from extra sampling diversity rather than the diagnosis note itself.

**Case 3: failure mode on `django__django-14311`.** The same framework

| Subset / pool | Contrast | Δ (pp) | 95% CI (pp) | $p$ |
|---|---|---|---|---|
| GLM-5.1 ($n$=196) | Hot − Baseline | +4.59 | [+0.0, +9.2] | 0.037 ✓ |
| | Note − Baseline | +1.53 | [−3.6, +6.6] | 0.348 |
| DeepSeek-V4-Pro ($n$=206) | Hot − Baseline | +3.40 | [−1.5, +8.7] | 0.132 |
| | Note − Baseline | +0.97 | [−4.4, +6.3] | 0.428 |
| Qwen3.6-35B-A3B ($n$=214) | Hot − Baseline | +1.40 | [−2.3, +5.1] | 0.332 |
| | Note − Baseline | +4.67 | [+0.0, +9.3] | 0.036 ✓ |
| Per-provider fired pooled ($n$=616 cells) | Hot − Baseline | +3.08 | [+0.3, +5.9] | 0.016 ✓ |
| | Note − Baseline | +2.44 | [−0.6, +5.5] | 0.064° |
| Common-fired pooled ($n$=288 cells) | Hot − Baseline | +3.82 | [+0.0, +8.0] | 0.045 ✓ |
| | Note − Baseline | +2.78 | [−2.1, +7.6] | 0.148 |

Table 8: Paired contrasts for Table 2. Per-provider rows use each provider's own fired iids; the two pooled rows stack provider × iid cells (clustered by iid). ✓ $p$<0.05; ° $p$<0.10.

| Provider | Baseline | Hot | Note |
|---|---|---|---|
| GLM-5.1 | no patch | ✓ (572 ch) | ✓ (572 ch) |
| DeepSeek-V4-Pro | ×(550 ch) | ✓ (572 ch) | ✓ (572 ch) |
| Qwen3.6-35B-A3B | no patch | no patch | ×(555 ch) |

Table 9: `pydata__xarray-4094` per-arm outcomes. ✓ marks resolved; × marks a submitted patch that fails the verified tests; "no patch" marks an empty submission. Parentheses show patch length in characters.

| Provider | Baseline | Hot | Note |
|---|---|---|---|
| GLM-5.1 | no patch | ✓ (1057 ch) | ✓ (1001 ch) |
| DeepSeek-V4-Pro | no patch | ✓ (1175 ch) | no patch |

Table 10: `django__django-13128` per-arm outcomes for the two providers discussed in the text. Parentheses show patch length in characters.

can also hurt performance when the note over-focuses the continuation on an overly narrow local pattern. On `django__django-14311`, the trigger fires while GLM-5.1 is already inspecting `tests/utils_tests/test_autoreload.py`, with trigger keys anchored to an edit candidate in the autoreload path and an earlier `RuntimeError` context. All three arms correctly identify the broad bug family: `get_child_arguments()` should reconstruct the `-m` target from `__main__.__spec__`. But the Note arm overcommits to the immediately visible package-with-`__main__` example from `utils_tests.test_module`. It rewrites the logic to always use `__spec__.name.removesuffix('.__main__')`, which fixes the package case but breaks the non-package module case and regresses three previously passing tests. By contrast, the successful *Baseline* arm makes the minimal string replacement `name.replace('.__main__', '')`, and the successful *Hot* arm reaches the same fix with a slightly different edit path. This is a concrete example of a trap-aware note becoming too prescriptive: it improves localization, but can also bias the model toward an over-specialized repair if the visible trigger context is only one branch of the full behavioral contract.

| Provider | Baseline | Hot | Note |
|---|---|---|---|
| GLM-5.1 | ✓ (694 ch) | ✓ (953 ch) | × (852 ch) |

Table 11: A regression-style intervention case. On `django__django-14311`, the Note arm fails `test_run_as_non_django_module_non_package` and three pass-to-pass autoreload tests, while the other two arms resolve the instance.

### B.12 Agent Framework and Recovery Templates

The SWE-BENCH continuation agent is a single-tool bash agent. We reproduce here the templates that fully specify what the agent sees on every step and what each arm injects: the system prompt that fixes the response format, the issue-statement user message, the observation wrapper used after every command, and the 5-block diagnosis-note template that the *Note* arm injects at the trigger step (with the *Baseline* and *Hot* arms injecting nothing).

**System prompt.** Used as the first message of every rollout, on every arm.

```
You are a software engineer tasked with
solving a GitHub issue. You have access to a
bash shell to explore the repository,
understand the codebase, locate the relevant
code, and make the necessary changes to fix
the issue.

## Response Format

Every response must contain exactly TWO
sections:

1. **THOUGHT**: Your reasoning about what to
do next. Analyze the problem, plan your
approach, and explain your next step.

2. **ACTION**: A single bash command to
execute. This must be wrapped in a bash code
block.

Example response:

THOUGHT:
I need to find the file that contains the
buggy function.

ACTION:
```bash
find . -type f -name "*.py" | xargs grep -l
"def process_data"
```

## Important Rules

- Each response must contain exactly ONE
bash command in the ACTION section.
- To submit your solution when done, use the
special command:
  ```bash
  submit
  ```
- Do NOT run interactive commands (vim,
nano, python REPL, etc.).
- Do NOT use `git push` or modify remote
state.
- Keep your changes minimal and focused on
the issue.
- Always verify your fix with relevant tests
before submitting.
- If you need to edit a file, use `sed`,
`awk`, or heredoc redirects.
- If a command produces too much output,
pipe through `head` or `tail`.
```

**Initial user message.** Inserted once at step 0 to communicate the GitHub issue. The placeholder `{problem_statement}` is filled with the verbatim SWE-BENCH Verified `problem_statement` field.

```
Here is the GitHub issue to solve:

<issue>
{problem_statement}
</issue>

You are in the repository root. The
repository has already been checked out to
the correct commit. Explore the repo,
understand the issue, make the fix, and run
tests to verify. When you are confident your
fix is correct, use the `submit` command.
```

**Per-step observation wrapper.** After every `ACTION` bash command, the harness executes it inside the docker workspace and returns the (truncated) combined stdout/stderr to the agent as a user message wrapped in:

```
OBSERVATION:
{observation}
```

**Trap-diagnosis note (Note arm).** At the trigger step both NOTE-on arms inject the following message as a user message immediately after the last observation, before the agent's next `THOUGHT`/`ACTION` turn. The header marks the message as internal; the five slots `{ctx_command}`, `{ctx_signal}`, `{ctx_files}`, `{ctx_confidence}`, `{ctx_trap_pattern}` are populated only from the agent's own per-step log and a per-pattern diagnosis sidecar; no gold-patch or verified-test information is used.

```
[INTERNAL DIAGNOSTIC -- not visible to
graders]
A trajectory pattern previously associated
with low task-resolution rates fired at this
step. Concretely:

  - Last command: {ctx_command}
  - Last error/test signal: {ctx_signal}
  - File(s) touched in this region:
  {ctx_files}
  - Detector confidence (trap similarity):
  {ctx_confidence}

  Trap pattern (from prior failed
  trajectories with this signature):
  {ctx_trap_pattern}

Before continuing:
  1. Re-read the failing test/traceback for
  the specific assertion or unexpected value
  (do not rely on memory of earlier steps).
  2. Localize to the smallest function or
  class implicated by that evidence, in the
  file(s) above.
  3. Propose ONE minimal change consistent
  with the evidence; do not rewrite
  unrelated code.
  4. Run the narrowest relevant test or
  check before submitting.
  5. If your current patch is not supported
  by the error/test evidence, revise or
  discard it.

Respond in the normal THOUGHT / ACTION
format with exactly one bash command.
```

**Slot filling.** At trigger time the slots are populated from the agent's own per-step record (no oracle signal): `{ctx_command}` is the most recent `ACTION` bash command, truncated to 200 characters; `{ctx_signal}` is the most recent observation passed through a fixed regular-expression family for traceback / pytest / syntax / import / missing-file / timeout cues, truncated to 300 characters; `{ctx_files}` is the top-3 file paths from the trigger key set (only files the agent itself has opened in its own commands); `{ctx_confidence}` is the trap similarity score $\mathrm{sim}_{\mathrm{trap}}$ formatted to two decimals; `{ctx_trap_pattern}` is the family-level diagnosis string for the matched trap pattern

(one of six pre-specified families), read from a sidecar JSON keyed on the trap key-set fingerprint. The *Baseline* and *Hot* arms inject nothing at the trigger step; the conversation proceeds directly from the trigger observation to the agent's next `THOUGHT`/`ACTION` turn at the arm's continuation temperature.

# C Algorithmic Details

## C.1 Signature Key Alphabet

A step's symbolic key set $K_i$ contains keys from the channels in Table 12. Keys are case-sensitive and prefix-tagged, so different channels never collide in the IDF dictionary. Observation keys are produced by fixed regular-expression families over the first 2,000 characters of the tool response; the families include error, traceback, test pass/fail, syntax error, import error, missing file, permission error, timeout, and explicit success messages. The search-specific keys (`URL_DOMAIN`, `QUERY_NOVELTY`, `EVIDENCE_COUNT`) are pre-specified from the search tool-call/result format, not selected from outcome inspection.

| Channel | Information captured |
|---|---|
| Tool name | Tool or API name after benchmark-specific prefix stripping. |
| Action type | Coarse action class such as bash, edit, read, search, submit, or other. |
| Command class | Shell-command families such as pytest, grep, cat, sed, python, and git. |
| Observation pattern | Error, pass/fail, timeout, missing-file, and success cues. |
| File cue | Last path components and file extensions mentioned in the action. |
| Temporal phase | Early, middle, or late third of the rollout. |
| Search-only | URL/result domain (eTLD+1), query novelty (new/refine/repeat), and cumulative distinct-domain count buckets. |

Table 12: Compact signature key alphabet used by the signature extractor.

## C.2 Reward Field, Core and Trap Masks, Block Quotient Graph

The *block quotient graph* $G_B = (B, E_B)$ has one node per non-trivial BCC and an undirected edge $(b, b') \in E_B$ weighted by the number of slices shared between the two blocks. Edge weights are row-stochastically normalized into a transition matrix $P_B$, and the teleport-$\alpha$ diffusion update is

$$v^{(t+1)} = \alpha\, P_B^\top v^{(t)} + (1-\alpha)\, s, \qquad (14)$$

where $\mathcal{R}_b$ is the set of rollouts visiting block $b$, and the seed $s_b = \hat{\mu}_b - \bar{y}_t$ is the Laplace-smoothed excess outcome of block $b$ relative to the task average, with $\hat{\mu}_b = (\sum_{r \in \mathcal{R}_b} y_r + \frac{1}{2})/(|\mathcal{R}_b| + 1)$. We iterate Eq. 14 for $T = 24$ steps with $\alpha = 0.65$ and L$\infty$-normalize to $[-1, 1]$. Let $Q^+_{.75}$ be the 75th percentile of positive field values and $Q^-_{.25}$ the 25th percentile of negative field values. The role overlays used in the main text are then:

$$C_t = \{b : v_b^{(T)} > 0 \ \wedge \ v_b^{(T)} \geq Q^+_{.75}\}, \qquad (15)$$

$$B_t = \{b : v_b^{(T)} < 0 \ \wedge \ v_b^{(T)} \leq Q^-_{.25}\}. \qquad (16)$$

Note that $B_t$ is a direct bottom-quartile mask of the negative diffusion field, which avoids a circularity between the trap definition and the $R_r$ event. Articulation points remain useful for graph visualization and the RQ1 sanity annotation, but they are not used as a main rollout event or as a recovery target.