BRANCH-RESET: ESCAPING IDEA BASINS IN AUTORESEARCH AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoresearch agents tackle open-ended problems by repeatedly proposing candidate solutions, evaluating them, and using feedback to guide subsequent experiments. We show that independent runs of the same agent on the same task often plateau at substantially different scores, with gaps that persist even after considerable additional compute. Embedding their candidate artifacts by functional similarity provides further evidence that trajectories remain in localized regions of the solution space, which we call *idea basins*. To help agents escape these basins, we study a simple periodic intervention, *branch-reset*. It forks the agent into parallel trajectories that each inherit a copy of the accumulated workspace but begin with fresh chat contexts. After running each trajectory for a fixed horizon, the agent continues from the highest-scoring one. Across 13 long-horizon research and engineering tasks, with individual agent runs lasting up to several days, *branch-reset* outperformed the single-run and best-of-N baselines by a relative improvement of 66.0% and 44.4%, respectively, on the min–max normalized average score under an equal compute budget.

1 INTRODUCTION

Large Language Model (LLM) agents have shown remarkable promise in autonomously solving hard, open-ended research problems requiring long execution horizons (Mang et al., 2026; Merrill et al., 2026; Terminal-Bench-Science Team, 2026). A prominent paradigm for addressing these problems is the *autoresearch* loop (Karpathy, 2026; Novikov et al., 2025), where an agent iteratively proposes a solution, evaluates it within an environment, and uses the execution feedback or submission score to guide its next experiment. Consider, for example, the task of finding the smallest CIFAR classifier that achieves a target accuracy. Although the objective is easy to state, the design

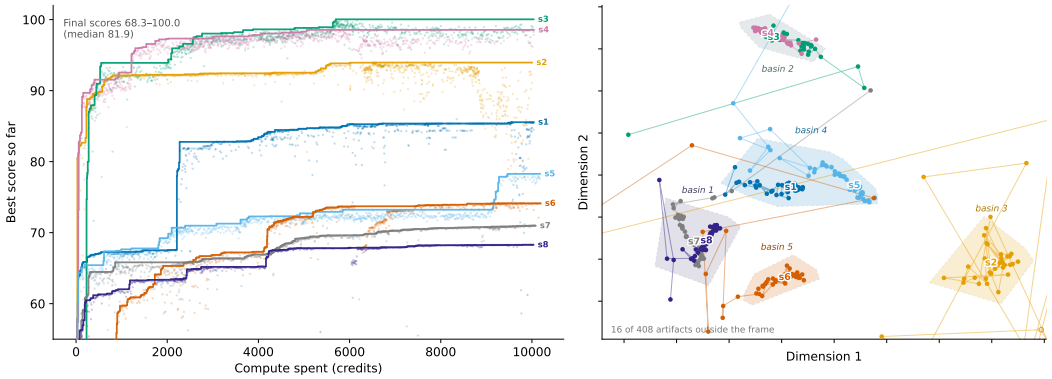

Figure 1: Eight independent agents on p171 research task. **Left:** Best-so-far score against compute spend. Runs tend to plateau at substantially different levels, with gaps persisting even after substantial additional compute. **Right:** The artifacts from these runs embedded by judged functional similarity. Agents converge into five distinct "idea basins" and remain there.

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.

space is broad: the agent can explore different architecture families, layer depths and widths, computational blocks, and training recipes, with many possible tradeoffs between accuracy and model size. A single autoresearch run can evaluate hundreds of candidate solutions and continue for multiple days. This scale raises a fundamental question: over such long horizons, does an agent explore the solution space broadly, or does it mainly refine the ideas it selected early on?

We first examine how solution quality evolves over long autoresearch runs. The score of the best solution found so far tends to plateau within a run. More revealingly, independent runs of the same agent on the same task plateau at substantially different levels, with gaps that persist even after considerable additional compute (Figure 1). This suggests that runs may differ not just in how quickly they improve, but in which parts of the solution space they explore. To investigate this possibility, we embed their candidate artifacts in two dimensions based on functional similarity (Figure 1). The resulting visualization provides additional evidence: submissions from individual runs tend to remain in localized regions of this space, which we call *idea basins*. Some runs appear to spend much of their subsequent effort refining solutions in less productive regions rather than discovering the higher-scoring solutions found by other runs. Notably, this localization persists despite periodic compaction of the agents' conversational histories.

Can we exploit the variation across independent runs to improve an ongoing run by encouraging it to explore more broadly? To this end, we consider two natural operations that are easy to incorporate into existing autoresearch loops. *Branching* lets the agent try several candidate continuations and select one based on their scores. *Context reset* clears the inherited chat history while retaining the accumulated workspace, allowing the continuations to explore with less dependence on earlier reasoning. Our intervention, *branch-reset*, combines these operations periodically: it forks the agent into parallel trajectories that each inherit a copy of the current workspace and begin with a fresh chat context, runs each for a fixed horizon, and continues from the highest-scoring trajectory.

We evaluate *branch-reset* across 13 long-horizon research and engineering tasks (Table 4). Eleven are drawn from Frontier-CS (Mang et al., 2026) and EdgeBench (Zhu et al., 2026), spanning problems from polyomino packing and job-shop scheduling to cryptanalysis and inferring a groundwater contaminant source from well readings. The remaining two tasks involve designing a compact CIFAR classifier under an accuracy constraint and optimizing a kernel for a simulated VLIW machine. Individual agent runs last from several hours to multiple days depending on the task. Under an equal compute budget, *branch-reset* outperforms both the single-run and best-of-N baselines, achieving 66.0% and 44.4% relative improvements in mim–max normalized average score, respectively. Our artifact embedding also shows broader exploration under branch-reset (Figure 2), connecting the gains to the idea-basin observation.

We summarize our key contributions as follows:

- We show that long-running autoresearch agents exhibit persistent outcome variance: independent runs on the same task plateau at substantially different scores, with gaps that remain even after considerable additional compute.
- We show that a simple intervention, *branch-reset*, improves performance by periodically branching and resetting the chat context while preserving accumulated work. In an evaluation across 13 open-ended tasks, including research and applied engineering challenges with individual runs lasting up to 107 hours, it outperforms single-run and best-of-N baselines under equal compute budgets.
- We develop an embedding of agent artifacts based on functional similarity to study where trajectories search. It reveals localized exploration consistent with *idea basins* and shows broader movement through this space under *branch-reset*.

2 Preliminaries: Autoresearch Setup

Open-ended Research Tasks. We evaluate on thirteen diverse open-ended tasks spanning algorithmic optimization, cryptanalysis, scientific inference, and engineering (Table 4). Nine come from Frontier-CS (Mang et al., 2026) (seven algorithmic tasks, one research task, and one Frontier-CS 2.0 task), and two from EdgeBench (Zhu et al., 2026). The remaining two are larger-budget engineering problems: Anthropic's public performance-engineering take-home exercise (Anthropic, 2026),

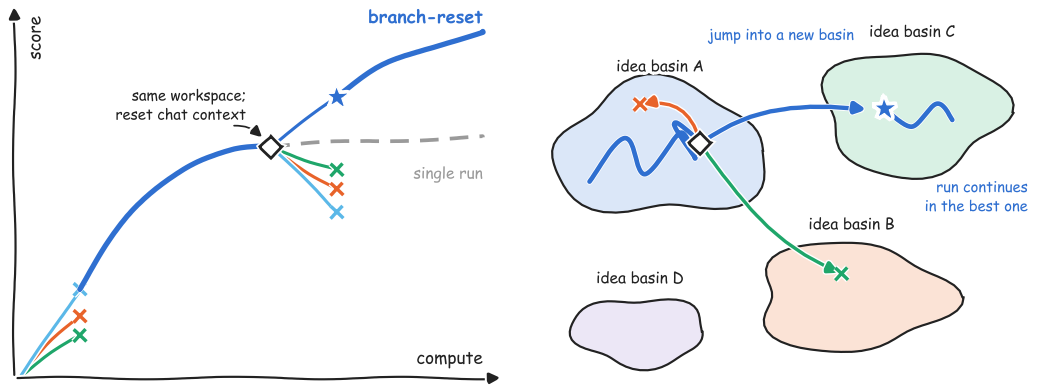

Figure 2: An illustration of *branch-reset*. **Left:** A single run (grey, dashed) keeps exploring similar ideas within one basin. Branch-reset clears the chat context on the same workspace ($\diamond$), fans out K probes, retains the best ($\star$), and discards the rest ($\times$). **Right:** While the leader's edits remain localized, a fresh-context probe jumps to a new basin via branch-reset and continues there.

which involves optimizing a kernel for a simulated VLIW machine, and a custom CIFAR problem that trades classification accuracy against compressed model size. Each task provides a deterministic scorer, but raw scores use task-specific scales and are not directly comparable. Measuring outcome variance and comparing interventions requires multiple independent, long-running trajectories on each task. Per-run budgets range from 6,000 to 100,000 GitHub Copilot CLI credits, and individual runs last from hours to several days. Across repeated runs and ablations, the evaluation consumed approximately 7 million credits, equivalent to about 70,000 USD at 100 credits per dollar.

Autoresearch Harness. An agent harness enables the interaction between an LLM agent and an isolated container environment (Falck et al., 2026). In our autoresearch setup (Karpathy, 2026), the agent's execution proceeds as a sequence of submission episodes including editing, execution, and evaluation in a loop; within an episode, the agent edits its solver, runs it, and submits for a score as often as it chooses. These episodes terminate upon a self-halting action by the agent or by reaching a predefined token budget. Upon the conclusion of an episode, our autoresearch harness resumes the agent via a short continuation prompt. Across these episodes, both the *workspace* W, consisting of the files it has produced, and the *conversational context* C, its accumulated chat history, are strictly preserved, and interventions occur only at episode boundaries. We define a state as $s_i = (W_i, C_i)$ and a *lineage* $\ell = (s_0, \ldots, s_t)$ as the trajectory of these states, starting from an empty base state $s_0 = (\emptyset, \emptyset)$. At each step, a deterministic grader evaluates the workspace to yield a score $f(W_i)$, and we let $J(\ell) = \max_i f(W_i)$ denote the maximum objective score achieved by this lineage. For a given task with a fixed total compute budget B, measured in compute *credits*, a fixed function of input/output model tokens (100 credits $\approx$ US$1), the final result is the maximum score discovered by an autoresearch agent across a long-horizon run $R = \max_{\ell \in \mathcal{L}} J(\ell)$, where $\mathcal{L}$ represents the set of all lineages instantiated by the method within budget B. All conditions use the same coding agent, GitHub Copilot CLI (version 1.0.73), with the same tools, prompt, and task workspace.

3 Idea Basins in Autoresearch

Persistent Variance. We begin by examining independent autoresearch runs on a single task. Figure 1 (left) shows the best score found so far by eight runs on `p171`. Although the runs share the same initial state and underlying model, they plateau for long stretches at substantially different levels. Their final best scores range from 68.3 to 100.0, and the gaps remain wide even after considerable additional compute. The combination of within-run plateaus and persistent between-run variation suggests that the runs differ not just in how quickly they improve, but also in which parts of the solution space they explore. Some appear to spend much of their remaining compute refining less promising approaches without reaching the higher-scoring approaches found by other runs. In

Appendix C, we provide additional plots demonstrating this wide spread of independent runs across all remaining tasks,

Idea Basins. A submission may be a complex program, and its score tells us little about how it relates to other submissions. Can we visualize these relationships to see what an agent explores as its score plateaus? To do so, we ask an LLM to compare artifacts in triplets: given an anchor artifact a and two candidates b and c, which candidate takes an approach more similar to a? We then seek a two-dimensional map that respects these relative comparisons. To construct the triplets, we collect graded submissions, remove exact duplicates by content hash, and subsample up to 51 evenly spaced artifacts per run. Before comparison, we rewrite each artifact as a functional specification under a fixed schema. This reduces cues from code syntax and shared scaffolding that might make submissions from the same run look similar for reasons unrelated to their approach. We fit the map using t-distributed stochastic triplet embedding (t-STE) (van der Maaten & Weinberger, 2012).

The map captures substantial structure in the LLM's similarity judgments. Across five mapped tasks, it agrees with 85% of held-out triplet judgments on average, compared with 50% for chance and 56% for a heuristic that knows only which run produced each artifact. Figure 1 (right) shows the map for `p171`. Submissions from individual runs tend to concentrate in localized regions, which we call *idea basins*. The shaded regions show k-means clusters fitted to baseline artifacts. Later submissions generally remain near the same region, providing a view of the repeated refinement suggested by the score plateaus. To test whether this clustered pattern depends on our choice of t-STE, we repeat the mapping using Soft Ordinal Embedding (SOE) (Terada & von Luxburg, 2014), achieving a mean held-out agreement of 0.83. The full mapping pipeline and the validation of both embeddings are detailed in Appendix E. Although the sharpness of the clusters changes, the localization of individual runs persist. In the following sections, we introduce the *branch-reset* intervention and use the same map to show how it helps trajectories move between idea basins.

4 Escaping Idea Basins via Branch-reset

Problem Setup. Open-ended research problems present a combinatorially large solution space, resulting in significant outcome variance across agent trajectories, as empirically shown in Section 3. This raises a natural question: can we leverage the variance observed across independent runs to help an ongoing trajectory escape its current basin? Committing a global budget B entirely to a single run (pure exploitation) often leads the agent to settle into one basin early and spend the remaining budget on marginal refinements. Conversely, parallel scaling approaches like best-of-N (Snell et al., 2024) represent pure exploration: N independent runs of budget B/N sample diverse basins, but the individual runs remain shallow, and the accumulated workspaces of the $N-1$ discarded runs are completely wasted. Interleaved branching provides a natural interpolation between these extremes. A process that never branches is a single run; one that branches exactly once from an empty workspace and terminates is best-of-N. To actively exploit this variance to escape idea basins, we introduce an approach that alternates between parallel sampling and serial execution, driven by two core mechanisms: branching and context resets.

Branching and Context Reset. Our method relies on two underlying mechanisms to escape idea basins. First, *branching* creates K parallel agents $\ell^1, \ldots, \ell^K$. Each branch inherits the current lineage and is allocated an exploration budget b_{explore}. Upon exhausting this budget, the highest-scoring branch $\ell^\star = \arg\max_k J(\ell^k)$ is selected, and the remaining branches are discarded. Branching places K probes into the search space simultaneously, allowing the selection mechanism to choose between basins rather than refining within just one. Second, a *context reset* retains the accumulated workspace but resets the conversational context: $(W, C) \to (W, \emptyset)$. A branch that retained C would carry the parent's reasoning that tends to re-enter the same local basin. Starting from $(W, \emptyset)$ restores search diversity while preserving the accumulated workspace.

The Branch-reset Algorithm. The proposed algorithm alternates between exploration and exploitation on a fixed schedule as shown in Figure 2. It begins with an initial round of branching from empty workspaces: K agents each receive b_{explore}, and the highest-scoring one is selected as the *leader*. The leader then continues alone until it exhausts an exploitative budget b_{exploit}. At that trigger point, the algorithm initiates a branch coupled with a context reset: K new parallel agents start from the current workspace $(W_t, \emptyset)$, and each is granted b_{explore}. When the round closes, the best branch is selected as the new leader, the others are pruned, and the run continues sequentially

Table 1: Final performance for *single-run*, *best-of-4*, and *branch-reset* (ours) agents evaluated under matched compute budgets. We report the maximum score, mean score, and standard deviation across 5 independent runs (4 for `tinyCIFAR`), with the highest max and mean per row highlighted in **bold**. The aggregate row reports the min–max normalized averages.

	single-run			**best-of-4**			**branch-reset (ours)**		
task	max	mean	sd	max	mean	sd	max	mean	sd
Benchmark tasks: 6,000–25,000 credits per run, hours to several days each									
`polyomino`	94.31	88.56	4.95	94.66	90.73	3.90	**95.29**	**94.41**	0.64
`job-shop`	10.36	9.66	0.87	10.27	9.91	0.24	**10.92**	**10.33**	0.59
`AHC014`	39.35	23.71	9.18	25.11	20.66	3.41	**105.57**	**57.46**	29.35
`p150`	130.80	123.47	4.90	133.25	123.47	5.99	**135.37**	**129.74**	4.43
`p171`	**100.20**	84.26	13.27	99.26	88.88	8.57	98.95	**92.79**	3.39
`sphere-pck`	44.70	43.87	0.83	44.64	43.76	0.99	**45.43**	**44.94**	0.34
`relay-lay`	72.73	72.59	0.12	72.77	72.74	0.04	**72.82**	**72.81**	0.02
`LWE-rec`	**158.0**	**155.3**	2.37	146.00	141.14	10.30	153.00	144.60	9.31
`LLM-SQL`	**27.05**	26.90	0.18	**27.05**	26.93	0.21	**27.05**	**27.04**	0.01
`order-add`	30.29	24.60	4.45	27.23	26.23	1.16	**66.78**	**49.52**	13.60
`Borden-inv`	79.55	64.62	18.12	86.28	76.45	9.47	**93.90**	**85.85**	5.54
Real-world tasks: 31,000–100,000 credits per run, several days each									
`kernel`	132.96	120.36	9.75	127.36	122.50	5.76	**142.46**	**135.23**	4.45
`tinyCIFAR`	0.586	0.429	0.129	0.633	0.560	0.080	**0.682**	**0.572**	0.115
min-max normalized avg.	0.75	0.47	0.26	0.73	0.54	0.21	**0.98**	**0.78**	0.16

from the new trunk. This loop repeats until the global budget B is exhausted. The run's final result is the best score found by any agent throughout the trajectory. In Section 5.3, we conduct ablation studies to isolate the independent contributions of branching and context reset. We opt for a fixed schedule to trigger *branch-reset* rather than dynamic plateau detection; full details on this design choice and its sensitivity are provided in Appendix D.

5 Experiments

5.1 Experimental Setup

Research Tasks. As introduced in Section 2, we evaluate across thirteen environments (Table 4). The nine Frontier-CS environments include classical algorithmic challenges (`polyomino`, `job-shop`, `sphere-pack`, `relay-lay`), AtCoder competitive programming (`AHC014`, `p150`, `p171`), cache optimization (`LLM-SQL`), and cryptanalysis (`LWE-rec`). The two EdgeBench problems target groundwater source recovery (`Borden-inv`) and black-box permutation optimization (`order-add`). Finally, our large-budget engineering tasks consist of Anthropic's VLIW optimization (`kernel`) and custom CIFAR model compression (`tinyCIFAR`). Peerformance is evaluated using verifiable metrics tailored to each domain, such as ratios to reference solvers, simulated cycle counts, and tier-gated judge scores, providing an unbounded signal for continuous improvement. We provide full descriptions of the tasks in Appendix B.

Evaluation Setup. The *single-run* baseline spends the full budget B on one lineage; we execute eight independent single-run baselines per task. The *best-of-N* (N=4) baseline spends $B/4$ on each of four independent runs and keeps the best result. Each task has a fixed per-run budget: 25,000 credits on `polyomino`; 10,000 on `job-shop`, `AHC014`, `p150`, `p171`, and `Borden-inv`; 6,000 on `sphere-pck`, `relay-lay`, `LWE-rec`, `LLM-SQL` and `order-add`; 100,000 on `kernel`; and 31,000 on `tinyCIFAR`. Budgets are set per task to where its single runs plateau, so that the comparison is made in the regime where extra serial compute stops paying. No agent is told its budget, so a run's score at budget B is its best graded score within its first B credits. Within a task every run uses the same agent model: GPT-5.6-terra at high reasoning effort on eleven tasks, and GPT-5.5 at extra-high reasoning effort on `kernel` and `tinyCIFAR`. Each task is evaluated across five independent runs (four on `tinyCIFAR` due to its high computational cost). Four baselines (one on `AHC014`, two on `p150`, one on `p171`) were cut off by an infrastructure interruption at 73–80% of the budget and are excluded. On `relay-lay` every baseline is read from a regrade on

the 20-case judge that graded branch-reset, and `order-add` is read on its official bounded score. We report the maximum, mean, and standard deviation of the final scores over these draws for each task, alongside a min–max normalized average across all thirteen tasks. We provide full descriptions of the tasks in Appendix B.

Branch-reset Configuration. Every branch-reset run reported in Table 1 uses the same schedule, a *forced cadence* with three fan-outs. A run opens with a round of $K{=}4$ agents from empty workspaces; the highest-scoring one becomes the leader. Two further branch-reset rounds are then forced when the run has spent 33% and 66% of its budget B: $K{=}4$ new branches start from the leader's workspace with an empty context, the best becomes the new leader, and the rest are discarded. Rounds are triggered by budget position alone, not by a plateau detector (Appendix D), so every run branches at the same points. Each branch receives a fixed exploration budget b_{explore} of about $0.06B$, recorded with every run; between rounds the leader runs alone. All compute spent on discarded branches counts against B, so every condition spends the same total. These constants were pinned before the study. On `kernel` and `tinyCIFAR`, run first, some runs use a single forced round at 50% instead (Appendix D compares the two).

5.2 Main Results

As shown in Table 1, our *branch-reset* method improves upon standard baselines on the majority of tasks. Out of the 13 evaluated environments, *branch-reset* achieves the highest mean score on 12 tasks and the highest maximum score on 10 (with one three-way tie). Under an equal compute budget, *branch-reset* yields a min–max normalized average score of 0.78, compared to 0.47 for the *single-run* agent and 0.54 for the *best-of-4* baseline, leading to relative improvements of 66.0% and 44.4%, respectively. The performance gap is evident in our larger-budget engineering problems, `kernel` and `tinyCIFAR`, where *branch-reset* achieves the highest maximum and mean scores. On open-ended combinatorial and optimization tasks, the method yields substantial improvements, more than doubling the mean score of the *single-run* baseline on `AHC014` (57.46 vs. 23.71) and `order-add` (49.52 vs. 24.60). Conversely, on tasks where baseline models already achieve near-peak performance, such as `relay-lay` and `LLM-SQL`, all methods perform similarly, resulting in a three-way tie for the maximum score on `LLM-SQL`.

The standard deviations in Table 1 clarify how *branch-reset* alters the search distribution. On tasks where the method reliably identifies a superior idea basin early on—such as `polyomino`, `kernel`, and `p171`—the variance falls compared to the *single-run* baseline. However, on tasks with the largest absolute gains, such as `AHC014` and `order-add`, the variance increases. This indicates that successfully escaping an early local optimum is itself a high-variance event as a jump does not always land, but when it does, it pays off substantially. The high variance observed in the *single-run* baselines elsewhere occurs because single-trajectory agents often exhaust their budget refining a suboptimal approach. These results suggest that exploiting this search variance via branching and context reset enables agents to escape initial idea basins.

5.3 Ablations: Impact of Branching and Resetting

To isolate the contributions of our core mechanisms, we evaluate the impact of branching without resetting the context, and resetting the context without branching across 11 tasks (excluding the two longest-horizon tasks that are expensive to run). In Table 2, we report the performance of all four methods as the min–max normalized final score at the task's budget, averaged across all tasks. The reset-only and branch-only ablations achieve scores of 0.55 and 0.57 respectively, compared to 0.45 for the standard single-run baseline, and 0.71 for the full *branch-reset* method. This demonstrates that while both individual components outperform the single-run baseline, neither mechanism alone accounts for the full effect. In addition, branching provides a slightly larger improvement than context resetting. The complete per-task breakdown is provided in Appendix A.

Table 2: Impact of branching and context reset: min–max normalized mean score over 11 tasks. Best in **bold**, second best in *italics*.

Method	Reset	Branch	Norm. avg
Single-run	×	×	0.45
Context Reset	✓	×	0.55
Branching	×	✓	*0.57*
Branch-reset	✓	✓	**0.71**

6 Analysis

6.1 Visualizing Escaping Idea Basins

Takeaway. Embedding the artifacts of baseline and branch-reset runs jointly, by judged functional similarity, makes the mechanism visible. Single runs each stay inside one idea basin for their whole budget. Branch-reset runs jump between basins and explore several: the context-cleared probes land in different basins and selection carries the run into a new one, which on polyomino ends at 93.9 against a baseline median of 84.6.

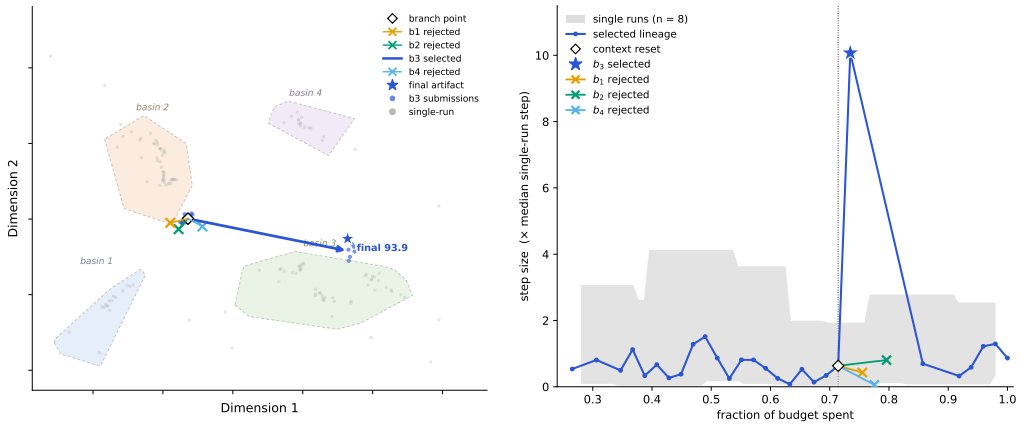

Figure 3: Branch-reset escaping an idea basin on `polyomino`. **Left:** A mid-run fan-out overlaid on baseline basins. Rejected branches ($\times$) from the branch point ($\diamond$) remain trapped, but the selected branch (blue arrow) escapes the idea basin to achieve its final score ($\star$). **Right:** Step distances between consecutive submissions. The selected branch's jump ($\star$) cleanly breaks out of the standard single-run refinement range (grey band).

Visualizing the Jump. We place the artifacts of branch-reset runs on the same geometric maps built in Section 3, embedded jointly with the single-run baselines so that both share an embedding space, enabling branch-reset r to be interpreted as a spatial trajectory across the solution landscape. As illustrated in Figure 3 (left), we overlay the mid-run fan-out of a *branch-reset* trajectory on the `polyomino` task. While leader edits remain highly localized, the intervention successfully forces exploration: branches b_1, b_2, and b_4 remain trapped in the basin of the branch point, but the fresh-context probe b_3 successfully jumps into a new region (Basin 3). This probe is selected by the scoring mechanism, allowing the run to escape the local optimum and finish at 93.9.

Quantifying the Jump. To rigorously measure this jump distance, we utilize SOE (Terada & von Luxburg, 2014) to learn a robust triplet embedding space. We report embedding distance in units of the *median single-run step*, defined as the median displacement between consecutive submissions across all single-run baselines. On `polyomino`, after the first quarter of the budget, no single run moves more than 4.1 units between consecutive submissions, and the branch-reset leader's own sequential steps remain within that localized range. However, at the reset fan-out (triggered at 71% of the global budget), the winning branch (b_3) moves 10.1 units in its very first submission following the context reset, definitively landing in a different cluster. The three rejected branches each move less than one unit. Once established in the new basin, its subsequent refinement steps fall back inside the standard single-run range.

6.2 What Drives Variance?

Takeaway. Outcome variance is highest when a task sits at the frontier of the agent's ability. On a difficulty ladder (MAX-3-SAT), runs agree when the task is trivial or hopeless and disagree most in between, peaking at half the reference solver's attainment. Open-ended research tasks sit at that frontier, which is the variance branch-reset exploits.

Variance at the capability frontier. Our hypothesis is that outcome variance is driven by task difficulty relative to the agent's capacity. When a task sits far below the agent's ability, every run

finds the same optimal solution and the spread collapses. Conversely, when a task sits far above its capacity, every run fails in the same way, collapsing the spread again. It is only in the intermediate regime—at the boundary of what the agent can achieve—that runs make partial, divergent progress, maximizing variance. Open-ended research problems inherently target this shifting frontier, suggesting that no matter how capable models become, boundary problems will always exhibit the variance necessary for selection mechanisms to succeed. To empirically test this hypothesis, we require a domain where difficulty can be dialed continuously, evaluated cheaply, and repeated enough to estimate spread with statistical confidence. Since our thirteen long-horizon research tasks do not permit this, we use MAX-3-SAT on random 3-CNF formulas: the number of variables sets the difficulty, random assignment and a strong classical solver bracket the attainable range, and freshly generated instances rule out memorization.

Empirical evidence from MaxSAT problem. We run three open-weight models (Qwen3.5-4B, 9B, and 27B) on random 3-CNF formulas with V variables, V from 10 to 100,000, and measure the number of unsatisfied clauses u of the best assignment each run produces. We normalize this raw quantity into an attainment metric h, representing the fraction of the performance gap closed between random guessing and a strong classical solver. Specifically, $h = (u_{\text{random}} - u_{\text{agent}})/(u_{\text{random}} - u_{\text{ref}})$, where $u_{\text{random}} \approx 6V/8$ is the expected number of unsatisfied clauses for a uniformly random assignment, and u_{ref} is the minimum u found by a multi-restart WalkSAT baseline. Thus, $h = 0$ indicates performance equivalent to random guessing, while $h = 1$ indicates matching the WalkSAT reference ceiling. We aggregate independent runs into evaluation cells defined by a specific (model, V) pair, with each cell containing between 8 and 32 runs that differ only in their initial seed (32 at most sizes, 8 at the smallest and largest formulas). For each cell, we compute the mean attainment a, and the execution spread s, defined as the population standard deviation. To quantify the uncertainty in s arising from sample size, we compute 95% confidence intervals via bootstrapping, resampling the runs within each cell with replacement 2,000 times. As shown in Figure 4, each point represents a single (model, problem size) pair, plotted according to the fraction of the reference solver's performance the model attains (horizontal axis) and the variance among its runs (vertical axis). Because difficulty is measured relative to model capability rather than absolute problem size, the three models trace a unified curve despite operating at different scales. Variance peaks when a model achieves roughly half of the reference performance, collapsing at both the trivial and intractable extremes. Fitting a least-squares quadratic reveals a strict inverted-U relationship between attainment and spread, peaking at $a = 0.497$. This suggests that the observed variance stems from genuine exploration of distinct idea basins, providing the empirical signal that our *branch-reset* algorithm exploits.

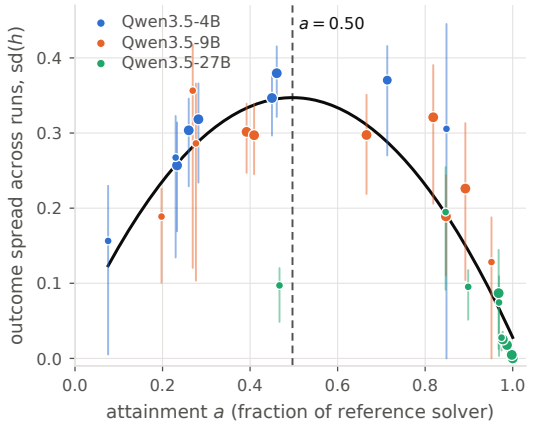

Figure 4: Outcome variance is largest at the frontier of what the model can do. Spread vanishes where every run solves or fails the formula and peaks in between.

7 Related Work

Autoresearch and Self-Improving Agents. Long-horizon agents (Wijk et al., 2024; Chan et al., 2025) are increasingly used to drive open-ended research, from automated paper writing (Lu et al., 2024; Schmidgall et al., 2025) to evolutionary program search (Romera-Paredes et al., 2024; Novikov et al., 2025; Lange et al., 2025) and autonomous experimentation (Boiko et al., 2023; Swanson et al., 2025; Gottweis et al., 2025; Huang et al., 2025). In fact, the research loop proposed by The AI Scientist (Lu et al., 2024; Yamada et al., 2025) automates the idea–experiment–writeup cycle, while AlphaEvolve (Novikov et al., 2025) evolves programs against a grader over time. Beyond a single run, LLM agents typically improve *across* tasks either by accumulating experience (Shinn et al., 2023; Wang et al., 2024; Zhou et al., 2025) or by distilling their own successful trajectories back into the model parameters (Zelikman et al., 2022; Singh et al., 2024; Qu et al., 2024; Falck

et al., 2026). *Branch-reset* is orthogonal to both paradigms as it improves a single run *within* one task, with no memory carried across tasks and no weight updates. The only state that propagates across branches is the workspace, as the intervention exists precisely to clear the conversational context. To evaluate LLM agents in these long-horizon settings, benchmarks such as Terminal-Bench (Merrill et al., 2026) and Terminal-Bench-Science (Terminal-Bench-Science Team, 2026) grade end-to-end terminal workflows on a strict pass/fail basis, while Frontier-CS (Mang et al., 2026), EdgeBench (Zhu et al., 2026), and DiscoGen (Goldie et al., 2026) pose open-ended algorithmic problems with graded, unbounded scores.

Test-Time Compute and Population-Based Search. *Go with the winners* (Aldous & Vazirani, 1994) keeps a population of searchers and periodically replaces its weak members with copies of strong ones. Population Based Training (Jaderberg et al., 2017) applies the same exploit-and-perturb cycle to neural-network training, where a worker copies a better worker's weights and then jitters its hyperparameters. Branch-reset is this scheme over agents: the state copied is a workspace of files rather than a parameter vector, and the perturbation is discarding the conversation that produced those files — the fork must re-derive its intent from the artifacts, which is what lets it land in a different basin. Successive halving and racing algorithms (Karnin et al., 2013) likewise spend rounds of budget on a shrinking survivor set, which is our selection step without the shared trunk; Go-Explore (Ecoffet et al., 2021) archives promising states and returns to them before exploring, the same return-to-the-best-workspace move our fan-out makes. In tree search, MCTS (Kocsis & Szepesvári, 2006) grows a tree by expanding the nodes its interim value estimates favour; Tree of Thoughts (Yao et al., 2023) and Language Agent Tree Search (Zhou et al., 2024) bring that machinery to LLMs at the granularity of individual thoughts or actions. Branch-reset searches the same way but coarser: a node is a whole agent episode-chain over a workspace, expansion is a context-cleared fork, and the value estimate is the task's own grader — the granularity at which long-horizon runs actually produce scores. Finally, sampling-based test-time scaling — AlphaCode's filtered mass generation (Li et al., 2022), repeated sampling (Brown et al., 2024; Snell et al., 2024), and evolutionary program search (Romera-Paredes et al., 2024) — resamples complete solutions from scratch, where branch-reset resamples continuations of the best partial work so far.

Plateaus in Long Reasoning. Before agentic workflows, related stalls were observed in reasoning models that solve problems with long chains of thought. Extending a reasoning model's chain of thought can lower its accuracy rather than raise it (Gema et al., 2025), and long reasoning chains tend to ruminate on problem formulations they have already explored, which obstructs further exploration (Marjanović et al., 2025). Reasoning models also fall into loops that repeat the same text (Pipis et al., 2026), which can be read as an extreme case of the same behaviour: the model keeps returning to what it has already produced. Our agents plateau in the same way, but the setting differs in one telling respect: their conversational histories are compacted periodically, so the context does not simply grow, yet runs still settle into one idea basin. Branch-reset treats that state directly, keeping the workspace and discarding the history.

8 Conclusion

In this work, we introduced *branch-reset*, a periodic intervention designed to help autonomous research agents escape localized regions of the solution space, which we term idea basins. We demonstrated that as agents tackle open-ended problems by repeatedly proposing, evaluating, and refining candidate solutions, independent runs often plateau at substantially different scores—a high-variance bottleneck that persists even with considerable additional compute. To overcome this, *branch-reset* forks the agent into parallel trajectories that each inherit a copy of the accumulated workspace but begin with fresh chat contexts. After exploring for a fixed horizon, the agent continues from the highest-scoring branch. Across thirteen long-horizon research and engineering tasks under an equal compute budget, *branch-reset* consistently outperforms serial baselines, demonstrating that branch-reset provides a practical path for scaling test-time compute in autonomous agents. We present exciting avenues for future work. First, the substantial computational expense and time required to evaluate frontier LLMs on long-horizon agentic workflows inherently limit the number of tasks we could practically include in this study. Finally, the branched trajectories currently operate independently without coordination. Future work will investigate inter-agent communication and role specialization across branches, as well as scaling our evaluations to a broader suite of tasks and newly released frontier models.

AI USE STATEMENT

We used generative AI tools to create and edit software code, modify scientific figures, format LaTeX tables and references, and edit the research paper to improve readability. We have not used generative AI tools to generate synthetic datasets, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or interpret results. The remaining required disclosure tasks (e.g., developing theoretical models, formulating mathematical claims, assisting with translation, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis) are not applicable to this work. We have carefully reviewed all AI-assisted work. All AI-assisted text, code, and mathematical content were reviewed and verified by the authors. We take full responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

AUTHOR CONTRIBUTIONS

If you'd like to, you may include a section for author contributions as is done in many journals. This is optional and at the discretion of the authors.

ACKNOWLEDGMENTS

Use unnumbered third level headings for the acknowledgments. All acknowledgments, including those to funding agencies, go at the end of the paper.

REFERENCES

David Aldous and Umesh Vazirani. "go with the winners" algorithms. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 492–501, 1994.

Anthropic. Original performance engineering take-home. `https://github.com/anthropics/original_performance_takehome`, 2026. Kernel-optimization exercise on a simulated VLIW machine.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624:570–578, 2023.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590:580–586, 2021.

Damon Falck, Samer Sabri, Anja Surina, Thom Foster, Anya Sims, Sam Devlin, Dylan Rogers, Tantum Collins, Kaloyan Aleksiev, Louis Kirsch, and Edward Hughes. Training ai scientists to replicate research. *arXiv preprint arXiv:2608.13331*, 2026.

Aryo Pradipta Gema, Alexander Hägele, Runjin Chen, Andy Arditi, Jacob Goldman-Wetzler, Kit Fraser-Taliente, Henry Sleight, Linda Petrini, Julian Michael, Beatrice Alex, Pasquale Minervini, Yanda Chen, Joe Benton, and Ethan Perez. Inverse scaling in test-time compute. *Transactions on Machine Learning Research*, 2025. arXiv:2507.14417.

Alexander D. Goldie, Zilin Wang, Adrian Hayler, Deepak Nathani, Edan Toledo, Ken Thampiratwong, Aleksandra Kalisz, Michael Beukman, Hannah Erlebach, Alistair Letcher, Shashank Reddy, Clarisse Wibault, Theo Wolf, Charles O'Neill, Uljad Berdica, Nicholas Roberts, Saeed Rahmani, Roberta Raileanu, Shimon Whiteson, and Jakob N. Foerster. DiscoGen: Procedural generation of algorithm discovery tasks in machine learning. *arXiv preprint arXiv:2603.17863*, 2026.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, et al. Towards an AI co-scientist. *arXiv preprint arXiv:2502.18864*, 2025.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, et al. Biomni: A general-purpose biomedical AI agent. *bioRxiv*, 2025.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning (ICML)*, 2013.

Andrej Karpathy. autoresearch. `https://github.com/karpathy/autoresearch`, 2026. GitHub repository, March 2026.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, 2006.

Robert Tjarko Lange, Qi Sun, Aaditya Prasad, Maxence Faldor, Yujin Tang, and David Ha. Towards robust agentic CUDA kernel benchmarking, verification, and optimization. *arXiv preprint arXiv:2509.14279*, 2025.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Qiuyang Mang, Wenhao Chai, Zhifei Li, et al. FrontierCS: Evolving challenges for evolving intelligence. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2512.15699.

Sara Vera Marjanović, Arkil Patel, Vaibhav Adlakha, Milad Aghajohari, Parishad BehnamGhader, Mehar Bhatia, Aditi Khandelwal, Austin Kraft, Benno Krojer, Xing Han Lù, Nicholas Meade, Dongchan Shin, Amirhossein Kazemnejad, Gaurav Kamath, Marius Mosbach, Karolina Stańczak, and Siva Reddy. DeepSeek-R1 thoughtology: Let's think about LLM reasoning. *Transactions on Machine Learning Research*, 2025. arXiv:2504.07128.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026. `https://www.tbench.ai`.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Charilaos Pipis, Shivam Garg, Vasilis Kontonis, Vaishnavi Shrivastava, Akshay Krishnamurthy, and Dimitris Papailiopoulos. Wait, wait, wait... why do reasoning models loop? In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2512.12895.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2407.18219.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, et al. Agent laboratory: Using LLM agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2303.11366.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. arXiv:2312.06585.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

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*, 2025.

Yoshikazu Terada and Ulrike von Luxburg. Local ordinal embedding. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *PMLR*, pp. 847–855, 2014.

Terminal-Bench-Science Team. Terminal-Bench-Science: Evaluating AI agents on research workflows across scientific domains, 2026. URL `https://github.com/harbor-framework/terminal-bench-science`.

Laurens van der Maaten and Kilian Weinberger. Stochastic triplet embedding. In *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2012.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. arXiv:2305.16291.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, et al. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2203.14465.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning (ICML)*, 2024.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guiyao Zhang, Kun Shao, Liyuan Yang, and Jun Wang. Memento: Fine-tuning LLM agents without fine-tuning LLMs. *arXiv preprint arXiv:2508.16153*, 2025.

Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, Zixin Wen, Zhonglin Xie, Chenhui Gou, Linxuan Ren, et al. Edgebench: Unveiling scaling laws of learning from real-world environments. *arXiv preprint arXiv:2607.05155*, 2026.

A Ablation: Full Results

Table 3 is the per-task form of the ablation summarised in Section 5.3. Each entry is the mean final score of that method at the task's budget, on the task's own scale, so the four columns are directly comparable within a row but not across rows.

Table 3: Ablation on the impact of branching and resetting: mean final score of each method at the task's budget, over the eight single-run baselines (seven on AHC014, p171 and LWE-rec, six on p150) and each method's five draws (four reset-only cells on LLM-SQL; *reset only*: $K=1$ with a context reset at each forced round; *branch only*: $K=4$ branches that continue the trunk's context). Scores are on each task's own scale, so rows are not comparable with each other; the last two rows min–max normalize every run of a task over all four methods before averaging. Best per row in **bold**, second best in *italics*. The ablation is run at 6,000 credits on `Borden-inv`, and every column there is read at that budget.

task	single-run	reset only	branch only	branch-reset
`polyomino`	88.56	85.91	*90.74*	**94.41**
`job-shop`	9.66	10.00	**10.43**	*10.33*
`AHC014`	23.71	23.04	*24.59*	**57.46**
`p150`	123.47	**160.80**	128.57	*129.74*
`p171`	84.26	79.73	*86.68*	**92.79**
`sphere-pack`	43.87	43.74	*44.36*	**44.94**
`relay-lay`	72.61	*72.80*	72.79	**72.81**
`LWE-rec`	**155.3**	*148.4*	147.0	144.6
`LLM-SQL`	26.90	*27.01*	26.93	**27.04**
`order-add`	24.60	*40.73*	30.09	**49.52**
`Borden-inv`	59.92	65.85	*78.02*	**80.24**
normalized avg. (5 original tasks)	0.37	0.41	*0.47*	**0.63**
normalized avg. (all 11 tasks)	0.45	0.55	*0.57*	**0.71**

B TASK DESCRIPTIONS

Table 4 lists every task in the study with its source, the problem the agent optimizes, the score the judge reports, and the credit budget. Sources are Frontier-CS (FCS; the algorithmic track is cited by problem number, plus its 2.0 and research tracks), EdgeBench, Anthropic's public performance-engineering take-home, or a custom task written for this study. Every task has eight independent single-run baselines and five branch-reset seeds (four on tinyCIFAR). A credit meters model tokens, not wall-clock.

C VARIANCE ANALYSIS PLOTS

Figure 5 illustrates eight independent single-run trajectories across four embedded tasks. On high-variance problems such as `polyomino` (final scores 82.2–94.9), `p150` (109.1–137.1), and `AHC014` (7.8–59.8), the runs clearly separate into distinct basins that track their final performance. For instance, on `AHC014`, the two highest-scoring runs share a single basin while all remaining runs fall into another. In contrast, on a low-variance task like `job-shop` (7.2–10.8), the geometric structure is weak: the two basins overlap heavily, and runs freely move between them. Because this basin structure is poorly defined, we draw no basin-based claims from `job-shop`.

D SCHEDULE DESIGN

Fixed Schedule vs. Plateau Detection. The natural trigger for a branch-reset round is a performance plateau. However, early development runs using a stall-clock trigger rarely executed the intervention: across five tasks, 15 of 25 plateau-triggered runs exhausted their budget before the stall criteria were met. Dynamic triggers proved practically difficult to tune for three reasons. First, credit costs per step vary widely (e.g., $6.1\times$ variance across `tinyCIFAR` runs), meaning a fixed credit threshold detects a stall prematurely in some runs and entirely misses it in others. Second, tasks often exhibit micro-improvements that restart the stall clock. This allows runs creeping upward within a narrow local basin to evade detection—the exact failure mode branch-reset is designed to fix. Third, requiring a sufficiently long stall window (e.g., 20% of the budget) to ignore noise pushes the first intervention too late in the trajectory. A fixed schedule resolves all three issues by guaranteeing the intervention fires, standardizing the branch spend, and removing task-dependent tolerance tuning.

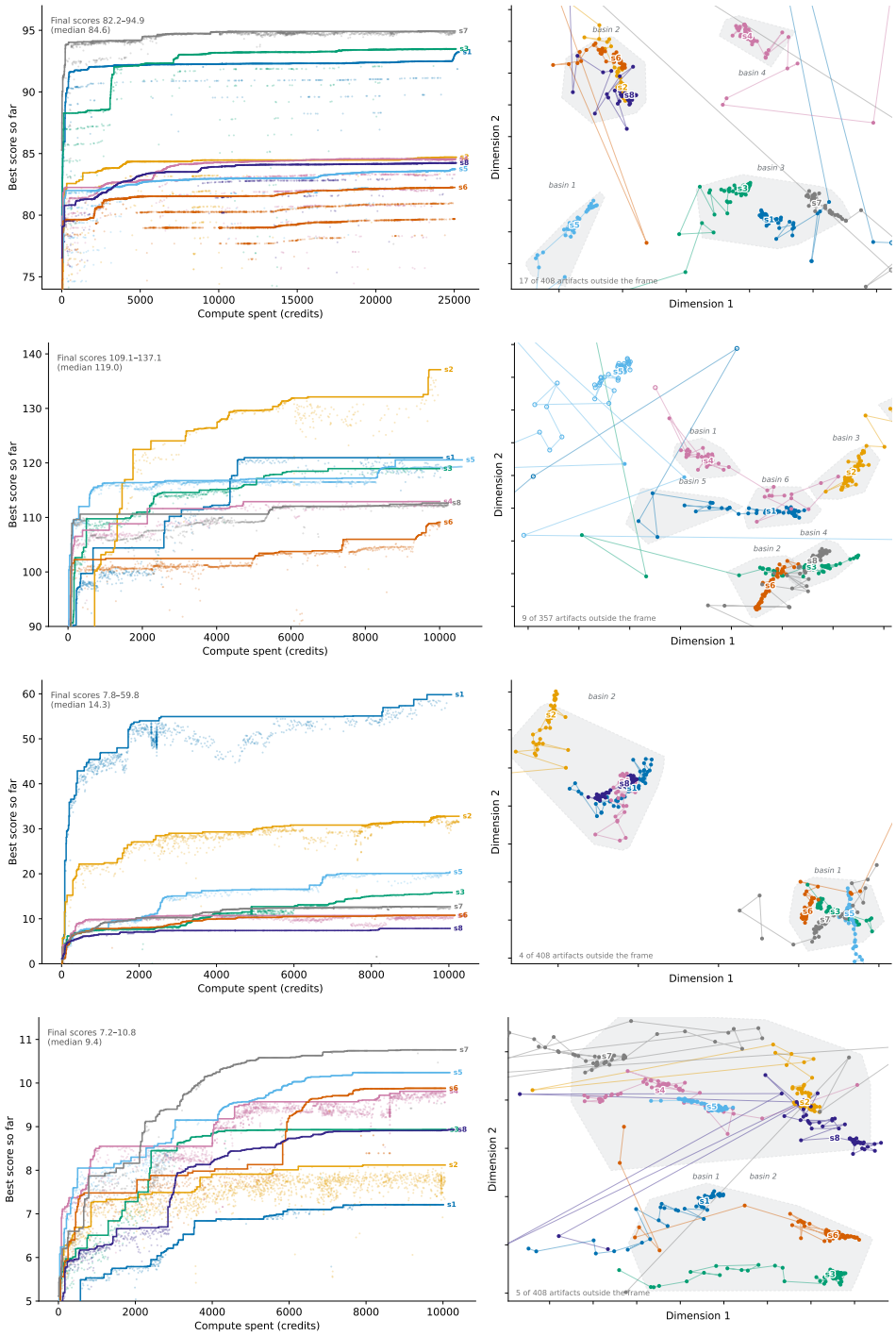

Figure 5: Independent single runs across four embedded tasks: `polyomino`, `p150`, `AHC014`, and `job-shop`. **Left:** Best score achieved versus compute spent. **Right:** Run artifacts embedded by judged functional similarity, with k-means basins shaded.

Table 4: The thirteen tasks. "Credits" is the per-run budget, set per task to where single runs plateau. On `tinyCIFAR` it is the length of the shortest single-run baseline. "Hours" is the median time a single-run baseline takes to reach the budget, counting only time the agent is working and excluding pauses between resumed legs. Most of it is the agent's own solvers and training jobs, which consume no credits; on LWE-rec and order-add single episodes of this kind run for up to two days.

task	source	problem	score	credits	hours
`polyomino`	FCS algo 0	Pack a given set of polyominoes into a bounding rectangle of minimal area (C++ heuristic).	Ratio to a reference solution.	25k	29
`job-shop`	FCS algo 46	Job-shop scheduling: order operations on machines to minimize the makespan.	Ratio to known bounds.	10k	15
`AHC014`	FCS algo 159	Port of AtCoder Heuristic Contest 014.	Ratio to a reference solution.	10k	68
`p150`	FCS algo 150	Port of an AtCoder Heuristic Contest problem.	Ratio to a reference solution.	10k	70
`p171`	FCS algo 171	Port of an AtCoder Heuristic Contest problem.	Ratio to a reference solution.	10k	80
`kernel`	Anthropic take-home (Anthropic, 2026)	Anthropic's performance-engineering take-home: optimize a kernel for a simulated VLIW machine; a deterministic simulator counts cycles.	Simulated-cycle score, higher is better.	100k	64
`tinyCIFAR`	custom	Train a CIFAR classifier that trades test accuracy against compressed model size (GPU).	Merit in $(0, 1)$: an accuracy gate times a log-size term, read on a sealed test split.	31k	45
`sphere-pck`	FCS algo 48	Pack equal spheres in a cube (continuous geometry).	Normalized between a grid baseline and the Kepler bound.	6k	20
`relay-lay`	FCS algo 307	Mobile relay layout: K-hub facility location with up to 2×10^5 clients.	$10^6\, C_{\text{base}}/(C_{\text{base}} + C)$, unclipped above the baseline.	6k	6
`LWE-rec`	FCS 2.0	Cryptanalysis: recover the secret of 200 structured-LWE instances drawn from ten families of graded difficulty.	Number of instances solved.	6k	83
`LLM-SQL`	FCS research	Reorder the rows and columns of an LLM-over-SQL workload to maximize the prefix KV-cache hit rate (deterministic objective).	$(\text{hit} - \text{base})/(1 - \text{base})$; 100 is a perfect hit rate.	6k	23
`order-add`	EdgeBench (optimization)	Black-box permutation optimization over m=1000 elements against a cost oracle (statistical experimental design); no statement to pattern-match.	Official judge score from the rescaled cost, at most 100.	6k	107
`Borden-inv`	EdgeBench (scientific)	Scientific inverse problem: recover a 3D groundwater contaminant source from well readings; the agent writes both the forward model and the inverter.	Tier-gated judge score.	10k	5

Sensitivity to Cadence. While we did not systematically sweep all schedule parameters, our development runs span three configurations (Table 5): the plateau trigger, a single forced round at 50% of the budget, and our main setting of two forced rounds at 33% and 66%. Both fixed schedules outperform the single-run baseline on all six tested tasks, with neither strictly dominating (the two-

round schedule wins on four tasks, the one-round on two). The plateau trigger performed poorly on `AHC014` and `p171` primarily because it never fired. Though sample sizes are small, this suggests the performance gains are robust to the exact cadence of the fixed schedule.

Sensitivity to Branch Count. We set the branch count to $K{=}4$. Because the branch spend per round is fixed at $K\, b_{\text{explore}}$, increasing K requires either drastically reducing the exploration budget per branch or running fewer rounds entirely. We found that $K{=}4$ strikes a practical balance, allocating roughly 6% of the total budget to each branch—enough depth to meaningfully develop a new idea—while preserving enough global budget for multiple rounds of intervention. The minimal case of $K{=}1$ corresponds to the reset-only ablation discussed in Section 5.3.

Table 5: Branch-reset under three schedules, development runs ($K{=}4$). We report the mean final score at the task's budget, with the number of runs in parentheses. The single-run column is the mean of the eight baselines from the same period.

task	single-run	plateau trigger	one round at 50%	rounds at 33/66%
`polyomino`	87.54	93.84 (1)	**94.43** (1)	93.82 (6)
`job-shop`	9.23	**11.44** (1)	10.24 (2)	10.99 (2)
`AHC014`	21.32	21.30 (3)	26.98 (4)	**40.97** (5)
`p150`	118.24	126.72 (4)	**145.13** (2)	135.52 (7)
`p171`	83.70	80.16 (4)	85.83 (5)	**89.11** (11)
`kernel`	120.36	–	132.11 (5)	**136.77** (2)

E Artifact Embedding Methodology

Each idea-space map is produced in four stages:

(i) Subsampling. From every trajectory we retain the exact graded submission artifacts, deduplicate identical files by content hash, and select a fixed number of evenly spaced snapshots per run in order of first occurrence (51 per run on every mapped task).

(ii) Specification. An LLM rewrites each sampled submission into a canonical functional specification under a fixed section schema with an exhaustive parameter table. The rewrite strips coding syntax and stylistic signal—formatting, naming, how the file is organized—while preserving the algorithm, data structures, and every numeric choice. This ensures that a competent engineer who has never seen the file could reimplement it from the specification alone and obtain materially the same behavior and score. Judging specifications rather than raw source keeps two artifacts edited from a common ancestor from being called similar merely because they *read* alike.

(iii) Triplet Comparison. An LLM judge answers ordinal queries—is anchor a's approach closer to b's or to c's?—over the specifications, mixing within-run triplets with cross-condition anchors balanced across conditions. The judge is the same LLM deployment that wrote the specifications, so a single model accounts for every generated token in the pipeline, and each map is built from 6,375 judgments (5,610 on `p150`).

(iv) Embedding. We fit a two-dimensional t-STE embedding (van der Maaten & Weinberger, 2012) to the judged rankings, keeping the best of eight random restarts by training agreement. One in five judged triplets per anchor is held out and never used in fitting; Table 6 reports agreement on those held-out judgments against chance and against a rule that answers from run membership alone, picking whichever candidate shares the anchor's run and abstaining, credited 0.5, when both or neither do. The *undecided* column is the map's agreement on exactly the held-out triplets that rule cannot decide, about three quarters of them on every task.

Validation. Each map is fit on 80% of the judged triplets and scored on the remaining 20%; agreement (Table 6) is the fraction of held-out judgments the two-dimensional coordinates reproduce, with chance at 0.50. The t-STE maps reach 0.81–0.87 (mean 0.85) with a train–test gap of 0.03–0.05, so they generalise rather than memorise. Refitting the same triplets with SOE, whose hinge loss does not inflate the gaps between clusters, gives 0.79–0.86 (mean 0.83): a slightly weaker fit, so t-STE is used for the figures and SOE as the check that the displacements are not artefacts of its heavy-tailed kernel. Agreement is lowest on job-shop, the task with the least outcome variance and

Table 6: Held-out validation of the idea-space maps. Agreement is the fraction of LLM triplet judgments the fitted two-dimensional map reproduces; chance is 0.50. t-STE is the embedding used in the figures; SOE refits the same triplets with soft ordinal embedding.

		t-STE		SOE	
task	judged triplets	train	held out	train	held out
`polyomino`	6,375	0.90	0.87	0.90	0.85
`AHC014`	6,375	0.90	0.87	0.91	0.86
`p171`	6,375	0.88	0.85	0.88	0.84
`p150`	5,610	0.87	0.83	0.86	0.80
`job-shop`	6,375	0.86	0.81	0.85	0.79
mean		0.88	0.85	0.88	0.83

the weakest basin structure. Run membership alone ("artifacts from the same run are closer") reproduces only 0.52–0.60 of held-out judgments, so the maps capture more than which run an artifact came from.

F BRANCH-RESET TRAJECTORIES

Figure 6 shows the trajectories of the branch-reset runs in Table 1, one row per task. Each panel plots score against credits up to the task's budget, the point at which the table reads it. The grey band is the p_{10}–p_{90} range of the same single-run baselines the table uses, and the grey line is their median. The blue line is the run's best score so far. Each branch is drawn in its own colour from the score it inherited at the fork, and every graded submission is a dot. A star marks the branch adopted when a round closes, a cross each branch that was dropped, and the shaded columns span the branch rounds. On `kernel` and `tinyCIFAR`, some runs use a single forced round at 50% of the budget (Section 5.1).

G TRAJECTORIES ACROSS IDEA BASINS

To further illustrate how the intervention navigates the solution space, Figure 7 visualizes nine distinct *branch-reset* trajectories mapped onto their respective task's idea basins. The embeddings reveal three distinct search behaviors. First, during the opening round, initial probes frequently scatter across multiple basins (e.g., `p171-fd2`, `ahc014-f2`), allowing the agent to evaluate fundamentally different structural approaches before committing. Second, later in the run, context resets can trigger a definitive jump into a new basin (as seen in `poly-d4`), enabling the run to successfully escape a local optimum. Finally, even when the context reset does not induce a basin jump—and both rejected and selected branches remain localized near the fork (e.g., `ahc014-t1`)—the intervention still drives robust local refinement, consistently driving the trajectory to final scores that surpass the single-run baselines.

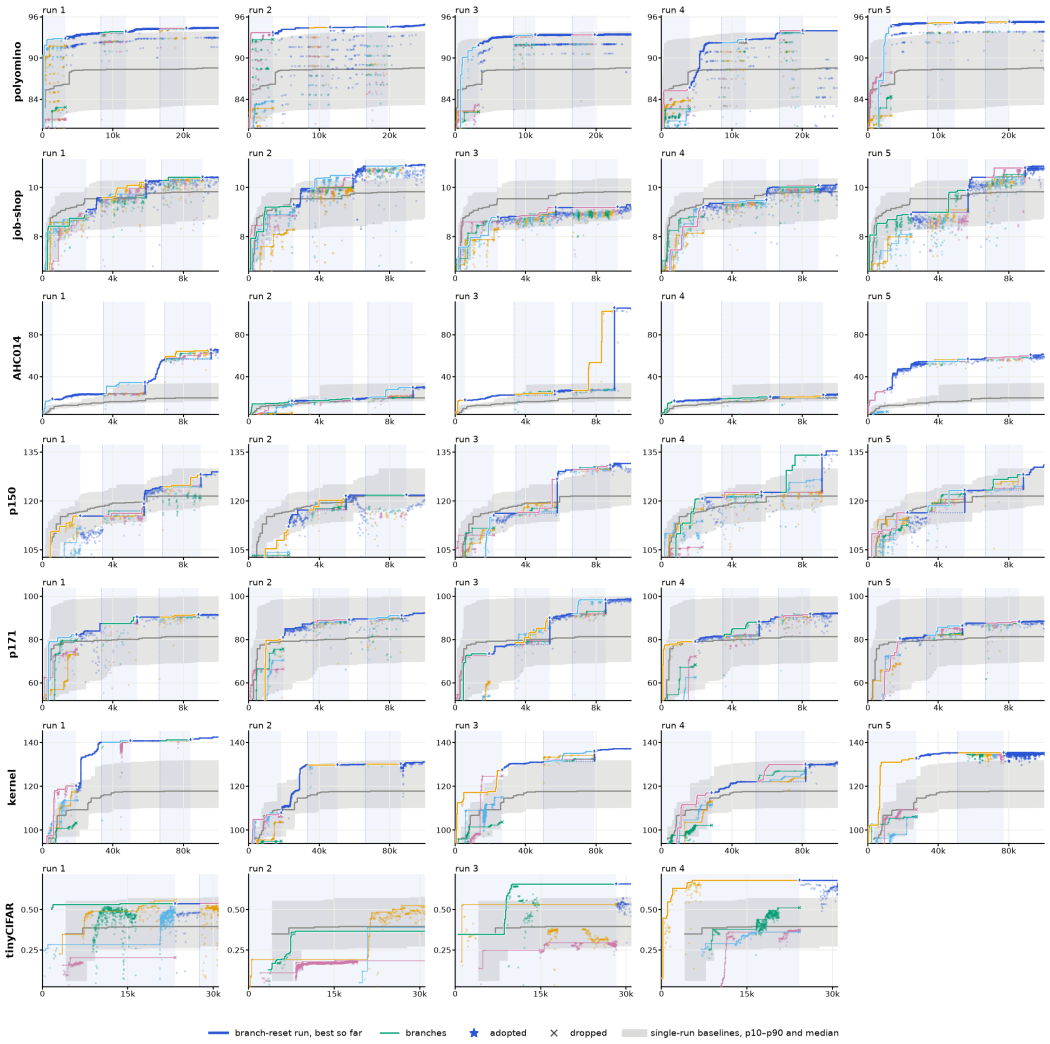

Figure 6: Branch-reset trajectories against the single-run baselines of each task.

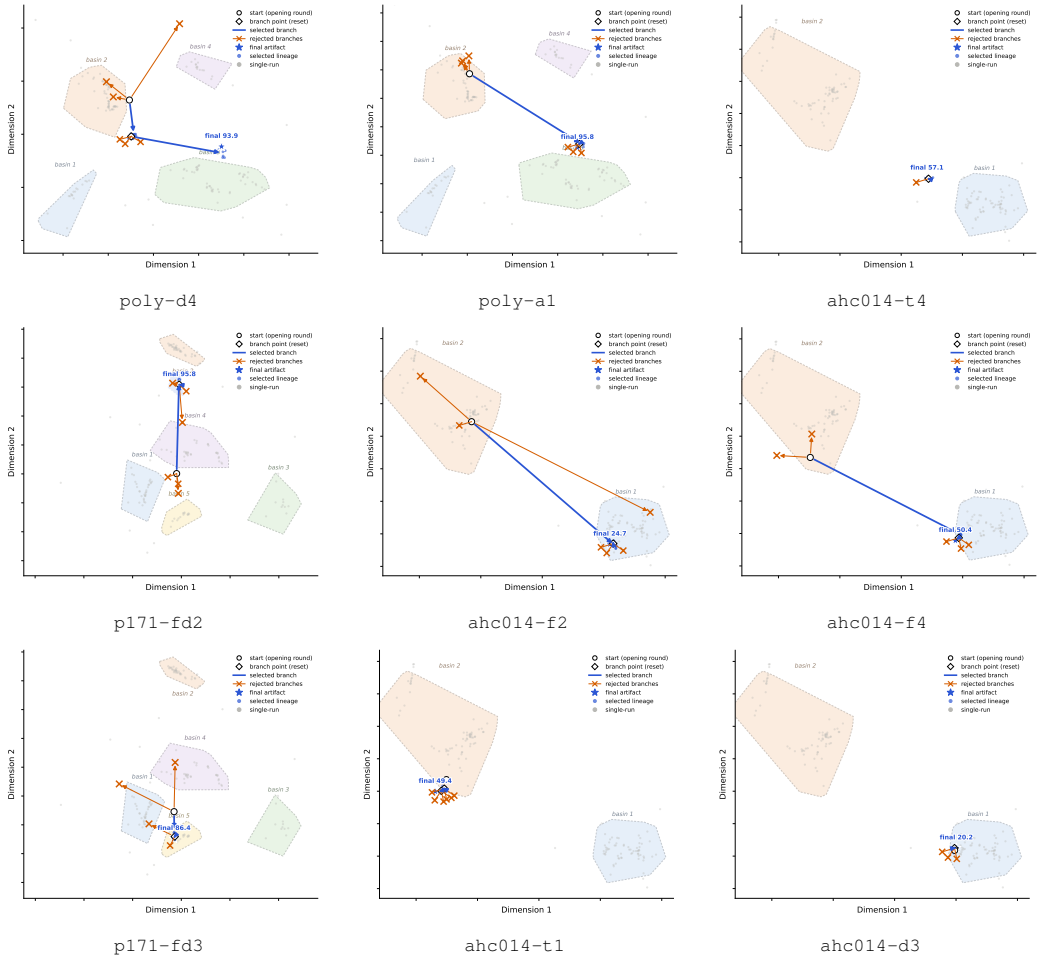

Figure 7: Nine branch-reset runs on their tasks' idea maps (task-map basins shaded). Blue: the selected path (○ opening round, ◇ context reset, ⋆ final artifact); vermillion ×: rejected branches.

H PROMPTS

H.1 AGENT PROMPTS

Every message an agent receives is either the task's own packaging (its statement, its `AGENT.md` workflow notes and its submission script) or one of the texts below, which we wrote. No message in any condition states the credit budget or how much of it remains; agents are never told how much compute they have. The episode messages are logged verbatim with every run.

Continuation message.. Sent at the start of every episode after the first, in every condition (single-run, best-of-4 draws, branch-reset trunks and branches), whenever the harness resumes a session:

> Continue — this task is open-ended and continuously scored; there is no fixed 'done'. Keep working in /app. Do not stop.

Branch directive.. Appended to the first message of a directed branch in a branch-reset round. In each round two of the K branches carry it; the others start from the same workspace with no directive:

> Before continuing, deliberately pick a DIFFERENT approach from the one currently implemented. Do not spend this session incrementally tuning what is already there. Ask whether a fundamentally different algorithm, data structure, or representation would do better, and if it would, implement that instead. It is acceptable for the measured result to get worse before it gets better.

The directive does not measurably change which branch wins: across the 178 mid-run rounds of the study, a directed branch was selected in 81 (46%), against the 50% expected if it had no effect (two-sided binomial p=0.26). We therefore treat the directed and undirected branches as interchangeable.

Protected-incumbent notice.. Appended to the first message of any branch, or any resumed leg, that inherits a workspace containing a scored submission. It exists because task scaffolds can overwrite the canonical solution file, and it is the only place a score is quoted to the agent:

> — PROTECTED LINEAGE INCUMBENT —
> The inherited lineage-best completed submission scored {score} and came from episode {episode}. Its verified read-only artifact is preserved at {protected path} and has been materialized at the canonical path {canonical path}. Scaffold, baseline, or bootstrap scripts can overwrite canonical files under /app. Preserve the inherited result or improve it by recovering from the protected artifact; do not silently replace it with the baseline.
> — END PROTECTED LINEAGE INCUMBENT —

Task wrappers.. For the EdgeBench tasks the first message is the text below, followed by the original EdgeBench agent query:

> You are solving a pinned EdgeBench task in `/app`. The upstream work image's task workspace has been copied to `/app`. Use `bash /app/submit.sh` for asynchronous evaluation. The submitted artifact is an exact tar.gz snapshot filtered by the upstream task spec; the hidden judge runs in an isolated disposable copy.

For `LLM-SQL` it is the text below, followed by the original task statement:

> You are solving the pinned Frontier-CS research task llm_sql/large. Create `/app/solution.py`. `Solution.solve()` may apply the supplied column merges and then change only column order: row order, index, values, dtypes, and exactly-once column membership must remain unchanged. Use `bash /app/submit.sh` for asynchronous black-box evaluation.

For `sphere-pack` we replaced the packaging's pass/fail scoring line with the first text below and added the second to its `AGENT.md`:

> Scoring is continuous: the mean normalized packing ratio across 15 cases; the judge reports the uncapped RatioUnbounded value

> Use fixed random seeds and fixed iteration counts. Do not seed from the clock or terminate search by wall time: exact source artifacts are replayed and must produce the same score.

All other instruction text is the task's own packaging.

H.2 IDEA-MAP PROMPTS

Its prompt is one template in which a task sentence and a list of the task's typical mechanisms are substituted per task; the `polyomino` instance is:

```
You are comparing three submitted solutions (A, B, C) to the
same polyomino-packing optimization task, where a submission is a self-contained C++
program that packs sets of polyomino pieces into a rectangular grid to minimize the
enclosed area, scored continuously on how compact its packings are across many problem
instances. Rank the THREE PAIRS -- AB, AC, BC --
by how similar the two members of each pair are in their IMPLEMENTED
OPTIMIZATION STRATEGY, from most similar to least similar.

Judge similarity primarily by computational mechanisms: piece-orientation and symmetry
enumeration, placement search with pruning, width and height selection under area
bounds, portfolio and heuristic construction, backtracking and beam strategies, and
other algorithmic choices.
Ignore naming, comments, formatting, and incidental textual overlap. Do not use
score or chronology; neither is provided. You must produce a strict ordering of
all three pairs even when they are close.

Return STRICT JSON only:
{"order":["AB","AC","BC"],
  "reason":"explain as fully as you need, naming the decisive shared and
            differing mechanisms that place each pair where it is in the order"}
where "order" lists all three pairs from MOST similar to LEAST similar.

=== SUBMISSION A ===
<specification A>

=== SUBMISSION B ===
<specification B>

=== SUBMISSION C ===
<specification C>
```

Each task has its own specification prompt with the same eleven-section schema (objective, overall strategy, input handling, representation, the search itself, task-specific selection rules, portfolio and tie-breaking, data structures, control flow and compute budget, an exhaustive parameter table, and techniques not implemented); only the task description and the per-section examples change. The `polyomino` prompt is:

```
You are writing a canonical FUNCTIONAL SPECIFICATION
of one submitted solution to a polyomino-packing optimization task. The task is
to place a given multiset of polyomino pieces (each an orthogonally-connected
set of unit cells) into a rectangular grid without overlaps, minimizing the
enclosed area (width times height); a submission is a self-contained C++ program
scored continuously on how compact its packings are, aggregated across many
problem instances of different sizes.

The specification has one purpose: a competent engineer who has never seen this
file must be able to reimplement it from your text alone and obtain materially
the same behaviour -- the same search strategy and materially the same packing
quality (area / score). Completeness is the priority. There is no length limit;
do not summarise, do not economise, and do not omit a decision because it seems
minor.

Two solutions that implement the same thing must receive near-identical
specifications even if the files look nothing alike, and two solutions that
implement different things must receive visibly different specifications even
if the files look almost identical. Everything below serves that goal. In this
corpus the differences between solutions are usually in the SEARCH HEURISTICS
(placement order, candidate scoring, tie-breaking, which grid widths are tried,
lookahead depth, portfolio of retried configurations), so those must be captured
in exact detail; a difference there is a difference in the solution.
```

```
ABSOLUTELY EXCLUDE -- these are properties of the file, not of the solution:
identifier, function, class, method and variable names; how the code is split
into functions, lambdas, structs or modules; the order definitions appear in;
comments, docstrings, logging and their wording; formatting, style and code
quality; unreachable, unused, vestigial or commented-out code; anything about
the author, the file's history, or how it was developed. Never quote the source.
Refer to every component by its ROLE.

ALWAYS INCLUDE -- these are properties of the solution: every algorithmic
decision, every data-layout decision, and EVERY quantitative choice with its
exact value. Numeric parameters -- candidate widths and how they are chosen,
lookahead / beam widths, area or bounding-box bounds, piece-ordering keys,
tie-break selectors, thresholds keyed on instance size, time or iteration
limits, magic constants in scoring functions -- are FUNCTIONAL CONTENT and must
all be reported with their value and their role. A reimplementation that guessed
these would not reproduce the packing quality.

Emit exactly the following sections, in exactly this order, with these exact
headings, even when a section is empty. Never reorder them to match the file.
If the solution does not do something a section asks about, write
"Not implemented." for that section rather than omitting it -- the absences are
as diagnostic as the presences.

## 1. OBJECTIVE
The computation the program must produce and how it is scored, stated
independently of how (pack the given pieces into a grid without overlap;
minimize enclosed area; scored on compactness across instances).

## 2. OVERALL STRATEGY
The core idea in 3-6 sentences: what this solution does that a naive one does
not (e.g. greedy skyline placement over several candidate widths, exact/backtrack
search on small instances, a portfolio of retried heuristics keeping the best),
and why that produces tighter packings.

## 3. INPUT PARSING AND INSTANCE HANDLING
How the instance is read (number of pieces, per-piece cell lists); whether
multiple instances or sizes are handled and how they are dispatched; any
size-based branching that selects different strategies or parameters for small
versus large instances, with the exact size thresholds.

## 4. PIECE REPRESENTATION AND ORIENTATION ENUMERATION
How each piece is stored; how orientations are generated (rotations, reflections,
how many, and how symmetric duplicates are removed); the per-orientation derived
data computed (bounding box, per-row cell counts, row bitmasks, bottom-contour
cells) and what it is used for.

## 5. PLACEMENT AND PACKING SEARCH
The packing procedure: how the occupied region is represented, how a legal
position for a piece is found (bottom-left / skyline / column-scan / bitmask
collision test), the order pieces are placed, whether placement is greedy,
backtracking, beam or exact, and any lookahead and its depth. Give the exact
scoring/selection rule for choosing among candidate placements. State explicitly
whether the search is DETERMINISTIC or RANDOMIZED: if randomized, report the use
of a seeded RNG, random restarts, shuffled piece/candidate orders, or
sampling-and-keep-best, the number of restarts or samples, and how the best
result is retained -- deterministic versus randomized-best-of-N is a first-order
difference between solutions in this corpus and must never be left implicit.

## 6. WIDTH, HEIGHT AND AREA SELECTION
Which grid widths are tried and how that set is derived (area lower bounds, piece
spans, enumerated ranges); how height results from a packing; how the final area
is chosen among tried widths; any pruning against a current-best area bound.

## 7. PORTFOLIO, HEURISTIC VARIANTS AND TIE-BREAKING
Any set of distinct configurations run and compared (multiple tie-break orders,
orientation orders, candidate orders, root choices, random seeds), how many, how
each is selected, and how the best result across them is retained. Enumerate the
distinct tie-break / ordering rules and what each changes. Distinguish a fixed
enumerated portfolio from a randomized ensemble of restarts.

## 8. DATA STRUCTURES AND BIT-LEVEL REPRESENTATION
Every significant data structure: bitset/word layout of rows, skyline or column
height arrays, candidate lists, caches of best packings, and how they are
addressed and sized.

## 9. CONTROL FLOW AND COMPUTE BUDGET
The end-to-end order of phases that produces the final packing, expressed as a
sequence of steps with their loop bounds and any early exits. Report any
COMPUTE BUDGET that governs how long the search runs: wall-clock time limits,
```

```
iteration or restart caps, and any ANYTIME behaviour (returning the best packing
found so far when a deadline is hit). A time-bounded anytime solver and a
fixed-work solver are materially different and this section must say which.

## 10. PARAMETERS
An exhaustive table, one row per numeric or categorical parameter, sorted
alphabetically by role name. Columns: role | value | what it controls | what
would change if it changed. Include size thresholds, lookahead/beam widths,
candidate-width bounds, tie-break selector values, RNG seeds and restart counts,
time or iteration limits, and any scoring constants. Do not skip a parameter
because it looks conventional.

## 11. NOT IMPLEMENTED
Techniques a strong solution to this task might use and that this solution
demonstrably does NOT use (e.g. exact ILP/branch-and-bound, simulated annealing,
per-instance width tuning, rotation-aware nesting, no-fit polygons). Be specific
and honest; this section distinguishes solutions that share a strategy from
solutions that share an ancestor.

If the source below is visibly truncated, specify everything determinable from
what is present and state plainly in the affected section that the remainder
could not be determined. Do not guess at missing code.

Output the specification only, beginning with "## 1. OBJECTIVE", with no
preamble and no closing commentary.

=== SUBMISSION ===
<submission source>
```