ON THE VALUE OF ITERATIVE REPAIR IN CODE GENERATION UNDER A TOKEN BUDGET

Anonymous authors
Paper under double-blind review

ABSTRACT

When a language model is asked to write a program and its first attempt fails the tests, the system spends further model calls to recover, and it does so in one of two ways: it either draws additional independent attempts from the same prompt, which is the resampling family behind pass@k evaluation and Best-of-N selection, or it feeds the failing program and its test output back to the model for repair, which is the family behind self-debugging agents. Practice is split between the two and the published evidence points in both directions, since repair has been reported both as a decisive improvement and as a poor deal compared with simply drawing more samples. Which of the two is worth its price, and what decides the sign? In this paper, we first define the solve-within-budget curve of an inference policy under the provider's own token accounting and take its normalized area as the quantity to compare, so that a longer repair prompt is charged for what it costs. Then, we record two elementary facts that organize the measurement: the two families can differ only on episodes whose first attempt fails, so the benchmark-level contrast is a first-attempt failure rate times a conditional advantage, and a repair turn is charged for carrying the previous program, so repair cannot lead below the price of a first repair call. Finally, we measure both policy families on BigCodeBench, its hard subset and HumanEval+ across four model families, and we repeat the comparison with neither family capped in the number of calls it may buy. The experimental results show that repair leads resampling on demanding benchmarks once the budget affords a first repair call, that at the price of one further fresh call the two are indistinguishable, that the lead survives uncapping, that it rides on the execution feedback and persists when the graded tests are withheld from every prompt, and that the extra solutions carry a premium in tokens per solved task.

1 INTRODUCTION

Generating programs with large language models (Chen et al., 2021; Austin et al., 2021; Li et al., 2022; Rozière et al., 2023) has become one of the most widely deployed applications of the technology, and the systems built on it are evaluated by whether generated code passes tests that the system did not write (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023; Lai et al., 2023; Zhuo et al., 2025; Jimenez et al., 2024). Because a single attempt frequently fails, essentially every deployed system spends more than one model call per task, and the question of how to spend the extra calls has become a design decision of its own. One family draws additional independent samples from the same prompt and keeps whichever passes, which is the mechanism behind pass@k reporting, Best-of-N selection, and the repeated sampling studies of inference scaling (Chen et al., 2021; Wang et al., 2023; Cobbe et al., 2021; Brown et al., 2024; Snell et al., 2025; Wu et al., 2025). The other family keeps a single line of work and feeds the failing program together with its execution feedback back to the model for repair, which is the mechanism behind self-debugging, self-refinement, and the interactive coding agents built on them (Chen et al., 2024; Madaan et al., 2023; Shinn et al., 2023; Zhong et al., 2024; Ridnik et al., 2024; Yang et al., 2024; Wang et al., 2025).

Apart from the engineering appeal of both families, the published evidence about which one to prefer does not agree. Repair has been reported to lift solve rates substantially once execution feedback is available (Chen et al., 2024; Shinn et al., 2023), while a cost-controlled study found self-repair often a worse use of the same budget than drawing more independent samples (Olausson et al., 2024), and

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.

a parallel line of work argues that models are poor judges of their own failures and cannot be relied on to correct themselves (Huang et al., 2024; Stechly et al., 2025; Kambhampati et al., 2024; Stroebl et al., 2026). Currently, it remains unclear what decides the sign of this comparison, because the two families are not naturally charged the same way. A repair turn must carry the previous program and its failure text in the prompt, so it is strictly more expensive per call than a fresh sample, and a comparison that counts calls rather than tokens quietly subsidizes it, while a comparison at a small budget quietly penalizes it. Reported comparisons also differ in their benchmarks, and one on which first attempts usually succeed leaves few episodes on which the two families can differ.

Though promising results have been achieved by both families, it still remains an open problem to say where each one is the right way to spend a budget, and the two difficulties above suggest that the comparison should be posed as a regime question rather than as a single verdict. In this paper we charge every policy the provider's own reported token usage, define the solve-within-budget curve and its normalized area as the comparison quantity, and analyze the resulting contrast. Because both families begin from the same first attempt, they can differ only on episodes whose first attempt fails, so the benchmark-level contrast is identically the product of a first-attempt failure rate and a conditional advantage, and a near-ceiling benchmark cannot exhibit an effect however large that advantage is. Because a repair turn costs more than a fresh sample, there is a budget below which the repair policy can afford strictly fewer attempts and resampling weakly dominates, which bounds any crossover from below. We then measure the two families on three public benchmarks across four model families, with the sandboxed benchmark tests as the only source of correctness, locate both regimes, and report what the additional solutions cost. We repeat the comparison with no call cap, so that each budget decides how many calls each family purchases, and we split each task's test methods by a frozen rule, restrict repair feedback to one half, and score every policy only on the other half, which no prompt ever sees.

Our contributions can be summarized as follows: 1) We set up the comparison as a solve-within-budget curve under the provider's token accounting, with a budget-adaptive protocol that lets each family buy as many calls as the budget affords and a held-out test split that scores every policy on tests no prompt has read. 2) We record two elementary facts that organize the measurement: the contrast is supported only on failing first attempts and so factorizes into a failure rate and a conditional advantage, and resampling cannot lose below the price of a first repair call, which bounds the crossover from below and fixes where the budget must be swept. 3) We measure both families on three code benchmarks and four model families, and find that repair leads on the demanding substrates on the grid where the call prices bite, that the lead survives uncapping and withheld tests, and that the extra solutions carry a stated premium. 4) We identify the channel: replacing the failure text removes the advantage on three families, and keeping the program without the failure text reverses it.

2 Related Work

Spending extra inference compute by drawing more samples is the oldest of the two families and is built into the standard pass@k protocol (Chen et al., 2021; Austin et al., 2021). Selecting among the samples requires a verifier, which may be a trained reward model (Cobbe et al., 2021; Lightman et al., 2024; Le et al., 2022), a set of generated tests (Chen et al., 2023; Zhang et al., 2023), execution agreement (Shi et al., 2022; Ni et al., 2023), or a model asked to judge (Zheng et al., 2023). Recent work characterizes how solve rates grow with the number of samples and how to allocate a budget across samples and depth (Brown et al., 2024; Snell et al., 2025; Wu et al., 2025), and also how imperfect verifiers cap those gains (Stroebl et al., 2026; Kapoor et al., 2025). When the benchmark ships its own tests, as ours do, the selection problem disappears and this family reduces to independent restarts, the favourable end of the analysis in which returns to resampling are bounded by the acceptance test (Stroebl et al., 2026; Kapoor et al., 2025).

Repairing a failing program instead of restarting descends from automated program repair (Goues et al., 2012; Xia et al., 2023) and was carried into the language model setting by self-debugging and self-refinement, which return execution feedback or self-criticism to the model and ask for a corrected program (Chen et al., 2024; Madaan et al., 2023; Shinn et al., 2023; Zhong et al., 2024; Ridnik et al., 2024; Xia et al., 2023; Jin et al., 2023). The same loop is the inner mechanism of coding agents that act on a repository (Yao et al., 2023b; Yang et al., 2024; Wang et al., 2025) and of

search procedures that branch over partial solutions (Yao et al., 2023a; Zhou et al., 2024; Zelikman et al., 2023). How much of the reported gain is genuine has been disputed, both for code, where Olausson et al. (2024) matched the budget in samples and tokens on competition problems, let the model write its own explanation of the failure before repairing, and found repair often inferior to more samples, and for reasoning, where models were shown to be unable to correct themselves without external signal (Huang et al., 2024; Stechly et al., 2025; Kambhampati et al., 2024). Our measurement reports the whole curve with the failure rate that a verdict on it depends on, on current open-weight families, and it includes an arm in which the model writes its own explanation of the failure before repairing, so that the feedback protocol of that study is covered as well.

Closest to our comparison is the treatment of repair as an exploration-exploitation problem, in which a bandit rule decides which failing program to invest the next repair turn in (Tang et al., 2024), building on classical bandit machinery (Thompson, 1933; Auer et al., 2002; Russo et al., 2018). That work reports gains from scheduling repairs well, and we adopt its rule as our second comparator with its own concentration parameter and its own sweep. Currently, there is no previous work that locates the budget at which the two families cross under the provider's token accounting, that removes the call cap from both families so that the budget alone decides how many attempts each buys, or that checks the repair advantage against tests withheld from every prompt.

3 Repair and Restart under a Token Budget

In this section, we first define an inference policy, its charged cost, and the solve-within-budget curve we use to compare policies. Then, we identify the set of episodes on which two policies that share a first attempt can differ, and give the resulting factorization of the contrast. Afterwards, we derive the budget threshold below which the cheaper family weakly dominates. Finally, we state what the two results ask the experiments to measure.

3.1 Policies, Cost, and the Solve-within-Budget Curve

We consider a task t drawn from a benchmark that ships a test suite T_t which the system does not author. A policy π interacts with a model by issuing calls $u_1, u_2, \ldots$; each call returns a candidate program and is charged the cost $c(u_i)$ that the provider reports for it, namely the sum of its prompt and completion tokens. A policy stops at the first candidate that passes T_t, and we write $K_\pi(t)$ for the index of that call and $C_\pi(t) = \sum_{i \le K_\pi(t)} c(u_i)$ for the tokens charged up to it, with $C_\pi(t) = \infty$ when the policy exhausts its calls without passing.

Definition 1 (Solve-within-budget curve)**.** *For a budget $\beta > 0$, the solve-within-budget value of π on a task distribution is $S_\pi(\beta) = \mathbb{P}[C_\pi(t) \le \beta]$, and for a grid B of budgets the normalized area, which the tables report as AUC, is $A_\pi = \frac{1}{|B|} \sum_{\beta \in B} S_\pi(\beta)$.*

The curve S_π is the operationally meaningful object, since a practitioner who buys β tokens per task receives exactly $S_\pi(\beta)$, and the area A_π summarizes it once a budget range is fixed. Charging reported tokens rather than calls is what makes the two families comparable, since a repair call carries the previous program and its failure text and is therefore dearer than a fresh call.

Reporting the whole curve rather than one operating point matters because Proposition 2 below shows that the two curves cross, so a report at a single budget can yield either sign by the choice of that budget. We therefore fix the grid before any call is issued, report the area over it, and also plot the curves so that the crossing stays visible. All contrasts are paired at the level of the task, since the two policies share their first call on each task and pairing removes the task-to-task variation in difficulty that otherwise dominates the variance of either policy's own solve rate.

The two families we compare are the simplest representatives of each. The **always-repair** policy issues one fresh call and then, while the candidate fails, spends each remaining call repairing the most recent candidate with its test feedback. The **Best-of-N** policy issues independent fresh calls from the same prompt until one passes, which is the restart family in the case where the benchmark tests decide acceptance and no learned verifier is needed. We additionally compare against **REx** (Tang et al., 2024), which treats the fresh candidates as bandit arms and spends the repair call on the arm a Thompson rule with concentration κ selects, so that the budget buys scheduling rather than an extra attempt.

3.2 WHERE THE TWO FAMILIES CAN DIFFER

Both families spend their first call on the same fresh prompt, which pins the set of episodes on which their outcomes can separate.

Proposition 1 (Support and factorization)**.** *Let π and π' be policies whose first call is identical in distribution and which stop at the first passing candidate, and let F be the event that the first candidate fails T_t. Then $S_\pi(\beta) - S_{\pi'}(\beta) = \mathbb{P}[F]\,(\mathbb{P}[C_\pi \le \beta \mid F] - \mathbb{P}[C_{\pi'} \le \beta \mid F])$ for every β, and the same factorization holds for $A_\pi - A_{\pi'}$.*

The proof can be found in Appendix A. We write $q_{\text{fail}} = \mathbb{P}[F]$ for the first-attempt failure rate. It can be observed from Proposition 1 that the benchmark-level contrast is not a property of the two policies alone: it is their conditional advantage on failing first attempts, attenuated by q_{fail}. Two consequences follow. A benchmark on which the model almost always succeeds immediately drives $\mathbb{P}[F]$ towards zero and reports no difference whatever the conditional advantage is, and a benchmark-level number reported without its failure rate does not carry over to a deployment with a different one. We therefore report the contrast both ways in experiments.

3.3 THE BUDGET THRESHOLD

Because repair calls are charged more than fresh calls, a small budget does not merely shrink the advantage of repair, it removes it. We write S_{repair} and S_{restart} for the curves of always-repair and Best-of-N.

Proposition 2 (Small-budget dominance and threshold)**.** *Suppose that on each task the first call costs c_0, each subsequent fresh call costs at most c_b, and each repair call costs at least $c_r > c_b$, where the three costs may depend on the task. Then for every $\beta < \min_t(c_0 + c_r)$ we have $S_{\text{repair}}(\beta) \le S_{\text{restart}}(\beta)$, and the inequality is strict whenever a second fresh call solves a positive-probability set of episodes within $c_0 + c_b$. If in addition the repair family has the larger asymptotic solve rate, then $S_{\text{repair}} - S_{\text{restart}}$ changes sign at a threshold $\beta^* \ge \min_t(c_0 + c_r)$ that, holding c_b and the success events fixed, is nondecreasing in c_r and hence in the cost ratio c_r/c_b.*

The proof can be found in Appendix A. It can be concluded from Proposition 2 that the sign of the comparison is a statement about the budget and not only about the policies, and that the crossing point cannot lie below the price of carrying a failing program in the prompt, whatever the model's ability to debug. The proposition also tells us how to measure: a comparison at a single budget can report either sign, so the budget must be swept, and swept finely in the first few multiples of a single call's cost, where the bound lies.

4 EXPERIMENTS

In this section, we first describe the benchmarks, the sandbox, the token accounting and the comparator policies. Then, we report the main comparison and the intervals it rests on. Afterwards, we remove the call cap, locate the crossover, and score every policy on tests withheld from every prompt. Finally, we identify the channel that carries the advantage and report what the additional solutions cost.

4.1 EXPERIMENTAL SETUP

We evaluate on three public benchmarks whose tests and reference solutions we did not author. BigCodeBench (Zhuo et al., 2025) and its harder subset BigCodeBench-Hard pose library-heavy tasks with `unittest` suites, and HumanEval+ (Liu et al., 2023) extends the original HumanEval problems (Chen et al., 2021) with additional test inputs and serves as our near-ceiling boundary substrate. Before any model call we ran each benchmark's own reference solution through our sandbox and excluded the tasks whose reference fails its own tests, together with tasks requiring network access or libraries the sandbox forbids; the eligible tasks of each benchmark were then shuffled once with a fixed seed and the first n taken, so the hard-subset draw overlaps the full-benchmark draw only partially, and the exclusion counts are recorded with the raw data. Candidate programs are executed in a namespace-isolated sandbox with a CPU limit and a wall clock timeout, and a candidate counts as passing only when the benchmark's suite runs and reports no failure. Every policy is

charged the provider's reported prompt and completion tokens for each call; every call in the main matrix carried usage fields, fewer than two per cent of fresh calls and fewer than seven per cent of repair calls reach the completion cap, and of the first attempts that fail about two thirds fail an assertion, nine in six hundred and forty-five fail to parse, and the rest raise a runtime exception, never a sandbox or import error. The reported area is taken over the budget grid recorded in the collection configuration before any call was issued, one thousand to twelve thousand tokens in steps of one thousand. We also report the same contrast on a finer grid over the range of realized spend, two hundred and fifty to six thousand two hundred and fifty tokens in steps of two hundred and fifty, on which the crossovers are also read. The confirmatory quantity is the pooled contrast on the demanding substrates with task identities resampled, and per-cell intervals are reported alongside it. We use four open-weight model families served by one endpoint through an OpenAI-compatible interface, under the identifiers DeepSeek-V4-Flash, Qwen3.6-35B-A3B, Gemma-4-31B-IT and Ministral-3-14B, at temperature 0.7 with a nine hundred token completion cap and with thinking disabled, so that the charged cost is comparable across families. All four carry every cell of the main matrix; the mechanism and robustness probes are separate collections on the flagship cell, BigCodeBench with DeepSeek, unless a family is named. BigCodeBench-Hard is the hard subset of BigCodeBench, so a task can appear in both, and every pooled or clustered statistic below counts each task and model once. The four families were not all run on the same draw, so every substrate is restricted throughout to the tasks all four answered, which are the first tasks of the fixed shuffle, while the flagship probes, which pair within themselves, use that model's full draw of three hundred. Each policy is allowed three model calls per task in the reference protocol, so the families are matched in calls as well as priced in tokens. We additionally collect a budget-matched corpus on the flagship substrate with two model families, twelve independent fresh samples and a chain of six repair turns per task, and there every policy is budget-adaptive: at a given budget it buys as many calls in its own order as that budget affords, with no call cap on either family. Both protocols are reported.

Two details of the harness matter for the comparison. Candidate processes that abort or exhaust memory are recorded as failing programs rather than discarded, since discarding them would remove exactly the episodes a repair turn exists to fix. Calls with missing usage fields invalidate the whole episode instead of being imputed, so every contrast below uses only episodes whose every call carries a reported cost.

We compare against several comparator policies. (1) **Best-of-N** issues independent fresh calls from the same prompt and accepts the first that passes, which under benchmark-decided acceptance is the strongest form of the resampling family. (2) **REx** (Tang et al., 2024) spends its third call repairing the bandit-selected one of its two fresh candidates, at $\kappa = 20$ in the main table and swept over four values in the probes. (3) **Best-of-**5 extends the resampling family beyond call parity so that resampling is given more attempts than repair, on its own collection of one hundred and twenty tasks. All contrasts are paired by task, and intervals are percentile bootstrap intervals over five thousand resamples of tasks (Efron, 1979) except where a cluster interval is stated, in which case the k named clusters rather than tasks are resampled; the two policies share the same first call on every task. Nineteen of the hard-subset tasks also fall in the full-benchmark draw, and pooled statistics count each task-model pair once.

4.2 Experimental Results

Main Experiment. The main results are reported in Table 1. We can draw the following conclusions based on the results: (1) On the two demanding benchmarks the always-repair policy has the larger area under the solve-within-budget curve in all eight cells on both grids. The finer grid over the realized spend is the one on which the call prices bite, and there the paired interval excludes zero in five of the eight cells and the pooled contrast in Table 2 excludes zero whether pairs or task identities are resampled, with a leave-one-family-out range that stays positive; on the registered grid six cells and the pooled contrast exclude zero. (2) On the near-ceiling benchmark, where fewer than a fifth of first attempts fail, the contrast is indistinguishable from zero on three families and negative on the fourth; Proposition 1 accounts for the attenuation, and both of its factors depend on the substrate. (3) The last column gives the contrast on the failing episodes, which is the registered-grid column divided by the failure rate and so the same measurement rather than a second one. It does not grow on the hard subset for three of the four families, so the harder substrate is not where a repair turn is worth more but where more episodes reach the choice. The same reading holds across families: the weakest of the four fails its first attempt most often on every substrate while its conditional advantage

Table 1: Normalized solve-within-token-budget AUC, with the area taken over the budget grid registered in the collection configuration, one thousand to twelve thousand tokens. Δ is always-repair minus Best-of-N, paired by task, on that grid and on a finer grid spanning the range of realized spend. The last column is the contrast on the episodes whose first attempt fails, which by the identity of Proposition 1 is the registered-grid column divided by q_{fail}; it is left out where fewer than sixty first attempts fail, which is every near-ceiling cell. The last block is the near-ceiling boundary substrate.

Model	q_{fail}	Repair	Best-of-N	REx	Δ (registered grid)	Δ (fine grid)	Δ (given failure)
BigCodeBench ($n = 200$)							
DeepSeek	0.49	0.647	0.593	0.647	$+0.055$ [$+0.007$, $+0.102$]	$+0.039$ [-0.001, $+0.081$]	$+0.110$ [$+0.013$, $+0.207$]
Qwen3.6	0.48	0.622	0.574	0.629	$+0.048$ [$+0.009$, $+0.089$]	$+0.037$ [$+0.003$, $+0.070$]	$+0.100$ [$+0.019$, $+0.181$]
Gemma-4	0.47	0.656	0.579	0.643	$+0.077$ [$+0.034$, $+0.122$]	$+0.056$ [$+0.020$, $+0.091$]	$+0.166$ [$+0.078$, $+0.257$]
Ministral-14B	0.58	0.586	0.523	0.585	$+0.063$ [$+0.016$, $+0.110$]	$+0.050$ [$+0.009$, $+0.090$]	$+0.108$ [$+0.029$, $+0.188$]
BigCodeBench-Hard ($n = 102$)							
DeepSeek	0.70	0.417	0.375	0.423	$+0.042$ [-0.021, $+0.105$]	$+0.023$ [-0.027, $+0.073$]	$+0.060$ [-0.029, $+0.148$]
Qwen3.6	0.70	0.400	0.363	0.412	$+0.038$ [-0.025, $+0.103$]	$+0.024$ [-0.029, $+0.078$]	$+0.054$ [-0.038, $+0.144$]
Gemma-4	0.71	0.415	0.315	0.376	$+0.100$ [$+0.044$, $+0.158$]	$+0.067$ [$+0.029$, $+0.108$]	$+0.141$ [$+0.067$, $+0.221$]
Ministral-14B	0.79	0.394	0.287	0.395	$+0.107$ [$+0.037$, $+0.179$]	$+0.081$ [$+0.024$, $+0.142$]	$+0.135$ [$+0.046$, $+0.226$]
HumanEval+ ($n = 150$, *near-ceiling boundary*)							
DeepSeek	0.09	0.944	0.932	0.949	$+0.013$ [-0.013, $+0.039$]	$+0.011$ [-0.011, $+0.033$]	—
Qwen3.6	0.14	0.892	0.921	0.921	-0.029 [-0.060, -0.003]	-0.030 [-0.058, -0.008]	—
Gemma-4	0.07	0.943	0.937	0.943	$+0.006$ [-0.002, $+0.018$]	$+0.004$ [-0.002, $+0.014$]	—
Ministral-14B	0.19	0.882	0.865	0.872	$+0.017$ [-0.015, $+0.051$]	$+0.013$ [-0.015, $+0.043$]	—

Table 2: The contrast on the demanding substrates under three choices of resampling unit. The rows differ only in what is treated as exchangeable; task identity is the unit that respects the four families answering the same tasks. Tasks appearing in both substrates for one model are counted once. The last row is not an interval but the range of the point estimates obtained by dropping one model family at a time.

Unit	k	vs. Best-of-N	vs. Best-of-N, fine grid	vs. REx
Task-model	1132	$+0.062$ [$+0.043$, $+0.081$]	$+0.045$ [$+0.029$, $+0.061$]	$+0.002$ [-0.013, $+0.016$]
Task identity	283	$+0.062$ [$+0.039$, $+0.088$]	$+0.045$ [$+0.025$, $+0.065$]	$+0.002$ [-0.013, $+0.016$]
Leave-one-family out	4	$+0.056$ to $+0.069$	$+0.041$ to $+0.050$	-0.004 to $+0.005$

Table 3: The budget-adaptive comparison: up to twelve fresh samples and six repair turns are collected per task, and at each budget every policy buys as many of its own calls as that budget affords, so a large budget buys the resampling family up to all twelve samples; the fourth column gives how many an episode can buy at the top of the grid. Both families are collected on the model's full draw, keeping episodes in which every call is valid. The fifth column is a pre-specified small budget, the price of the first call plus one further fresh call; the crossover is read off a grid in steps of two hundred and fifty tokens.

Model	n	Crossover	Fresh samples affordable (of 12)	Δ at $c_0 + c_b$	Δ whole range (250 to 12000)
DeepSeek	252	1000	12.0	$+0.008$ [-0.016, $+0.032$]	$+0.108$ [$+0.061$, $+0.155$]
Gemma-4	150	2000	11.9	-0.013 [-0.033, $+0.000$]	$+0.105$ [$+0.054$, $+0.158$]

sits in the same range as the other three, so its aggregate contrast is set by how many episodes reach the choice rather than by a larger gain once there. (4) REx tracks always-repair, with a pooled contrast that covers zero in Table 2, and lies above Best-of-N on the demanding substrates, so with three calls its gains are those of repairing at all; its concentration sweep is reported in Table A3.

What the Interval Depends On. The four model families answer the same tasks and the hard substrate overlaps the full one, so Table 2 recomputes the contrast under three choices of what is treated as exchangeable, on the deduplicated demanding set. It can be concluded from the results that (1) the contrast stays positive with an interval excluding zero whether the resampled unit is the task-model pair or the task identity, and it stays positive when any one family is left out; and (2) the comparison against REx covers zero under every unit.

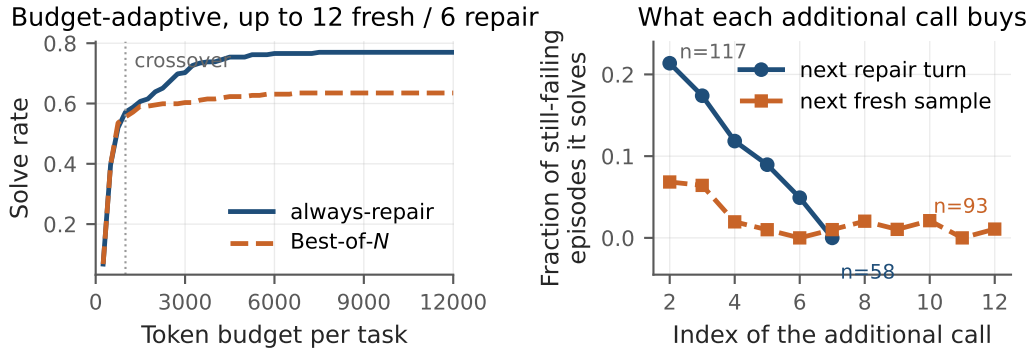


Figure 1: *(Left)* Solve rate against the charged token budget in the budget-adaptive collection on BigCodeBench with DeepSeek, where a large budget buys the resampling family up to all twelve of its fresh samples and the repair family its whole chain of six turns; the dotted line marks the crossover. *(Right)* What each additional call buys, measured as the fraction of episodes still failing before that call which the call solves, for the two kinds of call. The repair chain has six turns, so its curve ends at the seventh call, while fresh sampling continues to the twelfth; the annotations give the number of still-failing episodes at the first and the last point of each curve.

Table 4: Mean charged tokens for the first call, a further fresh call and a repair call, beside the budget at which the repair policy takes the lead. A repair call costs about twice a fresh call on every family. The first four cost columns are means over tasks; the fifth is the smallest first-plus-repair cost any task incurs, which is the region Proposition 2 covers, and every crossover lies above it. Intervals resample tasks.

Model	c_0	c_b	c_r	c_r/c_b	$\min_t(c_0 + c_r)$	Crossover
DeepSeek	494	491	1007	2.05	618	2000 [750, 3000]
Qwen3.6	599	600	1181	1.97	856	1750 [1500, 2750]
Gemma-4	702	706	1429	2.03	1177	2000 [1500, 2250]
Ministral-14B	482	481	1058	2.20	626	1500 [750, 2250]

Letting Resampling Spend the Budget. The reference protocol allows each family three calls, which prices the two kinds of call correctly but does not let a large budget buy more attempts. Since resampling is the cheaper family per call, the cap is the one setting under which it cannot spend a large budget, so we re-collected the flagship substrate on two families with twelve independent fresh samples and a chain of six repair turns per task, keeping the episodes in which every call is valid, and made both families budget-adaptive: at each budget every policy buys as many calls in its own order as the budget affords, and nothing else changes. The comparison is reported in Table 3 and the left panel of Figure 1. We can draw the following conclusions based on the results: (1) above the crossover the resampling family does not overtake at any budget on either model family, even at the top of a grid where an episode affords essentially all twelve fresh samples against a chain that has long since saturated; (2) at a small budget fixed in advance from the mean call prices, the first call plus one fresh call, the two families are indistinguishable; and (3) the whole-range contrast is positive with an interval excluding zero on both families. The right panel of Figure 1 says why the extra fresh samples do not close the gap. Among episodes still failing when the call is bought, the next repair turn solves several times the share that the next fresh sample solves, and the fresh curve is near zero by the fifth sample, on denominators the figure annotates. The twelve fresh samples of an episode are almost all distinct programs drawn from one fixed conditional distribution, so drawing again does not change what is drawn from, while a repair turn conditions on a program and a failure and so draws from a different one each turn: on the flagship family the fresh pool's solve rate rises by about a tenth from one sample to twelve, and of the first attempts that fail, further fresh samples rescue about a fifth while the repair chain rescues about a half.

What Sets the Crossover. Proposition 2 bounds the crossover from below by the two call prices. Table 4 reports those prices and the budget at which the repair policy takes the lead, with the crossover interval obtained by resampling tasks. We can draw the following conclusions based on the results:

Table 5: The contrast when the graded tests are withheld from every prompt. Repair feedback quotes only the visible half of each task's test methods, both families stop on the visible half, and the score counts only the held-out half; the last column repeats the contrast scored on the visible half. The n differ from the main table because tasks whose split leaves fewer than two methods on a side are excluded before collection, and the middle column gives the share of candidates that pass the visible half and still fail the held-out half. The pooled rows count each task-model pair once, since the hard subset overlaps the full benchmark.

Data	Model	n	Repair	Vis. pass, h.o. fail	Δ on held-out tests	Δ on visible
BigCodeBench	DeepSeek	183	0.670	0.087	$+0.073$ [$+0.029$, $+0.117$]	$+0.106$ [$+0.058$, $+0.157$]
BigCodeBench	Qwen3.6	196	0.605	0.087	$+0.033$ [-0.015, $+0.080$]	$+0.045$ [-0.003, $+0.094$]
BigCodeBench	Gemma-4	196	0.646	0.099	$+0.069$ [$+0.034$, $+0.107$]	$+0.091$ [$+0.055$, $+0.131$]
BCB-Hard	DeepSeek	92	0.468	0.259	$+0.103$ [$+0.026$, $+0.182$]	$+0.102$ [$+0.023$, $+0.181$]
BCB-Hard	Qwen3.6	102	0.364	0.230	$+0.039$ [-0.021, $+0.100$]	$+0.029$ [-0.036, $+0.096$]
BCB-Hard	Gemma-4	102	0.423	0.218	$+0.085$ [$+0.038$, $+0.138$]	$+0.092$ [$+0.043$, $+0.145$]
Pooled	all	816	—	—	$+0.066$ [$+0.046$, $+0.087$]	—
Cell cluster	$k = 6$	816	—	—	$+0.069$ [$+0.045$, $+0.090$]	—

(1) a repair call is charged about twice a fresh call on all four families, and on every one of the task-model pairs the repair call is the dearer of the two, so the premise of Proposition 2 holds task by task; the difference comes from carrying the previous program and its failure text rather than from longer outputs; (2) the region Proposition 2 covers ends at the cheapest first-plus-repair cost any task incurs, between six hundred and twelve hundred tokens across the families, and every measured crossover lies above it; and (3) the budget-adaptive collection of Table 3 places the crossover at or below the capped protocol's on both families it covers.

Tests Withheld from Every Prompt. The repair policy reads the failure text produced by the suite that also scores the episode, so its advantage has to be established against tests that no prompt has read. We therefore split each task's test methods by a content-blind rule frozen before collection, restrict the repair feedback to one failing method of the visible half, let both families stop on the visible half, and score every policy only on the held-out half, which never enters any prompt. The results are reported in Table 5. We can draw the following conclusions based on the results: (1) scored on tests that no prompt has seen, the always-repair policy retains its advantage over Best-of-N on all six cells, four of the six paired intervals exclude zero individually, and the pooled interval excludes zero; (2) on the full benchmark the advantage measured this way is about seven tenths of the advantage measured on the visible half, which quantifies the overfitting to the repaired-against tests, while on the hard subset the two scorings agree to within noise; and (3) the share of candidates that pass the visible half and still fail the held-out half is two to three times larger on the hard subset.

What Carries the Advantage. To identify the channel, we re-collected the flagship cell on that model's full three hundred task draw, and the same cell on two further families, with the previous program and the failure text replaced by a content-free notice that the previous program did not pass, and the flagship cell once more with the program kept and only the failure text replaced, leaving the policy, the call count and the accounting untouched, so each pair of collections pairs task by task. The results are in Table 6 with four further collections on the flagship cell. It can be concluded from the results that (1) with the failure text replaced the contrast falls to zero on two of the three families and to a quarter of its size on the third, and the paired difference between the two collections excludes zero on all three; keeping the previous program in the prompt while removing the failure text drives the contrast below zero on the flagship cell, so the advantage of repair is carried by the execution feedback and not by conditioning on the model's own previous attempt, which is consistent with the reasoning literature that finds self-correction without an external signal ineffective (Huang et al., 2024; Stechly et al., 2025); (2) a second independent draw of the flagship cell on the same tasks reproduces the contrast; (3) letting the model write its own explanation of the failure and repair from that explanation alone, with both calls charged to the repair turn, leaves repair ahead by a wider margin, so the sign does not depend on whether the repair prompt carries the raw test output or the model's account of it; (4) lowering the sampling temperature leaves the contrast unchanged; and (5) granting the resampling family five fresh calls against three for repair does not reverse the contrast.

What the Advantage Costs. Since the repair family is charged more per call, we finally report what the two policies pay for a solution rather than for an episode in Table 7. It can be observed that

Table 6: Robustness and mechanism probes on BigCodeBench. Each row is a separate collection whose only change from the reference protocol is the one named; the feedback rows, including the arm that keeps the previous program but removes the failure text, are paired task by task against the reference on that family's own draw, the paired difference being the mean over tasks of the reference contrast minus the arm's contrast, which is why the DeepSeek reference differs from its common-set cell in Table 1. The second draw repeats the reference protocol on the same three hundred tasks; in the explanation arm every repair turn is two charged calls, one that writes an explanation of the failure from the program and the test output and one that repairs from the program and that explanation alone. The temperature and Best-of-5 rows are unpaired DeepSeek collections.

Variation	n	Repair AUC	Δ(vs. Best-of-N)
Reference protocol	299	0.645	$+0.086$ [$+0.044$, $+0.129$]
Content-free failure notice	299	0.556	-0.003 [-0.026, $+0.020$]
paired difference	299	—	$+0.089$ [$+0.041$, $+0.138$]
Reference protocol, Gemma-4	200	0.656	$+0.077$ [$+0.035$, $+0.121$]
Content-free failure notice, Gemma-4	200	0.594	$+0.017$ [-0.002, $+0.040$]
paired difference	200	—	$+0.060$ [$+0.014$, $+0.107$]
Reference protocol, Ministral-14B	200	0.586	$+0.063$ [$+0.016$, $+0.111$]
Content-free failure notice, Ministral-14B	200	0.539	-0.005 [-0.044, $+0.036$]
paired difference	200	—	$+0.068$ [$+0.009$, $+0.130$]
Program kept, failure text removed	300	0.491	-0.045 [-0.070, -0.023]
paired difference	300	—	$+0.134$ [$+0.089$, $+0.182$]
Second independent draw, same tasks	300	0.641	$+0.114$ [$+0.075$, $+0.156$]
Explanation written by the model, then repair (both calls charged)	300	0.683	$+0.146$ [$+0.107$, $+0.186$]
Temperature 0.2	150	0.627	$+0.091$ [$+0.042$, $+0.142$]
Best-of-5 comparator (Δ vs. Best-of-5)	120	0.651	$+0.071$ [$+0.008$, $+0.137$]

Table 7: Charged cost per episode and per solved episode on BigCodeBench with DeepSeek, over the two hundred tasks all four families answered. The cost per solved episode is what a practitioner pays for a solution, and the last row prices the solutions repair adds over resampling.

Policy	Tokens per episode	Tokens per solve	Solve rate
always-repair	1322	1988	0.665
Best-of-N	927	1544	0.600
REx	1132	1715	0.660
always-repair minus Best-of-N	$+395$	6083 per added solve	$+0.065$

the repair policy spends about two fifths more tokens per episode and that its cost per solved task stays above that of resampling even though it solves more tasks; the marginal price of the solutions it adds, the extra tokens per episode divided by the extra solve rate, is three to four times the average price of a solution. Repair therefore buys additional solutions at a premium that a system can price against its need for them. Table A2 reports how much each repair turn contributes, and the marginal return falls sharply after the third turn; Table A1 gives the contrast within complexity strata and by failure type, where it is positive both on assertion failures and on runtime exceptions, and Figure A1 gives the per-family curves.

5 Conclusion

In this paper, we study how a coding system should spend an inference budget beyond one attempt. We first define the solve-within-budget curve under the provider's token accounting, which prices a repair turn for the program it carries, and record that the contrast is a failure rate times a conditional advantage and that resampling cannot lose below the price of a first repair call. We then measure both families and find that the advantage of repair holds on two demanding benchmarks across four open-weight families, persists on withheld tests, survives uncapping, and rides on execution feedback, and that the extra solutions cost more tokens each than the ones resampling finds.

These results suggest a simple allocation rule: below the price of a first repair call the budget should go to fresh samples, and once it covers that price it should go to about three repair turns that carry the execution feedback, a switch that matters most where first attempts fail often and that pays when a solved task is valued above the marginal price of the solutions repair adds.

Reproducibility Statement

The two propositions are proved in full in Appendix A. The benchmarks, the task exclusions, the sandbox in which generated programs are executed, and the provider token accounting that defines the charged budget are specified in Appendix B, and the held-out test split is described in Appendix B.4. Every number in the paper is computed from raw per-episode records produced by the collection scripts, and the scripts, prompts, configuration files, and raw records will be released with the paper so that each table can be rebuilt from the same data.

AI Use Statement

Generative AI tools were used in this work to draft and edit text, to write and refactor the collection and analysis code, and to assist the literature search. All generated code was reviewed and tested by the authors, every reported statistic was recomputed by the authors from the raw records, every cited work was verified against its published source, and the claims and their framing are the authors' own. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

References

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002. doi: 10.1023/a:1013689704352. URL https://doi.org/10.1023/a:1013689704352.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732. arXiv:2108.07732.

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, 2024. URL https://arxiv.org/abs/2407.21787. arXiv:2407.21787.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=ktrw68Cmu9c.

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

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to Self-Debug. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KuPixIqPiq.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168. arXiv:2110.14168.

B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1): 1–26, 1979. doi: 10.1214/aos/1176344552. URL https://doi.org/10.1214/aos/1176344552.

Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012. doi: 10.1109/tse.2011.104. URL https://doi.org/10.1109/tse.2011.104.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world Github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. InferFix: End-to-end program repair with LLMs. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1646–1656, 2023. doi: 10.1145/3611643.3613892. URL https://doi.org/10.1145/3611643.3613892.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: LLMs can't plan, but can help planning in LLM-Modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 22895–22907, 2024. URL https://proceedings.mlr.press/v235/kambhampati24a.html.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=Zy4uFzMviZ.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen tau Yih, Daniel Fried, Sida Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 18319–18345, 2023. URL https://proceedings.mlr.press/v202/lai23b.html.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21314–21328, 2022. doi: 10.52202/068431-1549. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/8636419dea1aa9fbd25fc4248e702da4-Abstract-Conference.html.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://doi.org/10.1126/science.abq1158.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation.

In *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572, 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-Tau Yih, Sida I. Wang, and Xi Victoria Lin. LEVER: Learning to verify language-to-code generation with execution. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 26106–26128, 2023. URL https://proceedings.mlr.press/v202/ni23b.html.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=y0GJXRungR.

Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code generation with AlphaCodium: From prompt engineering to flow engineering, 2024. URL https://arxiv.org/abs/2401.08500. arXiv:2401.08500.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code Llama: Open foundation models for code, 2023. URL https://arxiv.org/abs/2308.12950. arXiv:2308.12950.

Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on Thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018. doi: 10.1561/2200000070. URL https://doi.org/10.1561/2200000070.

Freda Shi, Daniel Fried, Marjan Ghazvininejad, Luke Zettlemoyer, and Sida I. Wang. Natural language to code translation with execution. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3533–3546, 2022. doi: 10.18653/v1/2022.emnlp-main.231. URL https://aclanthology.org/2022.emnlp-main.231/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4O0v4s3IzY.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=j8H84v6AZ1.

Hao Tang, Keya Hu, Jin Peng Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. Code repair with LLMs gives an exploration-exploitation tradeoff. In *Advances in Neural Information Processing Systems*, volume 37, pp. 117954–117996, 2024. doi: 10.52202/079017-3746. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/d5c56ec4f69c9a473089b16000d3f8cd-Abstract-Conference.html.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933. doi: 10.2307/2332286. URL https://doi.org/10.2307/2332286.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VNckp7JEHn.

Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pp. 1482–1494, 2023. doi: 10.1109/icse48619.2023.00129. URL https://doi.org/10.1109/icse48619.2023.00129.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652, 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html.

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*, volume 36, pp. 11809–11822, 2023a. doi: 10.52202/075280-0517. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D. Goodman, and Nick Haber. Parsel: Algorithmic reasoning with language models by composing decompositions. In *Advances in Neural Information Processing Systems*, volume 36, pp. 31466–31523, 2023. doi: 10.52202/075280-1367. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6445dd88ebb9a6a3afa0b126ad87fe41-Abstract-Conference.html.

Kexun Zhang, Danqing Wang, Jingtao Xia, William Yang Wang, and Lei Li. ALGO: Synthesizing algorithmic programs with generated oracle verifiers. In *Advances in Neural Information Processing Systems*, volume 36, pp. 54769–54784, 2023. doi: 10.52202/075280-2389. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/abe1eb21ceb046209c96a0f5e7544ccc-Abstract-Conference.html.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger via verifying runtime execution step by step. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 851–870, 2024. doi: 10.18653/v1/2024.findings-acl.49. URL https://aclanthology.org/2024.findings-acl.49/.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 62138–62160, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

A Proofs

A.1 Proof of Proposition 1

Let F denote the event that the first candidate fails the suite T_t. Both policies issue the same first call, so on F^c each of them stops at that call with $K_\pi = K_{\pi'} = 1$ and charged cost $C_\pi = C_{\pi'} = c(u_1)$. Conditioning on F and its complement,

$$S_\pi(\beta) = \mathbb{P}[F^c]\,\mathbb{P}[C_\pi \le \beta \mid F^c] + \mathbb{P}[F]\,\mathbb{P}[C_\pi \le \beta \mid F], \tag{1}$$

and the same identity holds for π'. Since the first call has the same distribution under both policies, the first terms of the two expansions coincide and cancel in the difference, which gives

$$S_\pi(\beta) - S_{\pi'}(\beta) = \mathbb{P}[F]\,(\mathbb{P}[C_\pi \le \beta \mid F] - \mathbb{P}[C_{\pi'} \le \beta \mid F])\,. \tag{2}$$

Averaging both sides over the budget grid B and using linearity gives the statement for $A_\pi - A_{\pi'}$. When the two policies are run on the same first call for each task, as in our protocol, the cancellation is exact per task rather than only in distribution.

A.2 Proof of Proposition 2

Fix $\beta < \min_t(c_0 + c_r)$. Any episode in which the repair policy stops after two or more calls is charged at least its task's $c_0 + c_r > \beta$, so $\{C_{\text{repair}} \le \beta\}$ is contained in the event that the first candidate passes and $c_0 \le \beta$. The restart policy contains that same event, because its first call is the same call, and it additionally contains the event that the first candidate fails, the second fresh candidate passes, and $c_0 + c_b \le \beta$. Therefore $S_{\text{repair}}(\beta) \le S_{\text{restart}}(\beta)$, with strict inequality whenever the additional event has positive probability, which is the stated condition.

For the second claim, write $D(\beta) = S_{\text{repair}}(\beta) - S_{\text{restart}}(\beta)$. By the first part $D(\beta) \le 0$ on $(0, c_0 + c_r)$, and D is nonnegative for β beyond the largest charged cost of either policy whenever the repair family has the larger asymptotic solve rate, since both curves are then flat at their asymptotes. Because S_{repair} and S_{restart} are nondecreasing right-continuous step functions, D changes sign at $\beta^* = \inf\{\beta : D(\beta) > 0\}$, and $\beta^* \ge \min_t(c_0 + c_r)$ follows from the first part. Finally, increasing c_r while holding c_b and all success events fixed can only enlarge the interval on which the repair policy cannot afford its second call, so the set $\{\beta : D(\beta) > 0\}$ shrinks and its infimum cannot decrease, which is the claimed monotonicity in c_r for fixed c_b and hence in the cost ratio.

B Experimental Details

B.1 Substrates and Exclusions

BigCodeBench and BigCodeBench-Hard tasks are taken from the released parquet snapshots, and HumanEval+ from its released split. Before any model call, each task's own reference solution was executed against its own suite in our sandbox, and tasks whose reference fails were excluded as broken gold; the excluded identifiers are listed in the run artifacts. Tasks declaring libraries the sandbox forbids for isolation reasons, including networking, subprocess, and multiprocessing modules, and tasks whose declared libraries are absent from the environment, were excluded by the same preflight. MBPP was examined and dropped before collection because its hidden-test contract does not pin an entry point, which makes an assert-based suite ambiguous for programs that are otherwise correct.

B.2 Sandbox and Charging

Candidates run inside a namespace-isolated sandbox with a read-only filesystem view, no network, a CPU rlimit, and a wall-clock timeout, and their standard output is parsed for a single result marker. Candidate processes that abort or exhaust memory are recorded as failing programs rather than dropped, because dropping them would remove exactly the episodes that a repair turn is supposed to fix. Cost is the provider's reported prompt plus completion tokens per call, so a repair prompt is charged for the previous program and the failure text it carries; calls whose usage fields are missing invalidate the episode rather than being imputed; in the budget-adaptive collection 252 of the 300 DeepSeek episodes and all 150 Gemma episodes have every call valid.

B.3 Prompts

Every call carries the system message "You are a Python coding assistant. Return only complete, runnable Python code that satisfies the task." The first call and every fresh call send the benchmark's task prompt followed by "Return only the complete self-contained Python code." A repair call sends the task prompt, then "The previous program failed external tests.", the previous program in a code block, "Observed test feedback:" followed by the last seven hundred characters of the suite's output, and "Return only a complete corrected self-contained Python program." The content-free arm of Table 6 replaces the program and the feedback by the sentence "The previous program failed its external tests. Your previous code did not pass."; the program-kept arm keeps the code block and replaces only the feedback line by "Your previous code did not pass.". Both are otherwise identical to the reference.

B.4 The Held-Out Test Split

The split rule was frozen before collection. For each task the test-method names are taken in loader order, the even positions form the visible half and the odd positions the held-out half, and tasks with fewer than two methods on either side are excluded before any model call. Every candidate is executed once and reports per-method outcomes, from which both halves are derived, so the two halves are scored on identical program behaviour. Repair feedback quotes one failing visible method, both policies stop at the first candidate that passes the visible half, and an episode counts as solved only if the accepted candidate passes the held-out half.

C Additional Results

Table 7 in the body reports the charged cost of each policy on the flagship cell, and Table A2 reports the marginal contribution of each repair turn. Two further observations from the same raw data support the body analysis. First, the crossing point lies above the cheapest first-plus-repair cost on every family taken separately, not only in the pooled curve. Second, the ratio between the held-out and the visible contrast in Table 5 is about seven tenths on the three full-benchmark cells and close to or above one on the three hard-subset cells, which bounds how much of the visible-test advantage is attributable to fitting the tests that were repaired against.

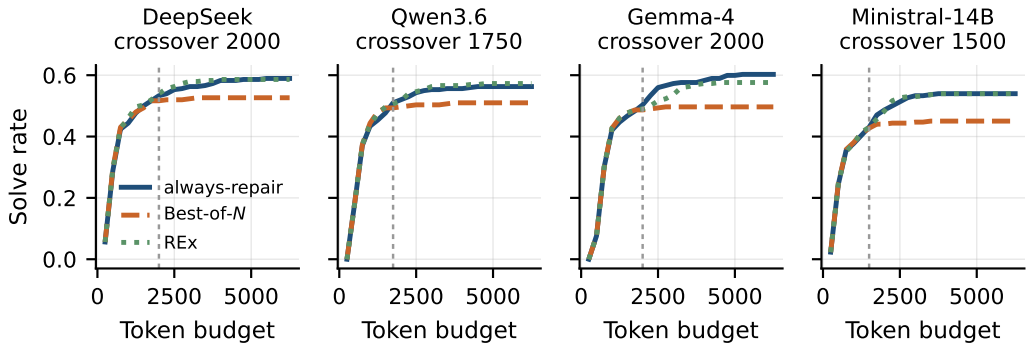


Figure A1: Solve rate against the charged token budget on the demanding benchmarks, separately for each model family, with the measured crossover marked. The measured crossovers lie in the region where a second call becomes affordable on all four families.

Table A1: The contrast within strata of two task properties that are fixed by the benchmark and independent of any policy's behaviour. The advantage is present in every stratum of both. The last block splits the failing first attempts by the kind of failure the suite reported and gives the contrast given failure.

Stratum	n	q_{fail}	Δ(vs. Best-of-N)
Declared library count			
two	372	0.50	+0.061 [+0.016, +0.112]
three	544	0.56	+0.057 [+0.028, +0.087]
four or more	216	0.72	+0.078 [+0.022, +0.138]
Reference solution length in lines			
up to 8	180	0.41	+0.065 [+0.012, +0.129]
9 to 14	412	0.51	+0.051 [+0.015, +0.089]
15 to 22	372	0.65	+0.071 [+0.027, +0.121]
23 or more	168	0.73	+0.066 [+0.007, +0.124]
First-attempt failure type (contrast given failure)			
assertion failure	409	—	+0.120 [+0.055, +0.187]
syntax or indentation error	9	—	—
other runtime exception	227	—	+0.091 [+0.023, +0.161]

Complexity Strata. The two substrates mix tasks of very different scope, so we ask whether the advantage is confined to a slice of them. We stratify by two properties the benchmark fixes and no policy can influence, the number of third-party libraries a task declares and the length of the reference solution, and recompute the contrast inside each stratum in Table A1. It can be concluded that (1) the contrast is positive in every one of the seven strata across both properties, with every one of the seven intervals excluding zero on its own, so the advantage is spread over the substrate rather than concentrated in a band of task complexity; (2) the first-attempt failure rate rises across the strata of both properties, confirming they are difficulty proxies rather than arbitrary partitions; and (3) the contrast varies little across them, so within these substrates its size is not explained by task scope.

Deeper Chains. The reference protocol allows three calls per task, and the marginal return reported above raises the question of whether the comparison changes when both families are deepened. We therefore re-collected the flagship cell allowing up to five calls per task and deepened both families together, so that a repair chain of length five is compared against five independent fresh samples at the same charged cost, with the results in Table A2. It can be concluded from the results that (1) the contrast does not shrink as both families are deepened, and the four estimates lie within one another's intervals; (2) the marginal fraction of still-failing episodes that each additional repair turn fixes decays quickly and is close to zero by the fifth call, so the repair family exhausts its own opportunity within a few turns and the episodes it has not fixed by then are ones the feedback does not help it with at all; and (3) because both families saturate together, the ordering at three calls transports to the depths a deployed system would use, while nothing here supports very long chains.

Table A2: Deepening both families together on BigCodeBench with DeepSeek, from two to five calls per task, charged and scored on the budget grid registered in the collection configuration. A repair chain of length k is compared against k independent fresh samples, each charged its own actual tokens. The last column is the fraction of episodes still failing before the deepest repair turn that this turn fixes, on 143 episodes with every call valid; it is a different collection from Figure 1, so the two marginal curves differ in level.

Calls	Repair AUC	Fresh AUC	Δ	Marginal turn
2	0.589	0.543	$+0.046\,[-0.001, +0.094]$	0.159
3	0.644	0.575	$+0.069\,[+0.019, +0.123]$	0.155
4	0.664	0.587	$+0.077\,[+0.023, +0.131]$	0.082
5	0.668	0.593	$+0.076\,[+0.025, +0.129]$	0.022

Table A3: The REx comparator on BigCodeBench with DeepSeek under four values of the bandit concentration parameter, replayed on the reference collection. Three of the four values coincide because the Thompson draw saturates, so the sweep realizes two distinct policies.

Concentration	n	REx AUC	Δ(repair vs. REx)
$\kappa = 5$	300	0.620	$+0.026\,[-0.002, +0.054]$
$\kappa = 20, 50, 100$	300	0.626	$+0.020\,[-0.007, +0.047]$

Scheduling. Table A3 sweeps the concentration parameter of the bandit rule that REx uses to choose which failing candidate to repair, replayed on the reference collection, and the REx column of Table 1 gives its area on every cell. The scheduled policy is indistinguishable from always-repair at every concentration value and on every cell, while both are ahead of resampling, so with three calls, where the scheduler decides only once two fresh candidates have failed, its contribution is not separable from that of repairing at all.

D LIMITATIONS

The reference protocol fixes three model calls per task and the uncapped collection extends to twelve fresh samples and six repair turns, so the measurement characterizes the shallow regime in which deployed systems mostly operate and does not describe long repair chains. The two demanding substrates share a task format and a test convention, and on the near-ceiling substrate the conditional advantage, estimated from fewer than thirty failing episodes per cell, is not positive for every family. The threshold of Proposition 2 is stated in charged tokens for a fixed prompt design, so a system that compresses the repair prompt moves the threshold rather than removing it. Finally, all models are served by one endpoint with thinking disabled, which keeps the accounting comparable across families but does not speak to reasoning-heavy serving modes.