WHERE BEST-OF-N GAINS ARE LOST: THE LIMITS OF SELF-SOURCED CODE SELECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Sampling multiple candidate programs is a standard way to scale inference-time performance for code generation. Although increasing the sample count makes it more likely that a correct program is present, realizing this gain requires reliably identifying a correct one. We study this problem when selection relies on signals available from the generator itself, including candidate agreement, self-generated tests, and self-judgment. We find a striking mismatch between where resampling creates opportunity and where these signals are effective: across four models on LiveCodeBench, 78–86% of the gap between single-sample and Best-of-N performance is concentrated on *minority-correct* problems, where only 1–5 of 10 candidates are correct. This is precisely the regime in which self-sourced selectors are weakest, and none significantly outperforms a simple filter that executes the worked examples provided in the problem statement. We isolate two mechanisms behind this shortfall. First, even an oracle behavioral partition leaves plurality selection weak: identifying behavioral classes does not identify the correct class when correct behavior is a minority. Second, generated tests are often discriminating but mislabeled; validating their expected outputs raises a simple test filter 5.7–11.9 points above the example filter on minority-correct problems. Together, these results show that producing additional distinctions among candidates is not enough: the central challenge is grounding those distinctions in correctness. Effective inference-time scaling through resampling therefore requires selection evidence whose connection to correctness can be independently grounded.

1 INTRODUCTION

Inference-time scaling, which allocates additional computation at inference, is a commonly used strategy to obtain better performance for large language models (LLMs) without retraining. On mathematical reasoning tasks, compute-optimal test-time scaling can outperform substantially larger models at matched inference compute (Snell et al., 2025). In code generation this is often done by sampling several candidate programs and using a selection mechanism to identify the most promising one, commonly known as Best-of-N. Generating more samples makes it more likely that the pool contains a correct program, but the accuracy actually realized is set by the selector. We denote the accuracy of a single sample by pass@1 and the probability that at least one of N samples is correct by pass@N, the accuracy attainable with perfect selection. We call the difference pass@N−pass@1 the *resampling gap*. Whether Best-of-N is worth its compute is therefore a question about how much of this gap a selector can recover.

Many selectors have been proposed, and in practice they are combined freely. We examine signals that fall into three families. *Model judgment*: a model reads the candidates and names the correct one, either as a pointwise verifier (Tong & Zhang, 2024) or as a comparative judge (Zheng et al., 2023). Trained verifiers and reward models fall in the model-judgment family as well (Cobbe et al., 2021; Inala et al., 2022; Ni et al., 2023; Zhang et al., 2025). *Execution against tests*: candidates are run on inputs and compared with expected outputs; the tests may be human-written, or generated by a model (Chen et al., 2023; Zhang et al., 2023; Ma et al., 2025). *Agreement*: exploits the intuition that correct programs tend to converge on the same behavior, while incorrect programs are dispersed across many distinct failure modes (Li et al., 2022; Chen et al., 2023). Candidates are grouped by behavior on shared inputs and the largest group is preferred (Li et al., 2022; Shi et al., 2022; Cho et al., 2026). The boundaries between methods are blurry. CodeT (Chen et al., 2023) scores groups

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.

of candidates by the generated assertions they jointly pass, and SRank (To et al., 2024) clusters by output on generated inputs and ranks clusters by functional overlap. S* (Li et al., 2025) revises candidates using public tests, clusters them by behavior on synthesized inputs, and uses an external model to adjudicate between clusters on distinguishing cases.

We restrict our analysis to signals computable solely from the generator: generated assertions, execution agreement across candidate samples, and the generator's internal self-judgment. We term these signals *self-sourced*. This self-contained regime reflects practical test-time selection settings in which multiple candidate solutions can be generated, but ground-truth tests, human annotations, and stronger external judges are unavailable. As a non-self-sourced baseline, we use the cheapest task-provided signal available for most programming problems: the few worked examples included directly in the problem description. Importantly, these examples are already visible to the generator when it produces its candidates; they are not unseen test information. Nevertheless, re-executing candidates against their provided outputs supplies deterministic correctness constraints. Prior work has observed that filtering candidates using provided tests can be surprisingly effective (Li et al., 2024); we ask why this small amount of task-provided evidence can outperform substantially richer self-sourced signals.

Our experiments yield three main findings:

1. **Self-sourced selectors rarely outperform a simple example filter.** Across 16 model–selector pairs on LiveCodeBench, none beats the public-example filter: 14 score significantly below it and 2 are statistically indistinguishable. On HumanEval+, 5 are below, 10 indistinguishable, and only 1 above.

2. **Agreement is weakest where selection is needed most.** Minority-correct problems account for 78–86% of the LiveCodeBench resampling gap, yet plain agreement achieves only 31–41% accuracy there. Even an oracle vote based on full grading-suite behavior reaches only 31–51%, showing that identifying behavioral classes does not reveal which class is correct.

3. **Generated tests contain useful inputs but unreliable expected outputs.** On minority-correct problems, only 34–48% of executable generated assertions are valid. After validating these assertions against known-correct programs, a simple hard filter exceeds the example filter by 5.7–11.9 points, indicating that unreliable labels rather than insufficiently discriminating inputs are the main bottleneck.

Taken together, these findings point to a common bottleneck. Self-sourced methods can distinguish or rank candidate programs through behavioral agreement, generated tests, or direct model assessment, but they do not reliably establish which observed behavior is correct. The example filter succeeds not because it provides richer evidence, but because its input–output constraints are externally grounded in correctness.

2 Related Work

Prior work has identified several limits to inference-time gains from resampling. Brown et al. (2024) show that pass@N can continue to yield performance improvements with additional samples even as practical selectors such as majority voting and reward models plateau. Stroebl et al. (2026) show that imperfect verification can fundamentally limit the gains from resampling, based on analyses of human-written unit tests on HumanEval and MBPP. However, they leave open whether these limitations persist when the tests are generated by LLMs. Zhou et al. (2026) further show that LLM-as-judge verifiers become less reliable on harder problems in mathematics and question answering, leaving code as an open setting. Through an extensive experimental study, we investigate the complementary setting in which the selection evidence is produced by the same model that generates the candidates. Rather than treating the verifier quality as a single error rate, we identify where the gains made possible by resampling are concentrated and isolate the failure modes of different self-sourced signals in code generation.

3 EXPERIMENTAL SETUP

Models and benchmarks. We evaluate four approximately 4B-parameter models: Qwen3-4B-Instruct-2507, Phi-4-mini-instruct, Qwen3.5-4B, and Gemma-4-E4B-it. For each problem, each model generates $N = 10$ candidate programs and also supplies its own generated tests and self-judgments. We evaluate on LiveCodeBench (LCB; Jain et al., 2025) and HumanEval+ (Liu et al., 2023). After excluding problems with grader or statement defects, our evaluation contains 1,010 LCB problems and 161 HumanEval+ problems. Full sampling settings are presented in Appendix J, and the exclusion criteria in Appendix A. A candidate is correct if it passes the benchmark's full official test suite. Because LCB's grader executes public tests before hidden tests, we additionally regrade candidates on hidden tests alone; this leaves the example-filter survivor sets and reported filter accuracies unchanged (Appendix N).

Difficulty. For each model and problem, let c be the number of correct candidates among the ten. We group problems into $c = 0$, 1–5, 6–9, and 10, calling 1–5 the *minority-correct* regime. Because difficulty is defined from each model's own samples, a problem may belong to different bins for different models. Appendix B repeats the difficulty-dependent analyses using a leave-one-model-out definition based on the other three models' pass rates.

Selection methods. Our baseline is the *example filter*: we execute candidates on the worked examples provided in the problem statement and discard any candidate that fails one. We compare this baseline with four self-sourced selectors. *Plain agreement* selects from the largest behavioral cluster on model-generated inputs; *CodeT* (Chen et al., 2023) groups candidates using the generated assertions they pass; *SRank* (To et al., 2024) clusters and ranks candidates by functional agreement on generated inputs; and the *self-judge* asks the candidate-generating model itself to choose among its programs. CodeT, SRank, and plain agreement use the same generated-test pools.

Oracle partition and statistics. To separate failures of behavioral comparison from failures of the candidate distribution itself, we construct a grading-suite oracle partition by executing candidates on the full grading suite. Candidates with matching recorded traces are grouped together, except that pure-TLE candidates are pooled into one class. We then call choosing a candidate from the largest partition *oracle vote*. This partition is used only as a diagnostic, not as a practical selector; its construction and known grader artifacts are detailed in Appendix C. Reported intervals are 95% bootstrap confidence intervals over problems, with paired resampling for differences between selectors. Intervals for quantities reported only in the text are collected in Appendix M, and the full statistical procedures are given in Appendix L.

4 RESULTS

4.1 SELF-SOURCED SELECTORS RARELY OUTPERFORM A SIMPLE EXAMPLE FILTER

The simplest selector we evaluate is also the strongest. The example filter uses only the few worked examples supplied with each problem—2.6 on average on LiveCodeBench, whereas self-sourced methods use generated tests, behavioral agreement, or an additional model judgment. We run every self-sourced selector on all ten candidates, without first applying the example filter, and ask whether it can recover more of the resampling gap.

Figure 1 presents overall results, and Table 1 reports paired differences from the example filter. Across the 16 model–selector pairs on LiveCodeBench, none exceeds the filter: 14 are significantly worse and the remaining two are not significantly different. The result is not that self-sourced evidence is useless. The selectors improve substantially over a random sample in several settings. Rather, the additional self-sourced evidence consistently recovers less of the resampling gap than simply re-executing the examples already present in the problem statement. The example filter recovers 42–79% of the resampling gap across models, compared with 35–66% for the best self-sourced selector.

This gap is not explained solely by imperfections in the implemented rerankers. Even the oracle vote fails to surpass the example filter: its corpus-level accuracy is significantly lower for three models

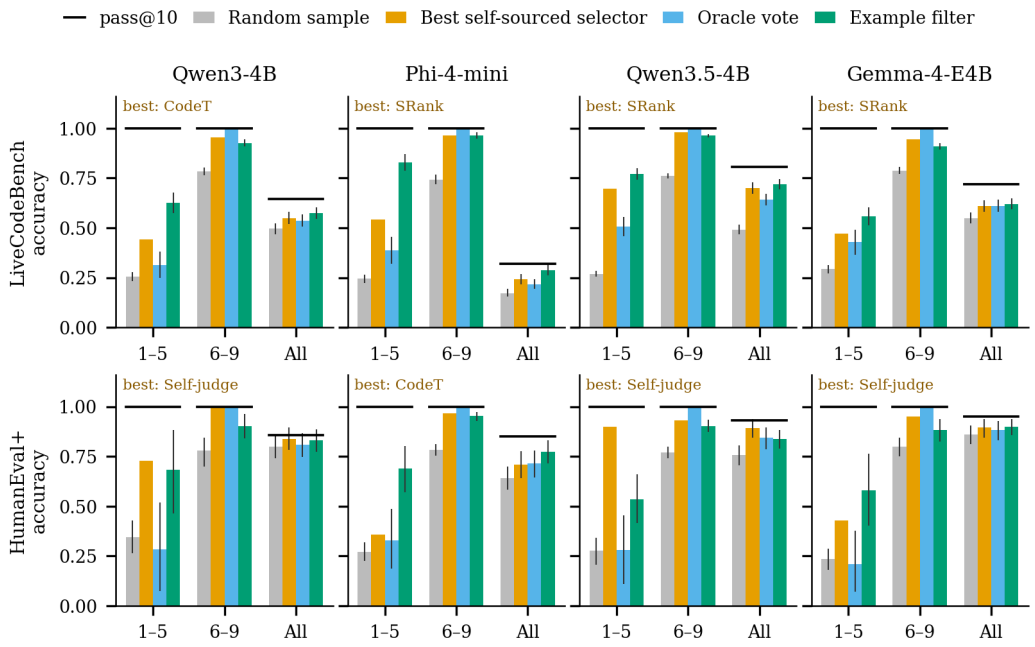

Figure 1: Accuracy per model and difficulty bin on LiveCodeBench and HumanEval+: a random sample, the best self-sourced selector on all ten candidates, the oracle vote, the example filter, and pass@10. Means over problems with 95% bootstrap intervals.

and not significantly different from it for Gemma. Section 4.2 shows why improving the comparator alone cannot solve the problem.

HumanEval+ provides a smaller and less decisive replication. With only 161 tasks overall, its paired comparisons have substantially lower statistical resolution: 5 of 16 self-sourced selectors are significantly below the filter, 10 are not significantly different from it, and one—Qwen3.5 self-judgment—is significantly above it. We therefore base the main comparative claim on LiveCodeBench and treat HumanEval+ as supporting evidence rather than an equally powered replication (Appendix F).

Example filtering does not eliminate the value of self-sourced evidence. The preceding comparison asks whether self-sourced evidence can replace the task-provided examples. We also apply the same selectors after first filtering candidates with the examples. On LiveCodeBench, this yields statistically significant corpus-level gains for nearly all model–selector pairs, but the improvements are modest: 0.5–4.6 accuracy points. The limitation is clearest on minority-correct problems, where most remaining selection headroom lies. Agreement, CodeT, and SRank improve over the filter by 0.8–9.6 points, but even the best of these methods recovers only 10–42% of the remaining gap to pass@10 across models. Thus, self-sourced evidence can refine a candidate pool after task-provided constraints are enforced, but substantial selection headroom remains unrecovered (Appendix E). To understand this limitation, we next step back to the full candidate pool and ask a more basic question: *where is the opportunity created by resampling concentrated, and why do self-sourced selectors struggle to exploit it?*

4.2 Agreement Is Weakest Where Selection Is Needed Most

The resampling gap is concentrated in minority-correct problems. The distribution of candidate-pool difficulty varies substantially across generators (Figure 2). Minority-correct problems ($c = 1$–5) constitute only 16–34% of LiveCodeBench problems across models, while problems with either no correct candidate ($c = 0$) or all correct candidates ($c = 10$) together constitute 37–75%. Despite their modest frequency, minority-correct problems account for 78–86% of the Best-of-N resampling gap (pass@10 minus pass@1). Thus, most of the opportunity created by

Table 1: Paired gain over the example filter on LiveCodeBench problems with at least one correct candidate ($c \geq 1$; 651 / 323 / 815 / 727 problems), in accuracy points. Bold: the 95% bootstrap interval excludes 0. Corpus-level intervals are in Appendix D.

	Paired gain over the filter on problems with $c \geq 1$ (accuracy points)				
Model	Agreement	CodeT	SRank	Self-judge	Oracle vote
Qwen3-4B	**−6.6** [−8.5, −4.7]	**−3.9** [−5.8, −2.2]	**−4.7** [−6.7, −2.7]	**−5.8** [−8.4, −3.4]	**−6.0** [−8.1, −4.0]
Phi-4-mini	**−26.6** [−31.4, −21.9]	**−17.9** [−22.4, −13.6]	**−14.9** [−19.4, −10.7]	**−27.5** [−32.5, −22.5]	**−22.1** [−26.8, −17.6]
Qwen3.5-4B	**−15.0** [−17.5, −12.7]	**−4.2** [−6.2, −2.2]	**−2.4** [−4.6, −0.4]	**−4.9** [−7.4, −2.5]	**−9.7** [−12.2, −7.4]
Gemma-4-E4B	**−3.4** [−5.0, −1.8]	**−1.7** [−3.3, −0.2]	−1.5 [−3.2, +0.1]	−2.2 [−4.6, +0.1]	−1.3 [−3.0, +0.3]

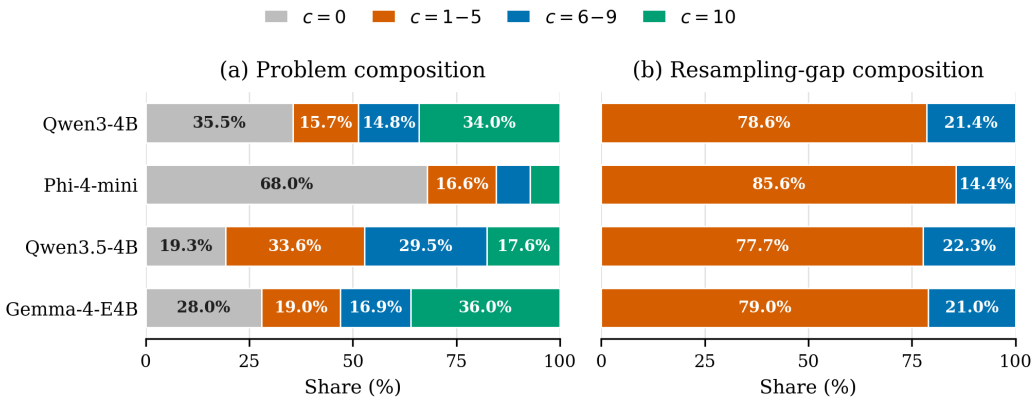

Figure 2: **The resampling gap is concentrated in minority-correct problems.** (a) Composition of LiveCodeBench problems by the number of correct candidates c among ten. (b) Composition of the aggregate resampling gap (pass@10−pass@1) by the same bins. The $c = 0$ and $c = 10$ bins contribute zero gap by definition.

resampling lies on problems where a correct program is present but correct behavior has relatively little support.

Agreement-based selection can still succeed in this regime. It relies on an appealing asymmetry: correct programs should converge on the same behavior, while incorrect programs should disperse across different mistakes. Correctness therefore need not be a majority. Let c be the number of correct candidates and w_1 the size of the largest incorrect behavioral class. Even when correct candidates are a minority, their class wins plurality whenever $c > w_1$. Agreement can therefore recover a correct minority if incorrect behaviors are sufficiently diverse.

Agreement recovers little of the available gap when correct solutions are rare. Raw selection accuracy necessarily increases with c: a random candidate is correct with probability $c/10$. To remove this baseline effect, for each value of c we measure the fraction of the available resampling gap recovered:

$$\text{GapRecovery}(c) = \frac{\text{Acc}_{\text{selector}}(c) - c/10}{1 - c/10}.$$

For $1 \leq c \leq 9$, pass@10 $= 1$, so 0 corresponds to random selection and 1 to recovering the entire available gap; negative values indicate performance below a random selection.

Over the minority-correct regime, plain agreement recovers only 9–19% of the available resampling gap across LiveCodeBench models. This poor recovery is not implied by minority correctness itself. At $c = 1$, plurality cannot strictly favor the correct class. For $c = 2$–5, however, the correct candidates could still form the largest behavioral class if the incorrect candidates were sufficiently dispersed. Instead, agreement extracts little additional information beyond random selection when correct solutions are rare. This could reflect either recurring incorrect behaviors or an inability of the generated inputs to distinguish them; we separate these explanations next.

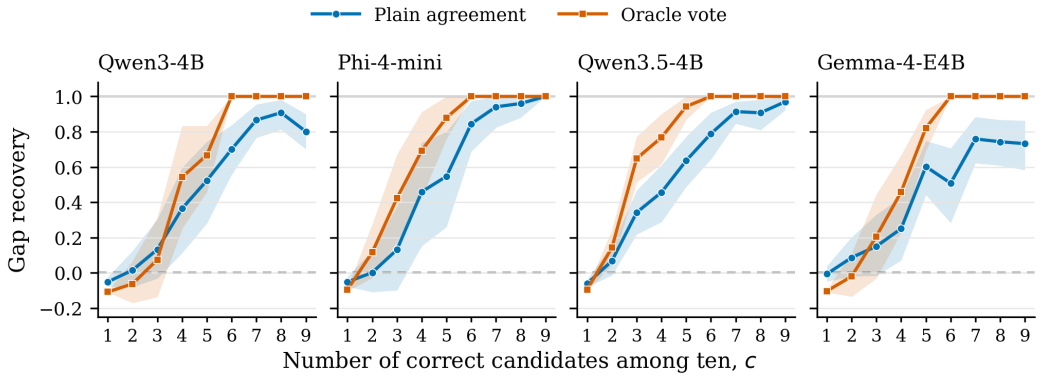

Figure 3: Gap recovery $(\mathrm{acc} - c/10)/(1 - c/10)$ by the exact number of correct candidates c on LiveCodeBench, per generator, selecting among all ten candidates (0 = random pick, 1 = perfect). Shaded regions show 95% bootstrap confidence intervals over problems.

Stronger behavioral comparison does not solve the low-support regime. One possible explanation is that plain agreement fails because its model-generated inputs do not adequately distinguish candidate behaviors. We isolate this factor by replacing the generated-input partition with the grading-suite oracle partition of Section 3, while retaining the same plurality rule.

Across minority-correct LiveCodeBench problems, oracle vote recovers only 8–32% of the available gap across models, compared with 9–19% for plain agreement. Thus, substantially stronger behavioral comparison helps in some settings, most notably for Qwen3.5, but does not generally resolve the failure. When the correct behavior itself has little support in the candidate pool, accurately distinguishing behavioral classes does not ensure that the largest class is the correct one.

HumanEval+ provides a lower-powered replication of the same pattern. Because its minority-correct regime contains only 11–33 problems per model, we use the coarser $c = 1$–5 versus $c = 6$–9 split rather than exact-c estimates. On minority-correct problems, plain agreement recovers between -4% and 2% of the available gap across models, while oracle vote recovers between -10% and 8%. In contrast, on $c = 6$–9 problems, plain agreement recovers 85–100% of the gap, and oracle vote recovers the full gap (Appendix F).

Figure 3 shows this dependence at finer resolution on LiveCodeBench. Macro-averaged across models, plain-agreement gap recovery rises from approximately -4% at $c = 1$ to 58% at $c = 5$, while oracle vote rises from -10% to 83%. Once $c \geq 6$, the correct class is a strict majority and oracle vote recovers the full gap by construction.

Agreement becomes informative only as the correct behavior receives enough repeated support to distinguish itself from competing incorrect behaviors. Improving the comparator strengthens that signal once several correct candidates are present, but cannot create consensus around a correct behavior that appears only rarely. Agreement therefore recovers the least of the available Best-of-N opportunity precisely in the minority-correct regime where most of that opportunity is concentrated.

4.3 Generated tests contain useful inputs but unreliable expected outputs

Generated-test quality depends on two distinct properties. We call an assertion *valid* when its input is in-domain and its expected output is correct; *discrimination* instead asks whether the input exposes incorrect candidate behavior. A test set is *sound* if every correct program passes it, so a sound hard filter cannot reduce uniform-selection accuracy by removing correct candidates. We ask whether generated-test selection is limited primarily by assertion validity or by insufficient discrimination.

The raw tests are unsound. On minority-correct problems only 34–48% of the model's executable tests expect the correct output. A single invalid assertion can reject correct candidates; with dozens of generated assertions per problem, the raw pools reject 97–100% of correct candidates in the 1–5 bin. CodeT and SRank avoid this by using the tests as soft scores, but they still lose to the

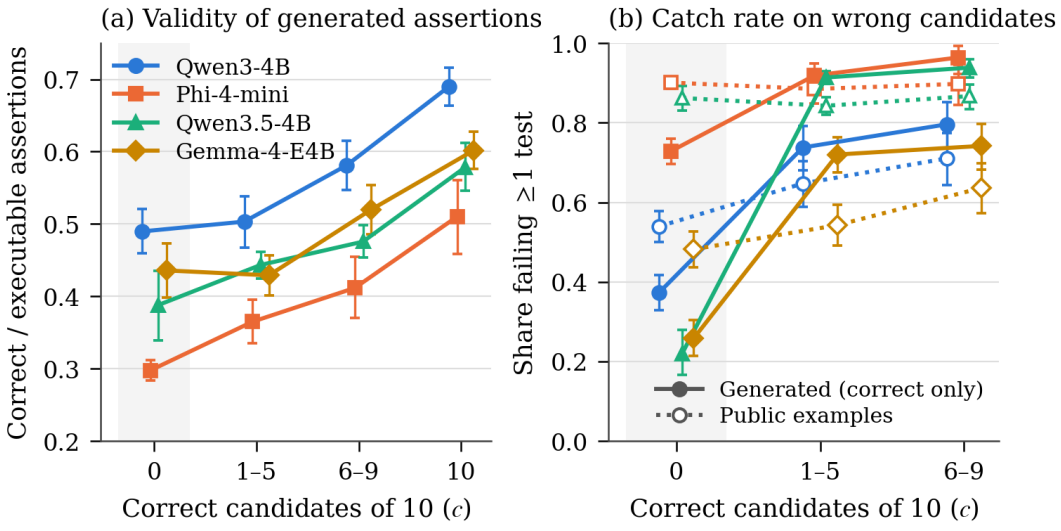

Figure 4: **Generated-test quality degrades on harder problems.** LiveCodeBench, grouped by the number of correct candidates c among ten. (a) Fraction of executable generated assertions with the correct expected output. (b) Fraction of wrong candidates rejected by at least one valid generated assertion (solid) or public example (dotted). Points are means over problems; error bars show 95% bootstrap intervals.

Table 2: Paired gain over the example filter on minority-correct LiveCodeBench problems ($c = 1$–5), in accuracy points. *Validated*: assertions kept only if every known correct program passes them. *Hard filter*: remove every candidate that fails a validated assertion, pick uniformly among the rest; hard-filter columns use a 4 s limit per input. Bold: the 95% bootstrap interval excludes 0.

	Raw assertions		Validated assertions		
Model	CodeT	SRank	CodeT	SRank	Hard filter
Qwen3-4B	**−18.5** [−25.1, −11.8]	**−21.5** [−28.8, −14.1]	**−13.2** [−19.7, −6.9]	**−18.7** [−25.6, −11.8]	**+7.7** [+2.6, +12.9]
Phi-4-mini	**−34.6** [−42.2, −27.1]	**−28.7** [−36.5, −21.1]	**−21.0** [−28.1, −14.2]	**−21.2** [−28.3, −14.4]	**+5.7** [+1.3, +10.3]
Qwen3.5-4B	**−10.4** [−15.0, −5.9]	**−7.4** [−12.2, −2.6]	−1.4 [−5.2, +2.4]	**−5.3** [−9.7, −1.2]	**+11.9** [+9.0, +14.9]
Gemma-4-E4B	**−8.9** [−14.2, −3.7]	**−8.8** [−14.7, −3.0]	−4.1 [−8.8, +0.4]	**−5.3** [−10.2, −0.6]	**+9.2** [+5.1, +13.3]

example filter (Section 4.1). We therefore validate the assertion pool using known-correct programs and test what remains.

Validating the tests. We keep an assertion only if every grader-correct program for the problem returns its expected value: up to 40 programs from all four generators. The rule removes any assertions contradicted by any known-correct program. Every retained assertion uses the input and expected output the model wrote. Because the rule uses known-correct programs, this is an oracle-informed diagnostic rather than a method. Every selector runs on all ten candidates without the example filter (Table 2).

Validated generated tests beat the examples. The validated assertions reject no correct candidate, so they are sound with respect to those pools. They are also more discriminating than the examples, catching 67–93% of wrong candidates in the 1–5 bin against 54–88% for the examples, more for every model. As a hard filter, they exceed the example filter for all four models by 5.7–11.9 points in the 1–5 bin and 1.1–4.6 points over the whole corpus. We use a 4 s per-input limit for this oracle-informed counterfactual to isolate the information content of correctly labeled generated tests. Repeating the hard filter under CodeT's 0.1 s execution harness yields positive gains for all four models, although the gains are smaller and Phi is no longer significantly above the example filter because the tighter limit rejects some grader-correct candidates (Appendix G). Applying both sound filters adds 10.8–15.3 points over the example filter alone. This shows that the generated pool contains sufficiently discriminating inputs, while assertion validity is the dominant bottleneck.

The rerankers do not turn validity into accuracy. Given the same validated assertions, CodeT gains 4.8–13.6 points over its raw version, yet it stays below the example filter for Qwen3 and Phi and is not significantly different from it for Qwen3.5 and Gemma; SRank stays below it for all four. Both still weigh assertion outcomes by agreement among candidates. We do not isolate this term.

Why the examples work. The worked examples in the statement are assertions of the same kind: an input with an expected output. Unlike the model-generated assertions, their expected outputs are task-provided. The audit of every example rejection found no correct program rejected (Appendix N), so the example filter is sound on our evaluated pools. It therefore cannot lower uniform-selection accuracy and can be composed with other sound filters. The contrast also grows with problem difficulty. Generated-assert validity falls significantly as the fraction of correct candidates decreases for all four models on LiveCodeBench. Public examples do not show the same degradation: their catch rate is comparatively stable across difficulty, whereas the catch rates of generated tests and agreement depend more strongly on the fraction of correct candidates (Appendix B). The examples are nevertheless far from complete: two or three examples per problem catch 54–88% of wrong candidates in the 1–5 bin. Where they catch fewer, other signals have room. On HumanEval+ they catch 49% of Qwen3.5's wrong candidates on minority-correct tasks, and there Qwen3.5's self-judge is the one selector that exceeds the filter (Section 4.1).

4.4 Robustness Analyses

Larger candidate pools. Our main experiments use $N = 10$. Stroebl et al. (2026) show that for code generation the optimal N is often fewer than 10. To test whether our conclusions depend on a small candidate pool, we generate 40 additional candidates per problem for Qwen3-4B and Gemma-4-E4B and repeat the main analyses for the example filter, agreement, CodeT, SRank, and oracle vote with up to $N = 50$. All pool-dependent quantities, including pass@N, example-filter accuracy, selector accuracy, and the resampling gap, are recomputed at each N. We also evaluate the 40 new samples separately using difficulty bins defined by the original ten, providing a held-out comparison as the pool grows. Full N-curves, recomputed-difficulty analyses, and sampling checks are in Appendix K.

The conclusions persist despite substantial changes in the selection problem. Previously zero-correct problems acquire correct candidates, shifting where the opportunity from resampling lies. The share of the resampling gap left unrecovered by the example filter that is carried by the original minority-correct problems falls from about 85% at $N = 10$ to 51% for Qwen3 and 58% for Gemma at $N = 50$. Nevertheless, agreement, CodeT, SRank, and even the grading-suite oracle vote still do not significantly exceed the recomputed example filter when selecting directly from all 50 candidates (Appendix K).

Applying the selectors after the examples yields larger absolute gains as the pool grows, but the remaining resampling opportunity grows at least as quickly: more than four fifths of the opportunity left after the example filter remains unrecovered on held-out minority-correct problems across the evaluated pool sizes. The validated hard-filter result also persists, exceeding the example filter by 2.4–2.5 corpus points on all 50 candidates and by 7.3–7.7 points on the held-out 40-candidate pools for originally minority-correct problems. Thus, increasing the sample count creates additional selection opportunities without removing the bottlenecks identified at $N = 10$ (Appendix K).

Cross-model selection evidence points to shared task difficulty. A stronger explanation for our results is that self-sourced selection fails specifically because the same model generates both the candidate programs and the evidence used to select among them. We test this by applying each model's generated-test pool to candidate pools from all four generators on LiveCodeBench, yielding 12 cross-model pairs in addition to the four self-sourced pairs. We run both CodeT and SRank in every pair. On minority-correct problems, switching to cross-model test pools does not qualitatively improve selection: across the 24 cross-model model–selector pairs, gains over the example-filter baseline range from -0.2 to $+8.3$ points, comparable to the self-sourced setting.

The same pattern appears in the test evidence itself. Test validity remains difficulty-dependent across models: its logit slope with generator pass rate is 0.35 $[0.25, 0.45]$, although the dependence is significantly steeper for self-sourced pairs by 0.48 $[0.43, 0.53]$. When each test generator's validity is modeled jointly using the difficulty of its own generator and the mean difficulty of the other

generators, however, the own-generator coefficient is indistinguishable from zero for three of four models (Qwen3, Qwen3.5, and Gemma); Phi is the only exception. Thus, the data do not support the stronger claim that selection fails primarily because a model is evaluating its own candidates. Instead, much of the dependence reflects shared task difficulty: problems that are difficult to solve across models also yield less reliable generated selection evidence. Changing the source model alone therefore does not remove the selection bottleneck (Appendix H).

5 Conclusion

Best-of-N code generation separates two problems that are often conflated: producing diverse candidate behaviors and determining which behavior is correct. Our results show that self-sourced selection is limited primarily by the latter. Agreement can accurately identify behavioral structure yet fail when correct behavior has little support, while generated tests contain useful discriminating inputs but unreliable expected outputs. Increasing the candidate pool creates additional correct solutions without removing this selection bottleneck.

The contrast with task-provided examples points to the missing ingredient: grounded correctness information. A small number of trustworthy examples can outperform much richer self-sourced evidence, and generated assertions become substantially more useful once their labels are validated. Further gains from inference-time resampling therefore require not simply more selection evidence, but mechanisms for establishing which evidence can be trusted without already knowing the correct solution. Developing such validation or certification mechanisms is a central direction for improving Best-of-N selection.

6 Limitations

Our primary evaluation uses four approximately 4B-parameter models. Larger models may generate more reliable tests or perform better self-judgment. The main experiments use $N = 10$; the $N = 50$ extension covers only Qwen3-4B and Gemma-4-E4B and does not include the self-judge or cross-model experiments. The cross-model analysis is likewise limited to LiveCodeBench and to the same four similarly sized model families. LiveCodeBench provides the statistical support for the main claims. HumanEval+ contains only 11–33 minority-correct problems per model, leaving most comparisons underpowered.

Correctness is defined by the benchmark graders and their finite test suites, including their runtime limits, and therefore does not establish semantic correctness for all possible inputs. We audit and exclude problems with identified grader or statement defects. The validated-assertion experiment is also an oracle-informed diagnostic: an assertion is retained only when the available known-correct programs agree with its expected output. This establishes soundness with respect to the known-correct programs evaluated by our validation harness, but does not guarantee semantic validity for every possible correct implementation.

AI Use Statement

Generative AI tools were used as part of the experimental methodology in this work. In particular, the language models evaluated in the paper were used to generate candidate programs, generated tests, and self-judgments that form the basis of the Best-of-N selection experiments.

Generative AI tools were also used to assist with the research workflow, including writing and debugging experimental code, identifying and summarizing related work, and drafting and revising portions of the manuscript. Literature identified with AI assistance was checked against the original sources, and AI-assisted code and manuscript content were reviewed by the authors. The authors are responsible for the experimental methodology, reported results, claims, citations, and final content of the paper.

Reproducibility Statement

We provide code to reproduce the quantitative analyses and result tables in this paper from the corresponding per-candidate experimental artifacts. All experiments use publicly available datasets and open-source LLMs. The release includes the analysis for the main headroom and difficulty analyses, signal-quality measurements, selector comparisons, cross-model generated-test experiments, the validated-test counterfactual, and the larger-candidate-pool robustness experiments. Each analysis uses fixed random seeds, and the repository specifies the Python environment and provides a single script for reproducing the reported tables.

References

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.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. CodeT: Code generation with generated tests. In *International Conference on Learning Representations*, 2023.

David Cho, Yifan Wang, Fanping Sui, and Ananth Grama. Inference-time code selection via symbolic equivalence partitioning. *arXiv preprint arXiv:2604.06485*, 2026. URL https://arxiv.org/abs/2604.06485.

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. *arXiv preprint arXiv:2110.14168*, 2021.

Jeevana Priya Inala, Chenglong Wang, Mei Yang, Andres Codas, Mark Encarnación, Shuvendu K. Lahiri, Madanlal Musuvathi, and Jianfeng Gao. Fault-aware neural code rankers. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-0976.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E. Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 15964–15978, 2025. doi: 10.18653/v1/2025.findings-emnlp.865.

Haau-Sing Li, Patrick Fernandes, Iryna Gurevych, and André F. T. Martins. DOCE: Finding the sweet spot for execution-based code generation. *arXiv preprint arXiv:2408.13745*, 2024.

Yujia Li et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

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.

Zeyao Ma, Xiaokang Zhang, Jing Zhang, Jifan Yu, Sijia Luo, and Jie Tang. Dynamic scaling of unit tests for code reward modeling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6917–6935, 2025. doi: 10.18653/v1/2025.acl-long.343.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-Tau Yih, Sida 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 of *Proceedings of Machine Learning Research*, pp. 26106–26128, 2023.

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.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. In *International Conference on Learning Representations*, 2026.

Hung Quoc To, Minh Huynh Nguyen, and Nghi D. Q. Bui. Functional overlap reranking for neural code generation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3686–3704, 2024. doi: 10.18653/v1/2024.findings-acl.220.

Weixi Tong and Tianyi Zhang. CodeJudge: Evaluating code generation with large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 20032–20051, 2024. doi: 10.18653/v1/2024.emnlp-main.1118.

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, 2023. doi: 10.52202/075280-2389.

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

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, Datasets and Benchmarks Track*, volume 36, pp. 46595–46623, 2023. doi: 10.52202/075280-2020.

Yefan Zhou, Austin Xu, Yilun Zhou, Janvijay Singh, Jiang Gui, and Shafiq Joty. Variation in verification: Understanding verification dynamics in large language models. In *International Conference on Learning Representations*, 2026.

A Exclusion Criteria

The LCB corpus is the frozen 1,055-problem set minus 45 problems with grader defects: float outputs compared by exact string match, special-judge problems with several accepted outputs, and a grader mock of `sys.stdin.__iter__` that mis-scores some stdin programs. Five further problems where the limit is our tooling rather than the grader are kept. The exclusion is enforced at every join, and no excluded problem appears in any table. On HumanEval+, three tasks are excluded. HumanEval/47 has an incorrect docstring example (`median([-10, 4, 6, 1000, 10, 20]) -> 15.0`; the answer is 8.0); HumanEval/116 has contradictory docstring examples and specified behavior; and HumanEval/95 has a test-suite serialization/spec-coverage defect.

B Difficulty Robustness

For generator m, leave-one-model-out (LOMO) difficulty is the mean pass rate of the other three generators on the same problem. We use bins 0, low ($0 < p \leq 0.5$), high ($0.5 < p < 1$), and 1. This removes the direct dependence between a model's own ten correctness labels and its difficulty

Table 3: LiveCodeBench robustness under LOMO difficulty. "Low headroom" is the share of the example filter's remaining pass@10 gap in the LOMO-low bin. The other columns are logistic slopes in LOMO difficulty; brackets give 95% bootstrap intervals.

Model	Low headroom	Assertion validity	Assertion catch	Agreement catch	Example catch
Qwen3-4B	84.5%	1.03 [0.76, 1.28]	5.29 [4.23, 6.59]	0.52 [−0.02, 1.14]	0.35 [−0.30, 1.00]
Phi-4-mini	7.9%	0.67 [0.50, 0.85]	4.44 [3.69, 5.45]	−0.06 [−0.28, 0.15]	−0.04 [−0.42, 0.36]
Qwen3.5-4B	55.2%	0.62 [0.46, 0.79]	8.77 [7.18, 10.89]	2.93 [2.26, 3.73]	0.51 [0.05, 0.99]
Gemma-4-E4B	81.5%	0.98 [0.76, 1.20]	4.61 [3.54, 5.84]	1.21 [0.57, 1.96]	−0.02 [−0.69, 0.59]

Table 4: Corpus-level accuracy on LiveCodeBench (1,010 problems), with every selector run on all ten candidates and no filter. The oracle vote, a plurality over the grading-suite behavior classes, is a reference rather than a selector.

Model	Random	Agreement	CodeT	SRank	Self-judge	Oracle vote	Filter	pass@10
Qwen3-4B	.495	.532	.550	.545	.537	.536	.575	.645
Phi-4-mini	.173	.203	.231	.241	.201	.218	.289	.320
Qwen3.5-4B	.492	.599	.686	.700	.680	.641	.719	.807
Gemma-4-E4B	.550	.596	.608	.610	.605	.611	.621	.720

assignment. The own-model and LOMO difficulties remain strongly related: Spearman correlations range from 0.71 to 0.81.

All four assertion-validity and generated-assert catch slopes remain positive with intervals excluding zero. Public-example catch is comparatively stable: three of four intervals include zero. Agreement catch remains difficulty dependent for Qwen3.5 and Gemma, while the Qwen3 and Phi intervals include zero. Across all 32 LiveCodeBench signal fits, 26 retain their original conclusion, two change sign, two newly include zero, one includes zero under both definitions, and one newly excludes zero. Thus the central validity result is not an artifact of defining difficulty from the evaluated model's own samples, although some weaker agreement and example-catch slopes are not robust to this stricter definition.

C Partition Construction and Known False Positives

Each candidate is assigned a trace against the official suite: LCB public tests first, then private tests; HumanEval+ base inputs first, then plus inputs. When a full regrade is available, the trace vector holds one token per test: the normalized output (whitespace-stripped line comparison for stdin problems, value equality for function-style problems), or the failure type (`raise`, `timeout`). Non-TLE candidates with complete traces share a class iff their recorded vectors are identical. Pure-TLE candidates are pooled into the class `tle` regardless of which tests time out, making the partition an upper bound on agreement among slow programs; all compile errors on a problem share a separate class. A candidate whose recorded evaluation stopped at its first failing test (the grader short-circuits) is a singleton keyed on that test unless it was forced through the full suite; for Qwen3.5 and Gemma, 3,495 and 4,209 of the 10,100 candidates were executed on the full suite and the rest carry the first-failure label. Two graded-correct LCB candidates pass only because of the grader's patched `stdin.read` and are known false positives; they keep their label everywhere and are barred as oracles. The identity filter + comparator share + shared-bug cost + lone-correct cost = pass@10 holds with maximum error zero on 4,040 LCB and 644 HumanEval+ (problem, model) rows.

D Full-Pool Selector Tables by Bin

Corpus-level intervals for Table 4 (all 1,010 problems; Qwen3 / Phi / Qwen3.5 / Gemma): agreement -0.043 $[-0.056, -0.031]$ / -0.085 $[-0.102, -0.068]$ / -0.121 $[-0.141, -0.101]$ / -0.024 $[-0.036, -0.013]$; CodeT -0.025 $[-0.037, -0.014]$ / -0.057 $[-0.073, -0.042]$ / -0.034 $[-0.050, -0.018]$ / -0.012 $[-0.023, -0.002]$; SRank -0.030 $[-0.043, -0.018]$ / -0.048 $[-0.063, -0.034]$ / -0.020 $[-0.036, -0.003]$ / -0.011 $[-0.023, +0.001]$; self-judge -0.038 $[-0.054, -0.022]$ / -0.088 $[-0.106, -0.070]$ / -0.040 $[-0.059, -0.020]$ / -0.016

Table 5: LiveCodeBench, every selector on all ten candidates with no filter, paired gain over the filter by bin. The oracle vote over all ten is a reference, not a selector.

Model	Bin	Filter	Agreement	CodeT	SRank	Self-judge	Oracle vote
Qwen3-4B	1–5	0.626	−0.300 [−0.363, −0.236]	−0.185 [−0.251, −0.118]	−0.215 [−0.288, −0.141]	−0.184 [−0.270, −0.098]	−0.314 [−0.380, −0.244]
Qwen3-4B	6–9	0.927	+0.031 [+0.016, +0.048]	+0.026 [+0.009, +0.044]	+0.026 [+0.008, +0.044]	−0.059 [−0.115, −0.006]	+0.073 [+0.056, +0.091]
Phi-4-mini	1–5	0.829	−0.517 [−0.589, −0.445]	−0.346 [−0.422, −0.271]	−0.287 [−0.365, −0.211]	−0.455 [−0.534, −0.376]	−0.442 [−0.515, −0.370]
Phi-4-mini	6–9	0.965	+0.011 [−0.013, +0.034]	+0.004 [−0.037, +0.036]	−0.001 [−0.043, +0.032]	−0.150 [−0.231, −0.074]	+0.035 [+0.019, +0.052]
Qwen3.5-4B	1–5	0.771	−0.365 [−0.413, −0.318]	−0.104 [−0.150, −0.059]	−0.074 [−0.122, −0.026]	−0.106 [−0.161, −0.053]	−0.265 [−0.317, −0.214]
Qwen3.5-4B	6–9	0.964	+0.006 [−0.010, +0.020]	+0.005 [−0.014, +0.022]	+0.018 [+0.003, +0.031]	−0.014 [−0.040, +0.009]	+0.036 [+0.028, +0.044]
Gemma-4-E4B	1–5	0.558	−0.145 [−0.200, −0.091]	−0.089 [−0.142, −0.037]	−0.088 [−0.147, −0.030]	−0.065 [−0.142, +0.012]	−0.130 [−0.185, −0.075]
Gemma-4-E4B	6–9	0.910	+0.020 [−0.002, +0.042]	+0.028 [+0.008, +0.046]	+0.035 [+0.014, +0.054]	−0.021 [−0.074, +0.027]	+0.090 [+0.073, +0.107]

$[-0.033, +0.001]$; oracle vote -0.039 $[-0.053, -0.025]$ / -0.071 $[-0.087, -0.055]$ / -0.078 $[-0.098, -0.059]$ / -0.010 $[-0.022, +0.002]$.

E Selection After Public-Example Filtering

The main comparison in Section 4.1 evaluates whether self-sourced selectors can replace the public-example filter by applying each selector directly to all ten candidates. Here we evaluate the complementary composition: we first remove candidates that fail a public example and then apply the same selector to the surviving pool. The baseline is therefore a uniform random choice among the example-filter survivors. Gains are paired differences from this baseline, with 95% bootstrap confidence intervals over problems.

Corpus-level results. Table 6 reports results over all LiveCodeBench problems. Every selector except Phi self-judgment improves significantly over the example filter, but the absolute gains are modest, ranging from 0.5 to 4.6 accuracy points. The largest gains occur for Qwen3.5, where SRank and self-judgment add 4.0 and 4.6 points, respectively. Thus, filtering does make the remaining self-sourced evidence more useful, but the improvement is much smaller than the gain already obtained from enforcing the examples.

Table 6: Paired gain over the public-example filter after applying each selector only to the surviving candidates on LiveCodeBench, in accuracy points. Brackets show 95% bootstrap confidence intervals. Bold indicates that the interval excludes 0.

Model	Agreement	CodeT	SRank	Self-judge
Qwen3-4B	$\mathbf{+0.7}$ $[+0.3, +1.2]$	$\mathbf{+1.2}$ $[+0.6, +1.7]$	$\mathbf{+1.1}$ $[+0.5, +1.7]$	$\mathbf{+2.0}$ $[+0.9, +3.0]$
Phi-4-mini	$\mathbf{+0.6}$ $[+0.3, +1.0]$	$\mathbf{+1.3}$ $[+0.8, +1.9]$	$\mathbf{+1.1}$ $[+0.6, +1.7]$	$+0.5$ $[-0.2, +1.3]$
Qwen3.5-4B	$\mathbf{+2.4}$ $[+1.8, +3.0]$	$\mathbf{+3.3}$ $[+2.5, +4.1]$	$\mathbf{+4.0}$ $[+3.2, +4.9]$	$\mathbf{+4.6}$ $[+3.4, +5.8]$
Gemma-4-E4B	$\mathbf{+1.7}$ $[+1.1, +2.2]$	$\mathbf{+1.6}$ $[+1.0, +2.3]$	$\mathbf{+1.5}$ $[+0.8, +2.2]$	$\mathbf{+3.7}$ $[+2.5, +4.9]$

Minority-correct problems. The remaining selection problem is substantially harder in the minority-correct regime ($c = 1$–5). Table 7 reports gains within this bin. Agreement, CodeT, and SRank add between 0.8 and 9.6 accuracy points, but these absolute gains should be interpreted relative to the much larger remaining gap between the example filter and pass@10. Taking the strongest of these three rerankers for each model recovers only 10% of this gap for Qwen3-4B, 38% for Phi-4-mini, 42% for Qwen3.5-4B, and 11% for Gemma-4-E4B. Thus, 58–90% of the available post-filter headroom remains unrecovered by the generated-test and agreement-based rerankers.

Table 7: Paired gain over the public-example filter on minority-correct LiveCodeBench problems ($c = 1$–5), after applying each selector only to example-filter survivors, in accuracy points. Brackets show 95% bootstrap confidence intervals. Bold indicates that the interval excludes 0.

Model	Agreement	CodeT	SRank	Self-judge
Qwen3-4B	$+0.8$ $[-1.5, +3.1]$	$\mathbf{+3.7}$ $[+0.7, +6.9]$	$\mathbf{+3.7}$ $[+0.4, +7.3]$	$\mathbf{+10.2}$ $[+4.5, +16.0]$
Phi-4-mini	$\mathbf{+2.4}$ $[+0.5, +4.4]$	$\mathbf{+6.5}$ $[+3.4, +9.7]$	$\mathbf{+5.3}$ $[+2.2, +8.4]$	$+2.5$ $[-1.6, +6.7]$
Qwen3.5-4B	$\mathbf{+4.7}$ $[+3.0, +6.4]$	$\mathbf{+7.6}$ $[+5.4, +9.8]$	$\mathbf{+9.6}$ $[+7.3, +12.0]$	$\mathbf{+11.7}$ $[+8.3, +14.9]$
Gemma-4-E4B	$\mathbf{+4.9}$ $[+2.3, +7.6]$	$\mathbf{+4.9}$ $[+1.9, +8.0]$	$\mathbf{+3.7}$ $[+0.6, +6.9]$	$\mathbf{+15.0}$ $[+9.5, +20.7]$

Self-judgment yields larger gains for Qwen3-4B, Qwen3.5-4B, and Gemma-4-E4B, recovering approximately 27%, 51%, and 34% of the minority-correct headroom, respectively; for Phi-4-mini its gain is not statistically distinguishable from zero. These results reinforce the same qualitative conclusion: self-sourced evidence becomes more useful after the examples remove candidates that violate known constraints, but substantial room remains between practical post-filter selection and pass@10.

F HumanEval+ Tables

The September 24, 2026 relabel changed the Qwen3.5 minority-correct count from 21 to 20 and updated the corpus-level baseline and selector comparisons. The qualitative result is unchanged: agreement and even the oracle vote are weak in the minority-correct regime, then become effective once correct candidates are a majority.

Table 8: HumanEval+ after the September 24, 2026 relabel (161 tasks). Every selector runs directly on all ten candidates, with no example pre-filter. The first three metric columns are accuracies; selector columns are paired gains over the example filter. Bold intervals exclude zero.

Model	Random	Filter	pass@10	Agreement	CodeT	SRank	Self-judge	Oracle vote
Qwen3-4B	.800	.830	.857	−0.019 [−0.045, +0.001]	**−0.026** [−0.054, −0.004]	−0.013 [−0.043, +0.013]	+0.009 [−0.012, +0.029]	**−0.022** [−0.048, −0.002]
Phi-4-mini	.642	.774	.851	**−0.076** [−0.124, −0.033]	**−0.065** [−0.112, −0.021]	**−0.076** [−0.125, −0.030]	**−0.078** [−0.128, −0.032]	**−0.061** [−0.107, −0.019]
Qwen3.5-4B	.756	.838	.932	−0.004 [−0.034, +0.023]	+0.017 [−0.012, +0.045]	+0.014 [−0.015, +0.041]	**+0.056** [+0.019, +0.092]	+0.005 [−0.027, +0.034]
Gemma-4-E4B	.859	.899	.950	−0.018 [−0.050, +0.011]	−0.019 [−0.050, +0.010]	−0.022 [−0.056, +0.007]	−0.005 [−0.032, +0.020]	−0.018 [−0.051, +0.011]

Table 9: HumanEval+ difficulty regimes. Gap recovery is $(\text{accuracy} - c/10)/(1 - c/10)$, averaged within the indicated coarse bin. Agreement and the oracle vote both run on all ten candidates.

Model	Regime	n	Agreement recovery	Oracle-vote recovery
Qwen3-4B	1–5	11	−4%	−10%
	6–9	9	100%	100%
Phi-4-mini	1–5	33	−1%	8%
	6–9	44	96%	100%
Qwen3.5-4B	1–5	20	2%	1%
	6–9	60	89%	100%
Gemma-4-E4B	1–5	14	2%	−3%
	6–9	20	85%	100%

G Validated-Assertion Experiment

This experiment is an oracle-informed diagnostic of the information already present in the generated assertion pools. For a problem, let C be the set of all grader-correct programs found among the four generators' candidates (up to 40 programs, deduplicated, with the two known grader false positives barred). For an assertion (x, y), the exact validation rule is

$$\text{keep}(x, y) \quad \Longleftrightarrow \quad \forall f \in C,\ f(x) \text{ terminates within 4 s and returns } y.$$

Assertions are only removed. Inputs and expected outputs are never repaired, rewritten, or relabeled; every retained assertion is exactly the assertion the model generated. A candidate that fails any retained assertion is discarded, and selection is uniform among the remaining candidates.

At the corpus level, the validated hard filter gains 1.1–4.6 points over the public examples. Under CodeT's 0.1 s harness the gains shrink to 0.7–1.4 points; Phi's interval includes zero because the tighter limit rejects seven grader-correct candidates. Under the 4 s validation harness, the only labeled correct candidates removed are the two barred grader false positives. Combining the public examples with the validated assertions gains 2.0–5.9 corpus points. On minority-correct problems, the corresponding combined gains are 10.8–15.3 points, and the 4 s hard-filter gains are 5.7–11.9 points as reported in Table 2.

Table 10: Corpus-level LiveCodeBench gains over the public-example filter. "Hard, 4 s" uses the validation harness above; "Hard, 0.1 s" reuses CodeT's tighter execution matrix; "Examples + hard" composes the public examples with the 4 s validated-assertion filter. Brackets give paired 95% bootstrap intervals.

Model	Example filter	Hard, 4 s	Hard, 0.1 s	Examples + hard
Qwen3-4B	.575	$+0.015$ [$+0.007$, $+0.024$]	$+0.014$ [$+0.005$, $+0.023$]	$+0.028$ [$+0.021$, $+0.035$]
Phi-4-mini	.289	$+0.011$ [$+0.003$, $+0.019$]	$+0.007$ [-0.001, $+0.015$]	$+0.020$ [$+0.014$, $+0.027$]
Qwen3.5-4B	.719	$+0.046$ [$+0.035$, $+0.057$]	$+0.014$ [$+0.002$, $+0.026$]	$+0.059$ [$+0.050$, $+0.069$]
Gemma-4-E4B	.621	$+0.021$ [$+0.013$, $+0.029$]	$+0.014$ [$+0.006$, $+0.022$]	$+0.034$ [$+0.027$, $+0.041$]

H Cross-Verifier Cells and Difficulty Regressions

Validity is a property of the verifier's pool and is identical along a row: Qwen3 0.588 [0.571, 0.604], Gemma 0.521 [0.505, 0.537], Qwen3.5 0.480 [0.467, 0.493], Phi 0.339 [0.326, 0.352]; coverage (executable assertions / all) is 0.73–0.80. What changes across a row is which problems count as hard and what the assertions do to that generator's candidates. Table 11 gives CodeT's and SRank's gain over the filter in the 1–5 bin for every cell.

Table 11: LiveCodeBench 1–5 bin. Gain over the filter (paired) of CodeT / SRank when the column generator's survivors are scored with the row verifier's assertion pool. Diagonal cells are the paper's self-sourced runs. Intervals including zero are marked †.

Verifier ↓ / Generator →	Qwen3-4B (ceiling 0.374)	Phi-4-mini (0.171)	Qwen3.5-4B (0.229)	Gemma-4-E4B (0.442)
Qwen3-4B	0.037 / 0.037	0.051 / 0.031	0.067 / 0.071	0.052 / 0.069
Phi-4-mini	$0.003^\dagger$ / $-0.002^\dagger$	0.065 / 0.053	0.071 / 0.082	0.063 / 0.072
Qwen3.5-4B	$0.023^\dagger$ / 0.039	0.056 / 0.043	0.076 / 0.096	0.067 / 0.073
Gemma-4-E4B	$0.025^\dagger$ / $0.023^\dagger$	0.049 / 0.039	0.076 / 0.083	0.049 / 0.037

No generator gains much from another model's pool. Comparing each generator's best off-diagonal result with its best diagonal result, the largest improvement is $+0.024$, for Gemma (5% of its 0.44 ceiling); for the other three, the best off-diagonal result is at most $+0.002$ above the best diagonal result. The five intervals that include zero are all in the Qwen3 candidate column: both selectors with Phi and Gemma assertions, and CodeT with Qwen3.5 assertions.

Table 12: Pooled logistic regression of assertion validity on the candidate generator's pass rate p_i, whether verifier and generator are the same model, and their interaction.

Term	Coefficient [95% CI]
Intercept	-0.447 [$-0.516, -0.376$]
p_i (off-diagonal slope)	0.350 [$0.247, 0.453$]
Same-model indicator	-0.205 [$-0.231, -0.180$]
$p_i \times$ same-model	0.482 [$0.431, 0.535$]

The interaction fit shows that validity falls more steeply with difficulty in self-sourced cells, but the within-verifier fit attributes most of the shared variation to cross-model task difficulty: after controlling for the other generators, the own-generator coefficient includes zero for Qwen3, Qwen3.5, and Gemma, with Phi the sole exception. Within five equal-count validity strata, shared-bug wrong candidates are accepted 1.39 times as often as one-off errors on the diagonal and 1.54 times as often off the diagonal; there is no self-specific excess.

I Self-Judge Details

Protocol (E3, arm 1). The judge is the generator itself, using the chat wrapper, system message, and decoding that produced the candidates, with thinking disabled. It sees the statement (plus starter code), the candidates labeled A–J in a seeded random order (seed 20260915; restricted rather than redrawn for the survivor pool), and the question "Which one is correct?" Its response must end

Table 13: Within-verifier logistic regressions of assertion validity on the verifier's own-generator difficulty and the mean difficulty of the other three generators.

Verifier	Own-generator coefficient	Other-generators coefficient	Spearman ρ
Qwen3-4B	0.209 [−0.078, 0.488]	0.779 [0.334, 1.209]	0.812
Phi-4-mini	0.498 [0.298, 0.715]	0.389 [0.190, 0.586]	0.710
Qwen3.5-4B	0.092 [−0.130, 0.322]	0.553 [0.320, 0.783]	0.751
Gemma-4-E4B	0.177 [−0.089, 0.430]	0.812 [0.494, 1.145]	0.769

with `ANSWER: <letter>`. The initial output budget is 4,096 tokens; while a cell's parse rate is below 95%, calls truncated at that limit are reissued at 8,192, 16,384, and 32,768 tokens with the same per-request seed. A trailing markdown fence is skipped by the parser (this recovered 406 answers, raising the parse rate from 92.1% to 97.3%). An unparsed call is scored as a uniform random choice among the candidates shown. The population is problems with at least one correct candidate. A pool with one candidate is scored without a call; an empty pool scores zero. Checks per cell are a parse rate of at least 95%, position-swap agreement on a 100-problem sample, and a survivor set identical to the reranker runs.

Table 14: LiveCodeBench, self-judge on the filter's survivors, by bin: accuracy (an unparsed call is scored as a uniform random choice among the survivors), gain over the filter, and the parse-failure rate among judge calls.

Model	Bin	n	Accuracy	Gain over filter	Parse failure
Qwen3-4B	1–5	159	0.728	+0.102 [+0.045, +0.160]	0.007
Qwen3-4B	6–9	149	0.952	+0.025 [−0.011, +0.056]	0.020
Phi-4-mini	1–5	168	0.854	+0.025 [−0.016, +0.067]	0.095
Phi-4-mini	6–9	83	0.976	+0.011 [−0.022, +0.037]	0.000
Qwen3.5-4B	1–5	339	0.887	+0.117 [+0.083, +0.149]	0.020
Qwen3.5-4B	6–9	298	0.988	+0.024 [+0.010, +0.036]	0.071
Gemma-4-E4B	1–5	192	0.708	+0.150 [+0.095, +0.207]	0.000
Gemma-4-E4B	6–9	171	0.959	+0.049 [+0.017, +0.077]	0.000

On all ten candidates, on the same population ($c \geq 1$), the self-judge scores 0.834 versus the filter's 0.892 (Qwen3), 0.627 versus 0.902 (Phi), 0.842 versus 0.892 (Qwen3.5), and 0.840 versus 0.862 (Gemma); the paired gains are in Table 1.

J Sampling Protocol and Assertion Prompt

Table 15: Decoding settings. Thinking is disabled for the two models that support it.

Model	LCB ($n = 10$)	HumanEval+	Max tokens
Qwen3-4B-Instruct-2507	T = 0.7, top-p 0.8	T = 0.7, top-p 0.95, first 10 of 20	default
Phi-4-mini-instruct	T = 0.7, top-p 0.8 (Qwen3-Instruct settings)	T = 0.7, top-p 0.95, first 10 of 20	default
Qwen3.5-4B	T = 1.0, top-p 0.95	T = 1.0, top-p 0.95	20,480
Gemma-4-E4B-it	T = 1.0, top-p 0.95	T = 1.0, top-p 0.95	20,480

LCB candidates were generated through `lcb_runner` (Jain et al., 2025) (Qwen3-4B and Phi each with its own CodeQwenInstruct or Phi4Mini prompt string; Qwen3.5 and Gemma through their chat templates with a "You are a helpful assistant." system turn). HumanEval+ candidates were generated through `evalplus` (Liu et al., 2023) with vLLM (Kwon et al., 2023), single user turn, no system message. The $n = 50$ extension for Qwen3-4B and Gemma used the same decoding with a new seed and is gated on exchangeability as described in Appendix K. Qwen3-4B's LCB corpus is spliced: its 611 stdin problems were regenerated under the stock CodeQwenInstruct prompt after the original run was found to have used a different prompt, and its 444 function-style problems are the original samples. Every Qwen3 number in the paper is on this corrected corpus.

Generated assertions. Each model wrote 100 test-writing samples per problem (max 8,192 output tokens; 1,024 for Phi) with the CodeT test-generation prompt (Chen et al., 2023) in its V1 form: the upstream `instruct` block (10–20 assertions, one per line, covering examples, edge cases, and boundaries) plus the instruction to precede each assertion with one comment line deriving the expected output step by step. The rerankers use the first five assertions of each sample. Deduplicated pool sizes on LCB: 109,040 (Qwen3), 281,655 (Phi), 266,563 (Qwen3.5), 181,333 (Gemma); on HumanEval+ 8,359 / 27,080 / 25,617 / 16,947. V1 was selected in a preregistered prompt sweep on a 60-problem LCB development subset: the true-assertion rate rose from 51.1% to 63.8% for Qwen3-4B and from 35.9% to 38.2% for Phi. V1 was then used for every model on both benchmarks; it is the only prompt for which every generator produced a pool on both benchmarks.

K $n = 50$ TABLES (QWEN3 AND GEMMA)

Forty further samples per problem (vLLM seed 1) were merged behind the original ten for Qwen3-4B and Gemma-4-E4B; indices 0–9 are the $n = 10$ corpus, 10–49 the new samples. Every problem keeps the bin it had at $n = 10$ (159 problems in Qwen3's 1–5 bin, 192 in Gemma's). Pools: *n10* (indices 0–9), *heldout40* (10–49), *all50*. Because heldout40 shares no candidate with the indices the bin was frozen on, its per-bin rows carry no selection effect; n10 and all50 do (pass@$N = 1$ in the 1–5 bin by construction). For both models, indices 0–9 of the $n = 50$ partition reproduce the frozen $n = 10$ tables exactly (every ledger column and per-candidate verdict, maximum difference zero).

Exchangeability gate. Pass@1 of the new 40 against the old 10, paired over problems. Qwen3 $+0.25$ points $[-0.28, +0.80]$ (function-style -0.02 $[-0.86, +0.83]$, stdin $+0.46$ $[-0.24, +1.16]$); Gemma -0.34 $[-0.89, +0.20]$ (function-style -0.60 $[-1.48, +0.26]$, stdin -0.15 $[-0.84, +0.54]$); both exchangeable. Qwen3's first gate, against its original $n = 10$ corpus, failed ($+1.51$ $[+0.86, +2.18]$, all of it on stdin problems, $+2.64$ $[+1.72, +3.58]$): every original Qwen3 stdin sample defines a `solve()` and about 90% use an `if __name__` guard, against about 20% and 0.5% in three independent later runs, because the stdin portion of that corpus had been generated under a different prompt. The stdin portion was regenerated under the stock prompt (Appendix J), and the gate above is on the corrected corpus.

Table 16: LiveCodeBench corpus-level selection from all 50 candidates with no example pre-filter. Selector cells give accuracy and, in parentheses, paired gain over the public-example filter with a 95% bootstrap interval. The oracle vote is a diagnostic reference.

Model	Filter	Agreement	CodeT	SRank	Oracle vote
Qwen3-4B	.584	.544 (-0.040 $[-0.053, -0.027]$)	.558 (-0.026 $[-0.038, -0.015]$)	.556 (-0.027 $[-0.040, -0.015]$)	.544 (-0.040 $[-0.054, -0.026]$)
Gemma-4-E4B	.624	.605 (-0.019 $[-0.031, -0.008]$)	.614 (-0.010 $[-0.020, -0.001]$)	.619 (-0.005 $[-0.015, +0.006]$)	.620 (-0.004 $[-0.017, +0.008]$)

Thus, direct selection from all 50 candidates does not exceed the recomputed example filter. All four Qwen3 comparisons are significantly below it; for Gemma, agreement and CodeT are below it while SRank and the oracle vote are statistically indistinguishable from it.

Table 17: LiveCodeBench 1–5 bin, bin frozen at $n = 10$ (159 problems for Qwen3, 192 for Gemma). The post-filter ledger on each pool; the oracle vote is over the survivors.

Model	Pool	Random	Filter	Oracle vote	Comparator share	Shared-bug cost	Lone-correct cost	pass@N	Gap
Qwen3-4B	n10	0.255	0.626	0.618	-0.007 $[-0.042, +0.028]$	0.214 [0.161, 0.271]	0.167 [0.118, 0.219]	1.000	0.374
Qwen3-4B	heldout40	0.289	0.608	0.632	$+0.024$ $[-0.013, +0.062]$	0.301 [0.234, 0.371]	0.036 [0.012, 0.065]	0.969	0.361
Qwen3-4B	all50	0.282	0.613	0.648	$+0.035$ $[-0.005, +0.076]$	0.335 [0.263, 0.409]	0.017 [0.000, 0.040]	1.000	0.387
Gemma-4-E4B	n10	0.293	0.558	0.589	$+0.032$ $[-0.004, +0.067]$	0.250 [0.197, 0.305]	0.161 [0.115, 0.208]	1.000	0.442
Gemma-4-E4B	heldout40	0.297	0.515	0.571	$+0.056$ $[+0.016, +0.095]$	0.341 [0.275, 0.410]	0.046 [0.020, 0.076]	0.958	0.443
Gemma-4-E4B	all50	0.296	0.527	0.591	$+0.065$ $[+0.024, +0.106]$	0.373 [0.307, 0.441]	0.035 [0.010, 0.061]	1.000	0.473

Against N on the held-out pool in the 1–5 bin ($N = 5$ / 10 / 25 / 40). Gemma: CodeT 0.029 / 0.045 / 0.064 / 0.069, SRank 0.034 / 0.049 / 0.065 / 0.078, agreement 0.011 / 0.031 / 0.052 / 0.066, oracle vote 0.011 / 0.029 / 0.047 / 0.056, ceiling 0.237 / 0.341 / 0.417 / 0.443. Qwen3: CodeT 0.025 / 0.039 / 0.058 / 0.065, SRank 0.022 / 0.038 / 0.051 / 0.047, agreement 0.008 / 0.017 / 0.035 / 0.040, oracle vote 0.002 / 0.011 / 0.027 / 0.024, ceiling 0.179 / 0.266 / 0.337 / 0.361. Every reranker is at or above the oracle vote at every N. At $N = 40$ the selectors reach 0.58–0.59 for Gemma (oracle vote 0.571 [0.503, 0.639]) and 0.65–0.67 for Qwen3 (0.632 [0.557, 0.704]).

Table 18: 1–5 bin, held-out pool, ledger against pool size N (200 subsamples without replacement per problem).

Model	N	pass@N	Filter	Comparator share	Shared-bug cost	Lone-correct cost
Qwen3-4B	5	0.696	0.517	0.002	0.070	0.106
Qwen3-4B	10	0.833	0.567	0.011	0.163	0.092
Qwen3-4B	25	0.937	0.600	0.027	0.259	0.052
Qwen3-4B	40	0.969	0.608	0.024	0.301	0.036
Gemma-4-E4B	5	0.718	0.481	0.011	0.089	0.137
Gemma-4-E4B	10	0.845	0.505	0.029	0.211	0.101
Gemma-4-E4B	25	0.931	0.515	0.047	0.316	0.054
Gemma-4-E4B	40	0.958	0.515	0.056	0.341	0.046

Table 19: Gain over the filter of CodeT / SRank / agreement / oracle vote on the filter survivors (existing V1 pool), corpus and 1–5 bin, with the share of the ceiling recovered.

Model	Pool	Corpus gain	Corpus ceiling	Share	1–5 gain	1–5 ceiling	Share
Qwen3-4B	n10	0.012 / 0.011 / 0.007 / 0.010	0.070	17% / 16% / 10% / 14%	0.037 / 0.037 / 0.008 / −0.007	0.374	10% / 10% / 2% / −2%
Qwen3-4B	heldout40	0.017 / 0.013 / 0.012 / 0.011	0.116	15% / 11% / 11% / 9%	0.065 / 0.047 / 0.040 / 0.024	0.361	18% / 13% / 11% / 7%
Qwen3-4B	all50	0.018 / 0.017 / 0.015 / 0.012	0.120	15% / 15% / 12% / 10%	0.066 / 0.069 / 0.048 / 0.035	0.387	17% / 18% / 12% / 9%
Gemma-4-E4B	n10	0.016 / 0.015 / 0.017 / 0.021	0.099	17% / 15% / 17% / 21%	0.049 / 0.037 / 0.049 / 0.032	0.442	11% / 8% / 11% / 7%
Gemma-4-E4B	heldout40	0.025 / 0.028 / 0.023 / 0.027	0.149	17% / 18% / 15% / 18%	0.069 / 0.078 / 0.066 / 0.056	0.443	16% / 18% / 15% / 13%
Gemma-4-E4B	all50	0.023 / 0.026 / 0.023 / 0.029	0.155	15% / 17% / 15% / 19%	0.064 / 0.077 / 0.068 / 0.065	0.473	13% / 16% / 14% / 14%

Bin transitions under $c_{50}/50$, using the same fractional cut points: 163 of Gemma's 192 and 138 of Qwen3's 159 originally minority-correct problems remain minority-correct; the rest become majority-correct. Sixty problems in each model's zero-correct bin gain a correct candidate, with 23 for each model gaining exactly one, and add 47.2 (Gemma) and 45.5 (Qwen3) problem-units of headroom to the all-50 pool. Of these totals, shared-bug cost is 26.2 and 29.0, lone-correct cost is 20.8 and 20.7, and comparator share is $+0.15$ and -4.2, respectively. Conversely, 113 of Gemma's 364 and 81 of Qwen3's 343 originally all-correct problems become majority-correct. Failure split at $n = 50$: the majority wrong class among survivors is too-slow on 105 of 126 minority-correct problems (83%) for Gemma and 62 of 107 (58%) for Qwen3; of the problems the zero-correct bin adds, 41 of 47 and 26 of 48 are too-slow.

X2 refit on $c_{50}/50$ (n10 / heldout40 / all50): agreement's catch slope Gemma 4.62 / 4.65 / 4.65, Qwen3 3.17 / 3.51 / 3.30; the examples' Gemma 0.79 / 0.93 / 0.90, Qwen3 1.05 / 1.07 / 1.06; interactions $p_i\times$ agreement Gemma 3.83 / 3.72 / 3.75, Qwen3 2.12 / 2.44 / 2.24, and $p_i\times$ self-asserts Gemma 2.85 / 2.58 / 2.72, Qwen3 2.22 / 2.24 / 2.35, every interval excluding zero. The new candidates that a public example rejected although a label calls them correct are all wrong candidates, as at $n = 10$: Gemma 122 (106 verified against a reference, 16 against the dataset's expected output), Qwen3 151 (130 and 21).

L Statistics

Every per-bin quantity is a mean over problems with a 95% percentile bootstrap interval, 10,000 replicates, seed 20260916. Every gain over the filter is a paired bootstrap: one resampled problem index, both arms read at it. Shares of a gap are ratio bootstraps over the same resamples. Slopes and interactions are binomial logistic regressions on grouped per-problem counts, fitted by IRLS (ridge 10^{-8} as a numerical floor only), with intervals from 2,000 problem-level resamples, each refitting the model; rows within a problem are not treated as independent. Where the unit is a candidate or an assertion rather than a problem (the shared-bug pass-through), Wilson score intervals are reported. Spearman correlations are computed on ranks. All tables carry the model key; nothing pools across models except the cross-verifier interaction fit (Appendix H), which is conditioned on the cell.

M Intervals for Quantities Reported in the Text

Table 20 collects quantities that the main text states without an interval and that no other table carries. The 95% intervals follow Appendix L.

Table 20: Text-only quantities with their intervals.

Quantity	Model	Section	Estimate [95% CI]
Public-example catch rate on wrong candidates, HumanEval+ 1–5 bin	Qwen3.5-4B	4.1	0.49 [0.32, 0.66]
Public-example catch rate on wrong candidates, LCB 1–5 bin	Qwen3.5-4B	4.1	0.84 [0.82, 0.86]
Public-example catch rate on wrong candidates, LCB 1–5 bin	Gemma-4-E4B	4.1	0.54 [0.49, 0.59]

The $n = 50$ exchangeability gate and comparator share are in Appendix K; the self-judge 1–5 gains are in Table 14.

N The Filter Is Not a Definitional Artifact: Further Checks

Hidden-only regrade. Removing the public tests from the grading suite leaves the example-filter survivor sets and reported filter accuracies unchanged. Thus, the filter's advantage is not created by counting the same examples both in filtering and in the final correctness label.

Example-soundness audit. Our audit of every example rejection finds no correct program rejected and no defective task-provided example among the rejected cases.

Additional checks. Three further checks point the same way.

- **The private tests are not a superset of the public ones.** LCB's private test set contains all public tests on 2.3% of problems, and only 6.6% of individual public tests recur in the private set (the private sets were regenerated from reference solutions rather than copied from the statement).
- **The examples catch what the hidden suite misses.** On 2–5% of problems in the 1–5 and 6–9 bins, the examples reject a candidate that passes the hidden suite. This runs in the examples' favor, not against it.
- **HumanEval+ overlap.** On HumanEval+ the docstring examples overlap the grader on about three quarters of tasks, which is one reason HumanEval+ is secondary.