MORE PROGRAMS OR MORE ROLLS? SEPARATING COVERAGE FROM SPECIALIZATION IN LLM HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Automated generation of LLM harnesses promises to improve inference through task specialization. Yet additional answer coverage can arise from repeated execution of the same program, making specialization difficult to identify. We introduce a controlled evaluation that separates answer coverage, repeatable task advantages, and gains from pre-execution selection. On 386 MATH-500 tasks, we compare eight generated harnesses plus a baseline with nine byte-identical baseline copies, using three executions per member. Identical programs yield 2.16 percentage points of repeat-averaged oracle headroom. Generated programs exhibit substantially more repeatable score patterns, but these chiefly reveal persistent weaknesses: losses relative to the baseline persist across all three repeats on 100 tasks, while persistent wins occur on only one task and are sensitive to answer extraction. The frozen selector gains 0.00 percentage points, and both populations reach 98.70% oracle coverage at 27 harness executions. Stable complementarity remains unresolved at three repeats. Supporting BIRD traces locate failures in mechanism implementation, activation, and output validity. Together, these findings establish why coverage and repeatability alone cannot justify claims of useful specialization. They motivate an evaluation standard for harness diversity: task advantages should persist across executions, guide usable decisions, and improve on additional fixed-program executions under matched inference budgets.

1 INTRODUCTION

Automated agent design allows language models to write the programs that organize a solver's inference. These LLM harnesses arrange prompts, tools, and control flow around a fixed solver (Hu et al., 2024; Nie et al., 2026; Zhang et al., 2026b). A population of such programs offers an attractive possibility: a division of labor in which different members address different task-specific failures. Realizing this possibility would let a system allocate inference to the program best suited to each task. It raises a concrete question: when should a fixed budget buy more programs, and when would further executions of one program suffice?

The most immediate evidence for this division of labor is also ambiguous. A population can contain more correct answers than any single member simply because it provides more attempts. Pass@k measures the probability of at least one correct answer among k samples; self-consistency and sampling-and-voting ensembles turn repeated attempts into an aggregated answer (Chen et al., 2021; Wang et al., 2022; Li et al., 2024). An oracle that selects after observing correctness can exploit complementary successes even when every member has identical code. Additional coverage therefore leaves open how much the population benefits from having different programs.

Repeating the evaluation introduces a second distinction. With few repeats, oracle estimates can still select favorable estimation errors; even strongly repeatable score patterns can reflect a program's persistent weaknesses. Exploiting specialization before execution requires task advantages that recur and can be recognized from the task. We therefore distinguish *answer coverage*, *repeat-stable task advantages*, and *usable selection*. These distinctions turn the promise of program specialization into an evaluation problem: identify which advantages persist, which can guide a choice, and what they add beyond repeated execution of a fixed program.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

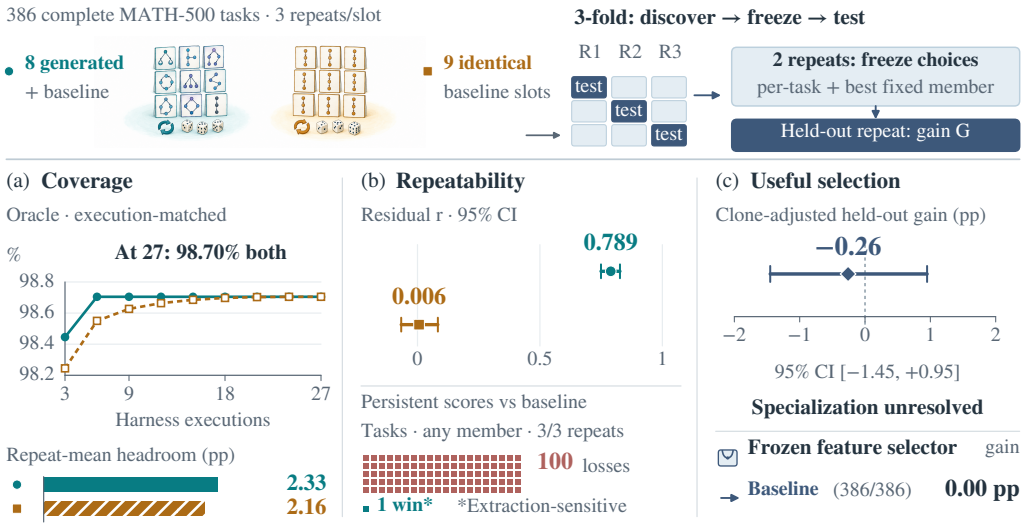


Figure 1: **Coverage, repeatability, and selection on** 386 **complete MATH tasks.** Oracle coverage matches executions, not tokens/calls; headroom uses repeat means. Persistent task counts require at least one generated member to win/lose against baseline in all three repeats; the sole win is extraction-sensitive. Here G is held-out task-wise selection gain over the discovery-selected fixed member; panel (b) labels residual Pearson correlation as r. R1–R3 are repeat labels; pp denotes percentage points and CI denotes confidence interval. Intervals are 95% bootstrap CIs. Program drawings are schematic.

We investigate this problem on MATH-500, starting with the coverage offered by a generated population. We then compare a selected low-call panel with byte-identical baseline copies, holding the solver, population size, and repeat count fixed. Choices learned from earlier executions are tested on held-out repeats. A selector frozen on development data tests pre-execution choice, and oracle replay compares coverage at equal harness execution counts. Separate BIRD traces examine whether proposed mechanisms are implemented, triggered, and able to return valid answers. This sequence follows the evidence needed to move from additional answers to useful specialization.

The central empirical finding is that the generated panel's repeatable differences predominantly identify weaknesses (Figure 1). Persistent losses span 100 tasks, while persistent wins occur on one task and are sensitive to answer extraction. Identical programs already yield positive oracle headroom. The clone-adjusted held-out gain is -0.26 pp, leaving stable complementarity unresolved at three repeats. The frozen selector gains 0.00 pp, and both arms reach 98.70% oracle coverage at 27 executions. Thus, substantial repeatability coexists with zero gain for the tested selector and matching full-budget coverage from further baseline executions.

The study provides a controlled basis for attributing population gains: a same-code sampling reference, a directional analysis of repeated wins and losses, and separate evaluations of pre-execution choice and post-execution coverage. Together with the execution traces, these analyses connect program differences to the behavior and decisions through which they could become useful. The resulting design target is a task advantage that persists, guides a usable decision, and adds value beyond further attempts with a fixed program.

2 Related Work

Automated agent and harness design. Executable programs provide a flexible space for adapting LLM behavior. ADAS searches this space directly (Hu et al., 2024); Self-Harness uses failure traces to propose edits and regression tests to retain them (Zhang et al., 2026b); TTHE adapts the harness during evaluation from unlabeled traces (Nie et al., 2026). Related systems improve task adaptation and validation allocation (Zhang et al., 2026a; Miyai et al., 2026), or expose harness behavior through verification, diagnosis, localization, and policy evolution (Xu et al., 2026; Chen

et al., 2026a; Lin et al., 2026; Luo et al., 2026; Wang et al., 2026; Chen et al., 2026b). El et al. (2025) report low diversity in task-level score patterns among the automatically designed agents they study. We examine program populations through both execution traces and repeated outcomes: traces reveal which mechanisms execute, and same-code repeats measure coverage from sampling. This perspective connects program equivalence and mutation testing (Budd & Angluin, 1982; Papadakis et al., 2019) with performance attribution in generated harness populations.

Repeated sampling and ensembles. Multiple executions can improve answer coverage even with a fixed program. Pass@k measures success within k samples; self-consistency and identical-agent ensembles aggregate attempts (Chen et al., 2021; Wang et al., 2022; Li et al., 2024), while nominally temperature-0 calls can also vary (Ouyang et al., 2023). These gains provide the sampling reference in our execution-matched comparison. A second distinction concerns when selection occurs. Post-execution verifiers rank candidate solutions (Kwok et al., 2026); ensemble selection chooses a model combination on validation data and averages predictions at inference (Caruana et al., 2004). Our frozen selector chooses a program before execution, while oracle replay measures coverage across recorded outputs.

Selection and resource allocation. Turning a population into a useful choice connects harness design to algorithm selection and portfolios (Rice, 1976; Xu et al., 2008; Kotthoff, 2014). HELIX uses complementary sibling trajectories to expand verified coverage and supply data for model improvement (Fan & Huang, 2026). STS compares six reasoning paradigms and trains an embedding-based router to recover part of their oracle gap (Zhou et al., 2026). Harness sensitivity across model–problem pairs (Gupta et al., 2026) and evaluated models (Cho, 2026), search routing (Kim, 2026), and preference-trained model routing (Ong et al., 2024) further motivate task-dependent choices. GRASP selects edits to a bounded skill library under regression constraints (Moll et al., 2026). We hold the target model fixed and measure oracle coverage, repeat-stable task advantages, and realized selector gain in the same program population. The distinction between quantitative and qualitative interaction (Gail & Simon, 1985) clarifies why varying effect sizes can coexist with an unchanged best action.

3 Problem Formulation

We distinguish the answers a population provides, the task advantages that persist across executions, and the gains a selector realizes. Let P be a population of harness members containing the single-call baseline H_0. Write $h \in P$ for a member, x_i for task i among n evaluated tasks, and $r \in \{1, \dots, R\}$ for one of R execution repeats per member–task pair. The score $Y(h, x, r) \in \{0, 1\}$ is one exactly when member h answers task x correctly in repeat r. Equations use proportions; multiplying by 100 gives accuracies in % and gains or headroom in percentage points (pp). Appendix A summarizes the notation.

Coverage: which additional answers are available? A one-run oracle selects a member after seeing correctness. For a fixed repeat r, its headroom over the best fixed member in the recorded matrix is

$$H(P) = \frac{1}{n} \sum_{i=1}^{n} \max_{h \in P} Y(h, x_i, r) - \max_{h \in P} \frac{1}{n} \sum_{i=1}^{n} Y(h, x_i, r). \tag{1}$$

This quantity captures available correct answers, including those obtained through repeated sampling. The repeat-averaged plugin $\widehat{H}$ replaces $Y(h, x, r)$ in Equation 1 with $\hat{p}_h(x) = R^{-1} \sum_{r=1}^{R} Y(h, x, r)$ on complete observations. With few repeats, maximizing those means can select favorable estimation errors, which motivates the same-code control.

Stable complementarity: which member is best for each task? Write $p_h(x) = \Pr\{Y(h, x, r) = 1\}$ for member h's success probability over fresh executions under a fixed execution condition. The corresponding population headroom is $H_{\text{stable}} = \mathbb{E}_x \max_h p_h(x) - \max_h \mathbb{E}_x p_h(x)$, where $\mathbb{E}_x$ averages over the target task distribution and all member maxima range over P. For $P = \{H_0, H_1\}$, abbreviate $p_{H_j}(x)$ as $p_j(x)$ for $j \in \{0, 1\}$ and define $\Delta(x) = p_1(x) - p_0(x)$. Then

$$H_{\text{stable}} = \tfrac{1}{2}\big(\mathbb{E}_x|\Delta(x)| - |\mathbb{E}_x \Delta(x)|\big). \tag{2}$$

Positive complementarity requires task-wise crossover: the better member changes with the task. If one member is better everywhere, headroom is zero even when its advantage varies in size. Identical programs under the same condition have equal $p_h(x)$, giving a zero-complementarity reference. This definition concerns population probabilities; the repeat diagnostic below tests choices learned from finite executions.

For a concrete example, consider two equally common task types with success probabilities $(0.90, 0.60)$ for H_0 and $(0.80, 0.20)$ for H_1. The performance gap is task-dependent, but H_0 remains preferable on both types. The task-wise oracle and the best fixed member therefore both achieve 0.75, leaving zero stable headroom. This constructed example separates variation in the size of an advantage from a change in the preferred action. It motivates checking the direction of repeated differences, alongside their repeatability.

Usable selection: which advantage can guide a decision? A selector π_Z maps the pre-execution features $Z(x)$ of task x to a member $\pi_Z(x) \in P$. Its usefulness depends on whether those features predict the relevant advantage. We evaluate this decision separately from choices learned from earlier outcomes on the same task, which test repeat generalization. An execution-matched oracle replay then compares the coverage offered by a portfolio with additional fixed-program executions. Its budget counts harness executions; each execution may contain multiple model calls and tokens.

4 Experimental Design

4.1 Harness Populations and Evaluation Setup

We hold the target solver, GLM-5.3-Flash, fixed and vary the Python programs that call it. A *builder* writes each harness, the *solver* answers its model calls, and an external *judge* scores the final output. Gold answers are available only to the judge. Each harness receives a question and can query the fixed solver. BIRD harnesses also have access to database schemas and SQL execution feedback. The single-call baseline makes one temperature-0 call. Generated programs can change prompts, decoding, sample count, and control flow; interface and model details appear in Appendix B.

MATH-500 tests whether additional coverage survives repetition and supports useful program choices. Table 1 maps these questions to their comparisons. BIRD separately examines how proposed mechanisms execute and what admission screening changes.

Question	Key comparison	Readout
Additional coverage	Full pool vs. best fixed	Oracle headroom $H(P)$
Repeat transfer	Generated vs. same-code	Held-out gain D
Usable selection	Frozen selector vs. dev-fixed	Selection gain $S(\pi_Z)$
Budgeted coverage	Portfolio vs. repeated baseline	Oracle coverage

Table 1: **Comparisons separating coverage from usable specialization.** Oracle comparisons use correctness after execution; the frozen selector chooses from the question before execution. Dev-fixed denotes the development-selected fixed member. D is generated-panel minus clone held-out selection gain (Equation 3); $S(\pi_Z)$ is the frozen selector's accuracy gain over dev-fixed.

MATH tasks and scoring. MATH-500 (Lightman et al., 2023) is split into 100 development and 400 evaluation problems. All primary MATH comparisons use the same 386 evaluation problems, selected after collection by requiring complete observations in both repeat arms and the single-run population. This excludes 14 problems with incomplete repeat observations; Appendix C gives completeness counts, exclusions, and conflict sensitivity. The judge extracts the final answer, preferring the designated final-answer line, and compares numeric values when both parse or normalized LaTeX otherwise (Appendix B.1).

MATH generation and repeat panel. Six builders—GLM-5.3, Qwen3.8-Max, DeepSeek-V4-Pro, Kimi-K3, MiniMax-M3, and ERNIE-5.0-Thinking-Preview—each receive eight slots under generation seed 0. A slot is an independent opportunity to improve on a single greedy call: the

builder chooses the strategy, with no cross-slot diversity constraint. Each slot allows up to three attempts at temperature 0.7 and 32,768 output tokens, using the same prompt without feedback from earlier attempts. Generation stops at the first program that runs successfully and returns nonempty answers on two development questions. The 48 slots yield 35 frozen programs (builder instructions: Appendix B.1).

The repeat panel samples eight of these programs. A historical mean-call ceiling of 3.05 leaves 21 eligible members; a fixed salted hash selects one per builder and two at-large seats. The resulting low-call panel includes draft–check–revise, agreement checking, and sample voting (Appendix B.2). Adding single-call gives nine members. This panel and nine identical baseline slots each receive three fresh, cache-disabled repeats, with decoding fixed by the source.

The selected programs differ in how they obtain and resolve additional answers. Some request several samples and vote on normalized answers; others verify a draft, obtain an alternative solution, or request adjudication when answers disagree. One uses differently prompted solutions and stops when three agree. Verification and adjudication are additional prompts to the same solver, without access to gold answers. A single logical call can request five samples, and the 3.05 eligibility threshold is a historical mean rather than a per-task maximum. The inventory describes implemented operations; their presence alone does not show that each branch activates or improves an answer.

BIRD discovery and admission. BIRD (Li et al., 2023) requires SQL for a question and database. In the discovery study, TTHE uses a GLM-5.3-Flash builder to produce 12 candidates that pass basic execution checks. These programs and two supplied baselines are evaluated on 60 questions. The admission study crosses free versus assigned ("forced") strategies with ungated versus conformance-gated admission: six builders, three seeds, and eight slots per builder–seed cell in each arm, with up to three attempts per slot (temperature 0.7; 16,384 output tokens). Forced strategies include execution repair, three-sample voting, and schema linking. All arms require executable programs and nonempty SQL; gated arms also check mechanism conformance and exclude some structures.

The admission split uses 365 questions from two databases for development and 1,169 from nine databases for evaluation, with a pre-drawn 400-question core shared by all four arms. An 18-item smoke test is a development-exposure exception. The official set-of-rows scorer compares predicted and gold query results, with a 30-second timeout and failures scored zero. Discovery uses an earlier stringified-row proxy with a row cap. Admission uses shared response caching, whereas the MATH repeats disable caching. Full split, scoring, and bundled gating specifications appear in Appendices F.1 and G.

4.2 Same-Code Controls and Repeat Validation

Learning a choice on two repeats, testing it on the third. Each fold uses two discovery repeats to select a member for every task and a single best fixed member. The third repeat measures their accuracy difference. Averaging over tasks and the three folds gives gain G: G_{real} for the generated panel plus baseline and G_{clone} for the same-code arm. We compare paired task-level gains on the common complete tasks through

$$D = G_{\text{real}} - G_{\text{clone}}. \tag{3}$$

A positive difference means that choices learned from past outcomes transfer better for distinct programs than for identical ones. Both the task-wise choice and the fixed comparator are determined entirely by the discovery repeats. Thus, G evaluates a choice learned from finite observations: it depends on both the underlying task advantages and how accurately the discovery repeats identify them. D compares this learning procedure across the two arms. It is distinct from H_{stable}, which assumes access to the success probabilities themselves. Its confidence interval is therefore not an interval for H_{stable}; a near-zero D need not imply a small population-level opportunity.

Positive support requires $D > 1.0$ pp, positive 95% bootstrap lower bounds for both D and G_{real}, and randomization $p < 0.05$. Before subsetting, incomplete tasks must account for at most 10% of either arm or their paired intersection, with at least 30 paired tasks retained. These conditions combine effect size, uncertainty, and observation completeness; failure to satisfy them yields Insufficient Evidence.

Distinguishing repeatability from advantage. We denote by ρ_{res} the mean held-out-versus-discovery Pearson correlation of score residuals across repeats, after removing additive member and task effects. This exploratory measure describes repeatability, while G describes held-out selection gain. Known-truth simulations characterize the diagnostic's sensitivity. Appendices C.1 and D specify these calculations, tie handling, uncertainty estimates, and decision rules. The acquisition protocol preceded collection; the repeat diagnostic and common-complete-set analysis were developed afterward. Recovery retained the programs, solver, tasks, and repeat design (Appendix C).

Checking the diagnostic against known truth. Calibration includes equal-ability and global-dominance settings with $H_{\text{stable}} = 0$, alongside alternatives in which the best member changes with the task. In the strong two-specialist case, 400 tasks form two equal halves. Each specialist succeeds with probability 0.95 on its own half and 0.60 on the other; the remaining seven members have probability 0.60 throughout. The oracle accuracy is 0.95 and the best fixed accuracy is 0.775, giving 17.5 pp of true headroom. These settings distinguish false support from failure to detect an existing advantage. The simulation variant uses fewer resampling draws and checks completeness in the real arm; the empirical rule also checks clones and the paired intersection. Appendix D specifies both rules and their tie handling.

4.3 Frozen Selection and Execution-Matched Replay

Selecting a program from the question. Using only development data, we fit π_Z to predict the member with the highest available-repeat mean correctness on each task; training-label ties follow member-ID order. Five task folds keep each task's repeats together. The fitted models share a term-frequency–inverse-document-frequency (TF-IDF) vocabulary learned from development questions and use training-fold member accuracies as additional inputs. At inference, we average the fitted fold models' probability vectors and select the top member, falling back to single-call when the top-two gap is below 0.15, including a tie for the highest probability. The policy is frozen before evaluation; Appendix E specifies fitting and fallback details. Comparators include the development-selected fixed member $h_{\text{dev-fixed}}$ and one randomly drawn fixed member. We write $S(\pi_Z)$ for the policy accuracy minus that of $h_{\text{dev-fixed}}$, averaging over evaluation tasks and repeats.

Allocating executions to a portfolio. An offline oracle replay compares a development-ranked portfolio with same-code executions at per-task budgets $b \in \{3, 9, 15, 27\}$ harness executions. Each included member contributes three repeats. For clones, we compute exact mean coverage over uniform size-b subsets of the 27 recorded executions. Both arms use post-execution correctness to measure available answers. The recorded selector choices and portfolio order stay fixed for the recovery analysis. The budget unit is a harness execution, with model-call and token use determined by the program. Appendix E gives the policy, replay formula, and accounting details.

5 Results

5.1 The Population Covers More Tasks Than Any Single Member

The full generated population plus single-call reaches 99.22% oracle coverage, compared with 95.08% for the best fixed member, single-call. Every generated member is individually worse (69.69–92.49%), yet their union adds 4.15 pp of coverage available to an oracle after execution.

This coverage comes with substantial variation in scored outcomes: 29 of the 36 correctness vectors are unique, mean pairwise disagreement is 10.94% (maximum 33.16%), and the union repairs 84.21% of baseline errors. The next question is how much of this variation survives repetition. The following comparisons use the selected panel and its fresh executions, with baseline accuracy 95.85%, to distinguish repeatable advantages from sampling. Their population and executions differ from the full-pool single-run coverage reported here (§4.1).

5.2 Identical Programs Also Produce Positive Oracle Estimates

Identical programs produce a 2.16 pp repeat-averaged oracle estimate, compared with 2.33 pp for the generated panel (Table 2). The nine clone slots implement the same program, so their population-

level stable complementarity is zero by construction. Thus, even after averaging three repeats, a positive oracle estimate alone does not identify specialization.

Measure	Generated + single-call	Same-code clones
Oracle headroom $\widehat{H}$ (pp)	2.33	2.16
Residual correlation ρ_{res}	0.789	0.006
Held-out selection gain G (pp)	0.10	0.35
Clone-adjusted gain: $D = -0.26$ pp; 95% CI $[-1.45, 0.95]$ pp		

Table 2: **Coverage, repeatability, and selection gain separate.** MATH-500, selected panel versus the same-code control (§4.1). $\widehat{H}$ uses repeat means; G tests choices on held-out repeats. D is their paired, unrounded between-arm difference, so displayed means may not subtract exactly.

5.3 Repeatable Score Patterns Are Dominated by Losses

Generated programs have more repeatable scored-outcome patterns than clones. Their mean residual correlation is 0.789, compared with 0.006 for clones; the paired-bootstrap difference is 0.783 (95% interval $[0.69, 0.87]$). The correlation measures repeatability after removing additive member and task effects (§4.2).

Persistent differences overwhelmingly favor single-call. All-three-repeat losses span 100 tasks (570 member–task pairs), whereas all-three-repeat wins occur on one task shared by all eight generated members. That apparent repair is sensitive to answer-format extraction (Appendix C.1). This directional asymmetry explains how strong score repeatability can coexist with few observed wins.

The winning task illustrates why the scoring path matters. Its gold answer is `12`. The baseline ends with `#### 12th grade`, whose complete final-answer payload is retained and scored incorrect. A generated member returns `12th grade`, which instead reaches numeric fallback and is scored correct. The recorded advantage therefore reflects an answer-format difference. We retain the original scores; this example explains the sole persistent winning task, without assigning a common cause to the persistent losses.

Held-out selection provides no clear positive evidence beyond clones. The clone-adjusted mean is $D = -0.26$ pp (Table 2), with randomization p-value $p = 0.8127$. Its interval includes zero and the positive-support conditions are not met, giving Insufficient Evidence rather than an equivalence conclusion. Excluding the three tasks containing conflicting scores gives $D = -0.61$ pp on 383 paired tasks and the same decision (Appendix C).

Three repeats have limited power for crossovers. In the two-specialist setting described in §4.2, the simulated diagnostic has 0% power at $R = 3$ and 100% at $R = 10, 20$, despite true $H_{\text{stable}} = 17.5$ pp. These are observed detection rates over 300 simulated datasets at $R = 3$ and 150 at each higher repeat count. Smaller cyclic crossovers remain largely undetected even at $R = 20$ (Appendix Table 8). Increasing repeats therefore helps in the strong positive case without ensuring sensitivity to every crossover. The result characterizes this diagnostic and these settings, rather than all procedures using three repeats. It explains why the empirical nonpositive result leaves stable complementarity unresolved. We next evaluate the tested pre-execution decision rule directly, whose realized gain is a separate question.

5.4 The Tested Selector Adds No Gain and Full-Budget Coverage Ties

The frozen selector matches the development-selected baseline. The tested feature policy yields 0.00 pp gain. Its frozen classifier directly selects single-call on every evaluation task; the confidence-based fallback is never triggered. These choices give 95.85% accuracy over three repeats, identical to the development-selected fixed member. Appendix E gives its construction and the same-set comparator results.

At the full execution budget, the oracle coverages tie. At $b = 27$ executions, the development-ranked portfolio and same-code controls both reach 98.70% oracle coverage (Table 3). At $b = 9$

and 15, the portfolio-minus-clone differences are $+0.08$ pp and $+0.02$ pp. At $b = 3$, the portfolio contains only single-call, so the $+0.20$ pp difference reflects execution variation between identical programs.

Harness executions per task b	3	9	15	27
Panel members, including single-call	1	3	5	9
Panel oracle coverage (%)	98.45	98.70	98.70	98.70
Clone oracle coverage (%)	98.24	98.63	98.69	98.70
Panel minus clones (pp)	$+0.20$	$+0.08$	$+0.02$	$+0.00$

Differences are computed before rounding the displayed coverage values.

Table 3: **Full-budget oracle coverage ties in the recorded executions.** MATH-500 coverage from the development-ranked panel and uniform same-code sampling. Clone values are exact subset means (§4.3). Budgets count harness executions; model-call and token costs vary by program.

Together, the selector and replay results separate two operational questions: the tested pre-execution policy yields zero gain, while post-execution oracle coverage ties at the full recorded budget.

5.5 BIRD Traces Locate Breaks Between Code and Execution

Source variation can disappear before it produces a valid final answer. The supporting BIRD study follows this process directly in independent program populations, using its task-specific judging and protocols. Here a mechanism is an implemented prompt transformation, tool use, or control-flow operation. This execution view complements the MATH controls by examining what the programs actually do.

Different source code often produces identical scored outcomes. All 12 discovery candidates pass a code-difference check, yet $21/66$ generated-candidate pairs have identical correctness vectors ($36/91$ including baselines). Source similarity has essentially no Spearman rank correlation with correctness disagreement ($\rho = -0.010$). This agreement on 60 tasks motivates examining the traces (Appendix Figure 2).

Traces distinguish absent, inactive, and broken mechanisms. T1: described but absent. Six of twelve candidates describe retrieval or self-checking but make a single model call without an implemented retrieval or self-checking step. This classification covers all twelve candidates; the five-harness trace audit supplies worked examples.

T2: present but untriggered. For the execution-repair baseline, the first SQL executes successfully on all ten audited tasks, so the repair branch never fires and final SQL equals single-call. A successful execution can therefore leave a semantically wrong answer unchanged.

T3: executed but broken. One candidate runs multi-turn repair but returns prose as final SQL, losing tasks that the baseline answers correctly. The three cases locate distinct checks: implementation, branch activation, and valid output. Appendix F provides worked traces and hand-written controls, with their development-exposure limitations.

The separate BIRD admission study tests whether prescribing strategies and screening programs increases headroom. Its implemented gate combines structural exclusion and conformance verification. Forced/gated minus free/ungated generation gives -0.16 pp, with no increase in observed mean headroom (Appendix G). The admission comparison measures what screening changes; the traces locate where proposed operations fail to deliver useful answers. Together, these BIRD studies locate execution failures that complement the MATH evidence about persistence and selection.

6 Discussion and Limitations

A program population can provide additional answers and highly repeatable score patterns while leaving useful specialization unestablished. Our same-code control exposes the first gap: byte-identical programs have zero stable complementarity by construction, yet produce positive oracle headroom even after averaging repeats. This headroom captures a real opportunity to obtain correct

answers. Attributing that opportunity to program identity, however, requires comparison with what further executions of a fixed program already provide. The central evaluation question is therefore what changing programs adds when additional sampling is also available.

The generated panel's strong residual correlation chiefly identifies recurrent weaknesses in the recorded scores. Equation 2 clarifies why the direction matters: variation in the size of a disadvantage can be highly repeatable without changing which member is preferable. Repeatability is informative about where a population fails; demonstrating a division of labor additionally requires different members to retain advantages on different tasks.

Mechanisms should target identifiable failures. The separate BIRD traces make the design problem concrete. An operation must be implemented, activated on the relevant input, and produce a valid answer before it can contribute to performance. The inactive repair branch illustrates a mismatch between a mechanism and the failure it needs to address: SQL can execute successfully while answering the question incorrectly, leaving an execution-error trigger unable to address that failure. Designing a mechanism therefore also means specifying the task conditions under which it should change an answer. A targeted follow-up would compare enabled and disabled versions of the same mechanism on tasks that exercise its intended trigger, then test any advantage on fresh executions. Such interventions would connect intended behavior to a causal contribution and test whether that contribution persists.

Population value depends on a usable decision. For pre-execution routing, useful task advantages must be recognizable from information available before the answer is known. A portfolio that runs several programs instead needs an answer-selection or aggregation procedure that can turn its outputs into a better final answer. Our frozen selector and oracle replay address different parts of this problem: the former retains single-call on every task, while the latter measures coverage after correctness is known. Their respective zero gain and full-budget coverage tie make additional fixed-program executions an essential comparator. Future evaluations should combine this comparator with repeat counts calibrated to detect task-wise crossovers and an implementable decision rule under a matched resource budget. This would establish whether the advantages associated with program identity improve inference beyond the sampling opportunities a larger population creates.

Limitations and unresolved questions. The MATH evidence concerns one solver, a selected low-call panel, and three repeats. Known-truth calibration shows that this repeat count can miss substantial task-wise crossovers for the tested diagnostic. The INSUFFICIENT EVIDENCE decision leaves stable complementarity unresolved and does not establish equivalence. The repeat diagnostic and common-complete-set analysis were developed after collection; recovery retained the programs, tasks, solver, and repeat count. The 14 excluded tasks are harder than the 386 included tasks: baseline repeat-mean accuracy is 71.43% on excluded tasks and 95.85% on included tasks (Appendix C). Recorded scores include answer-format effects. The selector result concerns the tested policy and features. Replay matches harness executions, with model calls and tokens varying by program, so the observed tie does not establish equal-cost utility. BIRD uses separate populations and scoring conditions; its traces identify execution failures without establishing the causes of the MATH losses.

7 CONCLUSION

The value of a harness population depends on how its differences improve inference. Our same-code control and repeated executions show why coverage, repeatability, and usable specialization require separate evidence: identical programs yield positive oracle headroom, while generated programs exhibit persistent differences dominated by losses. On the tested MATH panel, the frozen selector gains 0.00 pp, and repeated baseline executions match the portfolio's 98.70% oracle coverage at 27 harness executions. Stable complementarity remains unresolved at three repeats.

These findings give automated harness design a concrete objective: develop reproducible task advantages that a usable decision rule can exploit. To justify additional programs through specialization, evaluation must show what those decisions gain over spending the relevant inference budget on further executions of a fixed program.

Ethics Statement

All experiments use public BIRD and MATH-500 benchmark data and generated program artifacts; no human participants, private records, or personally identifiable information are collected. Model calls are reported as part of the experimental provenance, and the paper does not release credentials or provider responses that would expose private data. The results distinguish recorded answer coverage from the performance of a selector operating before execution.

Reproducibility Statement

The manuscript specifies the populations, task splits, execution conditions, scoring rules, statistical procedures, and sensitivity analyses used for the reported results. The appendices provide additional protocol details, calibration results, and worked examples.

AI use statement

Generative AI tools assisted with research questions, experimental and statistical design, counterexample analysis, code development, model-API workflows, data analysis, figures, and manuscript preparation. Versioned artifacts record the analyses and revisions; automated tests, local runtime checks, and same-model subagents supported mechanical verification. The human authors set the research direction and retain responsibility for scientific decisions, verification, the final content, and submission.

References

Timothy A. Budd and Dana Angluin. Two notions of correctness and their relation to testing. *Acta Informatica*, 18(1):31–45, 1982. doi: 10.1007/bf00625279.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the 21st International Conference on Machine Learning*, 2004. doi: 10.1145/1015330.1015432.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. doi: 10.48550/arXiv.2107.03374. URL https://arxiv.org/abs/2107.03374.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026a. doi: 10.48550/arXiv.2606.06324. URL https://arxiv.org/abs/2606.06324.

Mingju Chen, Can Lv, Guibin Zhang, Heng Chang, and Shiji Zhou. HarnessForge: Joint harness and policy evolution for adaptive agent systems. *arXiv preprint arXiv:2606.01779*, 2026b. doi: 10.48550/arXiv.2606.01779. URL https://arxiv.org/abs/2606.01779.

Yong-eun Cho. It's not the capability: Harness sensitivity is non-monotone across LLM agent tiers. *arXiv preprint arXiv:2605.26731*, 2026. doi: 10.48550/arXiv.2605.26731. URL https://arxiv.org/abs/2605.26731.

Batu El, Mert Yuksekgonul, and James Zou. Inefficiencies of meta agents for agent design. *arXiv preprint arXiv:2510.06711*, 2025. doi: 10.48550/arXiv.2510.06711. URL https://arxiv.org/abs/2510.06711.

Tianyu Fan and Chao Huang. HELIX: Model-harness co-evolution for recursive self-improvement. *arXiv preprint arXiv:2608.13951*, 2026. doi: 10.48550/arXiv.2608.13951. URL https://arxiv.org/abs/2608.13951.

Mitchell H. Gail and Richard Simon. Testing for qualitative interactions between treatment effects and patient subsets. *Biometrics*, 41(2):361–372, 1985. doi: 10.2307/2530862.

Akshat Gupta, Jermaine Lei, Alexander Lu, Gopala Anumanchipalli, and Leshem Choshen. Automated discovery has no universally superior harness. *arXiv preprint arXiv:2607.18235*, 2026. doi: 10.48550/arXiv.2607.18235. URL https://arxiv.org/abs/2607.18235.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024. doi: 10.48550/arXiv.2408.08435. URL https://arxiv.org/abs/2408.08435.

Minho Kim. When should LLMs search? counterfactual supervision for search routing. *arXiv preprint arXiv:2607.05752*, 2026. doi: 10.48550/arXiv.2607.05752. URL https://arxiv.org/abs/2607.05752.

Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35 (3):48–60, 2014. doi: 10.1609/aimag.v35i3.2460.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. LLM-as-a-verifier: A general-purpose verification framework. *arXiv preprint arXiv:2607.05391*, 2026. doi: 10.48550/arXiv.2607.05391. URL https://arxiv.org/abs/2607.05391.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, volume 36, pp. 42330–42357, 2023. doi: 10.52202/075280-1835. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More agents is all you need. *arXiv preprint arXiv:2402.05120*, 2024. doi: 10.48550/arXiv.2402.05120. URL https://arxiv.org/abs/2402.05120.

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. *arXiv preprint arXiv:2305.20050*, 2023. doi: 10.48550/arXiv.2305.20050. URL https://arxiv.org/abs/2305.20050.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026. doi: 10.48550/arXiv.2604.25850. URL https://arxiv.org/abs/2604.25850.

Xiaotian Luo, Dizhan Xue, Fengxingyu Wang, Chuanrui Hu, and Yafeng Deng. HarnessBank: Semantic gene-bank search with gated verification for agent-harness self-evolution. *arXiv preprint arXiv:2607.13683*, 2026. doi: 10.48550/arXiv.2607.13683. URL https://arxiv.org/abs/2607.13683.

Atsuyuki Miyai, Kiyoharu Aizawa, and Toshihiko Yamasaki. Task-CoEvolve: Efficient harness optimization via adaptive validation task selection. *arXiv preprint arXiv:2608.20169*, 2026. doi: 10.48550/arXiv.2608.20169. URL https://arxiv.org/abs/2608.20169.

Johannes Moll, Jean-Philippe Corbeil, Jiazhen Pan, Martin Hadamitzky, Daniel Rueckert, Lisa Adams, and Keno Bressem. GRASP: Gated regression-aware skill proposer for self-improving LLM agents. *arXiv preprint arXiv:2605.29668*, 2026. doi: 10.48550/arXiv.2605.29668. URL https://arxiv.org/abs/2605.29668.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. TTHE: Test-time harness evolution. *arXiv preprint arXiv:2607.08124*, 2026. doi: 10.48550/arXiv.2607.08124. URL https://arxiv.org/abs/2607.08124.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*, 2024. doi: 10.48550/arXiv.2406.18665. URL https://arxiv.org/abs/2406.18665.

Shuyin Ouyang, Jie M. Zhang, Mark Harman, and Meng Wang. An empirical study of the non-determinism of ChatGPT in code generation. *arXiv preprint arXiv:2308.02828*, 2023. doi: 10.48550/arXiv.2308.02828. URL https://arxiv.org/abs/2308.02828.

Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. Mutation testing advances: an analysis and survey. In *Advances in Computers*, volume 112, pp. 275–378. Elsevier, 2019. doi: 10.1016/bs.adcom.2018.03.015.

John R. Rice. The algorithm selection problem. In *Advances in Computers*, volume 15, pp. 65–118. Elsevier, 1976. doi: 10.1016/s0065-2458(08)60520-3.

Ruhan Wang, Yucheng Shi, Zongxia Li, Zhongzhi Li, Yue Yu, Junyao Yang, Kishan Panaganti, Haitao Mi, Dongruo Zhou, and Leoweiliang. Harness handbook: Making evolving agent harnesses readable, navigable, and editable. *arXiv preprint arXiv:2607.13285*, 2026. doi: 10.48550/arXiv.2607.13285. URL https://arxiv.org/abs/2607.13285.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. doi: 10.48550/arXiv.2203.11171. URL https://arxiv.org/abs/2203.11171.

Jinghan Xu, Yikai Zhang, Aili Chen, Weiyuan Li, Jiaqing Liang, and Deqing Yang. Verify smarter, evolve further: Efficient harness evolution through behavior-aware verification. *arXiv preprint arXiv:2608.27311*, 2026. doi: 10.48550/arXiv.2608.27311. URL https://arxiv.org/abs/2608.27311.

Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008. doi: 10.1613/jair.2490.

Guibin Zhang, Leo Lu, Fangzhou Xie, Kang Zhu, Junhao Wang, Zhifei Xie, Zhaochen Yu, Zihang Liu, Zhongxiang Sun, Qiankun Li, Yue Liao, Heng Chang, Xiaobin Hu, Qibing Ren, Wangchunshu Zhou, Chuanrui Hu, Yafeng Deng, and Shuicheng Yan. JIT-Agent: Scaling harness intelligence via just-in-time harness evolution. *arXiv preprint arXiv:2608.25593*, 2026a. doi: 10.48550/arXiv.2608.25593. URL https://arxiv.org/abs/2608.25593.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026b. doi: 10.48550/arXiv.2606.09498. URL https://arxiv.org/abs/2606.09498.

Heng Zhou, Zelin Tan, Zhemeng Zhang, Yutao Fan, Yibing Lin, Li Kang, Xiufeng Song, Rui Li, Songtao Huang, Ao Yu, Yuchen Fan, Yanxu Chen, Kaixin Xu, Xiaohong Liu, Yiran Qin, Philip Torr, Chen Zhang, and Zhenfei Yin. Select-then-solve: Paradigm routing as inference-time optimization for LLM agents. *arXiv preprint arXiv:2604.06753*, 2026. doi: 10.48550/arXiv.2604.06753. URL https://arxiv.org/abs/2604.06753.

APPENDIX GUIDE

The appendices document program generation, repeated execution, selection, calibration, and the supporting BIRD studies.

A Notation summary.
B Harness generation, program inventory, and data splits.
C Same-code controls and repeated-execution protocol.
D Statistical analysis, calibration, and sensitivity.
E Frozen selector and execution-matched replay.
F BIRD execution traces and supporting analyses.
G BIRD admission-policy study.

A NOTATION SUMMARY

Table 4 collects the notation used in the main text and the analysis appendices. Mathematical scores and gains use proportions; reported accuracies use % and reported gains use percentage points (pp). Thus a gain of 0.01 is 1 pp. Population expectations, finite-sample estimates, and held-out gains are distinct quantities.

Table 4: Symbols, definitions, and scope.

Notation	Meaning
Members, tasks, and outcomes (§3)	
P, h, M	Harness population, an indexed member $h \in P$, and population size $M = \lvert P \rvert$. Identical clone slots have distinct member identities.
H_0, H_1	Single-call baseline and the other member in the two-member identity.
x, x_i, i, n	A task, evaluated task i, its index, and the number of evaluated tasks.
r, R	Execution-repeat index and number of repeats per member–task pair; empirical repeat comparisons use $R = 3$.
$Y(h, x, r)$	Binary correctness of member h on task x in repeat r.
$p_h(x)$; $p_0(x), p_1(x)$	Success probability over fresh executions at fixed conditions. For two members, $p_j(x) = p_{H_j}(x)$, $j \in \{0, 1\}$.
$\hat{p}_h(x)$	Complete-repeat estimate $R^{-1} \sum_{r=1}^{R} Y(h, x, r)$.
$\mathbb{E}_x$, Pr	Expectation over the target task distribution and probability. In the Jensen argument, unsubscripted $\mathbb{E}$ instead averages execution sampling at a fixed task.
$H(P)$	Single-run oracle accuracy minus the best fixed member's accuracy on the same tasks and repeat (Equation 1).
$\widehat{H}$	Repeat-averaged plug-in headroom: replace Y in $H(P)$ by $\hat{p}_h(x)$ before taking either maximum.
H_{stable}	Population headroom computed from the success probabilities $p_h(x)$.
$\Delta(x)$	Two-member success-probability difference $p_1(x) - p_0(x)$ (Equation 2).
Repeat diagnostics, selection, and budgets	
G; $G_{\text{real}}, G_{\text{clone}}$	Mean held-out accuracy gain of the discovery-selected task-wise mapping over the discovery-selected fixed member; subscripts identify the generated-plus-baseline and same-code arms (§4.2).
D	Paired clone-adjusted gain $G_{\text{real}} - G_{\text{clone}}$, a finite-repeat diagnostic (Equation 3).
ρ_{res}	Mean Pearson correlation of residual score patterns between each held-out repeat and the other repeats' mean (Appendix C.1); labeled residual r in Figure 1.
ρ	BIRD Spearman rank correlation between source similarity and correctness disagreement (§5.5).

Continued on next page

Table 4 (continued)

Notation	Meaning
p; CI	Randomization p-value for the MATH diagnostic; confidence interval. A scalar p-value is distinct from success probability $p_h(x)$.
$Z(x)$, $\pi_Z(x)$	Pre-execution task features and the member selected from them; the implemented features use question text and training-fold member accuracies.
$h_{\text{dev-fixed}}$	Fixed member chosen by development accuracy, before evaluation.
$S(\pi_Z)$	Evaluation-task mean of repeat-mean accuracy differences between the frozen feature selector and $h_{\text{dev-fixed}}$ (Appendix E).
b	Number of harness executions per task in oracle replay, $b \in \{3, 9, 15, 27\}$; distinct from model-call and token counts.
k in Pass@k	Number of sampled answers; Pass@k measures the probability that at least one is correct.
Additional notation in the analysis appendices	
h_0, $A_r(h, x)$	Arm-specific reference (single-call or clone-c1) and baseline-relative score $Y(h, x, r) - Y(h_0, x, r)$ (Appendix C.1).
T	Number of simulated tasks in calibration; M and R retain their member-count and repeat-count meanings (Appendix D).
$U(0, 10^{-9})$	Uniform distribution used for independent random tie-breaking jitter.
θ^*	A given selector's population gain over the population-best fixed member; bounded above by H_{stable}.
$\mathcal{R}_{hx}$, $\bar{Y}_h(x)$	Observed development-repeat indices and their mean score. In evaluation, $\bar{Y}_h(x)$ averages all three repeats (Appendix E).
$y(x)$	Member-ID training label maximizing $\bar{Y}_h(x)$; distinct from binary correctness $Y(h, x, r)$.
C	Inverse regularization strength of the logistic classifier, fixed at 1.
K, k, $q_k(h \mid x)$	Number of fitted fold models ($1 \le K \le 5$), model index, and model k's predicted member-class probability. Here k indexes models, rather than answer samples.
$\bar{q}(h \mid x)$; $h_{(1)}, h_{(2)}$	Mean of fitted models' class probabilities; the two highest-probability members. A gap below 0.15 triggers baseline fallback.
f, $\binom{f}{b}$	Incorrect clone executions among 27 for a task, and the number of size-b subsets containing only incorrect executions; zero when $b > f$.
K_{cand}, K_{total}	Admitted BIRD candidate count without and with the baseline: $K_{\text{total}} = K_{\text{cand}} + 1$. K-matching matches candidate counts (Appendix G).
II-A–II-D; A–D	BIRD free/ungated, free/gated, forced/ungated, and forced/gated arms. A–D denote arm headrooms in factorial formulas; their D−A contrast differs from the MATH statistic D.

B Harness Generation, Program Inventory, and Data Splits

This appendix describes the MATH population, splits, frozen panel, and model versions. BIRD protocols appear in Appendices F and G.

Execution interface. Harnesses implement `solve(question)` and access the frozen solver through `self.llm`. BIRD additionally provides the database schema as `self.schema`; `self.execute(sql)` returns query results or execution errors. Gold answers remain external to these interfaces. The execution-repair baseline is labeled `react` in archived programs and traces.

B.1 MATH population, common analysis set, and coverage

The single-run study supplies coverage measurements and programs for the repeat panel, applying BIRD's free, ungated generation procedure to competition mathematics with a domain-specific comparator.

Protocol and analysis set. The builder instruction, summarized rather than quoted, requests a Python `solve(question)` harness that improves on one greedy call through `self.llm`, with

freely chosen prompting, decoding, sampling, and control flow and a final #### answer. Gold answers remain external. Slots receive no assigned mechanism or diversity target. Attempts reuse the instruction without prior code or performance feedback; admission requires a valid interface and nonempty output.

Six builders each receive eight slots under one seed, with at most three attempts per slot. Parsing, import, interface, and nonempty answers are checked on two development questions. The target is GLM-5.3-Flash, with member-specific decoding and temperature 0 for single-call. The pre-generation split reserves 100 development problems. Single-run, repeat, and policy analyses share 386 fully observed tasks; the post-collection completeness rule excludes 14 problems (Appendix C). Development tasks and the selector stay fixed.

The judge compares parsed numbers, including fractions and decimals, or otherwise normalized LaTeX. Normalization removes units, base subscripts, and degree symbols, retains text-command content, and maps matrices to tuples. Gold answers prefixed with #### pass $386/386$ self-comparisons. Single-run baseline accuracy on this set is 95.08%.

The gate admits $35/48$ slots. GLM-5.3 and Qwen3.8-Max exhaust the 32k-token generation budget before emitting code in most slots, leaving their styles under-represented in this fixed-budget population.

Results on the common complete set. All 36 members, including single-call, have one score per task: $13{,}896/13{,}896$ cells over 386 tasks. Mean pairwise disagreement is 10.94% (maximum 33.16%), with $29/36$ unique outcome vectors ($28/35$ generated) and 2.22% identical pairs. The generated union repairs 84.21% of baseline errors; all generated members fail on 0.82% of baseline-correct tasks. Oracle coverage of 99.22% exceeds the best fixed member single-call (95.08%) by 4.15 pp. Generated accuracies span 69.69%–92.49%.

The repeat panel compares eight generated members plus single-call with nine same-code slots on identical tasks. Its held-out diagnostic concerns this panel; single-run headroom concerns the full population. Both the archived judge and the versioned endpoint-sensitive judge reproduce all $13{,}896$ included scores.

Scope. These measurements describe one seed and one target on the completeness-selected MATH set; BIRD uses the separate protocols below.

B.2 Selected MATH Program Inventory

Table 5 describes the nine frozen members selected for the repeat study. H1–H8 are neutral labels for generated members, all constructed under seed 0. The listed operations describe what each program implements, not whether every branch fires or improves correctness. The mean-call eligibility ceiling is not a per-execution maximum, and one logical call may request several samples.

Frozen member	Source-defined execution path
single-call	One temperature-0 solution with the final-answer format.
H1 (DeepSeek)	Three temperature-0.4 solutions; normalized-answer vote; a greedy fallback when no answer or a tie remains.
H2 (ERNIE)	Draft a solution, then ask the solver to verify/correct the extracted answer; return the checked answer.
H3 (GLM)	Up to five differently prompted temperature-0 solutions; stop when three agree; adjudicate unresolved disagreement, with format fallbacks.
H4 (Kimi)	Request five temperature-0.7 samples in one call; extract answers and vote.
H5 (Kimi)	Initial solution, verification, and an alternative solution; vote, with an additional resolution call if no majority emerges.
H6 (MiniMax)	Draft, critique, and an additional tie-breaking call on disagreement; parse failures use an earlier available answer.
H7 (MiniMax)	Request five temperature-0.7 samples in one call; normalize and vote, resolving ties by first occurrence.
H8 (Qwen)	Draft plus an independent solve-and-audit call; compare answers and request adjudication on conflict, with a format-repair path.

Table 5: **What was selected for the repeat study.** All calls use the same frozen solver; verification and adjudication are solver prompts, not access to benchmark labels. Six seats are builder-stratified; H5 (Kimi) and H7 (MiniMax) are the two at-large seats.

B.3 Models and versions

Table 6 gives the exact API model identifiers. All roles use one commercial OpenAI-compatible aggregator. Target identity is fixed; decoding follows each harness, with temperature 0 for the single-call baseline. Phase-II generation uses temperature 0.7 and a 16,384-token budget; Phase-I conditions appear in Appendix F.1. Phase I uses a proxy execution judge, Phase II the official BIRD scorer (30 s timeout) with a secondary legacy-judge audit, and MATH-500 the answer comparator in Appendix B.1. Reported accuracy uses no LLM judge.

Role	Model (exact version string)	Used in
Frozen target solver	`GLM-5.3-Flash` (baseline temperature 0)	MATH / BIRD
BIRD discovery builder	`GLM-5.3-Flash`	App. F.1
MATH / BIRD admission builders	`GLM-5.3`, `Qwen3.8-Max`, `DeepSeek-V4-Pro`, `Kimi-K3`, `MiniMax-M3`, `ERNIE-5.0-Thinking-Preview`	MATH / BIRD
Judge	proxy / official BIRD execution scorer; MATH answer comparator	see text

Table 6: Model inventory for the reported benchmark studies. The target is held frozen within each phase; Phase-II builders come from six independent labs.

C Same-Code Controls and Repeated-Execution Protocol

All MATH primary estimates use the same 386 fully observed tasks. The protocol below specifies the panel, inclusion rule, and repeat analysis.

Panel and inclusion. Programs, solver, judge, development split, and three repeat identities are fixed. A salted hash selects one low-call member per builder plus two at-large members and single-call; the control contains nine identical baseline slots. The September 26, 2026, 05:15 UTC snapshot combines generated continuation 10 and clone continuation 20. After outcome inspection, we retained tasks with a recorded binary score in every required single-run and repeat cell. These 386 task identities define the common reanalysis set.

Completeness and exclusions. The single-run matrix contains $36 \times 386 = 13{,}896$ observed cells and each repeat arm $9 \times 3 \times 386 = 10{,}422$, giving 100% completeness for all three matrices. Policy evaluation and oracle replay use the same task identities.

The 14 excluded problems contain 61 unscored generated cells, distributed as DeepSeek 5, ERNIE 4, GLM 9, Kimi 30, MiniMax 9, and Qwen 4; baseline and clone records are complete. Their pre-subsetting fraction, 3.50%, is below the frozen 10% gate. Excluded tasks are harder and longer, with lower baseline accuracy (Table 7). Unknown scores and all incurred acquisition costs remain in the source records.

Observed characteristic	Included (386)	Excluded (14)
Baseline repeat-mean accuracy (%)	95.85	71.43
Mean benchmark difficulty level	3.47	4.29
Level-5 tasks (%)	26.42	64.29
Mean question length (characters)	198.75	284.71

Table 7: **Observed characteristics of included and excluded tasks.** Excluded tasks have higher benchmark difficulty, longer questions, and lower baseline accuracy.

Provenance and conflicts. Manifests are checked on 13 fields covering execution conditions, programs, task/split hashes, panel draw, and schedule; snapshots, matrices, and task lists are hash-bound. Completed scores replace unknowns, with first-completed precedence on disagreement. Five generated cell identities conflict, including three on three retained tasks; clones have none. Collector and concurrency changes are logged.

Estimates and sensitivity. Recorded binary scores give plug-in headroom of 2.33 pp (generated) and 2.16 pp (clone), and $D = -0.26$ pp (95% interval $[-1.45, 0.95]$ pp; INSUFFICIENT EVIDENCE). Removing the three conflict-affected tasks from both arms and reselecting on the remaining 383 tasks gives $D = -0.61$ pp (interval $[-1.96, 0.64]$ pp), with the same decision.

C.1 EXPLORATORY REPEATABILITY AND SCORE SENSITIVITY

For non-reference members, define $A_r(h, x) = Y(h, x, r) - Y(h_0, x, r)$, where h_0 is single-call or clone-c1. Within each repeat, subtract row and column means and add the grand mean. Correlate each held-out residual matrix with the residual of the other two repeats' mean, then average the three Pearson correlations to obtain ρ_{res}. A paired task bootstrap uses $2{,}000$ shared draws. Correlations are 0.789 ($[0.75, 0.83]$) for generated members and 0.006 ($[-0.07, 0.08]$) for clones, with difference 0.783 ($[0.69, 0.87]$). This post-collection diagnostic measures repeatable score patterns after removing additive effects.

Losses in all three repeats span 570 member–task pairs on 100 tasks. Wins in all three comprise eight pairs on one task, shared by all eight generated programs. That task is format-sensitive: the gold answer is `12`, baseline outputs end with `#### 12th grade`, and a generated member returns `12th grade`. The extractor retains the former's complete `####` payload and scores it zero, but uses numeric fallback for the latter and scores it one. Recorded scores are retained; the repeated repair here arises from answer formatting.

D STATISTICAL ANALYSIS, CALIBRATION, AND SENSITIVITY

Empirical diagnostics use 386 complete MATH tasks. Synthetic calibration uses the sample sizes specified below.

D.1 HELD-OUT-REPEAT STATISTIC AND DECISION RULE

In each of three folds, two discovery repeats select a member for each task and a single best fixed member by mean discovery accuracy. The third repeat scores the task-wise mapping against that fixed member. Averaging task-level gains over folds gives G; the paired difference between arms

is $D = G_{\text{real}} - G_{\text{clone}}$ (§4.2). The repeat-averaged oracle plug-in $\widehat{H}$ is descriptive and does not determine the decision.

Discovery scores lie on $\{0, \frac{1}{2}, 1\}$, so ties are common. Selection uses independent, identically distributed uniform jitter $U(0, 10^{-9})$ to break member ties symmetrically, averaging ten frozen tie realizations for the empirical report. Each calibration dataset uses one tie realization. For the empirical randomization reference, member labels are permuted independently within each task and each repeat in each arm. Both selections, both gains, and D are recomputed for each of $10{,}000$ draws, with the $+1$ Monte Carlo correction. A paired task bootstrap uses $2{,}000$ draws with a fixed random seed and the same resampled task identities in both arms to obtain the 95% interval.

Positive support requires: at most 10% incomplete tasks in either arm or their intersection before subsetting; at least 30 paired tasks; randomization $p < 0.05$; positive bootstrap lower bounds for D and G_{real}; and $D > 1.0$ pp. Repeats must have matched conditions and verified independent execution provenance. The current decision is INSUFFICIENT EVIDENCE, rather than an equivalence finding.

Analysis specification. The acquisition protocol and initial stopping rule preceded collection; the v3 diagnostic, recovery, and completeness-selected reanalysis are post-collection amendments. Estimates use the September 26 snapshot; inclusion, exclusions, and conflict sensitivity are specified in Appendix C.

D.2 POPULATION TARGET AND LIMITS OF THE DIAGNOSTIC

For member success probabilities $p_h(x)$, stable complementarity is

$$H_{\text{stable}} = \mathbb{E}_x \max_h p_h(x) - \max_h \mathbb{E}_x p_h(x).$$

It is zero when some fixed member is expectation-best on almost every task. For a given selector, its population gain θ^* relative to the *population-best fixed* member satisfies $\theta^* \le H_{\text{stable}}$, because its success probability on each task is at most $\max_h p_h(x)$.

Empirical G uses a fixed member selected from finite discovery data. Clone subtraction provides a sampling reference, while D and its interval remain distinct from H_{stable}: a near-zero D does not bound the population target near zero.

The randomization reference is exact under equal-ability member-label exchangeability. For the broader global-dominance null, error control is empirical over the calibration grid.

Finite-repeat oracle estimates. For repeat means $\hat{p}_h(x)$, with $\mathbb{E}$ here taken over repeated-execution sampling at fixed x, Jensen's inequality gives $\mathbb{E} \max_h \hat{p}_h(x) \ge \max_h \mathbb{E}\hat{p}_h(x)$ on each task. Headroom subtracts a second maximum, so this inequality does not imply upward bias of the difference in general. In a known-truth equal-ability simulation with $p_h(x) = 0.9$, $M = 9$ members, $T = 400$ tasks, and $R = 3$, the mean plug-in is 8.69 pp over 300 datasets despite $H_{\text{stable}} = 0$. The empirical same-code arm likewise gives 2.16 pp. These examples show that positive plug-in headroom is insufficient evidence of specialization.

D.3 KNOWN-TRUTH CALIBRATION AND STATISTICAL POWER

Binary tensors are sampled from specified member–task probability matrices. Null settings include equal ability, shared task difficulty, global dominance, the $0.5/0.9$ dominance configuration, and heterogeneous dominance. Alternatives include crossovers over latent task types and difficulty-related partial crossover. The grid spans $M \in \{3, 9, 20\}$, $T \in \{100, 400, 1000\}$, $R \in \{3, 5, 10, 20\}$, complete data, 5% missing completely at random (MCAR), and 3% hardness-related missingness.

For every repeat count R, the calibrated statistic uses R folds: each fold learns both choices from the other $R - 1$ repeats and scores them on its single held-out repeat. The table's *plug* comparator declares a positive result when the lower endpoint of its percentile 95% task-bootstrap interval exceeds zero ($1{,}000$ draws). It fixes the task-wise maxima and sample-best member before resampling. The simulation variant, labeled v3*, uses the positive-support conditions above with 300 bootstrap draws and 300 randomizations, but checks the completeness gate only on the real arm. The empirical rule

additionally gates the clone arm and paired intersection. Simulation abstention denotes failure to obtain positive support.

In the two-specialist setting, the 400 tasks form two equal halves. Each of two specialists succeeds with probability 0.95 on its own half and 0.60 on the other; the remaining seven members have probability 0.60 throughout. Outcomes are independent Bernoulli draws. The oracle mean is 0.95 and the best fixed mean is 0.775, so $H_{\mathrm{stable}} = 17.5$ pp. All nine clone slots share the first specialist's probability profile, with fresh independent draws.

Table 8 reports $M = 9, T = 400$, using 300 Monte Carlo datasets per cell (150 in sweeps) and 300 randomizations per dataset. Across the 100-cell grid, the largest observed null false-support rate is 2%. A two-specialist alternative with $H_{\mathrm{stable}} = 17.5$ pp has power 0% at $R = 3$, approximately 0.7% at $R = 5$, and 100% at $R = 10, 20$; smaller cyclic crossovers remain largely undetected at $R = 20$. These setting-specific results show why three repeats leave stable complementarity unresolved.

	R=3		R=10		R=20		abst.
regime	plug	v3*	plug	v3*	plug	v3*	R=3
equal p=0.9†	100	0	100	0	100	1	100
equal + difficulty†	100	0	100	1	100	1	100
0.5/0.9 dominance†	100	0	100	0	13	0	100
dominance†	100	0	100	0	100	0	100
cross H≈0.9 pp	100	1	100	0	100	0	99
cross H≈1.8 pp	100	1	100	1	100	0	99
cross H≈4.2 pp	100	0	100	0	100	0	100
two strong specialists, H=17.5 pp	100	0	100	100	100	100	100

†Null (H=0): false-support %; otherwise power %. M=9, T=400, 300 datasets at R=3 (150 in sweeps). *plug*: positive 95% bootstrap lower bound. *v3 simulation variant: real-arm completeness gate, 300 bootstrap draws and 300 randomizations. The empirical rule additionally gates clones and their paired intersection; details are given above.

Table 8: **Three repeats can miss even a strong positive case.** At $M = 9, T = 400$, null rows report false-support % and alternative rows report power %; the final column gives $R = 3$ abstention. The simulated clone-calibrated rule (v3*) rarely declares positive support when stable complementarity is zero, but has little sensitivity to the tested small and moderate crossovers.

E Frozen Selector and Execution-Matched Replay

This appendix specifies how development data determine the frozen program choices and portfolio ranking, and how each is evaluated.

E.1 Frozen feature selector

The reported selector π_Z uses multinomial logistic regression with five task folds on the 100 development tasks. The score summary for member h on development task x is the available-repeat mean,

$$\bar{Y}_h(x) = \frac{1}{|\mathcal{R}_{hx}|} \sum_{r \in \mathcal{R}_{hx}} Y(h, x, r),$$

where $\mathcal{R}_{hx}$ contains its observed development repeats. A task is eligible for fitting only when every member has at least one observed repeat. Its training label is $y(x) = \arg\max_h \bar{Y}_h(x)$, with ties resolved by ascending member ID. Eligible tasks, ordered by task ID, are split into five consecutive folds without shuffling; all repeats of a task stay together.

The TF-IDF vocabulary and inverse-document frequencies are fitted once on eligible development questions, using English stop-word removal and at most 300 features. Each fold model appends the vector of member accuracies computed on its four training folds to every question's features. This vector is constant across questions within that model; only the text features vary by task. Logistic regression uses inverse regularization strength $C = 1$ and at most 1,000 iterations. A training fold with only one label is omitted; absent member classes receive zero probability. If $1 \le K \le 5$

models are fitted and $q_k(h \mid x)$ is model k's predicted class probability, inference uses $\bar{q}(h \mid x) = K^{-1} \sum_{k=1}^{K} q_k(h \mid x)$. Let $h_{(1)}$ and $h_{(2)}$ have the two highest averaged probabilities. The final frozen choice is

$$\pi_Z(x) = \begin{cases} \text{single-call}, & \bar{q}(h_{(1)} \mid x) - \bar{q}(h_{(2)} \mid x) < 0.15, \\ h_{(1)}, & \text{otherwise.} \end{cases}$$

The threshold is applied after averaging; a tied top probability therefore triggers the baseline fallback. The policy table evaluates these final choices, including the baseline fallback. It reports repeat-mean accuracy, with harness executions as the unit of evaluation; a selected harness can make multiple model calls.

Comparators are single-call, the development-selected fixed member, one fixed member drawn once with seed 20260915 (`gsm_kimi_s0_g3`), and the development-selected member with baseline fallback on missing cells. All policies are evaluated on the common 386 tasks, with three scores per selected member–task pair. Paired differences against dev-fixed use 2,000 task-bootstrap draws. The feature, dev-fixed, and fallback policies retain their recorded single-call choices.

Policy	Eval acc. (%)	S vs dev-fixed (pp) [95% CI]
single-call (fixed)	95.85	0.00 [0.00, 0.00]
dev-fixed	95.85	—
One randomly drawn fixed member	86.01	-9.84 [-13.04, -6.91]
Dev-fixed + baseline fallback	95.85	0.00 [0.00, 0.00]
π_Z (frozen selector)	95.85	0.00 [0.00, 0.00]

Table 9: **Frozen selection on MATH-500.** Accuracy is repeat-mean correctness on fresh executions; S is the paired gain over the development-selected fixed member, with 95% task-bootstrap intervals.

E.2 Selection gain and the scope of budget utility

For the implemented selector, $Z(x)$ consists only of question features and training-fold member accuracies. Its gain over the development-selected fixed member is

$$S(\pi_Z) = \frac{1}{n} \sum_{i=1}^{n} \left[\bar{Y}_{\pi_Z(x_i)}(x_i) - \bar{Y}_{h_{\text{dev-fixed}}}(x_i) \right],$$

where $\bar{Y}_h(x)$ now averages the three evaluation repeats, n is the number of evaluation tasks, and $h_{\text{dev-fixed}}$ is the fixed member selected using development data. We use S for policy gain and reserve D for the real-minus-clone repeat diagnostic. Mechanism labels are not inputs to this feature selector.

Replay budget b counts *harness executions*, each potentially containing multiple model calls; coverage uses a post-execution oracle.

E.3 Execution-matched replay and collection accounting

Each arm contains nine members/slots with three repeats. Replay uses the common 386 tasks and frozen development ranking (§5.4), taking all three repeats of the first $b/3$ members for $b \in \{3, 9, 15, 27\}$. With f incorrect clone executions among 27, expected coverage of a uniform size-b subset is $1 - \binom{f}{b} / \binom{27}{b}$, where $\binom{f}{b} = 0$ for $b > f$. Table 3 compares this exact expectation with generated-panel coverage at matched execution counts.

Acquisition ledgers retain all attempts and known usage, including excluded tasks and recovery; outcome tables use the common complete set. Calls and tokens follow each program and acquisition record.

F BIRD Execution Traces and Supporting Analyses

Phase I studies BIRD discovery; Phase II studies admission. This appendix details the discovery programs and their execution paths.

F.1 Discovery populations and conditions

Population and judging. Discovery evaluates twelve TTHE-proposer candidates from GLM-5.3-Flash, plus single-call and `react`, on 60 tasks (§5.5). Admission checks import and smoke execution. The baseline flips on three of 60 tasks between discovery runs. The set-of-rows proxy judge agrees with the official BIRD scorer on 0.97–0.99 of outcomes in two matrices with retained SQL, with no proxy-correct/official-incorrect cases.

Development exposure. Hand-written controls were designed after trace inspection, and $24/151$ evaluation items overlap protocol development. Their headroom is descriptive. Equal generation budgets cannot be reconstructed from the early records, which lack generation-tool and per-attempt logs.

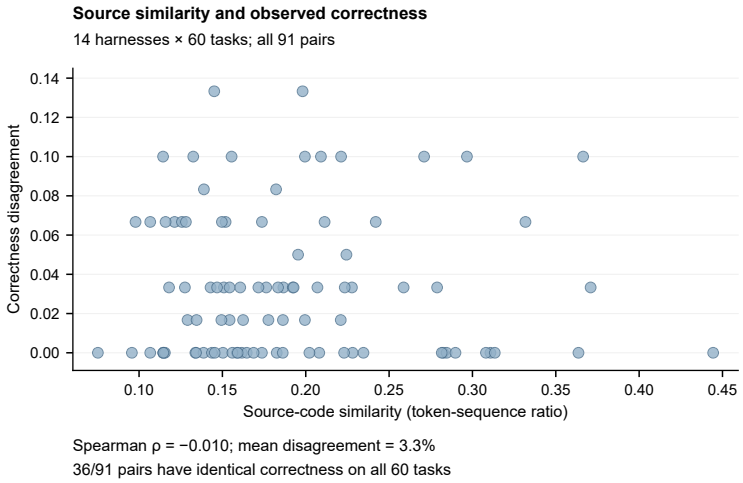


Figure 2: **Source differences need not change observed correctness.** BIRD discovery: 14 harnesses, 60 tasks, one execution each. Across 91 pairs including baselines, token-sequence source similarity has $\rho = -0.010$ rank association with correctness disagreement; $36/91$ pairs have identical correctness vectors. This is an outcome comparison on the observed tasks, not a test of program equivalence.

F.2 Discovery measurements and hand-written controls

All candidates pass the code-difference check, but $21/66$ generated pairs share correctness vectors ($36/91$ with baselines). Source similarity and correctness disagreement have $\rho = -0.010$ across 91 pairs (Figure 2). On these 60 tasks, mean disagreement is 3.3% overall and 3.8% among generated candidates, union repair is 7.7%, and single-run headroom is 3.3 pp. Traces explain how distinct programs produce these closely aligned outcomes.

F.3 Traces distinguish absence, non-activation, and failure

T1: described but absent. Six of twelve candidates claim retrieval or self-checking but make one model call without implementing either step. **T2: present but untriggered.** On all ten audited `react` tasks, the first SQL executes without error, leaving the repair branch idle and final SQL identical to single-call; execution success can coexist with semantic error. **T3: executed but broken.** One of twelve candidates executes multi-turn repair but returns prose as SQL, losing baseline-correct tasks.

The three cases separate implementation, branch activation, and output validity. Four hand-written controls with distinct paths yield 6.67 pp headroom on 60 tasks and 7.28 pp on 151 items. These post-inspection controls lack matched repeats and clones; their $24/151$ development overlap is documented in Appendix F.1. Worked traces follow in Appendix F.4.

F.4 WORKED EXAMPLES FROM THE TRACE AUDIT

The six-of-twelve T1 count covers the full discovery population. The separate trace audit follows five harnesses—the single-call baseline, `react`, a generated single-call candidate, a timeout-retry candidate, and a multi-turn repair candidate—on ten tasks (5×10). Below, one task illustrates these paths. Logged SQL excerpts are capped at 120 characters; truncation is marked *[trunc.]*.

The task. BIRD dev-set question 352 (`card_games` database; question text verbatim, the typo is the dataset's):

```
Calculate the percentage of the cards availabe in Chinese
Simplified.
```

Its BIRD hint:

```
'Chinese Simplified' is the language; percentage =
Divide(Sum(id where language = 'Chinese Simplified'),
Count(id)) *100
```

Gold SQL:

```
SELECT CAST(SUM(CASE WHEN T2.language = 'Chinese Simplified'
THEN 1 ELSE 0 END) AS REAL) * 100 / COUNT(T1.id) FROM cards
AS T1 INNER JOIN foreign_data AS T2 ON T1.uuid = T2.uuid
```

The single-call harness answers:

```
SELECT SUM(CASE WHEN language = 'Chinese Simplified'
THEN 1 ELSE 0 END) * 100.0 / COUNT(id) AS percentage FROM
foreign_data;
```

On the released database, the gold query yields 8.7734 and the baseline query yields 8.7728. The official scorer marks the baseline query wrong: it counts directly over `foreign_data`, whereas the gold query counts after joining to `cards`; the hint does not spell out that join.

`react` (T2: untriggered repair). The first prompt is byte-identical to single-call's. SQL executes successfully, so the database-error retry does not fire. All 10 audited tasks follow this path, with SQL execution but 10/10 final queries byte-identical to single-call's.

Single-call candidate (T1: absent repair). The docstring's *conservative repair* specifies an output-token cap and answer-only block. Execution consists of one model call followed by SQL extraction. The cap is an environment default (2,048 versus 32,000 at runtime), has no harness control-flow implementation, and never binds on these short answers. The worked task returns the same wrong `foreign_data` query, modulo whitespace, with one call and zero SQL executions. Across all 10 tasks, SQL executions remain zero and 6 final queries are byte-identical to single-call's. The remaining differences reflect a reworded prompt, without retrieval, self-check, or repair.

Timeout-retry candidate (untriggered watchdog). A daemon-thread watchdog wraps the coder call and issues one hedged duplicate after a 40 s stall. No stall occurs in the audit: SQL executions are zero and final SQL equals single-call's on all 10 tasks.

Multi-turn repair candidate (T3: a broken multi-turn path). The one audited candidate implementing genuine execute-inspect-repair returns the model's *prose* as final SQL. On a task single-call answered correctly ("State the alternative languages available for card named Annul numbered 29."), its logged final SQL begins, verbatim:

```
with "only a single read-only SELECT is allowed".  The
previous "query" was actually just prose text (the model
wrote rea[trunc.])
```

Under this repair re-prompt, the output contract fails again: asked to fix a query that "was actually just prose", the frozen target responds with more prose reasoning, no SQL block is found, and the reasoning text is submitted as the query. The item flips from baseline-correct to harness-wrong.

G BIRD ADMISSION-POLICY STUDY

The 2×2 study uses 18 builder–seed cells per arm (six builders and three seeds); II-E is development-only. Each cell has eight generation slots. Each slot allows three generation attempts and stops at first admission without top-up. Two development databases are excluded. Evaluation uses $1{,}169$ questions from nine databases and a pre-drawn 400-item core, with one disclosed 18-item smoke-test exposure. The official BIRD judge uses a $30\,\mathrm{s}$ timeout. Model-specific response caching uses cross-process locks after correction of a concurrency defect.

The primary D$-$A contrast (forced/gated minus free/ungated headroom) pairs builder–seed cells with hierarchical bootstrap and permutation inference; core analyses use Holm-corrected factorial contrasts and K-matched sensitivity. Instrument fixes preceded confirmatory outcomes. Subsequent analysis corrections share baseline records and database/task resampling. Canonical records use first-write-wins deduplication; conflicts and operational deviations remain in the source audit. Generation costs vary across arms.

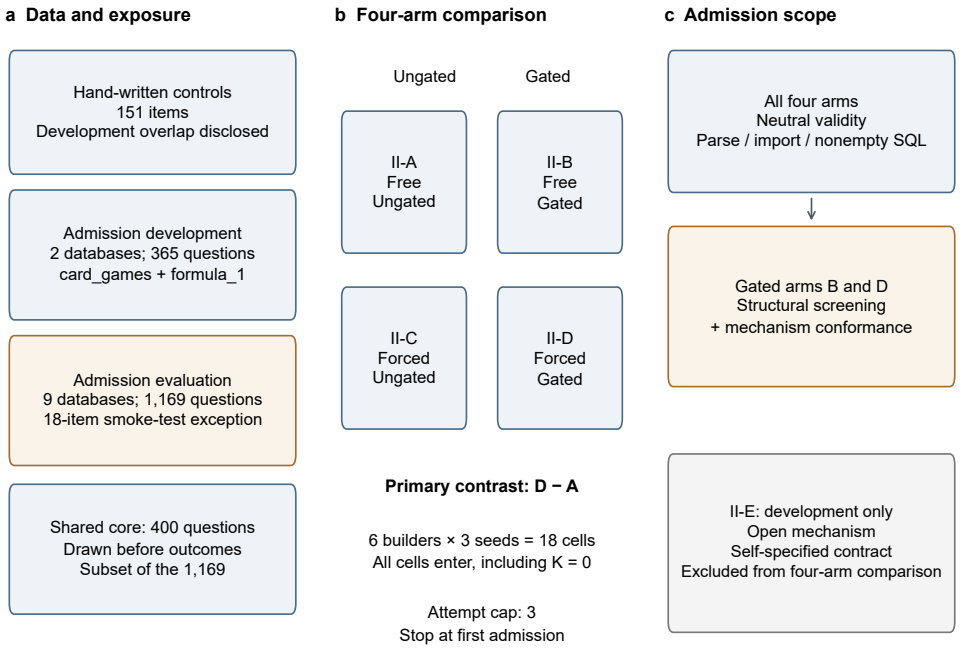


Figure 3: **BIRD admission design.** Left: data splits and disclosed development exposure. Middle: the 2×2 four-arm comparison with a common attempt cap. Right: neutral validity and the implemented structural-screening/conformance bundle in gated arms. The additional open-mechanism arm II-E is development-only.

G.1 DESIGN AND PRIMARY CONTRASTS

The four arms cross strategy forcing and gating: II-A (free, ungated), II-B (free, gated), II-C (forced, ungated), and II-D (forced, gated). Forced strategies come from a fixed set; free builders select a declaration within the vocabulary in Appendix G.4. Results below use the corrected reanalysis of the pre-outcome specification.

Metric	II-A	II-D
Candidate count	5.78	3.83
Oracle accuracy (%)	76.14	75.70
Best-fixed accuracy (%)	67.58	67.30
Headroom (pp)	8.56	8.40
D−A headroom (pp)	−0.16	
Unadjusted bootstrap 95% CI	[−0.83, +0.87]	
One-sided sign-flip p	0.647	

Table 10: **BIRD admission: full-set headroom.** Means over builder–seed cells on 1,169 questions; single-call is included in both the oracle and best-fixed comparators.

The forced/gated minus free/ungated contrast is −0.16 pp in baseline-inclusive oracle headroom, with cell contrasts −4.36 to + 2.74 pp. The implemented bundle does not improve observed mean headroom; Table 10 reports uncertainty.

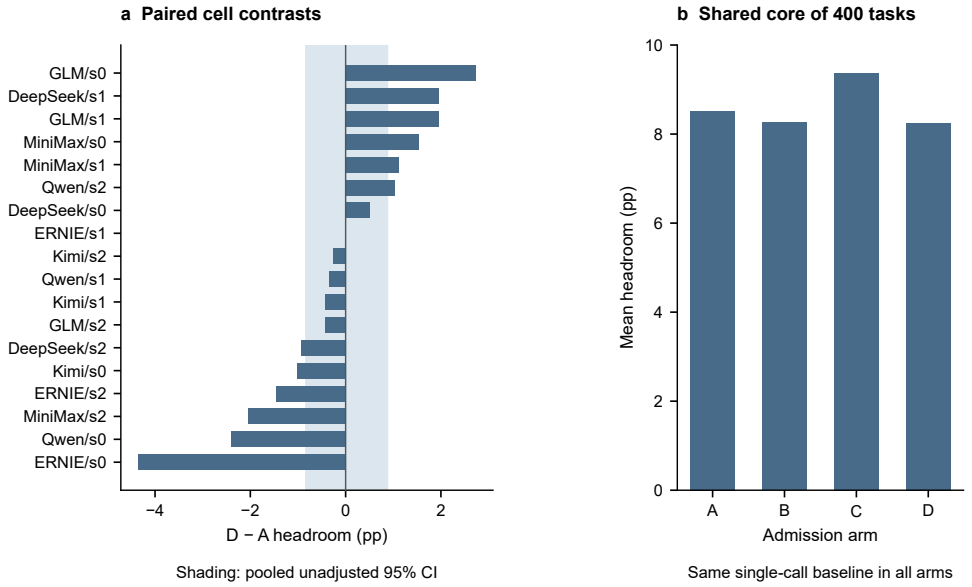


Figure 4: **BIRD admission contrasts.** Left: full-set D−A cell contrasts with the pooled 95% interval. Right: four-arm headroom on the shared core-400 tasks with a common single-call baseline. Headroom and contrast axes show percentage points.

On core-400, the gate contrast is −0.69 pp, with inconclusive strategy and interaction effects (Table 11). The candidate-count-matched contrast is −0.31 pp. II-E remains development-only. The R2/R3 audits (Appendix G.3) place these single-run coverage measurements alongside 8.54% R2 flips, 14.5% of baseline tasks changing across three runs, and a 6.50 pp same-code oracle gap.

Contrast	Effect (pp)	95% CI (pp)	Raw p	Holm p
Gate	−0.69	[−1.56, +0.20]	0.026	0.079
Strategy	+0.42	[−0.32, +1.51]	0.263	0.428
Interaction	−0.87	[−2.83, +0.65]	0.214	0.428

Table 11: **Corrected core-400 factorial reanalysis.** Here A–D denote arm mean headrooms. Gate is $[(B-A)+(D-C)]/2$, strategy $[(C-A)+(D-B)]/2$, interaction $(D-C)-(B-A)$; raw and Holm-adjusted p-values.

G.2 SAME-CORE HEADROOM DECOMPOSITION

Table 12 separates coverage, best-fixed accuracy, and headroom using the same core tasks and baseline records in every arm.

Arm	K_{cand}	K_{total}	Single-call (%)	Best (%)	Oracle (%)	H (pp)
A	5.78	6.78	63.25	66.00	74.51	8.51
B	6.11	7.11	63.25	66.12	74.39	8.26
C	5.61	6.61	63.25	65.64	75.01	9.38
D	3.83	4.83	63.25	65.65	73.90	8.25

Table 12: **Same-core headroom decomposition.** All arms use identical 400 tasks and identical single-call outcomes. K_{cand} counts admitted candidates, excluding single-call, and $K_{\text{total}} = K_{\text{cand}} + 1$; counts and accuracies are cell means. Headroom is oracle minus best-fixed accuracy, not realized routing gain.

G.3 SENSITIVITY, REPEAT AUDITS, AND RECORDED COST

K-matched and R2 replays. Replays use deterministic record ordering, shared database/within-database task draws, and seed resampling within fixed builders (10,000 replicates). K-matching uses the smaller candidate count in each A/B and C/D pair, exactly enumerates uniform subsets, retains single-call, and reselects best-fixed within each subset and task draw. It measures population-size sensitivity. R2, described below, retains all 18 cells, including two singleton A subsets; its D−A changes from +0.42 to +0.81 pp (change +0.39 pp, 95% CI [-0.95, +1.63]).

R2 covers 70 harnesses on 400 tasks, taking the first two callable admitted A/D candidates where available. Two cache-off acquisition groups remain separate despite their duplicated repeat label, with program identity checked against cached runs. Without matched single-call acquisitions, both R2 conditions exclude that baseline; singleton headroom is zero across the 18 retained pairs. Recorded flips are 2391/28,000 against cached outcomes (8.54%) and 1245/14,400 between D cache-off passes (8.65%). These describe observed instability: acquisition time and caching are confounded, and the rates are not standard errors or effect-size floors.

R3 single-call-only repeat audit. Three independent cache-off runs evaluate only the single-call baseline on core-400 with GLM-5.3-Flash at temperature 0. Correctness changes on 58/400 tasks (14.5%); pairwise flips are 39/400, 40/400, and 37/400 (9.75%, 10.0%, 9.25%). Treating runs as three pseudo-members, one execution each, gives official-scorer oracle coverage 72.25% and best-repeat accuracy 65.75%: a 6.50 pp same-code gap. This records baseline execution variability at temperature 0 and motivates the repeat controls used in the main study.

Sensitivity estimand	Estimate (pp)	95% CI (pp)
K-matched B−A headroom	-0.55	$[-1.53, +0.63]$
K-matched D−C headroom	-0.08	$[-1.26, +0.80]$
K-matched gate mean	-0.31	$[-1.07, +0.40]$
R2 subset D−A (cached)	$+0.42$	$[-0.61, +1.48]$
R2 subset D−A (cache-off)	$+0.81$	$[-0.35, +1.84]$
R2 change in subset D−A	$+0.39$	$[-0.95, +1.63]$

Table 13: **BIRD sensitivity on core-400.** CIs are unadjusted. K-matched comparisons always retain single-call; R2 comparisons exclude single-call and are conditional on the observed acquisition groups.

Recorded cost. The core records contain 259,304 logical solver calls across 153,600 candidate–task records (Table 14). These include cache-served requests and count logical rather than billed requests; token usage is unavailable. Generation attempts, admissions, and execution calls are reported separately. Population cost averages weight builder–seed cells equally, exclude single-call, and assign zero calls to zero-candidate populations.

Arm	Candidate-task records	Logged solver calls	Calls/task/population
A	41,600	67,258	9.34
B	44,000	64,399	8.94
C	40,400	75,604	10.50
D	27,600	52,043	7.23

Table 14: **BIRD generation and execution cost on core-400.** Logged logical calls are not billable requests or token counts. The last column averages candidate-population calls per task over cells.

G.4 Admission scope and implementation

Free-arm builders choose among repair, vote, twostage, and plain. Vote requires multiple samples and selection, including but not limited to three-way majority voting. Twostage consumes an artifact without a prescribed response interface. Plain declares no control-flow mechanism despite the prompt requesting a change. Thus strategy choice operates within this four-declaration vocabulary.

The evaluated gate combines structural screening with conformance verification; valid prompt-only transformations can fail the structural screen. Development checks found false rejections, request-attribution and response-interface defects, and unresolved judgments. Later revisions produced no replacement benchmark outcomes, so results describe the implemented bundle.

Generation interface. Format-guard is one of eight fixed strategies, each assigned one slot per builder–seed cell in both forced arms (C and D). It requests schema fidelity and a final SQL code block with no extra text. All 108 format-guard attempts across its 36 slots fail neutral validity: 71 logs report unterminated strings, 36 no Python fence, and one a provider timeout. Synthetic examples reproduce truncation in the archived non-greedy fence extractor. Full builder messages and exact historical extractor versions are unavailable, limiting attribution of these upstream validity failures.