VeRA: RENEWING REASONING BENCHMARKS WITH EXECUTABLE SPECIFICATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning benchmarks need fresh questions that preserve the tested computation and harder questions as models improve. We introduce VeRA, which compiles an existing problem into a reusable question template, input generator, and deterministic answer program. VeRA-E varies inputs and wording within a computational family; VeRA-H modifies the computation; VeRA-H Pro selects difficult candidates. Program checks and independent item audits validate the resulting releases. Across 16 models, VeRA-E leaves mean accuracy nearly unchanged on GSM8K (94.85% to 95.20%) and Beyond-AIME (58.34% to 57.30%), but reduces AIME-2024 accuracy from 84.46% to 70.25%. This selective drop makes computation-preserving renewal a diagnostic for benchmark-specific overfitting and possible data contamination. For modified tasks, a judge selects from up to five validated proposals per original problem. On AIME-2024-II, the selected audited release lowers accuracy from 84.91% to 58.57%. Within this budget, selection yields harder releases than the broader pools on all three sources; on AMO-Bench, the selected release's mean accuracy remains above that on the originals. Targeted repair raises usable construction yield from 75.4% to 95.1%. We release executable specifications, audited items, and validation records for repeated benchmark renewal.

1 INTRODUCTION

Reasoning benchmarks compare models on problems with known answers. As those problems become familiar and scores rise, evaluation needs fresh questions and enough remaining difficulty to distinguish improving models (Shi et al., 2023; Golchin & Surdeanu, 2024; Jain et al., 2025; White et al., 2025). New numerical inputs test whether a solver can apply the same computation beyond a familiar item. Modifying the computation can provide a harder test. Both require reliable reference answers, making repeated question construction and verification an ongoing cost.

VeRA represents each original problem, or seed, as an *executable task family*. A question template leaves slots for variable quantities; a generator fills those slots with valid inputs; an answer program computes the corresponding result. Once constructed, the family can produce further labeled questions locally. Execution checks test the programs, while experts independently solve questions in the audited releases. These checks connect the generated wording to the computation and its answer.

We use the family to renew a benchmark in three ways. VeRA-E varies inputs and wording while preserving the computation (E: *vary*). VeRA-H changes constraints or solution structure to seek harder problems (H: *modify*). VeRA-H Pro uses a model judge to select from up to five validated proposals per seed (H Pro: *select*). We evaluate E for robustness to fresh instances and H/H Pro for difficulty. More difficult releases provide *headroom*, allowing an evaluation to distinguish improving models after the original questions become easy.

The fresh-instance results distinguish task difficulty from sensitivity to familiar questions. Widely used GSM8K remains near 95%, while the newer, expert-curated Beyond-AIME benchmark remains near 58%. Beyond-AIME was designed to reduce memorization and to match or exceed the hardest AIME questions (ByteDance Seed et al., 2025). On the widely used AIME-2024 competition benchmark, accuracy instead falls from 84.46% to 70.25%. Stability on both elementary and demanding tasks supports difficulty-preserving renewal; the selective AIME drop flags a gap between performance on familiar questions and application of the same computations to fresh inputs.

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.

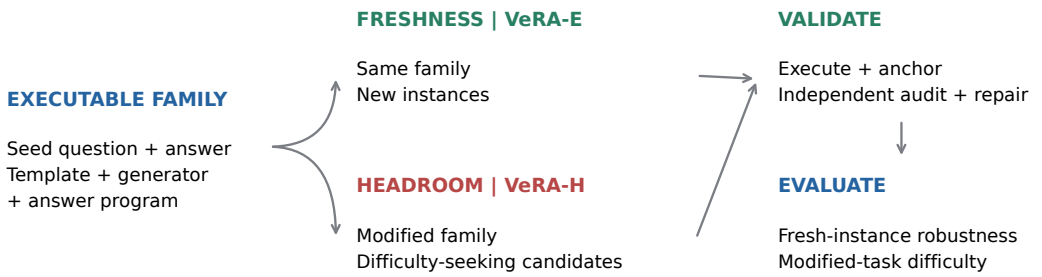


Figure 1: **Two axes of executable benchmark renewal.** A template, generator, and answer program support family-preserving variation (freshness) and family modification (headroom). Execution checks, mode-specific answer checks, and independent item audits precede solver evaluation on both branches.

Task modification seeks additional difficulty. AIME-2024-II accuracy falls from 84.91% on seeds to 68.71% on audited H variants and 58.57% on H Pro. Selection also lowers accuracy relative to H on Beyond-AIME and AMO-Bench within the five-candidate budget. These comparisons evaluate the challenge delivered by modification and selection, separately from the robustness tested by E.

We contribute executable task families, a validated construction pipeline, and evaluations of fresh-instance robustness and release difficulty. The specifications generate new labeled questions locally; the audited banks provide ready-to-use evaluation items.

2 Benchmarks as Executable Families

2.1 The Specification Interface

For a seed question–answer pair (q_i, a_i), a language model, called the teacher, proposes a specification $\mathcal{S}_i = (T_i, G_i, V_i)$. The template T_i turns values assigned to its slots, denoted θ, into a natural-language question. The generator G_i samples those values from a random seed. The answer program, called the verifier V_i, checks the assignment and returns a validity flag and canonical answer. Sampling follows

$$\theta \leftarrow G_i(r), \qquad x = T_i(\theta), \qquad (b, y) = V_i(\theta), \tag{1}$$

and accepts the instance when b is true. The specification also stores slot types, constraints, and an optional rationale for task modifications. A hash of the seed identifier, generator identifier, and sampling index fixes r, making each draw reproducible.

A generator can sample a new numerical range while retaining its answer program; a modified task can change both. An audit checks that the question generated from the template asks for the computation implemented by the program.

In Figure 2, changing quantities and names preserves a subtraction problem. Adding a purchase changes the calculation and requires an updated answer program; student performance determines whether that modification makes the task harder.

2.2 Family Variation and Task Modification

VeRA-E: variation within a computational family. The teacher parameterizes the seed's computational structure. Variants change numerical values, entity names, narrative form, or language. The original assignment θ_i^{seed} anchors the specification through $V_i(\theta_i^{\text{seed}}) = a_i$. Comparing seed answers with fresh draws tests whether the solver applies the same relationship across new values, cases, and renderings.

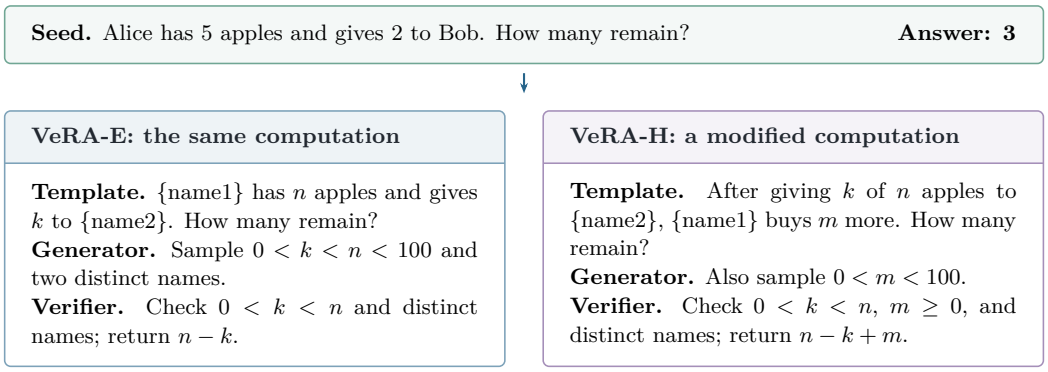


Figure 2: **One seed, two kinds of executable variation.** E changes quantities and names while preserving subtraction; H adds a purchase and updates the answer program. Student evaluation measures the difficulty of each modified computation.

VeRA-H: construction of modified tasks. The teacher introduces new slots, constraints, dependencies, or solution structure to seek more difficult problems. Each modified specification has its own generator and answer program. We evaluate the accepted H pool to measure what this proposal process delivers before difficulty-based selection.

VeRA-H Pro: selection within a fixed candidate pool. For each seed, the pipeline produces up to five candidates, validates them, and uses a fixed difficulty judge to select the highest-ranked accepted candidate. Experts audit the selected item before release. Comparing H Pro with H tests the delivered difficulty under this selection rule; we report the accepted item counts for both sets.

2.3 How a Specification Changes the Task

The H proposer uses four types of modification. *Constraint tightening* changes admissible assignments, for example by requiring positive or coprime parameters. *Composition* introduces intermediate quantities or dependencies, so the answer requires combining subcomputations. *Parameter-regime search* changes where the generator samples within a mathematical family. *Structural generalization* replaces a special condition with a parameterized one, such as a fixed perpendicularity condition with an angle parameter.

Each candidate records which parts of the specification change. Tightening can preserve the answer program; composition can require new slots and code; structural generalization changes both the question and computation. Figure 2 illustrates composition through an added purchase. Appendix H gives detailed construction workflows, and Section 5.2 measures the challenge the modifications deliver.

3 Validation and Synthesis

3.1 What Each Validation Layer Establishes

Validation checks that the generated program runs and that its answer matches the generated question. A successful execution establishes the first requirement; independent solving checks the second. Table 1 shows how the validation layers connect inputs, code, question wording, and answers before student evaluation.

Common gates check schema compliance, compilation, sampled execution, and nondegenerate valid yield. For VeRA-E, seed anchoring tests the known input. For modified tasks, noise discrimination presents two trials using the verifier answer and three using controlled answer perturbations. The judge must classify at least four correctly. On 50 calibration items, this threshold gives one false acceptance and three false rejections; requiring all five gives zero and fourteen. The chosen threshold

Table 1: **Validation layers and their evidence.** Each layer addresses a different relationship between the program, its inputs, and the rendered problem.

Layer	Check	Evidence supplied
Schema and execution	Parse fields, compile code, run sampled assignments	The specification can execute through the declared interface
Generator coherence	Check valid yield and constraints	Sampled inputs meet the implemented requirements
Seed anchoring (E)	Match the known seed answer	Agreement at the original input
Noise discrimination (H)	Judge correct and perturbed answers	Sensitivity to sampled answer errors
Expert item audit	Independently solve rendered problems	Semantic and label agreement on audited items
Student evaluation	Measure solution accuracy	Empirical robustness or difficulty of the retained set

retains more usable candidates before the independent audit. Appendix I details this precision–yield tradeoff.

Semantic audit. Six STEM PhD auditors independently solved rendered variants without access to the generated verifier code, then compared their answers with the verifier output and checked clarity and renderability. For VeRA-E, the audit additionally checked the intended relationship with the source task. Disagreements were referred to a senior annotator, with unresolved cases rejected. This audit tests the generated question itself, complementing execution checks on its program.

The specification makes audit feedback actionable. A disagreement can be localized to the wording, answer program, or parameter domain, rather than treated as an unexplained wrong label. The repaired candidate then passes through validation again, retaining the link between the item and the computation that generated it.

Each audited release records the inputs, rendered questions, and independently checked answers. A later program error or ambiguous rendering can be reproduced from the specification and repaired at its source. Further draws use the compiled specification and automatic checks; subsequent audited releases add their own independent solution records.

3.2 The Propose–Execute–Validate Loop

The teacher proposes a specification, receives structured feedback on failed checks, and revises it within the synthesis budget. Feedback identifies violated constraints, runtime errors, or mismatched seed answers. Generated functions execute in child processes with wall-clock timeouts and operating-system-supported CPU and memory limits. Accepted specifications retain their identifiers and generated instances.

Compilation amortizes task-family construction. Teacher inference, validation, and repair establish the specification once; subsequent sampling executes its generator and verifier locally. Program runtime and valid yield determine the marginal cost of further evaluation draws.

4 Experimental Design

We evaluate 16 models on GSM8K (Cobbe et al., 2021), AIME 2024 and 2025, Beyond-AIME (ByteDance Seed et al., 2025), and AMO-Bench (An et al., 2025). The teacher constructs problems, the judge evaluates proposals, and the models being tested are called students. The default teacher and judge are GPT-5 with high reasoning effort. Additional VeRA-E constructions use Gemini 3 Pro and Claude Opus 4.6 as teachers. Students receive zero-shot reasoning prompts with boxed-answer extraction (Wei et al., 2022).

Each evaluated problem has five independently sampled student responses. Avg@5 averages their correctness over problems and then equally over the fixed 16-model panel. Appendix A reports individual model scores.

Table 2: **Family-preserving variation.** Avg@5 accuracy (%) averaged over the fixed 16-model panel. The 220 AIME and Beyond-AIME variants receive direct expert audit; GSM8K uses the automatic validation pipeline.

Benchmark	Seeds	Variants	Seed score	VeRA-E score	Change (pp)
GSM8K	1,319	2,638	94.85	95.20	+0.35
AIME 2024	30	60	84.46	70.25	-14.21
AIME 2025	30	60	79.21	72.13	-7.08
Beyond-AIME	100	100	58.34	57.30	-1.04

More precisely, for correctness indicators Y_{mjr} on model m, item j, and response draw r, the model score on a declared item set $\mathcal{D}$ is

$$S_m(\mathcal{D}) = \frac{100}{5|\mathcal{D}|} \sum_{j \in \mathcal{D}} \sum_{r=1}^{5} Y_{mjr}. \tag{2}$$

The panel score gives each of the 16 models equal weight, with equal weight per item within each model. We measure *release-level difficulty*: accuracy on the accepted H pool or the selected H Pro items under the stated candidate budget.

Items used for scoring. For VeRA-E, GSM8K contains 1,319 seeds and 2,638 variants; each AIME year contains 30 seeds and 60 variants; Beyond-AIME contains 100 seeds and 100 variants. The human audit covers the 220 AIME and Beyond-AIME variants and accepts all 220. GSM8K is evaluated with the automatic specification checks. A separate GPQA-Diamond extension uses model arbitration; its distinct labeling procedure is reported in Appendix F.

The main difficulty results score the independently audited items, including accepted repairs; this set is named *Verified*. Its H/H Pro counts are 69/14 for AIME-2024-II, 466/93 for Beyond-AIME, and 217/44 for AMO-Bench, compared with seed sets of 14, 100, and 50 problems. The expanded release, *Verified Full*, retains this scoring set and adds accepted replacements at rejected positions. Table 4 reports both releases' construction counts, including historical AIME. Appendix E compares the automatically accepted pool with the audited scoring set.

5 Results

5.1 Equivalent Computations Expose a Selective AIME Drop

An almost unchanged average can hide a model-specific failure. GSM8K appears stable at the panel level: its mean changes by only +0.35 points. The models are less uniform. Kimi K2 Thinking falls from 95.0% to 86.6%, while most other models improve slightly, and the reported across-model standard deviation rises from approximately 0.6 to 2.3 points. Figure 3 shows why fresh-instance evaluation is useful even without a large aggregate drop: it identifies which solvers preserve their performance when the concrete questions change.

AIME shows a larger change for the earlier benchmark year. The suite mean decreases by 14.21 points for AIME-2024 and 7.08 points for AIME-2025. The same direction appears under the two alternative teachers: reported drops are 12.8 versus 6.9 points with Gemini 3 Pro, and 15.1 versus 6.3 with Claude Opus 4.6. Teacher variation therefore preserves the observed ordering of the year-level changes.

The earlier year starts with the higher seed score but loses more accuracy on fresh instances. Could this reflect larger changes to its inputs or wording? Figure 4 compares edit magnitude for the two years.

A similar edit budget exposes different solver sensitivity. The perturbation distributions overlap, with means of 0.28 and 0.30 (Figure 4), despite the different accuracy changes. This measure does not capture every change in computational difficulty: a small parameter change can activate a new

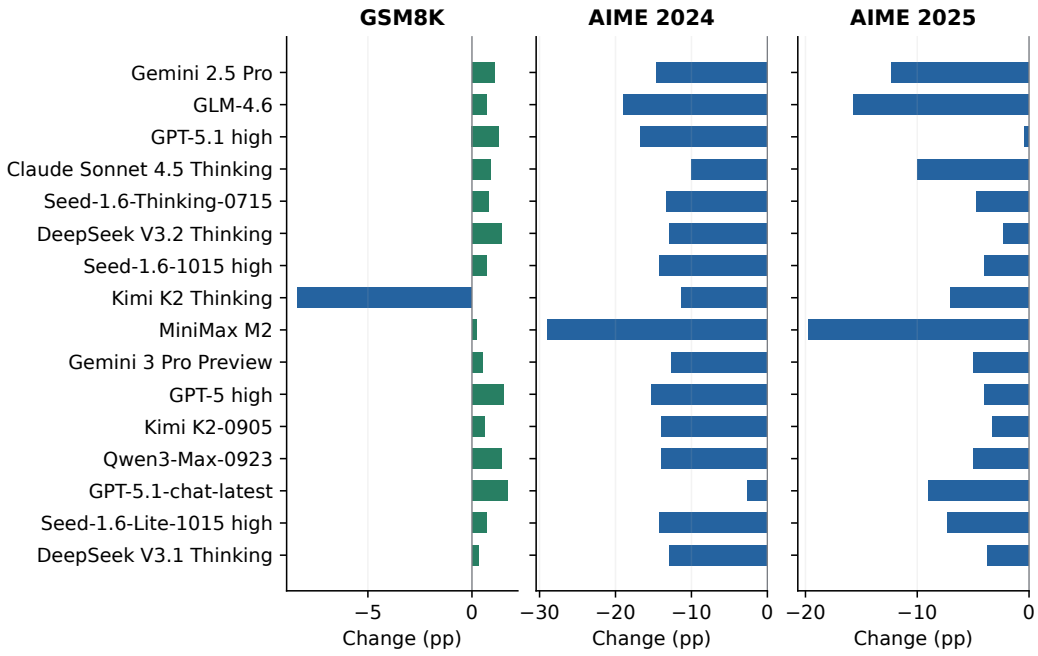


Figure 3: **Robustness is model- and benchmark-dependent.** Changes from seed accuracy to VeRA-E accuracy for the 16-model panel. GSM8K's stable suite mean coexists with a substantial drop for Kimi K2 Thinking. Both AIME years show broadly negative changes of different sizes. Appendix A gives the underlying scores.

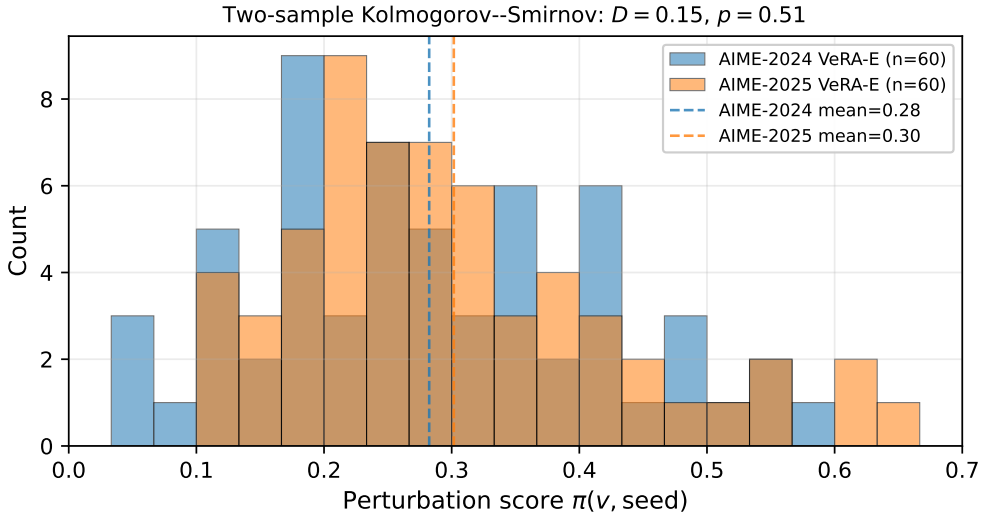


Figure 4: **Measured changes to the two AIME cohorts are similar.** Perturbation combines numerical-slot and textual edit distances. The 60 variants per year have mean scores of 0.28 and 0.30. These distributions describe edit magnitude; solver accuracy measures computational difficulty.

case or numerical regime. Appendix C defines the numerical and textual distances and their family-grouped interpretation.

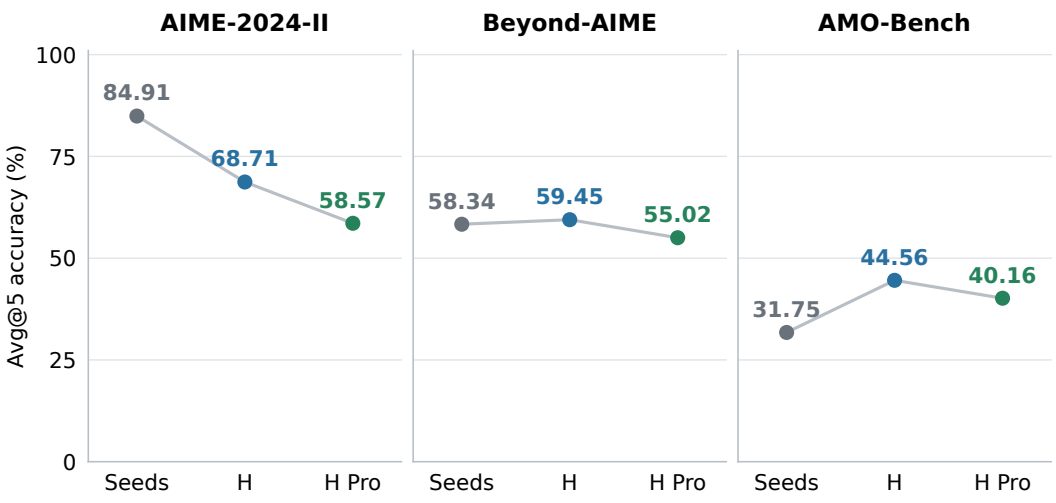


Figure 5: **Candidate selection changes release-level difficulty.** Avg@5 over the fixed 16-model panel on the Verified scoring release. H Pro is harder than H on every source; AMO-Bench seeds remain harder than either release. Construction uses up to five candidates per seed and a fixed judge (Section 4).

Stable scores on both easier and harder tasks anchor the comparison. Beyond-AIME changes from 58.34% to 57.30%, whereas GSM8K stays near 95%. The same E procedure thus preserves aggregate difficulty on a demanding, newer benchmark as well as an established elementary one. These observations complement the structural checks and expert audit of computational equivalence: they show that fresh-instance generation does not produce a uniform accuracy penalty. Against this background, the much larger AIME drops identify benchmark-specific sensitivity to replacing familiar instances.

5.2 Selected Releases Are Harder Than the Modified Pools

AIME recovers substantial evaluation headroom. On AIME-2024-II, mean accuracy decreases from 84.91% on seeds to 68.71% on H and 58.57% on H Pro (Figure 5). The 26.34-point seed-to-Pro difference shows that the construction can produce fresh, audited tasks that challenge models already strong on the source problems. Selecting a candidate judged difficult yields a lower score than evaluating the broader H pool.

Selection helps even when modification alone does not. On Beyond-AIME, the H pool is close to the seeds, at 59.45% versus 58.34%, but H Pro reaches 55.02%, 4.43 points below H. Ranking the validated proposals produces a more demanding release from up to five candidates per seed (Section 4).

Selection lowers mean accuracy on AMO-Bench within the fixed candidate budget. Accuracy is 31.75% on seeds, 44.56% on H, and 40.16% on H Pro. Selection lowers accuracy by 4.40 points relative to H, but the result remains 8.41 points above the seed mean. These means describe the challenge delivered by the proposal-and-selection process. H Pro is harder than H on every source, while AIME-2024-II and Beyond-AIME also gain difficulty beyond their seeds.

H Pro versus H measures the benefit of selection within the generated pool; either release versus seeds measures difficulty gained over the source. Selection helps across all three sources. Exceeding seed difficulty also depends on the proposals: the AMO-Bench pool improves through selection, while its original problems remain the harder test.

The qualitative pattern survives item auditing. Auditing changes which items enter evaluation, so Table 3 compares the raw automatic pool with the human-audited Verified release. The means shift modestly, but all three sources retain the H Pro–H ordering, and their relationships to seed diffi-

Table 3: **Difficulty before and after item audit.** Avg@5 (%) over the fixed 16-model panel and each pool's retained items. The direction of the hardening effect persists after audit.

	Seeds	Raw automatic pool		Verified	
Source		H	H Pro	H	H Pro
AIME-2024-II	84.91	67.80	58.57	68.71	58.57
Beyond-AIME	58.34	58.78	53.98	59.45	55.02
AMO-Bench	31.75	42.97	38.38	44.56	40.16

Table 4: **Hardened-release construction quality.** Candidates: automatic pool; final accepted: Verified after audit and repair. Verified Full preserves these 2,187 items and adds 112 accepted replacements. Totals include historical AIME alongside the three frontier-model evaluation sources.

Source	Mode	Candidates	Initial pass	Final accepted
AIME 1983–2001	H	1,321	1,043	1,284
AIME-2024-II	H	70	63	69
Beyond-AIME	H	500	365	466
AMO-Bench	H	244	155	217
AIME-2024-II	H Pro	14	11	14
Beyond-AIME	H Pro	100	69	93
AMO-Bench	H Pro	50	27	44
Total		2,299	1,733	2,187

culty remain unchanged. The main difficulty conclusions therefore hold on questions independently checked by human solvers.

5.3 Targeted Repair Recovers Rejected Items for the Release

Repair raises usable yield from 75.38% to 95.13%. We measure construction yield as the fraction of generated candidate items that become usable questions after expert review and any necessary repair. The initial audit accepts 1,733 of 2,299 candidates. Targeted repair adds 454 accepted items, bringing the release to 2,187. Validation therefore improves usable yield as well as checking quality: identified errors become corrections to the question, computation, or parameter constraints. The two stages quantify the contributions of automatic construction and subsequent repair.

Table 4 reports the source-specific acceptance counts before and after repair, allowing construction yield and evaluation difficulty to be read together.

Repair acts on the source of an error: wording for an ambiguous question, code for an arithmetic error, or domain constraints for a degenerate assignment. Each correction undergoes renewed validation and enters the yield count after acceptance. The audit record links the accepted answer to this repair history.

Coverage restoration adds 112 accepted replacements. Across seven source–mode groups, Verified contains 2,187 accepted items, including 1,284 historical AIME items. Adding the 112 accepted replacements yields Verified Full, whose complete set of positions supports future evaluations across the source banks. The reported frontier-model comparison follows the Verified scoring release on AIME-2024-II, Beyond-AIME, and AMO-Bench, with the retained counts given in Section 4.

6 Implications for Renewable Evaluation

Instance-level failures reveal what aggregate shifts conceal. The worked comparisons in Figure 6 illustrate two concrete failure patterns. In the colored-chip problem, DeepSeek V3.2 Thinking answers the 5×5 seed correctly but returns 5 for a 1×2 variant whose answer is 2. The accompanying trace analysis identifies a calculation for the wrong grid dimensions. In the polygon-coloring

problem, Gemini 3 Pro answers the octagon seed correctly but returns 2,685 for a 12-gon variant whose answer is 2,595. The case analysis is correct through four vertices of one color and fails when it must extend to larger cases.

The parameters, rendered question, and computed answer localize each error. Incorrect dimensions and incomplete case analysis call for different repairs, both exposed by fresh inputs despite correct seed answers.

A diagnostic for possible benchmark leakage. The AIME pattern is consistent with benchmark-specific overfitting, including possible contamination or leakage: strong performance on familiar items transfers incompletely to fresh instances of their computations. GSM8K's high, stable accuracy supplies an easier reference; Beyond-AIME's lower, stable accuracy supplies a demanding one. Together, they motivate using E near a solver's capability boundary, where familiar-item success may conceal fragile computation. The diagnostic flags candidates for exposure investigation; attributing a drop to training-data leakage requires exposure evidence in addition to these behavioral scores.

Further E draws can locate the unstable parameter regimes. When performance remains high across a family, H modifications and H Pro selection can supply more challenge. Fixing the accepted release before student evaluation preserves a common test, while versioned specifications make later renewal reproducible.

Separating validity from challenge. Validation establishes usable questions and answers; student evaluation measures their challenge. Repair improves construction yield, while candidate selection targets difficulty. The specification, audit outcome, and selection record preserve the evidence for both requirements as the benchmark is renewed.

7 Related Work

Freshness and mathematical robustness. GSM8K and MATH establish core reasoning benchmarks (Cobbe et al., 2021; Hendrycks et al., 2021b); Beyond-AIME and AMO-Bench extend their difficulty range (ByteDance Seed et al., 2025; An et al., 2025). Exposure analyses and live evaluation motivate fresh test material (Shi et al., 2023; Golchin & Surdeanu, 2024; Sainz et al., 2024; Jain et al., 2025; White et al., 2025). GSM-Plus, GSM-Symbolic, GSM-Ranges, and VAR-MATH vary statements or numerical assignments (Li et al., 2024; Mirzadeh et al., 2025; Shrestha et al., 2025; Yao et al., 2025); Putnam-AXIOM and RV-Bench extend functional evaluation to further mathematical settings (Gulati et al., 2025; Hong et al., 2026). VeRA compiles reusable families and adds computation modification plus candidate selection to renew headroom.

Dynamic construction and verification. DyVal and DARG generate reasoning graphs (Zhu et al., 2024; Zhang et al., 2024c); MaSTer searches for challenging rewrites (Hou et al., 2026); AutoLogi verifies generated logic puzzles through execution (Zhu et al., 2025). VeRA joins the question template, valid-input generator, and answer program. PAL and Program of Thoughts execute solutions (Gao et al., 2023; Chen et al., 2023); EvalPlus expands tests of generated code (Liu et al., 2023). Here, execution labels new benchmark instances, with program checks, independent item audits, and repair establishing their usability. Verification and multi-agent debate supply complementary checks (Lightman et al., 2024; Du et al., 2024). Appendix M compares construction interfaces and validation targets.

8 Conclusion

VeRA renews reasoning benchmarks through executable task families. Computation-preserving variation retains mean accuracy on GSM8K and Beyond-AIME while exposing a selective AIME drop, providing a diagnostic for benchmark-specific overfitting and possible contamination. Modified families and candidate selection supply additional challenge; independent audits and repair establish usable releases. Together, these components support repeated evaluation of both fresh-instance robustness and progress on harder problems.

AI USAGE STATEMENT

Language models proposed specifications for synthetic data generation, performed the stated validation and ranking tasks for LLM-as-a-judge, and served as evaluated student models for benchmarking. **With human in the loop for supervision and quality control**, coding agents assisted with writing, better result presentation, and experiment execution. **All research ideas, structure of the manuscript, and result analysis strategies originated with the human authors**, who have reviewed and revised the codebase and the manuscript, and human authors take full responsibility for all the scientific claims and artifacts.

REPRODUCIBILITY STATEMENT

Benchmark code and data are available at `https://anonymous.4open.science/r/VeRA-ICLR2027-5EC1`. The repository contains seed and generated datasets, generator–verifier specifications, and construction, validation, and evaluation code. The main hardening results use *Verified*, the release of items accepted after independent expert solving and repair. *Verified Full* preserves those accepted items and adds validated replacements to restore complete source coverage. Section 3 describes the expert audit, and Table 4 reports acceptance and repair counts. Sections 2–4 and Appendix I specify the construction and evaluation procedures; the repository README documents installation, model endpoints, and execution settings. The manuscript source archive includes release accounting, reported model tables, verification counts, and scripts that regenerate figures and extended tables.

REFERENCES

Shengnan An, Xunliang Cai, Xuezhi Cao, Xiaoyu Li, Yehao Lin, Junlin Liu, et al. AMO-Bench: Large Language Models Still Struggle in High School Math Competitions. *arXiv preprint arXiv:2510.26768*, 2025. URL https://arxiv.org/abs/2510.26768.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, et al. Program Synthesis with Large Language Models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

ByteDance Seed, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, et al. Seed1.5-Thinking: Advancing Superb Reasoning Models with Reinforcement Learning. *arXiv preprint arXiv:2504.13914*, 2025. URL https://arxiv.org/abs/2504.13914.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2211.12588.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging Procedural Generation to Benchmark Reinforcement Learning. *arXiv preprint arXiv:1912.01588*, 2019. URL https://arxiv.org/abs/1912.01588.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving Factuality and Reasoning in Language Models through Multiagent Debate. In *Proceedings of the International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2305.14325.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, et al. PAL: Program-aided Language Models. In *Proceedings of the International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2211.10435.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, et al. Datasheets for Datasets. *Communications of the ACM*, 2021. URL https://arxiv.org/abs/1803.09010.

Shahriar Golchin and Mihai Surdeanu. Time Travel in LLMs: Tracing Data Contamination in Large Language Models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.08493.

Aryan Gulati, Brando Miranda, Eric Chen, Emily Xia, Kai Fronsdal, Bruno Dumont, et al. Putnam-AXIOM: A Functional and Static Benchmark for Measuring Higher Level Mathematical Reasoning in LLMs. *arXiv preprint arXiv:2508.08292*, 2025. URL https://arxiv.org/abs/2508.08292.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, et al. Textbooks Are All You Need. *arXiv preprint arXiv:2306.11644*, 2023. URL https://arxiv.org/abs/2306.11644.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://arxiv.org/abs/2501.12948.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, et al. Measuring Coding Challenge Competence With APPS. In *NeurIPS Datasets and Benchmarks*, 2021a. URL https://arxiv.org/abs/2105.09938.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, et al. Measuring Mathematical Problem Solving With the MATH Dataset. In *NeurIPS Datasets and Benchmarks*, 2021b. URL https://arxiv.org/abs/2103.03874.

Zijin Hong, Hao Wu, Su Dong, Junnan Dong, Yilin Xiao, Yujing Zhang, et al. Benchmarking LLMs' Mathematical Reasoning with Unseen Random Variables Questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://arxiv.org/abs/2501.11790.

Yutao Hou, Zeguan Xiao, Fei Yu, Yihan Jiang, Ma Shuguang, Zhaoqian Dai, et al. Toward Automated Robustness Evaluation of Mathematical Reasoning. In *Findings of the Association for Computational Linguistics: ACL*, 2026. URL https://arxiv.org/abs/2506.05038.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, et al. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2403.07974.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2207.01780.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. GSM-Plus: A Comprehensive Benchmark for Evaluating the Robustness of LLMs as Mathematical Problem Solvers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2402.19255.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, et al. TACO: Topics in Algorithmic COde generation dataset. *arXiv preprint arXiv:2312.14852*, 2023. URL https://arxiv.org/abs/2312.14852.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158. URL https://arxiv.org/abs/2203.07814.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, et al. Let's Verify Step by Step. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

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. *arXiv preprint arXiv:2305.01210*, 2023. URL https://arxiv.org/abs/2305.01210.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.05229.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive Learning from Complex Explanation Traces of GPT-4. *arXiv preprint arXiv:2306.02707*, 2023. URL https://arxiv.org/abs/2306.02707.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, et al. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. *arXiv preprint arXiv:2311.12022*, 2023. URL https://arxiv.org/abs/2311.12022.

Oscar Sainz, Jon Ander Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. NLP Evaluation in trouble: On the Need to Measure LLM Data Contamination for each Benchmark. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024. URL https://arxiv.org/abs/2310.18018.

Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, et al. Detecting Pretraining Data from Large Language Models. *arXiv preprint arXiv:2310.16789*, 2023. URL https://arxiv.org/abs/2310.16789.

Safal Shrestha, Minwu Kim, and Keith Ross. Mathematical Reasoning in Large Language Models: Assessing Logical and Arithmetic Errors across Wide Numerical Ranges. *arXiv preprint arXiv:2502.08680*, 2025. URL https://arxiv.org/abs/2502.08680.

Gaurav Srivastava, Aafiya Hussain, Zhenyu Bi, Swastik Roy, Priya Pitre, Meng Lu, et al. BeyondBench: Contamination-Resistant Evaluation of Reasoning in Language Models. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2509.24210.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, et al. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations*, 2023a. URL https://arxiv.org/abs/2203.11171.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, et al. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2023b. URL https://arxiv.org/abs/2212.10560.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*, 2022. URL https://arxiv.org/abs/2201.11903.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, et al. LiveBench: A Challenging, Contamination-Limited LLM Benchmark. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.19314.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, et al. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, 2024a. URL https://arxiv.org/abs/2304.12244.

Cheng Xu, Shuhao Guan, Derek Greene, and M-Tahar Kechadi. Benchmark Data Contamination of Large Language Models: A Survey. *arXiv preprint arXiv:2406.04244*, 2024b. URL https://arxiv.org/abs/2406.04244.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, et al. LeanDojo: Theorem Proving with Retrieval-Augmented Language Models. In *NeurIPS Datasets and Benchmarks*, 2023. URL https://arxiv.org/abs/2306.15626.

Jian Yao, Ran Cheng, and Kay Chen Tan. VAR-MATH: Probing True Mathematical Reasoning in LLMS via Symbolic Multi-Instance Benchmarks. *arXiv preprint arXiv:2507.12885*, 2025. URL https://arxiv.org/abs/2507.12885.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, et al. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.12284.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, et al. MAmmoTH: Building Math Generalist Models through Hybrid Instruction Tuning. *arXiv preprint arXiv:2309.05653*, 2023. URL https://arxiv.org/abs/2309.05653.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning. *arXiv preprint arXiv:2203.14465*, 2022. URL https://arxiv.org/abs/2203.14465.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, et al. A Careful Examination of Large Language Model Performance on Grade School Arithmetic. In *NeurIPS Datasets and Benchmarks*, 2024a. URL https://arxiv.org/abs/2405.00332.

Huixuan Zhang, Yun Lin, and Xiaojun Wan. PaCoST: Paired Confidence Significance Testing for Benchmark Contamination Detection in Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024b. URL https://arxiv.org/abs/2406.18326.

Zhehao Zhang, Jiaao Chen, and Diyi Yang. DARG: Dynamic Evaluation of Large Language Models via Adaptive Reasoning Graph. *arXiv preprint arXiv:2406.17271*, 2024c. URL https://arxiv.org/abs/2406.17271.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, et al. Absolute Zero: Reinforced Self-play Reasoning with Zero Data. *arXiv preprint arXiv:2505.03335*, 2025. URL https://arxiv.org/abs/2505.03335.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. MiniF2F: a cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2109.00110.

Kaijie Zhu, Jiaao Chen, Jindong Wang, Neil Zhenqiang Gong, Diyi Yang, and Xing Xie. DyVal: Dynamic Evaluation of Large Language Models for Reasoning Tasks. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17167.

Qin Zhu, Fei Huang, Runyu Peng, Keming Lu, Bowen Yu, Qinyuan Cheng, et al. AutoLogi: Automated Generation of Logic Puzzles for Evaluating Reasoning Abilities of Large Language Models. *arXiv preprint arXiv:2502.16906*, 2025. URL https://arxiv.org/abs/2502.16906.

A Model-Level Results

The model-level tables resolve the panel averages into individual robustness and hardening profiles. Avg@5 averages the correctness of five sampled responses per problem. Entries are rounded accuracy values; differences calculated from these cells can differ by 0.1 point from differences calculated before rounding.

Table 5: GSM8K and AIME 2024 model-level Avg@5 accuracy (%).

Model	GSM8K		AIME 2024	
	Seed	Variant	Seed	Variant
Gemini 2.5 Pro	93.9	95.0	88.0	73.3
GLM-4.6	94.1	94.8	90.7	71.7
GPT-5.1 high	95.6	96.9	92.0	75.3
Claude Sonnet 4.5 Thinking	95.3	96.2	84.0	74.0
Seed-1.6-Thinking-0715	95.1	95.9	85.3	72.0
DeepSeek V3.2 Thinking	94.0	95.4	92.0	79.0
Seed-1.6-1015 high	95.6	96.3	89.3	75.0
Kimi K2 Thinking	95.0	86.6	86.7	75.3
MiniMax M2	94.7	94.9	84.7	55.7
Gemini 3 Pro Preview	94.5	95.0	94.0	81.3
GPT-5 high	95.6	97.1	90.0	74.7
Kimi K2-0905	95.0	95.6	65.3	51.3
Qwen3-Max-0923	95.5	96.9	82.7	68.7
GPT-5.1-chat-latest	94.3	96.0	48.7	46.0
Seed-1.6-Lite-1015 high	95.1	95.8	87.3	73.0
DeepSeek V3.1 Thinking	94.5	94.8	90.7	77.7

Table 6: AIME 2025 and Beyond-AIME model-level Avg@5 accuracy (%).

Model	AIME 2025		Beyond-AIME	
	Seed	Variant	Seed	Variant
Gemini 2.5 Pro	85.3	73.0	57.2	57.0
GLM-4.6	92.0	76.3	68.2	66.4
GPT-5.1 high	88.7	88.3	71.6	73.0
Claude Sonnet 4.5 Thinking	78.7	68.7	51.6	53.0
Seed-1.6-Thinking-0715	80.0	75.3	54.4	55.8
DeepSeek V3.2 Thinking	90.0	87.7	69.4	67.6
Seed-1.6-1015 high	80.7	76.7	58.8	58.2
Kimi K2 Thinking	78.7	71.7	54.8	44.0
MiniMax M2	76.7	57.0	56.8	54.4
Gemini 3 Pro Preview	90.7	85.7	76.6	74.0
GPT-5 high	89.3	85.3	68.0	69.8
Kimi K2-0905	47.3	44.0	37.0	34.6
Qwen3-Max-0923	74.7	69.7	53.6	58.8
GPT-5.1-chat-latest	50.7	41.7	28.8	30.6
Seed-1.6-Lite-1015 high	77.3	70.0	54.8	53.6
DeepSeek V3.1 Thinking	86.7	83.0	71.8	66.0

B Audit Scope and Accepted Cohorts

The direct VeRA-E audit covers 60 AIME-2024 variants, 60 AIME-2025 variants, and 100 Beyond-AIME variants. All 220 pass the audit. The hardened release additionally contains a historical AIME cohort. Across all seven source–mode groups in Table 4, 1,733 items pass initially and 454 pass after repair, totaling 2,187 accepted items out of 2,299 candidates. The audit unit is a rendered question and its answer; the release manifest identifies every accepted item.

Auditors independently solve the rendered questions, compare their answers with program outputs, check unresolved placeholders and ambiguity, and inspect family correspondence for VeRA-E. Un-

resolved disagreements are rejected after senior adjudication. The source audit describes failures involving edge-case arithmetic, ambiguous renderings, and degenerate parameter regimes. These failures motivate independent item review even after a program passes execution checks.

C Extent of Variation Across AIME Years

The perturbation diagnostic compares each variant with its seed using numerical and textual distance. Numerical distance averages the relative slot changes $|\theta_j - \theta_j^{\text{seed}}| / \max(|\theta_j^{\text{seed}}|, 1)$, clipped to $[0, 1]$. Textual distance is the token-level Levenshtein distance divided by the seed rendering's length. Their mean defines the perturbation score π.

Figure 4 reports these distributions in the main text.

Similar edit distances accompany different changes in solution accuracy. The variant-level Kolmogorov–Smirnov statistic is $D = 0.15$ (nominal $p = 0.51$ under independent observations). Each source seed contributes two related variants, making the source family the unit for cluster-based inference. The distributions characterize the extent of editing; the model-level profiles in Figure 3 measure its consequences for solving the resulting problems.

D Estimands and Selection

Let Y_{mijr} indicate correctness for model m, source family i, generated item j, and response draw r. A model's reported score on an item set $\mathcal{D}$ is

$$S_m(\mathcal{D}) = \frac{100}{5|\mathcal{D}|} \sum_{(i,j)\in\mathcal{D}} \sum_{r=1}^{5} Y_{mijr}.$$

The suite mean averages S_m over the fixed 16-model panel. Each generated item receives equal weight, so a family contributes in proportion to its accepted item count. Items generated from a common specification belong to one family cluster. Generalization across source problems therefore uses family-level resampling, while the model average summarizes the evaluated panel.

VeRA-H Pro selection, item audit, and coverage restoration define the evaluation sets. The main hardening scores compare the seed bank with the accepted items in Verified. Verified Full supplies additional accepted items for complete source coverage. Table 4 gives the item counts for both releases; all accuracy differences use the item-weighted score above on the stated sets.

E Raw-Pool and Verified Comparisons

Table 3 reports both comparisons in the main text. Audit selection changes the score level modestly while preserving the release-level comparisons: AIME-2024-II gains headroom over its seeds, Beyond-AIME gains mainly through Pro selection, and AMO-Bench's H Pro mean falls below H while remaining above the seed mean. The release accounting in Table 4 identifies the corresponding accepted cohorts.

F Arbitration-Based GPQA Extension

GPQA-Diamond (Rein et al., 2023) uses a separate arbitration-based answer-validation procedure. Its 198 seeds produce 871 variants, with a 16-model mean of 79.27% on seeds and 79.42% on variants. This extension broadens the application domain, while its label assurance comes from arbitration rather than an executable mathematical answer program. It is therefore reported separately from the mathematical results.

G Reproducibility and Exposure

A task-family record includes the seed identity, slot schema, template, generator, verifier, version, sampling index, and validation outcomes. Those fields make the resulting question and answer re-

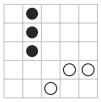
producible. Holding out sampling indices changes the concrete instances; holding out entire families additionally changes the computation being tested. Both splits are useful and answer different generalization questions. If task families are used for training, future evaluation should document that exposure and preserve an appropriate held-out family or input regime.

H Worked Comparisons and Construction Workflows

A. Maximal chip placements

Seed: a 5×5 grid

Place 25 white and 25 black chips on a 5×5 grid, with at most one chip per cell. Every occupied row and column must be monochromatic. Count the placements to which no chip can be added while preserving these conditions.



Gold answer: 902
DeepSeek V3.2 Thinking: **902 (correct)**.

Variant: a 1×2 grid

Place four red and three purple chips on a 1×2 grid, under the same occupancy, monochromaticity, and maximality conditions. How many placements are possible?

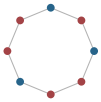


Gold answer: 2
DeepSeek V3.2 Thinking: **5 (incorrect)**.
Trace diagnosis. The solution uses a 2×2 grid instead of the stated 1×2 grid.

B. Rotations of a colored polygon

Seed: a regular octagon

Independently color each vertex of a regular octagon red or blue, each with probability $1/2$. Find the probability that some rotation maps every blue vertex to a vertex originally colored red. Write it as m/n in lowest terms and find $m + n$.



Gold answer: 371
Gemini 3 Pro: **371 (correct)**.

Variant: a regular 12-gon

Independently color each vertex of a regular 12-gon green or purple, each with probability $1/2$. Find the probability that some rotation maps every purple vertex to a vertex originally colored green. Write it as m/n in lowest terms and find $m + n$.
Gold answer: 2595
Gemini 3 Pro: **2685 (incorrect)**.
Trace diagnosis. The larger polygon requires cases with up to six purple vertices. The model handles at most four correctly but fails for five purple vertices.

Figure 6: **Correct seeds can coexist with incorrect variants.** The two worked comparisons show the task statements and model answers. DeepSeek V3.2 Thinking uses the wrong grid dimensions after the inputs change; Gemini 3 Pro leaves a case analysis incomplete after the number of polygon vertices increases. Executable families expose these failures at the level of the changed computation.

Seed: AIME 2024 II, Problem 10

Triangle ABC has incenter I, circumcenter O, $IA \perp OI$, circumradius $R = 13$, and inradius $r = 6$. Find $AB \cdot AC$. **Answer: 468**

↓

VeRA-E: change parameters and language

Retain $IA \perp OI$ and the same computation. Change (R, r) to $(89, 22)$ and render the question in Spanish.

En el triángulo ABC, I es el incentro y O el circuncentro. Si $IA \perp OI$, $R = 89$ y $r = 22$, halle $AB \cdot AC$.
Answer: $6Rr = 11{,}748$

VeRA-H: change the computational structure

Replace perpendicularity with an angle condition. Extend the specification and verifier to accept (R, r, θ).

Candidate: $\angle(IA, OI) = 60^\circ$, $R = 15$, $r = 5$. Find $AB \cdot AC$ under the modified specification.
Answer: $V_H(15, 5, 60^\circ)$

VeRA-H Pro: select a hard candidate

Generate a pool of H candidates from the seed. Select the candidate judged hardest within the pool.

Selected candidate: $\angle(IA, OI) = 75^\circ$, $R = 38$, $r = 10$.
Answer: $V_H(38, 10, 75^\circ)$

V_H is the modified verifier, including its angle convention and domain checks.

Figure 7: **Three construction modes for a geometry family.** The schematic contrasts parameter and language changes, a structural change to the angle condition, and selection from a candidate pool. V_H denotes the modified verifier's output, with its angle convention and domain checks. The E example has the perpendicularity condition and answer $6Rr = 11{,}748$.

1. Lift the seed into a specification

Triangle ABC has incenter I, circumcenter O, $IA \perp OI$, $R = 13$, and $r = 6$. Find $AB \cdot AC$.
Seed answer: 468

2. Define the generator, renderer, and answer program

E: retain the computation

Slots: (R, r); condition: $IA \perp OI$.

```
def gen_E():
    R, r = sample_R_r()
    spec = pick_spec_E()
    style = sample_LC_E_style()
    return spec, (R, r), style

def render_E(spec, R, r, style):
    return spec.template.render(
        R=R, r=r, **style)

def verify_label_E(R, r):
    assert valid_R_r(R, r)
    OI2 = R * (R - 2*r)
    IA2 = R*R - OI2
    return IA2 + 4*R*r
```

The perpendicularity condition gives $IA^2 = 2Rr$, hence $AB \cdot AC = 6Rr$.

H: modify the computation

Slots: (R, r, θ); angle condition.

```
def gen_H(mode="H"):
    spec = pick_spec_H()
    if mode == "H":
        args = sample_R_r_theta()
    else:
        pool = [sample_R_r_theta()
                for _ in range(K)]
        args = select_hardest(pool)
    style = sample_LC_H_style()
    return spec, args, style

def render_H(spec, R, r, th, style):
    return spec.template.render(
        R=R, r=r, theta=th, **style)

def verify_label_H(R, r, th):
    assert valid_R_r_theta(R, r, th)
    IA = solve_from_coslaw(R, r, th)
    return IA*IA + 4*R*r
```

3. Sample assignments, render questions, and return answers

E examples

English. Keep $IA \perp OI$ and set $R = 14$, $r = 5$. Find $AB \cdot AC$.
Answer: 420

Spanish. Si $IA \perp OI$, $R = 6$ y $r = 2$, halle $AB \cdot AC$.
Answer: 72

H candidates and H Pro selection

A: $(15, 5, 60^\circ)$ $V_H(15, 5, 60^\circ)$
B: $(38, 10, 75^\circ)$ $V_H(38, 10, 75^\circ)$
C: $(22, 7, 40^\circ)$ $V_H(22, 7, 40^\circ)$
H Pro example: select B.
The selected specification can also be rendered in Spanish; its verifier remains V_H.

Figure 8: **From a geometry seed to executable families.** The three stages are specification, sampling, and rendering. E keeps the perpendicularity condition; H exposes an angle parameter. The code blocks express the roles of sampling and answer computation. The H branches illustrate candidate specifications; validation and item auditing produce the evaluated releases described in the main text.

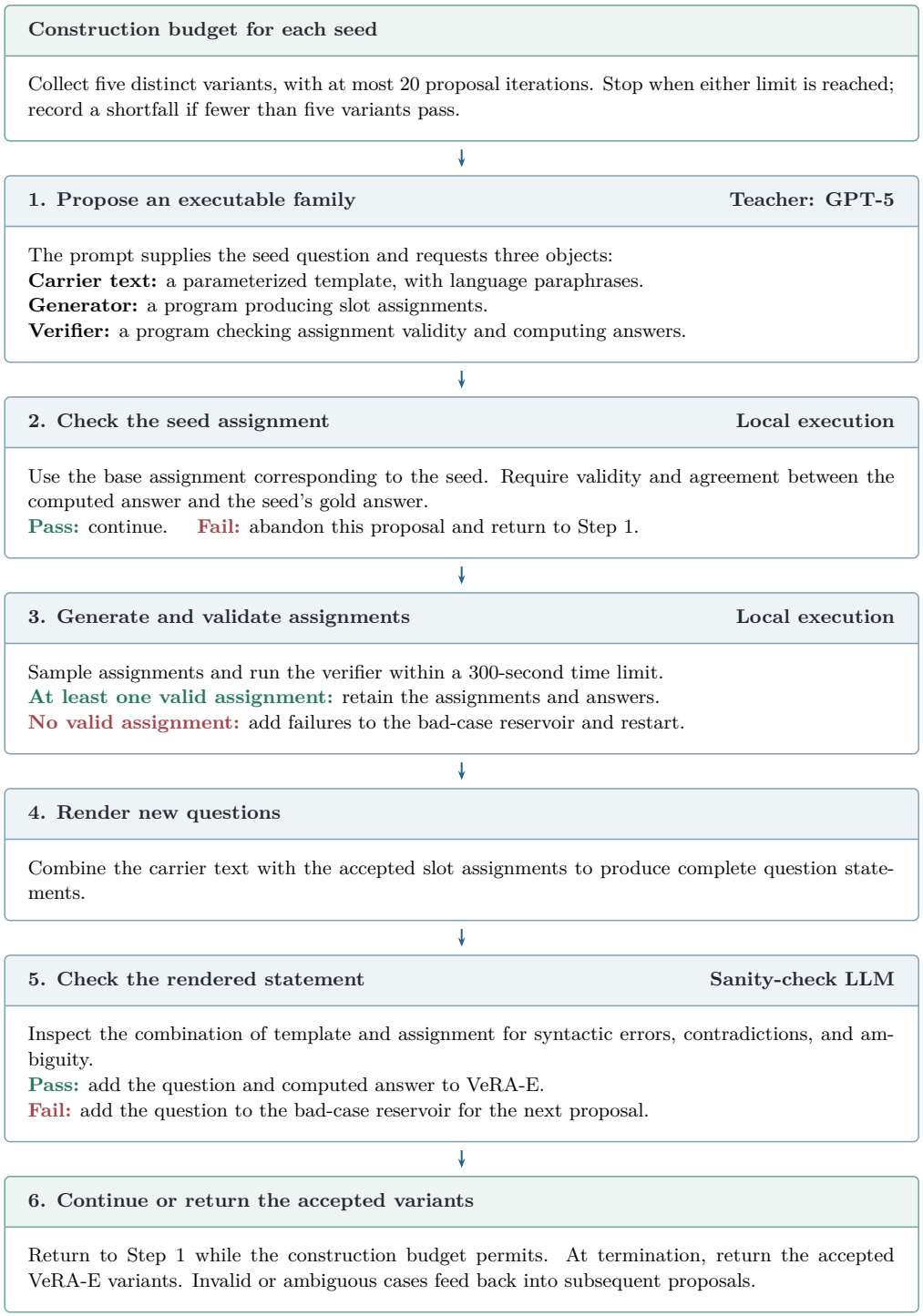


Figure 9: **The detailed E construction loop.** Seed anchoring, valid-assignment generation, and rendering checks provide feedback to the teacher. The diagram's five-variant stopping budget describes the construction interface; the evaluated draws per family follow the cohorts in Section 4.

Construction budget for each seed

Collect five distinct variants, with at most 20 proposal iterations. Stop at either limit; record a shortfall if fewer than five variants pass.

↓

1. Propose a hardened family — **Teacher: GPT-5**

Use the seed and prompt template to propose a modified carrier text, an assignment generator, and a verifier that checks validity and computes answers. The carrier text supports parameter and language variation.

↓

2. Generate and validate assignments — **Local execution**

Sample assignments and execute the verifier within a 300-second time limit.
Valid assignments: retain them with their computed answers.
No valid assignment: add failures to the bad-case reservoir and restart.

↓

3. Render questions; 4. Inspect them — **Sanity-check LLM**

Combine the carrier text and accepted assignments. Check the statements for syntactic errors, contradictions, and ambiguity.
Pass: move to answer inspection. **Fail:** add the statement to the bad-case reservoir and restart.

↓

5. Inspect the candidate answer — **Judge: GPT-5**

Supply the seed solution, the new question, and a candidate answer. Run five independent judgments:
Two trials: use the computed answer; expected judgment is True.
Three trials: add slight noise to that answer; expected judgment is False.
At least four expected judgments: accept the instance into VeRA-H.
Otherwise: retain it in the uncertain-answer reservoir, outside the accepted pool.

↓

6. Continue until the construction budget is reached

Return to Step 1 while the budget permits. Failed and ambiguous cases feed back to subsequent proposals; accepted instances form the H pool.

↓

7. Select one hard instance per seed — **Judge: GPT-5**

After construction, rank the accepted H candidates from the same seed and select exactly one judged hardest. These selected instances form VeRA-H Pro.

Figure 10: **The detailed H and H Pro construction loop.** Answer perturbation trials test the proposed labels, and a difficulty judge selects candidates. These validation gates precede the expert item audit and repair whose outcomes appear in Table 4.

I Specification Schema and Validation Procedure

I.1 A Complete Family Record

An executable family records slot names and types, admissible assignments, a rendering template, a generator, and a verifier. The base assignment links an E family to its seed. H families additionally record which condition changed and the rationale for the modification. These fields connect the question seen by a student with the computation used to produce its answer.

Table 7: Specification fields and their scientific roles.

Field	Role
Slot schema	Defines quantities, types, ranges, and relationships among inputs.
Template	Renders the assignment into the question presented to the student.
Generator	Samples assignments using the supplied random-number source.
Verifier	Checks an assignment and computes its canonical answer.
Base assignment	Recreates the seed case for E anchoring.
Modification rationale	Identifies the intended change in constraints or computation for H.
Validation record	Associates the family and sampled items with execution, judgment, and audit outcomes.

For example, consider a family asking for the total in a rows of b objects plus c additional objects. Let a, b be integers from 2 through 20, let c range from 1 through 10, and impose $a \neq b$. A coherent generator draws a, then selects b from the remaining permitted values, and draws c. The verifier checks these constraints and returns $ab + c$. The assignment $(5, 4, 3)$ renders a problem with answer 23. This small example isolates the interface: a constraint belongs in both the sampling procedure and the validity check, and the template must state the quantities used by the answer program.

I.2 Proposal, Execution, and Repair

The teacher proposes a structured family rather than a single free-form problem. The pipeline parses its schema, compiles its functions, and tests sampled assignments. In E mode, the original assignment must reproduce the seed answer. In H mode, the modified family is checked on its own constraints and answer program. A failed check returns concrete feedback, including compilation errors, invalid assignments, failure to yield a valid instance, or an answer mismatch. The teacher can repair the specification within the proposal budget.

The synthesis protocol uses a proposal limit of 20, five sampled assignments per proposal, and a target of up to five accepted H candidates per seed. These are construction budgets; the evaluation cohorts specify how many resulting questions are actually scored. Timeouts bound program execution, and the released helper uses child processes with supported CPU and memory limits. Reproducibility also requires a fixed random source: generator randomness is supplied through a seeded interface rather than an implicit global state.

The implementation has an explicit fallback that restates the seed when synthesis produces no accepted variant. Its metadata identifies this path. Such an item preserves a seed-level question and answer but represents a different transformation from successful parameter resampling. Tracking fallback status therefore separates coverage from the strength of the generated variation.

I.3 Synthesis Yield

The source synthesis study records how many seeds produce specifications within the proposal budget. Table 8 separates successful compilation from fallback in E, and full from partial candidate yield in H. Its counts concern construction; the student-evaluation and audited-release cohorts are specified separately.

Table 8: Reported construction yield. E counts seeds with a successful specification; H counts seeds by the number of accepted candidates. The H study targets five candidates per seed.

E source	Seeds	First try	By try 20	Fallback
GSM8K	1,319	1,319	1,319	0
AIME-2024	30	29	30	0
AIME-2025	30	27	30	0
Beyond-AIME	100	63	89	11
H source	**Seeds**	**Five candidates**	**Four**	**Zero**
Beyond-AIME	100	100	0	0
AMO-Bench	50	48	1	1

The E results show that repeated proposals recover specifications for some seeds that fail on the first try. The Beyond-AIME fallback count also shows why source coverage and successful family construction are different quantities. In H, the AMO-Bench counts yield $48 \times 5 + 1 \times 4 = 244$ candidates, matching the raw-pool size used in its model-level evaluation.

I.4 Answer-Noise Discrimination

The H validity filter checks whether a judge can distinguish the verifier's answer from perturbed alternatives. It presents two trials with the verifier answer and three with controlled noise answers in shuffled order. Integer perturbations use nonzero offsets; real-valued perturbations change the value relative to its scale. At least four correct classifications are required. This checks sensitivity to sampled answer errors in addition to agreement with a proposed answer.

The original calibration examined 50 AIME variants against human judgments. At threshold four it recorded one false acceptance and three false rejections; at threshold five it recorded zero and fourteen. These are error counts over the 50-item calibration set. Class-conditional error rates additionally require the numbers of truly valid and invalid candidates. The stricter threshold reduced accepted errors in this sample while also reducing usable yield. Expert item auditing supplies a further check on the released questions.

I.5 Independent Audit and Coverage Restoration

Auditors solve the rendered question before viewing the generated verifier, compare their solution with its answer, and check that the wording is complete and unambiguous. E additionally requires that the question preserve the intended family relation. Senior adjudication handles disagreements; unresolved cases are rejected. The release distinguishes initial audit passes from repaired-and-promoted items, making the final 2,187 accepted entries interpretable as a post-repair outcome.

Three qualitative failure types explain the need for this separation. A verifier can mishandle an integer or rational boundary. A template can permit two readings even when its code has one answer. A sampled parameter regime can collapse a proposed challenge into a degenerate case. Execution tests, semantic inspection, and student evaluation address these different issues. Verified Full restores source positions with accepted replacements and consequently defines a separate comparison set from the initially retained Verified cohort.

J Model-Level Hardening and Domain Extension

The following tables give the model-level raw-pool comparisons for historical AIME, AIME-2024-II, Beyond-AIME, and AMO-Bench. Historical AIME contributes 265 seeds and 1,321 H items. The latter three sources have H/H Pro item counts of 70/14, 500/100, and 244/50, respectively. The main tables report the corresponding Verified cohorts. Keeping the two sets explicit reveals both model heterogeneity and the effect of acceptance selection.

The raw-pool results also show why "hardest" refers to the selection rule, rather than every student's score. For Kimi K2 Thinking on AIME-2024-II, H Pro accuracy exceeds the all-candidate H accuracy, whereas most models show the opposite order. A judge's ranking of candidate difficulty can

differ from a particular student's difficulty profile. The aggregate AIME hardening effect coexists with these model-specific responses.

Table 9: Historical AIME (1983–2001) raw-pool Avg@5 accuracy (%). The source evaluation uses 265 seeds and 1,321 H variants. The separate Verified release retains 1,284 H items.

Model	Seed	H
Gemini 2.5 Pro	95.4	79.5
GLM-4.6	93.8	75.6
GPT-5.1 high	95.2	78.4
Claude Sonnet 4.5 Thinking	94.0	75.8
Seed-1.6-Thinking-0715	93.8	79.1
DeepSeek V3.2 Thinking	95.7	77.7
Seed-1.6-1015 high	94.3	79.1
Kimi K2 Thinking	59.6	71.4
MiniMax M2	90.9	74.7
Gemini 3 Pro Preview	93.4	74.7
GPT-5 high	94.5	77.4
Kimi K2-0905	84.1	53.7
Qwen3-Max-0923	92.2	74.2
GPT-5.1-chat-latest	72.1	55.3
Seed-1.6-Lite-1015 high	93.1	76.3
DeepSeek V3.1 Thinking	94.4	75.3

Table 10: AIME-2024-II raw-pool Avg@5 accuracy (%). These are the reported pre-audit model-level results; the main hardening table uses the separate Verified cohorts.

Model	Seed	H	H Pro
Gemini 2.5 Pro	92.9	82.0	65.7
GLM-4.6	94.3	81.1	67.1
GPT-5.1 high	94.3	90.9	74.3
Claude Sonnet 4.5 Thinking	90.0	72.0	60.0
Seed-1.6-Thinking-0715	87.1	76.6	67.1
DeepSeek V3.2 Thinking	91.4	84.0	68.6
Seed-1.6-1015 high	92.9	76.6	67.1
Kimi K2 Thinking	85.7	15.1	45.7
MiniMax M2	82.9	64.3	51.4
Gemini 3 Pro Preview	75.7	74.6	62.9
GPT-5 high	88.6	82.3	78.6
Kimi K2-0905	71.4	40.6	32.9
Qwen3-Max-0923	80.0	59.1	51.4
GPT-5.1-chat-latest	52.9	34.3	28.6
Seed-1.6-Lite-1015 high	88.6	70.3	57.1
DeepSeek V3.1 Thinking	90.0	81.1	58.6

Table 11: Beyond-AIME raw-pool Avg@5 accuracy (%). These are the reported pre-audit model-level results; the main hardening table uses the separate Verified cohorts.

Model	Seed	H	H Pro
Gemini 2.5 Pro	57.2	57.7	52.2
GLM-4.6	68.2	67.1	63.6
GPT-5.1 high	71.6	77.3	75.2
Claude Sonnet 4.5 Thinking	51.6	56.3	50.2
Seed-1.6-Thinking-0715	54.4	57.9	51.6
DeepSeek V3.2 Thinking	69.4	68.6	64.2
Seed-1.6-1015 high	58.8	61.8	55.0
Kimi K2 Thinking	54.8	43.8	46.6
MiniMax M2	56.8	52.8	47.6
Gemini 3 Pro Preview	76.6	72.2	65.4
GPT-5 high	68.0	77.6	70.6
Kimi K2-0905	37.0	30.4	25.8
Qwen3-Max-0923	53.6	60.1	52.2
GPT-5.1-chat-latest	28.8	33.0	32.6
Seed-1.6-Lite-1015 high	54.8	55.6	48.6
DeepSeek V3.1 Thinking	71.8	68.6	62.2

Table 12: AMO-Bench raw-pool Avg@5 accuracy (%). These are the reported pre-audit model-level results; the main hardening table uses the separate Verified cohorts.

Model	Seed	H	H Pro
Gemini 2.5 Pro	28.0	43.5	37.6
GLM-4.6	40.0	52.9	48.0
GPT-5.1 high	56.0	64.6	58.8
Claude Sonnet 4.5 Thinking	18.0	36.0	32.4
Seed-1.6-Thinking-0715	40.0	41.6	38.0
DeepSeek V3.2 Thinking	28.0	51.6	46.0
Seed-1.6-1015 high	48.0	43.1	36.4
Kimi K2 Thinking	22.0	26.5	17.2
MiniMax M2	20.0	36.6	28.4
Gemini 3 Pro Preview	56.0	61.1	56.0
GPT-5 high	40.0	58.4	54.4
Kimi K2-0905	8.0	19.9	18.8
Qwen3-Max-0923	14.0	39.2	35.2
GPT-5.1-chat-latest	6.0	20.7	17.6
Seed-1.6-Lite-1015 high	36.0	39.7	36.4
DeepSeek V3.1 Thinking	48.0	52.1	52.8

Table 13: GPQA-Diamond model-level Avg@5 accuracy (%). This extension uses arbitration-based validation, separately from the program-verified mathematical families.

Model	Seed	Variant
Gemini 2.5 Pro	82.1	81.4
GLM-4.6	78.8	80.2
GPT-5.1 high	86.4	84.5
Claude Sonnet 4.5 Thinking	81.0	80.1
Seed-1.6-Thinking-0715	78.4	78.5
DeepSeek V3.2 Thinking	82.1	83.3
Seed-1.6-1015 high	77.4	78.9
Kimi K2 Thinking	79.9	83.4
MiniMax M2	76.7	74.4
Gemini 3 Pro Preview	86.7	85.7
GPT-5 high	81.9	82.1
Kimi K2-0905	73.5	74.2
Qwen3-Max-0923	76.1	76.7
GPT-5.1-chat-latest	69.9	69.1
Seed-1.6-Lite-1015 high	77.6	76.9
DeepSeek V3.1 Thinking	80.0	81.3

K A Separate GSM8K Rewriting Probe

The source study also evaluates one sampled variant for each of 1,319 GSM8K test seeds with two models. This is a single-response probe, separate from the five-response, 16-model comparison in the main text. The task and response units are therefore stated independently.

Table 14: Reported single-response accuracy (%) on 1,319 GSM8K seeds and one sampled variant per seed.

Model	Original	Variant
Gemini 2.5 Pro	97.12	98.56
OpenAI GPT-5	96.66	98.56

The number of items missed by both models falls from 28 to 10, or from 2.12% to 0.76% of the bank. Rewriting thus improves agreement on correct answers as well as each model's accuracy on this set. This complements the AIME result: fresh instances can expose fragility on one source bank and improve consistency on another. The direction of the change is an empirical property of the generated tasks and evaluated solvers.

L Post-Hoc Answer Verification by Structured Debate

L.1 Evaluation Set and Decision Target

The separate verification study examines 195 candidate answers from 58 hard competition-mathematics problems spanning combinatorics, number theory, and algebra. Each problem contributes one hand-checked correct candidate and two or three incorrect candidates, yielding 58 positives and 137 negatives. A verifier classifies each candidate. A complete problem is counted correct only when its correct candidate is accepted and all its incorrect candidates are rejected.

This is a post-hoc candidate-answer experiment, distinct from the specification-level noise filter and the 16-model student benchmark. False acceptances matter because they introduce incorrect labels into an accepted pool; false rejections reduce its usable size. Precision and recall therefore quantify different construction requirements.

L.2 Agents, Exchanges, and Acceptance

Both the single-judge baseline and the debate system use GPT-5.2 at low reasoning effort. The debate system assigns five roles: algebraic and formal consistency, theorem applicability, independent resolution, boundary and optimality checking, and numerical or symbolic cross-checking. The roles share a model while using different verification instructions.

Each agent first restates the claim, lists possible failure modes, performs an independent check, considers the strongest opposing argument, and records a structured assessment. The assessment distinguishes an answer supported by evidence, an answer refuted by evidence, and insufficient reasoning without a refutation. This distinction prevents uncertainty alone from being treated as proof that an answer is wrong.

The configuration uses five agents and up to five rounds. During exchanges, agents receive the vote tally, a digest of unresolved disagreements, scheduler-generated challenges, and directed peer messages. Each round asks for a response to an opposing claim and a new concrete check. Short summaries, capped at 400 characters per prior rationale, carry the previous assessments forward. Unanimous decisions can terminate the exchange after at least two full rounds. Final acceptance requires at least three agents with an explicit supported-answer assessment.

L.3 Observed Precision–Yield Tradeoff

Table 15: Post-hoc answer verification on 195 candidates from 58 problems. Positive means a correct candidate answer. Rates are recomputed from the reported confusion counts. A complete problem requires accepting its correct candidate and rejecting every incorrect candidate.

System/run	Acc.	Prec.	Recall	FP	FN	Complete
Baseline 1	72.8	54.5	51.7	25	28	18/58
Baseline 2	72.8	54.9	48.3	23	30	18/58
Baseline 3	73.8	55.9	56.9	26	25	22/58
Debate 1	87.2	90.2	63.8	4	21	34/58
Debate 2	87.2	92.3	62.1	3	22	34/58
Debate 3	88.7	95.0	65.5	2	20	36/58

Across three independent runs, mean precision rises from 55.1% for the single judge to 92.5% for debate. Mean false acceptances fall from 24.7 to 3.0 per run, while mean false rejections fall from 27.7 to 21.0. Complete-problem classification rises from 19.3 to 34.7 of the 58 problems. The system therefore accepts a substantially cleaner answer pool and recovers more valid candidates, reducing both sources of verification error.

The false-rejection analysis identifies three recurring errors: difficulty checking the hardest solutions, incorrect mathematical objections, and mixed evidence falling short of the acceptance threshold. These cases point to a concrete verification objective: a judge must establish the validity of a difficult argument as well as detect a flawed one. The higher-precision system uses up to 25 calls per candidate, versus one for the single-judge baseline. The three-run comparison measures the resulting system-level precision–cost tradeoff, combining structured exchanges with a larger verification budget.

M Benchmark Renewal in Context

Refreshing instances and extending difficulty. Contamination studies examine the relationship between benchmark exposure and measured performance (Shi et al., 2023; Golchin & Surdeanu, 2024; Sainz et al., 2024). LiveCodeBench and LiveBench renew evaluation material over time (Jain et al., 2025; White et al., 2025). Perturbation-based evaluations ask a complementary question: how well does a solver handle changes to a known problem family? GSM-Plus varies mathematical statements, GSM-Symbolic and VAR-MATH parameterize instances, and GSM-Ranges varies numerical scale (Li et al., 2024; Mirzadeh et al., 2025; Yao et al., 2025; Shrestha et al., 2025). VeRA-E expresses instance relationships through a reusable program. VeRA-H additionally changes the com-

putation, and VeRA-H Pro selects within the resulting candidate space. Freshness and headroom are therefore separate outputs of the same construction interface.

Exposure diagnostics and newly constructed tests. Surveys of benchmark contamination distinguish detection approaches and sources of evaluation leakage (Xu et al., 2024b). PaCoST compares confidence on original and rephrased material (Zhang et al., 2024b); GSM1k examines performance on newly authored grade-school mathematics relative to an established benchmark (Zhang et al., 2024a). These studies motivate constructing related tests whose relationship to the source is inspectable. VeRA makes that relationship explicit in its executable specification: a seed identifies the originating problem, the generator defines valid changes, and the answer program labels each resulting instance. This supports repeated sampling while preserving a concrete link to the computation being evaluated.

Functional benchmarks and procedural generation. Procgen evaluates reinforcement-learning generalization with procedurally generated environments (Cobbe et al., 2019). In mathematics, Putnam-AXIOM combines static and functional evaluation, and RV-Bench introduces questions with unseen random-variable assignments (Gulati et al., 2025; Hong et al., 2026). BeyondBench generates algorithmic tasks with deterministic solutions (Srivastava et al., 2026). These approaches share the advantage that an evaluation family can supply more than one fixed item. VeRA contributes a construction pipeline that starts with natural-language mathematical problems, compiles reusable generators and answer programs, and searches for harder computations. The resulting representation supports both renewal within a family and expansion of its challenge.

Graphs, rewrites, and executable specifications. DyVal and DARG manipulate reasoning graphs to vary task structure and complexity (Zhu et al., 2024; Zhang et al., 2024c). MaSTer uses iterative rewriting and verification to search for robustness failures (Hou et al., 2026). AutoLogi constructs logic puzzles with executable verification (Zhu et al., 2025). VeRA represents a family through a natural-language renderer, a constrained input generator, and an answer program. This interface accommodates arithmetic, conditionals, iteration, and search within the verifier, while the generator determines valid assignments. Graph composition can be implemented inside such a program; a rendered rewrite must remain consistent with its slots and computation. The reusable object therefore joins instance generation, answer computation, and consistency checks in one inspectable family.

Execution-based evaluation of generated programs. Codex and program-synthesis studies evaluate programs through their outputs on tests (Chen et al., 2021; Austin et al., 2021); APPS and TACO extend this setting to diverse algorithmic problems (Hendrycks et al., 2021a; Li et al., 2023). EvalPlus strengthens code evaluation with additional generated tests (Liu et al., 2023). This line of work highlights the value of checking behavior across assignments rather than relying on a single successful example. VeRA uses execution on the benchmark-construction side: each accepted program becomes a reusable source of labels. Its validity depends jointly on the implementation, the allowed input domain, and agreement between the rendered statement and the computation.

Formal verification and executable answer generation. MiniF2F supplies mathematical statements for machine-checked proving, and LeanDojo connects language models to a proof environment and premise retrieval (Zheng et al., 2022; Yang et al., 2023). These systems verify derivations against a formal statement. VeRA instead targets generated question–answer instances: the specification computes an answer, and the validation pipeline checks the generated program and its agreement with the rendered question. The two interfaces supply complementary forms of machine feedback. Their common design principle is to expose an explicit object against which a proposed result can be checked.

Different roles for execution and model judgment. PAL and Program of Thoughts use code execution to produce solutions (Gao et al., 2023; Chen et al., 2023). In VeRA, execution produces reference labels. A generated implementation is checked against seed answers where applicable, sampled assignments, answer perturbations, and independently solved rendered items. Work on verifiers, self-consistency, and multi-agent debate supplies related mechanisms for evaluating candidate reasoning (Cobbe et al., 2021; Lightman et al., 2024; Wang et al., 2023a; Du et al., 2024).

The specification-level filter, item audit, and post-hoc debate study in this paper operate at different points in that chain. Their outcomes respectively describe screening, release validity, and candidate-answer classification.

Generation, testing, and selection. AlphaCode combines large-scale program generation with filtering and selection, while CodeRL uses execution feedback in learning and candidate refinement (Li et al., 2022; Le et al., 2022). STaR learns from successful generated reasoning (Zelikman et al., 2022); DeepSeek-R1 and Absolute Zero use verifiable feedback to improve reasoning, with Absolute Zero also proposing its own code-based tasks (Guo et al., 2025; Zhao et al., 2025). Across these settings, a proposal distribution and a selection rule jointly determine the useful outputs. VeRA applies this structure to benchmark construction. Validation produces the accepted H pool; difficulty-based selection produces H Pro. Their release-level means describe different stages of the same budgeted search process.

Synthetic data as a designed distribution. Self-Instruct and WizardLM generate instruction data through proposals, filtering, and progressive rewriting (Wang et al., 2023b; Xu et al., 2024a). Orca emphasizes explanation traces, and Textbooks Are All You Need studies curated and synthetic educational code data (Mukherjee et al., 2023; Gunasekar et al., 2023). MetaMath bootstraps mathematical questions, while MAmmoTH combines verbal and program-based rationales across mathematical sources (Yu et al., 2024; Yue et al., 2023). These works establish that generation choices shape the learning signal. VeRA applies the corresponding principle to evaluation: a candidate becomes a usable benchmark item through validation, and the selection policy shapes the challenge delivered to evaluators.

Reusable generation requires a defined acceptance rule. Acceptance fixes what counts as a usable item, while difficulty selection determines which usable candidates enter the selected release. Datasheets for Datasets motivates documenting such construction decisions alongside the dataset (Gebru et al., 2021). Here, seed identifiers, sampling indices, validation outcomes, and repair decisions connect each released question to its origin. This preserves the distinction among proposing many candidates, obtaining valid items, and assembling a benchmark with a particular difficulty profile. The executable family provides the reusable unit, and the declared construction budget determines the particular release evaluated in this paper.