VERA: Math Benchmarks as Executable Specifications

Zerui Cheng [1 * †] **Jiashuo Liu** [2 *] **Chunjie Wu** [2] **Jiayang Sun** [3] **Jianzhu Yao** [1]
Pramod Viswanath [1] **Ge Zhang** [2] **Wenhao Huang** [2]

Abstract

Most reasoning benchmarks are *static*: the same problems are reused repeatedly, enabling memorization, format exploitation, and saturation. To measure genuine AI progress we need evaluation that is **robust by construction, not by post-hoc detection**. We propose VERA (**V**erified **R**easoning Data **A**ugmentation), a framework that compiles benchmark problems into *executable specifications* consisting of a natural-language template with slots, a coherent generator that samples valid configurations, and a deterministic verifier that validates assignments and computes canonical answers. Once a specification is validated, labels for any newly sampled instance come from executing the verifier, so correctness amortizes across an unbounded stream of fresh variants at near-zero marginal cost. VERA has two modes. VERA-E, the *equivalent* mode, rewrites problems while keeping the underlying logic intact and supports memorization and robustness diagnostics. VERA-H and VERA-H PRO, the *hardened* modes, systematically increase complexity while remaining verifiable. Evaluating 16 frontier models on GSM8K, AIME-2024 and 2025, Beyond-AIME, AMO-Bench, and GPQA-Diamond shows that VERA-E exposes robustness and familiarity effects, with 4× dispersion on GSM8K and a Teacher-robust, statistically significant AIME-2024 versus 2025 gap, while VERA-H and VERA-H PRO generate fresh verified tasks that restore headroom on saturated benchmarks. Beyond the automatic pipeline, we release human-audited *Verified* datasets, comprising 220 of 220 VERA-E items and 2,187 of 2,299 expert-audited VERA-H and VERA-H PRO items at 95.13% retention, together with a stricter pair-preserving *Verified Full* release. All code and datasets are made open-source at https://github.com/Marco-Cheng/VeRA.

1. Introduction

Recent progress in large language models (LLMs) has been tightly coupled with progress in evaluation. Standardized benchmarks provide a shared yardstick for comparing models, diagnosing failure modes, and guiding research directions; reasoning benchmarks in particular have served as a clean and consequential testbed, demanding multi-step computation, compositional generalization, and faithful execution of instructions.

1.1. Why static benchmarks are failing

Benchmarks like GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), and AIME were instrumental in driving the advances of the last few years in language models, but they share a structural limitation that increasingly constrains their usefulness: *they are fixed collections of problems that get reused indefinitely*. The same GSM8K dataset of 7,473 training and 1,319 test problems, and the same 30 AIME problems per year, have been evaluated, finetuned on, and discussed repeatedly. Three failure modes follow. First, on **memorization vs. reasoning**, static items can leak into training data via web-scale pretraining or repeated public use, inflating scores hard to separate from genuine reasoning ability (Sainz et al., 2023; Xu et al., 2024; Cheng et al., 2025; Zhang et al., 2024a). Second, on **saturation vs. discrimination**, once models reach near-ceiling accuracy a benchmark becomes a checkbox: score gaps are dominated by variance and cannot reliably distinguish frontier systems. Third, on **surface-form artifacts**, fixed benchmarks come with fixed conventions of phrasing, answer formats, and parameter regimes that models can exploit, and even mild surface perturbations can reveal brittleness on supposedly "solved" benchmarks (Mirzadeh et al., 2024; Li et al., 2024). The upshot is that static scores conflate a model's reasoning ability with its familiarity with the benchmark. A natural

* Equal contribution. † Work done during internship at ByteDance. [1]Princeton University [2]ByteDance [3]Hong Kong University. Correspondence to: Zerui Cheng <zerui.cheng@princeton.edu>, Jiashuo Liu <liujiashuo77@gmail.com>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

response is to author fresh problems, but high-quality reasoning tasks are expensive to create and to grade, and expert annotators do not scale at the frontier. This tension between freshness and label reliability is a central bottleneck for evaluation.

1.2. The key shift: benchmarks as executable specifications

To overcome these limitations, we do not want to create more static benchmarks. Instead, we define a benchmark as a *specification* of the underlying task, so evaluation can draw on fresh instances guaranteed to be well-formed and correctly labeled. *A benchmark should be an executable specification that generates families of verified instances, not a static list of questions.* In this paper we propose VERA (**V**erified **R**easoning Data **A**ugmentation), a framework that compiles benchmark items into *executable specifications*. Each specification contains (i) a natural-language template with placeholder slots, (ii) deterministic generator code that samples valid slot assignments, and (iii) deterministic verifier code that checks validity and computes the corresponding gold answer.

With a validated specification, producing fresh instances with trusted labels requires no further LLM calls: we sample assignments locally, render the question, and compute the label by executing the verifier. This makes correctness a property of the program and enables *correctness amortization*: validate the verifier once, and correctness generalizes to an unbounded stream of generated instances. The generation pipeline is automatic in that it requires no per-instance human annotation, and the released evaluation subsets layer a light-weight expert audit on top of the automatic pipeline (§3.2).

1.3. Contributions and findings

VERA operates in two complementary modes. VERA-E, the *equivalent* mode, generates meaning-preserving rewrites that keep verifier semantics fixed and supports robustness diagnostics and exposure detection. VERA-H, the *hardened* mode, modifies the task to increase reasoning demand while remaining verifiable, and supports renewable boundary evaluation. VERA-H PRO is a paired protocol that selects, per seed, the *judge-ranked hardest verified* variant from a fixed candidate pool. It improves per-seed comparability while keeping labels grounded in verifier execution; case studies appear in Appendix D.

Verified benchmarks as renewable artifacts. We formalize benchmarks as executable specifications consisting of a template, a generator, and a verifier. These specifications can be sampled *post-training* to produce unlimited fresh instances with programmatically certified labels. **VERA-E: verified-equivalent resampling for evaluation quality and diagnostics.** On GSM8K, verified rewrites roughly quadruple inter-model dispersion and reduce a conservative artifact proxy from 2.12% to 0.76%. On AIME, year-controlled diagnostics in Section 3 reveal systematically larger seed-to-variant drops on older splits than on newer ones. The pattern is statistically significant, robust across independently synthesized Teacher LLMs, and consistent with benchmark familiarity effects. **VERA-H and VERA-H PRO: scalable generation of hard tasks with certified labels.** By validating verifier-generated answers with a conservative spec-level filter (Section 2.4), VERA-H produces challenging, fresh instances on which frontier models achieve only around 50% Avg@5 in several settings, restoring headroom on saturated benchmarks. The same setup also lets us measure the *proposer–solver gap*: whether a proposer, routed through VERA, can generate verifiable tasks beyond its own current solving frontier. **Human-audited *Verified* datasets.** On top of the automatic pipeline we release *Verified* subsets, comprising 220 of 220 audited VERA-E items on AIME-2024, AIME-2025, and Beyond-AIME and 2,187 of 2,299 expert-audited VERA-H and VERA-H PRO items, together with a pair-preserving *Verified Full* release (Section 3.2). All specifications, code, raw automatic outputs, and human-audited Verified subsets are released to support reproducible renewable evaluation.

Roadmap. Section 2 introduces the specification view, synthesis, and validation; Section 3 evaluates VERA-E and VERA-H on frontier models and presents human-audit evidence; Section 4 discusses related work; Section 5 concludes.

2. The VERA Pipeline: Executable Specifications and Synthesis

At its core, a VERA benchmark is not a fixed question set but rather a collection of *executable specifications* capable of generating verified instances on demand. We first formalize the specification view in §2.1 and §2.2. We then describe the executable specification format in §2.3, the three generation modes in §2.3, validity criteria in §2.4, and the propose–execute–validate synthesis loop in §2.4.

2.1. Problem formulation: from seeds to verified instance distributions

Conventional reasoning benchmarks come as finite datasets $\mathcal{D} = \{(q_i, a_i)\}_{i=1}^{N}$. A seed item (q, a) captures human intent and difficulty calibration, but as an evaluation artifact it degrades with repeated exposure. VERA addresses this by compiling each seed into a *task family* $\mathcal{F}$, which is a renewable generator that produces fresh instances with trusted labels. We learn a mapping $\mathcal{M} : (q, a) \mapsto \mathcal{F}$, where $\mathcal{F}$ can cheaply sample unlimited verified variants without GPU or

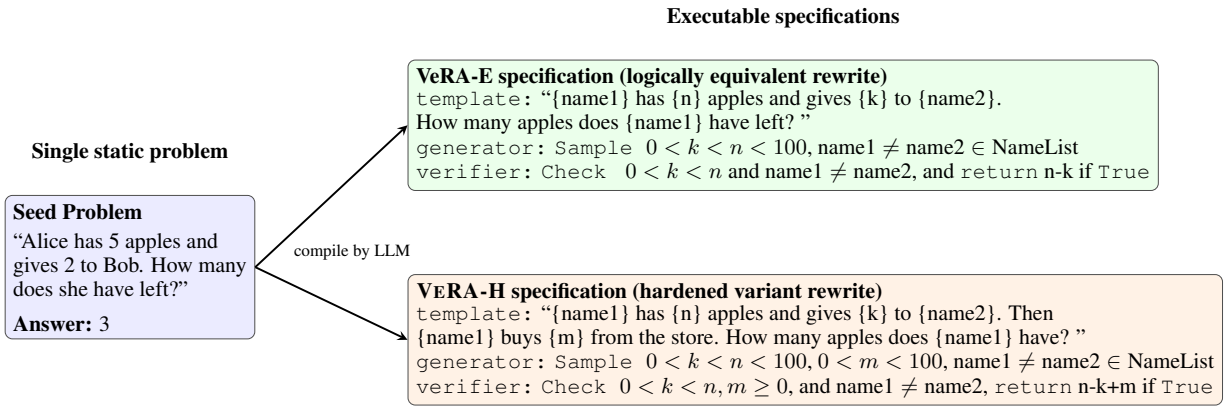


Figure 1. **VERA represents a benchmark as executable specifications.** Each specification contains (i) a natural-language template, (ii) *generator code* that samples valid slot assignments, and (iii) *verifier code* that deterministically checks validity and computes the label. Sampling the specification is GPU/LLM-free and yields fresh instances with labels certified by executing the verifier. VERA-E preserves the original task while changing surface form; VERA-H defines a hardened task with an updated verifier.

LLM inference at generation time.

Task family interface. Each task family $\mathcal{F}$ is an executable object consisting of three components. The **template** $T(\cdot)$ is a natural-language wrapper with slots for variable content. The **generator** $G(\cdot)$ is deterministic sampling code that produces valid slot assignments. The **verifier** $V(\cdot)$ is deterministic code that checks whether an assignment is valid and computes the canonical answer. To generate an instance, we sample $\hat{\theta} \leftarrow G$, render $x = T(\hat{\theta})$, and run $(\texttt{valid}, y) = V(\hat{\theta})$, accepting only when `valid`. Correctness is no longer defined by human or model agreement; it follows from executable semantics.

Design requirements. We impose five requirements that, taken together, make task families suitable for frontier evaluation. **(R1) Correctness by construction.** Labels come from the verifier, not from LLM voting or heuristic rules. **(R2) Coherent sampling.** The generator should sample from the valid manifold with a low rejection rate so generation remains efficient even under tight constraints. **(R3) Contamination resistance.** Since evaluation draws from freshly sampled instances, memorizing a finite training set provides no advantage. **(R4) Scalability.** Beyond a one-time *Teacher model* call per seed, generating additional instances costs almost nothing. **(R5) Reproducibility.** Sampling is deterministic given a seed identifier and sampling index, yielding stable benchmark artifacts.

A generic view: each seed induces a distribution of verified instances. Once compiled, each family $\mathcal{F}_i$ induces a distribution over verified instances. Writing θ for the slot assignment drawn by the generator, a benchmark item becomes a stochastic yet reproducible process: $\theta \sim P_i(\theta) \Rightarrow x = T_i(\theta), y = V_i(\theta)$. By varying the deterministic RNG seed, we obtain an unbounded stream of fresh instances whose labels are certified by construction. This mechanism underpins contamination-resistant evaluation (R3) and scalable benchmark renewal (R4 and R5). For analyses bounding how far a sampled variant drifts from the seed, e.g. for gradual robustness studies, we include a perturbation score and budgeted sampling option in Appendix E.

2.2. Two modes: Equivalent families and Hardened families

Starting from a single seed, VERA produces two flavors of verified families, each serving different evaluation goals; see Appendix E, Figure 4.

VERA-E: verified equivalent families. VERA-E generates *equivalent variants*: problems that pose the *same underlying question* as the seed but in a different surface form. Across variants we may alter numerical values, entity names, narrative framing, and even language, e.g. English versus Spanish, while leaving the required reasoning unchanged. The purpose is robustness diagnostics. VERA-E reveals whether a model's performance holds up under meaning-preserving perturbations, and exposes score inflation that stems from benchmark familiarity rather than genuine reasoning ability.

VERA-H and VERA-H PRO: verified hardened families. VERA-H generates *hardened variants*: problems deliberately more challenging than the seed, yet still automatically checkable. Hardening raises the reasoning bar by adding constraints, deepening dependency chains, or composing sub-steps, so that the variant demands strictly more work than the seed. VERA-H PRO adds a pairing step: for each seed we generate multiple hardened candidates and select a single representative, the *judge-ranked hardest* among candidates that already passed verifier valida-

tion. This produces a per-seed "hardness delta" comparable across models. Selection ranks among already-verified candidates rather than over the space of all possible hardenings, so ranking noise affects which candidate is chosen but never the correctness of its label.

Why VERA verification enables challenging task generation at scale. Hard tasks are notoriously expensive to label, and human grading does not scale at the frontier. VERA-H sidesteps this bottleneck through *correctness amortization*: once the verifier itself is validated, label correctness becomes a property of the *program*, not of any individual instance. VERA-H can therefore generate diverse hardened instances and certify every label by executing the same program; difficulty and scale grow with computation rather than with repeated re-validation. Put differently, the one-time investment is establishing a correct verifier, and after that scaling the number and diversity of hard, correctly-labeled tasks costs little more than local compute.

The scientific question behind VERA-H. VERA-H lets us pose a question that static benchmarks cannot: whether a proposer, routed through VERA, can generate *verifiable* tasks beyond its own current solving frontier. We call this the *proposer–solver gap*; a positive gap is the key precondition for self-hardening, in which today's model produces tomorrow's verifiably correct training curriculum. Where the proposer is itself the solver bottleneck, as on AMO-Bench in §3.2, hardening becomes non-monotonic, informative about the proposer rather than the framework.

2.3. End-to-end pipeline overview

We now introduce the VERA pipeline: the executable specification format, how the three generation modes VERA-E, VERA-H, and VERA-H PRO fit into a unified interface, and the automatic propose–execute–validate synthesis loop that compiles seeds into *validated* specifications while making failures actionable.

Roles and terminology. Three models play distinct roles. The *Teacher* M_T is a frontier LLM that proposes candidate executable specifications, namely the template, the generator, and the verifier. The *Judge* M_J serves *only* as a conservative validity filter at the specification level for hardened families, and optionally ranks verified candidates in VERA-H PRO. The *Student models* M_S are the models under evaluation. Crucially, M_T and M_J are proposal and filter mechanisms, not label oracles: once a specification passes validation, sampling and labeling require no LLM inference, and labels are produced deterministically by executing the verifier. We use GPT-5 as the default Teacher, and report cross-Teacher robustness with Gemini 3 Pro and Claude 4.6 Opus in §3.1; full details appear in §3 and Appendix K.

Executable specification format. A VERA specification $\mathcal{S}$ is an executable object with five fields. **(1) Slots** $\theta = (\theta_1, \ldots, \theta_k)$ are typed variables with constraints, where types include integer, rational, and categorical, with range bounds and structural constraints. **(2) Template** $T(\theta)$ is a natural-language wrapper with named placeholders. **(3) Generator** $G(\text{rng})$ is deterministic sampling code returning $\hat{\theta}$. **(4) Verifier** $V(\theta)$ is deterministic code returning $(\texttt{valid}, y)$. **(5) Hardness rationale** is an optional brief explanation used for hardened families and auditing. Given a deterministic RNG seed, $\mathcal{S}$ produces an instance via $\hat{\theta} \leftarrow G(\text{rng})$, $x \leftarrow T(\hat{\theta})$, $(\texttt{valid}, y) \leftarrow V(\hat{\theta})$. An instance is accepted iff $\texttt{valid}$, and the label is y from executing V (R1). For deterministic reproducibility (R5), we seed $\texttt{rng}$ using a cryptographic hash of immutable identifiers, $\texttt{seed_val} = \texttt{Hash}(\texttt{seed_id}, \texttt{generator_id}, \texttt{sample_idx})$, so the same identifiers always yield the same sampled instance.

Generation modes (VERA-E, VERA-H, VERA-H PRO). All three modes ultimately produce an instance by executing an executable specification, but they differ in *what stays fixed across variants* and *what requires validation*. **VERA-E (equivalent families: fixed semantics):** VERA-E compiles a seed into an *equivalent specification* $\mathcal{S}_E$ that preserves the seed task. Once $\mathcal{S}_E$ passes the validity checks of §2.4, it defines a renewable family: we draw $\hat{\theta}$, render $x = T_E(\hat{\theta})$, and compute $y = V_E(\hat{\theta})$. Because the verifier semantics are fixed, all accepted variants are labeled by the same validated verifier, so any performance difference reflects sensitivity to surface form rather than changes in the underlying task. **VERA-H (hardened families: transformed semantics):** VERA-H compiles the same seed into *hardening specifications* $\mathcal{S}_H$ that intentionally alter the task while keeping it verifiable. Hardening operates at the specification level: the Teacher may introduce new slots, new constraints, and new solution structure, and must supply a corresponding verifier for the modified task. Different Teacher attempts can therefore yield different hardened specifications, and hence different verifiers. There is no single canonical hardened semantics per seed. VERA requires only that each accepted $\mathcal{S}_H$ pass the same validity suite and define a coherent executable family whose rendered instances are correctly labeled by its own verifier (§2.4). **VERA-H PRO (paired hardest selection: selection after validation):** VERA-H PRO strengthens per-seed comparability by selecting among verified hardened candidates. For each seed we generate K candidates with default K=5, possibly drawn from different hardened specifications, and validate each one. A fixed *judge-based* ranking rule then selects one $h^\star$ as the paired "judge-ranked hardest verified" variant. Selection happens *after* candidates are verified and

labeled by deterministic execution; it never defines labels and cannot override verifier correctness.

2.4. Specification validity and synthesis loop

In §2.2 we discussed *amortized correctness*: once a specification is validated, its verifier generalizes to newly sampled instances from that specification. Each specification is proposed by the Teacher and is never assumed correct by default, so VERA validates specifications before amortizing correctness. The validity criteria differ by mode. VERA-E is anchored to a seed answer, whereas VERA-H and VERA-H PRO define new tasks without an external gold label. *Common gates* apply in all modes: every candidate must (i) parse under the schema, (ii) compile in the sandbox, (iii) execute without runtime errors on a small batch of sampled assignments, and (iv) achieve non-degenerate yield, in the sense that the generator samples valid assignments without pathological rejection.

Equivalence validity (VERA-E): seed-anchored consistency. For VERA-E, correctness reduces to "same underlying task as the seed." We extract a canonical seed assignment θ_{seed}, namely the seed's parameter values under the slot schema. The Teacher reports this canonical base assignment at compile time (see Appendix P), and we require $V_E(\theta_{\text{seed}}) = (\texttt{True}, a)$. Specifications failing this check are rejected as non-equivalent. Once it passes, every subsequent VERA-E instance inherits correctness by executing the same validated verifier. Seed-anchored consistency is a single-point property and does not by itself certify program equivalence across the entire slot manifold; for the released VERA-E evaluation subsets we additionally provide direct human verification and cross-Teacher consistency (§3.1).

Hardening validity (VERA-H and VERA-H PRO): judge-based noise discrimination. For hardened tasks there is no seed label to match, and the main risk is a misaligned verifier that silently propagates errors. We therefore validate $\mathcal{S}_H$ with a *noise-discrimination check*. For each sampled instance with verifier output y, the Judge model decides correctness across $k_{\text{correct}}{=}2$ true-answer trials and $k_{\text{noise}}{=}3$ controlled-perturbation trials, and an instance passes if the Judge is right in $\geq\tau{=}4$ of 5 trials. Labels remain defined solely by executing V_H, and the Judge is only a spec-level filter. We use $\tau{=}4$ as a conservative default calibrated to bound false positives, with 1/50 false positives versus 3/50 false negatives in a human audit; full calibration appears in Appendix E. Threshold calibration alone is not a proof of verifier correctness, and the released *Verified* subsets layer expert audit on top (§3.2). For applications requiring higher-precision post-hoc answer checking, Appendix F describes a complementary multi-agent debate pipeline.

Automatic synthesis loop and implementation. We compile seeds into validated specifications via an iterative propose–execute–validate loop. The Teacher proposes a triple of template, generator, and verifier, which we execute in a sandbox to catch runtime errors, test generator yield, and run mode-specific validity checks: seed-anchored consistency for VERA-E, and the noise-discrimination filter for VERA-H and VERA-H PRO. On failure we return structured feedback that includes violated constraints, mismatched seed assignments, and failing judge trials, and we request a revised specification. LLM-generated code runs in a restricted subprocess sandbox with resource limits and no network access. Synthesis is a one-time cost linear in the number of seeds, and generating fresh instances and labels post-validation is CPU-only and essentially free. The full pipeline diagram and pseudocode appear in Figure 5 and Algorithm 1, which makes the VERA-E seed-consistency check explicit alongside the VERA-H judge path; implementation details appear in Appendices K and E.

3. Experiments

We evaluate VERA on frontier models and widely used reasoning benchmarks. We focus on three questions. First, does VERA-E expose robustness and familiarity effects under *verified equivalence* stably across Teacher LLMs and not driven by differential perturbation drift? Second, can VERA-H and VERA-H PRO generate fresh, verifiable instances that restore headroom on saturated benchmarks? Third, do these effects hold under release-scale expert audit on the *Verified* datasets that ship with this paper, as discussed in §3.2 and Table 3?

Setup. We evaluate 16 frontier models spanning proprietary and open-weight systems, with the full list in Appendix U. We report Avg@5 accuracy, taking 5 independent samples per problem in the spirit of self-consistency decoding (Wang et al., 2023a), and where relevant we report rank stability across models. The default Teacher and Judge are GPT-5 (high). As a robustness check we additionally synthesize VERA-E under Gemini 3 Pro and Claude 4.6 Opus as Teachers (§3.1). Student prompts use zero-shot chain-of-thought (Wei et al., 2022) with boxed-answer extraction; decoding details in Appendix K. Our evaluation covers GSM8K, GPQA-Diamond under arbitration-based verification (Appendix N), AIME 2024 and 2025, Beyond-AIME (ByteDance Seed Team, 2025), and AMO-Bench (An et al., 2025). Seeds are the original benchmark items, and verified variants are sampled from validated specifications. For VERA-E we generate 2 variants per seed on AIME, so 30 problems become 60 per year, and we sample larger pools for GSM8K and GPQA to probe dispersion. For VERA-H we generate multiple hardened candidates per seed and report both the raw VERA-H pool and the paired

Table 1. **Main results in Avg@5 accuracy (%), averaged across 16 frontier models.** Numbers in parentheses are problem counts. VERA-E produces *verified-equivalent* variants, while VERA-H and VERA-H PRO produce *verified-hardened* variants. VERA-H PRO is a paired protocol that selects **exactly one** hardest verified variant per seed from a fixed candidate pool. **For VERA-H and VERA-H PRO the table reports numbers on the human-audited *Verified* releases of §3.2**, and the variant counts are VERA-H and VERA-H PRO Verified counts after audit. GPQA-Diamond is evaluated under *arbitration-based* verification rather than deterministic program verification; see Appendix N, marked †.

Benchmark	Seeds	Verified variants	Avg@5: Seeds → Variants
VeRA-E (equivalent families): robustness and familiarity diagnostics			
GSM8K	(1319)	(2638)	94.85 → 95.20
AIME-2024	(30)	(60)	84.46 → 70.25
AIME-2025	(30)	(60)	79.21 → 72.13
GPQA-Diamond†	(198)	(871)	79.27 → 79.42
Beyond-AIME	(100)	(100)	58.34 → 57.30
VeRA-H and VeRA-H Pro *Verified* (hardened families): renewable boundary evaluation			
AIME-2024-II	(14)	(69 / 14)	84.91 → 68.71 / 58.57
Beyond-AIME	(100)	(466 / 93)	58.34 → 59.45 / 55.02
AMO-Bench	(50)	(217 / 44)	31.75 → 44.56 / 40.16

VERA-H PRO protocol. All specifications are produced by a Teacher model and filtered through deterministic compilation and execution checks together with mode-specific validity gates (§2.4). For hardened families we apply a conservative spec-level judge filter, with calibration in Appendix E. **For VERA-H and VERA-H PRO, the numbers in Table 1 are reported on the human-audited *Verified* releases of §3.2**, i.e. the recommended evaluation-ready artifacts. The corresponding raw-pipeline means, per-benchmark audit retention, and a side-by-side comparison appear in Table 3. Complete model-by-model tables appear in Appendix U, and per-model breakdowns and additional diagnostics in Appendix G.

3.1. VERA-E: verified-equivalent augmentation

Before presenting results, we emphasize what makes seed→VERA-E gaps interpretable. Because variants are accepted only after passing seed-anchored verifier checks (§2.4), any performance difference cannot be attributed to relabeling errors. The verifier semantics are identical by construction, so when performance shifts it must reflect sensitivity to meaning-preserving variation in surface form, parameterization, or language.

Saturation can mask brittleness. GSM8K provides a clear illustration. The model-suite mean barely moves, from 94.85 to 95.20 in Table 1, which on the surface appears reassuring. However, dispersion across models increases sharply under VERA-E, with the standard deviation rising from 0.6 to 2.3 in Table 2. This is consistent with a "solved" benchmark that masks heterogeneous brittleness: some models remain stable while others drop substantially under verified rewrites. The aggregate mean obscures considerable variation in robustness.

Familiarity effects on AIME, with two robustness controls. AIME-2024 and AIME-2025 share the same format and pipeline, making them a natural controlled comparison. Across 16 models we observe a mean seed→VERA-E drop of -14.2 on AIME-2024 versus -7.1 on AIME-2025 (Appendix G). A paired t-test on per-model drop differences yields $t_{15}=3.82$, $p=0.0017$, confirming the gap is not due to chance. The year-controlled setup makes it hard to attribute the gap to format or pipeline differences, and a more plausible interpretation is benchmark-specific familiarity: models likely encountered AIME-2024 problems more frequently during training and are consequently more brittle when reformulated. Two alternative explanations could in principle produce this asymmetry without invoking familiarity, namely Teacher-specific bias and differential perturbation drift, and we rule both out. *Cross-Teacher.* Independently synthesized VERA-E families using Gemini 3 Pro and Claude 4.6 Opus as Teachers preserve the same pattern, with AIME-2024 and AIME-2025 drops of -12.8 and -6.9 under Gemini and -15.1 and -6.3 under Claude. *Perturbation drift.* Per-year perturbation-score distributions are statistically indistinguishable, with KS $p=0.51$ (Appendix E.1).

Manual verification of released VERA-E items. Because the seed-anchored check is a single-point guarantee (§2.4), we additionally audited all 220 items in the released VERA-E evaluation subsets, comprising 60 of 60 on AIME-2024, 60 of 60 on AIME-2025, and 100 of 100 on Beyond-AIME, with STEM-PhD annotators independently solving each item and cross-checking against the verifier output. All 220 items are confirmed semantically equivalent with correct labels (Appendix A).

Stable benchmarks stay stable; reduced artifact exposure. Not every benchmark exhibits these effects, and this is

Table 2. **VERA-E diagnostic statistics.** For each benchmark we report the mean and standard deviation of Avg@5 (%) across 16 models, together with the Spearman rank correlation ρ between seed and VERA-E scores. Higher std on VERA-E indicates increased discriminability in saturated regimes, and lower ρ indicates rank reshuffling between seeds and verified-equivalent variants.

Benchmark	#Seeds	#VERA-E	Mean Seed	Mean VERA-E	Std Seed	Std VERA-E	Spearman ρ
GSM8K	1319	2638	94.9	95.2	0.6	2.3	0.712
AIME-2024	30	60	84.5	70.2	11.2	9.8	0.813
AIME-2025	30	60	79.2	72.1	12.7	13.7	0.884
GPQA-Diamond	198	871	79.3	79.4	4.2	4.2	0.927
Beyond-AIME	100	100	58.3	57.3	12.4	12.2	0.871

itself informative. On GPQA-Diamond, seed and VERA-E means and dispersion are nearly unchanged (Tables 1 and 2), suggesting that when a benchmark has not saturated and contamination is less of a concern, verified equivalence behaves as expected. GPQA is evaluated under arbitration-based verification (Appendix N) and marked †. On GSM8K, VERA-E rewrites also reduce a conservative artifact proxy from 2.12% to 0.76% (Appendix T), so the variants are substantially less likely to match surface patterns tied to the original split. The "both-wrong" statistic is a heuristic, and the more decisive GSM8K signal is the roughly fourfold dispersion increase above.

3.2. VERA-H and VERA-H PRO: verified hardening for renewable evaluation

The motivation behind hardening is straightforward: when models have largely saturated a benchmark, harder verified variants are needed to restore meaningful separation between models. We now examine whether this approach works in practice.

Restoring headroom on saturated AIME. Consider the 14-problem AIME-2024-II subset. The model-suite mean is 84.91 on seeds, which is strong performance but leaves almost no room to distinguish capable models. On the human-audited *Verified* release, VERA-H drops the mean to 68.71 and the paired VERA-H PRO protocol drops it further to 58.57 (Table 1). Because hardening is verifier-certified and labels come from executing the verifier, this recovered headroom does not require human grading for each instance; expert audit is applied once per released item (§3.2) to turn the automatic pipeline into evaluation-ready artifacts.

Expert audit at release scale. Judge-threshold calibration bounds false positives in a small sample but does not by itself certify an end-to-end benchmark release. We therefore layer a one-time expert-audit step on top of the automatic pipeline, applied *once per released item* rather than per evaluation instance. STEM-PhD annotators independently solved and audited 2,299 automatically generated VERA-H and VERA-H PRO math items, of which **2,187 (95.13%) passed audit**: 2,036 of 2,135 VERA-H items at 95.4%, and 151 of 164 VERA-H PRO items at 92.1%. Retention remains high even on the hardest benchmark in our suite, with 88.93% on VERA-H AMO and 88.00% on VERA-H PRO AMO. The audited VERA-H *Verified* and VERA-H PRO *Verified* releases are the artifacts reported in Table 1, and Table 3 additionally lists the raw-pipeline means and retention for transparency. For strict per-seed pairing, we also release *Verified Full* variants in which any rejected paired candidate is regenerated until audit passes (Appendix A).

Different benchmarks, different effects. Hardening does not produce uniform effects, and examining the differences is instructive. On Beyond-AIME *Verified*, the VERA-H pool is very close to seeds on average, moving from 58.34 to 59.45, while VERA-H PRO induces a clearer difficulty increase from 58.34 to 55.02. The paired protocol, which selects the single hardest verified variant per seed, is more effective at consistently raising difficulty than the raw pool. AMO-Bench presents a different pattern: even on the audited *Verified* release, both VERA-H (31.75 to 44.56) and VERA-H PRO (31.75 to 40.16) *increase* accuracy relative to seeds (Table 1). We hypothesize this is an artifact of the verification process and represents a "ceiling" of our approach. Because VERA relies on a solver–verifier loop to confirm validity, extremely difficult seed problems that models fail to solve, or that generate intractable search spaces, are frequently rejected during generation. The set of successfully verified VERA-H instances therefore selects for problems that are solvable, inadvertently filtering out the hardest instances from the original distribution. This "survivorship bias" makes the hardened subset effectively easier than the noisy, unfiltered seed set, and expert auditing, which removes items with ambiguous verifiers, only sharpens this survivorship effect. Under the self-hardening framing of §2.2, this places the proposer-over-solver gap on a continuum: meaningful on AIME-2024-II and Beyond-AIME Pro, but bounded by proposer quality. Even on AMO-Bench, VERA still produces brand-new meaningful instances that frontier models struggle to solve, at around 40% Avg@5. Full model-level tables and additional diagnostics are in Appendices U and G.

Table 3. **Provenance of VERA-H and VERA-H PRO numbers.** The headline VERA-H and VERA-H PRO numbers in Table 1 are reported on the human-audited *Verified* releases. For transparency we also list the corresponding raw-pipeline numbers and the per-benchmark audit retention. The two columns give essentially the same qualitative conclusions: AIME-2024-II is substantially harder than seeds, on Beyond-AIME VERA-H PRO is harder than seeds while VERA-H is close, and AMO-Bench is a non-monotonic boundary case.

Benchmark	**Raw automatic pipeline**			**Human-audited *Verified* (Table 1)**			**Retention**
	Seed	VERA-H	VERA-H PRO	Seed	VERA-H	VERA-H PRO	
AIME-2024-II	84.91	67.80	58.57	84.91	68.71	58.57	H: 69/70, Pro: 14/14
Beyond-AIME	58.34	58.78	53.98	58.34	59.45	55.02	H: 466/500, Pro: 93/100
AMO-Bench	31.75	42.97	38.38	31.75	44.56	40.16	H: 217/244, Pro: 44/50
Totals							**2,187/2,299 (95.13%)**

4. Related Work

VERA synthesizes benchmark integrity, dynamic evaluation, programmatic verification, and synthetic data into a single seed-anchored executable specification. A full comparison is in Appendix I, Table 12; we summarize the closest threads here.

Contamination, freshness, and dynamic evaluation. Test items leak via direct inclusion, near-duplicates, and online exposure (Sainz et al., 2023; Xu et al., 2024; Zheng et al., 2025; Cheng et al., 2025). Post-hoc detectors—substring/n-gram matching (Brown et al., 2020), membership inference (Shi et al., 2023), and behavior-based leakage (Golchin & Surdeanu, 2023; Zhang et al., 2024b)—and temporal filtering (Jain et al., 2024; Zheng et al., 2025; White et al., 2024) remain reactive or tied to finite human-authored pools. VERA makes freshness intrinsic: seeds compile to executable specifications, and new instances are sampled post-training. Procedural generation (Cobbe et al., 2020) probes robustness; GSM-Symbolic (Mirzadeh et al., 2024), GSM-Plus (Li et al., 2024), and Putnam-AXIOM (Gulati et al., 2025) all perturb reasoning items, while recent concurrent efforts RV-BENCH (Hong et al., 2025) and Beyond-Bench (Srivastava et al., 2025) also generate variants with deterministic verification.

Closer comparison with DyVal. DyVal (Zhu et al., 2023) samples tasks over structured objects (e.g., DAGs) and computes labels by traversal, achieving freshness and programmatic correctness. VERA differs on three axes. (i) Seed-anchored vs. de-novo. DyVal builds tasks from scratch over handcrafted object families. VERA instead compiles each existing reasoning benchmark item into a reusable (T, G, V) triple, preserving human-calibrated intent and difficulty (AIME, Beyond-AIME, AMO-Bench, GPQA) and direct comparability to static benchmarks. (ii) Human involvement. DyVal requires hand-designed semantics, templates, and traversal rules per task type; new domains demand new handcrafted classes. VERA is fully synthetic and human-free in generation: a Teacher LLM proposes the entire triple, and validation and labeling are automatic (§2.4). Humans only audit and verify outputs, enabling two minimal-effort modes. (a) Training data is entirely human-free, with verifier-certified instances sampled at scale. (b) Evaluation data combines automatic generation with a one-time light expert audit—220/220 VERA-E and 2,187/2,299 VERA-H/VERA-H PRO items pass, without per-instance grading (§3.2). (iii) Framework scope. DyVal offers a single evaluation mode. VERA provides two complementary modes—equivalence-preserving VERA-E and verifier-certified VERA-H/VERA-H PRO with a paired Pro protocol—and deterministic verification prevents the shallow, problem-breaking variation typical of naive perturbation pipelines.

Programmatic verification, synthetic data, and label noise. Programs in reasoning ordinarily serve as inference-time aids, e.g., PAL (Gao et al., 2023) and Program-of-Thoughts (Chen et al., 2023). VERA instead uses them at benchmark construction time. RLVR (Guo et al., 2025) showed programmatic correctness checks yield cleaner training rewards, and HardTests (He et al., 2025) synthesizes test cases for code evaluation. VERA applies the execute-to-verify principle to general reasoning and uses the infrastructure to measure the proposer–solver gap, a prerequisite for self-evolution loops. Purely LLM-generated QA pairs risk compounding label noise when one family produces both questions and answers (Wang et al., 2023b; Gunasekar et al., 2023; Liu et al., 2024). VERA separates responsibilities: LLMs produce natural language, deterministic execution supplies labels, scaling without sacrificing integrity. A fuller discussion is in Appendix I.1.

5. Conclusion

Static reasoning benchmarks face saturation, contamination, and surface-form artifacts. VERA reframes them as executable specifications that generate verified instances on demand—VERA-E preserves equivalence, while VERA-H and VERA-H PRO produce harder verified variants. Correctness is grounded in deterministic verification, and renewal is built into the artifact. The fully synthetic pipeline

yields human-free training data; for evaluation, a light one-time expert audit turns the same process into the released Verified and pair-preserving Verified Full artifacts. All code and datasets are at: Link to GitHub Repository.

Impact Statement

VERA improves reasoning-benchmark integrity by addressing saturation and contamination. More reliable evaluation aids model selection, and diagnosing familiarity effects may encourage more robust models. The framework targets evaluation, not capability enhancement; we foresee no negative consequences. Since specifications could be reused as training data, we document a sealed-evaluation protocol in Appendix S.

References

An, S., Cai, X., Cao, X., Li, X., Lin, Y., Liu, J., Lv, X., Ma, D., Wang, X., Wang, Z., and Zhou, S. AMO-Bench: Large language models still struggle in high school math competitions. *arXiv preprint arXiv:2510.26768*, 2025.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

ByteDance Seed Team. BeyondAIME: A challenging math benchmark for llms. https://huggingface.co/datasets/ByteDance-Seed/BeyondAIME, 2025.

Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023.

Cheng, Z., Wohnig, S., Gupta, R., Alam, S., Abdullahi, T., Ribeiro, J. A., Nielsen-Garcia, C., Mir, S., Li, S., Orender, J., et al. Benchmarking is broken–don't let ai be its own judge. *arXiv preprint arXiv:2510.07575*, 2025.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. *International Conference on Machine Learning*, 2020.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. arXiv preprint arXiv:2305.14325.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. PAL: Program-aided language models. *International Conference on Machine Learning*, 2023.

Gebru, T., Morgenstern, J., Vecchione, B., Vaughan, J. W., Wallach, H., Daumé III, H., and Crawford, K. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

Golchin, S. and Surdeanu, M. Time travel in LLMs: Tracing data contamination in large language models. *arXiv preprint arXiv:2308.08493*, 2023.

Gulati, A., Miranda, B., Chen, E., Xia, E., Fronsdal, K., Dumont, B., Obbad, E., and Koyejo, S. Putnam-AXIOM: A functional and static benchmark for measuring higher level mathematical reasoning in LLMs. *arXiv preprint arXiv:2508.08292*, 2025.

Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

He, Z., Choi, Y. M., Zhang, K., Ji, J., Zhou, J., Xu, D., Bercovich, I., Zhang, A., and Li, L. Hardtests: Synthesizing high-quality test cases for llm coding. *arXiv preprint arXiv:2505.24098*, 2025.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *Advances in Neural Information Processing Systems*, 2021.

Hong, Z., Wu, H., Dong, S., Dong, J., Xiao, Y., Zhang, Y., Wang, Z., Huang, F., Li, L., Yang, H., and Huang, X. Benchmarking LLMs' mathematical reasoning with unseen random variables questions. *arXiv preprint arXiv:2501.11790*, 2025.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S., Solar-Lezama, A., Sen, K., and Stoica, I. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Li, Q., Cui, L., Zhao, X., Kong, L., and Bi, W. GSM-Plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. *arXiv preprint arXiv:2402.19255*, 2024.

Liu, R., Wei, J., Liu, F., Si, C., Zhang, Y., Rao, J., Zheng, S., Peng, D., Yang, D., Zhou, D., et al. Best practices and lessons learned on synthetic data. *arXiv preprint arXiv:2404.07503*, 2024.

Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. GPQA: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Sainz, O., Campos, J. A., García-Ferrero, I., Etxaniz, J., Lopez de Lacalle, O., and Agirre, E. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of EMNLP*, 2023.

Shi, W., Ajith, A., Xia, M., Huang, Y., Liu, D., Blevins, T., Chen, D., and Zettlemoyer, L. Detecting pretraining data from large language models. *arXiv preprint arXiv:2310.16789*, 2023.

Srivastava, G., Hussain, A., Bi, Z., Roy, S., Pitre, P., Lu, M., Ziyadi, M., and Wang, X. BeyondBench: Contamination-resistant evaluation of reasoning in language models. *arXiv preprint arXiv:2509.24210*, 2025.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *International Conference on Learning Representations*, 2023a.

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., and Hajishirzi, H. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023b.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022.

White, C., Dooley, S., Roberts, M., Pal, A., Feuer, B., Jain, S., Shwartz-Ziv, R., Jain, N., Saifullah, K., Naidu, S., et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

Xu, C., Guan, S., Greene, D., and Kechadi, M.-T. Benchmark data contamination of large language models: A survey. *arXiv preprint arXiv:2406.04244*, 2024.

Yang, Z., Guo, Z., Huang, Y., Wang, Y., Shi, W., Wang, Y., Liang, X., and Tang, J. Accordion-thinking: Self-regulated step summaries for efficient and readable LLM reasoning. *arXiv preprint arXiv:2602.03249*, 2026.

Zhang, H., Da, J., Lee, D., Robinson, V., Wu, C., Song, W., Zhao, T., Raja, P., Zhuang, C., Slack, D., et al. A careful examination of large language model performance on grade school arithmetic. *Advances in Neural Information Processing Systems*, 37:46819–46836, 2024a.

Zhang, H., Lin, Y., and Wan, X. PaCoST: Paired confidence significance testing for benchmark contamination detection in large language models. *arXiv preprint arXiv:2406.18326*, 2024b.

Zheng, Z., Cheng, Z., Shen, Z., Zhou, S., Liu, K., He, H., Li, D., Wei, S., Hao, H., Yao, J., Sheng, P., Wang, Z., Chai, W., Korolova, A., Henderson, P., Arora, S., Viswanath, P., Shang, J., and Xie, S. LiveCodeBench Pro: How do olympiad medalists judge llms in competitive programming? *arXiv preprint arXiv:2506.11928*, 2025.

Zhu, K., Chen, J., Wang, J., Gong, N. Z., Yang, D., and Xie, X. Dyval: Dynamic evaluation of large language models for reasoning tasks. *arXiv preprint arXiv:2309.17167*, 2023.

A. Human audit methodology and per-item verification

This appendix documents the expert-auditing protocol behind the *Verified* and *Verified Full* releases, which convert the automatic VeRA pipeline into evaluation-ready artifacts whose label reliability does not rest on Judge calibration alone.

A.1. Annotator pool and instructions

Auditors were six STEM PhD students with prior AIME / USAMO or graduate-level mathematical training. For each rendered variant, annotators independently:

1. solved the problem, without access to the Teacher-generated verifier code;
2. compared their independently computed answer to the verifier output;
3. checked template renderability (unambiguous, no unresolved placeholders, no trivially-solvable leakage of the gold quantity);
4. for VERA-E only, additionally checked semantic equivalence to the original seed (same underlying task, differing only in parameters / surface form / language).

An item was *accepted* only on full agreement between independent annotator computation and verifier output *and* pass on renderability (and equivalence, for VERA-E).

A.2. Released VERA-E manual verification (220/220)

All 220 items in the released VERA-E evaluation subsets were audited: AIME-2024 (60/60), AIME-2025 (60/60), Beyond-AIME (100/100). All 220 items passed audit. The larger but easier GSM8K and GPQA-Diamond VERA-E pools are *not* part of the 220-item Verified release; GPQA is additionally flagged as arbitration-based rather than verifier-certified (Appendix N).

A.3. Released VERA-H/ VERA-H PRO expert audit (2,187/2,299)

Expert auditing over 2,299 automatically generated VERA-H/ VERA-H PRO math items retained 2,187 (95.13%) for evaluation use: 2,036/2,135 for VERA-H and 151/164 for VERA-H PRO. Per-benchmark retention:

- AIME-2024-II: 69/70 VERA-H, 14/14 VERA-H PRO (≈99% / 100%)
- Beyond-AIME: 466/500 VERA-H, 93/100 VERA-H PRO (≈93% / 93%)
- AMO-Bench: 217/244 VERA-H, 44/50 VERA-H PRO (≈89% / 88%)

Retention remains high even on AMO-Bench, directly supporting the claim that the automatic pipeline is already high-quality in practice.

A.4. Pair-preserving *Verified Full*

Because VERA-H PRO pair attrition (151/164) is undesirable for per-seed paired comparability, we additionally built a stricter *Verified Full* variant: when an item failed audit, we restarted the pipeline with the same seed to regenerate a new candidate and re-audit, repeating until audit passed. This yields VERA-H Verified Full and VERA-H PRO Verified Full, both one audited item per seed. Because this pipeline is no longer purely automatic, we flag it clearly: it is *automatic synthesis + iterative expert audit* rather than "no humans in the loop".

A.5. Common rejection reasons (qualitative)

Across audits, the dominant rejection reasons were (i) subtle verifier miscalibration on edge cases (e.g. integer/rational boundaries), (ii) ambiguous rendering where two reasonable interpretations yielded different answers, and (iii) generator choosing degenerate parameter regimes that made the problem trivial. All three are issues of *specification quality*, not of mathematical verification semantics: for every accepted item, the verifier's computed label matches the annotators' independently derived answer.

Table 4. **Released artifact summary.** "Auto" = raw automatic pipeline output; "Verified" = expert-audited subset; "Verified Full" = pair-preserving re-generated-until-audit-pass subset. Code-verified: labels from executing the deterministic verifier. Arbitration: LLM-consensus filter (Appendix N).

Release	Source benchmark	Mode	Verification	Size
VERA-E (Auto)	GSM8K (test)	Equivalent	Code-verified	2638 items
VERA-E (Verified)	AIME-2024	Equivalent	Code + human	60 items
VERA-E (Verified)	AIME-2025	Equivalent	Code + human	60 items
VERA-E (Verified)	Beyond-AIME	Equivalent	Code + human	100 items
VERA-E (Auto)	GPQA-Diamond	Equivalent	Arbitration†	871 items
VERA-H (Auto)	AIME-2024-II	Hardened	Code-verified	70 items
VERA-H (Verified)	AIME-2024-II	Hardened	Code + human	69 items
VERA-H PRO (Auto)	AIME-2024-II	Paired hardest	Code-verified	14 items
VERA-H PRO (Verified)	AIME-2024-II	Paired hardest	Code + human	14 items
VERA-H (Auto)	Beyond-AIME	Hardened	Code-verified	500 items
VERA-H (Verified)	Beyond-AIME	Hardened	Code + human	466 items
VERA-H PRO (Auto)	Beyond-AIME	Paired hardest	Code-verified	100 items
VERA-H PRO (Verified)	Beyond-AIME	Paired hardest	Code + human	93 items
VERA-H (Auto)	AMO-Bench	Hardened	Code-verified	244 items
VERA-H (Verified)	AMO-Bench	Hardened	Code + human	217 items
VERA-H PRO (Auto)	AMO-Bench	Paired hardest	Code-verified	50 items
VERA-H PRO (Verified)	AMO-Bench	Paired hardest	Code + human	44 items

A.6. Threats to validity

We note three: (i) Annotator difficulty ceiling: some AMO-level items approach annotator capability. We therefore cross-checked all audit disagreements with a third senior annotator; items with residual disagreement were rejected, i.e., the audit is biased conservative, not generous. (ii) Sample coverage: the audit covers released items, not the full infinite instance stream of each specification. The *specification*'s correctness claim over its full parameter manifold is inherited from the validated verifier, not from the audit. (iii) Judge-model ranking in VERA-H PRO: judge ranking noise can change which candidate is chosen, but never changes label correctness (§2.2). The *Verified Full* release mitigates any remaining pair-attrition concern.

B. Datasheet for the VERA dataset suite

We follow the "Datasheets for Datasets" template of Gebru et al. (2021) to document the released artifacts. VERA ships as a *layered* artifact: framework code, raw automatic outputs, human-audited *Verified* subsets, and pair-preserving *Verified Full* subsets (§C).

B.1. Motivation

For what purpose was the dataset created? To enable *renewable, contamination-resistant* evaluation of LLM reasoning by making benchmarks *executable specifications* rather than static question lists, and to supply evaluation-ready hardened sets that restore headroom on saturated reasoning benchmarks.

B.2. Composition

What do instances represent? Each instance is a rendered natural-language reasoning problem together with a programmatically computed gold answer, produced from an executable specification (§2.3).

How many instances are there in total, and from which underlying seed benchmarks? See Table 4. Seed benchmarks: GSM8K (Cobbe et al., 2021), AIME-2024/2025, Beyond-AIME, AMO-Bench, GPQA-Diamond (Rein et al., 2023).

What data does each instance consist of? (i) the rendered question text; (ii) the deterministic gold answer from the validated verifier; (iii) the slot assignment and RNG seed used; (iv) provenance metadata (seed id, generator id, sample idx); (v) for *Verified* items, an audit record (annotator agreement, rejection reason if any).

Are there any sensitive attributes, PII, or offensive content? No. All seed benchmarks are widely used mathematical /

scientific reasoning datasets. Templates and slot assignments are parameter-level transformations and do not introduce sensitive attributes.

Is the data split into training / validation / test? VERA is an *evaluation* dataset suite. Standard practice is to sample a disjoint evaluation instance stream *post-training* from each specification. For reproducibility we release fixed instance IDs used for the numbers reported here; reviewers / downstream users may sample fresh IDs at any time via the released specifications.

B.3. Collection process

How was the data acquired? Seed items are taken from publicly released benchmarks. Executable specifications are automatically synthesized by a Teacher LLM (GPT-5 default; Gemini 3 Pro and Claude 4.6 Opus also used for cross-Teacher controls) in a propose–execute–validate loop (Algorithm 1). For VERA-H/VERA-H PRO, a Judge LLM applies a spec-level noise-discrimination filter; for the *Verified* and *Verified Full* subsets, STEM PhD annotators independently audited every item (Appendix A).

What was the compute budget? Synthesis is a one-time cost per seed (a small number of Teacher proposals per seed on average). Instance sampling at evaluation time is CPU-only (no LLM calls). Quantitative cost statistics are reported in Appendix K.

B.4. Preprocessing / labeling

Labels are produced by executing the validated verifier code on the sampled slot assignment within a restricted sandbox (resource-limited subprocess, no network, import whitelist; Appendix K).

B.5. Uses

Intended uses. (i) Robustness / familiarity diagnostics via VERA-E; (ii) renewable frontier evaluation via VERA-H/VERA-H PRO; (iii) research on self-hardening and proposer–solver-gap estimation; (iv) controlled contamination studies. The *Verified* releases are recommended for direct model-comparison use; the raw Auto layer is recommended for methodology reproduction.

Uses that should be avoided. (i) The automatic pipeline's raw outputs are *not* claimed to be a drop-in replacement for human-authored olympiad problems; (ii) GPQA-Diamond VERA-E numbers rely on arbitration-based verification and should not be conflated with code-verified numbers.

B.6. Distribution and licensing

Framework code, specifications, raw Auto outputs, Verified subsets, and Verified Full subsets are released under a permissive research license (exact license specified in the code release). Seed benchmarks retain their original licenses; we release only specifications (T, G, V) and newly generated instances.

B.7. Maintenance

Because specifications are executable objects, bug fixes propagate cleanly: fixing a verifier once invalidates all previously generated instances from that specification and re-sampling is automatic. We commit to (i) versioned specifications with cryptographic hashes, (ii) a changelog of verifier revisions, and (iii) an audit log of accepted / rejected items for the Verified and Verified Full releases.

B.8. Responsible-release notes

The release includes **(a)** a datasheet (this appendix); **(b)** reproducibility artifacts (sandbox, RNG shim, deterministic hashing); **(c)** hosting on a public repository with version control; **(d)** clearly documented licensing; **(e)** an intended-use statement (above) and an explicit no-misuse boundary (above); **(f)** author responsibility for maintenance, specified in the repository; **(g)** a sealed-evaluation option for contamination-sensitive leaderboards (Appendix S).

C. Released datasets and responsible publishing.

For responsible release, we publish VERA as a *layered artifact*. L1 contains framework code, schema, and sandbox runner. L2 contains raw automatic outputs for every benchmark. L3 contains human-*Verified* evaluation subsets, comprising 220 of 220 VERA-E items on AIME-2024, AIME-2025, and Beyond-AIME, together with 2,187 expert-audited VERA-H and VERA-H PRO items. L4 contains pair-preserving *Verified Full* subsets. For evaluation-ready use we recommend L3 and L4, and L2 is for methodology reproduction. A full datasheet for datasets (Gebru et al., 2021), audit protocol, and responsible-publishing guidelines, including sealed evaluation distributions and hashed specifications and RNG state, appear in Appendices B, A, and S.

D. Additional VeRA-E case studies and broader discussion

This appendix collects qualitative examples and extended discussion that are useful for interpretation but too space-intensive for the 9-page main paper.

D.1. Our Deliverables (extended)

VERA ships three deliverables, corresponding to two operational modes and one methodological advance.

(1) VERA-E: Logic-preserving equivalent rewrites for quality and diagnostics. VERA-E generates logic-preserving rewrites that preserve a problem's underlying reasoning structure while changing its surface form (names, numbers, wording, language). These rewrites serve two purposes:

1. **Improving quality.** On GSM8K, verified rewrites eliminate ambiguous or mislabeled problems. A conservative proxy (problems that both GPT-5 and Gemini-2.5-Pro failed) drops from 2.12% on seeds to 0.76% on VERA-E rewrites. The rewriting process exposes and removes artifacts introduced during original labeling.

2. **Detecting exposure.** On AIME, we apply identical rewriting to both AIME-2024 and AIME-2025. Some model families show larger seed-to-variant drops on 2024 than on 2025: for instance, GPT-5.1-high drops 16.7 Avg@5 points on 2024 but only 0.3 on 2025. Since the same rewriting pipeline was applied to both years, this differential is hard to explain unless models have greater familiarity with the older benchmark, which is exactly the signal VERA-E is designed to expose.

(2) VERA-H and VERA-H Pro: Human-free challenge generation at scale. VERA-H applies systematic transformations to increase problem difficulty while maintaining verifiable correctness: composition, constraint tightening, step inflation. VERA-H Pro then selects the single most difficult variant based on a predetermined ranking of the variants the verifier produces.

Using seeds from Beyond-AIME and AMO-Bench, VERA-H generates large numbers of verified instances that are challenging enough to hold frontier models to around 50% Avg@5. This is the regime where tasks are neither obviously solvable nor completely out of reach.

(3) Verified benchmarks as infrastructure. VERA illustrates a general principle: *in any domain with programmatic verification, we can scale up data production without sacrificing label quality*. The economics separate a high-cost, one-time expense (synthesizing a specification that defines the problem) from near-zero marginal cost (sampling additional instances).

On GSM8K, for instance, synthesizing specifications costs roughly \$100 for 1,319 seeds; after that, each additional instance costs essentially nothing (millisecond-scale consumer CPU usage). For harder AIME problems, synthesis is more expensive ($\approx$\$3,000 for 270 seeds), but the investment amortizes over the unlimited number of verified instances that can be generated afterward.

All of our source code and generated data are publicly available. VERA represents infrastructure for renewable evaluation, with benchmarks that cannot be depleted because they generate fresh instances on demand.

Your model answers the **seed** correctly, but fails on a **logic-preserving** rewrite **verified** to be equivalent.

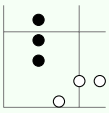
SEED ✓ → VeRA-E ×

Seed (AIME 2024-II-9) Seed

Problem. There is a collection of 25 indistinguishable white chips and 25 indistinguishable black chips. Find the number of ways to place some of these chips in the 25 unit cells of a 5×5 grid such that:

- each cell contains at most one chip;
- all chips in the same row and all chips in the same column have the same colour;
- the placement is **maximal**: any additional chip placed on the grid would violate one or more of the previous two conditions.

Gold: 902
DeepSeek-V3.2-thinking: 902 ✓



VeRA-E: verified rewrite →

VERA-E variant Same logic

Problem. You have 4 indistinguishable chips of color red and 3 indistinguishable chips of color purple. Some of these chips are placed into the unit cells of a 1 **by** 2 grid, with at most one chip per cell. The placement obeys:

- (i) within any single row, all chips present (if any) are the same color;
- (ii) within any single column, all chips present (if any) are the same color.

Moreover, the placement is **maximal**: adding a chip of either color to any empty cell would violate (i) or (ii). How many distinct placements are possible?

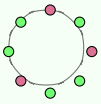
Gold: 2
DeepSeek-V3.2-thinking: 5 ×

Failure mode: The model solves for 2 by 2 configuration instead, suggesting seed-shape anchoring and overgeneralizing from the default 5×5 square structure in the seed.

Seed (AIME 2024-I-11) Seed

Problem. Each vertex of a regular octagon is independently colored either red or blue with equal probability. The probability that the octagon can then be rotated so that all of the blue vertices end up at positions where there had been red vertices is $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m + n$.

Gold: 371
Gemini-3-Pro: 371 ✓



VeRA-E: verified rewrite →

VERA-E variant Same logic

Problem. Each vertex of a regular 12-**gon** is independently colored green or purple with equal probability. Let the probability that the polygon can be rotated so that every purple vertex lands on a position that was originally green be $\frac{m}{n}$ in lowest terms. Compute $m + n$.

Gold: 2595
Gemini-3-Pro: 2685 ×

Failure mode: The task requires case work on the number of purple nodes which is at most 4 in seed and 6 in variant. For the variant, the model computes for up to 4 purple nodes correctly, but immediately fails on 5, showing a lack of genuine reasoning to generalize beyond memorizing the seed pattern.

Immediate insights:

- **Benchmark familiarity is not reasoning.** Correct on a known seed can coexist with failure on a verified-equivalent rewrite.
- **What breaks:** Models lean on **surface cues** (grid size, typical case splits, memorized patterns) instead of the underlying invariant.

Figure 2. **Insights by VERA.** Two AIME seeds are solved correctly, yet the same models fail on **VeRA-E variants** that preserve the logical constraints but change surface form. This exposes a common failure mode: **seed performance can be inflated by familiarity / surface heuristics**, while verified-equivalent rewrites reveal brittleness.

D.2. Broader Implications

VERA addresses a structural issue in AI evaluation: the tradeoff between benchmark quality and benchmark age. High-quality human-authored benchmarks are difficult to create but static and finite. High-quality synthetic benchmarks are dynamic but often lower in quality or poorly calibrated.

VERA offers a middle ground: use human expertise to create seed problems, then use programmatic verification to generate instances from those seeds indefinitely. The expensive part (creating seeds) happens once; the cheap part (generating instances) scales without limit.

This has implications beyond evaluation. The same specifications used to generate test problems can also generate training data, with reliable correctness labels and adjustable difficulty levels.

We believe VERA provides evidence for a general principle: *programmatic verification is the next frontier for scaling up high-quality data production*.

D.2.1. PAPER STRUCTURE

The rest of the paper is organized as follows. Section 2 describes what VERA is, namely benchmark specifications that can be executed. Section 2 explains how we create specifications and generate variants. Section 3 presents experimental results across multiple math and science benchmarks. Section H explores limitations and potential applications of VERA, including future research directions. Section 4 compares VERA with related work. Section 5 concludes.

Positioning. Common responses to benchmark failure include auditing contamination, sourcing fresher "live" test sets, and perturbing surface form to probe robustness. VERA is complementary but structurally different: it makes renewal a property of the benchmark itself. By compiling each seed into an executable specification consisting of a template, generator, and verifier, VERA supports instant sampling of fresh instances while certifying labels through deterministic, lightweight CPU execution, without requiring models, GPUs, or LLMs. We unfold how this fundamentally differs from existing work in Section 4.

E. Extended method details

This appendix collects method details that are useful for replication and ablations but are too space-intensive for the 9-page main paper. In particular, we include (i) the optional perturbation score used for controlled robustness studies, (ii) the schematic contrast between VERA-E and VERA-H/VERA-H PRO, (iii) the calibration details for the judge-based validity threshold, and (iv) the full synthesis-loop diagram and pseudocode.

E.1. Perturbation scoring for controlled robustness

Quantifying the extent of change. For equivalence testing, it can be useful to constrain how far a variant drifts from the seed realization. We compute a perturbation score $\pi(v, \text{seed}) \in [0, 1]$ that combines (i) a normalized slot distance π_{num} for numeric changes, defined as the mean relative change $|\theta_i - \theta_i^{\text{seed}}| / \max(|\theta_i^{\text{seed}}|, 1)$ clipped to $[0, 1]$, and (ii) a lightweight text distance π_{tok} between template realizations, defined as the token-level Levenshtein edit distance normalized by the length of the seed rendering. We take $\pi = \frac{1}{2}(\pi_{\text{num}} + \pi_{\text{tok}})$ and accept variants within a configurable perturbation budget (default $\pi_{\max} = 0.5$).

Per-year perturbation distributions on AIME (VERA-E). The familiarity interpretation of the AIME-2024 vs. AIME-2025 seed→VERA-E gap (§3.1) relies on the two years having comparable variant-level perturbation. Figure 3 shows that the AIME-2024 and AIME-2025 VERA-E variants have statistically indistinguishable perturbation-score distributions, with two-sample Kolmogorov–Smirnov $D = 0.15$, $p = 0.51$. The larger performance drop on AIME-2024 is therefore not explained by variants being "further from seed" on 2024 than on 2025.

E.2. Hardening operator taxonomy

The current VERA-H Teacher prompts bias the proposer toward four families of hardening operators. Making this taxonomy explicit is useful both for interpreting where hardening transfers well and for guiding future extensions:

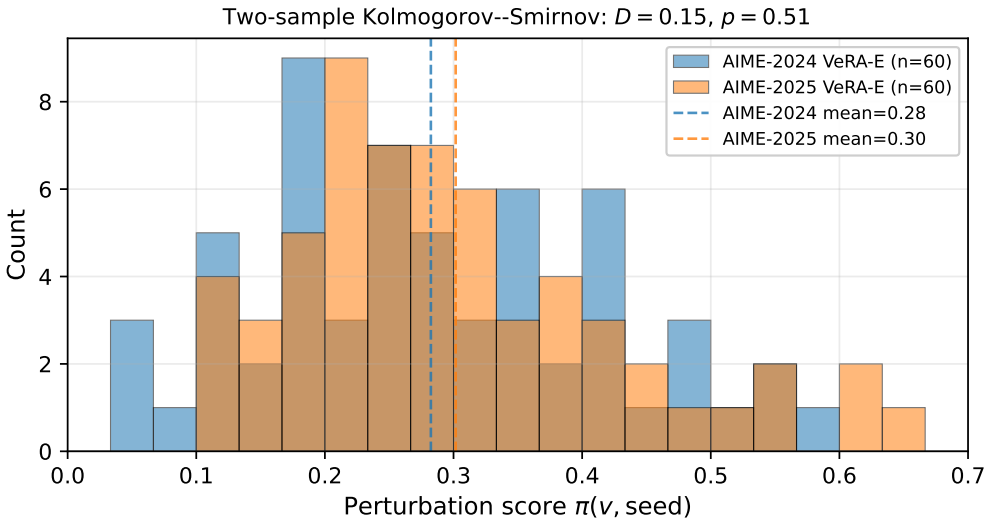


Figure 3. Perturbation score distributions for AIME-2024 vs. AIME-2025 VERA-E variants ($n = 60$ per year). The two distributions are statistically indistinguishable under a two-sample Kolmogorov–Smirnov test ($D = 0.15$, $p = 0.51$), so the 2024-vs-2025 seed→VERA-E drop asymmetry reported in §3.1 cannot be attributed to differential drift in how far variants move from their seeds.

- **Constraint tightening.** Narrowing numeric / structural constraints already present in the seed, e.g. requiring coprime parameters, positivity, or strict inequalities. Transfers well across arithmetic and combinatorics seeds; low risk of breaking the task.
- **Composition / step inflation.** Adding intermediate sub-goals, chained conditions, or dependency steps (e.g. seed computes quantity X; hardened variant computes a quantity that depends on both X and a newly introduced constraint). Transfers well to problems with compositional structure; visible strongest on AIME.
- **Parameter-regime search.** Replacing fixed parameter values with slots whose generator searches for regimes that increase reasoning depth (e.g. non-trivial modular structure). Powerful on Beyond-AIME; miscalibrates on AMO-Bench, where olympiad problems often rely on narrow algebraic coincidences that are fragile under re-parameterization.
- **Structural generalization.** Replacing a specialized condition with a parameterized one (e.g. perpendicularity → angle slot θ); this is the operator illustrated in the AIME 2024-II-10 example of Figure 4. High reward when it applies; requires that the seed admits a coherent generalization.

Most hardened variants on AIME and Beyond-AIME draw from (1) and (2); AMO survivorship bias is most strongly associated with (3).

E.3. Two-mode schematic

E.4. Validity-threshold calibration

Why $\tau = 4$? Calibration as a validity bound. The threshold τ trades off false positives (accepting incorrect specifications) against false negatives (rejecting correct ones). We calibrate τ to bound false positives rather than to optimize downstream model scores. In a small-scale human validation on 50 randomly sampled AIME variants, STEM PhD students independently verified each rendered variant. At $\tau = 4$ we observed 1/50 false positives and 3/50 false negatives. At $\tau = 5$, false positives dropped to 0/50 but false negatives rose to 14/50 (22%), reflecting Judge limitations on harder items. We therefore adopt $\tau = 4$ as a conservative default that prioritizes validity while maintaining usable throughput. τ is a tunable conservatism knob for the spec-level filter rather than an optimal threshold; the released *Verified* subsets (§3.2, Appendix A) layer expert auditing on top of any threshold setting. For use cases requiring higher-precision post-hoc answer checking, Appendix F describes a complementary multi-agent debate pipeline that reduces single-judge false positives by 8.3×, raising precision from 55% to 92.5% on 195 hard competition-mathematics variants.

E.5. Human-free synthesis loop (full)

Algorithm 1 VERA Augmentation Pipeline

Require: Seed problem $(q, a, \text{id}, \text{year})$, config c, LLM Compiler $\mathcal{C}$, LLM Judge $\mathcal{J}$
Ensure: List of valid variants, generator artifacts, summary

```
seed ← (q, a, id, year)
valid_variants ← []
generator_artifacts ← []
feedback ← None
for prompt_attempt = 1, ..., c.prompt_attempt_limit do
    if |valid_variants| ≥ c.variants_per_seed then
        break
    end if
    attempt_results ← []
    payload ← C.convert_to_spec(q, a, id, feedback)
    (spec, err) ← _parse_compiler_payload(seed, payload)
    if err ≠ None then
        feedback ← "Missing required fields"
        continue
    end if
    gen_fn ← compile_generator(spec.generator_code)
    ver_fn ← compile_verifier(spec.verifier_code)
    generator_id ← id + "_prompt" + prompt_attempt
    generator_artifacts.append(GeneratorArtifact(spec, generator_id))
    for sample_idx = 1, ..., c.samples_per_prompt do
        if |valid_variants| ≥ c.variants_per_seed then
            break
        end if
        seed_val ← _hash_seed(id, generator_id, sample_idx)
        variant ← _sample_variant(spec, gen_fn, ver_fn, seed_val)
        if variant ≠ None then
            attempt_results.append(variant)
            if c.mode ∈ {VeRA-H, VeRA-H-Pro} then
                variant.judge_consistent ← _run_judge(variant, c)
                if variant.judge_consistent then
                    valid_variants.append(variant)
                end if
            else if c.mode = VeRA-E then
                (valid_E, y_E) ← ver_fn(θ_seed) {seed-anchored consistency check}
                if valid_E and y_E = a then
                    valid_variants.append(variant)
                else
                    feedback ← "Verifier disagrees with seed answer"; break
                end if
            end if
        end if
    end for
    feedback ← GenerateFeedback(attempt_results)
end for
if valid_variants = [] then
    valid_variants ← _fallback_rephrase(seed)
end if
return valid_variants, generator_artifacts, AugmentationSummary
```

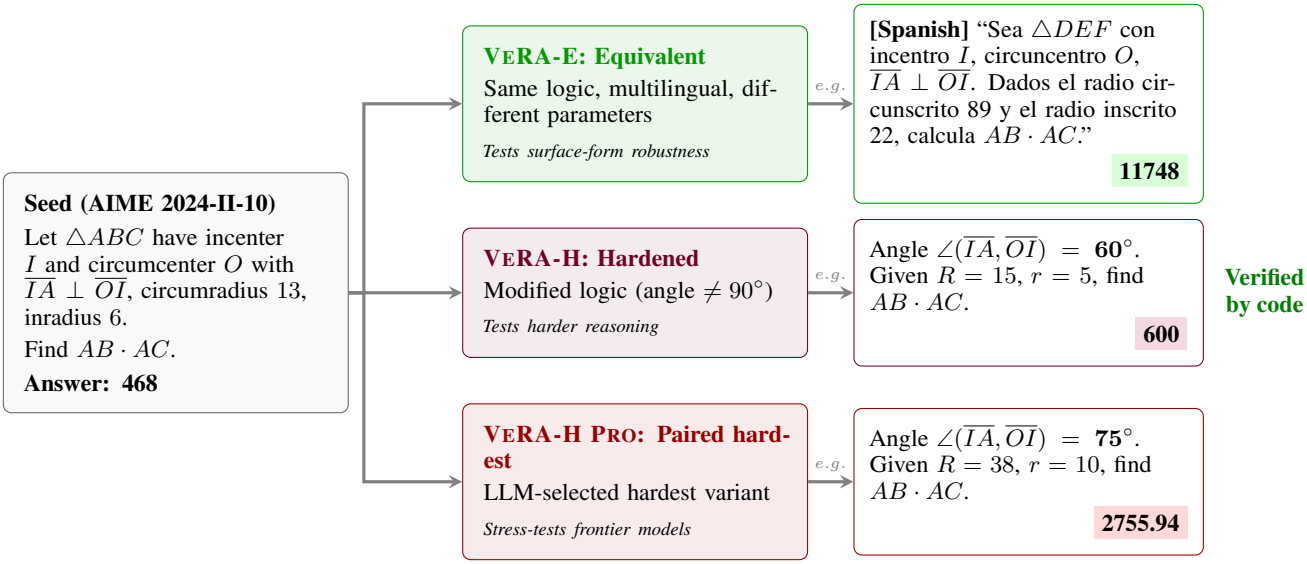


Figure 4. VERA difficulty stratification on AIME 2024-II-10. VERA-E changes surface form and language while preserving the underlying logic. VERA-H modifies the mathematics, here generalizing perpendicularity to arbitrary angles. VERA-H PRO picks the single hardest variant from the VERA-H pool via *Judge model* ranking, stress-testing the reasoning limits of frontier models.

Synthesizing specifications from seeds is a one-time cost. VERA has an LLM propose candidate specifications, then validates and repairs them using the gates above. Algorithm 1 and Figure 5 sketch the workflow: (1) LLM Compiler proposal; (2) schema validation; (3) sandbox compilation; (4) mode-specific validity checks (seed-anchored consistency for VERA-E; noise-discrimination validity for VERA-H/VERA-H PRO); (5) deterministic sampling via hash-seeded RNG; and (6) structured feedback and retry until enough valid instances are obtained.

Feedback mechanism. We return actionable diagnostics, including parse, compile, and runtime failures, seed-consistency failures for VERA-E, noise-discrimination failures for VERA-H and VERA-H PRO, and low-yield generator warnings, so the Teacher can attempt targeted repairs rather than regenerate blindly.

E.6. Sandboxing and security

All LLM-generated code runs in a restricted subprocess sandbox with resource limits, no network or filesystem writes, and an import whitelist. This keeps execution safe and debugging reproducible. Implementation details appear in the appendix.

E.7. Cost structure

VERA separates one-time synthesis cost from near-zero marginal sampling cost. For N instances drawn from a single spec:

$$\text{cost per instance} = \frac{C_{\text{synth}}}{N} + C_{\text{sample}} \xrightarrow[N \to \infty]{} C_{\text{sample}}.$$

We report synthesis costs, acceptance rates, and end-to-end benchmark costs in experiments.

F. Multi-Agent Debate Pipeline for Answer Verification

VERA's noise-discrimination validity check (§2.4) uses a Judge model as a specification-level filter. For applications requiring higher-precision post-hoc answer checking—for example, when auditing a dense pool of VERA-H candidates before release—we additionally develop a multi-agent debate pipeline as an engineering supplement. On a controlled evaluation set of hard competition-mathematics variants it substantially reduces single-judge false positives; we report this as practical evidence of the approach rather than as a fully validated contribution, and discuss the limitations of the current evaluation below.

① Seed Problem from Existing Datasets

Seed benchmark item (AIME 2024-II-10)

"$\triangle ABC$ has incenter I and circumcenter O with $\overline{IA} \perp \overline{OI}$, circumradius $R = 13$, inradius $r = 6$. Find $AB \cdot AC$."

Gold label: $AB \cdot AC = 468$

② Two Executable Pools for Variant Generation (Equivalent (E) vs Hardened (H))

Compile the single seed problem into two executable problem families (separate specs + separate language carriers)

VERA-E executable specification

VERA-E *keeps the underlying logic fixed.*

Spec pool (E): templates/verifiers for the E family
Slots: (R, r) (no θ, $IA \perp OI$ by default)

Generator-E: (samples valid (R, r) and rewrite)

```
def gen_E():
  (R,r) = sample_R_r()
  style = sample_LC_E_style()
  spec  = pick_spec_E()   # pool of E-specs
  return spec, {R,r}, style
```

Language Carrier-E:

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

Verifier-E: (computes label for E-spec)

```
def verify_label_E(R,r):
  assert valid_R_r(R,r)
  R2  = R*(R-2*r)     # OI^2
  IA2 = R*R - R2      # IA is perpendicular to OI
  return IA2 + 4*R*r # AB*AC
```

VERA-H executable specification

Hardness modifies/extends the task (e.g., replace $\perp$ with θ), so slots and rendering can differ.

Spec pool (H): templates/verifiers for the H family
Slots: (R, r, θ) (introduce θ for added hardness)

Generator-H: (searches for harder parameters)

```
def gen_H(mode='H'):
  spec = pick_spec_H()  # pool of H-specs
  if mode == 'H':
    (R,r,th) = sample_R_r_theta()
  else:  # Pro
    C = [sample_R_r_theta() for _ in range(K)]
    (R,r,th) = select_hardest(C)
  style = sample_LC_H_style()
  return spec, {R,r,th}, style
```

Language Carrier H: (H-specific slots/structure)

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

Verifier-H: (computes label for H-spec)

```
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 Generate verified families

VERA-E outputs: equivalent logic

Same underlying E-spec; different LC-E renders.

E-Instance 1 (English)
$R = 14,\ r = 5$ with $\overline{IA} \perp \overline{OI}$
420 **(verified by Verifier-E)**

E-Instance 2 (Spanish render)
"Dado $R = 6$ y $r = 2$ con $\overline{IA} \perp \overline{OI}$, calcula $AB \cdot AC$."
72 **(verified by Verifier-E)**

Renewable evaluation with the same underlying logic

VERA-H/ VERA-H PRO outputs: harder variants

H-spec can change slots/structure.

H-candidates (verified by V-H)
A: $R = 15,\ r = 5,\ \theta = 60^\circ \Rightarrow 600$
B: $R = 38,\ r = 10,\ \theta = 75^\circ \Rightarrow 2755.94$
C: $R = 22,\ r = 7,\ \theta = 40^\circ \Rightarrow$ (verified)
VERA-H PRO pick: B (paired hardest)

H-render (Spanish example)
"Dado $R = 38,\ r = 10,\ \theta = 75^\circ$, calcula $AB \cdot AC$."
2755.94 **(verified by Verifier-H)**

Human-free scaling with reliable labels

Figure 5. **VERA pipeline.** A seed item is compiled into executable specifications that can generate verified families.

F.1. Motivation

The evaluation set consists of 195 variants derived from 58 competition-mathematics problems spanning combinatorics, number theory, and algebra at AMO/USAMO/IMO difficulty. Each problem contributes 3–4 variants: exactly one is labeled `correct` (verified by hand), and the remainder are labeled `wrong` (cross-checked to confirm the candidate answer differs from the ground truth). The pipeline thus operates as a binary classifier over this controlled pool. *Problem-level accuracy* (*Prob. Acc* in Table 5) counts a problem as correctly classified only when the pipeline accepts its correct variant *and* rejects all its wrong variants, i.e., the correct answer is uniquely identified within the problem's variant pool; a problem with a false positive among its wrong variants is counted as incorrect regardless of whether the correct variant was also accepted.

A single-judge baseline (`gpt-5.2`, `effort="low"`) achieves only 55.1% mean precision on this set, meaning roughly one in two accepted answers is wrong. Problem-level solve-from-scratch accuracy for the same model is approximately 39% across three independent runs, confirming that incorrect variants cannot be trivially dismissed by re-computation.

The overriding design principle is **precision first**: for benchmark construction a false positive (mislabeled entry) silently degrades all downstream evaluations, whereas a false negative merely reduces accepted-variant yield without compromising quality.

F.2. Pipeline Overview

Agents. Five heterogeneous agents each carry a distinct verification persona: Formalist Verifier (algebraic consistency and bound handling), Theorem Auditor (theorem applicability and lemma usage), Independent Resolver (short independent re-derivation), Optimization Skeptic (boundary conditions and optimality gaps), and Pragmatic Cross-Checker (numerical and symbolic spot checks). All agents use `gpt-5.2` at `effort="low"` with different system instructions. Heterogeneity is critical: homogeneous agents reinforce shared errors (Du et al., 2024), while agents with distinct analytical lenses are more likely to surface flaws others miss.

Initial deliberation. Each agent follows a five-step protocol before any exchange: (A) restate the claim precisely; (B) enumerate 2–4 failure modes and assess each; (C) perform at least one concrete independent check; (D) steelman the opposition and classify the strongest objection as FATAL, UNCERTAIN, or REFUTED; (E) commit to a verdict. Only a FATAL steelman permits a stance change, preventing cascading updates driven by unverifiable speculation. Each agent outputs a structured JSON with `assessment_type` $\in$ {`answer_supported`, `answer_refuted`, `reasoning_insufficient_but_answer_not_refuted`}, `evidence_grade`, a full rationale, and a $\leq$400-character `summary`.

Debate rounds. The pipeline runs up to five rounds. In each round every agent receives a scheduler broadcast—current vote tally, assessment-type breakdown, and a structured *disagreement digest* enumerating the top unresolved opposing claims—and an inbox of targeted direct messages assembled by two independent mechanisms: scheduler-generated challenges (cross-challenge, weak-supporter pressure, undecided resolution) derived programmatically from prior-round outputs with no additional LLM call, and peer-to-peer messages from the previous round's `outbound_dm` fields. Each agent must directly address the strongest opposing argument (Step 1), perform one new concrete check (Step 2), re-steelman the other side (Step 3), and commit (Step 4); silent stance changes are prohibited. Per-round context is kept manageable by propagating only the $\leq$400-character `summary` of each agent's prior rationale, following the Accordion-Thinking principle (Yang et al., 2026). Early stopping on unanimous 5-0 is permitted after a minimum of two full rounds.

Verdict rule. A candidate answer is accepted if and only if $\geq$3 of 5 agents carry `assessment_type = answer_supported` in the final round. In implementation this is decomposed into two sequential checks: (1) a supermajority fast-fail requiring $\geq$3 support votes (fewer $\Rightarrow$ immediately labeled `wrong`), and (2) an assessment gate requiring $\geq$3 `answer_supported` among those support votes. Check (2) is strictly stronger on the acceptance path; check (1) is retained as a fast-fail and as a guard against JSON-inconsistent outputs.

F.3. Results

The debate pipeline reduces false positives from 24.7 to 3.0 per run on average (8.3$\times$), raising precision from 55.1% to 92.5% with low run-to-run variance (precision range: 90.2%–95.0%). Problem-level accuracy improves from 19.3/58 (33.3%) to 34.7/58 (59.8%).

Table 5. Answer-verification results on 195 competition-mathematics variants (58 problems). Three independent runs per system with fresh LLM caches. Positive class = `correct`. All systems use `gpt-5.2` at `effort="low"`.

System	Acc	Prec	Recall	F1	FP	FN	Prob. Acc
Baseline r1	72.8%	54.5%	51.7%	53.1%	25	28	18/58
Baseline r2	72.8%	54.9%	48.3%	51.4%	23	30	18/58
Baseline r3	73.9%	55.9%	56.9%	56.4%	26	25	22/58
Baseline avg	73.2%	55.1%	52.3%	53.6%	24.7	27.7	19.3/58
Debate r1	87.2%	90.2%	63.8%	74.7%	4	21	34/58
Debate r2	87.2%	92.3%	62.1%	74.2%	3	22	34/58
Debate r3	88.7%	95.0%	65.5%	77.6%	2	20	36/58
Debate avg	**87.7%**	**92.5%**	**63.8%**	**75.5%**	**3.0**	**21.0**	**34.7/58**

F.4. Limitations

Recall is 63.8%, yielding $\approx$21 false negatives per run. Three qualitatively distinct failure modes account for all false negatives. *Capability saturation* ($\approx 70\%$ of FNs, avg. 14.7/run): all five agents converge on `reasoning_insufficient_but_answer_not_refuted` because none can independently verify the hardest solutions at `effort="low"`; the assessment gate then rejects the variant. These cases cannot be resolved by prompt changes alone. *Active false refutation* ($\approx 25\%$, avg. 5.3/run): at least one agent outputs `answer_refuted` based on a flawed mathematical objection; stronger steelman enforcement or tool-augmented cross-checking could address these. *Mixed evidence* ($\approx 5\%$, avg. 1.0/run): partial support that falls short of the ≥ 3 `answer_supported` threshold. Per-run totals: $14.7 + 5.3 + 1.0 = 21.0$ FNs on average. The false negatives reduce accepted-variant yield; the quality of accepted variants is unaffected.

Inference cost is up to $5 \times 5 = 25$ LLM calls per variant versus one for the single-judge baseline. Adaptive early stopping is already active ($\approx 40\%$ of variants reach consensus before round 5); a cheap pre-filter could further reduce this overhead for large-scale generation.

Statistical evidence is limited. Three independent runs provide directional evidence but no confidence intervals, bootstrap estimates, or significance tests. For a system with up to 25 LLM calls per sample and observed run-to-run variation (precision range 90.2%–95.0%), three runs are insufficient to precisely characterize the precision distribution. No ablations are reported on agent count, round limit, verdict threshold, or model configuration, so the contribution of each design choice to the observed gains remains unquantified. The results should be read as "empirically effective under this configuration" rather than "methodologically fully validated."

Scope is narrow. Results are specific to hard competition-mathematics variants verified by `gpt-5.2` at `effort="low"`. Generalization to open-domain QA, code verification, or problems with less unambiguous ground truth is unknown. The pipeline's behavior under weaker or stronger judge models, different agent counts, or other mathematical domains has not been studied, and no claims are made about performance beyond the setting described here.

G. Additional experiment tables

This appendix collects synthesis statistics and model-by-model result tables that are referenced in the main paper but omitted for space.

G.1. Synthesis validation statistics

G.2. Model-by-model tables omitted from the main paper

H. Discussion

VERA reframes a benchmark from a finite list of questions into a renewable generator of verifiable instances. This shift changes how we should interpret benchmark scores at the frontier: progress ought to persist under verified resampling, and near-ceiling seed scores should not be mistaken for a reliable separator. Below we offer practical guidance, clarify failure modes, and highlight broader implications.

Table 6. **Synthesis reliability under default budgets.** LLM retries are capped at 20 per seed; the generator–verifier loop uses a 300-second timeout per run. For VERA-E, we target 1 ideal execution spec consistent with the seed and generate all variants from that spec. For VERA-H we target 5 accepted variants per seed.

VERA-E				
Dataset	**Seeds**	**1 try**	**≤20 tries**	**Fallback**
GSM8K (test)	1319	1319	1319	0
AIME-2024	30	29	30	0
AIME-2025	30	27	30	0
Beyond-AIME	100	63	89	11
VERA-H (target 5 variants/seed)				
Dataset	**Seeds**	**5/5**	**4/5**	**0/5**
Beyond-AIME	100	100	0	0
AMO-Bench	50	48	1	1

Table 7. **Dominant rejection sources under default budgets.** Percentages are measured over attempted candidates within the corresponding stage, with denominators in parentheses from the synthesis runs in Table 6 for Beyond-AIME. For VERA-H, the *Judge model* serves as a conservative spec-level validity gate; a manual audit of 50 Judge rejections suggests some false negatives on extremely hard problems due to Judge limitations.

Stage	**Observed rejection source**
Spec-level	(1) compilation failure; (2) verifier mismatch on the seed base assignment; (3) low-yield / timeout (cannot generate a single verifier-accepted instance within 300 seconds).
Instance-level	Sanity-check rejects 3.1% (e.g., contradictory constraints from over-parameterization or ambiguous rendering).
VERA-H judge filter	Judge rejects 44.0% of verifier-accepted candidates; a manual audit indicates that some rejections are false negatives attributable to Judge model limitations.
Runtime	Generator–verifier loop timeouts are rare (0.7%).

H.1. Practical guidance: using VERA in future evaluations

Report seeds *and* VERA-E, not seeds alone. Seed accuracy remains a useful anchor, but it is increasingly confounded by benchmark familiarity and ceiling effects. We recommend reporting, for each benchmark: (i) the standard seed metric, (ii) the corresponding VERA-E metric on verified-equivalent variants sampled post-training under a fixed budget, and (iii) a robustness statistic that makes the relationship explicit, such as the seed→VERA-E drop, the conditional stability $\Pr[\text{variant correct} \mid \text{seed correct}]$, or rank stability across models. In saturated regimes, a *low* rank correlation between seeds and VERA-E is often the intended signal: it suggests the static benchmark was hiding meaningful differences behind near-perfect scores and benchmark-specific shortcuts.

Use year-controlled or split-controlled VERA-E whenever possible. The strongest interpretability comes from comparisons where the only change is verified resampling. AIME-2024 vs. AIME-2025 is a case in point: identical format, identical rewriting pipeline, yet systematically different drops (Table 9). When the same verified rewriting induces a larger drop on an older, well-circulated split than on a fresher control, the natural explanation is seed familiarity rather than genuine reasoning improvement.

Use VERA-H to refresh saturated benchmarks; use VERA-H PRO for paired hardening. When a benchmark approaches saturation, the right response is not to retire it and restart a human authoring pipeline, but to renew it through verifier-certified hardened families. VERA-H provides a scalable mechanism for creating harder instances with reliable labels. When per-seed comparability matters, VERA-H PRO offers a paired protocol: each seed is evaluated against its selected hardest verified variant from a fixed candidate pool, reducing confounds from comparing unrelated hard problems.

What to release for reproducibility. To make VERA benchmarks stable scientific artifacts, we recommend releasing: (i) the specifications (template, generator, verifier), (ii) deterministic sampling rules (hash-seeding scheme), (iii) acceptance

Table 8. **Per-model VeRA-E results across all benchmarks.** Avg@5 (%) on seeds, verified-equivalent VERA-E variants, and Δ (variant−seed). This table covers every evaluated model and dataset in the VeRA-E suite. All values are from the provided CSV; Δ is computed before rounding.

Model	GSM8K			AIME-2024			AIME-2025			GPQA-Diamond			Beyond-AIME		
	Seed	VERA-E	Δ	Seed	VERA-E	Δ	Seed	VERA-E	Δ	Seed	VERA-E	Δ	Seed	VERA-E	Δ
Gemini-2.5-Pro	93.9	95.0	1.2	88.0	73.3	-14.7	85.3	73.0	-12.3	82.1	81.4	-0.8	57.2	57.0	-0.2
GLM-4.6	94.1	94.8	0.7	90.7	71.7	**-19.0**	92.0	76.3	**-15.7**	78.8	80.2	1.4	68.2	66.4	-1.8
GPT-5.1-high	95.6	96.9	1.4	92.0	75.3	-16.7	88.7	88.3	-0.3	86.4	84.5	-1.8	71.6	73.0	1.4
Claude-Sonnet-4.5-thinking	95.3	96.2	0.9	84.0	74.0	-10.0	78.7	68.7	-10.0	81.0	80.1	-0.9	51.6	53.0	1.4
Seed-1.6-Thinking-0715	95.1	95.9	0.8	85.3	72.0	-13.3	80.0	75.3	-4.7	78.4	78.5	0.1	54.4	55.8	1.4
DeepSeek-V3.2-thinking	94.0	95.4	1.4	92.0	79.0	-13.0	90.0	87.7	-2.3	82.1	83.3	1.2	69.4	67.6	-1.8
Seed-1.6-1015-high	95.6	96.3	0.7	89.3	75.0	-14.3	80.7	76.7	-4.0	77.4	78.9	1.5	58.8	58.2	-0.6
Kimi-K2-thinking	95.0	86.6	**-8.4**	86.7	75.3	-11.3	78.7	71.7	-7.0	79.9	83.4	3.5	54.8	44.0	**-10.8**
Minimax-M2	94.7	94.9	0.2	84.7	55.7	**-29.0**	76.7	57.0	**-19.7**	76.7	74.4	-2.3	56.8	54.4	-2.4
Gemini-3-Pro-Preview	94.5	95.0	0.5	94.0	81.3	-12.7	90.7	85.7	-5.0	86.7	85.7	-1.0	76.6	74.0	-2.6
GPT-5-high	95.6	97.1	1.6	90.0	74.7	-15.3	89.3	85.3	-4.0	81.9	82.1	0.1	68.0	69.8	1.8
Kimi-K2-0905	95.0	95.6	0.6	65.3	51.3	-14.0	47.3	44.0	-3.3	73.5	74.2	0.7	37.0	34.6	-2.4
qwen3-max-0923	95.5	96.9	1.3	82.7	68.7	-14.0	74.7	69.7	-5.0	76.1	76.7	0.7	53.6	58.8	5.2
GPT-5.1-chat-latest	94.3	96.0	1.7	48.7	46.0	-2.7	50.7	41.7	-9.0	69.9	69.1	-0.8	28.8	30.6	1.8
Seed-1.6-Lite-1015-high	95.1	95.8	0.7	87.3	73.0	-14.3	77.3	70.0	-7.3	77.6	76.9	-0.7	54.8	53.6	-1.2
DeepSeek-V3.1-thinking	94.5	94.8	0.3	90.7	77.7	-13.0	86.7	83.0	-3.7	80.0	81.3	1.3	71.8	66.0	-5.8
Mean (16 models)	**94.9**	**95.2**	**0.3**	**84.5**	**70.2**	**-14.2**	**79.2**	**72.1**	**-7.1**	**79.3**	**79.4**	**0.1**	**58.3**	**57.3**	**-1.0**

Table 9. **Model-by-model AIME rewrite sensitivity under VERA-E.** We report Avg@5 (%) on seeds and verified-equivalent VERA-E variants, and the absolute drop Δ (variant−seed). All values are from the provided CSV; Δ is computed before rounding. The largest drop in each year is highlighted in red.

Model	AIME-2024 (30 seeds → 60 VERA-E)			AIME-2025 (30 seeds → 60 VERA-E)		
	Seed	VERA-E	Δ	Seed	VERA-E	Δ
Gemini-2.5-Pro	88.0	73.3	-14.7	85.3	73.0	-12.3
GLM-4.6	90.7	71.7	-19.0	92.0	76.3	-15.7
GPT-5.1-high	92.0	75.3	-16.7	88.7	88.3	-0.3
Claude-Sonnet-4.5-thinking	84.0	74.0	-10.0	78.7	68.7	-10.0
Seed-1.6-Thinking-0715	85.3	72.0	-13.3	80.0	75.3	-4.7
DeepSeek-V3.2-thinking	92.0	79.0	-13.0	90.0	87.7	-2.3
Seed-1.6-1015-high	89.3	75.0	-14.3	80.7	76.7	-4.0
Kimi-K2-thinking	86.7	75.3	-11.3	78.7	71.7	-7.0
Minimax-M2	84.7	55.7	**-29.0**	76.7	57.0	**-19.7**
Gemini-3-Pro-Preview	94.0	81.3	-12.7	90.7	85.7	-5.0
GPT-5-high	90.0	74.7	-15.3	89.3	85.3	-4.0
Kimi-K2-0905	65.3	51.3	-14.0	47.3	44.0	-3.3
qwen3-max-0923	82.7	68.7	-14.0	74.7	69.7	-5.0
GPT-5.1-chat-latest	48.7	46.0	-2.7	50.7	41.7	-9.0
Seed-1.6-Lite-1015-high	87.3	73.0	-14.3	77.3	70.0	-7.3
DeepSeek-V3.1-thinking	90.7	77.7	-13.0	86.7	83.0	-3.7
Mean (16 models)	**84.5**	**70.2**	**-14.2**	**79.2**	**72.1**	**-7.1**

Table 10. **Model-by-model headroom restoration on AIME-2024-II under VERA-H/VERA-H PRO.** Avg@5 (%) on seeds (14 items), VERA-H (70 hardened items; ≈5 per seed), and VERA-H PRO (14 paired hardest-per-seed items). Δ values are variant−seed in absolute points (computed before rounding).

Model	Seed	VERA-H	$\Delta_{\text{VERA-H}}$	VERA-H PRO	$\Delta_{\text{VERA-H PRO}}$
Gemini-2.5-Pro	92.9	82.0	-10.9	65.7	-27.1
GLM-4.6	94.3	81.1	-13.1	67.1	-27.1
GPT-5.1-high	94.3	90.9	-3.4	74.3	-20.0
Claude-Sonnet-4.5-thinking	90.0	72.0	-18.0	60.0	-30.0
Seed-1.6-Thinking-0715	87.1	76.6	-10.6	67.1	-20.0
DeepSeek-V3.2-thinking	91.4	84.0	-7.4	68.6	-22.9
Seed-1.6-1015-high	92.9	76.6	-16.3	67.1	-25.7
Kimi-K2-thinking	85.7	15.1	-70.6	45.7	-40.0
Minimax-M2	82.9	64.3	-18.6	51.4	-31.4
Gemini-3-Pro-Preview	75.7	74.6	-1.1	62.9	-12.9
GPT-5-high	88.6	82.3	-6.3	78.6	-10.0
Kimi-K2-0905	71.4	40.6	-30.9	32.9	-38.6
Qwen3-max-0923	80.0	59.1	-20.9	51.4	-28.6
GPT-5.1-chat-latest	52.9	34.3	-18.6	28.6	-24.3
Seed-1.6-Lite-1015-high	88.6	70.3	-18.3	57.1	-31.4
DeepSeek-V3.1-thinking	90.0	81.1	-8.9	58.6	-31.4
Mean (16 models)	**84.9**	**67.8**	**-17.1**	**58.6**	**-26.3**

Table 11. **Per-model VeRA-H results on Beyond-AIME and AMO-Bench.** Avg@5 (%) on seeds, hardened variants (VERA-H), and paired hardest-per-seed variants (VERA-H PRO). Beyond-AIME: 100 seeds → 500 (VERA-H) / 100 (VERA-H PRO). AMO-Bench: 50 seeds → 244 (VERA-H) / 50 (VERA-H PRO). Δ values are variant−seed (computed before rounding).

Model	Beyond-AIME					AMO-Bench				
	Seed	VERA-H	$\Delta_{\text{VERA-H}}$	VERA-H PRO	$\Delta_{\text{VERA-H PRO}}$	Seed	VERA-H	$\Delta_{\text{VERA-H}}$	VERA-H PRO	$\Delta_{\text{VERA-H PRO}}$
Gemini-2.5-Pro	57.2	57.7	0.5	52.2	-5.0	28.0	43.5	15.5	37.6	9.6
GLM-4.6	68.2	67.1	-1.1	63.6	-4.6	40.0	52.9	12.9	48.0	8.0
GPT-5.1-high	71.6	77.3	5.7	75.2	3.6	56.0	64.6	8.6	58.8	2.8
Claude-Sonnet-4.5-thinking	51.6	56.3	4.7	50.2	-1.4	18.0	36.0	18.0	32.4	14.4
Seed-1.6-Thinking-0715	54.4	57.9	3.5	51.6	-2.8	40.0	41.6	1.6	38.0	-2.0
DeepSeek-V3.2-thinking	69.4	68.6	-0.8	64.2	-5.2	28.0	51.6	23.6	46.0	18.0
Seed-1.6-1015-high	58.8	61.8	3.0	55.0	-3.8	48.0	43.1	-4.9	36.4	-11.6
Kimi-K2-thinking	54.8	43.8	-11.0	46.6	-8.2	22.0	26.5	4.5	17.2	-4.8
Minimax-M2	56.8	52.8	-4.0	47.6	-9.2	20.0	36.6	16.6	28.4	8.4
Gemini-3-Pro-Preview	76.6	72.2	-4.4	65.4	-11.2	56.0	61.1	5.1	56.0	0.0
GPT-5-high	68.0	77.6	9.6	70.6	2.6	40.0	58.4	18.4	54.4	14.4
Kimi-K2-0905	37.0	30.4	-6.6	25.8	-11.2	8.0	19.9	11.9	18.8	10.8
qwen3-max-0923	53.6	60.1	6.5	52.2	-1.4	14.0	39.2	25.2	35.2	21.2
GPT-5.1-chat-latest	28.8	33.0	4.2	32.6	3.8	6.0	20.7	14.7	17.6	11.6
Seed-1.6-Lite-1015-high	54.8	55.6	0.8	48.6	-6.2	36.0	39.7	3.7	36.4	0.4
DeepSeek-V3.1-thinking	71.8	68.6	-3.2	62.2	-9.6	48.0	52.1	4.1	52.8	4.8
Mean (16 models)	**58.3**	**58.8**	**0.4**	**54.0**	**-4.4**	**31.8**	**43.0**	**11.2**	**38.4**	**6.6**

rates and common failure categories from synthesis, and (iv) the sampled instance identifiers used in each reported evaluation. This supports exact reproduction while preserving the ability to sample *new* post-training instances for future audits.

H.2. Limitations and failure modes

Verifier coverage limits the domains VERA can address. VERA works best where correctness is programmatically verifiable. Certain tasks, such as subjective judgments, open-ended writing, or long-form reasoning without checkable certificates, do not admit clean deterministic verifiers. Even in math and science, some problems may require heavy external solvers or expensive symbolic reasoning, complicating verification and reproducibility.

Specification correctness is a one-time risk that must be managed explicitly. VERA amortizes correctness across instances, which makes spec-level validation critical. Our pipeline surfaces seed mismatches, runtime failures, and low-yield generators as actionable signals, but rare incorrect specs remain possible. Spec-level tests, seed-consistency checks (VERA-E), randomized fuzzing, and targeted audits of a small sample of specifications are principled ways to reduce residual risk without reintroducing per-instance labeling.

Difficulty is a distribution-design problem. A generator defines a distribution, and a hardening library defines which difficulty axes get emphasized. Hardening is not guaranteed to be monotone across all cognitive skills or across all seed distributions. Non-monotonic cases such as AMO-Bench in Table 1 are therefore not merely "bad news"; they are diagnostic feedback about which operators are miscalibrated for which domains. Mitigations include diversifying operators, monitoring diversity statistics, and using VERA-H PRO-style selection to concentrate evaluation on the most discriminative verified candidates.

Surface realization can fail even when symbolic correctness holds. A verifier certifies a label for a slot assignment, but not necessarily the linguistic clarity of a rendered question. This motivates a lightweight judge or heuristic sanity filter in certain settings: it rejects confusing or misleading renders but never defines correctness. A longer-term direction is to incorporate surface-faithfulness checks directly into the synthesis and validation suite.

H.3. Broader implications: renewable evaluation

Benchmarks that do not "rot." Static benchmarks degrade under repeated reuse: leakage becomes likely and saturation becomes inevitable. VERA offers an alternative structure: pay a one-time cost to compile seeds into executable specifications, then sample fresh verified instances indefinitely. This shifts incentives for evaluation: progress must generalize across verified variants, not merely fit a fixed test set.

From benchmark artifacts to benchmark infrastructure. Releasing a VERA benchmark means releasing a specification suite, including programs, deterministic sampling rules, and audit artifacts, rather than only a finite list of questions. This enables transparent auditing, controlled stress tests, and continual augmentation under consistent semantics, while keeping label integrity anchored in deterministic verification.

I. Extended related work comparison

The main paper includes a concise related-work discussion. For completeness, we reproduce the detailed comparison table here, with concurrent renewable-benchmark efforts (RV-BENCH, BeyondBench) explicitly added. The columns separate two distinct properties that are sometimes conflated: *verifier-certified* labels (obtained by executing code rather than by LLM voting) and *seed-anchored (T,G,V)* specifications (compiling each *existing* benchmark item into a reusable template–generator–verifier triple). The two properties are independent: a system can produce labels programmatically (e.g. from a structured object) without compiling existing benchmark items, and conversely.

Note on DyVal / GSM-Symbolic classification. DyVal computes labels programmatically from structured objects (e.g. DAGs), but its tasks are generated de novo over algorithmic object classes rather than compiled from existing benchmark items; we therefore treat it as not "seed-anchored" in the sense above. GSM-Symbolic is seed-anchored and supports equivalent families, but its symbolic templates are manually authored and GSM8K-specific rather than a general compilation pipeline, and it does not include a general hardening protocol; we mark it "Limited" on hardness scaling for that reason.

Table 12. Comparison of VERA with closely related evaluation and data-generation paradigms. "Limited" indicates partial support. "Human audit on release" marks whether a documented human-verified subset is distributed.

Approach	Verifier-certified	Post-train renewal	Equiv. families	Hardness scaling	Seed-error surfacing	Training ready	Human audit on release
Static benchmarks (GSM8K, AIME)	✗	✗	✗	✗	✗	✗	✓(seeds only)
Temporal filtering (LiveCodeBench, LiveBench)	✗	Limited	✗	✗	✗	✗	✓
LLM-only augmentation (paraphrase)	✗	Limited	Limited	✗	✗	✓	✗
Template / symbolic perturbation	Limited	Limited	Limited	✗	✗	✓	Limited
GSM-Symbolic (Mirzadeh et al., 2024)	Limited	Limited	✓	Limited	✗	✓	Limited
Distributional eval. (e.g., DyVal (Zhu et al., 2023))	✗	✓	✗	✓	✗	✗	✗
RV-BENCH (Hong et al., 2025)	✓	✓	✓	✗	✗	✗	✗
BeyondBench (Srivastava et al., 2025)	✓	✓	✗	Limited	✗	✗	✗
VERA (ours)	✓	✓	✓	✓	✓	✓	✓

RV-BENCH and BeyondBench produce verifier-certified labels and post-training freshness; RV-BENCH is partially seed-anchored at the benchmark-item level, whereas BeyondBench generates algorithmic problems de novo.

I.1. Programmatic verification, synthetic data, and label noise

Programs appear in reasoning work mainly as inference-time aids: PAL (Gao et al., 2023) and Program-of-Thoughts (Chen et al., 2023) have models write and execute code to solve problems; VERA takes the opposite approach, using programs at *benchmark construction time* to define correctness and certify labels (PAL-style solvers can still be evaluated on VERA-generated benchmarks). On the training side, RLVR (Guo et al., 2025) demonstrated that programmatic correctness checks yield cleaner reward signals; VERA applies the same principle to evaluation, with the verifier as source of truth, so fresh labeled instances can be generated without requiring human graders to solve each problem. Under the self-hardening framing of §2.2, the same infrastructure also lets us measure the *proposer–solver gap*: whether a proposer can generate verifiable tasks beyond what it can itself solve, a prerequisite for future self-evolution loops.

Synthetic data has become widespread (Wang et al., 2023b; Gunasekar et al., 2023), but purely LLM-generated QA pairs have a problematic failure mode: when the same model family produces both questions and answers, label noise can compound into false supervision (Liu et al., 2024). VERA addresses this by separating responsibilities. LLMs handle natural language, including phrasing, diversity, and specification proposals, while labels come from deterministic execution. This preserves scalability while maintaining label integrity, and is paired with light-weight human audit on the released *Verified* subsets to combine throughput with expert oversight.

J. Model Schema and Example

This appendix walks through the specification schema that VERA uses and provides a concrete example.

J.1. Schema Overview

A VERA specification consists of five components:

- **Slot definitions**: variable names, types (integer, rational, categorical), valid ranges, and any constraints between slots.
- **Natural language template**: parameterized text with placeholders for slots, rendering well-formed questions.
- **Verification code**: a Python function `verifier(assign)` → `(bool, answer)` that computes the canonical answer deterministically.
- **Generator code**: a Python function `generator(rng)` → `assign` that produces valid slot assignments without rejection.
- **Base assignment**: the canonical `assign_0` that reproduces the original question and answer.

J.2. Specification Example

The following simplified example illustrates how the generator and verifier work together:

```python
# Slots: a, b, c are integers with constraints
def generator(rng):
    a = rng.randint(2, 20)
    b = rng.randint(2, 20)
    c = rng.randint(1, 10)
    # Enforce constraints coherently (no rejection needed)
    if a == b:
        b += 1
    return {"a": a, "b": b, "c": c}

def verifier(assign):
    a, b, c = assign["a"], assign["b"], assign["c"]
    # Validate constraints
    if a == b:
        return False, None
    # Compute gold answer deterministically
    return True, a * b + c

# Template: "If you have {a} rows of {b} items each,
#            plus {c} extra items, how many total?"
# Base assignment: assign_0 = {"a": 5, "b": 4, "c": 3}
# Expected answer: verifier(assign_0) = (True, 23)
```

Notice that the generator guarantees $a \neq b$ by construction rather than by rejection, which is what we mean by coherent sampling. The verifier computes the answer through pure arithmetic, so label correctness is guaranteed for all valid assignments.

K. Implementation Details

This section covers the implementation details that matter most for reliable operation at scale.

K.1. Sandboxed Execution

LLM-generated code needs secure, reproducible execution. VERA runs all such code in a restricted sandbox via subprocess isolation with resource limits:

```python
def _subproc_worker(q, code, fn_name, payload, time_limit, mem_limit):
    # Resource limits (Unix-like systems)
    import resource
    resource.setrlimit(resource.RLIMIT_AS, (mem_limit, mem_limit))
    resource.setrlimit(resource.RLIMIT_CPU, (time_limit, time_limit))

    # Execute in trusted sandbox with math/random available
    glob = {"__builtins__": __builtins__, "math": math, "random": random}
    loc = {}
    exec(code, glob, loc)
    result = loc[fn_name](payload)
    q.put((True, result, None))
```

The security constraints are:

- No network access or filesystem writes
- Import whitelist: only `math`, `random`, `fractions` allowed
- CPU timeout (default 300s) and memory limit (default 2GB)
- Deterministic RNG seeds for reproducible debugging

The Python snippet above is illustrative. In production, user code runs in an isolated container/jail with outbound networking disabled and a read-only file system (except for a temporary working directory). The import whitelist is enforced by the sandbox runtime configuration.

K.2. RNG Shim for Reproducibility

VERA provides a deterministic random number generator wrapper (`RNGShim`) that exposes a consistent interface:

```
class RNGShim:
    def __init__(self, seed: int):
        self._seed = int(seed)
        self._r = random.Random(self._seed)

    # Core methods
    def random(self) -> float: return self._r.random()
    def randint(self, a: int, b: int) -> int: return self._r.randint(a, b)
    def uniform(self, a: float, b: float) -> float: return self._r.uniform(a, b)
    def choice(self, seq): return self._r.choice(seq)
    def shuffle(self, x): self._r.shuffle(x); return x

    # Distribution methods
    def gauss(self, mu, sigma): return self._r.gauss(mu, sigma)
    def gammavariate(self, alpha, beta): return self._r.gammavariate(alpha, beta)
```

K.3. Judge-Based Answer Verification

For VERA-H variants, VERA uses LLM-based judge verification with noise answers to prevent rubber-stamping:

```
def _run_judge(self, question_text, correct_answer, seed_value, config):
    prompts = []
    # Add correct answer trials
    for _ in range(config.judge_correct_trials):  # default: 2
        prompts.append((correct_answer, True, False))
    # Add noise (incorrect) answer trials
    noises = _generate_noise_answers(correct_answer,
                                     config.judge_noise_trials,  # default: 3
                                     rng)
    for noise in noises:
        prompts.append((noise, False, True))

    rng.shuffle(prompts)  # Randomize order

    successes = 0
    for candidate, expected, is_noise in prompts:
        verdict = judge_llm(question, candidate)
        if verdict == expected:
            successes += 1

    return successes >= config.judge_consistency_threshold  # default: 4
```

K.4. Noise Answer Generation

To guard against judge bias, VERA generates plausible-but-incorrect noise answers. Here is an example for real and integer answer perturbation (e.g., GSM8K, AIME):

```
def _generate_noise_answers(correct_ans: str, count: int, rng) -> List[str]:
    val = float(correct_ans)
    is_int = abs(val - round(val)) < 1e-9
    noises = []
    while len(noises) < count:
        if is_int:
            delta = rng.randint(1, 9)
            candidate = int(val) + delta if rng.random() < 0.5 \
                        else max(0, int(val) - delta)
        else:
            span = max(1.0, abs(val) * 0.1)
            delta = rng.uniform(0.05 * span, span)
            candidate = val + (delta if rng.random() < 0.5 else -delta)
        noises.append(str(candidate))
```

```
    return noises
```

K.5. Configuration Parameters

VERA exposes tunable parameters through `GenerationConfig`:

Table 13. VERA Configuration Parameters

Parameter	Default	Description
`variants_per_seed`	5	Target variants per seed problem
`prompt_attempt_limit`	20	Maximum *Teacher model* retry attempts
`samples_per_prompt`	5	Generator samples per specification
`generator_timeout_sec`	300.0	Max time per generator call
`judge_consistency_threshold`	4	Required *Judge model* successes (out of 5)
`judge_correct_trials`	2	Trials with correct answer
`judge_noise_trials`	3	Trials with noise answers
`base_seed`	0	RNG base for determinism
`debug`	False	Enable verbose logging

K.6. Fallback Mechanisms

When the synthesis loop exhausts its attempts without producing valid variants, VERA falls back to deterministic rephrasing:

```
def _fallback_rephrase(self, seed: SeedProblem) -> List[VariantOutcome]:
    templates = [
        f"In the {seed.year} AIME, contestants faced: {seed.question}",
        f"Rephrased challenge: {seed.question}",
        f"Consider this AIME-style task: {seed.question}",
        f"Alternate wording: {seed.question}",
        f"Restatement for clarity: {seed.question}",
    ]
    # Variants flagged with metadata["fallback"] = True
    return [make_variant(t, seed.answer) for t in templates]
```

L. Detailed Pipeline of VERA

L.1. VERA-E: Generation of equivalent instances

L.1.1. Definition of equivalence

We define equivalence in terms of **latent program equivalence**: two instances are equivalent if they share the same underlying solution program except for parameter substitution. For text tasks, this means the arithmetic or symbolic structure is preserved even when surface realizations change.

This definition is deliberately stronger than paraphrase. A pure paraphrase only touches surface form, while VERA-E also varies numerical parameters, entity names, and contextual framing, all while preserving the underlying computation graph.

L.1.2. Perturbation Control

Generating equivalent instances requires that variants stay "close" to the seed family and do not drift arbitrarily. We score each instance using a *perturbation score* that combines:

- **Numerical Perturbation**: normalized distance between sampled slots and base assignment (e.g., relative change in numerical values).
- **Text Perturbation**: normalized template divergence (e.g., token-level edit distance between rendered questions under fixed θ).

Variants are accepted within a configurable perturbation budget, and we report statistics to make distribution shifts explicit.

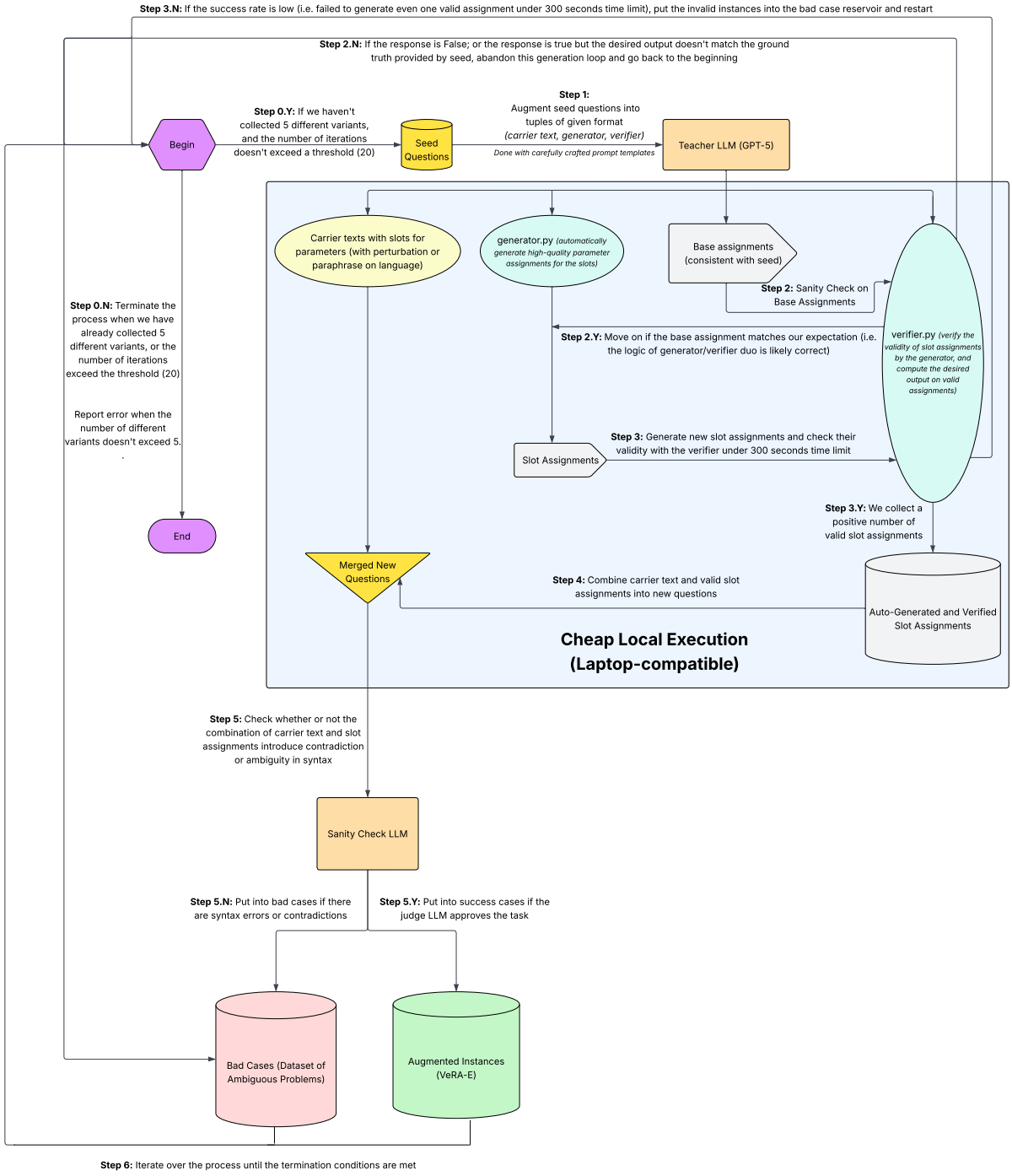


Figure 6. End-to-end workflow for VERA-E dataset curation

The specific formula is not the point; what matters is that VERA detects and controls perturbations rather than relying on ad hoc rewriting prompts.

L.2. VERA-H and VERA-H Pro: Hardening Transformations

While VERA-E tests stability under isomorphic resampling, many benchmarks are approaching saturation for frontier models. VERA-H tackles this by applying increasingly difficult transformations that remain program-verifiable.

M. AIME Evaluation Details

The AIME results are generated from five evaluations per item. We report several metrics:

- **Seed**: performance on the original competition problems.
- **VERA-E**: performance on equivalent instance variants (same latent program, varied parameters).
- **VERA-H**: performance on hardened variants (increased difficulty, same verification).
- **Gen columns**: conditional accuracy on augmented variants among the correct instances of the seed, which measures generalization stability.

The AIME-2024 and AIME-2025 subsets each contain 30 problems. Variants are sampled with fixed RNG seeds to ensure reproducibility.

N. GPQA-Diamond Augmentation

For GPQA-Diamond (Rein et al., 2023), deterministic program verification is not always possible given the domain knowledge involved. We therefore use a *repeated arbitration* protocol:

1. Generate a candidate augmented question (paraphrase, distractor modification, or controlled substitution).
2. Query a robust evaluation model to solve the augmented question K times independently.
3. Accept the instance if the evaluator returns the expected answer with majority agreement across K attempts.
4. Reject instances where the evaluator produces inconsistent answers, which signals ambiguity or augmentation failure.

This approach is less rigorous than program verification, but it provides a meaningful quality filter. It catches artifacts early (e.g., accidental leakage of gold option letters) and flags unstable questions before they enter the evaluation set.

Augmentation statistics. From 198 GPQA-Diamond seeds, we generate 871 verified variants that pass the arbitration filter. Acceptance rates vary by augmentation type: paraphrasing augmentations pass at roughly 85%, while distractor modifications pass at around 70%, reflecting the greater difficulty of maintaining answer stability when modifying incorrect options.

O. Statistical Estimation and Confidence Intervals

Because VERA allows resampling from task families, it is both natural and recommended to report uncertainty alongside point estimates.

O.1. Confidence Intervals Based on Resampling

We recommend the following procedure:

1. Sample R sets of independent variants from the same collection of families using different RNG seeds.

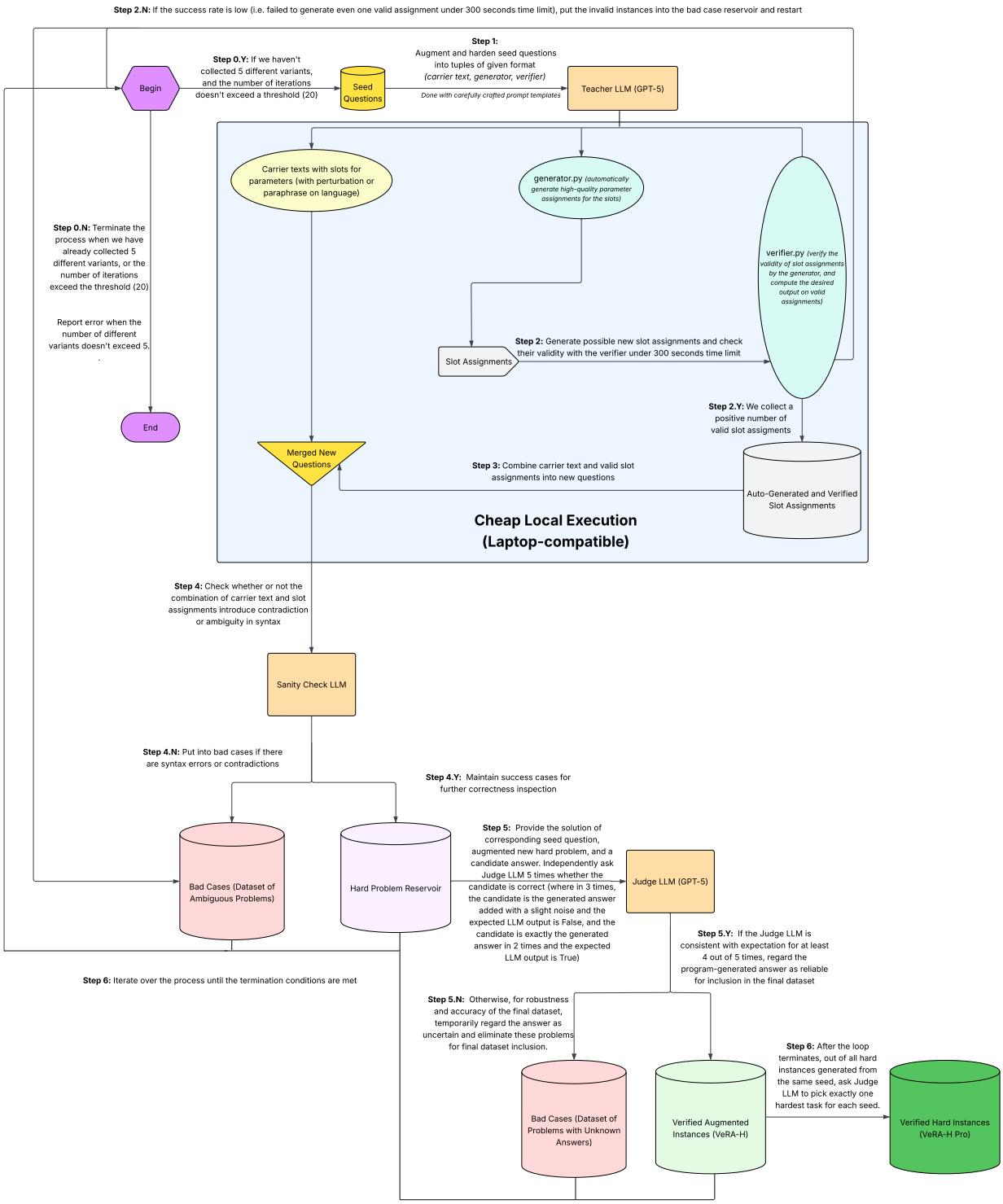


Figure 7. End-to-end workflow of VERA-H and VERA-H Pro dataset curation

2. Compute the metric of interest (e.g., mean AVG@5) on each set.

3. Report the empirical mean and standard deviation across all runs, or construct bootstrap confidence intervals.

This gives a more accurate picture of model performance than a one-off evaluation on fixed instances. It also enables formal hypothesis testing when comparing models.

O.2. Practical Considerations

For computational efficiency, we recommend $R \geq 5$ resampling runs when reporting uncertainty. The marginal cost is low: once specifications are synthesized, instance generation reduces to rendering templates and running verifiers.

When comparing models, pairwise resampling, in which all models are evaluated on the same sets of variants, reduces variance and increases statistical power for detecting real performance differences.

P. Prompt Templates

P.1. AIME Teacher Prompt

System Prompt

```
You are an experienced math olympiad problem setter who specializes in making
existing AIME problems harder and more generalizable.
You are given a seed problem from an AIME contest, and need to generate a more
difficult problem family based on that.
Please modify at least one condition of the seed problem to design a new, more
difficult problem family.  This new problem family must require a different and more
advanced solution approach from the original and should not be solvable by guessing.
The problem family must be of such quality and novelty that any instance could be
accepted at IMO, AIME, or serve as valuable training exercise.
Furthermore, please provide a proof of correctness, and a clear explanation for your
generalized family in the metadata.
Always return ONE JSON object.  No commentary.
```

P.2. Required JSON Schema

```
{
  "language_wrapper": "In a math contest, {alpha} students ...",
  "slots": {
    "alpha": {"kind": "int", "description": "total students",
              "harder_than_seed": true}
  },
  "generator": {
    "type": "python",
    "code": "def generator(rng):\n    # use rng.* for randomness\n    return {'alpha':
    ...}"
  },
  "verifier": {
    "type": "python",
    "code": "def verifier(assign):\n    # validate and compute\n    return True, answer"
  },
  "hardness_rationale": "Explain why generated family is harder than seed.",
  "notes": "Optional implementation notes.",
  "meta": {"seed_id": "<example_id>", "source_year": <year>}
}
```

P.3. Judge Prompt

System Prompt

```
You are a meticulous mathematics judge.  Determine if a proposed answer solves the
problem exactly.
You will see a Teacher model's reasoning and proposed generalization, but it may
contain mistakes.  Evaluate the problem + candidate answer independently; only
output 'True' if the reasoning and computations fully justify the answer.
Return an explanation followed by a final line that is exactly True or False.
```

P.4. Hardest Variant Selection Prompt

System Prompt

```
You are ranking math contest problems by difficulty.
Given several variants derived from the same seed, choose the hardest one.  Consider:
complexity, number of steps, tricky reasoning, boundary cases.
Respond with JSON: {"hardest_variant":  "<id>", "reason":  "..."}
```

Q. Data Structures

Q.1. Core Data Classes

```python
@dataclass(frozen=True)
class SeedProblem:
    """Normalized representation of a seed problem."""
    id: str
    year: int
    question: str
    answer: str

@dataclass
class TeacherSpec:
    """Structured response from the Teacher model."""
    seed_id: str
    language_wrapper: str
    generator_code: str
    verifier_code: str
    hardness_rationale: str
    notes: Optional[str] = None
    metadata: Dict[str, Any] = field(default_factory=dict)

@dataclass
class GenerationConfig:
    """Tunable knobs for augmentation workflow."""
    variants_per_seed: int = 5
    prompt_attempt_limit: int = 20
    samples_per_prompt: int = 5
    generator_timeout_sec: float = 300.0
    judge_consistency_threshold: int = 4
    judge_correct_trials: int = 2
    judge_noise_trials: int = 3
    base_seed: int = 0
    debug: bool = False

@dataclass
class VariantOutcome:
    """Result of a single generator sample after judge filtering."""
    seed_id: str
    generator_id: str
    prompt_attempt: int
```

```python
    sample_index: int
    assignment: Dict[str, Any]
    question_text: str
    correct_answer: str
    numeric_answer: Optional[float]
    generator_attempts: int
    generator_elapsed_sec: float
    judge_trials: List[JudgeTrial] = field(default_factory=list)
    judge_consistent: bool = False
    judge_successes: int = 0
    noise_answers: List[str] = field(default_factory=list)
    metadata: Dict[str, Any] = field(default_factory=dict)

@dataclass
class GeneratorArtifact:
    """Final combined generator export."""
    generator_id: str
    seed_id: str
    language_wrapper: str
    combined_code: str
    teacher_generator_code: str
    teacher_verifier_code: str
    hardness_rationale: str
    notes: Optional[str]
    metadata: Dict[str, Any] = field(default_factory=dict)

@dataclass
class AugmentationSummary:
    """Aggregated stats for reporting."""
    seed_id: str
    total_prompt_attempts: int
    total_samples: int
    valid_variants: int
    failures: List[str] = field(default_factory=list)
```

R. Command-Line Interface

R.1. Augmentation Pipeline (`prepare_vera.py`)

```cpp
# AIME dataset augmentation
python cli/prepare_vera.py \
  --teacher_impl \vera.oracles:PromptTeacher \
  --judge_impl \vera.oracle_llm_io:judge_llm_call \
  --dataset_name di-zhang-fdu/AIME_1983_2024 \
  --dataset_format aime \
  --variants_per_seed 5 \
  --prompt_attempt_limit 20 \
  --samples_per_prompt 5 \
  --judge_consistency_threshold 4 \
  --out_augmented artifacts/vera-H/aime_augmented.jsonl \
  --out_augmented_hard artifacts/vera-H-Pro/aime_hard.jsonl \
  --generators_dir artifacts/logs/generators \
  --progress_dir artifacts/logs/progress \
  --summary_json artifacts/reports/aime_summary.json

# BeyondAIME dataset
python cli/prepare_vera.py \
  --dataset_name ByteDance-Seed/BeyondAIME \
  --dataset_format beyond-aime \
  ...

# AMO-Bench dataset
python cli/prepare_vera.py \
  --dataset_name meituan-longcat/AMO-Bench \
```

```
  --dataset_format amo-bench \
  ...
```

R.2. Evaluation Pipeline (`eval_vera.py`)

```
# Evaluate on seed problems
python cli/eval_vera.py \
  --student_impl \vera.oracles:PromptStudent \
  --dataset_mode seed \
  --dataset_name di-zhang-fdu/AIME_1983_2024 \
  --min_year 2024 --max_year 2024 \
  --runs 5 \
  --tolerance 1e-3 \
  --report_json results/seed_eval.json

# Evaluate on \vera-H augmented dataset
python cli/eval_vera.py \
  --student_impl \vera.oracles:PromptStudent \
  --dataset_mode augmented \
  --dataset_path artifacts/vera-H/aime_augmented.jsonl \
  --runs 5 \
  --report_json results/vera_h_eval.json

# Evaluate on \vera-H Pro (hardest variants)
python cli/eval_vera.py \
  --dataset_mode augmented-hard \
  --dataset_path artifacts/vera-H-Pro/aime_hard.jsonl \
  ...
```

R.3. Dataset Format Support

The `-dataset_format` flag controls column parsing:

Format	ID Column	Question Column	Answer Column
aime	ID	Question	Answer
beyond-aime	ID/problem_id	problem/prompt	answer
amo-bench	question_id/id	prompt/question	solution/answer

S. Responsible Publishing Guidelines

A framework that can generate unlimited verified test instances raises practical questions about responsible publishing. We recommend a three-level separation:

1. **Framework code**: schema definitions, sandbox runner, and synthesis loop. Publishing these enables replication and community extension.
2. **Public training distributions**: specifications explicitly designated for training data generation. These can be released openly since their purpose is capacity development, not evaluation.
3. **Private evaluation distributions**: specifications reserved for evaluation, with seeds and RNG keys kept confidential. Access to evaluation can be provided via an API that samples new variants on each run, similar to interactive evaluation servers or sealed execution environments.

This separation preserves VERA's contamination resistance while enabling community reproduction and methodological advancement.

Version control and auditing. We recommend keeping cryptographic hashes of the specification files and RNG seeds used for each evaluation run. This allows post-hoc auditing and ensures that reported results can be tied to specific evaluation conditions, even when the underlying instances are not made public.

T. GSM8K artifact proxy case study

Table 14 supports the "both-wrong" GSM8K artifact proxy reported in the abstract.

Table 14. GSM8K accuracy before and after rewriting into verified VERA specifications (PASS@1, %). "Generalized GSM8K" evaluates one sampled VERA variant per seed. In addition to accuracy, the rate of items missed by *both* GPT-5 and Gemini 2.5 Pro drops from $28/1319 = 2.12\%$ on the original set to $10/1319 = 0.76\%$ after rewriting, consistent with reduced ambiguity/noise.

Model	GSM8K Test (orig)	Generalized GSM8K (VERA)
Gemini 2.5 Pro	97.12%	98.56%
OpenAI GPT-5	96.66%	98.56%

U. Result Tables

This section reports per-model Avg@5 results in Tables 15, 16, 17, and 18. We also include the mean across the 16-model suite as a final row.

Model	AIME_1983_2001_seeds (265)	AIME_1983_2001_VeRA_H (1321)	AIME2024_seeds (14)	AIME2024_VeRA_H (70)	AIME2024_VeRA_H_Pro_1230 (14)
external-api/Gemini-2.5-Pro	95.4	79.5	92.9	82.0	65.7
GLM-4.6	93.8	75.6	94.3	81.1	67.1
GPT-5.1-high	95.2	78.4	94.3	90.9	74.3
Claude-Sonnet-4.5-thinking	94.0	75.8	90.0	72.0	60.0
Seed-1.6-Thinking-0715	93.8	79.1	87.1	76.6	67.1
DeepSeek-V3.2-thinking	95.7	77.7	91.4	84.0	68.6
Seed-1.6-1015-high	94.3	79.1	92.9	76.6	67.1
Kimi-K2-thinking	59.6	71.4	85.7	15.1	45.7
Minimax-M2	90.9	74.7	82.9	64.3	51.4
Gemini-3-Pro-Preview	93.4	74.7	75.7	74.6	62.9
GPT-5-high	94.5	77.4	88.6	82.3	78.6
Kimi-K2-0905	84.1	53.7	71.4	40.6	32.9
qwen3-max-0923	92.2	74.2	80.0	59.1	51.4
GPT-5.1-chat-latest	72.1	55.3	52.9	34.3	28.6
Seed-1.6-Lite-1015-high	93.1	76.3	88.6	70.3	57.1
DeepSeek-V3.1-thinking	94.4	75.3	90.0	81.1	58.6
Mean (16 models)	**89.8**	**73.6**	**84.9**	**67.8**	**58.6**

Table 15. **VeRA-H / VeRA-H Pro results (Avg@5).** (Part 1/2) Seeds vs hardened variants across AIME (1983–2001) and AIME-2024.

V. AI Usage Disclosure

Large language models were used in two distinct capacities during this work. First, for data used in the evaluation process, we applied multiple LLMs in the end-to-end pipeline to for both data generation and quality assurance, and the methodology itself has been described in detail in the paper.

Second, we used AI writing assistants exclusively for polishing and editing the prose in this manuscript. All research ideas, experimental design, implementation, analysis, and interpretation of results were conducted solely by the human authors. To better present our work and convey the message more clearly, we leveraged AI assistants to help refine word choice and clarity, but AI assistants did not contribute to any intellectual content of the work.

Model	AMOBench_seeds (50)	AMOBench_VeRA_H_1230 (244)	AMOBench_VeRA_H_Pro (50)	BeyondAIME_seeds (100)	BeyondAIME_VeRA_H (500)	BeyondAIME_VeRA_H_Pro (100)
external-api/Gemini-2.5-Pro	28.0	43.5	37.6	57.2	57.7	52.2
GLM-4.6	40.0	52.9	48.0	68.2	67.1	63.6
GPT-5.1-high	56.0	64.6	58.8	71.6	77.3	75.2
Claude-Sonnet-4.5-thinking	18.0	36.0	32.4	51.6	56.3	50.2
Seed-1.6-Thinking-0715	40.0	41.6	38.0	54.4	57.9	51.6
DeepSeek-V3.2-thinking	28.0	51.6	46.0	69.4	68.6	64.2
Seed-1.6-1015-high	48.0	43.1	36.4	58.8	61.8	55.0
Kimi-K2-thinking	22.0	26.5	17.2	54.8	43.8	46.6
Minimax-M2	20.0	36.6	28.4	56.8	52.8	47.6
Gemini-3-Pro-Preview	56.0	61.1	56.0	76.6	72.2	65.4
GPT-5-high	40.0	58.4	54.4	68.0	77.6	70.6
Kimi-K2-0905	8.0	19.9	18.8	37.0	30.4	25.8
qwen3-max-0923	14.0	39.2	35.2	53.6	60.1	52.2
GPT-5.1-chat-latest	6.0	20.7	17.6	28.8	33.0	32.6
Seed-1.6-Lite-1015-high	36.0	39.7	36.4	54.8	55.6	48.6
DeepSeek-V3.1-thinking	48.0	52.1	52.8	71.8	68.6	62.2
Mean (16 models)	**31.8**	**43.0**	**38.4**	**58.3**	**58.8**	**54.0**

Table 16. **VeRA-H / VeRA-H Pro results (Avg@5).** (Part 2/2) Seeds vs hardened variants across AMO-Bench and Beyond-AIME.

Model	GSM8k_seeds (1319)	GSM8k_Vera_E (2638)	AIME2024_seeds_E (30)	AIME2024_VeRA_E (60)	AIME2025_seeds (30)	AIME2025_VeRA_E (60)
Gemini-2.5-Pro	93.9	95.0	88.0	73.3	85.3	73.0
GLM-4.6	94.1	94.8	90.7	71.7	92.0	76.3
GPT-5.1-high	95.6	96.9	92.0	75.3	88.7	88.3
Claude-Sonnet-4.5-thinking	95.3	96.2	84.0	74.0	78.7	68.7
Seed-1.6-Thinking-0715	95.1	95.9	85.3	72.0	80.0	75.3
DeepSeek-V3.2-thinking	94.0	95.4	92.0	79.0	90.0	87.7
Seed-1.6-1015-high	95.6	96.3	89.3	75.0	80.7	76.7
Kimi-K2-thinking	95.0	86.6	86.7	75.3	78.7	71.7
Minimax-M2	94.7	94.9	84.7	55.7	76.7	57.0
Gemini-3-Pro-Preview	94.5	95.0	94.0	81.3	90.7	85.7
GPT-5-high	95.6	97.1	90.0	74.7	89.3	85.3
Kimi-K2-0905	95.0	95.6	65.3	51.3	47.3	44.0
qwen3-max-0923	95.5	96.9	82.7	68.7	74.7	69.7
GPT-5.1-chat-latest	94.3	96.0	48.7	46.0	50.7	41.7
Seed-1.6-Lite-1015-high	95.1	95.8	87.3	73.0	77.3	70.0
DeepSeek-V3.1-thinking	94.5	94.8	90.7	77.7	86.7	83.0
Mean (16 models)	**94.9**	**95.2**	**84.5**	**70.2**	**79.2**	**72.1**

Table 17. **VeRA-E results (Avg@5).** (Part 1/2) Seeds vs equivalent variants across GSM8K and AIME-2024/25.

Model	GPQA_Diamond_seeds (198)	GPQA_Diamond_VeRA_E (871)	BeyondAIME_seeds (100)	BeyondAIME_VeRA_E (100)
Gemini-2.5-Pro	82.1	81.4	57.2	57.0
GLM-4.6	78.8	80.2	68.2	66.4
GPT-5.1-high	86.4	84.5	71.6	73.0
Claude-Sonnet-4.5-thinking	81.0	80.1	51.6	53.0
Seed-1.6-Thinking-0715	78.4	78.5	54.4	55.8
DeepSeek-V3.2-thinking	82.1	83.3	69.4	67.6
Seed-1.6-1015-high	77.4	78.9	58.8	58.2
Kimi-K2-thinking	79.9	83.4	54.8	44.0
Minimax-M2	76.7	74.4	56.8	54.4
Gemini-3-Pro-Preview	86.7	85.7	76.6	74.0
GPT-5-high	81.9	82.1	68.0	69.8
Kimi-K2-0905	73.5	74.2	37.0	34.6
qwen3-max-0923	76.1	76.7	53.6	58.8
GPT-5.1-chat-latest	69.9	69.1	28.8	30.6
Seed-1.6-Lite-1015-high	77.6	76.9	54.8	53.6
DeepSeek-V3.1-thinking	80.0	81.3	71.8	66.0
Mean (16 models)	**79.3**	**79.4**	**58.3**	**57.3**

Table 18. **VeRA-E results (Avg@5).** (Part 2/2) Seeds vs equivalent variants across GPQA-Diamond and Beyond-AIME.