Measuring Progress in Reasoning Toward Mathematical Discovery with Automatic Verification

Erik Y. Wang [*1,2] **Sumeet Ramesh Motwani** [*3] **James V. Roggeveen** [4] **Eliot Hodges** [5] **Dulhan Jayalath** [3] **Charles London** [3] **Kalyan Ramakrishnan** [3] **Cheng Zhang** [6] **Flaviu Cipcigan** [6] **Philip Torr** [3] **Alessandro Abate** [3]

Abstract

Can AI make progress on important, unsolved mathematical problems? Large language models are now capable of sophisticated mathematical and scientific reasoning, but whether they can perform novel research is still widely debated and underexplored. We introduce **HorizonMath**, a benchmark of 113 predominantly unsolved problems spanning eight domains in mathematics and the mathematical sciences, paired with an open-source evaluation framework for automated verification. Our benchmark targets the generator-verifier gap: problems where discovery is hard and requires meaningful mathematical insight, but verification is computationally straightforward. This contrasts with most existing research-level benchmarks, which instead rely on formal proof verification or manual review, both of which are expensive to scale. Because these solutions are unknown, HorizonMath is immune to data contamination, and most state-of-the-art models score near 0%. Using this framework, we find three research problems for which GPT 5.4 Pro proposes novel solutions that either resolve previously open questions or improve on the best-known published results. Across six frontier models, reasoning efficiency and behavior also vary substantially. We release HorizonMath as an open challenge and a growing community resource, where each verified solution is a candidate contribution to the mathematical literature.

[*]Equal contribution [1]Stanford University [2]Benchmark [3]University of Oxford [4]Harvard University [5]Princeton University [6]Ellison Institute of Technology. Correspondence to: Erik Y. Wang <erikwang@stanford.edu>.

The 3^{rd} *AI for Math Workshop* at the 43^{rd} International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.

1. Introduction

Making autonomous mathematical discoveries is a north star in AI research, as mathematics is a key driver of scientific discovery. FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Georgiev et al., 2025) improved several open bounds in combinatorics, geometry, and number theory; frontier LLM-based agents have resolved previously unsolved Erdős problems (Feng et al., 2026; Sothanaphan, 2026); and human-AI collaboration has produced new results across theoretical computer science, physics, and pure mathematics (Knuth, 2026; Guevara et al., 2026; Woodruff et al., 2026). As these capabilities continue to advance, standardized methods are needed to systematically measure AI progress toward autonomous mathematical discovery.

Many mathematics benchmarks exist, but most were designed to test *problem-solving* rather than *discovery*, which requires assessing novel mathematical contributions. This problem-solving framing of applying known techniques to problems with known answers has a natural ceiling—human knowledge—and even the most challenging Olympiad-style and graduate-level benchmarks (Luong et al., 2025; Tsoukalas et al., 2024; Roggeveen et al., 2025; Rein et al., 2024) evaluate problems with known solutions. As such, there exist very few benchmarks that can provide signal on whether AI systems can perform research and produce results not already in the literature.

On the other hand, measuring capabilities on unsolved problems is *naturally difficult* because solutions are unknown, but it is crucial since it requires genuinely novel mathematical contributions. We therefore exploit the generator-verifier gap, a feature in certain classes of problems in which candidate solutions are hard to produce but efficient to check. We identify three such forms that are well-suited to automated evaluation: 1) problems where a closed-form solution is currently unknown, but a candidate expression can be checked against a high-precision numerical reference, 2) construction and optimization problems seeking objects that improve upon a current baseline not known to be optimal, and 3) existence problems where a target object has not been found but a candidate can be validated by checking that it satisfies all required properties. All three classes can be efficiently

verified by deterministic computation, making them natural targets for a scalable benchmark of mathematical discovery (Figure 2).

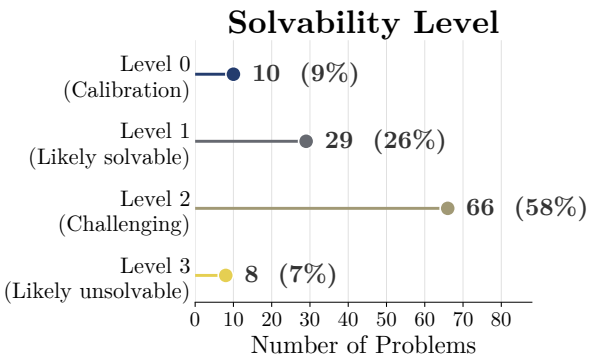
With these classes of problems, we present **HorizonMath**, a benchmark of 113 predominantly unsolved problems and associated verification scripts that automatically validate proposed answers (Figure 1). Our framework is defined by three key contributions:

1. **A Contamination-Resistant Benchmark of Open Problems:** We present a benchmark of 113 problems across eight subject areas in applied and computational mathematics. Because the solutions are unknown, they do not exist in any training corpus, and any correct solution produced by a model would therefore signal genuine reasoning ability and autonomous discovery.
2. **Automated Verification:** Evaluations of mathematical research capabilities are traditionally bottlenecked by human review. We automate verification using high-precision numeric comparison and deterministic constraint-checkers, leveraging the generator-verifier gap to provide a fast and objective signal of correctness.
3. **Open-Source and Standardized Evaluation:** Unlike other benchmarks of research-level mathematics, our framework includes complete problem definitions, ground-truth computations, and verifier scripts, and is publicly available.

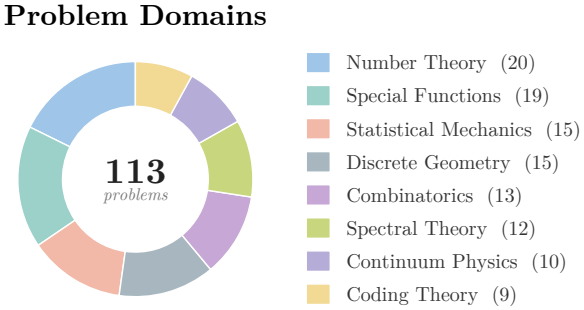
We note that while matching a high-precision numerical reference does not formally prove that a closed-form expression is exactly correct, our combination of numerical comparison and verification of permitted operations provides significant evidence for correct solutions and can effectively filter inadmissible ones. Using GPT 5.4 Pro, we identify three problems for which the model proposes novel solutions that either resolve previously open questions or improve on the best-known published results; these are described in Section 6 and detailed in Appendix A. **HorizonMath** is an open-source effort that welcomes contributions from the community and is available at this anonymous link: `https://anonymous.4open.science/r/HorizonMath-A2026`.

2. Related work

We organize related work into three categories that highlight the contrast between solution and discovery, introduced in Section 1. We first review *solution-driven* mathematical benchmarks, which evaluate models on problems with known answers (Section 2.1). We then survey *research-level* benchmarks, which target unsolved problems and help define our design space, and also note the key differences between these works and **HorizonMath** (Section 2.2). Finally,



(a) By solvability level.



(b) By mathematical domain.

Figure 1. Benchmark composition by solvability level (top) and mathematical domain (bottom). HorizonMath comprises 113 problems, distributed across several areas of mathematics and physics.

we discuss recent demonstrations of AI-driven mathematical discovery, which establish the capabilities that **HorizonMath** seeks to measure (Section 2.3).

2.1. Solution-driven mathematical benchmarks

Early benchmarks like GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) are now saturated, and while recent benchmarks targeting higher difficulty levels, such as GPQA (Rein et al., 2024) and the HARD-Math series (Fan et al., 2025; Roggeveen et al., 2025) for graduate-level problems have emerged, these benchmarks all evaluate problems with *known* solutions and therefore cannot measure novel discovery. Benchmarks based on competitions and Olympiads (e.g., OlympiadBench (He et al., 2024), IMO-Bench (Luong et al., 2025), and MathArena (Balunovic et al., 2025)) have provided new and difficult problems but are typically conducted on an annual basis and do not readily support continuous measurement of research progress, especially as they are all already-solved problems.

Table 1. Comparison of math reasoning benchmarks.

Dataset	Problems	Difficulty	Unsolved	Non-Proof-Centric	Open-evaluation	Auto-verify
FrontierMath (Glazer et al., 2024)	300+	Research	✗	✓	✗	✓
FrontierMath: Open Problems (Epoch AI, 2026)	14	Research	✓	✓	✗	✓
IMProofBench (Schmitt et al., 2025)	39	Research	✗	✗	✗	✗
First Proof (Abouzaid et al., 2026)	10	Research	✗	✗	✗	✗
Erdős Problems (Bloom, 2021)	1,000+	Research	✓	✗	✗	✗
IMO-AnswerBench (Luong et al., 2025)	400	Olympiad	✗	✗	✓	✓
Opt. Constants (Davis et al., 2026)	96	Research	✓	✓	✗	✗
HorizonMath	113	Research	✓	✓	✓	✓

2.2. Research-level benchmarks

In response to the rapidly-advancing capabilities of AI research agents and the saturation of existing benchmarks, several research-level benchmarks have recently appeared. These benchmarks require novel solutions to unsolved problems rather than reproductions of known answers and are typically hand-crafted by experts.

FrontierMath (Glazer et al., 2024) comprises several hundred unpublished mathematics problems spanning difficulty tiers from undergraduate through research level. Its most advanced tier targets research-level mathematics, and its "Open Problems" subset contains 14 unsolved problems, some of which have already been solved and removed from the dataset (Epoch AI, 2026). Moreover, their verifiers are all private; proposed solutions can only be checked by purchasing access to the verifier. IMProofBench (Schmitt et al., 2025) also offers a set of 39 research-level proof problems written by human experts. Both keep their problems and verifiers private or rely heavily on expert human grading, limiting reproducibility.

Similarly, First Proof (Abouzaid et al., 2026) comprises 10 contamination-resistant questions drawn from its authors' unpublished research, but its small scale, manual evaluation, and proof orientation limit its scalability for fast and rigorous model evaluation.

2.3. AI-driven mathematical discovery

One of the first LLM-driven results on open problems came from FunSearch (Romera-Paredes et al., 2024), which evolved programs to construct large cap sets beating the best known bounds in extremal combinatorics and discover improved heuristics for bin-packing. AlphaEvolve (Novikov et al., 2025) generalized this evolutionary approach and made improvements to complex matrix multiplication algorithms and a kissing-number lower bound. AlphaEvolve's results targeted many topics outside of mathematics, so Georgiev, Tao, Gómez-Serrano, and Wagner (Georgiev et al., 2025) applied AlphaEvolve to 67 problems across analysis, combinatorics, geometry, and number theory, rediscovering most best-known constructions and improving several—such as new bounds for the finite field Kakeya problem and an optimal tile arrangement for a problem from the 2025 International Mathematical Olympiad.

In parallel, human-AI collaboration has resolved further open problems: Woodruff et al. (Woodruff et al., 2026) used Gemini Deep Think on problems in theoretical computer science, information theory, and physics, and Knuth (Knuth, 2026) similarly used Claude Opus 4.6 to obtain a closed-form Hamiltonian-cycle decomposition of a Cayley digraph—a longstanding open problem.

Recent systems push even further toward full autonomy. DeepMind's Aletheia (Feng et al., 2026) produced multiple papers with minimal human intervention. OpenAI's GPT-5.2 through 5.5 Pro models have similarly resolved many Erdős problems (Sothanaphan, 2026; Alexeev et al., 2026). In physics, Guevara et al. (Guevara et al., 2026) used GPT-5.2 Pro to conjecture a closed-form formula for single-minus tree-level n-gluon scattering amplitudes in a specific half-collinear kinematic regime which was then verified by the human authors. This rapid growth in research problems solved by AI motivates a standardized and scalable supply of unsolved yet meaningful research problems that can be used to efficiently evaluate such systems.

2.4. Related domains and datasets

While not strictly mathematics, FrontierCS shares our design principles: it provides 100+ open-ended problems across algorithmic optimization and computer science. Each problem is hand-crafted, but none have a known optimal solution, and any proposed solution can be deterministically scored on a 0-100 scale by an automated evaluator. FrontierCS provides expert-authored reference solutions, reproducible evaluation harnesses, and dynamic difficulty scaling. Though not mathematics-focused, its open-ended design, automatic verification, and unsolved nature inspire our work. HeuriGym (Chen et al., 2026) is another algorithmic optimization benchmark with automatic verification, but it also targets combinatorial problem-solving rather than purely mathematical research questions.

Beyond formal benchmarks, curated databases of open problems have also become proving grounds for AI. Bloom's Erdős Problems site (Bloom, 2021) maintains a collection of

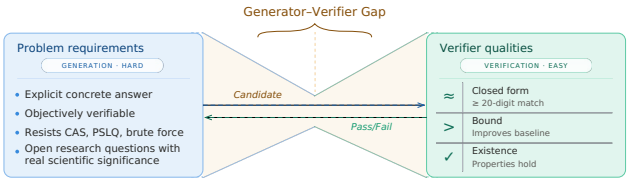


Figure 2. The *generator–verifier gap* of **HorizonMath**. Problems require hard mathematical discovery, while submitted candidates are checked by deterministic verifiers. Formal proofs are not required for any problems in the dataset.

Erdős problems, and the Tao-Ivanisvili-Davis optimization-constants repository lists problems and known bounds but provides no evaluation harness. Finally, another related direction includes the UQ (Unsolved Questions) benchmark (Nie et al., 2025), which evaluates models on unsolved questions sourced from Stack Exchange. However, practical issues such as question ambiguity, verifier brittleness, and the need for community involvement limit its reliability as a true measure of AI progress on unsolved problems.

3. Benchmark design and evaluation

3.1. Design principles

HorizonMath follows four design principles. First, each problem must require an *explicit answer* in the form of a definite mathematical object such as a number, polynomial, set, or graph, rather than a natural-language proof. Second, this answer must be *objectively verifiable* by a simple deterministic computational procedure, whether by numerical comparison against a high-precision reference value, by checking improvement over a published baseline, or by validating that a construction satisfies all required properties. Third, the problem must demand long-horizon *mathematical reasoning* and resist solution by standard computational algorithms, computer algebra systems, numerical optimization, or brute-force search. Fourth, each problem must have *scientific significance*, meaning it is sourced from the mathematical literature or active research rather than artificially constructed for the benchmark.

All problems are sourced from the research literature (such as journal and conference publications, textbooks, and arXiv preprints) across three categories: problems without known closed-form solutions, optimization problems whose current results are not proven optimal, and construction problems where the target object has not been found. AI agents—specifically, Codex and Claude Code—with web search were used to compile a large, initial list of relevant problems, which was iteratively refined to this subset by human experts. All human authors involved in the paper were researchers with a PhD or current PhD students in mathematics, applied mathematics, or computer science. Importantly, this made the collection process significantly cheaper and more efficient than problems in comparable benchmarks, which were all compiled and produced by human experts. Furthermore, since none of the problems in the unsolved categories of the benchmark have a known exact or optimal solution in the required form, contamination of solutions is not possible.

3.2. Dataset composition

We classify each problem along three axes: the structural output type of the expected solution, the solvability level, and the mathematical domain.

Output types: All problems require concrete solutions formulated in Python code rather than natural-language proofs. The majority are constant discovery problems, where the model must identify an exact closed-form expression for a numerically approximated target value. Construction tasks form the second largest category, requiring the model to produce discrete mathematical objects—such as lattices, point configurations, or adjacency matrices—that improve upon existing baselines. The remaining problems involve generating computable functions with specific properties or discovering general formulas.

Solvability levels: We define four levels to gauge difficulty. Level 0 problems are problems that have already been solved by humans (meaning there exist known closed forms or constructions), included for calibration. This does not imply that all models are capable of solving these problems, however. Levels 1–3 are problems currently unsolved by humans and are assigned the following tiers based on the expertise of the authors. Level 1 problems are judged to be likely solvable with known techniques or near-term capabilities; Level 2 problems require significant new insights and are more challenging than Level 1 problems; and Level 3 problems are problems whose potential solutions would be considered major breakthroughs. Since these assignments were made by authors of the benchmark who have mathematical experience in the relevant subfields, they are not objective and are intended as a guide rather than a precise difficulty score.

Mathematical domains: The benchmark contains problems across eight subject areas: *Number Theory*, *Special Functions*, *Statistical Mechanics*, *Discrete Geometry*, *Combinatorics*, *Spectral Theory*, *Continuum Physics*, and *Coding Theory*. Each problem is assigned to its corresponding mathematical subject. The distribution in Figure 1 reflects the current snapshot, but since it is public and welcomes new contributions from the community, its composition will evolve.

3.3. Admissible solutions for closed forms

For closed-form discovery problems, we enforce strict structural requirements. While the definition of a closed-form is somewhat field-dependent, we adopt a fairly open-ended convention: it must be a finite symbolic expression built exclusively from the following set of operations: rationals, algebraic numbers, standard transcendentals (π, e, Euler's γ, Catalan's G), elementary functions (exp, log, trigonometric, and hyperbolic functions), and special functions as long as they are defined in `mpmath` (Borwein & Crandall, 2013). We choose this flexible definition given the lack of a universal agreement on what "closed-form" means.

We explicitly forbid solutions that include unevaluated integrals, infinite series, limits, implicit definitions, and numerical approximations presented as exact values—except for those that are special functions. We also do not permit solutions that use computational tools such as numerical integration, truncated infinite series, numerical root-finding, and computation of resultants by evaluating one polynomial at the roots of another. We also exclude problem classes that can be solved via standard deterministic algorithms, such as the Remez algorithm, PSLQ, gradient descent, or symbolic integration. Finally, we discourage symbolic parameter fitting to the ground-truth numerical solution by only showing at most five significant figures of precision in the prompt.

4. Automated verification

A core feature of our benchmark is a fully automated and reproducible evaluation pipeline. Unlike theorem-proving benchmarks, which require complex formalization in proof assistants or expert grading, and unlike other verifiable benchmarks that do not release their evaluation infrastructure, our framework enables fast, automatic verification. We note that while proof assistants like Lean also provide automatic verification, using them typically requires the theorem to be very carefully written in the corresponding language, which can be very challenging depending on the problem and domain.

4.1. Execution pipeline and admissibility

Models are prompted to generate a self-contained Python function, `def proposed_solution()`, returning the solution as either an `mpmath` symbolic expression or a JSON-serializable dictionary representing a discrete construction. Strict admissibility criteria are enforced on the proposed solutions in the system prompt as well as via the compliance checker. Closed-form expressions must be finite symbolic combinations of permissible operations (rational numbers, elementary functions, standard transcendentals at algebraic arguments); infinite series, limits, implicit definitions, and numerical approximations are forbidden. Explicit constructions must follow the output type defined in the function template. These constraints are enforced via an LLM-based compliance checker, which inspects the proposed solution for forbidden operations such as numerical root-finding, quadrature, and hard-coded constants.

4.2. Evaluation modes and scoring mechanics

Closed-form problems require the model to produce a symbolic closed-form expression, evaluated to high precision via `mpmath` and compared against a reference value typically computed via integration or series expansion. A solution is accepted if it matches the reference to $\min(20, D)$ decimal digits, where D is the number of verified digits available for that ground truth. **Bound-improvement** problems target optimization tasks where the global optimum is unknown but a published baseline exists. The model's output is executed and scored by problem-specific validators on the relevant metric (e.g., packing density, asymptotic bounds). A solution passes if it strictly improves upon the best-known result, and solutions are further ranked by relative improvement over the baseline. **Existence** problems concern construction questions, such as finding a Hadamard matrix of an unresolved order or a set of mutually orthogonal Latin squares of a given size. Scoring is pass/fail: deterministic, problem-specific validators exhaustively check whether the proposed object satisfies all required properties, with no baseline comparison.[1]

We acknowledge that while matching a high-precision reference value does not formally prove a closed-form expression is exactly correct, *such solutions that pass the permissibility check can be regarded as high-confidence conjectures until proven*. While a formal proof would be needed to guarantee the correctness, this numerical check remains essential to producing novel mathematical research, since new theorems often first begin with highly-accurate conjectures. Constructions that beat a published bound or satisfy the corresponding validator, by contrast, are verified deterministically by checking all required properties.

5. Representative problem examples

We illustrate the benchmark with two representative problems covering the `closed-form` and `bound-improvement`; the examples below show the core task and output specification given to the model. An example of an `existence` problem with full prompt boilerplate is in Appendix A.4.

[1] In the released dataset these modes correspond to the identifiers `ground_truth_computable`, `benchmark_best_known`, and `new_construction`, respectively.

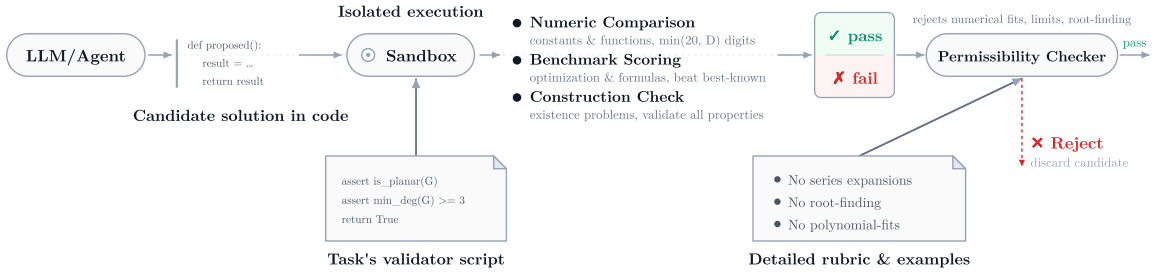


Figure 3. **HorizonMath**'s automated evaluation pipeline. A model generates a proposed solution, which is screened by a compliance checker that rejects forbidden operations. Admissible solutions are routed to one of three evaluation modes: numeric comparison against a ground-truth value, benchmark scoring for improvement over the best-known result, and construction checking to validate all required structural properties.

1. Spinor Norm Elliptic Integral (I_0)

Special Functions | *Constant* | `Closed-Form` | *Solvability 1*

Task. Let $K(m)$ and $E(m)$ denote the complete elliptic integrals of the first and second kinds in the parameter convention. Define

$$I_0 = \frac{4\sqrt{2}}{\pi} \int_0^1 \frac{1+\sqrt{z}}{(1+z)^3} \left(2E(z) - (1-z)K(z)\right) dz.$$

This elliptic-integral constant arises in a spinor norm calculation and has numerical value $I_0 \approx 1.77425\ldots$, but no closed-form expression is given in (Takahashi, 2024). Find a closed-form symbolic expression for I_0.

Output format. An expression using only mpmath-permitted functions. No quadrature, infinite series, root-finding, or hard-coded numerical constants. (Exact output format specified in Appendix A.1.)

2. Minimum-Scope Difference Triangle Set $(7, 5)$

Combinatorics & Design Theory | *Construction* | `Bound-Improvement` | *Solvability 1*

Task. An (n, k)-DTS is an $n \times (k+1)$ integer array A with rows strictly increasing from 0 whose positive within-row differences $\{a_{i,j} - a_{i,j'} : j' < j\}$ are pairwise distinct across all rows; the *scope* is $m(A) = \max_{i,j} a_{i,j}$, with applications in coding and communications (Shehadeh et al., 2026). Find a valid $(7, 5)$-DTS with scope strictly below the published bound $m(7, 5) \leq 112$, i.e. $m(A) \leq 111$.

Output format. A dictionary with integer entries and strictly increasing rows. The validator computes the scope and verifies pairwise distinct differences exactly.

6. Evaluation results

We evaluate six frontier models on HorizonMath: three closed models (**Claude Opus 4.6**, **Gemini 3.1 Pro**, and **GPT 5.4 Pro**) and three open-weight models (**Kimi K2.6**, **GLM 5.1**, and **DeepSeek V4 Pro**). For GPT 5.4 Pro and Gemini 3.1 Pro, the reasoning effort is set to high. Claude Opus 4.6 is run with its maximum output-token limit. The open-weight models are run with maximum reasoning effort and a 128k token limit. All evaluations are pass@1 due to the high costs of running frontier models on problems of this difficulty, except on questions where GPT 5.4 Pro produced a verified solution on its first attempt, where we ran one additional query with GPT 5.4 Pro to see whether we could obtain a larger improvement over the baseline.

On solvability 1–3 problems, the only model that makes any progress is **GPT 5.4 Pro**, which proposes novel solutions on three problems in the solvability 1 tier: the `Thin-Triangle Kakeya (128 slopes)` and `Asymptotic Upper Bound Constant for Diagonal Ramsey Numbers` optimization problems, where its constructions improve on the best-known published baselines, and the `Spinor-norm integral` (I_0) closed-form problem, where it produces a symbolic expression matching the high-precision numerical reference. The problems and proposed solutions have been verified by domain experts and are detailed in Appendix A. Neither the other closed models (Claude Opus 4.6, Gemini 3.1 Pro) nor any of the open-weight models (Kimi K2.6, GLM 5.1, DeepSeek V4 Pro) produced candidate solutions that passed the compliance checker or verifiers on the unsolved tiers.

On the 10 solvability-0 calibration problems, performance varies considerably. Among the open-weight models, Kimi K2.6 solves 1, GLM 5.1 solves 2, and DeepSeek V4 Pro solves 3. Among the closed models, Claude Opus 4.6 and Gemini 3.1 Pro each solve 3, while GPT 5.4 Pro correctly solves 5. We evaluate the strongest models from each provider and use Gemini-3.1-Pro since Gemini-3.1-Deep-Think does not provide API access. We analyze these results in Section 7.

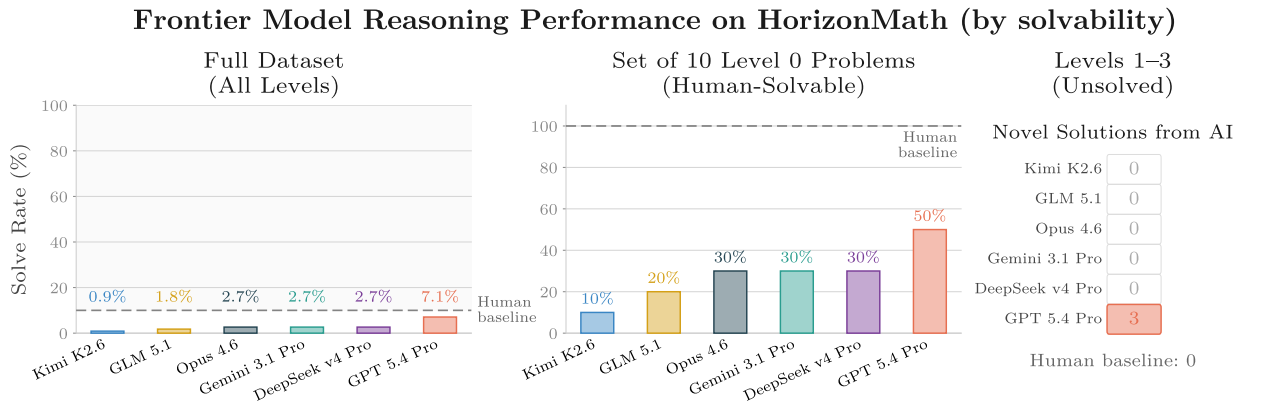


Figure 4. Model performance on **HorizonMath** across six frontier models: Kimi K2.6, GLM 5.1, DeepSeek V4 Pro, Claude Opus 4.6, Gemini 3.1 Pro, and GPT 5.4 Pro. The dashed line (human baseline) indicates Level 0 problems with known solutions. *Left:* Solve rate on the full benchmark (113 problems). GPT 5.4 Pro leads at 7.1% (8 of 113), comprising five problems from the calibration tier (solvability 0) and three from the unsolved tiers (solvability 1–3). *Center:* Solve rate restricted to the ten solvability-0 calibration problems, all of which are solvable by humans. *Right:* Number of admissible candidate solutions on the unsolved tiers (solvability 1–3). GPT 5.4 Pro is the only model to produce novel solutions. We discuss these results further in Section 6 and Appendix A.

7. Analysis

Here, we study the reasoning behavior across models. We find notable differences in token and cost efficiency, and compare open-source models against closed-source ones.

7.1. Model reasoning and behavior

In Figure 5, we contrast the reasoning traces of GLM 5.1 on two solvability-0 problems: a correct response uses 13,776 tokens, while an incorrect one on a similarly difficult problem uses 53,593, nearly four times as many. This pattern holds across all six models, with average tokens on incorrect runs uniformly exceeding those on correct ones (**24.7k vs 63.0k** for DeepSeek V4 Pro, **18.2k vs 62.1k** for GLM 5.1, **43.5k vs 59.2k** for Kimi K2.6, **16.3k vs 23.1k** for Gemini 3.1 Pro, **43.8k vs 82.4k** for Claude Opus 4.6, and **30.1k vs 40.3k** for GPT 5.4 Pro). In Appendix B, we measure the rate at which open-weight models switch reasoning strategies. We find that this rate increases with problem difficulty within each model, and that weaker models switch more often than stronger ones.

The composition of reasoning tokens also looks different between correct and incorrect runs. We classify each segment of the open-weight models' traces into one of six categories (verification, pursuing/solving a path, backtracking, initial planning, exploration, or finding errors) and observe that correct responses spend a greater share of tokens on planning and verification, while incorrect ones spend more on exploration. This is consistent with the pattern observed above, where incorrect runs use more tokens overall by trying more exploration strategies. We discuss reasoning behaviors, especially related to models giving up early, in Appendix B.4.

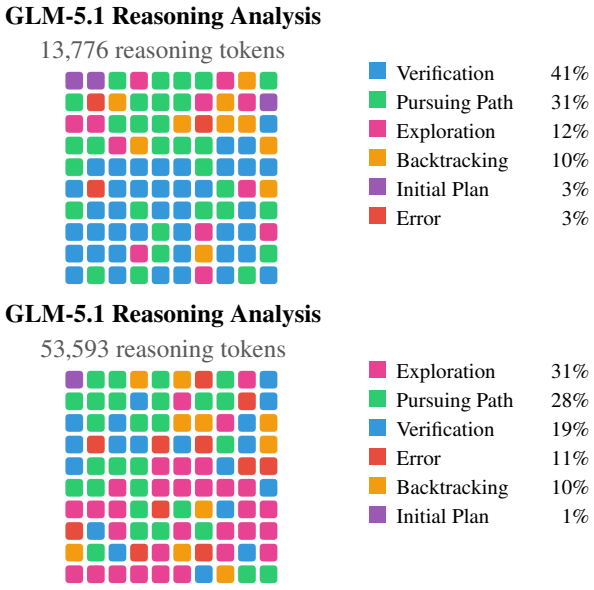


Figure 5. Reasoning trace analysis for GLM 5.1. A correct response to a solvable problem uses fewer tokens compared to an incorrect response to a similarly difficult problem, which involves significantly more exploration attempts and errors along with less verification. Read left-to-right, top-to-bottom; each square represents 1% of the model's reasoning tokens for that problem.

Finally, we count solutions that pass naive numerical verification but are flagged by the compliance checker for operations not permitted by the admissibility criteria. GPT 5.4 Pro produces by far the most non-compliant solutions (43), followed by Claude Opus 4.6 (13), Gemini 3.1 Pro (10), DeepSeek V4 Pro (6), GLM 5.1 (5), and Kimi K2.6 (4). GPT 5.4 Pro's high count is consistent with its higher overall attempt rate. These rates illustrate why compliance checking is necessary: without it, numerical agreement

alone would significantly overstate model capability.

7.2. Token and cost efficiency

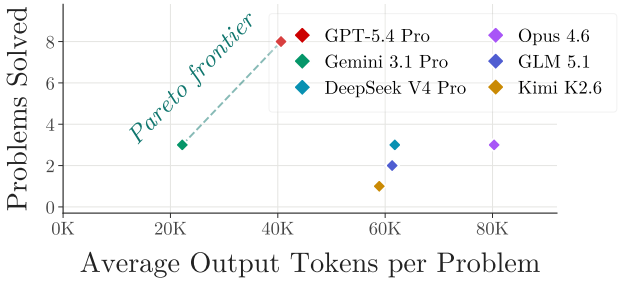


Figure 6. The Pareto frontier of token efficiency versus accuracy across models.

Model	Tokens	Total $	Tok. Eff.	$/Solve
GLM 5.1	61.3k	24.24	0.289	12.12
DeepSeek V4 Pro	61.8k	24.44	0.430	**8.15**
Kimi K2.6	58.9k	23.30	0.150	23.30
Opus 4.6	80.3k	226.85	0.331	75.62
Gemini 3.1 Pro	22.2k	30.10	1.196	10.03
GPT-5.4 Pro	40.6k	825.84	**1.744**	103.23

Table 2. Model efficiency across tokens and cost. *Tokens* is average output tokens per problem; *Tok. Eff.* is solved problems per 1M tokens.

Figure 6 and Table 2 characterize the accuracy-cost tradeoff across the six models. The Pareto frontier on solved problems versus average output tokens is defined by two closed models, Gemini 3.1 Pro (3 problems at 22.2k tokens on average) and GPT 5.4 Pro (8 at 40.6k); the other four lie below it. Token efficiency (solved problems per 1M tokens) follows the same ordering: GPT 5.4 Pro leads at 1.744 and Gemini 3.1 Pro at 1.196, while the open-weight models fall between 0.150 and 0.430. However, for cost per solution, DeepSeek V4 Pro is lowest at $8.15, followed by Gemini 3.1 Pro ($10.03) and GLM 5.1 ($12.12), with Claude Opus 4.6 ($75.62) and GPT 5.4 Pro ($103.23) substantially higher.

8. Discussion

We introduced **HorizonMath**, a benchmark of over 100 mathematical problems from several subdomains of mathematics, physics, and computer science that are predominantly unsolved and sourced from the research literature. Each problem outside of the calibration set lacks a known solution but can automatically be verified with high probability, eliminating the possibility of data contamination and ensuring that a correct solution would constitute a novel contribution to the literature. The evaluation framework is fully computational, avoiding the scalability bottlenecks of formal proof verification and the subjectivity of expert review, and is publicly available.

We acknowledge key limitations. Matching a high-precision numerical reference, even to 20 decimal digits, does not formally prove that a closed-form expression is exactly correct; such solutions are best regarded as strong conjectures until proven. Our compliance checker, which uses an LLM to detect forbidden operations in proposed solutions, is similarly imperfect: it may occasionally accept solutions that exploit subtle loopholes or reject valid ones that use unusual but legitimate constructions. HorizonMath therefore does not entirely replace expert verification, but substantially reduces the human effort required by automatically filtering incorrect or inadmissible solutions.

We see two complementary directions for extending HorizonMath: (1) expanding what counts as an admissible solution, and (2) expanding the class of problems themselves. The first direction could involve accepting solutions that are simplifications but not necessarily exact closed forms according to the definition used in this benchmark. This flexibility would help capture a fuller spectrum of research, especially in fields like physics, where the goal is not always an exact closed-form solution but a reduction in complexity that increases computational tractability. The second direction would expand beyond the existing problem categories to include open problems requiring proof-based verification, by integrating with formal verification systems such as Lean. This would require carefully producing theorem statements in Lean to begin with, but would enable HorizonMath to encompass a broader class of unsolved problems, including those where the contribution is a proof rather than a concrete object, while maintaining our automated and reproducible evaluation strategy. Overall, HorizonMath benchmarks frontier models on unsolved research problems with automated verification, and improvements on it reflect real advances in mathematical reasoning capabilities.

Impact Statement

Our work aims to advance AI for mathematical discovery. Ideally, our benchmark aids the development of models that ultimately accelerate human-driven mathematical research rather than entirely automate it. We focus on reasoning rather than problem-generation, which remains a crucial area requiring human expertise.

References

Abouzaid, M., Blumberg, A. J., Hairer, M., Kileel, J., Kolda, T. G., Nelson, P. D., Spielman, D., Srivastava, N., Ward, R., Weinberger, S., et al. First proof. *arXiv preprint arXiv:2602.05192*, 2026.

Alexeev, B., Barreto, K., Li, Y., Lichtman, J. D., Price, L., Shah, J. I., Tang, Q., and Tao, T. Primitive sets and von mangoldt chains: Erdős problem #1196 and beyond, 2026.

URL https://arxiv.org/abs/2605.00301.

Balunovic, M., Dekoninck, J., Petrov, I., Jovanović, N., and Vechev, M. Matharena: Evaluating llms on uncontaminated math competitions. *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Bloom, T. F. Erdős problems. https://www.erdosproblems.com/, 2021. Accessed: 2026-03-03.

Borwein, J. M. and Crandall, R. E. Closed forms: What they are and why we care. *Notices of the AMS*, 60(1), 2013.

Chen, H., Wang, Y., Cai, Y., Hu, H., Li, J., Huang, S., Deng, C., Liang, R., Kong, S., Ren, H., Samaranayake, S., Gomes, C. P., and Zhang, Z. Heurigym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=HWxHUO15Yy.

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

Davis, D., Ivanisvili, P., Tao, T., and contributors. Optimization constants in mathematics. GitHub repository, 2026. URL https://github.com/teorth/optimizationproblems.

Epoch AI. Frontiermath: Open problems, 2026. URL https://epoch.ai/frontiermath/open-problems. Accessed: 2026-03-06.

Fan, J., Martinson, S., Wang, E. Y., Hausknecht, K., Brenner, J., Liu, D., Peng, N., Wang, C., and Brenner, M. Hardmath: A benchmark dataset for challenging problems in applied mathematics. In *The Thirteenth International Conference on Learning Representations*, 2025.

Feng, T., Trinh, T., Bingham, G., Kang, J., Zhang, S., Kim, S.-h., Barreto, K., Schildkraut, C., Jung, J., Seo, J., et al. Semi-autonomous mathematics discovery with gemini: A case study on the erdos problems. *arXiv preprint arXiv:2601.22401*, 2026.

Feng, Y., Kempe, J., Zhang, C., Jain, P., and Hartshorn, A. What characterizes effective reasoning? revisiting length, review, and structure of cot, 2025. URL https://arxiv.org/abs/2509.19284.

Georgiev, B., Gómez-Serrano, J., Tao, T., and Wagner, A. Z. Mathematical exploration and discovery at scale. *arXiv preprint arXiv:2511.02864*, 2025.

Glazer, E., Erdil, E., Besiroglu, T., Chicharro, D., Chen, E., Gunning, A., Olsson, C. F., Denain, J.-S., Ho, A., Santos, E. d. O., et al. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai. *arXiv preprint arXiv:2411.04872*, 2024.

Guevara, A., Lupsasca, A., Skinner, D., Strominger, A., and Weil, K. Single-minus gluon tree amplitudes are nonzero. *arXiv preprint arXiv:2602.12176*, 2026.

He, C., Luo, R., Bai, Y., Hu, S., Thai, Z., Shen, J., Hu, J., Han, X., Huang, Y., Zhang, Y., et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Knuth, D. E. Claude's cycles. Preprint revised 02 March 2026, March 2026. URL https://www-cs-faculty.stanford.edu/~knuth/papers/claude-cycles.pdf.

Luong, M.-T., Hwang, D., Nguyen, H. H., Ghiasi, G., Chervonyi, Y., Seo, I., Kim, J., Bingham, G., Lee, J., Mishra, S., et al. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35406–35430, 2025.

Nie, F., Liu, K., Wang, Z., Sun, R., Liu, W., Shi, W., Yao, H., Zhang, L., Ng, A. Y., Zou, J., et al. Uq: Assessing language models on unsolved questions. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*, 2025.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Pipis, C., Garg, S., Kontonis, V., Shrivastava, V., Krishnamurthy, A., and Papailiopoulos, D. Wait, wait, wait... why do reasoning models loop?, 2025. URL https://arxiv.org/abs/2512.12895.

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. In *First conference on language modeling*, 2024.

Roggeveen, J. V., Wang, E. Y., Ettel, D., Flintoft, W., Donets, P., Ward, R., Roman, A., Graf, A. M., Dandavate, S., Williamson, A., et al. Hardmath2: A benchmark for applied mathematics built by students as part of a graduate class. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

Schmitt, J., Bérczi, G., Dekoninck, J., Feusi, J., Gehrunger, T., Appenzeller, R., Bryan, J., Canova, N., de Wolff, T., Gaia, F., et al. Improofbench: Benchmarking ai on research-level mathematical proof generation. *arXiv preprint arXiv:2509.26076*, 2025.

Shehadeh, M., Kingsford, W., and Kschischang, F. R. New difference triangle sets by a field-programmable gate array-based search technique. *Journal of Combinatorial Designs*, 34(1):37–50, 2026.

Sothanaphan, N. Resolution of erdos problem# 728: a writeup of aristotle's lean proof. *arXiv preprint arXiv:2601.07421*, 2026.

Takahashi, D. A. Basis of spinors expressed by differential forms and calculating its norm. *arXiv preprint arXiv:2403.13003*, 2024.

Tsoukalas, G., Lee, J., Jennings, J., Xin, J., Ding, M., Jennings, M., Thakur, A., and Chaudhuri, S. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition. *Advances in Neural Information Processing Systems*, 37:11545–11569, 2024.

Woodruff, D. P., Cohen-Addad, V., Jain, L., Mao, J., Zuo, S., Bateni, M., Branzei, S., Brenner, M. P., Chen, L., Feng, Y., et al. Accelerating scientific research with gemini: Case studies and common techniques. *arXiv preprint arXiv:2602.03837*, 2026.

Xiang, V., Snell, C., Gandhi, K., Albalak, A., Singh, A., Blagden, C., Phung, D., Rafailov, R., Lile, N., Mahan, D., Castricato, L., Franken, J.-P., Haber, N., and Finn, C. Towards system 2 reasoning in llms: Learning how to think with meta chain-of-thought, 2025. URL https://arxiv.org/abs/2501.04682.

Appendix

A. Detailed problem statements and solutions

We detail three problems of the `solvability 1` tier for which GPT-5.4 Pro produced a novel closed-form solution or an optimization that beat the best known baseline.

A.1. Spinor Norm Elliptic Integral

Domain: Special Functions | **Output Type:** Constant
Evaluation Mode: `ground_truth_computable` | **Solvability:** 1
Background: This benchmark asks for a symbolic closed form for an elliptic-integral constant arising in a spinor norm calculation. The integral representation involves complete elliptic integrals of the first and second kinds, and the source paper derives the representation but does not provide a closed form.
Source: Takahashi (2024), *Basis of spinors expressed by differential forms and calculating its norm*, arXiv:2403.13003.

Model Prompt

Consider the following elliptic-integral expression arising in a spinor norm calculation.

Let $K(m)$ and $E(m)$ denote the complete elliptic integrals of the first and second kinds with parameter m. Define

$$I_0 = \frac{4\sqrt{2}}{\pi} \int_0^1 \frac{1+\sqrt{z}}{(1+z)^3} \left(2E(z) - (1-z)K(z)\right) dz.$$

Task: Find a closed-form symbolic expression for the constant I_0.

Inadmissible approaches:

- Numerical quadrature or numerical integration routines such as `mp.quad`.
- Returning this integral, an equivalent transformed integral, or a custom integral transform.
- Hardcoding the decimal value or using a tuned approximation.
- Infinite series or truncations whose accuracy depends on the number of terms.

REQUIRED OUTPUT FORMAT:

```
def proposed_solution():
    from mpmath import mp
    mp.dps = 100
    result = ...
    return result
```

Generated Code

```
def proposed_solution():
    from mpmath import mp
    mp.dps = 100
    result = (
        mp.gamma(mp.mpf(1)/4)**2 / (8 * mp.sqrt(mp.pi))
```

```
        + mp.gamma(mp.mpf(3)/4)**2 / mp.sqrt(mp.pi)
    )
    return result
```

Closed-form symbolic solution.

$$I_0 = \frac{\Gamma\left(\frac{1}{4}\right)^2}{8\sqrt{\pi}} + \frac{\Gamma\left(\frac{3}{4}\right)^2}{\sqrt{\pi}}.$$

Numerically, $I_0 \approx 1.774250423444665045...$ matches the ground-truth computed formula to 140 digits of precision, as defined in our numerical validation script.

Approach. Consider $I_0 = 2E\left(\frac{1}{2}\right) - \frac{1}{2}K\left(\frac{1}{2}\right)$. Using the self-complementary Legendre relation at $m = \frac{1}{2}$,

$$I_0 = \frac{\Gamma\left(\frac{1}{4}\right)^2}{8\sqrt{\pi}} + \frac{2\pi^{3/2}}{\Gamma\left(\frac{1}{4}\right)^2},$$

we get this compact derivation:

$$2E(t^2) - (1-t^2)K(t^2) = (1+t)\,E\left(\tfrac{4t}{(1+t)^2}\right)$$

(Landen), so with $z = t^2$ and then $u = \frac{4t}{(1+t)^2}$,

$$\begin{aligned} I_0 &= \frac{4\sqrt{2}}{\pi}\int_0^1 \frac{u\,E(u)}{(2-u)^3\sqrt{1-u}}\,du \\ &= \frac{8\sqrt{2}}{\pi}\int_0^1 \frac{1-x^2}{(1+x^2)^3}\,E(1-x^2)\,dx. \end{aligned}$$

If $A(x) = E(1-x^2)$, $B(x) = K(1-x^2)$, then

$$\frac{d}{dx}\frac{x(A-B)}{(1+x^2)^2} = \frac{2(1-x^2)A + (3x^2-1)B}{(1+x^2)^3},$$

hence

$$\begin{aligned} I_0 &= \frac{2\sqrt{2}}{\pi}\,F''(1), \\ F(k) &:= \int_0^1 \frac{k\,K(1-x^2)}{k^2+x^2}\,dx. \end{aligned}$$

Now the Beltrami transform plus the imaginary-modulus formula gives

$$F(k) = \frac{\pi}{2\sqrt{1+k^2}}\,K\left(\frac{1}{1+k^2}\right).$$

Differentiating twice and setting $k = 1$ yields

$$F''(1) = \frac{\pi}{4\sqrt{2}}\left(4E\left(\tfrac{1}{2}\right) - K\left(\tfrac{1}{2}\right)\right),$$

so

$$I_0 = 2E\left(\tfrac{1}{2}\right) - \frac{1}{2}\,K\left(\tfrac{1}{2}\right).$$

A.2. Thin-Triangle Kakeya (128 slopes): Minimize Union Area

Domain: Geometry & Discrete Geometry | **Output Type:** Construction
Evaluation Mode: `benchmark_best_known` | **Solvability:** 1
Background: This benchmark is a discrete, thickened Kakeya-type construction in the style of Schoenberg and Keich. The classical Kakeya needle problem asks for the smallest-area planar set containing a unit line segment in every direction; Besicovitch showed such sets can have measure zero. Keich studied a thickened variant with N equally-spaced slopes, where each segment is replaced by a thin triangle of base-width $\delta = 1/N$, and sought to minimize the area of their union. AlphaEvolve (Google DeepMind, 2025) found a construction with union area ≈ 0.11481, improving on Keich's earlier construction (≈ 0.11921).
Source: AlphaEvolve (Google DeepMind, 2025). Baseline from the AlphaEvolve construction; see Novikov et al. (2025), *arXiv preprint* arXiv:2506.13131.

Model Prompt

Consider the following optimization problem.

Thin-Triangle Kakeya (128 slopes): Minimize Union Area

Definition: Fix $N = 128$ and $\delta = 1/128$. For each $i = 0, 1, \ldots, 127$, specify a unit line segment

$$\ell_i = \{(x,\ a_i x + b_i) : x \in [0, 1]\},$$

with slope $a_i = i/128$. From each segment ℓ_i define the thin triangle $R_\delta(\ell_i)$ as follows:

- The **upper edge** is ℓ_i.
- The **lower edge** is the segment from $(0,\ b_i - \delta)$ to $(1,\ a_i + b_i)$.
- The **vertical edge** closes the triangle at $x = 0$.

Equivalently, for $x \in [0, 1]$, the vertical cross-section of $R_\delta(\ell_i)$ is the interval

$$y \in \big[\, a_i x + b_i - \delta(1 - x),\ \ a_i x + b_i \,\big].$$

The output defines the set $E = \bigcup_{i=0}^{127} R_\delta(\ell_i)$.
Goal: MINIMIZE $\mathrm{Area}(E)$.
Current State-of-the-Art:

- **Metric:** $\mathrm{Area}(E)$
- **Best Known Value:** ≈ 0.11481 (AlphaEvolve, Google DeepMind, 2025)
- **Direction:** MINIMIZE (lower area is better)
- **Source:** Novikov et al. (2025), arXiv:2506.13131

Mathematical framework. The slopes are fixed at $a_i = i/128$ for $i = 0, \ldots, 127$. The only free parameters are the 128 intercepts $b_0, b_1, \ldots, b_{127}$. Each thin triangle is the convex hull

$$R_\delta(\ell_i) = \mathrm{conv}\big\{(0,\ b_i - \delta),\ (0,\ b_i),\ (1,\ a_i + b_i)\big\}.$$

The area of the union $E = \bigcup_i R_\delta(\ell_i)$ is computed by exact piecewise-linear integration of the union of cross-sections at each x-coordinate.

The AlphaEvolve baseline achieves area $\approx 0.1148103258186177$. The AlphaEvolve triangles $\mathrm{conv}\{(x_i, 0),\ (x_i + i/128, 0),\ (x_i + (i+1)/128, 1)\}$ map to the formulation above via the area-preserving coordinate swap $(x, y) \to (y, x)$ with $b_i = x_i + i/128$.

To beat the baseline, find intercepts giving $\mathrm{Area}(E) < 0.1148103258186177$.

REQUIRED OUTPUT FORMAT:

```
def proposed_solution():
    # Must output b_i for each slope i/128.
    return {
        "intercepts": [b_0, b_1, ..., b_127]
    }
```

- `intercepts`: a list of exactly 128 floats $[b_0, b_1, \ldots, b_{127}]$.
- Slopes are fixed to $a_i = i/128$.
- The validator computes $\mathrm{Area}(E)$ by exact piecewise-linear integration of union cross-sections (deterministic).

Return the dictionary.

Generated Code

```
def proposed_solution():
        N = 128
        INV_Q_UNIT = 1.0 / 8.0   # q-values are stored in eighths
        INV_B_UNIT = 1.0 / 1024.0  # since b_i = q_i / 128, and q_i is in eighths

        # Full Haar-style dyadic basis:
        # parameter (k, s, e) subtracts coeff on indices [s, e),
        # where k is the binary level (higher k = coarser block).
        params = []
        for k in range(6, -1, -1):
            half = 1 << k
            block = half << 1
            for start in range(0, N, block):
                params.append((k, start + half, start + block))

        def precompute_grid(M):
            rows = []
            for j in range(M):
                x = (j + 0.5) / M
                lower = [((i + 1) * x - 1.0) for i in range(N)]
                upper = [(i * x) for i in range(N)]
                rows.append((lower, upper))
            return rows, 1.0 / (M * N)

        # Search grids: coarse / medium / fine
        g64 = precompute_grid(64)
        g192 = precompute_grid(192)
        g384 = precompute_grid(384)

        def approx_area(q_units, grid):
            rows, scale = grid
            total = 0.0
            inv = INV_Q_UNIT
            for lower, upper in rows:
```

```
                intervals = [(q_units[i] * inv + lower[i], q_units[i] * inv + upper
[i]) for i in
 range(N)]
                intervals.sort()
                cl, cr = intervals[0]
                acc = 0.0
                for l, r in intervals[1:]:
                    if l <= cr:
                        if r > cr:
                            cr = r
                    else:
                        acc += cr - cl
                        cl, cr = l, r
                total += acc + (cr - cl)
            return total * scale

        def q_from_coeffs(coeff_units):
            q = [0] * N
            for cu, (_, s, e) in zip(coeff_units, params):
                if cu:
                    for i in range(s, e):
                        q[i] -= cu
            return q

        def q_from_weights(weight_units):
            coeff_units = [weight_units[k] for (k, _, _) in params]
            return q_from_coeffs(coeff_units)

        def optimize_weights(seed_units, coarse_grid, eval_grid):
            # Monotone nondecreasing level-weights as a stable seed search.
            w = list(seed_units)
            cur = approx_area(q_from_weights(w), coarse_grid)
            for step in (32, 16, 8, 4, 2, 1):  # 4,2,1,1/2,1/4,1/8
                improved = True
                while improved:
                    improved = False
                    for k in range(6, -1, -1):
                        best_v = w[k]
                        best_a = cur
                        for delta in (-step, step):
                            nv = w[k] + delta
                            if nv < 0:
                                continue
                            if k > 0 and nv < w[k - 1]:
                                continue
                            if k < 6 and nv > w[k + 1]:
                                continue
                            old = w[k]
                            w[k] = nv
                            a = approx_area(q_from_weights(w), coarse_grid)
                            w[k] = old
                            if a < best_a - 1e-9:
                                best_a = a
                                best_v = nv
                        if best_v != w[k]:
                            w[k] = best_v
                            cur = best_a
                            improved = True
            return approx_area(q_from_weights(w), eval_grid), w

        def optimize_coeffs(coeff_units, q_units, stages):
            for step, kmin, grid, passes in stages:
                active = [idx for idx, (k, _, _) in enumerate(params) if k >= kmin]
```

```
                cur = approx_area(q_units, grid)
                for pnum in range(passes):
                    order = active if (pnum % 2 == 0) else list(reversed(active))
                    for idx in order:
                        _, s, e = params[idx]
                        best_delta = 0
                        best_area = cur
                        for delta in (-step, step):
                            for i in range(s, e):
                                q_units[i] -= delta
                            a = approx_area(q_units, grid)
                            for i in range(s, e):
                                q_units[i] += delta
                            if a < best_area - 1e-9:
                                best_area = a
                                best_delta = delta
                        if best_delta:
                            for i in range(s, e):
                                q_units[i] -= best_delta
                            coeff_units[idx] += best_delta
                            cur = best_area
            return coeff_units, q_units

        try:
            # Several structured seeds (in eighths)
            seeds = [
                [8, 8, 10, 16, 26, 42, 74],    # harmonic-ish
                [2, 6, 10, 22, 42, 86, 170],   # near linear i/3
                [8, 8, 16, 24, 40, 64, 104],   # Fibonacci-ish
                [8, 8, 16, 32, 48, 80, 144],   # more aggressive large-scale
overlap
                [4, 8, 12, 20, 36, 64, 120],   # intermediate alternative
            ]

            # Pick the best globally-optimized seed.
            best_eval = float("inf")
            best_weights = None
            for seed in seeds:
                score, w = optimize_weights(seed, g64, g192)
                if score < best_eval:
                    best_eval = score
                    best_weights = w

            coeff_units = [best_weights[k] for (k, _, _) in params]
            q_units = q_from_coeffs(coeff_units)

            # Hierarchical local search in the full dyadic basis.
            stages = [
                (32, 4, g64, 2),   # step 4 on very coarse blocks
                (16, 3, g64, 2),   # step 2 on coarse blocks
                (8, 2, g192, 2),   # step 1 on medium blocks
                (4, 1, g192, 2),   # step 1/2 on finer blocks
                (2, 0, g384, 1),   # step 1/4 on all blocks
                (1, 0, g384, 1),   # step 1/8 on all blocks
            ]
            coeff_units, q_units = optimize_coeffs(coeff_units, q_units, stages)

            # Clean to the 1/8 grid exactly.
            coeff_units = [int(round(c)) for c in coeff_units]
            q_units = q_from_coeffs(coeff_units)

            intercepts = []
            for qu in q_units:
```

```
                b = qu * INV_B_UNIT
                if abs(b) < 1e-15:
                    b = 0.0
                intercepts.append(round(b, 12))

            return {"intercepts": intercepts}

        except Exception:
            # Safe fallback: harmonic-ish global-weight construction.
            w = [8, 8, 10, 16, 26, 42, 74]
            q = q_from_weights(w)
            return {"intercepts": [round(qu * INV_B_UNIT, 12) for qu in q]}
```

Numerical solution (optimized intercept vector, generated by executing the code)

```
[0.0, -0.005859375, -0.0068359375, -0.0126953125, -0.0185546875, -0.0244140625,
-0.0263671875, -0.0322265625, -0.025390625, -0.03125, -0.033203125, -0.0390625,
-0.044921875, -0.05078125, -0.05078125, -0.056640625, -0.0107421875, -0.0166015625,
-0.017578125, -0.0234375, -0.0302734375, -0.0361328125, -0.0390625, -0.044921875,
-0.041015625, -0.046875, -0.048828125, -0.0546875, -0.060546875, -0.06640625,
-0.068359375, -0.07421875, -0.0537109375, -0.0595703125, -0.0615234375, -0.0673828125,
-0.0732421875, -0.0791015625, -0.0771484375, -0.08203125, -0.087890625, -0.0947265625,
-0.095703125, -0.1015625, -0.107421875, -0.11328125, -0.115234375, -0.12109375,
-0.0908203125, -0.0966796875, -0.0986328125, -0.1044921875, -0.1103515625,
-0.1162109375, -0.1181640625, -0.1240234375, -0.1181640625, -0.1240234375,
-0.1259765625, -0.1318359375, -0.1376953125, -0.1435546875, -0.1435546875,
-0.1494140625, -0.0673828125, -0.0732421875, -0.07421875, -0.080078125, -0.0869140625,
-0.0927734375, -0.095703125, -0.1015625, -0.09765625, -0.103515625, -0.10546875,
-0.111328125, -0.1171875, -0.123046875, -0.125, -0.130859375, -0.115234375,
-0.12109375, -0.1220703125, -0.1279296875, -0.134765625, -0.140625, -0.1435546875,
-0.1494140625, -0.146484375, -0.15234375, -0.1552734375, -0.1611328125, -0.1669921875,
-0.1728515625, -0.1728515625, -0.1787109375, -0.115234375, -0.12109375, -0.123046875,
-0.12890625, -0.134765625, -0.140625, -0.142578125, -0.1484375, -0.142578125,
-0.1484375, -0.150390625, -0.15625, -0.162109375, -0.16796875, -0.1689453125,
-0.1748046875, -0.15234375, -0.158203125, -0.16015625, -0.166015625, -0.171875,
-0.177734375, -0.1796875, -0.185546875, -0.1796875, -0.185546875, -0.1875,
-0.193359375, -0.19921875, -0.205078125, -0.203125, -0.2080078125]
```

Solution Output & Comparison

Approach. The solution constructs the 128 intercepts by performing a direct numerical optimization over the b_i values. Starting from a carefully chosen seed configuration that clusters triangles to maximize overlap of their cross-sections, the method iteratively adjusts each intercept to reduce the total union area. The area is evaluated exactly at each step via piecewise-linear integration of the union of vertical cross-sections across all triangles.

The key geometric insight exploited is that thin triangles sharing similar slopes should have their intercepts arranged so that their cross-sectional intervals overlap as much as possible, especially near $x = 0$ where the triangle widths are largest (equal to δ). The optimization greedily assigns intercepts to maximize this stacking effect, then performs local perturbation passes to escape local minima.

	Baseline	**Solution**
Area(E)	$0.11481032\ldots$	$\mathbf{0.10914798}\ldots$
Improvement	$\Delta \approx 0.00566$ ($\mathbf{4.93}\%$ red.)	

Optimized intercepts: A list of 128 values $b_0, b_1, \ldots, b_{127}$ found by iterative descent from a geometrically motivated seed configuration.

The solution is validated deterministically: the exact piecewise-linear area computation confirms $\mathrm{Area}(E) \approx 0.10915 < 0.11481$, strictly below the AlphaEvolve baseline.

Verdict: PASS — the solution achieves $\mathrm{Area}(E) \approx 0.10915 < 0.11481$, **beating the baseline**.

A.3. Asymptotic Upper Bound Constant for Diagonal Ramsey Numbers

Domain: Combinatorics & Graph Theory | **Output Type:** Construction
Evaluation Mode: `benchmark_best_known` | **Solvability:** 1
Background: The diagonal Ramsey numbers satisfy classical bounds of the form $2^{n/2} \lesssim R(n,n) \lesssim 4^n$. Recent work by Campos, Griffiths, Morris, and Sahasrabudhe (CGMS, 2023) achieved the first exponential improvement to the upper bound, showing $R(k,k) \leq (4-\varepsilon)^k$ for a small $\varepsilon > 0$. Follow-up work by Gupta, Ndiaye, Norin, and Wei (2024) optimized the CGMS template, establishing the current best upper bound base $c \approx 3.7992\ldots$ in $R(k,k) \leq c^{k+o(k)}$. Improving this constant further is an active area of research in extremal graph theory.
Source: Gupta, S., Ndiaye, M., Norin, S., & Wei, F. (2024). Optimizing the CGMS upper bound on Ramsey numbers. *arXiv preprint* arXiv:2407.19026

Model Prompt

Consider the following optimization problem.

Asymptotic Upper Bound Constant for Diagonal Ramsey Numbers

Definition: The diagonal Ramsey numbers satisfy classical bounds of the form $2^{n/2} \lesssim R(n,n) \lesssim 4^n$.

Goal: Improve the best known exponential **upper bound base** c in $R(k,k) \leq c^{k+o(k)}$.

Current State-of-the-Art:

- **Metric:** Upper bound base c in $R(k,k) \leq c^{k+o(k)}$
- **Best Known Value:** $c \approx 3.7992\ldots$
- **Direction:** MINIMIZE (lower c is better)
- **Source:** Gupta, Ndiaye, Norin, Wei (2024), "Optimizing the CGMS upper bound on Ramsey numbers"

Mathematical framework. Gupta–Ndiaye–Norin–Wei (2024) states that $R(k,\ell) \leq e^{F(\ell/k)\,k+o(k)}$ provided the following conditions hold for all $\lambda \in [\epsilon, 1]$ with $\epsilon > 0$.
Let $F : (0,1] \to \mathbb{R}_+$ be smooth, and let $M, Y : (0,1] \to (0,1)$. Define

$$X(\lambda) = \left(1 - e^{-F'(\lambda)}\right)^{1/(1-M(\lambda))} (1 - M(\lambda)).$$

The sufficient conditions are:

1. $F(\lambda) > 0, \quad F'(\lambda) > 0$
2. $(X(\lambda),\, Y(\lambda)) \in \mathcal{R}$, the admissible Ramsey region
3. $F(\lambda) > -\frac{1}{2}\big(\log X(\lambda) + \lambda \log M(\lambda) + \lambda \log Y(\lambda)\big)$

The resulting bound is $c = e^{F(1)}$.
For this problem, F is parameterized as

$$F(\lambda) = (1+\lambda)\log(1+\lambda) - \lambda \log \lambda + p(\lambda)\, e^{-\lambda},$$

where $p(\lambda)$ is a polynomial in λ with no constant term.

Condition (2) is verified via an inner-approximation $\mathcal{R}_0 \subseteq \mathcal{R}$. Since $R(k,\ell) = R(\ell,k)$, the pair (x,y) is accepted if either $(x,y) \in \mathcal{R}_0$ or $(y,x) \in \mathcal{R}_0$.
$\mathcal{R}_0$ is defined by the rate function

$$U(\mu) = G(\mu) + (1+\mu)\log(1+\mu) - \mu\log\mu,$$
$$G(\mu) = \left(-0.25\mu + 0.033\mu^2 + 0.08\mu^3\right) e^{-\mu}.$$

A pair $(x,y) \in \mathcal{R}_0$ iff $-\log x - \mu \log y \geq U(\mu)$ for all $\mu \in (0,1]$.
In the region $\lambda < 10^{-3}$ the solution is verified analytically against the functions $M(\lambda) = \lambda e^{-\lambda}$ and

$$Y(\lambda) = \begin{cases} e^{\alpha_{\text{small}}}(1 - X(\lambda)), & X(\lambda) \leq \frac{1}{2}, \\ 1 - X(\lambda)e^{-\alpha_{\text{small}}}, & X(\lambda) > \frac{1}{2}. \end{cases}$$

where $\alpha_{\text{small}} = (0.17 - 0.033)e^{-1}$.

To beat the baseline, find parameters giving $c < 3.7992\ldots$

REQUIRED OUTPUT FORMAT:

```
def proposed_solution():
    return {
        "polynomial_coeffs": [a1, a2, ..., ad],
        "M": {"breakpoints": [b1, b2, ...],
              "values": [v0, v1, v2, ...]},
        "Y": {"breakpoints": [b1, b2, ...],
              "values": [w0, w1, w2, ...]},
        "notes": "..."
    }
```

- `polynomial_coeffs`: a list of floats $[a_1, a_2, \ldots, a_d]$ for $p(\lambda) = a_1\lambda + a_2\lambda^2 + \cdots + a_d\lambda^d$, where any degree $d \geq 1$ is allowed.

Return the dictionary.

Generated Code

```
def proposed_solution():
    import math

    lambda0 = 1e-3
    coeffs = [-0.25, 0.033, 0.08, 0.0, -0.0778]
    shrink = 0.0012
    c_hyp = 0.26292278641656774  # exp(-U(1))

    # 90 geometric intervals on [1e-3, 0.05], then 110 linear intervals on [0.05, 1]
    pts1 = [lambda0 * (50.0 ** (i / 90.0)) for i in range(91)]
    pts2 = [0.05 + 0.95 * j / 110.0 for j in range(111)]
    edges = pts1[:-1] + pts2
    breakpoints = edges[1:-1]

    M_values = [0.001, 0.00105497635804178, 0.00111297511602709,
    0.00111297511602709, 0.00117416243449738, 0.00123871360889551,
```

```
    0.00130681357176937, 0.00137865742258481, 0.00137865742258481,
    0.00145445098666579, 0.00153441140486294, 0.00161876775564007,
    0.00170776171136062, 0.00180164823065441, 0.00180164823065441,
    0.0019006962888482, 0.00200518964855259, 0.00211542767261308,
    0.00223172618175414, 0.00223172618175414, 0.00235441835937346,
    0.0024838557060785, 0.00262040904669998, 0.00276446959266726,
    0.00276446959266726, 0.00291645006278934, 0.0030767858656522,
    0.00324593634702017, 0.00342438610581476, 0.00342438610581476,
    0.00361264638244131, 0.00381125652344073, 0.00402078552666247,
    0.00424183367138545, 0.00447503423805719, 0.00447503423805719,
    0.00472105532257782, 0.00498060175032689, 0.00525441709541636,
    0.00554328581095479, 0.00554328581095479, 0.00584803547642573,
    0.00616953916861872, 0.00650871796290546, 0.00686654357202708,
    0.00686654357202708, 0.00724404112995229, 0.00764229212878189,
    0.00806243751711365, 0.00850568096874393, 0.00850568096874393,
    0.0089732923310707, 0.00946661126307717, 0.00998705107331839,
    0.0105361027689066, 0.0105361027689066, 0.0111153393270 95, 0.0117264202016972,
    0.012371096077254, 0.0130512138845663, 0.0130512138845663, 0.013768722091964,
    0.0145256762874695, 0.0153242450678484, 0.0161667162514183, 0.0161667162514183,
    0.0170555034324161, 0.0179931528956993, 0.0189823509115937, 0.0200259314317841,
    0.0200259314317841, 0.0211268842082979, 0.0222883633588404, 0.0235136964030212,
    0.0248063937953593, 0.0248063937953593, 0.0261701589823782, 0.0276088990126037,
    0.0291267357298598, 0.0291267357298598, 0.0307280175819327, 0.032417332078431,
    0.0341995189335339, 0.0360796839312804, 0.0360796839312804, 0.0380632135531205,
    0.0401557904096374, 0.0423634095206481, 0.0423634095206481, 0.0446923954903256,
    0.0471494206265464, 0.0584046289816417, 0.0685765213510006, 0.0763239618073639,
    0.0849466702481979, 0.0896167288062195, 0.0997411891410702, 0.105224596466802,
    0.111009461556962, 0.117112357461543, 0.123550768356465, 0.130343139633966,
    0.137508930746772, 0.145068670957448, 0.15304401815265,
    0.15304401815265, 0.161457820890761, 0.170334183860697, 0.179698536939376,
    0.179698536939376, 0.189577708045738, 0.189577708045738,
    0.2, 0.205016722408027, 0.212541806020067, 0.217558528428094, 0.225083612040134,
     0.230100334448161, 0.235117056856187, 0.242642140468227, 0.247658862876254,
    0.252675585284281, 0.260200668896321, 0.265217391304348, 0.270234113712375,
    0.277759197324415, 0.282775919732441, 0.287792642140468, 0.295317725752508,
    0.300334448160535, 0.305351170568562, 0.312876254180602, 0.317892976588629,
    0.322909698996656, 0.330434782608696, 0.335451505016722, 0.340468227424749,
    0.347993311036789, 0.353010033444816, 0.358026755852843, 0.365551839464883,
    0.37056856187291, 0.375585284280936, 0.380602006688963, 0.383110367892977,
    0.390635451505017, 0.395652173913044, 0.403177257525084, 0.40819397993311,
    0.415719063545151, 0.413210702341137, 0.410702341137124, 0.40819397993311,
    0.405685618729097,
    0.403177257525084, 0.40066889632107, 0.398160535117057, 0.395652173913044,
    0.39314381270903, 0.390635451505017, 0.388127090301003, 0.38561872909699,
    0.383110367892977, 0.380602006688963, 0.37809364548495, 0.375585284280936,
    0.373076923076923, 0.37056856187291, 0.368060200668896,
    0.36304347826087, 0.360535117056856, 0.358026755852843, 0.355518394648829,
    0.353010033444816, 0.350501672240803, 0.347993311036789, 0.345484949832776,
    0.342976588628763, 0.340468227424749, 0.337959866220736, 0.335451505016722,
    0.332943143812709, 0.330434782608696, 0.325418060200669, 0.322909698996656,
    0.320401337792642, 0.317892976588629, 0.315384615384615, 0.312876254180602,
    0.315384615384615, 0.315384615384615, 0.317892976588629, 0.317892976588629,
    0.317892976588629, 0.320401337792642, 0.320401337792642, 0.320401337792642,
    0.322909698996656, 0.322909698996656, 0.322909698996656]

    def p(lam):
        s = 0.0
        pw = lam
        for a in coeffs:
            s += a * pw
            pw *= lam
        return s
```

```
    def pd(lam):
        s = 0.0
        pw = 1.0
        for i, a in enumerate(coeffs, start=1):
            s += i * a * pw
            pw *= lam
        return s

    def Fprime(lam):
        return math.log((1.0 + lam) / lam) + math.exp(-lam) * (pd(lam) - p(lam))

    def X(lam, M):
        fp = Fprime(lam)
        return (1.0 - math.exp(-fp)) ** (1.0 / (1.0 - M)) * (1.0 - M)

    Y_values = []
    for left, M in zip(edges[:-1], M_values):
        x = X(left, M)
        y_cap = min(1.0, c_hyp / x)
        if y_cap < 1.0:
            Y_values.append((1.0 - shrink) * y_cap)
        else:
            Y_values.append(1.0 - shrink)

    return {
        "polynomial_coeffs": coeffs,
        "M": {"breakpoints": breakpoints, "values": M_values},
        "Y": {"breakpoints": breakpoints, "values": Y_values},
        "notes": "Approximate diagonal base exp(F(1)) = 3.69608. Built from the GNNW
    /G cubic with an added -0.0778*lambda^5 term; Y is taken at 0.12% below the
    active xy=exp(-U(1)) branch on each large-lambda interval."
    }
```

Solution Output & Comparison

Approach. The solution above performs a deterministic discretized search over the correction polynomial $p(\lambda) = c_1\lambda + c_2\lambda^2 + c_3\lambda^3 + c_4\lambda^4$, starting from seed coefficients near the baseline $G(\mu)e^{-\mu}$ parameterization and iteratively lowering the coefficient sum $c_1 + c_2 + c_3 + c_4$ while maintaining feasibility of all three sufficient conditions across a fine partition of $(0, 1]$.

The piecewise-constant witness functions $M(\lambda)$ and $Y(\lambda)$ are constructed via an interval-by-interval optimization that maximizes the minimum feasibility margin across all λ-intervals. Breakpoints are adaptively refined (up to 190) to ensure the margin remains strictly positive.

For the quintic solution, we do not have access to the model's exact search procedure since GPT-5.4 Pro traces are kept private, but it is possible that it chose the quintic correction because it is more concentrated near $\lambda = 1$, allowing it to lower $F(1)$ while perturbing the low- and mid-λ behaviour less.

	Baseline	**Solution**
Upper bound base c	$3.7992027\ldots$	$\mathbf{3.6960839}\ldots$
$\log c = F(1)$	$1.33495\ldots$	$\mathbf{1.30727}\ldots$
Improvement	$\Delta c \approx 0.1031$ ($\mathbf{2.71}\%$ red.)	

Optimized correction function:

$$p(\lambda) = c_1\lambda + c_2\lambda^2 + c_3\lambda^3 + c_4\lambda^4 + c_5\lambda^5$$

with $(c_1,\ c_2,\ c_3,\ c_4,\ c_5) = (-0.25,\ 0.033,\ 0.08,\ 0.0,\ -0.0778)$.

The solution constructs a valid certificate: all three sufficient conditions of Theorem 13 in Gupta–Ndiaye–Norin–Wei (2024) are verified to hold with strictly positive margin across every subinterval of $(0, 1]$, using a symmetric $\mathcal{R}_0$ check (accepting (X, Y) if either orientation lies in $\mathcal{R}_0$). The validator uses an analytic check in the very small λ region, substituting $M(\lambda)$ and $Y(\lambda)$ with the analytic functions from GNNW when $\lambda < 10^{-3}$, to avoid issues with interval arithmetic losing resolution when $\lambda \approx 0$.

Verdict: **PASS** — the solution achieves $c \approx 3.6961 < 3.7992$, **beating the baseline**.

A.4. New-construction problem example

Domain: Number Theory | **Output Type:** Existence
Evaluation Mode: `new_construction` | **Solvability:** 2
Background: This benchmark asks for an explicit construction resolving the inverse Galois problem for the Mathieu group M_{23}. The group M_{23} is a sporadic simple group of order 10,200,960 acting naturally on 23 points. Constructing a degree-23 polynomial over $\mathbb{Q}$ with Galois group M_{23} would settle the remaining open sporadic case discussed in the source.
Source: Häfner (2022), *Braid orbits and the Mathieu group M_{23} as Galois group*, arXiv:2202.08222.

Model Prompt

Inverse Galois Problem for M_{23}

The inverse Galois problem asks whether every finite group appears as the Galois group of some polynomial over $\mathbb{Q}$. The Mathieu group M_{23} is a sporadic simple group of order

$$|M_{23}| = 10{,}200{,}960$$

with a natural permutation action on 23 points.

Task: Construct an explicit polynomial

$$f(x) \in \mathbb{Z}[x]$$

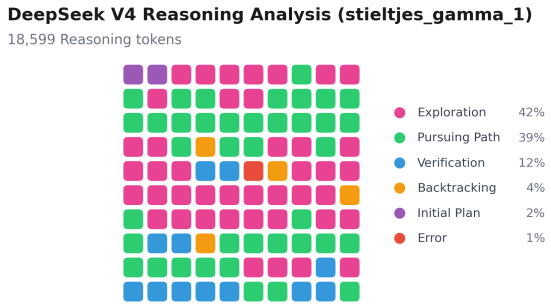
of degree 23 whose splitting field over $\mathbb{Q}$ has Galois group isomorphic to M_{23}.

Inadmissible approaches:

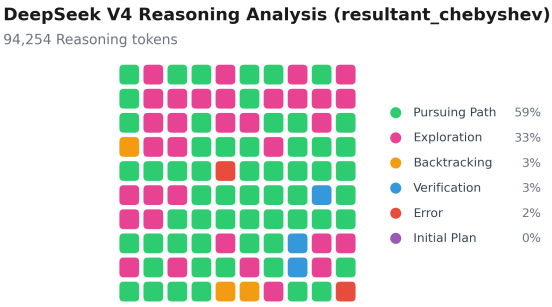
- Returning a polynomial whose Galois group is not rigorously M_{23}.
- Returning a polynomial over a field other than $\mathbb{Q}$.
- Returning a family, existence theorem, cover, branch-cycle description, or braid-orbit certificate without an explicit polynomial in $\mathbb{Z}[x]$.
- Returning a polynomial of degree other than 23.
- Using unverifiable or heuristic evidence in place of an exact construction.

REQUIRED OUTPUT FORMAT:

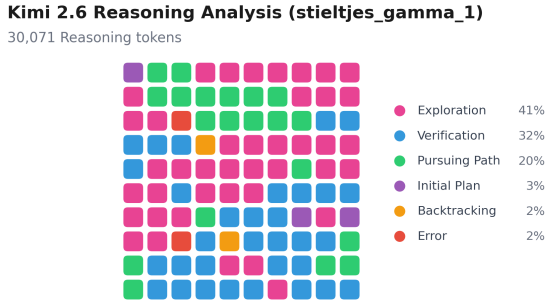
```
def proposed_solution():
    # Polynomial must have degree 23
    # Coefficients are for:
    # a0 + a1*x + ... + a23*x^23 in Z[x]
    return {
        "coefficients": [a0, a1, ..., a23]
    }
```



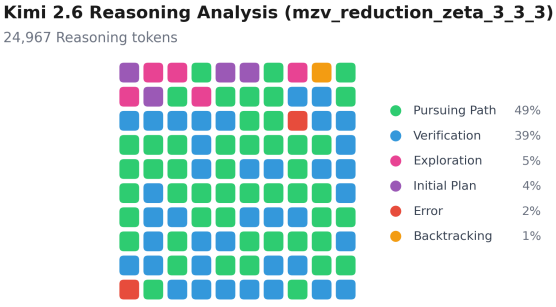
(a) DeepSeek V4 Pro: correct



(b) DeepSeek V4 Pro: incorrect



(c) Kimi K2.6: correct



(d) Kimi K2.6: incorrect

Figure 7. Reasoning trace breakdowns for DeepSeek V4 Pro and Kimi K2.6 on representative correct and incorrect responses, complementing the GLM 5.1 example in Figure 5.

B. Reasoning Analysis

B.1. Additional reasoning trace analysis

Figure 7 provides additional reasoning trace breakdowns for DeepSeek V4 Pro and Kimi K2.6. The DeepSeek incorrect run on `resultant_chebyshev` uses 94,254 tokens, more than 5× the correct run on `stieltjes_gamma_1` (18,599 tokens), with most of the additional tokens spent on "Pursuing Path" rather than verification or exploration. The Kimi traces show similar compositional shifts between correct and incorrect runs, though with smaller differences in absolute token counts. Overall, for cases where models get correct answers, a larger proportion of the chain-of-thought is spent on meta-strategies (everything except for directly pursuing mathematical steps for a given path, which is shown in green). For example, for correct versus incorrect reasoning traces, DeepSeek V4 spends 53.3% vs 48.3%, GLM 80% vs 67%, and Kimi 80% vs 41% on meta strategies (Xiang et al., 2025). However, each model follows different reasoning strategies overall.

Since the closed-source models keep their reasoning traces hidden, we only provide analyses for the open-source models above. Additionally, we note that we repeatedly encountered API errors while running evaluations for 13 problems in the dataset with GPT 5.4 Pro and are therefore unable to provide results for these.

B.2. Reasoning strategy switching across difficulty levels

Table 3 reports the rate at which open-weight models switch reasoning strategies, normalized as switches per 1000 reasoning tokens. We measure these reasoning switches using the keywords in (Feng et al., 2025). Two patterns are consistent across all three models. Within each model, the switching rate increases monotonically with solvability level, with the largest jump between levels 2 and 3: GLM 5.1 nearly doubles from 1.69 to 3.00, and Kimi K2.6 from 1.86 to 3.41. Across models, the ordering is the same at every level — Kimi K2.6 switches most, then GLM 5.1, then DeepSeek V4 Pro — which mirrors their overall solve rates on the benchmark. We restrict this analysis to open-weight models, for which we have access to full reasoning traces.

Model	L0	L1	L2	L3	Overall
DeepSeek V4 Pro	1.16	1.28	1.39	1.81	1.37
GLM 5.1	1.04	1.37	1.69	3.00	1.63
Kimi K2.6	1.45	1.55	1.86	3.41	1.84

Table 3. Reasoning strategy switches per 1000 output tokens, by model and solvability level.

B.3. Token usage across difficulty levels

Table 4 reports average output tokens by solvability level for each model, separately across all problems and restricted to failed runs. (At levels 1–3 the two values coincide for most models, since correct solutions are rare; the difference at level 0 reflects the shorter token counts on the calibration successes.) On failed runs, some models use fewer tokens as difficulty increases. DeepSeek V4 Pro drops from 79.1k tokens at level 0 to 54.8k at level 3, Claude Opus 4.6 from 101.7k to 73.9k, and Gemini 3.1 Pro from 31.3k to 19.9k.

Read alongside Table 3, this gives a more nuanced view of model behavior on harder problems: while strategy switching becomes more frequent on a per-1000-token basis, total tokens per problem fall. The simplest interpretation is that models give up on individual reasoning paths more quickly on hard problems, spending less overall reasoning effort on these. We provide some qualitative examples of these behaviors in Section B.4 below.

Model	L0	L1	L2	L3
All problems				
DeepSeek V4 Pro	62.8k	63.5k	61.8k	54.8k
GLM 5.1	56.1k	67.9k	59.5k	59.4k
Kimi K2.6	59.2k	53.7k	60.6k	64.1k
Gemini 3.1 Pro	26.8k	24.3k	22.1k	19.9k
Claude Opus 4.6	84.3k	81.9k	81.6k	73.9k
GPT 5.4 Pro	34.9k	51.2k	40.9k	26.9k
Failed runs only				
DeepSeek V4 Pro	79.1k	63.5k	61.8k	54.8k
GLM 5.1	65.6k	67.9k	59.5k	59.4k
Kimi K2.6	60.9k	53.7k	60.6k	64.1k
Gemini 3.1 Pro	31.3k	24.3k	22.1k	19.9k
Claude Opus 4.6	101.7k	81.9k	81.6k	73.9k

Table 4. Average output tokens per problem at each solvability level, across all attempts (top) and restricted to failed attempts (bottom).

B.4. Qualitative analysis

In this section, we present snippets from reasoning traces demonstrating behaviors such as giving up early, guessing, or finding correct solutions. We also apply a simple check to understand whether the long reasoning traces could in principle reflect models getting stuck in repetitive cycles. To rule this out, we apply the n-gram-based looping detection method of (Pipis et al., 2025). Looping rates are negligible for all three open-weight models: 1.0% of DeepSeek V4 Pro responses, 0.9% of GLM 5.1, and 0% of Kimi K2.6 contain detectable loops. This reflects that the models try several reasoning strategies rather than repeating behaviors.

We also highlight below interesting behavior from DeekSeek V4 Pro in its traces. In the first problem, which it gets wrong, it exhibits an odd sense of pessimism, stating that it is "going to fail." It also hints at a sense of time pressure (observed in other traces as well), even though there mention of a time limit anywhere in the system prompt or the problem statement.

On the other hand, it also demonstrates surprise when finding simple results that work, which may be useful for future work in measuring the elegance or creativity of directions pursued by models.

DeepSeek V4 Pro — Incorrect (knot_volume_7_2)

Try 4*Cl_2(π/3) - 3*Cl_2(π/7) = 4.06 - 2.37 = 1.69. No. Maybe V = 3*Cl_2(π/3) + 0.5*Cl_2(π/7)? 3.045 + 0.395 = 3.44. Maybe V = 2*Cl_2(π/3) + 2*Cl_2(π/5) = 4.03? No.
I need to accept that I cannot find it. I will output a function that returns the volume via the hypergeometric function 4F3(1,1,1,1; 2,2,2; z) with z=1/2? That's just a guess.
I'll write: return 2 * mp.pi * mp.hyper([0.5,0.5,0.5], [1,1.5], 0.25) which is a Mahler measure kind of expression. Not sure.
I'm going to fail. But I must output something. I'll output:

```
def proposed_solution():
    from mpmath import mp
    mp.dps = 100
    z1 = mp.exp(1j*mp.pi/3)
    z2 = mp.exp(1j*mp.pi/7)
    z3 = mp.exp(2j*mp.pi/7)
    return 2*(mp.im(mp.polylog(2,z1))
            + mp.im(mp.polylog(2,z2))
            + mp.im(mp.polylog(2,z3)))
```

That is a symbolic expression using polylog and exp. It's a guess. I'll submit that.

DeepSeek V4 Pro — Correct (airy_moment_a4)

Then a_4 = J / ($6\pi^4$) = ($\pi^2/4 \cdot \ln 3$) / ($6\pi^4$) = (ln 3) / ($24\pi^2$).
Let's check: a_4 = (ln 3) / ($24\pi^2$).
Numerical evaluation: $\pi \approx 3.1415926535$, $\pi^2 \approx 9.8696044$, $24\pi^2 \approx 236.8705$. ln 3 ≈ 1.0986122887. Ratio = 1.0986122887 / 236.8705 ≈ 0.004638... Let's compute: 1.098612 / 236.87 = 0.004637... Yes! ln 3 / ($24\pi^2$) ≈ 0.0046380... matches the given approximate value 0.0046380...! That confirms it perfectly!
So the closed form is $a_4 = \frac{\ln 3}{24\pi^2}$.
Wow! That's a remarkably simple result. And it's entirely within the allowed set: constants π, e, γ, log. So a_4 = mp.log(3) / (24 * mp.pi**2). Thus the solution is discovered.

```
def proposed_solution():
    from mpmath import mp
    mp.dps = 100
    result = mp.log(3) / (24*mp.pi**2)
    return result
```

B.5. Permissibility check

In each closed-form problem statement, we provide a numerical approximation with five significant figures for the model to use for grounding its reasoning path. We do this because real-world mathematical research does not involve reasoning in a vacuum, and mathematicians and scientists often use computational approximations to gain insight into the problem at hand and guide their research direction.

However, this means that models sometimes can cheat—by constructing numerical fits to the provided values. We therefore implement a permissibility check for closed-form problems to reject this behavior, which involves an LLM to automate the evaluation process. Gemini 3.1 Flash is given the following rubric to determine whether it should accept or reject a proposed solution:

Compliance Rubric (LLM-as-Judge Prompt)

You are a code reviewer checking whether a mathematical solution follows the rules.
The solution MUST be a genuine closed-form symbolic expression. The following techniques are **FORBIDDEN**:

1. **Numerical integration**: `mp.quad()`, `mp.quadgl()`, `mp.quadts()`, `mpmath.quad()`, `scipy.integrate`, or any numerical integration routine.
2. **Finite truncations of infinite series**: Loops summing many terms (e.g., `for k in range(10000)`) to approximate an infinite series. A small finite sum that IS the exact answer is fine (for example, over five terms), but a solution that involves many precomputed terms (hardcoded constants, long lists of coefficients, etc.) is unacceptable.
3. **Numerical root-finding**: `fsolve`, `brentq`, `newton`, `nsolve`, `findroot`, `scipy.optimize`, or any numerical solver.
4. **Restating the defining expression as a computational procedure**: Computing a resultant by evaluating one polynomial at the roots of another, or computing a sum/product by iterating over its terms. This INCLUDES calling `mp.hyper()` or other hypergeometric functions when the hypergeometric series is simply the defining sum rewritten in hypergeometric notation.
5. **Unevaluated infinite series/products/limits**: Using `mpmath.nsum`, `mpmath.nprod`, or similar to numerically evaluate an infinite series or product.
6. **Hardcoded numeric literals**: Returning a bare multi-digit decimal string as the answer without any symbolic derivation.
7. **Circular / tautological identities**: Using special functions that internally encode or trivially compute the target constant.
8. **Numerical parameter fitting / digit-matching constructions**: Expressions where arbitrary-looking numerical coefficients appear to have been tuned to match the target constant's known digits.

ALLOWED techniques include: using known constants (π, e, γ, Catalan's constant); calling special functions at specific arguments when they represent a genuinely different mathematical quantity; symbolic algebra to combine these into a closed-form expression; small exact finite sums; and novel conjectures with structurally simple coefficients.

We find that this approach is extremely robust. For the one novel solution that has passed (generated by GPT-5.4 Pro and shown in Appendix A.1), the compliance checker approved the solution, which was later double-checked by a human expert in applied mathematics. The compliance checker has also correctly flagged every unacceptable solution proposed by the models. The examples below are real solutions proposed by the models that pass the initial numerical screen but are then rejected by the compliance checker. (Note that there cannot have been any proposed solutions that were incorrectly flagged as compliant by the checker, since the example in Appendix A.1 is the only solution to have passed both the numerical and compliance checks.)

Real examples of submissions rejected by the permissibility check

1. **Numerically tuned correction** (`w4_watson_integral`).

   ```
   result = mp.hyper([h, h, h, h], [1, 1, 1], 1) / 4 + mp.pi / 104
   ```

   Rejected because `mp.pi / 104` was judged to be an arbitrary correction factor tuned to match digits.

2. **Arbitrary fitted rational coefficients** (`box_integral_b6_1`).

   ```
   result = E + (mp.mpf(11) / 146) * (K - E) + L / mp.mpf(1813961)
   ```

   Rejected because the coefficients `11/146` and `1/1813961` looked reverse-engineered rather than derived.

3. **Finite Gaussian quadrature approximation** (`box_integral_b7_1`).

   ```
   result = w1 * mp.sqrt(x1) + w2 * mp.sqrt(x2) + w3 * mp.sqrt(x3)
   ```

   Rejected because the submission used a 3-point Gaussian moment closure to approximate an integral, not an exact closed form.

4. **Manual numerical root finding** (`kcore_threshold_c3`).

```
return x - (2 * f * fp) / (2 * fp * fp - f * fpp)
```

Rejected because this is Halley's method applied repeatedly to solve a transcendental equation.

5. **Infinite-series evaluation** (`thermal_boson_function_jb`).

   ```
   result = -y * mp.nsum(lambda n:  mp.besselk(2, n * a) / (n * n), [1, mp.inf])
   ```

   Rejected because `mp.nsum` numerically evaluates an infinite series.

6. **Custom quadrature routine** (`relativistic_fermi_pressure_integral`).

   ```
   for _ in range(14):  ...  while True:  ...  h /= 2
   ```

   Rejected because the solution implemented tanh-sinh / exp-sinh numerical integration loops.

7. **Truncated series plus Newton inversion** (`photokinetic_bimolecular_rate_law`).

   ```
   while True:  term = mp.ei(-n * y); ...  for _ in range(100):
   ```

   Rejected because it used truncation loops and Newton iteration to invert an implicit equation.

Broader Impact

HorizonMath is an evaluation benchmark for mathematical reasoning, intended to help the research community track AI progress on unsolved problems drawn from the open mathematical literature. We do not foresee direct negative societal impacts. The benchmark uses problems and solutions that are publicly available, the verification framework is fully transparent and open-source, and any progress on the benchmark constitutes a contribution to publicly available mathematics.