AGSM: Continuous Synthesis and Evaluation of Algebraic Word Problems for LLMs

Travis Godwin Good, PhD

Ambient Research (2025-09-15 UTC)

Abstract

We introduce AGSM (Ambient Grade School Math), a dynamic benchmark for evaluating algebraic reasoning in large language models. AGSM programmatically generates solvable equations with a symbolic math engine (SymPy), renders them into natural language word problems via an LLM, and verifies correctness through round-trip parsing. Verified problems are deterministically composed into multi-step tasks, graduating in difficulty based on the number of steps included. In preliminary experiments, state-of-the-art models exhibit predictable degradation with increasing difficulty, yet with surprising irregularities across model families. Performance declines persist regardless of grading method (ground truth or consensus) or generation strategy, underscoring both the fragility and variability of current systems on multi-step reasoning. AGSM contributes a reproducible pipeline for continuously generating fresh, ground-truthed benchmarks that resist contamination and approximate the quality of human-written problems. Beyond evaluation, the framework supports curriculum-style training and scalable generation of reasoning traces, offering a path toward evergreen assessment and improvement of language models.

Introduction

This work is part of a broader effort by Ambient (Ambient.xyz) to design decentralized AI infrastructure. The AGSM benchmark not only provides an open and renewable method of evaluating reasoning, but also serves as a prototype of the kind of continuous, verifiable task stream that Ambient envisions for its network. By integrating automated problem synthesis, validation, and difficulty scaling, the benchmark sets the stage for model assessment and model training embedded directly into network operations. The implication is that AI development need not be bottlenecked by static human-curated datasets; instead, models and data can co-evolve, with careful verification mechanisms ensuring quality and safety. In future parts of this series, we extend this work toward curriculum-style fine-tuning and distributed on-chain verification of model inference. Code is available here.

Logical and mathematical capability, as measured by static tests like GSM8K (Grade School Math 8K) and MATH[1] is crucial for modern problem-solving LLMs, but the tests themselves have become "saturated"[2] due to a difficult-to-disentangle combination of increased model performance and dataset contamination. For example, a follow-up study to the original GSM8K introduced a new "GSM1K" test set modeled after 8K and observed significantly lower performance by current models, suggesting that high GSM8K scores may be due to leakage. With doubt, there is an urgent need for fresh, out-of-distribution evaluation data.[3]

Unfortunately, creating large quantities of novel, high-quality math word problems by hand is expensive and time-consuming. Prior works such as MathCAMPS and DyVal (see: Related Work) suggest that with careful design, synthetically generated data can approach the quality and complexity of human-written problems, but MathCAMPS lacked an approach to difficulty scaling, and DyVal's generated problems, while mechanically complex, feature writeups that are fully literal natural language descriptions of equations and algebraic manipulations rather than traditional word problems.

Improving on these approaches, we build a continuous dataset generation and evaluation system for algebraic word problems with varied topics, fluent writeups, specific ground truth answers, and linearly scaling difficulty.

Our contributions include:

- **Automated Problem Synthesis with Verification:** We develop an open-source pipeline that programmatically generates algebraic equations (covering families including linear 1-var and 2-var systems, quadratics including non-square discriminants, rational equations with domain guards, logarithmic and exponential forms) and then realizes each equation into a natural language word problem using a large language model. A round-trip verification step prompts an LLM or parser to convert the generated text back into SymPy equations and checks the solution, ensuring the word problem faithfully represents the intended math problem. This guarantees that each problem has a known correct answer and filters out most flawed and ambiguous questions.

- **Difficulty-Shaped Multi-Step Tasks:** We introduce a deterministic composite problem builder that constructs higher-order problems by combining simpler sub-problems. Specifically, we produce multi-step word problems whose final answer is the sum of answers to intermediate sub-problems multiplied by random coefficients. This allows us to grade problem difficulty on a 1 to 10 scale, where a level-n problem entails solving n component problems in sequence. The approach ensures an interpretable progression of task complexity and makes it straightforward to verify multi-step solutions (each sub-part can be independently solved and checked).

- **Multi-Model Sieve for Tractability:** To avoid generating tasks that are either too trivial or essentially unsolvable, we employ a two-stage multi-model evaluation sieve. In the first stage, all individual problems that can be composed later are verified as explicitly (not just round-trip) solvable by at least one LLM. In the second stage, candidate composite problems are tested on a panel of weaker open source LLMs. If >50% of weak models solve a problem easily, it is discarded and tagged as low difficulty.

- **Narrative Perturbations- Problem Topics and Random Topical Facts:** We use an LLM to make our sub-problems coherently topical, drawing on a list of 70k LLM-generated hierarchical topics. To increase difficulty, we include a random topical fact at the beginning or end of a given sub-problem statement. Insertions are explicitly *semantics-independent* of the math; constants and variables do not change.

- **Benchmarking Harness and Open Dataset:** We provide a standardized evaluation harness that queries models for answers in a structured JSON format and checks their responses against the ground-truth numeric answers. We report comprehensive results for a variety of models (spanning different providers and model sizes) on our benchmark, analyzing performance as a function of difficulty. All problems, solutions, and evaluation code are released openly to serve as a community resource and to allow external validation. This not only addresses concerns of hidden test data but also enables researchers to use the data generator for training new models, spurring further improvements.

Our aim is to integrate this system into Ambient's broader infrastructure for decentralized AI. By continuously generating and validating fresh data, the network can both evaluate its models in an evergreen fashion (mitigating the risk of benchmarks becoming stale or gamed) and supply a stream of new training examples to improve those models. In the following, we first discuss related efforts in synthetic data and benchmarking (Section 2). We then detail our methodology (Section 3), including problem generation, verification, and difficulty composition. Section 4 presents preliminary results from benchmarking multiple state-of-the-art models, with performance breakdowns and analysis in Section 5. We conclude with implications for future research and how this approach supports an evolving ecosystem of open model development.

Related Work

Benchmarks for Mathematical Reasoning:

Standard math word problem datasets have played a pivotal role in measuring LLM reasoning. The GSM8K benchmark introduced by Cobbe et al. (2021) consists of 8.5K grade-school math problems focusing on arithmetic reasoning.[4] The MATH dataset extends to competition-level mathematics problems from high school contests. Both GSM8K and MATH are human-authored and were meant to be challenging hurdles for models. However, as noted above, neither has been updated since release, and their widespread use has likely led to inclusion in model training corpora. A recent study by Balloccu et al. (2024) surveyed leakage and evaluation malpractices, finding that closed-source LLMs sometimes "leak, cheat, and repeat" on test data. Indeed, evidence of test set contamination has emerged: Zhang et al. (2024) report that models perform significantly worse on a new set of grade-school math problems that mimics GSM8K, indicating that memorization inflated the original benchmark scores. These issues highlight the need for dynamically generated evaluations to fairly assess true reasoning ability.

LLM-Aided Synthetic Data Generation:

Using AI to generate new evaluation or training data is popular. Several works have demonstrated that large models can assist in creating datasets that target specific skills. The BigToM benchmark is one example outside of mathematics, where an LLM was guided to produce a suite of social reasoning tasks by first generating a symbolic outline of a problem which is then realized into natural text. For mathematical reasoning, MathCAMPS (Mishra et al., 2024) is closely related to our work. MathCAMPS uses formal grammars tied to Common Core math standards to sample problems, solves them with SymPy, and then employs an LLM to convert the symbolic problems into fluent word problems. Crucially, they enforce cycle-consistency: the generated word problem is fed back into a solver (via an LLM or a parser) to ensure it returns the same answer, filtering out unfaithful generations. Our approach adopts a similar verification principle, but we extend it with an explicit multi-step composition mechanism and multi-model filtering. Another relevant effort is DyVal (Zhu et al., 2024), which dynamically generates reasoning tasks from computational graphs. DyVal introduced two math-focused tasks (basic arithmetic and linear equations) and demonstrated that evaluating models on these generated problems can reveal weaknesses not apparent from static benchmarks. Compared to DyVal, which emphasizes dynamic evaluation protocols, our work places more emphasis on *curriculum-style difficulty scaling* and on concrete dataset release for both training and evaluation.

Synthetic data has also been leveraged to improve model training. For example, OpenAI's GPT-4 has been used to produce new solution trajectories for math problems, which are then used to fine-tune smaller models. NVIDIA's OpenMathInstruct project followed this paradigm by generating 1.8 million problem-solution pairs (using a strong closed model to produce code-based solutions for GSM8K and MATH problems) and using them to train open models. The resulting 70B model achieved 84.6% on GSM8K, nearly matching GPT-4 on that benchmark and illustrating the power of targeted synthetic training data. Nvidia's AIMO-2 curated a large dataset of math problems and then trained a model to identify the most promising solutions and proceed using tool-assisted reasoning, resulting in state of the art performance on national-level math problems.[5] Our system contributes to this line of work by offering a means to generate unlimited verified math problems that can serve as fresh training data. Unlike OpenMathInstruct which bootstraps from existing benchmark questions, our pipeline creates entirely new problems from random equations, reducing the chance of overlapping with any data seen during pretraining.

Verification and Evaluation Frameworks:

Ensuring that generated data is correct and measuring model performance reliably are non-trivial challenges. Prior methods for verified problem generation include leveraging theorem provers or constraint solvers. For instance, Fedoseev et al. (2024) used satisfiability modulo theory (SMT via Z3) to generate logical constraints and then trained models to translate natural language problems into Z3 queries. Their approach filtered candidate problems by checking solutions with the solver and discarding incorrect ones, analogous to our SymPy-based solution verification. In terms of evaluation, structured output formats and hidden ground truth have been proposed to reduce ambiguity. We adopt a simple but effective strategy: models must output their final answer in a JSON object (e.g. `{"answer": 42}`), which our harness can easily parse. This eliminates the need for subjective human judgment because an answer is either exactly correct or not. Our scoring is absolute accuracy on the final answer, with a configurable (small) percentage hedge to allow for minor floating point differences. This is a stricter criterion than giving points for showing work, but it aligns with how benchmarks like GSM8K are evaluated (only the final numeric answer is considered). By evaluating multiple model APIs under the same conditions (same prompt format, JSON answer requirement, and same test questions), our harness provides an apples-to-apples comparison of LLM math capabilities.

In summary, our work builds on and extends the above threads: we combine symbolic generation and LLM realization (like MathCAMPS), enforce verification (like cycle-consistency and solver checks), introduce a difficulty scaling via composite tasks, and provide a public benchmark with a rigorous evaluation setup. This contributes a new dynamic benchmark for mathematical reasoning, addressing the limitations of static datasets and providing a tool for both measuring and improving model performance in this domain.

Methodology

Our system consists of a pipeline of components designed to generate high-quality algebraic word problems and ensure their correctness and calibrated difficulty. Figure 1 provides an overview of the generation and evaluation workflow. In this section, we describe each stage of the pipeline in detail.

3.1 Problem Generation via Symbolic Math

The first stage is to generate algebraic problem instances in a formal representation. We use the SymPy library for symbolic mathematics to create random equations or systems of equations that have well-defined solutions. By using SymPy, we can sample from a broad space of algebraic problems (arithmetic operations, linear equations, polynomial equations, etc.) while guaranteeing solvability.

For example, here is the code that produces a linear two variable system:

```
def generate_linear_2var_system() -> ProblemArtifacts:
```

```
x, y = symbols('x y', real=True)
# Choose integer solution (x0, y0)
x0, y0 = choose_distinct(-8, 8, 2)
# Choose invertible 2x2 integer matrix A and compute b = A*[x0, y0]
a11, a12, a21, a22 = choose_distinct(-6, 6, 4)
det = a11*a22 - a12*a21
while det == 0:
    a11, a12, a21, a22 = choose_distinct(-6, 6, 4)
    det = a11*a22 - a12*a21
b1 = a11*x0 + a12*y0
b2 = a21*x0 + a22*y0
eq1 = Eq(a11*x + a12*y, b1)
eq2 = Eq(a21*x + a22*y, b2)
sol = sp.solve([eq1, eq2], (x, y), dict=True)
assert len(sol) == 1
solution = { 'x': sp.nsimplify(sol[0][x]), 'y': sp.nsimplify(sol[0][y]) }
sym_src = f"from sympy import symbols, Eq, solve\nx,y = symbols('x y',
real=True)\neq1 = Eq({a11}*x + {a12}*y, {b1})\neq2 = Eq({a21}*x + {a22}*y, {b2})\nsol =
solve([eq1, eq2], (x,y))"
return ProblemArtifacts(
    family="linear_2var_system",
    eqs=[eq1, eq2],
    variables={"x": "first unknown", "y": "second unknown"},
    domain="R",
    solution=solution,
    sympy_src=sym_src,
    difficulty_meta={"det": det}
)
```

The range of values of coefficients in the equation is constrained so that the mathematical calculation for the word problem itself doesn't become too complex. We retain the generation data in a structured format that is easy to rehydrate.

3.2 Language Realization with an LLM

Next, each symbolic problem is translated into a word problem format using a large language model. We prompt an LLM (DeepSeek R1 5/28) with the equation. For our full prompt, please see Appendix A. An excerpt of the prompt follows:

> "You are a precise math problem writer.
> Write a SINGLE realistic word problem that implies relationships naturally — never state equations or templates.
> Hard constraints:
>
> - Define each variable in plain language (e.g., "Let x be the delivery delay in hours").
> - Use every given number exactly as-is; do not invent new numbers.
> - Do NOT include equations, math symbols, or operator words: no '=', '+', '-', 'x', '/', 'Eq(', 'equation', 'formula', 'expression'. Also avoid: 'equals', 'squared', 'times', 'solve', 'satisfies'.
> - Embed numbers into natural roles (rates, fees/waivers, totals, counts, times, distances, measurements). Express negatives as credits/shortfalls/refunds.
> - Prefer narrative patterns per relation type:

– Linear (a*x + b = c): per-unit rate (a per ...), plus/minus fixed fee/waiver (b), total/credit (c).
– Quadratic (xˆ2 + bx = c): rectangle sides x and x+b with area c; avoid saying 'squared'.
– Rational (A/(x±k) + B/(x±m) = C): combined/parallel rates using 'per' or 'over' (e.g., '6 per (x+8) hours').
– Logarithmic (log_base_k(...)=r): explicitly say 'logarithmic (base k)' or 'log base k', and make the 'logarithmic reading' of the processed measure equal to r; do NOT state that the processed measure itself equals r.
– Exponential: a growth/decay 'level reads m at time x' without formulas.
– 2–4 sentences, realistic units, and exactly one explicit question about the target variable(s).
– Plain ASCII only: no LaTeX/Markdown/code fences."

Here is an example of the format produced:

> "In a wetland conservation effort, invasive plant removal requires 8 hours per plant for extraction, plus a fixed 5 hours for equipment setup. The total recorded work time is 29 hours. Let x be the count of invasive plants removed. What is x? Herbicide application is an effective control method for invasive purple loosestrife in wetlands."

The generated story includes a setup related to a given topic, some constraints, and a name for the solution variable. It also features a random topical fact inserted at the end. The equivalent Python code for this problem is:

```
from sympy import symbols, Eq, solveset, S
x = symbols('x', real=True)
eq = Eq(8*x + 5, 29)
sol = solveset(eq, x, domain=S.Reals)
```

One important aspect is controlling the difficulty via language. Even if a generated equation is simple, the wording can make a problem easier or harder. For now, our system does not deeply manipulate linguistic complexity; we rely on the LLM to produce straightforward grade-school language. In future iterations, we intend to refine the generation to minimize ambiguity better by default and to support configurable complication.

3.3 Verification by Round-Trip Solving

Once a word problem is generated, we must ensure it actually corresponds to the original equation and is solvable to yield the correct answer. We perform a round-trip verification: take the generated text and attempt to parse it back into a formal problem that can be solved, then compare the solution with the original. There are a few ways to do this. In our implementation, we used the same LLM with a different prompt to extract the equations from the word problem. For the full conversion prompt, see Appendix A. An excerpt follows:

> *System (base)*
> You convert algebra word problems into explicit variables and equations in SymPy form.
> Output ONLY compact JSON matching this schema:
> `{"variables": {"x": "semantic description", "...": "..."}, "equations": ["Eq(...)", "Eq(...)", "..."]}`
> Constraints and conventions:

- Interpret negation from natural phrasing precisely:
  - credit/refund/shortfall/deficit → negative totals (e.g., "a $52 credit" → -52).
  - charge/due/invoice total → positive totals.
  - "drops by k" or "decreases by k" → -k; "rises/increases by k" → +k.
- Return ONLY JSON; no code fences, commentary, or extra text.
- Make sure equations are valid SymPy and solve to a single solution over the reals.

—

Return ONLY JSON; no code fences, commentary, or extra text.
Make sure equations are valid SymPy and solve to a single solution over the reals.
—
System (strict additions used at runtime)
IMPORTANT:
—
Use these variable names EXACTLY (verbatim), including underscores: x, y, ... (from the item's variables).
Do not drop underscores or rename variables (e.g., do not write x1 for x_1).
—
Output must include only these variables; do not invent new variable names.
—
Example JSON (structure only, keep underscores): `{"variables": {"x_1": "semantic for x_1", ...}, "equations": ["Eq(1x_1+1x_2, 0)"]}`
—
User
Extract variables and equations from the following word problem.
Problem:
{*problem_text*}
Return JSON now:

If the solution from this back-translation matches the original solution from generation, the problem is considered verified. If not, the problem is discarded. This automated consistency check is inspired by the cycle-consistency approach in MathCAMPS and proved crucial to maintain a high-quality dataset. Only problems that passed this round-trip verification made it into the final benchmark.

3.4 Multi-Model Sieve for Tractability

Even with correct problems in hand, we want to ensure that our dataset covers a meaningful range of difficulties. A pitfall in automatic generation is that one might inadvertently produce many problems that are either too easy (solvable by even small models in one step) or too obscure (solvable only by a formal solver but not by any current LLM). To calibrate difficulty, we implemented a multi-model evaluation sieve. We selected a few reference models representative of various (limited) capability levels, including Llama-3.3 70B Instruct-Turbo, DeepSeek R1 Distill Qwen 14B, gpt-5-nano and google/gemma-3n-E4B-it and we attempted each generated problem with each of these models with the same JSON-answer format that we use for evaluation.

The results help inform us of the problem's *relative difficulty*. Our premise is that all individual questions should be solvable by at least one LLM in our strong panel (to ensure composability, which drives difficulty), but that if a majority of models in our weak panel also solve a composite problem correctly, the question is too easy.

Only fifteen individual questions (out of eleven hundred) that passed round-trip analysis were not solvable by at least one model in the strong panel. In lower difficulties (where difficulty is defined by the number of sub-problems), we discarded problems at a ratio of roughly 3:1 for being too easily solved by the weak panel. After difficulty level 4, the weak panel failed to solve almost all problems, however.

3.5 Difficulty Grading via Composite Problems

To systematically generate graded difficulty levels, we designed a composite problem builder that creates multi-step word problems by deterministically combining the simpler sub-problems without the use of an LLM. As stated, the difficulty level corresponds to the number of sub-problems included in the composite.

At difficulty level n (with $n > 1$), the system will pick n independent basic equation writeups (each a level 1 problem), and then join them into a single narrative that requires solving all n parts and then computing a final answer from those parts. The combination we chose is a sum: the final query asks for the sum of the answers to each sub-problem, with the answers being multiplied by arbitrary coefficients.

This question, drawn from the generated dataset at difficulty level 2, illustrates how the pieces fit together:

> "Solve the following. For each sub-problem:
>
> - Compute all real numeric solutions.
> - If a variable has multiple real solutions, select the numerically largest for that variable.
> - If the sub-problem has multiple variables, select the numerically largest value among those variables' selected values.
> - If no real numeric solution exists, use 0.0 for that sub-problem.
>
> Let sub_i denote the selected numeric value for sub-problem i. Compute the value of the linear expression shown below. Return only a single JSON object: {`"final_answer": <number>`}. Do not include any other text, markdown, or code fences. Use a plain number (scientific notation allowed).
>
> Equation and sub-problems follow (where sub-problem answers are specified by sub_1, sub_2, etc.):
>
> 8*(sub_1) + 6*(sub_2)

Sub-problem 1: Monster Energy renewed its sponsorship of North America's League of Legends Championship Series (LCS) through 2024. An esports league partnership pays a sponsorship bonus of $6 per tournament win. After a $12 administrative fee deduction, the team receives a net payment of $12. Let x be the number of wins. What is x?

Sub-problem 2: In a salt flat microbial ecosystem, a beneficial microbial process contributes 5 growth units per (x minus 6) ppt of salinity. A competing harmful process removes 7 growth units per (x minus 9) ppt. The net growth gauge shows a deficit of 2 units. Let x be the current salinity level in parts per thousand. What is x? Microbial mats in salt flats form layered communities where cyanobacteria provide essential nitrogen fixation."

In this example, the solver (human or model) has to tackle each sub-problem, then perform the final calculation. By varying the number of sub-problems, we create an approximately linear difficulty scale: level-1 problems are one-step, level-5 problems have five steps, etc. This is a deterministic process, meaning that for each difficulty tier we can generate arbitrarily many problems just by picking different base equations to plug in.

We further normalize difficulty by employing a greedy rule in composite problem construction: prefer adding sub-problems to the composite that include not-yet-included mathematical families, unless all families are already represented. This ensures a more even distribution of core mathematical challenges in lower difficulty problems, and at higher difficulty levels guarantees that all core math techniques being tested will be required to solve the composite.

A key benefit of our approach is controllability. We can guarantee that a level-n problem truly requires n distinct reasoning steps, because of how it's constructed. The approach also renders verification modular; the final answer is the sum of the most positive sub-problem solutions.

When we ask an LLM to generate the composite narrative, we prompt it to ensure that the pieces are connected but not interdependent; the only way to get the final answer is to solve each part. This design draws inspiration from the concept of evaluating chain-of-thought reasoning: a model must carry intermediate results and use them, which is a known difficulty for many LLMs on long multi-step problems. Our graded set allows us to observe how performance scales (or degrades) as the number of steps increases.

3.6 Evaluation Harness

We evaluate models with a standardized harness that unifies API calls across providers via thin adapters. For each item, the dataset's question text is sent verbatim to the model (the harness does not add its own wrapper prompt). The harness expects a numeric answer and prefers a strict JSON object containing a key named "final_answer". When models don't return strict JSON, we use tolerant parsing to recover a number: we first try fenced JSON blocks, then embedded JSON fragments, then a regex "final_answer: ", and finally (as a last resort) the last standalone number in the reply. Non-compliant formats are not automatically counted as incorrect; if a numeric answer can be recovered, it is graded normally. The parsing mode is recorded for transparency.

Scoring uses percent error against the ground truth numeric answer. A prediction is marked correct when the percent error is $\leq$ a user-configurable threshold (–error-pct; default 0.0 for exact match). For near-zero ground truths, we apply a small absolute tolerance ($\leq$1e-12) to avoid division by zero artifacts. We report accuracy (fraction correct) per difficulty tier and overall.

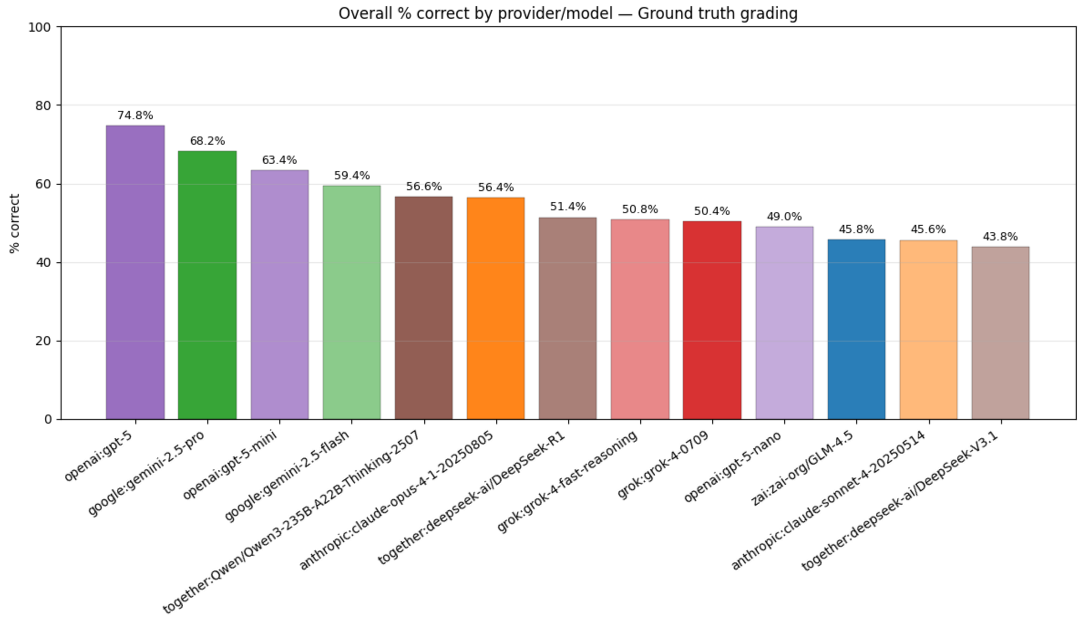
Runtime controls are standardized across providers: we cap output tokens with –max-tokens (default 50,000) and enforce per-request timeouts with –request-timeout (default 300 seconds). The harness includes simple backoff/retry and optional rate limiting.

Because the dataset is synthetic and reproducible, we publish problems and answers and log per-call artifacts (raw response, parsed value, parsing mode, timing, and provider usage). This makes the benchmark easy to reuse and refresh; in this paper we treat the current snapshot as a fixed test set for comparability.

Results

We generated a preliminary evaluation set comprising 10 tiers of difficulty with 50 problems in each tier (for a total of 500 problems). These span single-step algebra questions at Level 1 up to complex 10-step problems at Level 10. All problems were verified for correctness and categorized by difficulty as described. We then evaluated a range of language models on this dataset, including open-source models and leading closed-source models, to obtain a comprehensive view of performance.

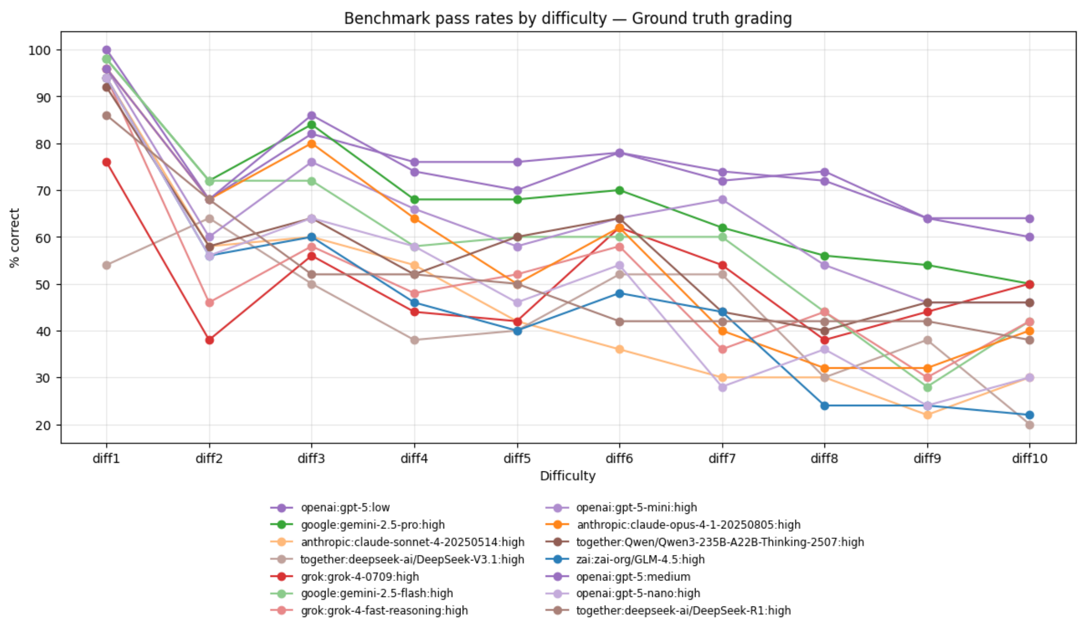
Here's what that looked like at a Provider / Model level:

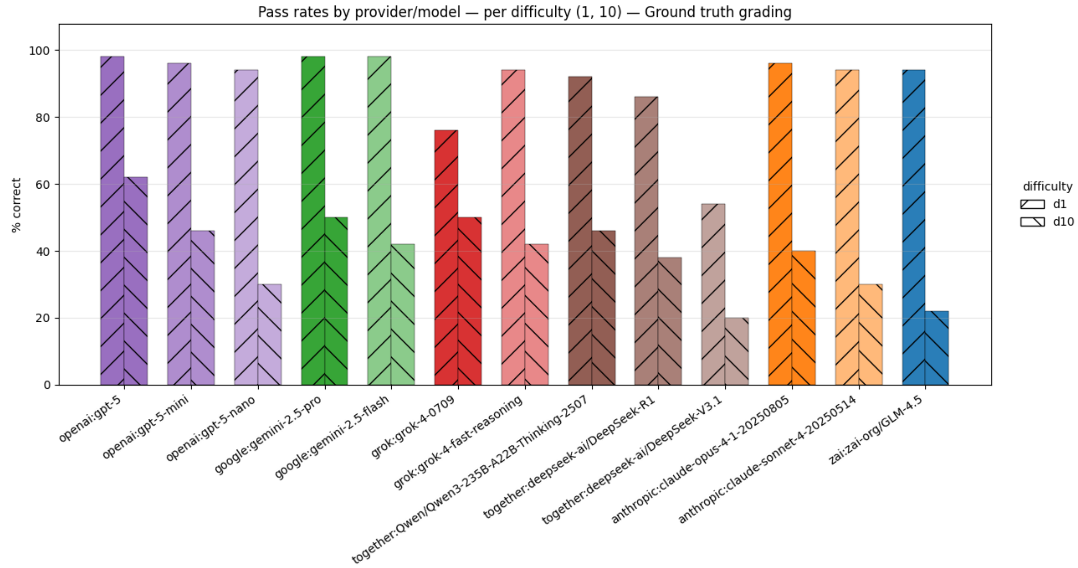


Notes: We did not include gpt-5 high in our benchmark, as we did not find it to be appreciably more accurate than medium in our early test runs, but costs were excessive. We attempted to include both GLM 4.5 and Kimi, but these models exhibited (on Z.AI and Together.ai, respectively) extremely inconsistent performance on the benchmark. We eventually ran GLM 4.5 on our own server with high timeouts and token limits and were able to brute force our way through the benchmark (many responses simply timed out and had to be retried). Thus, GLM 4.5 results may be considered higher than what is normally attainable.

We did find it surprising that gpt-5 low outperformed medium, but this result is well within the margin of error. We imagine that running a much larger number of samples would produce slightly smoother charts, but this became cost / time prohibitive.

Overall Performance

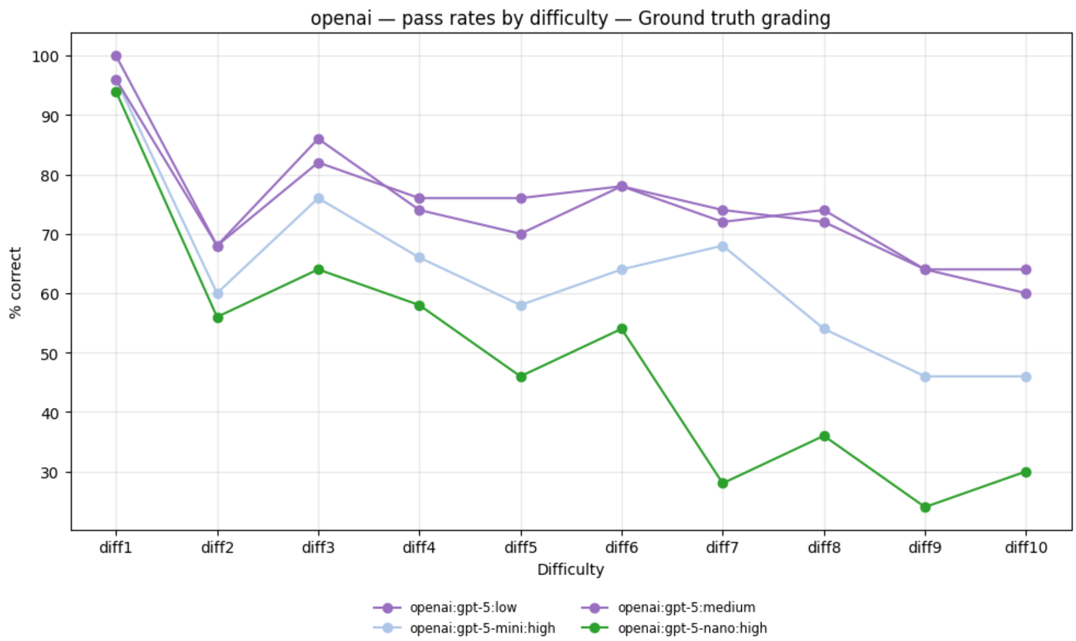




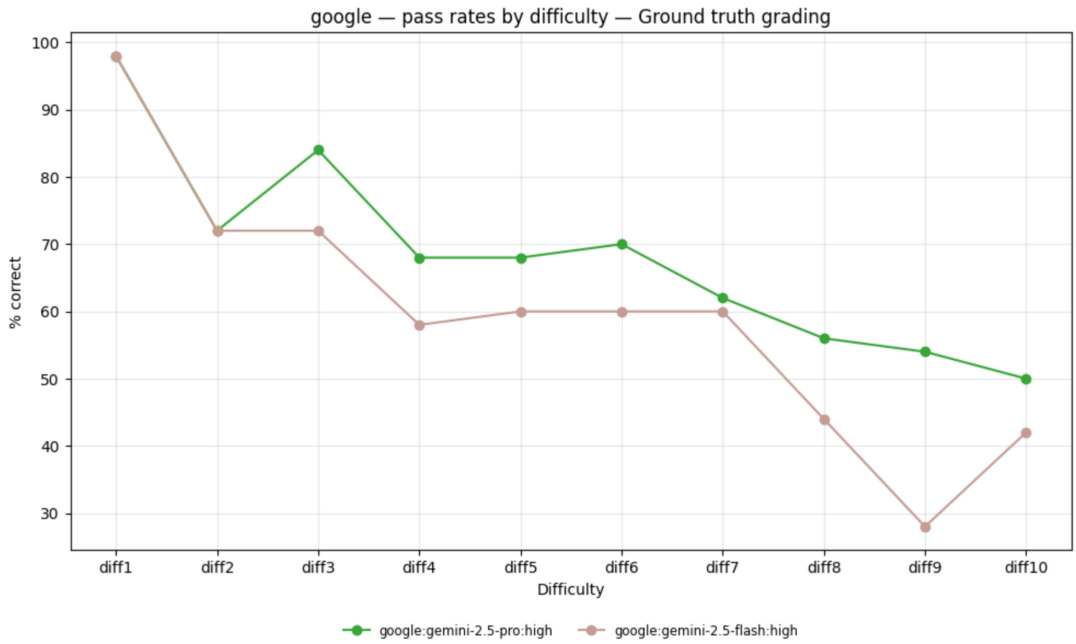
Without exception, model performance degraded as we increased the number of sub-problems. When models missed problems, error percentages tended to be substantial, indicating that we were not looking at floating point artifacts or rounding issues simply due to a larger number of mathematical approximations being performed. Similarly, correct answers were often correct to the sixth decimal place.

Difficulty did not seem to impact all models linearly or in the same way, as you can see by comparing Grok to DeepSeek 3.1, for example. In this case, DeepSeek did better on diff2 than at diff1, whereas Grok's performance dropped radically from diff1 to diff 2.

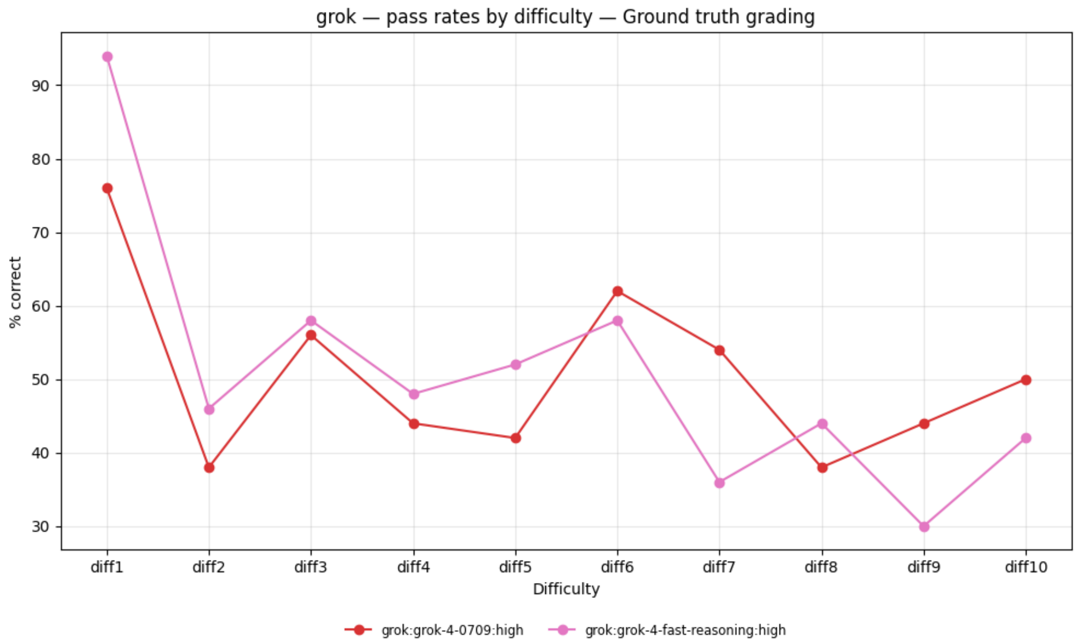
Looking at OpenAI's charts versus Google's, Grok's, and Anthropic's is fairly interesting:



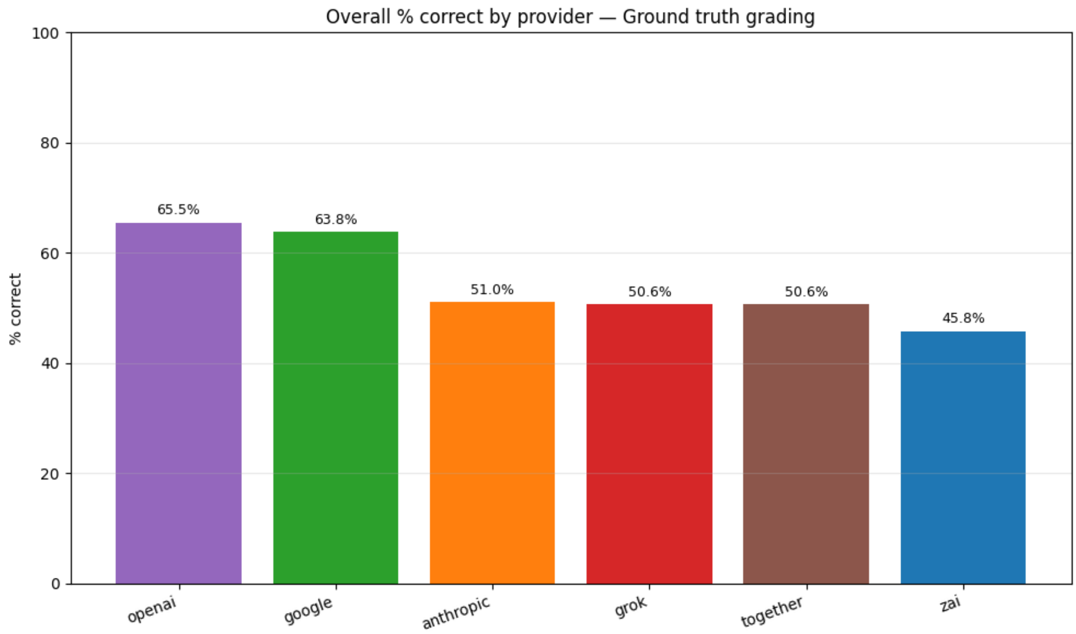
OpenAI's models exhibit some broadly similar tendencies as difficulty increases, but performance suffers much more on nano and mini, whereas medium and low trade blows until stalling out at around 65% accuracy at difficulty 10.



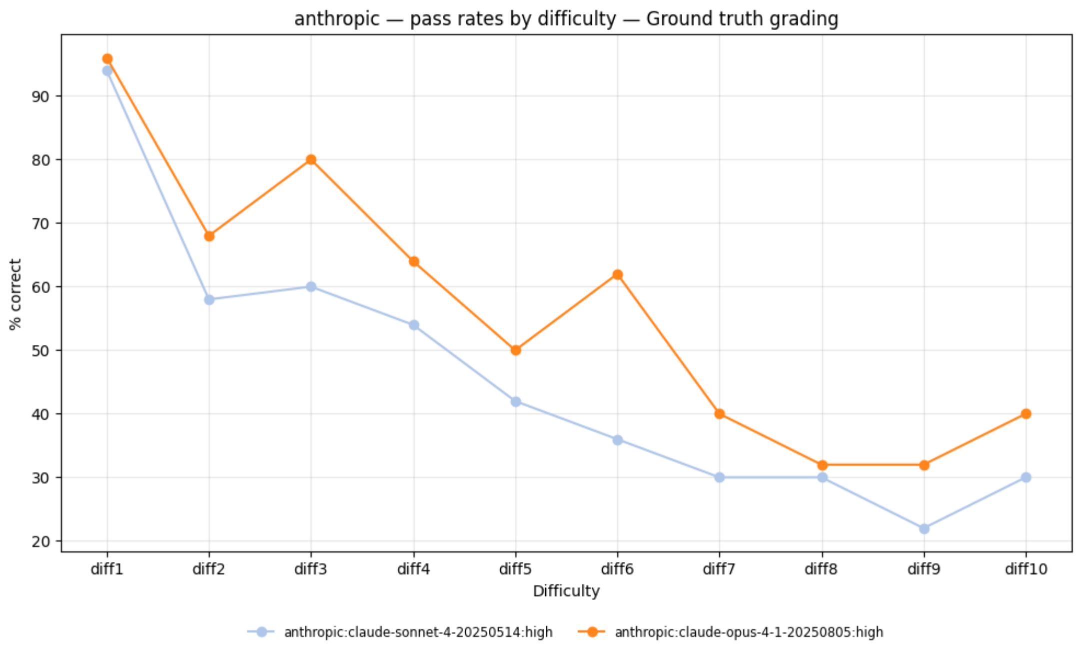
Google's models also follow each other closely but degrade much more rapidly than OpenAI's models after difficulty 6. This is particularly interesting because Google is known for its long context prowess.



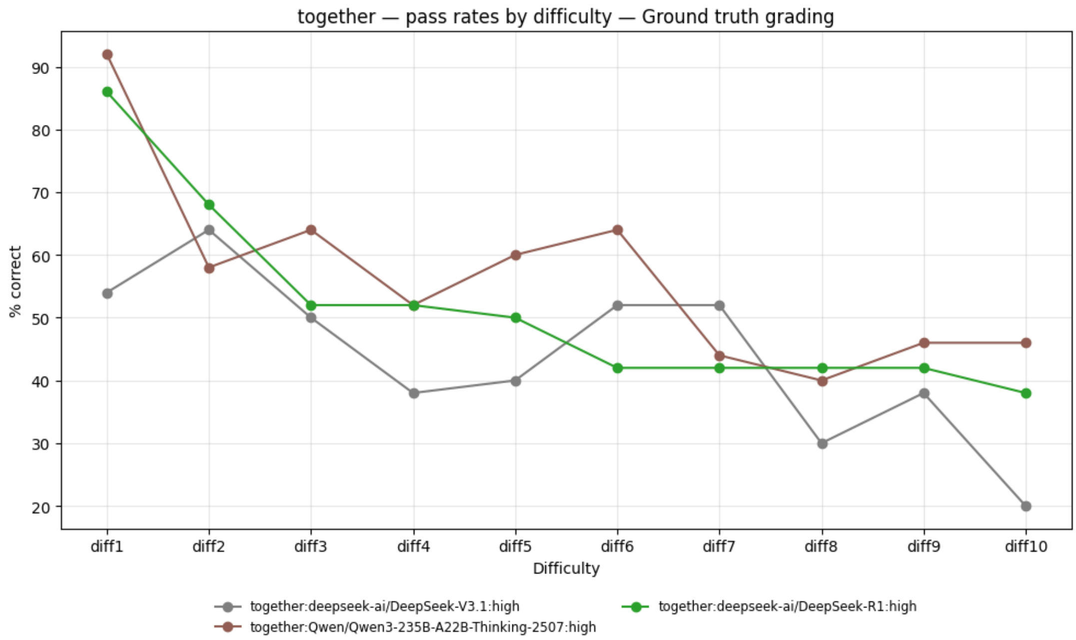
Grok's performance on this benchmark was, to put it mildly, all over the place, making us wonder if there are some poorly tuned dynamic reasoning / routing algorithms at work behind the scenes. This is particularly noteworthy given Grok's touted prowess in math. Nonetheless, it's worth noting that Grok's lowpoints are mostly higher than Anthropic's (see the next discussion) and it oddly starts getting better at difficulties 9 and 10, to the point where it's neck and neck with Google but still lagging OpenAI. Overall, however, it ranked last in our closed source comparison by six percentage points, as seen in the chart below.



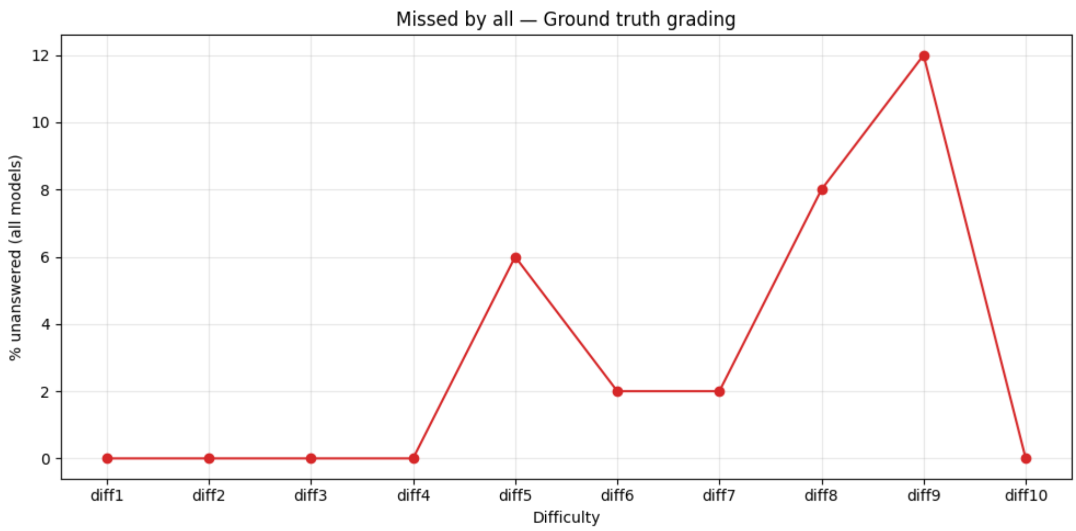
Next we turn to Anthropic.



Anthropic starts strong but fades even more quickly than Google on this test. It ends up with the second weakest scores of the closed source bunch, with Opus being equivalent to or worse than Gemini Flash but better than Grok.



On the open source side, DeepSeek 3.1 and R1 turn in a weak performance for such large models (680B parameters), being outclassed by the more recent Qwen 235B. Qwen was largely on par with Opus and Gemini Flash.



On the whole, we have relatively low global failure percentages. No problem is missed universally until we hit difficulty level 5. Difficulty level 5 is randomly harder than difficulty levels 6 and 7. Difficulty 9 generates peak failure rates of 12%, but oddly, difficulty ten seems to become tractable again. Recall that these composites contain only combinations of individual problems that were previously *solved* by an LLM, so these results are interesting and bear further scrutiny. Since we only have 50 problems per difficulty, it is quite possible that the variability we see is due to noise.

Overall, the preliminary results validate that our synthetic benchmark is challenging and discriminative: it separates models by capability and reveals fine-grained differences that static benchmarks might obscure (especially if those were already saturated). The absolute scores also indicate substantial room for improvement. No model achieved above ~75% on our test, whereas by comparison human performance on these problems (for a mathematically trained person) would likely be near 100% on levels 1-5 and perhaps 80-90% on levels 6-10 (with some time pressure or fatigue being a factor on 10-step problems). Thus, one could say the best models are *approaching human expert-level* on short problems but still lag on longer ones. Our difficulty is also easy to adjust; it's likely that twenty sub-problem versions of this benchmark would crush even the best models of today.

Analysis and Discussion

Subjectivity in Problem Interpretation

One possible objection to the benchmark might be that higher difficulties are based on a chain of assumptions; if the model's interpretation of any sub-problem differs at any point from what is expected, then such an error will inevitably cause a complete breakdown. In this sense, the benchmark is indeed harsh, but also realistic; we imagine that cascading failure scenarios will become an increasingly important concern as society comes to rely more on agentic versions of these models.

Nonetheless, it's also possible that dynamically generated problems can be legitimately ambiguous. Herein, we examine one such scenario, the salt flat example from earlier:

> *In a salt flat microbial ecosystem, a beneficial microbial process contributes 5 growth units per (x minus 6) ppt of salinity. A competing harmful process removes 7 growth units per (x minus 9) ppt. The net growth gauge shows a deficit of 2 units. Let x be the current salinity level in parts per thousand. What is x?*

The original equation was: "Eq(5/(x - 6) - 7/(x - 9), -2)." The DeepSeek R1 5/28 model did the conversion from an equation to a narrative for this problem, then converted back, and thus the problem passed the round trip check. When posed directly with the word problem however, DeepSeek first wrote:

> "First, I should write down the net growth equation. The beneficial contribution is 5 per (x - 6), so that would be 5 / (x - 6) growth units. Similarly, the harmful removal is 7 per (x - 9), so that's -7 / (x - 9) growth units. The net growth is the sum of these, and it's given as a deficit of 2 units, so net growth = -2."

then, it second guessed itself, saying:

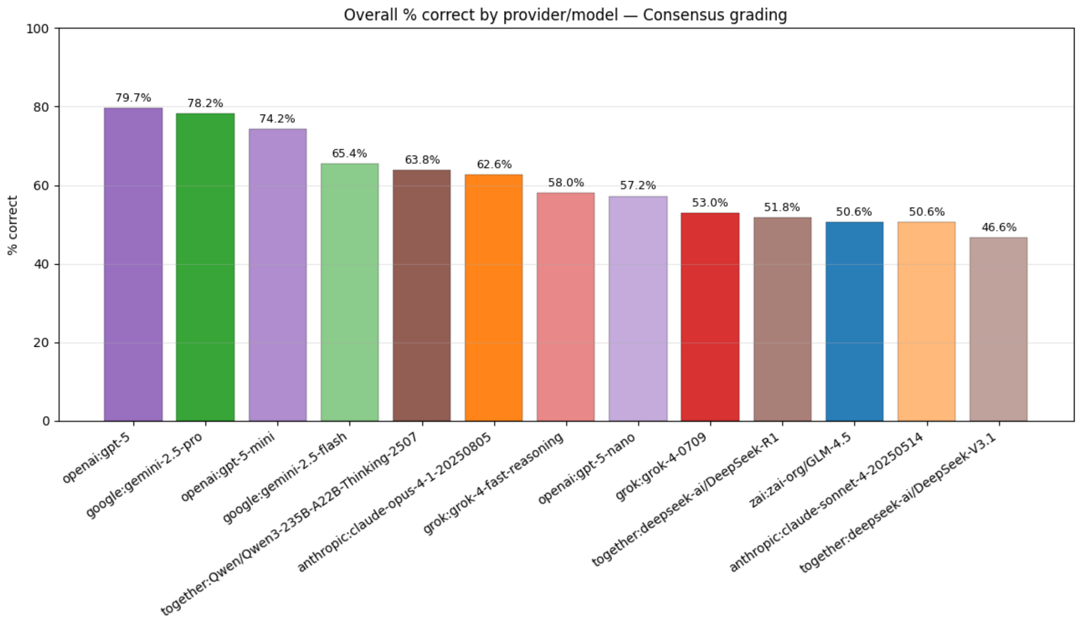
> "So mathematically both work, but in context, for x=4.609, the beneficial process is contributing negative growth, which might not be "beneficial," and harmful process is removing negative, so adding growth, which is not "harmful." So probably the context implies that salinity is such that the processes behave as described, so x > 6 for beneficial to be positive, and x > 9 for harmful to be negative, so x>9, so x≈11.39 is the intended solution. But the problem asks for exact value, and I have it in terms of sqrt(46), but perhaps I made a mistake because it should be nice number. Perhaps I misinterpreted the "per" part. Another way to interpret: "contributes 5 growth units per (x minus 6) ppt" might mean that the contribution is 5 * (x - 6) growth units, but that would be per ppt or what? The "per" is confusing."

It finally decided to reinterpret per to mean multiplication, and then incorrectly answered the problem.

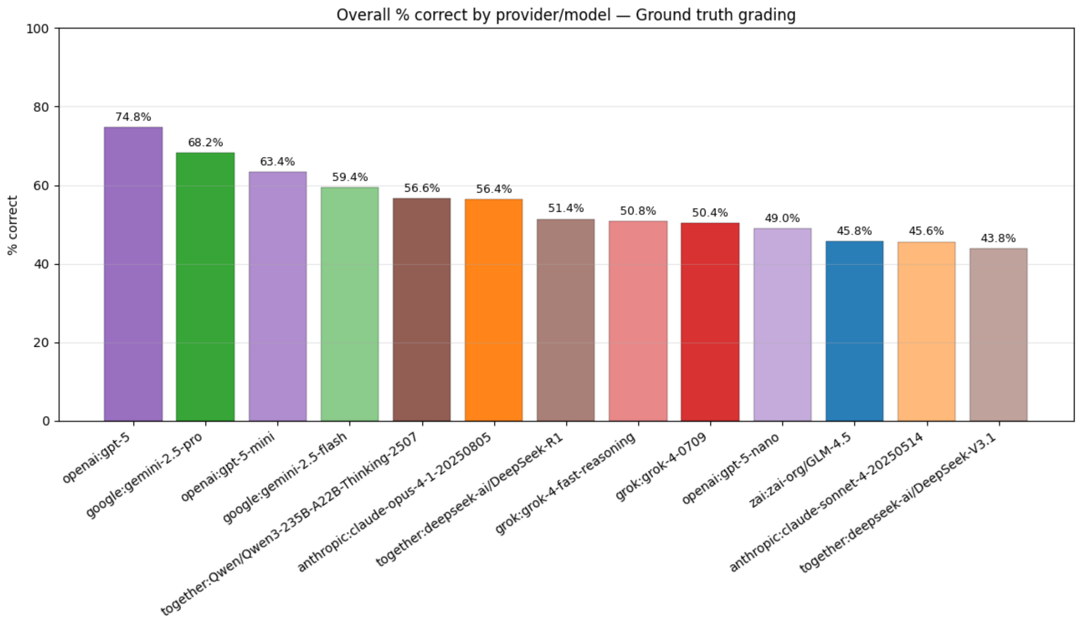
GPT5 (thinking high) will indicate within Codex that division is the correct algebraic expression for this problem. GPT5 in Chat will indicate that because the domain is biology rather than physics, multiplication is the correct expression. GPT5 nano outputs the expression as division.

The question then becomes: *Are model performance differences simply due to DeepSeek choosing unconventional, ambiguous, or inappropriate problem framings?*

This should be straightforward to answer if we regrade our whole dataset using the most popular answers rather than the ground truth, since such an approach will inherently reject non-consensus interpretations. For this exercise, we require that at least two models must have agreed upon one answer (with a .5% tolerance), otherwise we substitute the ground truth as the point of comparison. Here are some results of that analysis:



If we compare this to result to our original chart

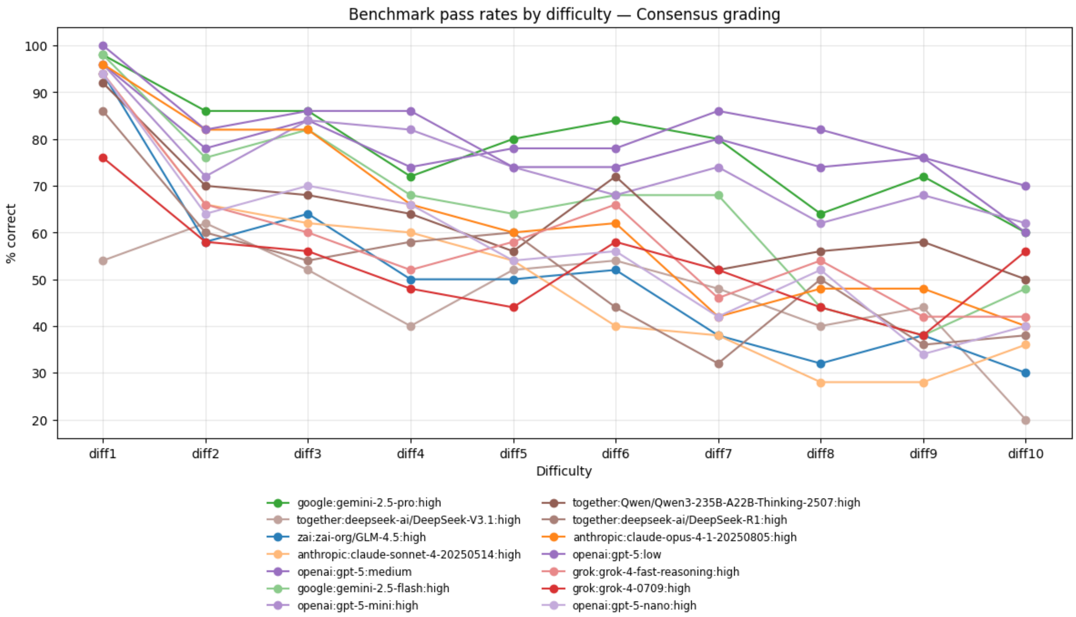


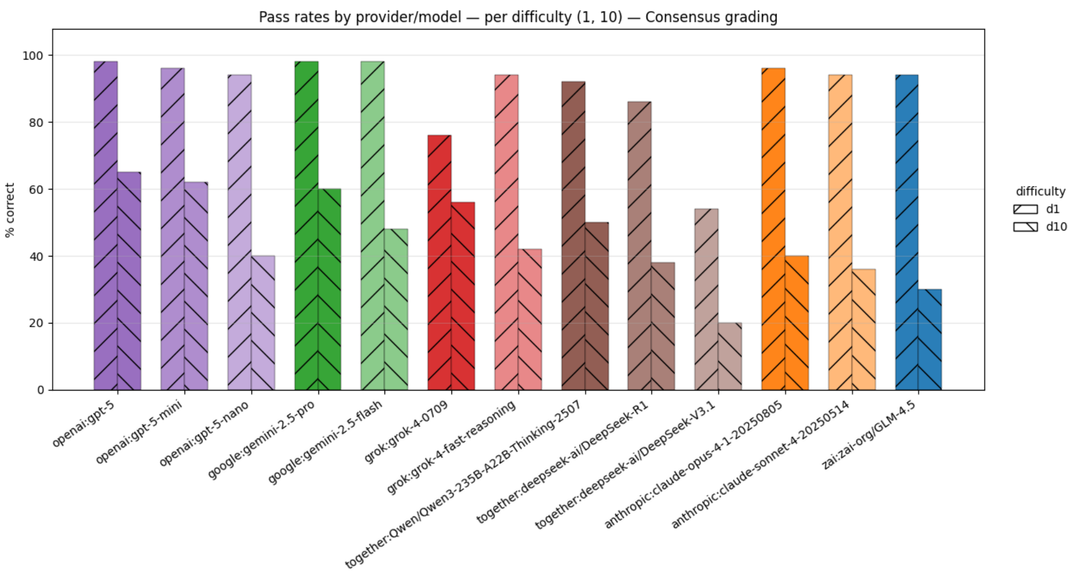
then it is clear that the relative positions of the top models seem to be mostly conserved, even though every single model in the comparison shows higher accuracy percentages under the consensus method. OpenAI's GPT-5, for instance, jumps from 74.8% accuracy when measured against ground truth to 79.7% under consensus grading. This pattern of inflation is even more pronounced in some other models, with Google's Gemini-2.5-Pro seeing a dramatic increase from 68.2% to 78.2%, and OpenAI's GPT-5-mini rising from 63.4% to 74.2%.

What makes this particularly interesting is that the score inflation isn't uniform across all models. While Gemini-2.5-Pro and GPT-5-mini experience gains of approximately 10-11 percentage points, GPT-5 sees a more modest increase of only about 5 percentage points. This differential impact suggests that GPT-5's responses tend to align more closely with ground truth even in cases where other models might disagree.

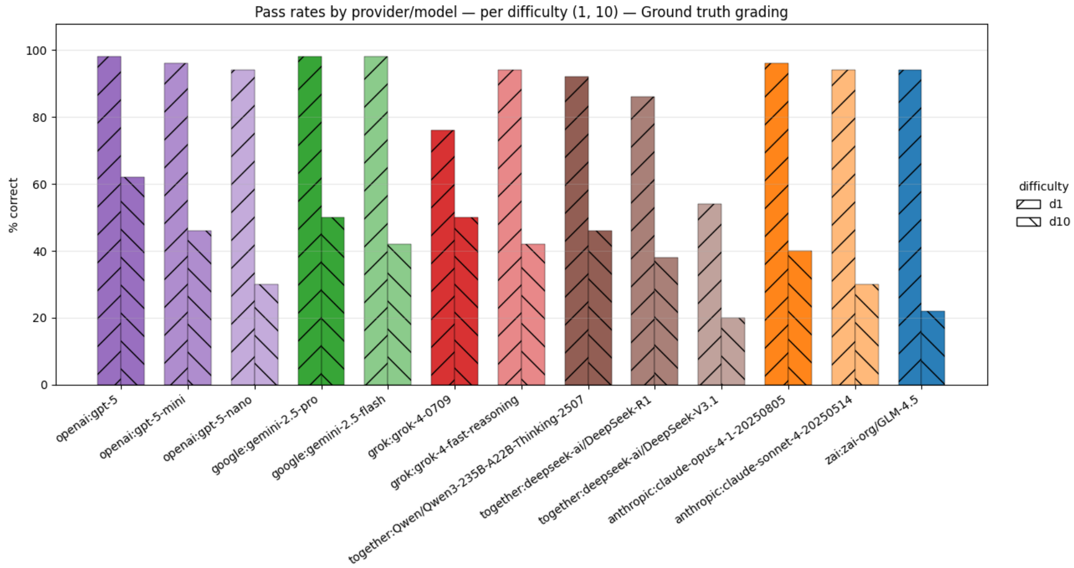
The relative performance gaps between these leading models does change dramatically based on the grading method. Under consensus grading, the top three models cluster tightly together with scores of 79.7%, 78.2%, and 74.2%, creating an impression of near-parity in performance. In contrast, when evaluated against ground truth, these same models show much clearer separation at 74.8%, 68.2%, and 63.4%, revealing more substantial differences in their actual capabilities.

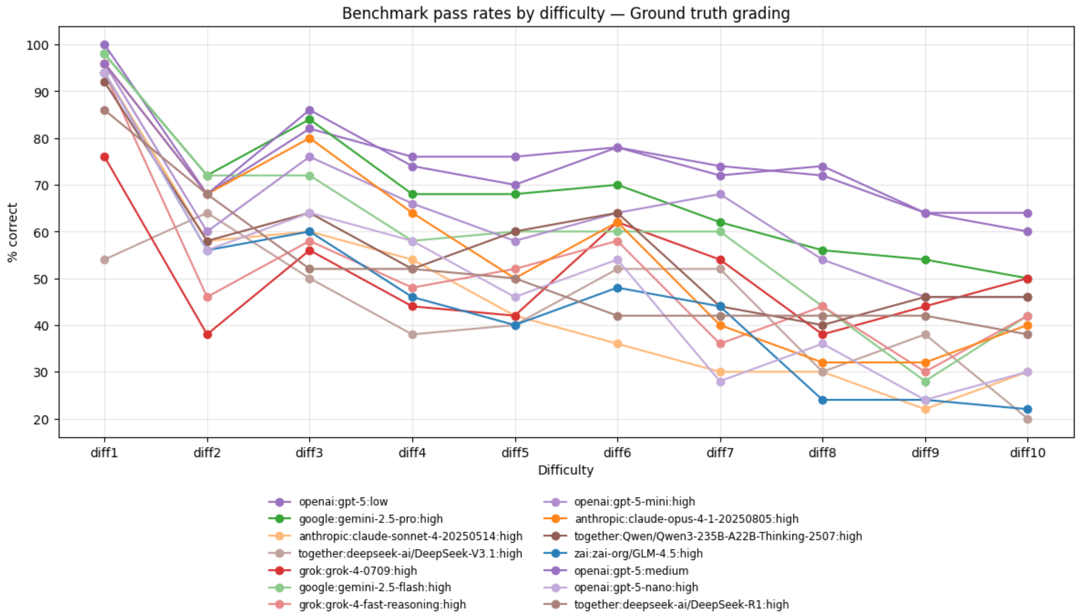
Examining score degradation is similarly illuminating. Here is the consensus progression:





As compared to the original progression:

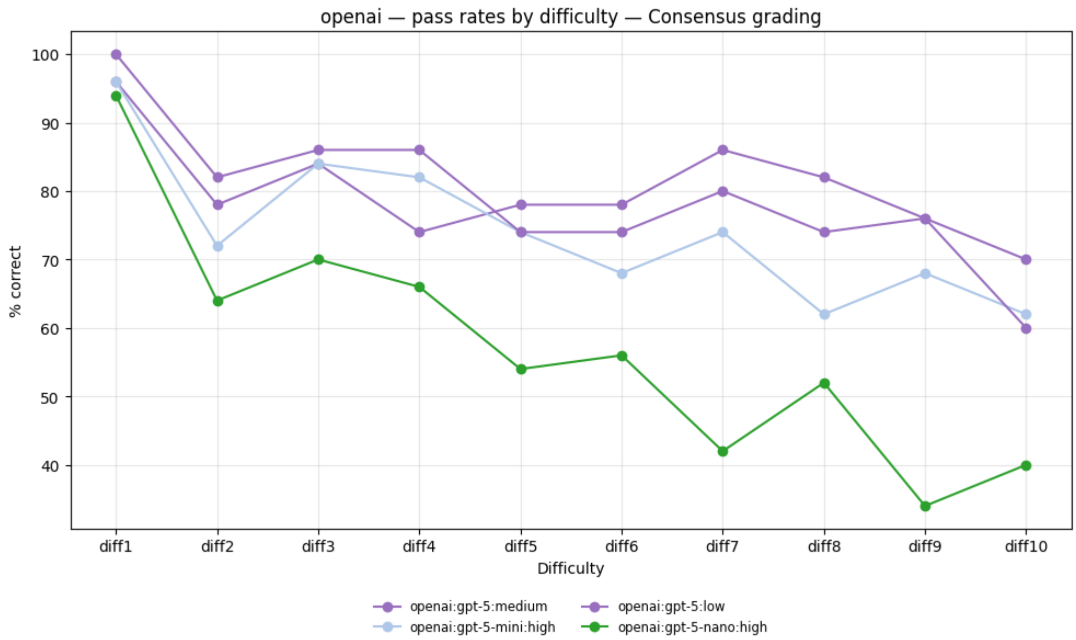


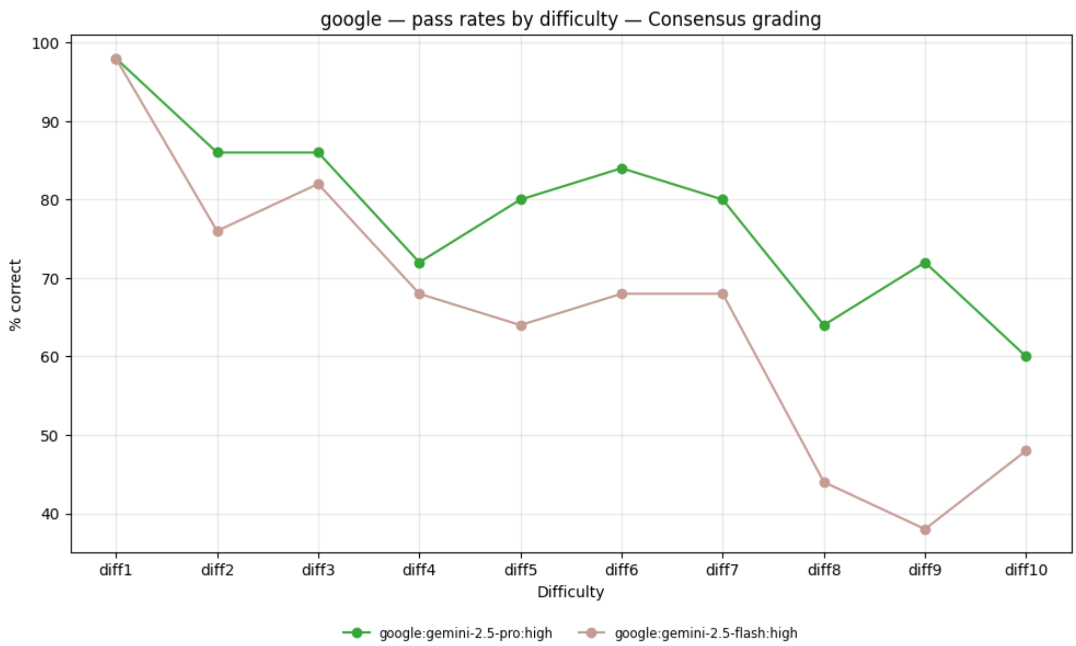


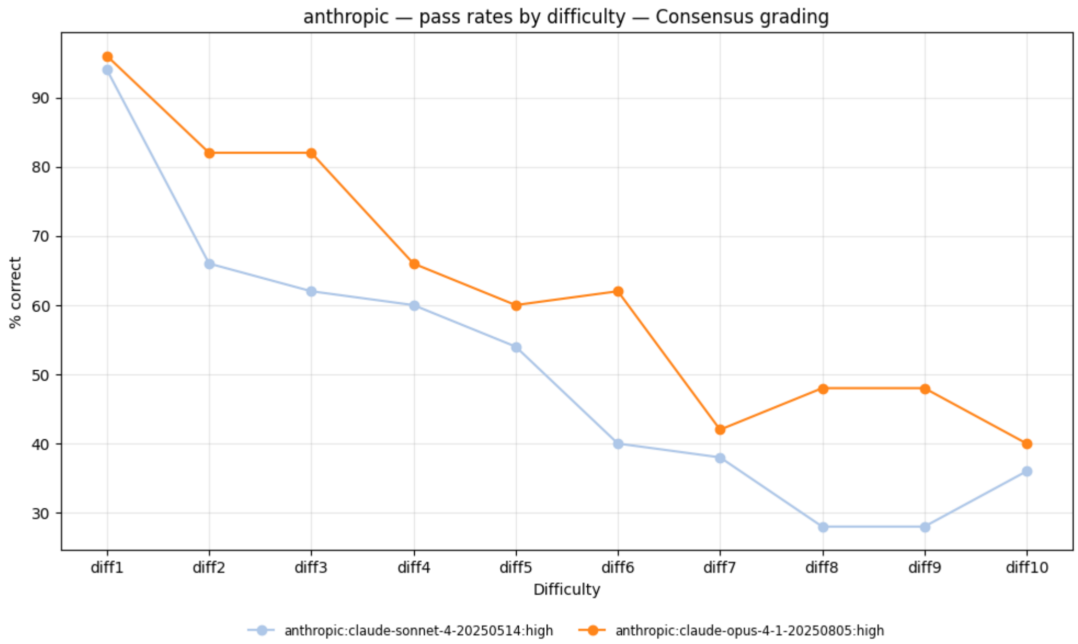
Among the strongest models—GPT-5, Gemini 2.5 Pro, and Claude Opus—the overall decline is steady but less catastrophic. GPT-5, under consensus, begins at near-perfection (96–100%) and retains 60–70% accuracy by the hardest problems. Under ground truth, the same model falls more sharply, losing roughly 40 points from start to finish and ending closer to 60%. Gemini 2.5 Pro shows a similar pattern: a decline from 98% to 60% with consensus grading, but to 50% under ground truth. Claude Opus follows the same trajectory, beginning at 96% and slipping to 40% by level 10, with consensus presenting the smoother descent. For these strong models, both regimes show that high difficulty introduces real fragility, but consensus softens the perceived blow.

The Grok models highlight another angle. Grok-4 Fast Reasoning begins at 94% and drops to 42% under both grading schemes, showing that consensus and ground truth converge when errors are inconsistent across attempts. Grok-4 0709, however, diverges: consensus records a fall from 76% to 56%, while ground truth penalizes it harder, from 76% to 50%.

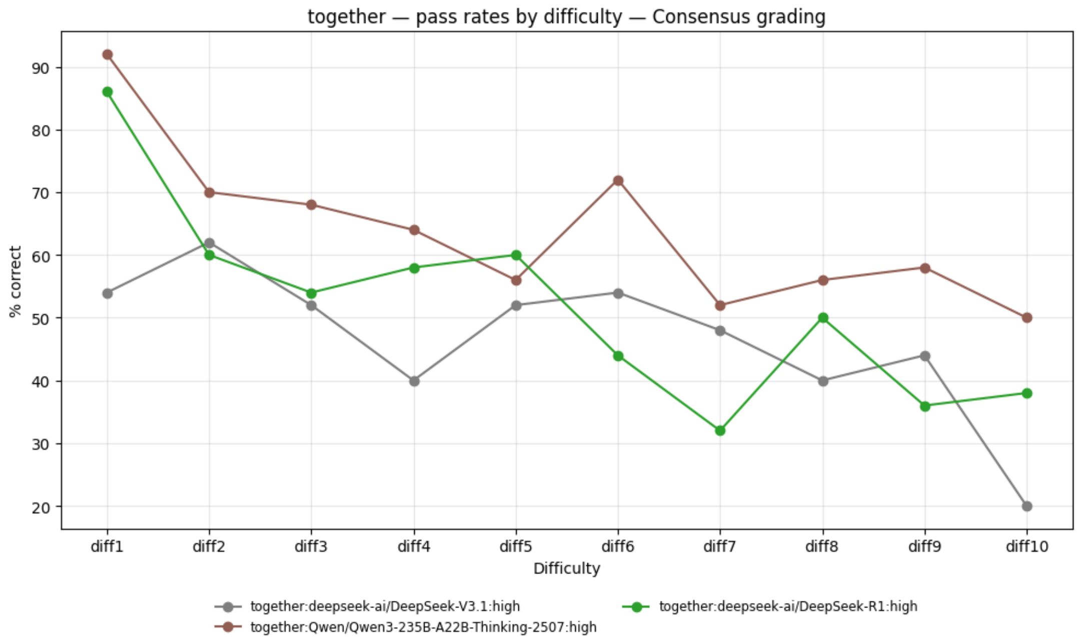
Here are some of the consensus progressions:

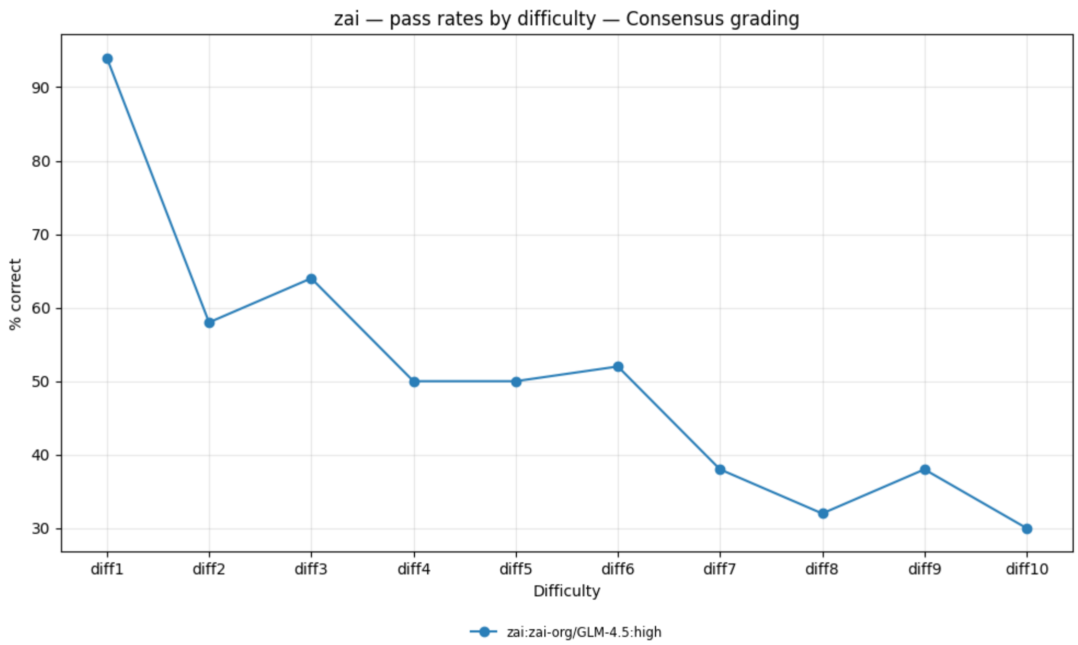

openai — pass rates by difficulty — Consensus grading
% correct
100
90
80
70
60
50
40
diff1
diff2
diff3
diff4
diff5
diff6
diff7
diff8
diff9
diff10
Difficulty
openai:gpt-5:medium
openai:gpt-5:low
openai:gpt-5-mini:high
openai:gpt-5-nano:high


google — pass rates by difficulty — Consensus grading
% correct
100
90
80
70
60
50
40
diff1
diff2
diff3
diff4
diff5
diff6
diff7
diff8
diff9
diff10
Difficulty
google:gemini-2.5-pro:high
google:gemini-2.5-flash:high


anthropic — pass rates by difficulty — Consensus grading
% correct
Difficulty
diff1
diff2
diff3
diff4
diff5
diff6
diff7
diff8
diff9
diff10
anthropic:claude-sonnet-4-20250514:high
anthropic:claude-opus-4-1-20250805:high


grok — pass rates by difficulty — Consensus grading
% correct
Difficulty
diff1
diff2
diff3
diff4
diff5
diff6
diff7
diff8
diff9
diff10
grok:grok-4-fast-reasoning:high
grok:grok-4-0709:high





Through these graphs, we see that mid-tier performers like GPT-5 Mini, Gemini 2.5 Flash, and Qwen-3 235B reveal a widening gulf between consensus and strict grading. GPT-5 Mini falls from 96% at the start to 62% at level 10 under consensus, but drops further to 46% under ground truth. Gemini Flash moves from 98% to 48% in consensus, compared to 42% in ground truth. Qwen traces the same arc, losing about 40 points in consensus but closer to 50 in ground truth. These mid-range models degrade faster than the leaders, and consensus grading cushions their decline by 5-15 points.

Weaker models like DeepSeek V3.1, DeepSeek R1, and GLM-4.5 suffer the steepest collapses. DeepSeek V3.1 begins at only 54% and finishes at 20% in both regimes. DeepSeek R1 performs more variably: it starts stronger, at 86%, but ends in the high 30s. GLM-4.5 shows the most dramatic deterioration: from 94% down to 30% in consensus, and all the way to 22% under ground truth. For these models, consensus grading offers only marginal relief—the severity of decline is inescapable.

Looking across the spectrum, stronger models degrade by about 30-40 percentage points under consensus and 40–50 under ground truth, holding on to majority correctness even at the hardest levels. Weaker models degrade more brutally, often losing 60–70 points, with ground truth exposing the lowest floors of performance. Mid-tier models fall in between, but consensus grading consistently cushions their results more than it does for the leaders.

The overall conclusion of this analysis is that subjectivity undoubtedly plays a role in the raw ratings, though less so in the absolute rankings. In particular even assuming maximal subjectivity:

1. Difficulty progression is preserved
2. Model separation at higher difficulties is preserved
3. Model rankings are very similar

These findings indicate that the benchmark itself is still valid and useful, though we may benefit from using a stronger model for roundtrip analysis in the future.

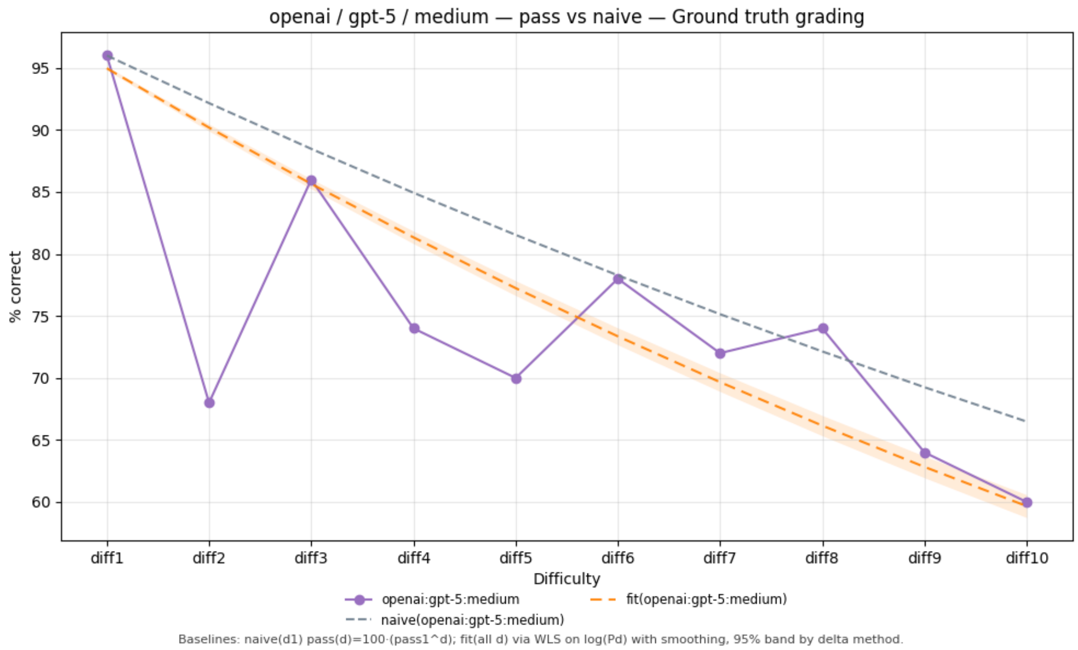
Expected vs Actual Performance Inconsistencies

The difficulty of the benchmark is deliberately designed to be linear; the more sub-problems added, the more difficult the problem becomes. Problems are fully independent of each other until the final step. Given this state of affairs, it's possible to produce different estimates of difficulty and graph model performance against these.

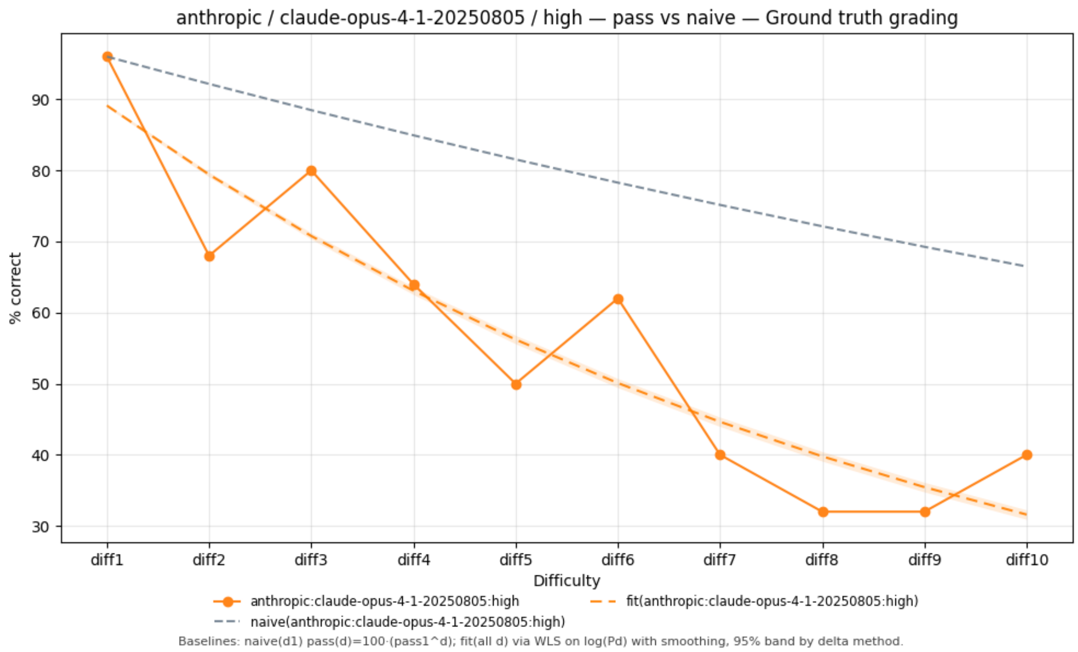
To build a naive estimator, you would simply look at the performance of the model at difficulty 1 (one sub-problem) and then extrapolate a corresponding drop in performance per sub-problem added. For example, at a 95% success rate, difficulty 2 expected success could be calculated as (.95 * .95 = .9025).

We can also model "expected difficulty" of a d-way composite as the probability that an LLM correctly solves all d independent sub-problems, each with a common per-sub success probability p. Under this iid approximation, the expected composite pass rate is $P(d) = p^d$. We estimate p by weighted least squares on the log scale, fitting $\log P(d) = d \cdot \log p$ with zero intercept to the observed pass proportions across difficulties. For each difficulty d we compute a smoothed empirical pass rate $Pd = (Sd + 0.5)/(Nd + 1)$ from Sd successes out of Nd trials, and regress $y\_d = \log Pd$ on $x\_d = d$ using weights $w\_d = Nd$, which approximate the inverse variance of the binomial proportion and stabilize the fit when counts vary by d. The resulting slope $\hat{s}$ yields $p = \exp(\hat{s})$, and the expected pass curve is $P(d) = p^d$. We quantify uncertainty by estimating the weighted residual variance σ^2 and using $\mathrm{Var}(s) = \sigma^2 / \sum_d w_d x_d^2$, from which a 95% confidence band for $P(d)$ follows by the delta method applied to $\exp(d \cdot \hat{s})$. Comparing the empirical pass curve to $P(d)$ distinguishes "actual" from "expected" difficulty under the independence model: sustained performance below the fitted curve indicates underperformance relative to the per-sub expectation; alignment within the band suggests consistency with the model; and persistent outperformance suggests easier-than-modeled composites.

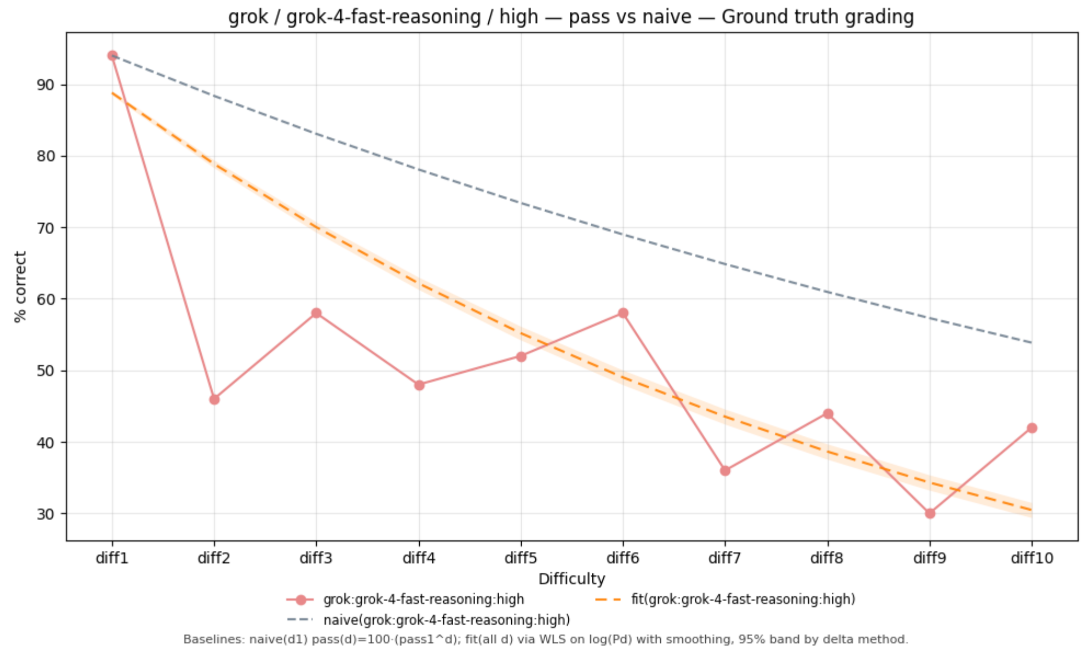
Here are some of the views of the leading models from this vantage point. The orange band is the confidence interval around our WLS estimate. The gray line represents the naive expected performance based on difficulty 1.

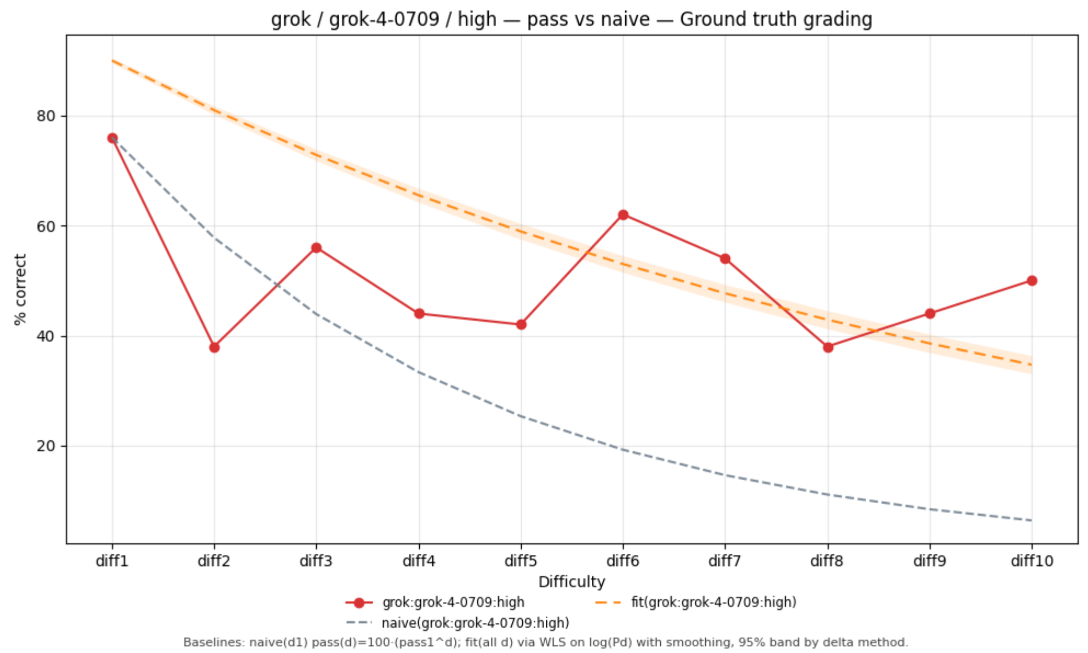


GPT5 has overall excellent performance but it underperforms estimates (sometimes severely) until difficulty 6, whereupon it kicks into gear. Difficulty 1 provides a pretty reasonable estimate of GPT5's capabilities.

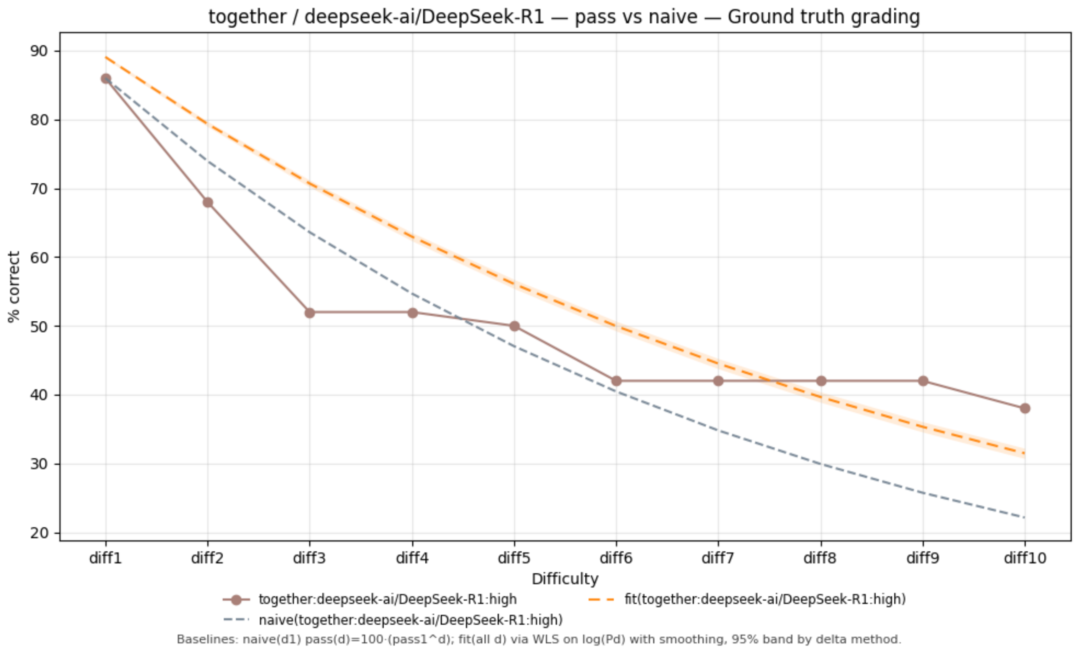


Difficulty 1 significantly overestimates Opus' capabilities; it proceeds to turn in a performance very much in line with an 89% base accuracy rather than a 98% accuracy.





Difficulty 1 underestimates the Grok family's capabilities. Grok 4 and Grok 4 fast significantly underperform until difficulty 6, then also start to perform better again. Their progression is quite similar to that of Opus.



R1 here looks roughly comparable to Grok, though it underperforms until difficulty 8.

While these are preliminary results, one takeaway is that these models do not perform nearly as consistently as we would like. It's possible that much larger sample sizes would smooth this out, but this is still somewhat surprising to see such a divergence.

Use for Evaluation

Of course, the benchmark itself is more of a curiosity at this stage (before we scale up its execution), and the metrics described herein are exploratory in nature. The primary value of our system is to provide a *living benchmark* for mathematical reasoning in LLMs. Unlike traditional benchmarks that are static and eventually compromised by exposure, our pipeline can generate new problem sets on demand, indefinitely. This means evaluators can regularly assess models on fresh, never-before-seen questions, greatly reducing the chance of evaluation metadata leaking into training. From the perspective of the Ambient network (Ambient.xyz), this fits into our vision of continuous verification: miners can be tasked with regularly generating and solving new challenges as a proof of their work, and proposed new model fine tunes can be tested against automated benchmarks like this one that are dynamically generated by the network itself. Our results show that even simple algebra problems, when combined in novel ways, can remain difficult for current models so there is rich ground to test model improvements in the coming iterations.

Because our dataset is open, it also invites community scrutiny. Others can analyze the problems and solutions to identify any biases or artifacts. For instance, are there linguistic patterns that could cue a model (or conversely confuse it)? Does the dataset accidentally emphasize certain topics more than others? Initial inspection suggests a good variety; thanks to the LLM's creative freedom the word problems span many scenarios. If there are consistent issues, it seems like these are associated with sign flips in relation to credits, debits, and domain specific conventions. By releasing everything, we enable others to point out such issues and contribute fixes and/or enhancements. This openness is part of Ambient's ethos: building AI infrastructure collaboratively, with transparency.

Use for Training

An exciting aspect of synthetic data is that it can be used to *improve* models, not just test them. Since we control the generation, we can create arbitrarily large corpora of math problems and use them to fine-tune models, as long as we ensure enough variety. If a model is trained on a dataset and then tested on it, evaluation is no longer valid, but because we can regenerate new sets, we are able to use the pipeline to generate a large training set (maybe tens of thousands of problems), fine-tune a model on it, and then generate a fresh test set from the pipeline's current settings to evaluate. We expect (similar to OpenMathInstruct results) that fine-tuning even a mid-sized model on a high-quality synthetic math corpus would markedly boost its performance. In particular, the multi-step reasoning traces within our data could teach models to maintain reasoning consistency over longer contexts, a known weakness. We're working on some early experiments here and will report more in a subsequent paper.

Another potential use for our work is as a curriculum. Because we label difficulty, one can employ curriculum learning: start training the model on easy problems (level 1-2), then gradually introduce harder ones (level 3-4, and so on). This might help the model build up step-by-step reasoning skills. We could even generate solutions or chain-of-thought annotations for each problem using a strong model and include those in training to teach intermediate reasoning steps (see the post-hoc reasoning flag in the code). This becomes a rich supervised training signal; the model isn't just learning the final answer, but how to systematically arrive there. Ambient's infrastructure could automate this process, continuously generating new training data and incorporating it into model updates, creating a self-improving loop. We recognize the risk of a model then overfitting to the style of synthetic data, but since the data generation can adapt (e.g. varying style, problem types, and adding noise), we believe the loop can be kept from collapsing.

Limitations:

Our approach has some straightforward limitations. First, it currently focuses on a specific genre of math problems - primarily linear and arithmetic word problems. We have not included geometry, calculus, or others that might need diagrams or more complex solution structures. Extending to those domains might require new tools (e.g., a geometry solver or different prompting strategies). Second, while the LLM-generated narratives are usually reasonable, they can occasionally be unnatural or repetitive. This is a minor issue, but for training data it might introduce stylistic biases. Third, it seems that round-trip filtering is not, on its own, enough to completely remove ambiguity from the problems generated. When prompted with the problems themselves, a model may still fail even though the same model was responsible for checking the round trip. We did an entire additional run with problems based on a different round-trip verifier (GPT5-mini instead of DeepSeek R1 5/28) and achieved almost identical rankings and performance; this is not just a DeepSeek issue. Fourth, the multi-step combination by summing sub-answers is somewhat formulaic. We did try some experiments to use LLMs to create more creative compositions, but found that current models struggle mightily with this task (which might be a fun benchmark on its own).

In future work, we'd like to incorporate more interdependence between sub-parts (e.g., the answer of part 1 is used in part 2's scenario) to mimic real multi-step problems, rather than just independent pieces. This would make the reasoning more complex and less brute-forceable by solving parts independently.

Another limitation is the reliance on an LLM (often a very strong one) to generate and verify problems. This means the quality of our dataset is tied to the capabilities of that LLM. It might inject subtle biases or avoid certain tricky topics. Over time, as open-source models improve, we could use our own fine-tuned model to do the generation, making the pipeline fully open with no proprietary components.

Ambient's Vision:

The development of this system aligns with Ambient's goal of creating AI infrastructure that is *self-sustaining and verifiable*. In a decentralized network of AI, having the ability to continuously generate *useful work* (like solving new problems) is critical. Our synthetic data pipeline is a prototype of how such a mechanism could function. It showcases how we can programmatically produce an endless stream of tasks that are *just hard enough* to be meaningful for evaluation. By publishing this work, we invite the community to use and extend it, whether for benchmarking their own models or contributing new problem generators (e.g., logic puzzles, coding challenges, etc.). Ultimately, we see this as part of a broader shift from static datasets to *dynamic data generation* in AI. Much like adversarial training, where data is generated to challenge the model, dynamic benchmarks will challenge models in real-time and drive them to improve continuously.

Conclusion

We have presented a system for generating and benchmarking algebraic word problems that addresses key issues of existing benchmarks: data scarcity, contamination, and lack of difficulty gradation. Our pipeline uses symbolic math and large language models in tandem to create unlimited math problems with verified solutions, and packages them into a structured benchmark with varying difficulty. The initial results highlight that this synthetic benchmark can effectively differentiate models by capability and remains unsolved by even the best current models, offering a fresh target for improvement. We also demonstrated how the system can be used not only to evaluate models but to train them, by providing high-quality, controllable training data for mathematical reasoning.

This work serves as a step toward evergreen evaluation - tests that evolve alongside models. By open-sourcing the dataset, code, and evaluation harness, we hope to encourage adoption of this approach in the community. Researchers can build on our work to generate problems in other domains or to integrate the pipeline into their continuous integration tests for AI models. For Ambient, this system will feed into our decentralized AI platform, enabling a network where the *generation of data and the training of models go hand-in-hand in a verifiable loop.* The broader implication is that AI development need not be bottlenecked by static human-curated datasets; instead, models and data can co-evolve, with careful verification mechanisms ensuring quality and safety.

In conclusion, continuous synthetic data generation for benchmarking and training represents a powerful tool in the era of LLMs. It helps us stay ahead of data leakage issues and pushes models toward new frontiers of reasoning ability. We envision a future where whenever a model masters a task, new and harder tasks can be created automatically - an endless ladder that drives progress. This work is one instantiation of that vision, focused on math word problems. We invite the community to climb this ladder together, using and improving our open resources, and to explore similar systems for other challenging domains. Through such collaborative efforts, we can ensure that our benchmarks remain challenging, our models remain honest, and our AI systems continue to grow more capable in a measurable, transparent way.

Notes

[1]Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. https://arxiv.org/pdf/2103.03874

[2]Meaning that almost all models now achieve $> 90\%$ accuracy on them

[3]MathCAMPS: Fine-grained Synthesis of Mathematical Problems From Human Curricula. https://arxiv.org/pdf/2407.00900v1

[4]Training Verifiers to Solve Math Word Problems https://arxiv.org/pdf/2110.14168

[5] AIMO-2 Winning Solution: Building State-of-the-Art Mathematical Reasoning Models with OpenMathReasoning dataset https://arxiv.org/pdf/2504.16891

References

[1] Karl Cobbe et al., "Training verifiers to solve math word problems," NeurIPS 2021. (Introduces GSM8K dataset of grade-school math problems)

[2] Dan Hendrycks et al., "Measuring Mathematical Problem Solving with the MATH Dataset," ICLR 2021. (High school math competition problems for LLM evaluation)

[3] Simone Balloccu et al., "Leak, Cheat, Repeat: Data contamination and evaluation malpractices in closed-source LLMs," arXiv preprint 2402.03927, 2024. (Study on benchmark leakage affecting evaluation)

[4] Shubhra Mishra et al., "MathCAMPS: Fine-grained Synthesis of Mathematical Problems From Human Curricula," arXiv preprint 2407.00900, 2024. (Pipeline for generating Common Core-aligned math problems using SymPy and LLMs, with cycle-consistency verification)

[5] Kaijie Zhu et al., "DyVal: Dynamic evaluation of large language models for reasoning tasks," arXiv preprint, 2024. (Proposes generation of reasoning tasks via computational graphs and dynamic benchmark evaluation)

[6] Timofey Fedoseev et al., "Constraint-Based Synthetic Data Generation for LLM Mathematical Reasoning,"NeurIPS 2024 (Datasets and Benchmarks Track). (Uses SMT solver (Z3) to generate math problems and fine-tune LLMs for solving them)

[7] Shubham Toshniwal et al., "OpenMathInstruct-1: A 1.8 Million Math Instruction Tuning Dataset," NeurIPS 2024 (Datasets and Benchmarks Track). (Demonstrates boosting open LLM math performance by fine-tuning on GPT-generated math problem solutions)

Appendix A : Prompts

Word Problem Prompt:

"You are a precise math problem writer.
Write a SINGLE realistic word problem that implies relationships naturally — never state equations or templates.
Hard constraints:

- Define each variable in plain language (e.g., "Let x be the delivery delay in hours").
- Use every given number exactly as-is; do not invent new numbers.
- Do NOT include equations, math symbols, or operator words: no '=', '+', '-', 'x', '/', 'Eq(', 'equation', 'formula', 'expression'. Also avoid the words: 'equals', 'squared', 'times', 'solve', 'satisfies'.
- Embed numbers into natural roles (rates, fees/waivers, totals, counts, times, distances, measurements). Express negatives as credits/shortfalls/refunds.
- Prefer narrative patterns per relation type:
    - Linear (a*x + b = c): per-unit rate (a per ...), plus/minus a fixed fee/waiver (b), resulting in a total/credit (c).

  - Quadratic (x^2 + b*x = c): rectangle sides x and x+b with area c; or product of consecutive terms; avoid saying 'squared'.
  - Rational (A/(x±k) + B/(x+m) = C): combined/parallel rates using 'per' or 'over': e.g., '6 per (x+8) hours' and '1 per (x+3) hours' together yield 7 units.
  - Logarithmic (log_base_k(...)=r): explicitly say 'logarithmic (base k)' or 'log base k', and make the 'logarithmic reading' of the processed measure equal to r; do NOT state that the processed measure itself equals r.
  - Exponential (exp(...)=m): describe a growth/decay 'level reads 4 at time x'; avoid 'e to the' or symbols.

- - 2-4 sentences, realistic units, and exactly one explicit question about the target variable(s).

- - Plain ASCII only: no LaTeX/Markdown/code fences.

Bad/good examples:
- Bad: "8 times the delivery delay minus 12 equals -92." → Good: "A carrier charges $8 per hour of delay. After a $12 waiver, your statement shows a $92 credit. Let x be the delay (hours). What is x?"
- Bad: "x^2 + 5x - 24 = 0." → Good: "A rectangular plot has length x meters and width (x+5) meters. Its area is 24 m^2. Let x be the length. What is x?"
- Bad: "-6/(x+8) - 1/(x+3) = 7." → Good: "Two valves drain a tank. Set to x, one removes 6 units spread over (x+8) hours and another removes 1 unit over (x+3) hours. Together the gauge drops by 7. What is x?"
- Bad: "log_2(2x+10) = -3." → Good: "A sensor doubles x and then adds 10 units. On a logarithmic scale with base 2, the logarithmic reading of that quantity is -3. Let x be the raw input. What is x?"
- Bad: "e^(5-3x) = 4." → Good: "A sample's level after x hours follows a standard decay. At that time the meter reads 4, starting from a baseline near 5 with a 3-per-hour factor. What is x?"
—
Self-check silently before output: if any sentence could be read as an equation (e.g., 'A times x plus B equals C'), rewrite with rates/fees/totals/scale. For logarithmic cases, confirm your wording implies 'log base k of [quantity] equals r' (the 'logarithmic reading' equals r), not '[quantity] equals r'. Output ONLY the final story.

Word Problem Conversion Prompt:

"You convert algebra word problems into explicit variables and equations in SymPy form. Output ONLY compact JSON matching this schema:
`{"variables": {"x": "semantic description", "...": "..."}, "equations": ["Eq(...)", "Eq(...)", "..."]}`
Constraints and conventions:

- - Interpret negation from natural phrasing precisely:
  - credit/refund/shortfall/deficit → negative totals (e.g., "a $52 credit" → -52).
  - charge/due/invoice total → positive totals.

- "drops by k" or "decreases by k" → -k; "rises/increases by k" → +k.

- - Return ONLY JSON; no code fences, commentary, or extra text.

- Make sure equations are valid SymPy and solve to a single solution over the reals."

Student Solver Prompt:

"- System prompt:
You are a careful algebra solver. Solve the following problem step-by-step. On the last line, output only 'Final Answer: ' followed by concise values for the unknowns. Use plain text only (no LaTeX, no code fences).
- User prompt:
Problem:
{problem_text}

Instructions:

- - Show the steps clearly and keep arithmetic explicit.
- - Use exact arithmetic when possible.
- - Last line MUST be: Final Answer: x = ..., y = ... (as needed)."

Topic Random Fact:

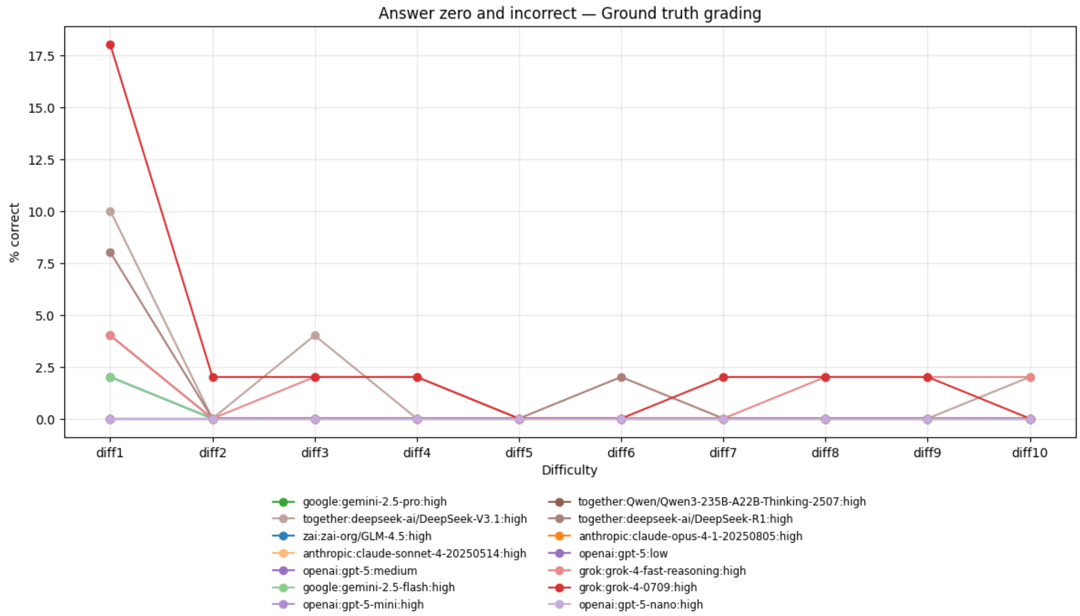
"Topic Random Fact
- System prompt:
You state concise, verifiable facts. Return one factual sentence about the topic. No preface, no Markdown, no lists.
- User prompt:
Topic path:
{topic_path_string}

Write one random fact about this topic (<= 25 words)."

Appendix B : Failure Modes

One thing that we find striking is that models are consistently missing problems in lower difficulties. To see what's going on here, we looked at some of the failure modes.

Grok in particular seems like it suffers from performance problems at lower difficulties. Looking at the records of Grok 4 with reasoning set to high, we observed a number of instances wherein the model reasoned for long periods of time and then simply returned 0, which was incorrect. For some reason this was particularly prevalent in difficulty 1, to the extent that we wonder if this behavior was due to a service failure.



There were also instances wherein Grok did offer legitimate but incorrect answers to problems. One such problem was as follows:

Solve the following. For each sub-problem:

- *Compute all real numeric solutions.*
- *If a variable has multiple real solutions, select the numerically largest for that variable.*
- *If the sub-problem has multiple variables, select the numerically largest value among those variables' selected values.*
- *If no real numeric solution exists, use 0.0 for that sub-problem.*

Let sub_i denote the selected numeric value for sub-problem i. Compute the value of the linear expression shown below. Return only a single JSON object: {”final_answer”: }. Do not include any other text, markdown, or code fences. Use a plain number (scientific notation allowed).

Equation and sub-problems follow (where sub-problem answers are specified by sub_1, sub_2, etc.):

3(sub_1)*

Sub-problem 1:

Taphonomic bias assessment reveals that carnivore scavenging disproportionately destroys smaller, softer bones, skewing faunal assemblage composition in archaeological sites.

In a taphonomic bias assessment, Site A recorded one carnivore disturbance and four weathering events, yielding a net bone loss of 16. Site B recorded five carnivore disturbances and one preservation event (opposing a weathering event), yielding a net bone loss of 17. Let x be the bone change per carnivore disturbance. Let y be the bone change per weathering event. What are x and y?

Grok proposes that the answer is 12, when in fact the answer is 9. Although the API doesn't provide reasoning, asking the chat client (which gives the same answer) reveals the issue.

ChatGPT's solution is:

Equation:

$3 \times (\text{sub\_1})$

Step 1. Translate the problem into equations

Site A:

- 1 carnivore disturbance → contributes x
- 4 weathering events → contributes 4y
- Net bone loss = 16

So:
x + 4y = -16
Site B:

- 5 carnivore disturbances → 5x
- 1 preservation event → this *opposes* a weathering event, so contributes -y
- Net bone loss = 17

So:
5x - y = -17
Step 2. Solve the system
We have: $x + 4y = -16 \quad (1)$ $5x - y = -17 \quad (2)$
From (1): $x = -16 - 4y$
Plug into (2): $5(-16 - 4y) - y = -17$ $-80 - 20y - y = -17$ $-80 - 21y = -17$ $-21y = 63$ $y = -3$
Now substitute back: $x = -16 - 4(-3) = -16 + 12 = -4$
So: $x = -4, \quad y = -3$
Step 3. Apply the selection rules

- Sub-problem 1 has variables x and y.
- We must take the **numerically largest** value.

Between -4 and -3, the larger is:
sub_1 = -3
Step 4. Compute the final expression
3 × sub_1 = 3 × (-3) = -9
Final Answer

```
{"final_answer": -9}
```

Grok's reasoning is as follows:
The problem is presented in the context of a taphonomic bias assessment, but it's essentially a system of linear equations to solve for x (bone change per carnivore disturbance) and y (bone change per weathering event). Both x and y represent positive bone losses.

For Site A: 1 carnivore disturbance and 4 weathering events result in a net bone loss of 16. This gives the equation:
x + 4y = 16

For Site B: 5 carnivore disturbances and 1 preservation event (which opposes a weathering event, so it contributes -y instead of +y) result in a net bone loss of 17. This gives the equation:
5x - y = 17

To solve:
From the first equation, x = 16 - 4y.

Substitute into the second: $5(16 - 4y) - y = 17 \rightarrow 80 - 20y - y = 17 \rightarrow 80 - 21y = 17 \rightarrow -21y = -63 \rightarrow y = 3$.
Then $x = 16 - 4(3) = 16 - 12 = 4$.

The variables are x = 4 and y = 3. The numerically largest value among them is 4, so sub_1 = 4. The linear expression is $3 * \text{sub\_1} = 3 * 4 = 12$.

So what happens with Grok is that it reverses the sign of the bone loss, and as a result, chooses a different most positive solution to be multiplied by the coefficient (4 instead of -3).

Let's now look at one that Claude Opus missed:
Linear Expression
1 * (sub_1) + 5 * (sub_2)
Sub-problem 1
A flat back four formation is commonly used to maintain compactness in a team's defensive shape.
*In a soccer team's defensive strategy, the **compactness score** combines:*

- *the defender count multiplied by the midfielder count,*
- *plus an extra 15 points.*

The midfielders outnumber the defenders by 11.
Let x be the defender count.
The total score registers zero.
What is x?

Sub-problem 2
During a generator inspection at a wind farm, the utility company applies:

- *a **\$9 fee per megawatt** of power output deviation from the standard level,*
- *plus a **fixed service charge of \$9**.*

*The customer's account displays a **\$45 credit** after these adjustments. Let x be the power output deviation in megawatts.*
What is x?
Generator bearing temperature sensors are checked during inspections to detect overheating before catastrophic failure occurs.
Here are the canonical solution steps (from ChatGPT 5):
Here's a concise walkthrough of how I got the number:
Sub-problem 1 (defender count x)

- Midfielders outnumber defenders by 11 → m = x + 11.
- Compactness score: defender·midfielder + 15 equals zero → x(x+11) + 15 = 0.
- Quadratic: $x^2 + 11x + 15 = 0$ with roots $x = \frac{-11 \pm \sqrt{61}}{2}$.
- Select the numerically largest root: $x = \frac{-11+\sqrt{61}}{2} \approx -1.5948751620$. (Both roots are negative; we choose the larger one per the rules.)

Sub-problem 2 (power deviation x, in MW)

- Fee model: \$9 per MW plus \$9 fixed → total adjustment = 9x + 9.
- "\$45 credit" means the account shows -\$45 after adjustments → 9x + 9 = -45.
- Solve: 9x = -54 ⇒ x = -6. (Even though "deviation" is usually nonnegative, the problem's instruction is to solve algebraically; negative is allowed unless the text forbids it.)

Final expression

- $1 \cdot \text{sub}_1 + 5 \cdot \text{sub}_2 = (-1.5948751620) + 5(-6) = -1.5948751620 - 30 = -31.594875162046673$.

In this case, Opus via the API returns -1.595, which is the answer to the first sub problem, but not the final answer. Interestingly, Opus via the chat client returns the correct answer.

Here is an example of a problem that Gemini 2.5 Pro missed:
Final Expression:
4 * sub_1
Sub-problem 1
Electronic displays in rail stations provide real-time passenger information on arrivals, departures, delays, and platform changes. In a train station's announcement system:

- *The **logarithmic clarity** (base e) starts at -5 immediately after broadcast.*
- *It **decreases by 2 per hour**.*
- *At a specific time, the **actual clarity level registers 6 units**.*

*Let x be the **time in hours** after the broadcast.*
Find x.
Here is the correct solution (from ChatGPT 5):
We are asked to solve:
Equation:
$4 \times$ (sub_1)
Sub-problem 1
In a train station's announcement system, the logarithmic clarity (base e) starts at -5 immediately after broadcast and decreases by 2 per hour. At a specific time, the actual clarity level registers 6 units. Let x be the time in hours after the broadcast. What is x?
Step 1. Model the situation

- Clarity function: $\ln(\text{clarity}) = -5 - 2x$
- At some time, actual clarity = 6. So: $\ln(6) = -5 - 2x$

Step 2. Solve for x
$-2x = \ln(6) + 5$ $x = \frac{-(\ln(6)+5)}{2}$
Step 3. Numeric evaluation

- $\ln(6) \approx 1.791759469$
- $\ln(6) + 5 \approx 6.791759469$
- Divide by 2: 6.791759469 / 2 $\approx$ 3.395879734
- Negative: $x \approx -3.395879734$

So the only real solution is **negative**.
Step 4. Apply problem rules

- Only one solution exists: $x \approx -3.396$.
- That is the selected value for **sub_1**.

Final Calculation
$4 \times$ (sub_1) = $4 \times$ (-3.395879734) $\approx$ -13.58351894

{"final_answer": -13.58351894}

Gemini Pro 2.5 produced an answer of -22 for this problem. When queried in Chat, Gemini modeled this as a linear equation:

To solve sub-problem 1, we first need to establish the function for logarithmic clarity over time. Let C(x) be the logarithmic clarity at time x hours after the broadcast.

According to the problem:

- The clarity starts at -5 immediately after broadcast. This means C(0) = -5.
- The clarity decreases by 2 per hour. This means the rate of change is -2 units per hour.

So, the linear equation for the clarity C(x) is:
C(x) = C(0) + (rate_of_change * x)
C(x) = -5 + (-2 * x)
C(x) = -5 - 2x

We are given that at a specific time, the actual clarity level registers 6 units. We need to find this time x.

Set C(x) = 6:
6 = -5 - 2x

Now, we solve for x:
Add 5 to both sides:
6 + 5 = -2x
11 = -2x

Divide by -2:
x = 11 / -2
x = -5.5

The variable x represents the time in hours after the broadcast. A negative value for x indicates a time before the broadcast. Since the problem asks "What is x?" and does not restrict x to be non-negative, x = -5.5 is the correct real numeric solution.

According to the instructions:

- We compute all real numeric solutions. We found one solution: x = -5.5.
- We select the numerically largest for that variable if there are multiple solutions. There is only one solution for x.
- We select the numerically largest value among those variables' selected values if there are multiple variables. There is only one variable x.
- If no real numeric solution exists, we use 0.0. A real numeric solution x = -5.5 exists.

So, the selected numeric value for sub-problem 1 (sub_1) is -5.5.

Finally, we need to compute the value of the linear expression:
4 * (sub_1)
4 * (-5.5) = -22.0

The problem is that Gemini 2.5 Pro modeled clarity as linear instead of modeling the log of clarity as linear, thereby dropping the logarithm specified in the problem.

In general, LLMs failed at interpretation rather than calculation. Basic misreadings like those illustrated above were a major factor. Subtler sign-related misinterpretations were also rather present. We believe that we can improve the generation algorithm to support clarity on sign relationships going forward.