BARSA: An Adaptive Test-Time Scaling Strategy for Mathematical Reasoning under Global Compute Budgets

Anonymous Authors[1]

Abstract

This paper presents our submission to the AI Mathematical Olympiad - Progress Prize 3 (AIMO 3) competition (Frieder et al., 2025). We propose **Budget-Aware Recursive Self-Aggregation (BARSA)**, an adaptive test-time scaling framework for mathematical reasoning under a global inference budget. BARSA extends Recursive Self-Aggregation (Venkatraman et al., 2026) by using answer-distribution statistics and runtime estimates to decide whether to accept the current answer or continue with another aggregation round. We evaluate BARSA on the AIMO 3 leaderboards and a curated benchmark of AI-hard problems. Our analysis shows that recursive aggregation is most effective when the correct answer appears as a minority candidate or when the answer distribution is unstable, but remains vulnerable to deceptive wrong majorities. BARSA improves the mean accuracy and reduces score variance when evaluated under a fixed time budget. Across public leaderboard submissions, BARSA achieved a mean score of 40.38 with standard deviation 0.744, compared with 36.70 ± 1.567 for Majority@8 with a dynamic scheduler and 37.91 ± 1.676 for RSA with the same dynamic scheduler. This corresponds to improvements of $+3.68$ points over the strongest majority-voting baseline and $+2.47$ points over non-budget-aware RSA, while reducing score standard deviation by more than half.

1. Introduction

Large Language Models have achieved strong performance on mathematical reasoning tasks, especially when their inference-time compute is increased through test-time scaling techniques such as self-consistency (Wang et al., 2023), best-of-N sampling (Huang et al., 2025), Monte-Carlo Tree Search (Xie et al., 2024), and sequential self refinement (Madaan et al., 2023).

Most test-time scaling methods, however, are studied in a single-problem setting, where the goal is to maximize accuracy under a fixed local compute budget. This abstraction is useful, but it misses an important constraint in many practical settings. In realistic deployments, a solver often receives a batch of tasks under a shared compute or wall-clock budget. In this setting, the solver must maximize global accuracy, and this depends on ensuring that every task has enough time to complete under the adopted test-time scaling strategy. This motivates the central question of this paper:

How should a mathematical reasoning system allocate test-time compute across a sequence of problems under a global inference budget?

This question arises directly in the AI Mathematical Olympiad - Progress Prize 3 (AIMO 3) (Frieder et al., 2025) In AIMO 3, participants were required to solve 50 difficult mathematical problems at the level of the International Mathematical Olympiad within 5 hours on a single H100 GPU. Problems are served one at a time, and must be answered before moving on to the next one. There is no opportunity to revisit an earlier problem. These problems are not publicly available, which reduces the risk of data leakage and provides a more realistic test of mathematical reasoning ability.

A naive strategy is to allocate a fixed time budget to every problem. This is simple but inefficient. From empirical observations, easy problems may receive more time than necessary, while hard problems may be cut off too early. Another strategy is to continue sampling whenever the current answer distribution is uncertain. This improves accuracy on hard tasks, but risks exhausting the global budget early when the order of the problems by difficulty is unknown. In both cases, the inference method and time scheduler are treated as separate components. We argue that, under a global budget, they should instead be designed jointly.

Olympiad-style mathematical problems are well suited for studying adaptive test-time scaling because they vary widely

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

in difficulty. They also expose a key weakness of simple self-consistency. From our experiment results, the most frequent answer is not always the correct one. On difficult problems, the correct answer may appear only as a minority candidate, or may be absent from the initial sample altogether (see section 4.2). Conversely, some problems (see appendix B.1) produce a strong but wrong majority, creating a deceptive signal that makes early stopping unsafe.

To address these challenges, we propose Budget-Aware Recursive Self-Aggregation (BARSA), an adaptive inference framework that combines recursive solution aggregation with compute-aware scheduling. BARSA builds on Recursive Self-Aggregation(Venkatraman et al., 2026), where later solution attempts are conditioned on earlier attempts and can therefore synthesize partial progress across multiple reasoning traces. Our contribution is to make this process budget-aware. After each round, the system decides whether to stop or continue by observing the current answer distribution and the estimated cost of another round. In addition, BARSA reserves a small number of fresh attempts in later rounds to preserve diversity and reduce the risk that all future generations collapse onto the same incorrect reasoning path.

We evaluate this approach under fixed-budget mathematical reasoning settings, including the AIMO 3 competition environment and a curated diagnostic benchmark of AI-hard olympiad-style problems. Beyond aggregate accuracy, we analyze the structure of answer distributions produced by the model and identify several recurring failure modes. This analysis shows that recursive aggregation is especially useful when the model is uncertain or when the correct answer appears in a minority class, but that it remains vulnerable to problems where the model converges quickly and confidently to a wrong answer.

Our main contributions are:

1. We introduce Budget-Aware Recursive Self-Aggregation, an adaptive inference method that combines recursive aggregation, fresh attempts, consensus-based stopping, progress-based continuation, and global budget scheduling.

2. We provide a diagnostic analysis of answer-distribution failure modes in olympiad-style reasoning, highlighting when aggregation improves over majority voting and when it fails.

3. We evaluate BARSA under fixed-budget settings and show that it improves the accuracy-stability tradeoff: across public leaderboard submissions, BARSA obtained 40.38 ± 0.744, compared with 36.70 ± 1.567 for Majority@8 with a dynamic scheduler and 37.91 ± 1.676 for non-budget-aware RSA.

Overall, our results suggest that the effectiveness of test-time scaling depends not only on how many samples are generated, but also on when and where additional samples are spent. For mathematical reasoning systems operating under realistic compute constraints, inference should be treated as an adaptive allocation problem rather than a fixed sampling procedure.

2. Related Work

2.1. Tree-Based Approaches

Tree-based approaches treat reasoning as a search problem over intermediate states rather than as a single beginning-to-end generation. Tree of Thoughts (Yao et al., 2023) generalizes chain-of-thought prompting by sampling multiple intermediate "thoughts", evaluating them, and selectively expanding promising branches. Monte Carlo Tree Search methods (Xie et al., 2024; Zhou et al., 2024) similarly maintain a search tree over partial reasoning trajectories and use value estimates to guide exploration.

These methods are attractive because they allow backtracking and deliberate exploration of multiple reasoning paths. However, they also introduce a structural challenge. The reasoning process must be decomposed into meaningful intermediate states that can be evaluated and expanded. For olympiad-style mathematics, such branch points are often hard to define reliably. In addition, tree-based methods usually require repeated generation and model calls at many intermediate states, so the search space can become expensive to explore on difficult problems.

2.2. Generation-Based Approaches

Generation based approaches generate entire solutions at once, multiple times. One common example is self-consistency (Wang et al., 2023), which samples multiple reasoning paths in parallel, and selects the most frequent final answer. Best-of-N sampling (Huang et al., 2025) also begins by generating multiple candidates in parallel, but is usually paired with a scoring or verification mechanism, followed by selecting the best solution.

Self-consistency and Best-of-N have a sample complexity of $\frac{1}{\Delta^2}$ and $\frac{1}{\Delta}$ respectively (Huang et al., 2026), where Δ is the difference in probability of producing the correct answer and the probability of the second most likely answer. Self-consistency also requires that Δ is positive, while Best-of-N depends on having a good scoring function. Designing such a scoring function is itself non-trivial. Moreover, neither of them can recover the correct solution when it is absent altogether from the initial pool of N candidates.

Another method is sequential self-refinement (Madaan et al., 2023). In this framework, the model generates an initial

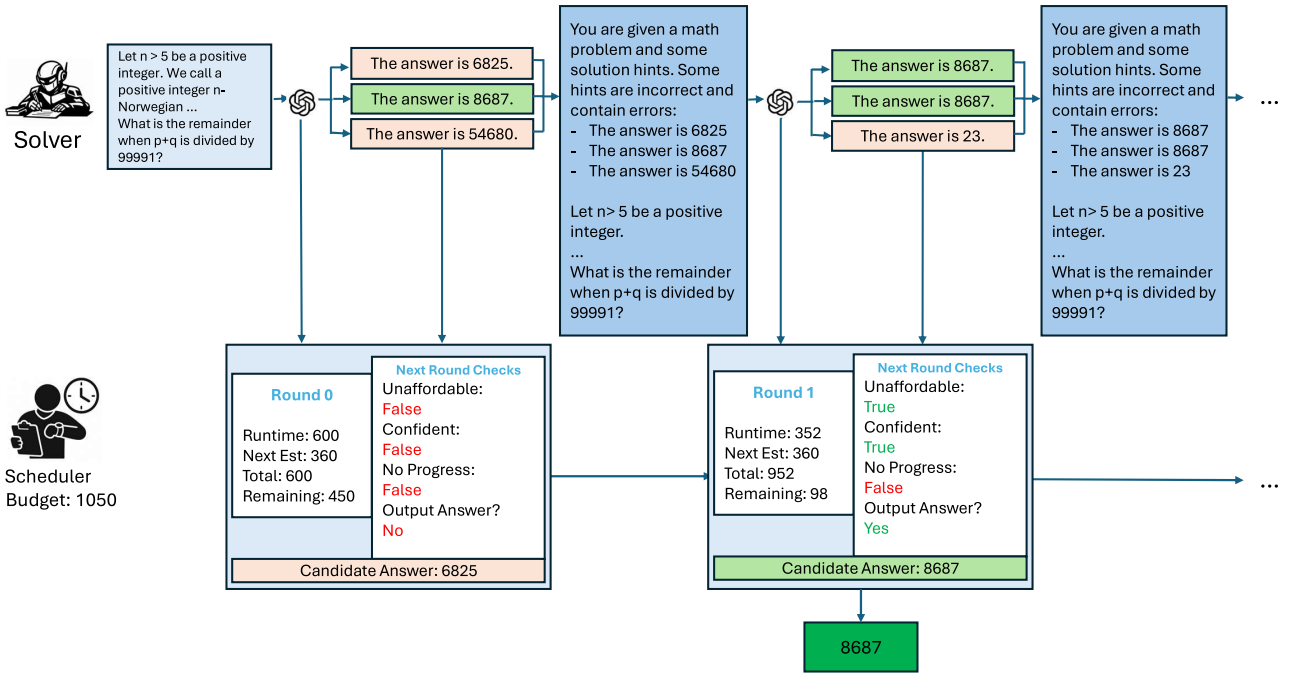

Figure 1. Example run of BARSA. The total time budget is 1050 seconds. Round 0 produces a diffuse answer distribution in 600 seconds, in which the correct answer is not the majority. Based on this, the scheduler estimates that each future round of RSA costs 360 seconds. It then checks for three conditions: **(i)** Is the next round unaffordable? **(ii)** Does the distribution indicate the answer is confident? **(iii)** Is RSA not making progress? If any of these is `TRUE`, return the answer. Otherwise, continue to the next round.

answer, critiques it, and iteratively revises the response using its own feedback. (Chen et al., 2024) adopts this for code generation tasks using external feedback from the executor.

Recursive Self-Aggregation (Venkatraman et al., 2026) combines the parallel and sequential frameworks. It maintains a population of candidate solutions, then uses earlier candidates as hints for later rounds of generation. This preserves some of the parallelism of self-consistency while allowing later attempts to synthesize partial progress from earlier traces.

2.3. Scoring-Based Approaches

Best-of-N methods are usually used with a scoring mechanism. One such mechanism is verifier-based Best-of-N, where a model first samples N candidate solutions and then uses a learned verifier or reward model (as opposed to internal metrics) to select the highest-scoring one (Cobbe et al., 2021). Scores can also be assigned to intermediate reasoning steps. Process reward models (Lightman et al., 2023) do this using a separate trained verifier. rStar-Math (Guan et al., 2025) combines Monte-Carlo Tree Search with a Process Reward Model, using step-level evaluation to guide the search over mathematical reasoning trajectories.

Scores can also be assigned via internal metrics produced by the inference engine during the generation process. Chen et al. (2026) identifies certain tokens in a sequence as "deep-thinking tokens" by measuring how early in a model the token's underlying distribution stabilises. The proportion of deep thinking tokens in a generated solution is then used to calculate the score. Other studies have also found relationships between task accuracy and token count in sequential scaling (Gema et al., 2025) and entropy (Agarwal et al., 2025).

However, scoring-based scaling depends strongly on score quality. Recent work studies this issue theoretically and shows that the performance of Best-of-N and rejection sampling is governed by the verifier's ability to distinguish correct from incorrect candidates (Dorner et al., 2026). This is a serious limitation in olympiad-style mathematics, where incorrect solutions can be fluent, concise, and superficially convincing. V_1-Infer (Singh et al., 2026) explicitly addresses this problem and uses tournament-style pairwise comparisons instead. They argue that models are better at making relative judgements as opposed to pointwise absolute judgements of solution quality. Our system further improves on this by allowing the model access to multiple solutions in each aggregation round.

3. Methodology

Our method is designed for mathematical reasoning under a strict global time budget. The key constraint is that the solver receives a sequence of problems of unknown relative difficulty and must decide how much compute to spend on each one, while preserving enough time for the remaining problems. This changes the role of test-time scaling. Each additional unit of scaling is useful only when it improves the expected global score more than the time they consume.

We call our method **Budget-Aware Recursive Self-Aggregation (BARSA)**. BARSA combines two components: recursive solution aggregation and a scheduler that jointly decides whether the inference process should stop or continue. Additional priors such as the difficulty distribution of the problems can also be baked in, if known. The method is built around the idea that inference scaling and time allocation should not be treated as independent modules. The scheduler should understand the cost structure of the inference algorithm, and the inference algorithm should expose signals that help the scheduler decide whether further computation is worthwhile.

Algorithm 1 Budget-Aware RSA

Input: problem q, scheduler state S, max rounds $R_{\max}$
Output: final answer $\hat{a}$
$B \leftarrow \textsc{ComputeBudget}(S)$
$(C_0, H_0, \hat{a}, \widehat{T}) \leftarrow \textsc{InitialRound}(q)$
if $\textsc{Confident}(H_0)$ **then**
  return $\hat{a}$
end if
if $\neg\textsc{Affordable}(S, B, \widehat{T})$ **then**
  return $\hat{a}$
end if
for $r = 1$ to $R_{\max}$ **do**
  $(C_r, H_r, \hat{a}) \leftarrow \textsc{RSARound}(q, C_{r-1})$
  if $\textsc{Confident}(H_r)$ **then**
    return $\hat{a}$
  end if
  if $\neg\textsc{Affordable}(S, B, \widehat{T})$ **then**
    return $\hat{a}$
  end if
  if $\neg\textsc{Progress}(H_{r-1}, H_r)$ **then**
    return $\hat{a}$
  end if
  if $r = R_{\max}$ **then**
    return $\hat{a}$
  end if
end for
return $\hat{a}$

Here C_i denotes the set of solution candidates generated in round i, and H_i denotes the corresponding histogram of parsed final answers. The current answer $\hat{a}$ is the majority answer under the latest histogram. The estimate $\widehat{T}$ is the predicted cost of one additional RSA round, used by the scheduler to determine whether another round is affordable.

3.1. Recursive Self Aggregation

The core inference algorithm is Recursive Self-Aggregation (Venkatraman et al., 2026). An example is shown in *Figure 1* above. Given a problem, the model first generates a batch of independent solution attempts. We refer to this as Round 0. Each attempt is parsed into a final answer, and the answers are grouped into a histogram.

If the answer distribution is not convincing, the system launches another aggregation round. In this round, the model is shown the original problem together with the solution attempts from the previous round, presented as hints that may be incomplete or incorrect. The model is then asked to solve the problem again. This produces a new batch of candidate solutions, which are again parsed, aggregated, and evaluated. This process can continue for multiple rounds.

The answer histogram after each round is the main observable state of the RSA process. It provides three useful signals. First, the size of the largest answer class measures consensus. Second, the gap Δ between the largest and second-largest classes measures how decisive that consensus is. Third, the number of distinct answers measures how diffuse the distribution is. These statistics are cheap to calculate, and will be used to decide the time scheduler's behaviour later on, to be discussed next.

The pseudocode is given in Appendix A.

3.2. Time Scheduler and Problem Runtimes

The second component of BARSA is a time scheduler. The scheduler controls how much time may be spent on the current problem while preserving enough time for the remaining problems. Its role is twofold: it sets an upper bound on the time available to each problem, and it decides whether additional aggregation rounds are affordable under the global budget.

Time Allocations: At the beginning of each problem, the scheduler computes a provisional budget from the remaining wall-clock time. It first reserves time for the unsolved problems that will appear later in the sequence. This reserve prevents the system from spending too aggressively on early problems and then failing to complete later ones. The reserve policy can be chosen according to the deployment setting. For example, if the difficulty distribution is known beforehand, then the reserve policy can be made to follow the estimated number of easy, medium and hard problems remaining.

In deployment scenarios where the difficulty distribution is unknown, BARSA can also use a more adaptive policy through a time bank. The time bank tracks whether the run is ahead of or behind schedule. If earlier problems finish faster than expected, the bank becomes positive, allowing the scheduler to allocate more time to later problems. If earlier problems take longer than expected, the bank becomes negative, making later budgets tighter. Importantly, the time bank is not applied greedily. Time debts do not need to be repaid immediately, and time surpluses are not made fully available to the next problem, especially early in the run. Instead, the bank only partially adjusts the current budget. This makes the scheduler responsive to the actual runtime trajectory of the run, while avoiding unstable behavior caused by overreacting to a single unusually easy or unusually hard problem.

Round-Cost Estimation: The scheduler also estimates the cost of continuing RSA on the current problem. This estimate is needed because recursive aggregation is useful only if the next round can be completed safely. Starting an aggregation round that cannot finish is usually worse than stopping early, since the system may consume substantial time without obtaining enough completed answers to change the final decision. BARSA estimates this continuation cost from the observed runtime of Round 0. Empirically, we found that an aggregation round usually required about 60–70% of the Round 0 runtime. This estimate provides a conservative runtime proxy that is used to decide whether another round is affordable.

3.3. Joint Design of Inference and Scheduling

BARSA combines RSA and the time scheduler through a round-by-round continuation rule. After each round, the system decides whether to accept the current answer or launch another RSA round. This decision is based on three conditions: consensus, affordability, and progress.

BARSA stops and returns the current answer if any of the following evaluates to `TRUE`:

1. **Affordability**: The next round is unaffordable.
2. **Confidence**: The current answer distribution is confident enough.
3. **Progress**: RSA is not making progress. (In round 0, this is always `FALSE`).

Affordability: BARSA first checks whether another RSA round is affordable. This check uses the current problem budget and the round-cost estimate supplied by the scheduler. A new round is allowed only if it is likely to fit within both the per-problem budget and the remaining global wall-clock budget, including a safety margin. If the next round is unaffordable, RSA stops and outputs the current answer.

Confidence: Next, BARSA checks if the current answer histogram has a sufficiently strong top answer class and a sufficiently large Δ (gap between largest and second largest majority class). If these metrics are strong enough, the system accepts the majority answer and stops. This prevents the method from wasting time on problems that are likely already solved.

Progress: This is always `FALSE` in Round 0. For subsequent rounds, BARSA checks if the previous aggregation round made progress. This is measured using changes in the histogram across rounds. Progress is made when the **top answer class becomes larger** (convergence), when Δ **increases** (confidence), the number of **distinct answers decreases** (concentration), or **new distinct answers appear** (diversity).

4. Experiment Results and Analysis

4.1. Classification of AI-Hard Problems

During development, we observed that failures on the internal benchmark were not homogeneous. Instead, the answer histograms produced by repeated sampling tended to fall into a small number of recurring regimes. We used these regimes qualitatively to understand when RSA was likely to help and when early stopping was risky.

These classifications were made specifically based on answer distributions observed from the `gpt-oss-120b` model. For other models, the distributions may change.

Class 1: diffuse distributions. The answer distribution is highly spread out. Several answers have similar support, only one of them is correct. These problems are useful for testing whether RSA can synthesize partial progress from multiple imperfect attempts. The Norwegian integers problem `86e8e5` and `imo-bench-algebra-015` fall into this class.

Class 2: unstable two-answer competitions. The top two answer classes are close, often resembling a noisy 60–40 or 70–30 split. Majority answer after eight samples is unstable and may change across runs. RSA was often effective here because the correct answer was usually already present in the candidate pool, even when it was not the initial majority. Examples include `imo-bench-combinatorics-013`, `039`, `042`, `054`, `068`, and `imo-bench-number_theory-023`.

Class 3: correct minority with several wrong modes. The correct answer appears as a small minority, while two or three wrong answers account for most of the remaining probability mass. RSA had mixed performance in this setting. When the correct answer appeared in

Class	Answer-distribution pattern	Majority-vote behavior	Aggregation/Scheduler Behaviour	Examples
Class 1: diffuse	Many distinct answers with little or no repeated support. The correct answer may be absent from the initial sample pool.	Majority voting has no meaningful consensus signal and effectively selects one sampled answer at random.	**Aggregation:** Effective **Scheduler:** Effective	NORWEGIAN INTEGERS; algebra-015
Class 2: unstable two-answer competition	Two answer classes dominate, often with a noisy 60–40 to 70–30 split.	The majority answer after a small number of samples is unstable and may change across runs.	**Aggregation:** Effective **Scheduler:** Effective	combinatorics-013, 039, 042, 054, 068; number_theory-023
Class 3: correct minority with several wrong modes	The correct answer appears with small support, while several wrong answers occupy most of the probability mass.	Majority voting is likely to select one of the wrong modes.	**Aggregation:** Effective **Scheduler:** Somewhat Effective	combinatorics-014; algebra-056
Class 4: deceptive wrong majority	A single wrong answer dominates, while the correct answer appears with non-negligible but smaller probability.	Majority voting confidently selects the wrong answer.	**Aggregation:** Effective **Scheduler:** Less Effective	algebra-026, 052; combinatorics-057
Class 5: near-deterministic wrong majority	One wrong answer receives overwhelming support; the correct answer appears rarely or not at all.	Majority voting almost always returns the wrong answer with high apparent confidence.	**Aggregation:** Less Effective **Scheduler:** Less Effective	algebra-055, 100; combinatorics-031, 049

Table 1. Qualitative classification of AI-hard problems based on answer distributions observed from gpt-oss-120b. The classes are not intended as a model-independent taxonomy, but as a diagnostic tool for understanding when recursive aggregation helps and when histogram-based stopping is risky.

the initial batch, RSA could often amplify it; when it did not, recovery was substantially less reliable. Examples include imo-bench-combinatorics-014 and imo-bench-algebra-056.

Class 4: deceptive wrong majority. The correct answer appears with non-negligible probability, but a single wrong answer dominates the distribution. These cases are difficult because the histogram resembles that of an easy problem with a correct majority. Examples include imo-bench-algebra-026, 052, and imo-bench-combinatorics-057.

Class 5: near-deterministic wrong majority. The most challenging cases are those in which the correct answer appears only rarely, while one wrong answer receives overwhelming support. These problems expose a key limitation of both majority voting and histogram-based stopping rules: the system may terminate early with high confidence despite being wrong. Examples include imo-bench-algebra-055, 100, imo-bench-combinatorics-031, and 049.

4.2. Examples

We present two representative runs of our pipeline. These examples were taken from experiements run on a small but carefully constructed internal benchmark. More details on the benchmark are in appendix B.

4.2.1. IMO-BENCH-COMBINATORICS-068

Table 2. Answer-level trace for one RSA run on imo-bench-combinatorics-068. The correct answer is 93447. The first eight rows correspond to the initial independent attempts, and are ordered by response completion order. The final four rows correspond to the RSA aggregation round.

Attempt	Length	Tool Calls	Entropy	Answer
7	19947	5	0.871	71207
6	20188	11	0.934	93447
4	20796	7	0.897	93447
2	24726	13	0.820	71207
3	24172	24	0.888	71207
1	25188	12	0.847	71207
8	35687	18	0.859	93447
5	44807	20	0.888	71207
11	10256	4	0.863	93447
14	11532	6	0.882	93447
10	11656	2	0.943	93447
15	13256	8	0.885	93447

Problem statement.

> Sir Alex plays the following game on a row of 9 cells. Initially, all cells are empty. In each move, Sir Alex is allowed to perform exactly one of the following two operations:
>
> 1. Choose any number of the form 2^j, where j is a non-negative integer, and put it into an empty cell.
> 2. Choose two, not necessarily adjacent, cells with the same number in them; denote that number by 2^j. Replace the number in one of the cells with 2^{j+1} and erase the number in the other cell.
>
> During the game, Sir Alex encounters a mysterious genie that grants him a wish. However, the genie warns Sir Alex that he can only make a limited number of moves. At the end of the game, one cell contains the number 2^{40}, while the other cells are empty. Determine the maximum number of moves that Sir Alex could have made. Give your answer modulo 100000.

Table 2 shows a run on `imo-bench-combinatorics-068`. This is an example of a class 2 problem. The unconditioned Round 0 answer distribution is an unstable two-answer competition, where the top two answer classes are close in size. Majority voting after eight samples is unstable and often changes across runs. However, a single round of RSA is able to rescue the correct answer.

4.2.2. NORWEGIAN INTEGERS

Table 3. Answer-level trace for one RSA run on the Norwegian integers problem from the official AIMO 3 reference set. The correct answer is 8687. The first eight rows correspond to the initial independent attempts and are ordered by completion order. The remaining rows correspond to the RSA aggregation round.

Attempt	Length	Tool Calls	Entropy	Answer
2	39891	35	0.737	25238
4	42215	43	0.744	99991
6	50564	43	0.743	98449
1	60352	35	0.717	23
7	57860	51	0.782	5
8	55358	128	0.628	<NA>
3	62878	61	0.677	40205
5	68260	62	0.713	41754
12	16795	16	0.681	98449
9	28171	41	0.630	8687
14	36232	22	0.742	5
15	34367	42	0.698	8687
11	36730	41	0.693	8687
13	37193	104	0.568	8687

Problem statement. The Norwegian integers problem from the official AIMO 3 reference set is the following.

> Let $n \geq 6$ be a positive integer. We call a positive integer n-Norwegian if it has three distinct positive divisors whose sum is equal to n. Let $f(n)$ denote the smallest n-Norwegian positive integer. Let $M = 3^{2025!}$ and, for a non-negative integer c, define
>
> $$g(c) = \frac{1}{2025!} \left\lfloor \frac{2025! f(M+c)}{M} \right\rfloor.$$
>
> We can write
>
> $$g(0) + g(4M) + g(1848374) + g(10162574) + g(265710644) + g(44636594) = \frac{p}{q},$$
>
> where p and q are coprime positive integers. What is the remainder when $p + q$ is divided by 99991?

The example in Table 3 is the Norwegian integers problem from the official AIMO 3 reference set. This is an example of a class 1 problem. In the initial unconditioned attempts, the answer distribution is highly diffuse. None of the eight parsed answers is repeated, and the correct answer 8687 is absent from the initial candidate pool. Majority voting therefore has no meaningful consensus signal and effectively selects one of the sampled answers at random. After one round of recursive aggregation, however, RSA produces four responses with the correct answer and selects 8687. We hypothesize that this occurs because the initial solution traces, although individually incorrect, contain useful partial progress toward the correct solution. By conditioning later attempts on these traces, RSA can combine complementary intermediate observations and recover an answer that was not produced by any single initial attempt.

4.3. Leaderboard Submission Results

We evaluated the system through repeated public leaderboard submissions during the AIMO 3 competition. Having an effective time scheduler is of paramount importance, as problems are presented in random order and had to be answered sequentially. There is no opportunity to revisit earlier problems, even if there is time remaining at the end. A weak scheduler could easily starve hard problems of compute while spending too much time on easy ones, especially when several hard problems appeared early in the ordering. Table 1 summarizes the main inference and scheduling combinations explored during development.

Inference Strategies:

1. **Majority@8**: Generate 8 independent solutions and return the majority answer.
2. **RSA 8**: Uses the same initial batch size, but applies Recursive Self-Aggregation when the initial answer distribution is not sufficiently convincing. Aggregation continues until a round produces a clear majority, or until the time limit is reached.

Inference strategy	Scheduler	# Subs.	Mean	Sd.	Obs. min	Obs. max	Rep. min	Rep. max
Majority@8	Fixed	4	32.00	2.828	30	36	30	38
Majority@8	Dynamic	10	36.70	1.567	34	39	34	44
RSA 8	Dynamic	23	<u>37.91</u>	1.676	35	41	–	–
Cog-Well Prompt RSA 8	Dynamic	6	33.33	1.966	30	35	–	–
V_1-Infer	Dynamic	5	37.60	<u>0.894</u>	37	39	–	–
RSA 8	RSA-Aware Dynamic	8	**40.38**	**0.744**	40	42	–	–

Table 4. Public leaderboard results for different inference and scheduling strategies. Scores are out of 50. Observed statistics are computed from our own submissions. Reported ranges summarize scores reported in public discussion posts or notebooks when available. Best results (ours) are in **bold**, and second-best results are <u>underlined.</u>

3. **Cog-Well Prompt RSA 8**: Uses the same RSA mechanism, but with the system prompt proposed in Dang et al. (2026). This prompt made generations faster, but reduced the reliability of individual solutions. The intended tradeoff was that cheaper generations might allow more RSA rounds, but this did not happen in practice.

4. **V_1-Infer**: This is the inference method proposed in Singh et al. (2026) that uses pairwise comparisons to rank the solutions. Our version generated $N = 8$ solutions, and then took majority vote in the top 4 ranked candidates.

Schedulers:

1. **Fixed**: Allocates a fixed amount of time to each problem. This is implemented by dividing the total 5-hour budget into a sequence of 50 evenly spaced per-problem deadlines. This gives each problem a nominal time allocation, and later problems can only use extra time if earlier ones finish early.

2. **Dynamic**: Reserves a fixed amount of time for each remaining problem (less than the average time) and allows the current problem to use the remaining slack. This lets hard problems borrow time from the global budget, while easier problems are expected to return surplus time by finishing early.

3. **RSA-Aware Dynamic**: Used in the final BARSA system. It extends the dynamic scheduler by explicitly accounting for the cost of continuing RSA. In particular, it uses the runtime of the initial batch to estimate whether another aggregation round is likely to fit safely within the remaining per-problem and global budgets.

Our final submission used the full BARSA pipeline: `gpt-oss-120b` (with high reasoning) as the base model, Recursive Self-Aggregation as the main inference method, and RSA aware scheduler-based time allocation. Across public leaderboard submissions, this family of systems achieved scores in the range of 40–42. For the final private leaderboard evaluation, the submission was run twice with different random problem orderings, and the reported score was the average of the two runs. **The final private score was 41.5**, which lies within the observed public leaderboard range.

5. Conclusion

In this work, we introduced Budget-Aware Recursive Self-Aggregation, or BARSA, an adaptive test-time scaling strategy for mathematical reasoning under a global compute budget. The main motivation behind BARSA is that, in realistic deployment settings, inference cannot be treated as a collection of independent single-problem decisions. A solver must decide not only how much reasoning to apply to the current problem, but also how much compute to preserve for the problems that remain. BARSA addresses this by coupling recursive solution aggregation with a scheduler that uses answer-distribution statistics, runtime estimates, and round-to-round progress signals to decide whether further aggregation is worthwhile.

Our leaderboard results suggest that this joint design improves both average performance and reliability. Compared with the majority-voting baselines used during early development, the final BARSA system achieved a higher average public leaderboard score while also substantially reducing run-to-run variance.

6. Future Work

The main weakness of BARSA is the fast wrong-majority regime where the system converges quickly and confidently to a wrong answer. On the internal benchmark, these were class 4 and class 5 problems where the answer histogram looks deceptively clean. For some problems the runtime is also short, so neither answer histograms nor timing heuristics reliably identifies them as hard cases. This creates a fundamental tension in the scheduler. Aggressive early stopping saves time on easy problems, but also risks missing these failures. A promising direction for future work is to develop stronger continuation rules that use richer signals than answer histograms and runtime alone.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning for Mathematics. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agarwal, S., Zhang, Z., Yuan, L., Han, J., and Peng, H. The unreasonable effectiveness of entropy minimization in llm reasoning, 2025. URL https://arxiv.org/abs/2505.15134.

Chen, W.-L., Peng, L., Tan, T., Zhao, C., Chen, B. J., Lin, Z., Go, A., and Meng, Y. Think deep, not just long: Measuring LLM reasoning effort via deep-thinking tokens, 2026. URL https://arxiv.org/abs/2602.13517.

Chen, X., Lin, M., Schärli, N., and Zhou, D. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KuPixIqPiq.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Dang, X., Agarwal, R., Porto, R., Goyal, A., Fowl, L. H., and Arora, S. Escaping the cognitive well: Efficient competition math with off-the-shelf models, 2026. URL https://arxiv.org/abs/2602.16793.

Dorner, F. E., Chen, Y., Cruz, A. F., and Yang, F. ROC-n-reroll: How verifier imperfection affects test-time scaling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3Gy5mmyuxn.

Frieder, S., Bealing, S., Vonderlind, P., Li, S., Nikolaiev, A., Smith, G. C., Buzzard, K., Gowers, T., Liu, P. J., Loh, P.-S., Mackey, L., de Moura, L., Roberts, D., Sculley, D., Tao, T., Balduzzi, D., Coyle, S., Gerko, A., Holbrook, R., Howard, A., and Markets, X. AI Mathematical Olympiad - Progress Prize 3. https://kaggle.com/competitions/ai-mathematical-olympiad-progress-prize-3, 2025. Kaggle.

Gema, A. P., Hägele, A., Chen, R., Arditi, A., Goldman-Wetzler, J., Fraser-Taliente, K., Sleight, H., Petrini, L., Michael, J., Alex, B., Minervini, P., Chen, Y., Benton, J., and Perez, E. Inverse scaling in test-time compute. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=NXgyHW1c7M. Featured Certification, J2C Certification.

Guan, X., Zhang, L. L., Liu, Y., Shang, N., Sun, Y., Zhu, Y., Yang, F., and Yang, M. rStar-Math: Small LLMs can master math reasoning with self-evolved deep thinking, 2025. URL https://arxiv.org/abs/2501.04519.

Huang, A., Block, A., Liu, Q., Jiang, N., Krishnamurthy, A., and Foster, D. J. Is best-of-N the best of them? coverage, scaling, and optimality in inference-time alignment. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=QnjfkhrbYK.

Huang, B., Li, S., Wu, T., Yang, Y., Talwalkar, A., Ramchandran, K., Jordan, M. I., and Jiao, J. Sample complexity and representation ability of test-time scaling paradigms. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RJDIX75TEE.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Luong, T., Hwang, D., Nguyen, H. H., Ghiasi, G., Chervonyi, Y., Seo, I., Kim, J., Bingham, G., Lee, J., Mishra, S., Zhai, A., Hu, C. H., Michalewski, H., Kim, J., Ahn, J., Bae, J., Song, X., Trinh, T. H., Le, Q. V., and Jung, J. Towards robust mathematical reasoning, 2025. URL https://arxiv.org/abs/2511.01846.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., and Clark, P. Self-Refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Singh, H., Li, X., Sareen, K., Maheswaran, M., Tan, S., Wu, X., Wang, J., Ariyak, A., Wu, Q., Khaki, S., Tiwari, R., Lian, L., Lu, Y., Li, B., Suhr, A., Athiwaratkun, B., and Keutzer, K. V_1: Unifying generation and self-verification for parallel reasoners, 2026. URL https://arxiv.org/abs/2603.04304.

Venkatraman, S., Jain, V., Mittal, S., Shah, V., Obando-Ceron, J., Bengio, Y., Bartoldson, B. R., Kailkhura, B., Lajoie, G., Berseth, G., Malkin, N., and Jain, M. Recursive Self-Aggregation unlocks deep thinking in large language models, 2026. URL https://arxiv.org/abs/2509.26626.

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Xie, Y., Goyal, A., Zheng, W., Kan, M.-Y., Lillicrap, T. P., Kawaguchi, K., and Shieh, M. Monte Carlo Tree Search boosts reasoning via iterative preference learning, 2024. URL https://arxiv.org/abs/2405.00451.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of Thoughts: Deliberate problem solving with large language models, 2023. URL https://arxiv.org/abs/2305.10601.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language Agent Tree Search unifies reasoning acting and planning in language models, 2024. URL https://arxiv.org/abs/2310.04406.

A. Implementation Details for BARSA

This appendix gives implementation details for the three decision subroutines used in Algorithm 1: CONFIDENT, AFFORDABLE, and PROGRESS. These subroutines are intentionally lightweight. They use only quantities already observed during inference: parsed answer histograms, elapsed time, remaining wall-clock time, and the estimated cost of another RSA round.

A.1. Histogram Statistics

For an answer histogram H, let $\mathrm{supp}(H)$ denote the set of distinct parsed answers with nonzero count. Let

$$m_1(H) = \max_{a \in \mathrm{supp}(H)} H(a)$$

be the count of the largest answer class. Let $m_2(H)$ be the count of the second-largest answer class, with $m_2(H) = 0$ if there is only one distinct parsed answer. We define the top-two gap by

$$\Delta(H) = m_1(H) - m_2(H),$$

and the number of distinct answers by

$$D(H) = |\mathrm{supp}(H)|.$$

The current predicted answer $\hat{a}$ is the answer attaining $m_1(H)$, with ties broken by a fixed deterministic rule. In AIMO 3 submissions, the rule was based on the average entropy of all tokens in a generation. Other metrics like generation length may also be used.

A.2. Maximum Time Budget

This is the maximum time a problem is allowed to take. Initially, this is set to 2-3 times the average time per problem allowed under the global time limit.

A.3. Confidence Rule

The confidence rule decides whether the current answer histogram is already decisive enough to stop. It combines two metrics: **(i)** $m_1(H)$ the size of the largest answer class and **(ii)** $\Delta(H)$, the gap between the largest and second-largest answer classes. The first condition measures raw consensus, while the second condition prevents stopping when the top answer is only marginally ahead of a competing answer.

Algorithm 2 CONFIDENT

Input: answer histogram H, consensus threshold τ_{top}, gap threshold τ_{gap}
Output: Boolean decision
$m_1 \leftarrow$ count of largest answer class in H
$m_2 \leftarrow$ count of second-largest answer class in H
$\Delta \leftarrow m_1 - m_2$
if $m_1 \geq \tau_{\mathrm{top}}$ and $\Delta \geq \tau_{\mathrm{gap}}$ **then**
  return `TRUE`
else
  return `FALSE`
end if

In our experiments, the thresholds are treated as scheduler hyperparameters. For a batch size of eight candidates per round, a natural setting is to require a top answer class of at least four candidates ($m_1(H) = 4$), together with a nontrivial top-two gap ($\Delta = 2$).

A.4. Affordability Rule

The affordability rule checks whether launching another RSA round is safe under both the current problem budget and the remaining global wall-clock budget. The rule uses the estimated cost $\widehat{T}$ of one additional RSA round. This estimate is computed from the observed Round 0 runtime T_0:

$$\widehat{T} = \max(\lambda T_0, T_{\min}),$$

where λ is a fixed multiplier and $T_{\min}$ is a small minimum round-cost estimate. In our implementation, λ is chosen to reflect the empirical observation that later aggregation rounds usually cost less than the initial independent generation round.

Algorithm 3 AFFORDABLE

Input: scheduler state S, current problem budget B, estimated next-round cost $\widehat{T}$
Output: Boolean decision
$t_{\mathrm{cur}} \leftarrow$ elapsed time spent on the current problem
$T_{\mathrm{rem}} \leftarrow$ remaining global wall-clock time
$R_{\mathrm{future}} \leftarrow$ reserved time for future unsolved problems
$\epsilon_{\mathrm{prob}} \leftarrow$ per-problem safety margin
$\epsilon_{\mathrm{global}} \leftarrow$ global safety margin
if $t_{\mathrm{cur}} + \widehat{T} + \epsilon_{\mathrm{prob}} > B$ **then**
  return `FALSE`
end if
if $T_{\mathrm{rem}} - \widehat{T} - \epsilon_{\mathrm{global}} < R_{\mathrm{future}}$ **then**
  return `FALSE`
end if
return `TRUE`

The first condition prevents the current problem from ex-

ceeding its allocated budget. The second condition prevents the system from spending time needed for later problems. The reserve term R_{future} may be computed from a fixed per-problem reserve policy, from a known difficulty prior, or from an adaptive time-bank policy.

A.5. Progress Rule

The progress rule decides whether the most recent RSA round produced useful movement in the answer distribution. Since Round 0 has no previous histogram for comparison, the progress rule is only applied after at least one aggregation round has completed.

Progress is defined broadly. A round is considered useful if it makes the distribution more concentrated, more decisive, or explores new plausible answer classes. In particular, we count any of the following as progress: the largest answer class increases, the top-two gap increases, the number of distinct answers decreases, or a new answer appears.

Algorithm 4 PROGRESS

Input: previous histogram H_{prev}, current histogram H_{cur}
Output: Boolean decision
$m_1^{\text{prev}} \leftarrow$ count of largest answer class in H_{prev}
$m_1^{\text{cur}} \leftarrow$ count of largest answer class in H_{cur}
$\Delta_{\text{prev}} \leftarrow$ top-two gap of H_{prev}
$\Delta_{\text{cur}} \leftarrow$ top-two gap of H_{cur}
$D_{\text{prev}} \leftarrow$ number of distinct answers in H_{prev}
$D_{\text{cur}} \leftarrow$ number of distinct answers in H_{cur}
$A_{\text{prev}} \leftarrow \text{supp}(H_{\text{prev}})$
$A_{\text{cur}} \leftarrow \text{supp}(H_{\text{cur}})$
if $m_1^{\text{cur}} > m_1^{\text{prev}}$ **then**
  return TRUE
end if
if $\Delta_{\text{cur}} > \Delta_{\text{prev}}$ **then**
  return TRUE
end if
if $D_{\text{cur}} < D_{\text{prev}}$ **then**
  return TRUE
end if
if $A_{\text{cur}} \setminus A_{\text{prev}} \neq \emptyset$ **then**
  return TRUE
end if
return FALSE

The first three conditions capture convergence: the top answer becomes stronger, the leading answer separates from the runner-up, or the distribution becomes less diffuse. The final condition captures exploration. This is useful because RSA can sometimes recover from an initially poor or diffuse answer set by introducing a correct answer that was absent from the initial candidate pool. If none of these changes occurs, the system treats the latest RSA round as having failed to make progress and stops to avoid spending further budget on an unproductive trajectory.

A.6. Discussion of the Stopping Logic

The three subroutines implement the stopping rule used by BARSA. After each round, the system returns the current majority answer if the answer histogram is already confident, if another round is not affordable, or if the most recent RSA round did not make progress. Otherwise, BARSA launches another aggregation round, subject to the maximum round limit $R_{\max}$.

This design deliberately separates the roles of the three checks. CONFIDENT asks whether the current answer is sufficiently stable. AFFORDABLE asks whether another round can be safely completed under the global budget. PROGRESS asks whether the previous round produced enough useful signal to justify continuing. Together, these checks make RSA adaptive rather than fixed-depth: easy or high-consensus problems stop early, while uncertain problems can receive additional test-time compute when there is both time available and evidence that aggregation is helping.

B. Internal Benchmark Details

A core difficulty in developing inference strategies for AIMO 3 was that submissions to the public leaderboard only returned a final score. Moreover, problems were served in random order, and this introduced additional variance that made harder to pinpoint which system improvements were helpful. We therefore constructed a small internal benchmark of AI-hard problems to guide development. The benchmark was not designed to approximate the full AIMO 3 test distribution. Its purpose was to identify the various failure modes of our inference system, and help us think of ideas of how to best remedy them.

We constructed the benchmark from IMO-AnswerBench (Luong et al., 2025), supplemented by one problem from the official AIMO 3 reference benchmark. The construction process was as follows:

1. Filter the problems to retain only problems whose answers were integers, matching the answer format used in AIMO 3.

2. Run a majority@8 baseline on the filtered set. For each problem, we performed four independent runs, each consisting of eight sampled solutions followed by answer parsing and majority voting.

3. Select problems that were not solved in at least one of the four runs. This produced an initial pool of 53

candidate hard problems.

4. Manually review these candidate problems to remove cases that were unsuitable for benchmarking. This included problems with incorrect ground-truth answers, ambiguous wording, missing information, or failures caused primarily by convention mismatches rather than mathematical difficulty.

5. From the remaining pool, we retained 17 problems that were both valid and genuinely difficult for the baseline system.

B.1. Sample Problems

All sample problems are drawn from the `imo_answerbench` dataset (Luong et al., 2025).

B.1.1. IMO-BENCH-ALGEBRA-100

Table 5. This problem is based on Vasc's inequality. It is an example of a problem in class 5. There is a short but tempting solution leading to the wrong answer, while the correct answer requires substantially deeper reasoning that the model often skips. The first 8 rows correspond to the initial generations, and the histogram forms a 7–1 split between two answers. This resembles an easy question, but the majority class is wrong.

Attempt	Length	Tool Calls	Entropy	Answer
4	21156	32	0.705	2
3	25723	41	0.675	2
7	25633	41	0.722	2
2	26634	51	0.659	2
5	30561	50	0.635	2
6	31621	46	0.681	2
1	33759	66	0.628	8
8	37998	52	0.685	2
–	—-	–	—-	–
16	5251	0	0.739	2
11	9546	19	0.654	8
9	11637	24	0.689	8
14	13079	12	0.710	2
12	13168	22	0.721	8
15	13478	27	0.762	8
13	30743	60	0.678	2
10	35389	59	0.659	2
–	—-	–	—-	–
21	8553	16	0.627	8
17	10289	15	0.691	8
20	10872	21	0.645	8
24	12165	27	0.634	8

Problem. Find the number of triples (x, y, z) of real numbers satisfying

$$x^2 + y^2 + z^2 = xy^3 + yz^3 + zx^3 = 3.$$

Answer. 8.

An easy guess-and-check argument finds two solutions:

$$(1, 1, 1) \quad \text{and} \quad (-1, -1, -1).$$

From here, the model often concludes that there are no other solutions. However, there are additional solutions which are difficult to find.

Vasc's inequality states that for $x, y, z \in \mathbb{R}$,

$$(x^2 + y^2 + z^2)^2 \geq 3(xy^3 + yz^3 + zx^3).$$

The problem statement says that

$$x^2 + y^2 + z^2 = 3 \quad \text{and} \quad xy^3 + yz^3 + zx^3 = 3,$$

so we are in the equality case of Vasc's inequality.

In the equality case, there are additional solutions when x, y, and z are in the ratio

$$x : y : z = \sin^2\left(\frac{4\pi}{7}\right) : \sin^2\left(\frac{2\pi}{7}\right) : \sin^2\left(\frac{\pi}{7}\right).$$

We may also cyclically permute x, y, z; transpositions are not allowed. Flipping all signs also preserves equality. Hence there are an additional 6 solutions to the equality expression, giving a final answer of 8.

B.1.2. IMO-BENCH-COMBINATORICS-031

This problem is from the USA IMO Team Selection Test 2020, Day 1 Problem 3.

Problem. Let $\gamma \geq 1$ be a real number. Sun Wukong and the Sea God play a turn-based game on an infinite grid of unit squares. Before the game starts, the Sea God chooses a finite number of cells to be flooded with seawater. Sun Wukong is building a magical barrier, which is a subset of unit edges of the grid, called walls, forming a connected, non-self-intersecting path or loop. Additionally, there is a magical artifact that randomly generates a finite number of extra walls on the grid, with no specific pattern or distribution.

The game then begins with Sun Wukong moving first. On each of Sun Wukong's turns, he adds one or more walls to the magical barrier, as long as the total length of the barrier is at most γn after his nth turn. On each of the Sea God's turns, every cell which is adjacent to an already flooded cell and with no wall between them becomes flooded as well. Sun Wukong wins if the magical barrier forms a closed loop such that all flooded cells are contained in the interior of the loop, hence stopping the flood. What is the largest constant C such that, for all $\gamma > C$, Sun Wukong can guarantee victory in a finite number of turns no matter how the Sea God chooses the initial cells to flood?

Answer. 2.

This is another example of a problem in class 5. The "magical artifact" statement is a distractor, but it does not affect strong models such as `gpt-oss-120b`.

The short but tempting solution is to observe that, when unrestricted, the perimeter of the flood grows by 8 each turn, and then conclude that $C > 8$ is needed for Sun Wukong to guarantee victory. The model often makes this observation and stops here, returning the answer 8.

However, the correct solution uses a construction based on a different intuition. Sun Wukong can first cut off the flood from one direction. In this step, he only needs to ensure that the wall on that side grows fast enough compared with the flood's spread, so it suffices to have $C > 2$. Next, he can restrict the flood to a long, narrow corridor by constructing two parallel sides. Finally, he can construct the last wall to seal off the flood. Thus, Sun Wukong only needs to ensure that the narrow corridor grows in length fast enough compared with the spread of the flood, which means $C > 2$ is sufficient. The correct answer is therefore 2.

B.1.3. IMO-BENCH-COMBINATORICS-013

Problem. There are 42 students participating in the Team Selection Test. Each of them is assigned a positive integer from 1 to 42 such that no two students have the same number and every number from 1 to 42 is assigned to a student. The team leader wants to select a subset of these students such that no two selected students have numbers whose difference is 1 or 21. For example, the team leader can pick the set

$$\{1, 3, 6, 25, 28, 34, 42\},$$

but not

$$\{1, 2, 4, 6, 24\} \quad \text{or} \quad \{1, 3, 24, 26, 28\}.$$

How many ways can the team leader pick such a subset? Give your answer modulo 100000.

Answer. 26555.

This is an example of a problem in class 2.

The model formulates this problem as an independent-set count on a graph. The numbers 1–42 form a 2×21 ladder, and the task becomes counting subsets with no adjacent chosen vertices. From this, the model sometimes counts that there are 131836323 valid choices, and returns the answer 36323.

However, the constraints also mean that 21 and 22 cannot be selected at the same time. There are 11309768 such selections, and hence the correct count is

$$131836323 - 11309768 = 120526555.$$

Therefore, the correct answer modulo 100000 is

$$120526555 \equiv 26555 \pmod{100000}.$$

The model only makes this extra reasoning step around half of the time.

B.2. Filtered Problems

During benchmark construction, we manually reviewed problems whose evaluation behaviour suggested possible ambiguity, parser sensitivity, or dataset-quality issues. We excluded the following problems from the final internal benchmark because their failures were not cleanly attributable to mathematical reasoning difficulty.

Here, we give details on some problems we have decided to exclude from our internal benchmark.

B.2.1. IMO-BENCH-GEOMETRY-002

Problem. Given right triangle XYZ with hypotenuse XZ and $\angle X = 50°$, points P and Q on the side YZ are such that

$$\angle PXZ = \angle QXY = 10°.$$

Compute the ratio

$$2 \times \frac{YQ}{ZP}.$$

Reason for exclusion. The modified problem asks for $2 \times \frac{YQ}{ZP}$. The original problem labelled the points as A, B, C instead of X, Y, Z, and asked for the equivalent of $\frac{ZP}{YQ}$. It appears that there was a mix-up when relabelling the points to make the problem more AI-robust. The answer should be 1 instead of 4.

B.2.2. IMO-BENCH-GEOMETRY-044

Problem. Given that $PQRS$ is a parallelogram, $\angle S = 60°$, $PS = 2$, and

$$PQ = \sqrt{3} + 1.$$

Point N is the midpoint of PS. Segment RE is the angle bisector of $\angle R$. Find the angle $\angle REQ$ in degrees.

Reason for exclusion. The original problem used a diagram that specified the intended position of the point E. In the modified text-only version, this information is missing. Without specifying that E lies on the relevant segment, the model can reasonably infer a different location for E, leading to a different answer. We excluded this problem because the text statement is under-specified without the missing diagram or condition.

B.2.3. IMO-BENCH-NUMBER_THEORY-055

Problem. A positive integer m consisting of distinct digits is considered "good" if it is a single-digit number, or if removing one of its digits results in a divisor of m that is also a good number. Find the largest good number.

Reason for exclusion. The original problem includes the additional condition that numbers do not start with zero. This condition is absent from the modified dataset version, and its absence fundamentally changes the answer. With the no-leading-zero condition, the answer is 146250; without it, the answer is 903125. We excluded this problem because a crucial condition from the original problem was omitted.

B.2.4. IMO-BENCH-ALGEBRA-039

Problem. Let p, q, r, and s be constants such that the equation

$$py^3 + qy^2 + ry + s = 0$$

has three distinct real roots. Find all possible values for the number of distinct real roots of the equation

$$(pz^3 + qz^2 + rz + s)(6pz + 2q) = (3pz^2 + 2qz + r)^2.$$

Reason for exclusion. The numerical modification in the dataset fundamentally changes the problem. The original Ukraine 1997 problem asks for the equation

$$4(px^3 + qx^2 + rx + s)(3px + q) = (3px^2 + 2qx + r)^2,$$

whose answer is 2. The modified version is equivalent to

$$2(px^3 + qx^2 + rx + s)(3px + q) = (3px^2 + 2qx + r)^2,$$

for which the answer is 0. Since the modified constants change the mathematical structure of the solution, we excluded this problem from the benchmark.

B.2.5. IMO-BENCH-COMBINATORICS-005

Problem. Determine the number of natural numbers n that have at most 16 digits and satisfy the following conditions:

- $3 \mid n$;
- the digits of n in decimal representation are in the set $\{2, 0, 1, 8\}$.

Reason for exclusion. The issue is a convention mismatch over whether 0 is considered a natural number. The dataset solution assumes that 0 is natural, while many models assume that natural numbers are positive. This leads to an off-by-one discrepancy. Since the failure depends on a convention rather than mathematical reasoning, we excluded the problem.

B.2.6. IMO-BENCH-COMBINATORICS-020

Problem. Suppose there are 40 professional baseball teams participating in a tournament. In each round, the 40 teams are divided into 20 pairs, and each pair plays a game at the same time. After the tournament, it is known that every two teams have played at most one game. Find the smallest positive integer a such that we can arrange a schedule satisfying the above conditions, and if we take one more round, there is always a pair of teams who have already played each other.

Reason for exclusion. There is a tension here between a "maximally cooperative scheduling" interpretation and an "adversarial scheduling" interpretation.

Under the maximally cooperative interpretation, the schedule is designed so that every pair of teams can eventually play each other in a round-robin style tournament. Such a schedule can always be found. With this interpretation, every team can play against every other team, and the answer is 39.

Under the adversarial interpretation, the pairs are picked in a way that tries to create an obstruction as early as possible. The problem must then be interpreted in a more specific manner:

1. A team might not play against every other team.
2. Let a be the number of round pairings scheduled so far. For small values of a, the next pairing can be found without obstruction.
3. As the selection proceeds and a increases, there is eventually an obstruction after X rounds that prevents the next round of pairings. This means that, for every possible pairing, there is some team that has already played against its proposed opponent in a previous round.
4. There exists an adversarial pair-scheduling strategy that minimizes the value of X.
5. The task is to find this minimal value of X.

In this interpretation, the answer is 21.

The second interpretation makes the problem much harder, and is probably what the problem author intended, considering that this was a Team Selection Test problem. However, it is unfair to expect the model to infer this interpretation. It is not unreasonable for a model to default to the simpler interpretation.

B.2.7. IMO-BENCH-COMBINATORICS-030

Problem. A cube with size $18 \times 18 \times 18$ consists of 5832 unit cubes, all colored white. Anton and Berta play a game

on this cube. Anton chooses some pillars with size $1\times1\times18$ such that no two pillars share a vertex or side, and turns all chosen unit cubes to black. Berta is allowed to choose some unit cubes and ask Anton their colors. In addition, Anton also tells Berta that he painted at least two pillars. How many unit cubes, at least, does Berta need to choose so that, for any answer from Anton, Berta can always determine the black unit cubes?

Reason for exclusion. It is unclear from the statement whether entire pillars are selected and painted as units, or whether there are additional hidden constraints on how black cubes are chosen. If entire pillars are chosen at once, the problem appears to reduce to an 18×18 grid of possible pillars, suggesting a different answer from the dataset ground truth. The dataset answer of 486 suggests that the intended interpretation includes additional structure that is not explicit in the statement. We therefore excluded this problem as an interpretation issue.

B.2.8. IMO-BENCH-COMBINATORICS-048

Problem. Find the smallest positive integers n for which the numbers in the set

$$S = \{1, 2, \ldots, n\}$$

can be colored red and blue, with the following condition being satisfied: the set $S \times S \times S$ contains exactly 2007 ordered triples (x, y, z) such that:

1. x, y, z are of the same color;
2. $x + y + z$ is divisible by n.

Reason for exclusion. There are two integers satisfying the condition, namely 69 and 84. The dataset ground truth is 69, the smaller of the two. However, the statement asks for the smallest positive "integers" in the plural. A model output such as

$$\boxed{69, 84}$$

is therefore a reasonable response to the literal wording, but would be graded as wrong. We excluded the problem because the wording creates an avoidable answer-format ambiguity.

B.2.9. IMO-BENCH-COMBINATORICS-076

Problem. The Lucas numbers $L_0, L_1, L_2, \ldots$ are defined by $L_0 = 2$, $L_1 = 1$, and

$$L_{n+1} = L_n + L_{n-1}$$

for $n \geq 1$. The Fibonacci numbers $F_0, F_1, F_2, \ldots$ are defined by $F_0 = 0$, $F_1 = 1$, and

$$F_{n+1} = F_n + F_{n-1}$$

for $n \geq 1$. Determine the smallest size of a set S of integers such that for every $k = 2, 3, \ldots, 125$, there exist $x, y \in S$ such that

$$x - y = F_k.$$

Also, there exist some $a, b \in T$ for some set T such that

$$a - b = L_{100}.$$

Reason for exclusion. The final sentence is intended as a red herring, since it refers to a separate set T. However, a model may reasonably interpret T as a typo for S and treat the sentence as an additional condition. Since language models are often expected to be robust to minor prompt errors, this behaviour is not clearly a mathematical reasoning failure. We excluded the problem because the added distractor creates an ambiguity between irrelevant information and a possible typo.

B.2.10. IMO-BENCH-COMBINATORICS-079

Problem. Determine the largest N for which there exists a table T of integers with N rows and 16 columns that has the following properties:

1. Every row contains the numbers $1, 2, \ldots, 16$ in some order.
2. For any two distinct rows r and s, there is a column c such that
$$|T(r, c) - T(s, c)| \geq 2.$$

In addition to the given constraints, every column in the table must contain distinct integers. Here $T(r, c)$ means the number at the intersection of row r and column c.

Reason for exclusion. The added condition that every column must contain distinct integers imposes a much stronger constraint than the original problem. This fundamentally changes the set of valid configurations. Since the model is solving a materially altered problem rather than the intended one, we excluded this problem from the benchmark.

B.2.11. IMO-BENCH-COMBINATORICS-085

Problem. Evan fills the fields of a 78×78 board with numbers from 1 to 6084, each number being used exactly once. She then counts the total number of good paths on the board. A good path is a sequence of fields of arbitrary length, including length 1, such that:

1. the first field in the sequence is one that is only adjacent to fields with larger numbers, and also one that has an even row number and an even column number;
2. each subsequent field in the sequence is adjacent to the previous field;

3. the numbers written on the fields in the sequence are in increasing order.

Two fields are considered adjacent if they share a common side. Find the smallest possible number of good paths Alice can obtain.

Reason for exclusion. The added condition that the starting field must have both even row number and even column number fundamentally changes the problem. There are

$$39 \times 39 = 1521$$

such fields. If all non-even-even fields are filled with

$$1, 2, \ldots, 4563$$

and all even-even fields are filled with

$$4564, 4565, \ldots, 6084,$$

then every even-even field has an adjacent non-even-even field with a smaller number. Hence no even-even field is adjacent only to larger numbers, so there are no valid starting fields and the number of good paths is 0. This contradicts the dataset ground truth of 12013. We excluded the problem because the added condition changes the answer.

B.2.12. IMO-BENCH-COMBINATORICS-097

Problem. In a circular seating arrangement at a party, there are 16 guests sitting at the table playing a game. Each guest has a unique name tag created by the game master, and these name tags are randomly distributed among the guests. In each of the next n rounds, the guests play the game according to the following rule:

1. Any guest who has their own name tag exits the table.
2. The remaining guests pass their name tags to the guest sitting immediately to their right.

We are interested in finding the number of ways the name tags can be distributed such that there exists at least one guest who does not leave the table after 4 rounds.

Reason for exclusion. The phrase "we are interested in finding" is weaker than a direct instruction to compute and return the final integer. In our runs, this sometimes interacted badly with the answer-formatting prompt: the model returned code or a formula for computing the answer rather than the final integer itself. We therefore classified this as an instruction-following or answer-formatting issue rather than a mathematical reasoning failure, and excluded it from the benchmark.