Scaling Mathematical Reasoning through Data, Tools, and Generative Selection

Anonymous Authors[1]

Abstract

This paper presents our high-scoring submission to the AI Mathematical Olympiad - Progress Prize 2 (AIMO-2) competition. Our recipe for building state-of-the-art mathematical reasoning models relies on three key pillars. First, we create a large-scale dataset comprising 540K unique high-quality math problems, including olympiad-level problems, and their 3.2M long-reasoning solutions. Second, we develop a novel method to integrate code execution with long-reasoning models through iterative training, generation, and quality filtering, resulting in 1.7M high-quality Tool-Integrated Reasoning solutions. Third, we create a pipeline to train models to select the most promising solution from many candidates. We show that such generative solution selection (GenSelect) can significantly improve upon the majority voting baseline. Combining these ideas, we train a series of models that achieve state-of-the-art results on mathematical reasoning benchmarks. To facilitate further research, we will release our code, models, and the complete `MathReason` dataset upon publication.

1. Introduction

Recent advances in large language models (LLMs) have significantly improved their ability to solve complex reasoning tasks, including olympiad-level mathematics. A key idea behind this progress has been to allow models to spend more tokens thinking about the solution before producing the final answer. Initially, models were trained to produce a series of intermediate solution steps (chain-of-thought (CoT) (Wei et al., 2022)). More recently, *long reasoning* models (Jaech et al., 2024; Guo et al., 2025) have learned to reflect on their work, exploring and refining multiple strategies within a single generation. This has led to further improvements across mathematics, coding, and scientific domains. To keep pace with this rapid development, the community has introduced increasingly challenging benchmarks and competitions that help to evaluate the progress.

The AI Mathematical Olympiad - Progress Prize 2 (AIMO-2) is an initiative designed to assess advancements in this domain by challenging participants to create models capable of solving 50 difficult, national-level mathematical problems within strict computational limits. These problems were never published online, ensuring a more rigorous evaluation compared to traditional benchmarks. This paper details our high-scoring submission to the competition. To develop the state-of-the-art recipe, we focused on addressing several limitations of the publicly available reasoning models that we describe below.

Large-scale long-reasoning dataset (§2). To improve existing models, we started by collecting an extensive set of mathematical problems from the internet. We developed an LLM-based problem extraction and refinement pipeline to construct a dataset of 540K unique problems. Using this dataset, we then generated 3.2M long-reasoning CoT solutions by prompting `DeepSeek-R1` (Guo et al., 2025) and `QwQ-32B` (Team, 2025b). Training `Qwen2.5-Base` models (Yang et al., 2025) on this large-scale distillation data, we are able to surpass the accuracy of all other open-weight models of comparable size, except for `QwQ-32B`, which is slightly better than our 32B model.

Tool-Integrated Reasoning (§3). To improve the results further, we developed a method for integrating code execution into long-reasoning generations. Our initial attempts to elicit Tool-Integrated Reasoning (TIR) from `DeepSeek-R1` and `QwQ-32B` through simple prompting proved unsuccessful. We hypothesize that these models struggle to deviate from their standard solution format due to extensive training on reasoning tasks and limited exposure to instruction-following. To overcome this challenge, we built a pipeline that starts with a small-scale reasoning finetuning of an *instruction-following* model (Ye et al., 2025). By prompting this model to generate long-reasoning TIR solutions followed by aggressive quality filtering, we established an initial dataset suitable for training. Through multiple iterations of training, generation, and filtering, we constructed a 1.7M TIR solution set that was crucial for improving the ac-

[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.

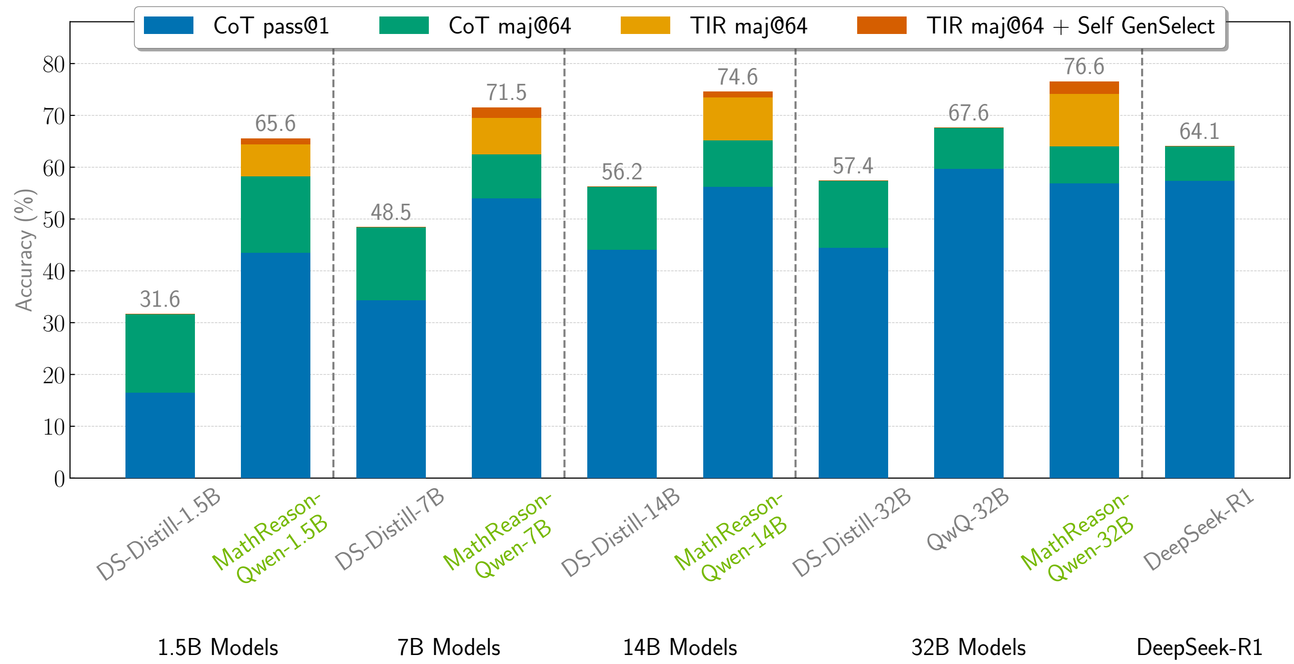


Figure 1: Accuracy of `MathReason-Qwen` models and comparable baseline models on math problems from AIME and HMMT competitions. Even our `MathReason-Qwen-1.5B` model outperforms `DeepSeek-R1`, which has 400x more parameters (24x more active parameters).

curacy of our final models. To make TIR more efficient, we also developed a method to accurately control the number of code executions the model is allowed to make for each generation.

Generative Solution Selection (§4). A common approach to maximize model accuracy is to generate multiple candidate solutions and select the most promising one. While majority voting (Wang et al., 2023) serves as a strong baseline, its performance falls significantly short of the theoretical maximum performance of pass@k. To address this limitation, we developed a pipeline for training models to identify the most promising solution when presented with multiple candidates. We generated 566K selection examples to train our models.

Combining these three innovations, we developed a series of state-of-the-art open-weight math reasoning models with 1.5B, 7B, 14B, and 32B parameters. Each model supports CoT, TIR, and GenSelect inference modes when appropriately prompted.

To accelerate progress in open-source mathematical reasoning, we will release our code, finetuned `MathReason-Qwen` models, and the complete `MathReason` dataset upon publication.

2. Data Preparation

In this section, we outline our validation and training data curation pipeline. Section 2.1 presents our methodology for preparing a large-scale problem set for training. Section 2.2 describes our validation set collection process. Finally, Section 2.3 details our approach to synthesizing long-reasoning Chain-of-Thought (CoT) solutions.

2.1. Problem preparation

To collect math problems, we leverage the Art of Problem Solving (AoPS) community forums. We include all forum discussions except "Middle School Math", which we found to be too elementary and unhelpful for training in our preliminary experiments. After retrieving forum discussions, we implement a systematic process to extract problems and their corresponding answers. Throughout our pipeline, we utilize `Qwen2.5-32B-Instruct` (Yang et al., 2025) for all processing steps unless otherwise specified.

Our data processing pipeline starts with the problem extraction from initial forum posts. Subsequently, each potential problem is classified to filter out multiple-choice questions, binary (yes-or-no) questions, and invalid problems (e.g., those lacking context). Proof-based problems are transformed into equivalent answer-seeking questions. For the remaining non-proof questions, we attempt to extract final answers from the forum discussions. Finally,

Table 1: Comparison with other datasets sourced from AoPS forums. Our work was done concurrently with (Mahdavi et al., 2025) and (LI et al., 2024).

Dataset	# of Problems
MathReason (ours)	540K
AoPS-Instruct (Mahdavi et al., 2025)	650K
NuminaMath-1.5 (AoPS part) (LI et al., 2024)	68K

following (Yang et al., 2023), we perform an LLM-based decontamination to remove potential paraphrases of questions in popular math benchmarks. Starting with approximately 620K forum discussions, our pipeline yields a final dataset of 540K unique problems. A more detailed description of each pipeline stage, including specific prompts and classification criteria, as well as the detailed breakdown of the dataset size after each processing stage and the final dataset composition, is provided in Appendix A. Table 1 compares our approach with other popular datasets sourced from AoPS forums.

Table 2: Final distribution of CoT solutions in our dataset.

Model	CoT solutions after filtering	CoT solutions all
QwQ-32B	0.5M	1.0M
DeepSeek-R1	2.7M	4.2M
Total	3.2M	5.2M

2.2. Comp-Math-24-25 Benchmark

To create a robust validation dataset for our evaluation, we combined problems from American Invitational Mathematics Examinations (AIME) and Harvard-MIT Mathematics Tournaments (HMMT) gathered from the AoPS forums. We restricted our selection to 2024 and 2025 competitions to minimize potential data contamination. AIME and HMMT problems were selected for our validation set due to their strong alignment with AIMO-2 competition requirements. We excluded proof-based questions and those awarding partial credit based on estimate accuracy, as these are generally incompatible with an exact match evaluation framework. The resulting dataset, which we call **Comp-Math-24-25**, consists of 256 problems, as detailed in Appendix B.

2.3. Text-based Solution Synthesis

To generate CoT solutions, we follow a common pipeline of directly prompting an existing open-weight LLM to solve problems collected in Section 2.1. We utilize `DeepSeek-R1` and `QwQ-32B` models and generate up to 32 solution candidates for each problem in our dataset. We use temperature 0.7, top-$p = 0.95$, and limit generations to 16384 tokens. We generate more solutions for *harder* problems with known answers, where the hardness was estimated by computing an average pass-rate across 32 generations from the `Qwen2.5-72B-Math-Instruct` model (Yang et al., 2024).

As the final filtering step, we remove any solutions that do not reach the expected answer. Following (Toshniwal et al., 2025), predicted and expected answers are compared by prompting `Qwen2.5-32B-Instruct` to judge whether they are equivalent in the context of the problem. For each problem where we were unable to extract the final answer, as well as for all converted proofs, we treat the most common answer across all available solution candidates as the ground truth. Table 2 shows the final distribution of CoT solutions in our dataset. *Note that out of the 540K problems, we could synthesize solutions for only 428K problems using this pipeline.*

3. Tool-Integrated Reasoning

Allowing LLMs to integrate natural language reasoning with Python code execution is a known way of improving accuracy on challenging math problems (Toshniwal et al., 2024; Yang et al., 2024). However, the best open-weight reasoning models (most notably `DeepSeek-R1` (Guo et al., 2025) and `QwQ-32B` (Team, 2025b)) are not able to directly produce such Tool-Integrated Reasoning (TIR) solutions. Our initial attempts to induce TIR generations by prompting these reasoning models with direct instructions or few-shot examples turned out to be unsuccessful. Unable to solve this via prompting, we had to develop a more elaborate pipeline for building reasoning models capable of producing TIR solutions.

In our early experiments, we noticed that when non-reasoning *instruct* LLMs are trained on a limited quantity of reasoning data (Ye et al., 2025), they tend to retain their good instruction-following abilities. Building on this intuition, we were able to successfully prompt the `LIMO-Qwen-32B` model (Ye et al., 2025) to produce TIR solutions, but found that they tended to be *low-quality* on average. The produced code was often irrelevant or was merely used to verify calculations of preceding CoT steps. To overcome this, we developed a filtering step to retain only high-quality examples where code execution provides substantial reasoning benefits. Using this filtered dataset, we then fine-tuned our reasoning model, achieving significant performance improvements over the CoT-only predecessor. Finally, we employed an iterative model improvement approach by training a more powerful TIR model in each iteration and using it to generate and filter additional TIR examples, further enhancing model performance. In the following subsections, we detail each stage of this pipeline.

3.1. Instruction-following reasoning model

Prior work (Muennighoff et al., 2025; Ye et al., 2025) shows that fine-tuning on as few as 1K samples is sufficient to make LLM produce long-CoT solutions. We hypothesize that an *instruct* model fine-tuned on such a small dataset can potentially preserve its instruction-following and long-reasoning capabilities.

To test this, we prompted LIMO-Qwen-32B to solve the problem using Python code for the steps that require complex calculations. Appendix E.1 provides the zero-shot prompt we designed for this purpose. For roughly half of the problems, the model produced a solution that contained at least one Python code block. We then synthesized 1.2M solutions for MathReason problems, using temperature = 0.7, top-p = 0.95, allowing maximum sequence length of 16384 tokens and stopping generations if the solution contained more than 8 code executions.

3.2. Filtering TIR data

Careful inspection of generated solutions revealed that code execution often does not benefit the solution and could easily be replaced with several simple CoT steps (see example in Appendix I.2). Instead, we want an ideal TIR solution to provide significant shortcuts by implementing otherwise infeasible brute-force approaches. We apply several filters to remove solutions with unwanted code usages. First, we utilize Qwen2.5-32B-Instruct to classify each code block by two criteria:

- **novel calculation / verification**. Whether the code execution leads to a novel result or it simply verifies the previous steps (see the prompt in Appendix E.2).
- **significant / moderate / trivial**. Whether the code implements an important part of the solution or is easily substitutable with several CoT steps (see the prompt in Appendix E.3).

We then only keep solutions that either have at least one novel and significant code block or more than half novel and moderate code blocks. Additionally, we apply rule-based filtering and remove solutions with incorrect final answer and solutions without code execution. We also remove solutions with more than two code blocks, as we found it to be helpful in our preliminary experiments. As part of preprocessing, we also replace the tags marking the start and end of code blocks. In particular, we instruct the LIMO-Qwen model to place code between "```python" and "```\n", following a markdown-like style that models can easily produce; we then replace these with <tool_call> and </tool_call> tags, respectively, to make the code ending tag distinguishable from regular markdown and facilitate code extraction. All described filtering steps result in the TIR dataset, consisting of 15K samples, which we will refer to as *stage-0 TIR data*.

3.3. Iterative data generation

For the next stage of TIR solution generation, we leverage QwQ-32B as it proved to be a powerful yet lightweight synthetic reasoning data generator. For this purpose, we fine-tune it on the *stage-0* data for 7 epochs with a constant learning rate of 5e-6. We then synthesize solutions for MathReason problems. We generate 700K samples and filter them down to 260K by removing incorrect solutions and solutions not using code. Novelty and significance filters degrade the performance at this stage, so we do not use them.

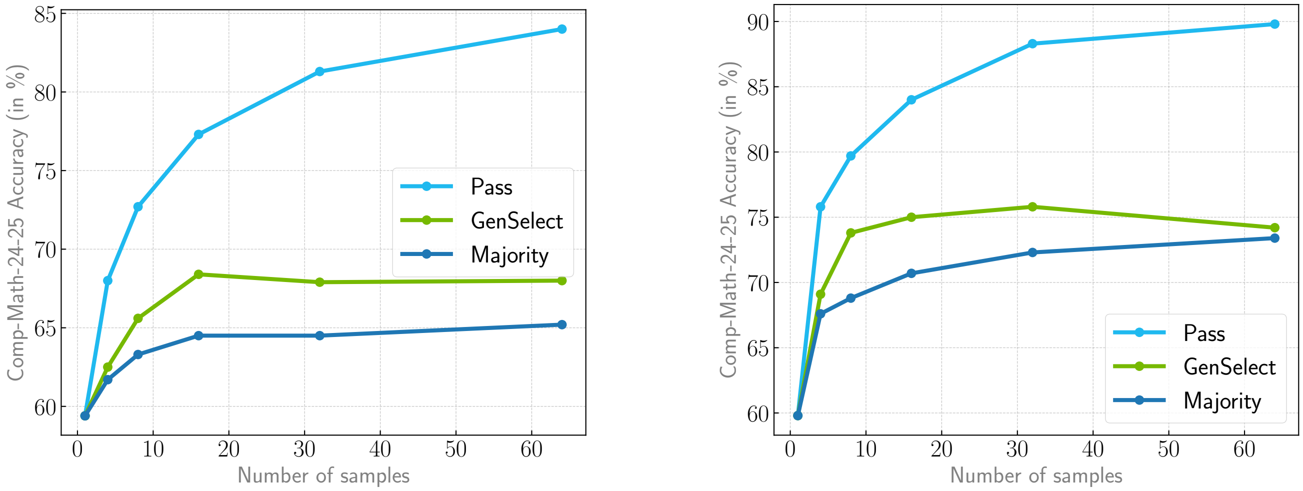
To further improve results, we repeat this process one more time using an intermediate version of our 14B model, which was finetuned on the CoT-only subset of MathReason data. We train this 14B model on QwQ-32B solutions and then execute a final round of data generation and filtering, ultimately resulting in the final 1.7M TIR dataset.

3.4. Controlling the number of code blocks

We developed a simple, yet effective method to control the number of code blocks that the model can use. During all data generation stages, we format the code output as shown in Appendix I.1, appending additional notification warnings about how many code executions are remaining. The model often refers to this message in its thinking process, refraining from further code usage when the limit is reached. For each problem, we randomly select between 1 and 8 allowed code executions and provide this information in the prompt. We remove generations that try to use more code blocks than requested to reinforce the correct behavior in training. As a result, the model learns to follow the specified code execution limit. An example of this behavior is provided in Appendix I.3.

4. Generative Solution Selection

We observe a considerable gap between the majority@k vs pass@k performance for our models, implying the models' theoretical ability to solve far more problems than can be achieved with a majority answer. To bridge this gap, we explore training a model that, given a set of candidate solution *summaries*, picks the most promising solution. In our early experiments, we found that comparing multiple solutions yields significantly better results than judging each solution in isolation. Following (Zhang et al., 2025), we do not change the model's architecture and instead let it reason in natural language before selecting one of the provided solutions. We detail the pipeline to prepare the training data for such *selection* generations (GenSelect) in the following



(a) 14B CoT

(b) 14B TIR

Figure 2: Comparison of majority, GenSelect and pass metrics for different number of generation samples. To construct the input for GenSelect, we use subsets of 16 solutions (or all if fewer samples were generated). For the final answer, we perform majority@8 over the answers selected by the GenSelect. `MathReason-Qwen -14B` model is used to perform CoT, TIR, and GenSelect inference. We find that GenSelect becomes unstable when using more than 32 generations as we can no longer show all solutions in a single prompt.

sections.

4.1. Creating New Summaries

Solutions generated by reasoning models have a *thinking* part and a *summary* which follows it. We noticed that summaries generated by reasoning models, such as `DeepSeek-R1`, could be very succinct; in extreme cases, they could just be stating the final answer. Since we require a representative summary for comparing different solutions during inference, we replace the *native* summary of the reasoning models by synthesizing new summaries with the `Qwen2.5-32B-Instruct` model. We synthesize four candidate summaries per solution with a maximum length of 2048 tokens. To ensure the summary is faithful, we filter out summaries where the predicted answer is different from the original solution's predicted answer. If there are no valid summaries, we discard the sample[1], otherwise we select the longest summary to replace the original summary. We regenerate summaries for the entire `MathReason` dataset using this process, so that models trained on it can produce these summaries directly. See Appendix H for a comparison between one-word `DeepSeek-R1` summary and a new one generated by `Qwen2.5-32B-Instruct`.

4.2. Generating Selection Candidates

We observed that modest accuracy gains over majority voting can be achieved by simply presenting new solution summaries to reasoning models and prompting them to compare and select one (see prompt in Appendix F.3). Building on this observation, we develop the following pipeline to generate training data for this GenSelect inference. For each problem in the `MathReason` dataset, we randomly sample between 2 and 16 candidate solution summaries. We ensure that each sample group contains at least one correct and one incorrect solution. This process is repeated until we obtain 8 distinct comparison groups for each problem. Using the GenSelect prompt (Appendix F.3), we then task `QwQ-32B` with selecting the most promising solution from each group. See Figure 4 illustrating this pipeline in the Appendix. This procedure generates 1M selections, which we subsequently filter down to 566K by eliminating any instances where incorrect solutions were chosen.

4.3. Reducing computational cost

While this dataset is suitable for training, the comparison generations can be as long as the original solutions, making GenSelect inference computationally expensive. To address this challenge, we explored training models to directly generate the final comparison *summary* rather than learning the full reasoning trace. Consistent with our previous observations, the *native* comparison summaries produced by `QwQ-32B` proved suboptimal. We therefore again used `Qwen2.5-32B-Instruct` to regenerate all comparison summaries (see the prompt in Appendix G.1) and trained our models using these summarized comparisons. Our early experiments revealed only a slight reduction in accuracy

[1]No more than 5% of all samples were discarded this way.

Table 3: Accuracy with majority@64 on the Comp-Math-24-25 benchmark after the first and second SFT rounds. We see significant gains for CoT generations and comparable results for TIR generations.

Model	First SFT	Second SFT
1.5B CoT	55.1	58.2
1.5B TIR	64.1	64.5
7B CoT	61.3	62.5
7B TIR	71.1	70.7
14B CoT	62.9	65.2
14B TIR	74.6	73.4

(1–2%) compared to models trained on the whole reasoning traces.

This final setup makes GenSelect inference remarkably efficient compared to the original long-reasoning generations. With output tokens capped at 2048, most computation occurs in a highly-parallelizable pre-filling phase. Since each solution summary is similarly limited to 2048 tokens, the total input context typically cannot exceed 32768 tokens when using the maximum of 16 solutions per problem. Although more than 16 solutions could theoretically be included in a prompt, we generally observe diminishing returns as the context becomes too large. For scenarios requiring evaluation of more solution candidates, we propose sampling 16 solutions multiple times and then performing majority voting to determine the final answer. Nevertheless, our findings indicate that the most significant accuracy improvements occur when GenSelect is applied to a smaller number of generations (Figure 2).

5. MathReason-Qwen models

In this section, we present the training and evaluation details of the `MathReason-Qwen` models.

5.1. Training

To build our final models we perform supervised-finetuning (SFT) on a series of `Qwen2.5-Base` models (1.5B, 7B, 14B and 32B) (Yang et al., 2025). For 1.5B and 7B models, we start from the special model versions pretrained for mathematical reasoning tasks (Yang et al., 2024). Unlike general Qwen2.5 models, the math versions only support a limited context window of 4096 tokens, which is inadequate for the long-reasoning generations. To overcome this, we follow (bloc97, 2023) and change RoPE (Su et al., 2021) base to 500K.

All models are trained for six epochs on a combination of three tasks: CoT solution generation, TIR solution generation, and GenSelect, where the task is to select one correct solution out of multiple candidates. Each task is defined by a unique prompt that we can use at inference time to switch between different generation modes (see prompts in Appendix F). We found that training on a mix of all tasks results in a similar accuracy compared to training on each task sequentially (first CoT, then TIR, then GenSelect). The total SFT dataset size is 5.5M samples (3.2M CoT, 1.7M TIR, and 566K GenSelect).

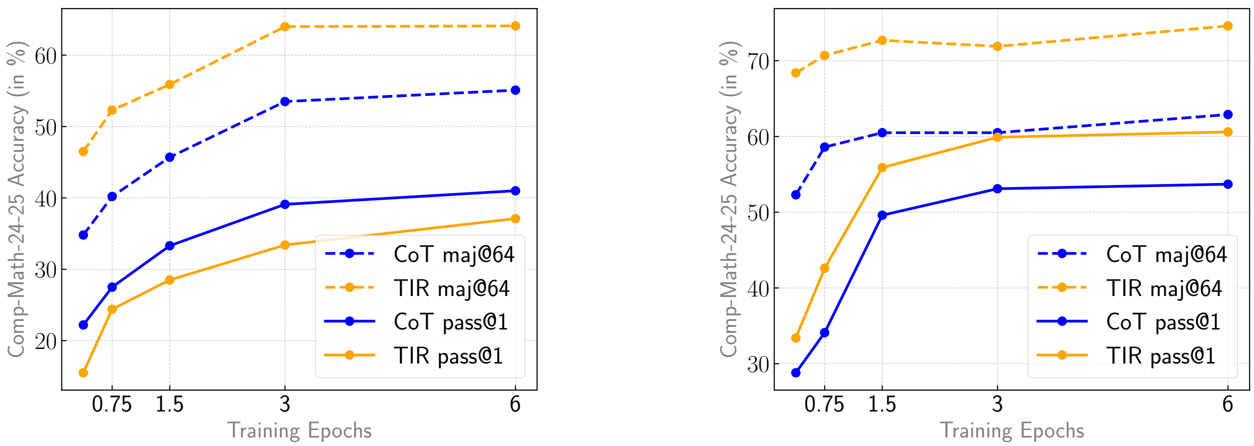
We train all models using AdamW optimizer (Loshchilov and Hutter, 2019) with weight decay of 0.01 and a cosine learning rate decay schedule with a 10% linear warmup. We use a starting learning rate of 3e-4 for 1.5B, 2e-4 for 7B, and 1e-4 for 14B and 32B models. The final learning rate is set to be 1000 times smaller. We use a batch size of 1024 samples and leverage sequence packing and context parallelization techniques from NeMo-Aligner (Shen et al., 2024) that significantly accelerate training on the long-reasoning data. Following (Toshniwal et al., 2025), we save 4 equally spaced checkpoints during the training runs, which are averaged to create the final model. We show the accuracy on the Comp-Math-24-25 benchmark (Section 2.2) of intermediate 1.5B and 14B model checkpoints in Figure 3.

After the first round of training, we perform another SFT on a subset of harder problems. These problems are selected only from forums discussing Olympiad math, and we discard any problems for which `Qwen2.5-Math-72B-Instruct` TIR model has a pass-rate bigger than 0.3 out of 32 generations. Additionally, we filter any solutions that have fewer than 5K tokens. The total SFT data size of this harder set is 2.2M samples. We follow the same setup as for the first round of SFT, except we train for four epochs instead of six. We do this second round of training for all models except 32B, as we found some degradation in results. Models' accuracy after the first and second round of training is presented in Table 3. We find that CoT results tend to significantly improve while TIR results stay stable or slightly degrade.

5.2. Results

Final evaluation results of our models are presented in Table 4. In addition to Comp-Math-24-25, introduced in Section 2.2, we also use Humanity's Last Exam dataset (et al., 2025). We only evaluate on a subset consisting of 975 text-only problems from the "Math" category. We refer to it as HLE-Math.

We notice that despite being superior in majority@k setting with TIR prompt, smaller models perform on par or even worse in pass@1, compared to CoT prompt. The results in Table 5 suggest that the reason is that with the TIR prompt there are more unfinished solutions across all model sizes, with 1.5B clearly standing out. We hypothesize that the reason behind this is that smaller models are less consistent in using tools effectively.



(a) 1.5B

(b) 14B

Figure 3: Accuracy improvement through the course of training. We observe that smaller models need to be trained for longer to achieve meaningful improvements.

6. Related Work

We briefly describe the relevant work in this section, and defer an extended discussion to Appendix J.

Math Reasoning Datasets. In the pursuit of improving mathematical reasoning in large language models, prior work has introduced several large-scale, high-quality instruction-tuning datasets. Skywork-MathQA (Zeng et al., 2024) stands out with its 2.5M question-answer pairs, generated using a trio of augmentation methods and built upon a varied set of foundational problems. NuminaMath (Li et al., 2024) consists of 860K challenging competition-style math problems, each carefully annotated with step-by-step reasoning chains (Wang et al., 2023), enabling more interpretable and structured model outputs. More recent work has focused on curating "harder" problems. BackMATH (Zhang and Xiong, 2025) is a novel dataset focused on backward reasoning. It contains approximately 14K problems specifically designed to support backward problem-solving, along with 100K detailed reasoning steps. The OpenR1-Math-220K (OpenR1 Team, 2025) consists of 220K math problems derived from NuminaMath 1.5 (LI et al., 2024), which are paired with two to four solutions generated by DeepSeek-R1. In addition, Zhao et al. (Zhao et al., 2025) presented AM-DeepSeek-R1-Distilled, a large-scale dataset featuring 1.4M question-response pairs with associated thinking traces for general reasoning tasks. Following a similar direction, Liu et al. (2025) introduced a Chinese version of the DeepSeek-R1 distilled dataset, consisting of 110K question-solution pairs. The DolphinR1 team (Team, 2025a) released a dataset of 800K samples, combining outputs from various reasoning models, including DeepSeek-R1, Gemini 2.0 Flash Thinking, and Dolphin Chat.

Generative Reward Models. Conventional reward models and verifiers are often trained as discriminative binary classifiers (Cobbe et al., 2021), underutilizing the generative strengths of large language models (LLMs). To address this, Mahan et al. (2024) introduced Generative Reward Models (GenRM), which reformulates verification as a generation task—using the log probabilities of tokens like "Yes" or "No" to represent correctness. This framing allows GenRM to better exploit LLMs' natural language generation capabilities, leading to improved alignment with human judgments across both in-distribution and out-of-distribution tasks. Concurrently, Zhang et al. (2025) introduced Generative Verifiers, training CoT-GenRM with a supervised fine-tuning (SFT) objective to serve as a verifier for mathematical reasoning. Building on a similar motivation, Ankner et al. (2024) combined Chain-of-Thought (CoT) reasoning generation with Bradley-Terry reward modeling, enabling reward models to explicitly reason about response quality before assigning scores. Extending this line of work, Wang et al. (2024b) proposed self-taught evaluators, jointly training generative models and LLM-as-a-Judge frameworks to produce both intermediate reasoning traces and final judgments. In related approaches, Wang et al. (2024a) trained large language models as generative judges by leveraging Direct Preference Optimization (DPO) on both positive and negative data, demonstrating improved evaluation performance across diverse tasks. Wu et al. (2024) introduced a Meta-Rewarding step in the self-improvement process, enabling the model to evaluate its own judgments and use the feedback to refine its evaluation capabilities.

Table 4: Evaluation results on mathematical benchmarks. All models are evaluated with a maximum of 32768 output tokens, temperature of 0.6, and top-p 0.95. We present metrics as pass@1 (maj@64) where pass@1 is the average accuracy across 64 generations and maj@64 is the result of majority voting. For HMMT and HLE-Math benchmarks, we use the LLM-as-a-judge setup described in Toshniwal et al. (2025) to verify the answers. To construct the input for GenSelect, we use subsets of 16 solutions from the set of 64 solutions. We repeat this process 64 times and perform majority voting over the answers selected by GenSelect.

Model	**Comp-Math-24-25**			**HLE-Math**
	AIME24	**AIME25**	**HMMT-24-25**	
DeepSeek-R1-Distill-Qwen-1.5B	26.8 (60.0)	21.4 (36.7)	14.2 (26.5)	2.9 (5.0)
MathReason-Qwen-1.5B CoT	61.6 (80.0)	49.5 (66.7)	39.9 (53.6)	5.4 (5.4)
MathReason-Qwen-1.5B TIR	52.0 (83.3)	39.7 (70.0)	37.2 (60.7)	2.5 (6.2)
+ Self GenSelect	83.3	70.0	62.2	7.9
+ 32B GenSelect	83.3	70.0	62.8	8.3
DeepSeek-R1-Distill-Qwen-7B	54.4 (80.0)	38.6 (53.3)	30.6 (42.9)	3.3 (5.2)
MathReason-Qwen-7B CoT	74.8 (80.0)	61.2 (76.7)	49.7 (57.7)	6.6 (6.6)
MathReason-Qwen-7B TIR	72.9 (83.3)	57.5 (76.7)	54.6 (66.3)	7.8 (10.8)
+ Self GenSelect	86.7	76.7	68.4	11.5
+ 32B GenSelect	86.7	76.7	69.9	11.9
DeepSeek-R1-Distill-Qwen-14B	65.8 (80.0)	48.4 (60.0)	40.1 (52.0)	4.2 (4.8)
MathReason-Qwen-14B CoT	76.3 (83.3)	63.0 (76.7)	52.1 (60.7)	7.5 (7.6)
MathReason-Qwen-14B TIR	76.3 (86.7)	61.3 (76.7)	58.6 (70.9)	9.5 (11.5)
+ Self GenSelect	86.7	76.7	72.4	14.1
+ 32B GenSelect	90.0	76.7	71.9	13.7
QwQ-32B	78.1 (86.7)	66.5 (76.7)	55.9 (63.3)	9.0 (9.5)
DeepSeek-R1-Distill-Qwen-32B	66.9 (83.3)	51.8 (73.3)	39.9 (51.0)	4.8 (6.0)
MathReason-Qwen-32B CoT	76.5 (86.7)	62.5 (73.3)	53.0 (59.2)	8.3 (8.3)
MathReason-Qwen-32B TIR	78.4 (93.3)	64.2 (76.7)	59.7 (70.9)	9.2 (12.5)
+ Self GenSelect	93.3	80.0	73.5	15.7
DeepSeek-R1	79.1 (86.7)	64.3 (73.3)	53.0 (59.2)	10.5 (11.4)

Table 5: Percentage of unfinished solutions on the Comp-Math-24-25 dataset. We generate 32k tokens and consider solution unfinished if it does not contain "\boxed".

Model	**Prompt**	**Unfinished (in %)**
1.5B		2.23
7B	CoT	0.98
14B		1.13
1.5B		40.31
7B	TIR	6.45
14B		4.06

7. Conclusion

We present a pipeline for developing state-of-the-art mathematical reasoning models. Our contributions can be summarized as follows:

- We develop a method to combine code execution with long chain-of-thought (CoT) generations to produce tool-integrated reasoning (TIR) solutions.
- We create a pipeline for training models to generate samples that select the most promising solution from multiple candidates (GenSelect).
- We release a large-scale `MathReason` dataset. It contains 540K unique mathematical problems, 3.2M long chain-of-thought (CoT) solutions, 1.7M long tool-integrated reasoning (TIR) solutions, and 566K generative solution selection (GenSelect) traces.
- We release a series of `MathReason-Qwen` models capable of operating in CoT, TIR, or GenSelect inference modes. With this release, we establish a new state-of-the-art in mathematical reasoning among open-weight models.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning by improving the mathematical reasoning capabilities of AI models. While the primary impact of our contribution is to foster further research, we acknowledge that any application of this technology in sensitive domains, such as for educational purposes, carries risks and requires significant validation and human oversight. The long-reasoning paradigm is also computationally intensive, which may impact accessibility. Beyond these considerations, we do not feel there are additional societal consequences of our work that must be specifically highlighted here.

References

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D Chang, and Prithviraj Ammanabrolu. Critique-out-Loud Reward Models. *arXiv preprint arXiv:2408.11791*, 2024.

bloc97. NTK-Aware Scaled RoPE allows LLaMA models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation. https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/, 2023.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *TMLR*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

Long Phan et al. Humanity's last exam, 2025. URL https://arxiv.org/abs/2501.14249.

Silin Gao, Jane Dwivedi-Yu, Ping Yu, Xiaoqing Ellen Tan, Ramakanth Pasunuru, Olga Golovneva, Koustuv Sinha, Asli Celikyilmaz, Antoine Bosselut, and Tianlu Wang. Efficient Tool Use with Chain-of-Abstraction Reasoning. *arXiv preprint arXiv:2401.17464*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.

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. In *NeurIPS Datasets and Benchmarks*, 2021.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. NuminaMath: The largest public dataset in AI4Maths with 860k pairs of competition math problems and solutions, 2024.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. [https://huggingface.co/datasets/AI-MO/NuminaMath-1.5](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. ToRL: Scaling Tool-Integrated RL. *arXiv preprint arXiv:2503.23383*, 2025.

Minpeng Liao, Chengxi Li, Wei Luo, Wu Jing, and Kai Fan. MARIO: MAth Reasoning with code Interpreter Output - A Reproducible Pipeline. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of ACL*, 2024.

Cong Liu, Zhong Wang, ShengYu Shen, Jialiang Peng, Xiaoli Zhang, ZhenDong Du, and YaFang Wang. The Chinese dataset distilled from DeepSeek-R1-671b. https://huggingface.co/datasets/Congliu/Chinese-DeepSeek-R1-Distill-data-110k, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019.

Dakota Mahan, Duy Van Phung, Rafael Rafailov, and Chase Blagden1 Nathan Lile1 Louis Castricato. Generative Reward Models. *arXiv preprint arXiv:2410.12832*, 2024.

Sadegh Mahdavi, Muchen Li, Kaiwen Liu, Christos Thrampoulidis, Leonid Sigal, and Renjie Liao. Leveraging Online Olympiad-Level Math Problems for LLMs Training and Contamination-Resistant Evaluation. *arXiv preprint arXiv:2501.14275*, 2025.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori

Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

OpenR1 Team. OpenR1 Math 220k, February 2025. URL https://huggingface.co/datasets/open-r1/OpenR1-Math-220k. Dataset available on Hugging Face.

Gerald Shen, Zhilin Wang, Olivier Delalleau, Jiaqi Zeng, Yi Dong, Daniel Egert, Shengyang Sun, Jimmy Zhang, Sahil Jain, Ali Taghibakhshi, et al. NeMo-Aligner: Scalable Toolkit for Efficient Model Alignment. *arXiv preprint arXiv:2405.01481*, 2024.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv preprint arXiv:2104.09864*, 2021.

DolphinR1 Team. Dolphin r1. https://huggingface.co/datasets/cognitivecomputations/dolphinr1, February 2025a. Accessed April 2025.

Qwen Team. QwQ-32B: Embracing the Power of Reinforcement Learning, March 2025b. URL https://qwenlm.github.io/blog/qwq-32b/.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. OpenMathInstruct-1: A 1.8 Million Math Instruction Tuning Dataset. In *NeurIPS Datasets and Benchmarks*, 2024.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. OpenMathInstruct-2: Accelerating AI for Math with Massive Open-Source Instruction Data. In *ICLR*, 2025.

Peifeng Wang, Austin Xu, Yilun Zhou, Caiming Xiong, and Shafiq Joty. Direct Judgement Preference Optimization. *arXiv preprint 2409.14664*, 2024a.

Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. Self-Taught Evaluators. *arXiv preprint arXiv:2408.02666*, 2024b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *ICLR*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.

Junde Wu, Jiayuan Zhu, and Yuyuan Liu. Agentic Reasoning: Reasoning LLMs with Tools for the Deep Research. *arXiv preprint arXiv:2502.04644*, 2025.

Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. Meta-Rewarding Language Models: Self-Improving Alignment with LLM-as-a-Meta-Judge. *arXiv preprint arXiv:2407.19594*, 2024.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, et al. Building math agents with multi-turn iterative preference learning. *arXiv preprint arXiv:2409.02392*, 2024.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report, 2025. URL https://arxiv.org/abs/2412.15115.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. Rethinking Benchmark and Contamination for Language Models with Rephrased Samples, 2023.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. LIMO: Less is More for Reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

Liang Zeng, Liangjun Zhong, Liang Zhao, Tianwen Wei, Liu Yang, Jujie He, Cheng Cheng, Rui Hu, Yang Liu, Shuicheng Yan, Han Fang, and Yahui Zhou. Skywork-Math: Data Scaling Laws for Mathematical Reasoning in Large Language Models – The Story Goes On, 2024.

Bo-Wen Zhang, Yan Yan, Lin Li, and Guang Liu. InfinityMATH: A Scalable Instruction Tuning Dataset in Programmatic Mathematical Reasoning. *arXiv preprint arXiv:2408.07089*, 2024.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative

Verifiers: Reward Modeling as Next-Token Prediction. In *ICLR*, 2025.

Shaowei Zhang and Deyi Xiong. BackMATH: Towards Backward Reasoning for Solving Math Problems Step by Step. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 466–482, 2025.

Han Zhao, Haotian Wang, Yiping Peng, Sitong Zhao, Xiaoyu Tian, Shuaiting Chen, Yunjie Ji, and Xiangang Li. 1.4 Million Open-Source Distilled Reasoning Dataset to Empower Large Language Model Training. *arXiv preprint arXiv:2503.19633*, 2025.

A. Data Pipeline

1. **Problem Extraction:** We prompt an LLM to identify and extract all problems from the initial forum posts (Appendix D.7). While most posts contain a single problem, some include multiple problems or none at all.

2. **Problem Classification:** Each extracted problem is classified into the following categories. We use an LLM to perform the classification:

   - Proof problem or not (Appendix D.4)
   - Multiple choice question or not (Appendix D.3)
   - Binary question (yes-or-no answer) or not (Appendix D.1)
   - Valid problem or not (Appendix D.2)[2]

   We remove all multiple-choice questions, binary questions, and invalid problems from the final dataset.

3. **Question Transformation:** For proof questions, we convert them into answer-based questions that require similar problem-solving techniques (Appendix D.5).

4. **Answer Extraction:** For non-proof questions, we attempt to extract the final answer from the forum discussions (Appendix D.6)[3].

5. **Benchmark Decontamination:** Following (Yang et al., 2023) we use an LLM-based comparison to remove questions that closely resemble those in popular math benchmarks.

Table 6 has a breakdown of the dataset size after each processing stage, and Table 7 shows the final dataset composition.

B. Comp-Math-24-25 dataset

Table 8: Composition of our Comp-Math-24-25 validation dataset.

Problem source	**# of Problems**
AIME 2024	30
AIME 2025	30
HMMT Nov 2024	62
HMMT Feb 2024	68
HMMT Feb 2025	66
Total	256

[2]E.g. problems that are lacking context or referring to other problems are considered invalid.

[3]We do not try to extract the full solution, just the final answer.

Pipeline Stage	Data Size
Original forum discussions	620K
Extracted problems	580K
Removing "bad" problems	550K
Benchmark decontamination	540K

Table 6: Dataset size after each processing stage.

Subset	Size
Converted proofs	260K
With extracted answer	190K
No extracted answer	90K
Total problems	540K

Table 7: Final dataset composition.

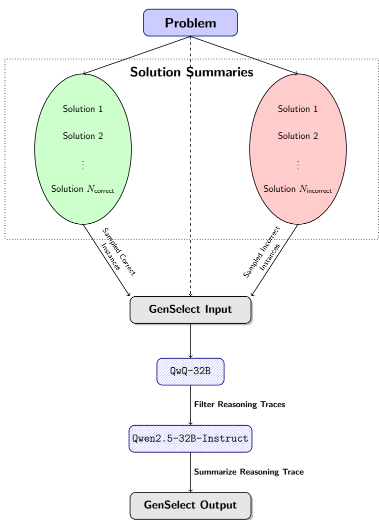


Figure 4: Data construction pipeline of GenSelect. The GenSelect input is constructed by sampling solution summaries of both correct and incorrect instances, ensuring that the input contains at least one correct and one incorrect solution. The input is then fed to `QwQ-32B`, which is tasked with selecting the best solution among the candidate solutions. The reasoning traces that select correct solutions are summarized with `Qwen2.5-32B-Instruct`, which forms the GenSelect output.

C. GenSelect Pipeline

Figure 4 illustrates the GenSelect training data synthesis pipeline.

D. Problem Preparation Prompts

D.1. Binary Problem Classification

Prompt: Binary Problem Classification

```
I will provide a math problem, and you need to determine whether it is a binary
question.
Respond only with 'binary' if the problem meets the criteria, and 'not binary'
otherwise.

A problem qualifies as a binary question if and only if:

1. The problem explicitly asks for a binary response, such as "yes or no", "true or
false", or another equivalent two-choice response.
2. The problem is phrased as a question or statement that naturally leads to a
binary response (e.g., "Is this true?" or "Determine whether the statement is true
or false").

If the problem does not explicitly ask for a binary response, even if it can be
interpreted that way, it should be classified as 'not binary question'.


Here are a few examples.

Example 1

Problem:
Is it true that $0.4395308999999\\ldots = 0.4395309$?

Output: binary



Example 2

Problem:
Write first several terms of a geometric progression in which the difference between
 the third and first terms is equal to 9, and that between the fifth and third terms
 equal 36.

Output: not binary



Example 3

Problem:
Solve the following equations: $\frac{{\sin(60^\circ+x)+\sin(60^\circ-x)}}{{2}} =\
frac{{\tan x}}{{(1+\tan^2 x)^2}}+\frac{{\cot x}}{{(1+\cot^2 x)^2}}$

Output: not binary




Example 4

Problem:
Given the quadratic expression \\( ax^2 + bx + c \\) with coefficients \\( a, b, c
\\) such that \\( b - c > a \\) and \\( a \\neq 0 \\), is it true that the equation
\\( ax^2 + bx + c = 0 \\) always has two distinct real roots?
```

```
Output: binary



Example 5:
Problem:
Can the vertices of a cube be colored in red, yellow, and blue such that every set
of four coplanar vertices contains all three colors?

Output: binary



Example 6:
Problem:
Can the numbers \\(\\frac{{14x + 5}}{{9}}\\) and \\(\\frac{{17x - 4}}{{12}}\\) both
be integers for some integer \\(x\\)? If so, find that integer.


Output: not binary


Example 7:
Problem:
Can the distances from a point on the plane to the vertices of a certain square be
equal to $1, 1, 2,$ and $3$?


Output: binary




Now here is the problem you need to extract the answer from.


Problem:
{problem}

Output:
```

D.2. Valid Problem Classification

Prompt: Valid Problem Classification

```
I will provide a problem statement from a math forum. Your task is to determine
whether it is a valid, solvable math problem based on the given text.

Respond with 'not invalid' if the problem meets all of the following conditions:

1. It is a well-defined math question, such as solving an equation, finding a
minimum, computing an expression, or proving a result.
2. It contains enough information to be solved using standard mathematical
techniques, even if the solution requires advanced concepts (e.g., limits,
logarithms, recursion).
3. It is not just a conceptual or definitional question (e.g., "What does the
notation  mean?" is not a valid math problem).
4. It does not rely on external resources such as images or missing context.
```

```
Otherwise, respond with 'invalid', but only if there is a clear and strong reason
why the problem cannot be solved. If you are uncertain, default to 'not invalid'.

Important Notes:

1. The vast majority (>99%) of problems will be valid math problems.

2. Only extremely rare cases are invalid, such as: Problems relying on external
images or missing definitions. Vague or incomplete statements that cannot be
interpreted mathematically. Open-ended conceptual discussions rather than problem-
solving.

3. A problem is still valid even if solving it requires advanced methods like
recursion, limits, or logarithms.

4. Do not evaluate whether the problem has a solution or not.

5. Do not analyze the difficulty of the problem or the methods required to solve it.

6. Only check whether it is a well-formed math problem that can be meaningfully
interpreted.



Here are a few examples.

Example 1

Problem:
Solve the equation \\(\\log(x - 2)(2x - 3) = \\log(x^2)\\).

Output: not invalid



Example 2

Problem:
Solve the math problem found on Facebook (image provided)

Output: invalid



Example 3

Problem:
Solve the following equations: $\frac{{\sin(60^\circ+x)+\sin(60^\circ-x)}}{{2}} =\
frac{{\tan x}}{{(1+\tan^2 x)^2}}+\frac{{\cot x}}{{(1+\cot^2 x)^2}}$

Output: not invalid



Example 4

Problem:
Find the area of square T?

Output: invalid
```

```
Example 5:

Problem:
Provide another example of a similar problem involving remainders and squaring a
number.

Output: invalid



Example 6:
Problem:
What does the notation $\\vec{{B}}$ mean in the context of vectors?

Output: invalid


Example 7:
Problem:
Is there a quick way to multiply 59 and 61? If so, explain the method

Output: invalid



Example 8:
Problem:
None\n\n  (Note: There is only one problem in the given forum post.)

Output: invalid



Example 9:
Problem:
If $a+b=31$ and $ab=240$, find the sum of the reciprocals of $a$ and $b$.

Output: not invalid


Example 10:
Problem:
What is the value of $35461^54593428$ \\mod 11$?

Output: not invalid


Now here is the problem you need to extract the answer from.


Problem:
{problem}

Output:
```

D.3. Multiple Choice Problem Classification

Prompt: Multiple Choice Problem Classification

```
I will provide a math problem, and you need to determine whether it is a multiple-
choice problem.
Respond only with 'mcq' if the problem meets the criteria, and 'not mcq' otherwise.


A multiple-choice problem must satisfy all of the following conditions:

1. The problem explicitly presents a set of answer choices to select from.
2. The problem asks for a final answer rather than requiring a proof, justification,
 or explanation.
3. The problem has at least one correct answer among the given choices.
If the problem does not include answer choices, even if it has a numerical answer,
it should be classified as 'not mcq'.


Here are a few examples.

Example 1

Problem:
Simplify the expression \\(\\frac{{{{2}} \\sqrt{{6}}}}{{\\sqrt{{2}}}} + \\sqrt{{3}}
+ \\sqrt{{5}}\\) and choose the correct option from the following:\n\n
A. \\(\\sqrt{{2}} + \\sqrt{{3}} - \\sqrt{{5}}\\)\n\n
B. \\(4 - \\sqrt{{2}} - \\sqrt{{3}}\\)\n\n
C. \\(\\sqrt{{2}} + \\sqrt{{3}} + \\sqrt{{6}} - 5\\)\n\n
D. \\(\\frac{{1}}{{2}} (\\sqrt{{2}} + \\sqrt{{5}} - \\sqrt{{3}})\\)\n\n
E. \\(\\frac{{1}}{{3}} (\\sqrt{{3}} + \\sqrt{{5}} - \\sqrt{{2}})\\)

Output: mcq



Example 2

Problem:
Write first several terms of a geometric progression in which the difference between
 the third and first terms is equal to 9, and that between the fifth and third terms
 equal 36.

Output: not mcq



Example 3

Problem:
Solve the following equations: $\frac{{\sin(60^\circ+x)+\sin(60^\circ-x)}}{{2}} =\
frac{{\tan x}}{{(1+\tan^2 x)^2}}+\frac{{\cot x}}{{(1+\cot^2 x)^2}}$

Output: not mcq




Example 4

Problem:
```

```
Simplify the expression \\(\\log{{\\frac{{a}}{{b}}}} + \\log{{\\frac{{b}}{{c}}}} +
\\log{{\\frac{{c}}{{d}}}} - \\log{{\\frac{{ay}}{{dx}}}}\\).
Choose from the following options:\n\\[\n
\\textbf{{(A)}}\\ \\log{{\\frac{{y}}{{x}}}}\\qquad
\\textbf{{(B)}}\\ \\log{{\\frac{{x}}{{y}}}}\\qquad
\\textbf{{(C)}}\\ 1\\qquad
\\textbf{{(D)}}\\ 0\\qquad
\\textbf{{(E)}}\\ \\log{{\\frac{{a^2y}}{{d^2x}}}}\n\\]

Output: mcq



Example 5:
Problem:
What is the maximum possible magnitude of the difference between two vectors? Choose
 from the following options and provide reasoning:
A. The magnitude of one of the vectors.
B. The magnitude of both vectors.
C. The magnitude of their sum.
D. Their scalar product.

Output: mcq



Example 6:
Problem:
Compare the numbers $a$ and $b$: $a=3(\log 7-\log 5),\ b=2\left(\frac{{1}}{{2}}\log
9-\frac{{1}}{{3}}\log 8\right)$

Output: not mcq


Example 7:
Problem:
Which of the two numbers $31^{{11}}$ and $17^{{14}}$ is greater?

Output: not mcq


Example 8:
Problem:
Let $ABCD$ be a rectangle and $E$ the reflection of $A$ with respect to the diagonal
 $BD$. If $EB = EC$, what is the ratio $\frac{{AD}}{{AB}}

Output: not mcq



Now here is the problem you need to extract the answer from.


Problem:
{problem}

Output:
```

D.4. Proof Problem Classification

Prompt: Proof Problem Classification

```
I will give you a math problem and ask to identify if it's a "proof" problem.
Respond only with "proof" if it is a proof problem, and "not proof" if it is not.

Consider the following characteristics of proof problems:

1. They often use phrases like "prove that", "show that", or "demonstrate that".
2. They may ask to justify or explain why a statement is true.
3. They don't have a well-defined answer in the form of a number or expression.

Here are a few examples.

Example 1

Problem:
Prove the identity $a^\frac{{1}}{{2}}-\frac{{a-a^{{-2}}{{a^\frac{{1}}{{2}}-a^{{-\
frac{{1}}{{2}} +\frac{{1-a^{{-2}}{{a^\frac{{1}}{{2}}+a^{{-\frac{{1}}{{2}}+\frac
{{2}}{{a^\frac{{3}}{{2}}=0$

Output: proof



Example 2

Problem:
Write first several terms of a geometric progression in which the difference between
 the third and first terms is equal to 9, and that between the fifth and third terms
 equal 36.

Output: not proof



Example 3

Problem:
Solve the following equations: $\frac{{\sin(60^\circ+x)+\sin(60^\circ-x)}}{{2}} =\
frac{{\tan x}}{{(1+\tan^2 x)^2}}+\frac{{\cot x}}{{(1+\cot^2 x)^2}}$

Output: not proof



Example 4

Problem:
Denoting the sums of the first $n_1$, first $n_2$ and first $n_3$ terms of an
arithmetic progression by $S_1$, $S_2$ and $S_3$, respectively, show that $$\frac{{
S_1}}{{n_1}}(n_2-n_3)+\frac{{S_2}}{{n_2}}(n_3-n_1)+\frac{{S_3}}{{n_3}}(n_1-n_2)=0.$$

Output: proof




Now here is the problem you need to extract the answer from.


Problem:
```

```
{problem}

Output:
```

D.5. Proof Problem Conversion

Prompt: Proof Problem Conversion

```
I will give you a math problem that asks to prove something.
Your task is to create an equivalent problem that instead has some kind of numerical
 or expression answer that can be used to automatically grade the solution.
Make sure the new problem is at least as difficult as the original proof problem.

Here are a few examples.

Example 1

Problem:
Prove that the system \begin{{align*}}x^6 + x^3 + x^3y + y &= 147^{{157}} \\ x^3 + x
^3y + y^2 + y + z^9 &= 157^{{147}}\end{{align*}} has no solutions in integers $x$ ,
$y$ , and $z$ .

Output:
Let $x$, $y$ and $z$ be a solution to the following system of equations \begin{{
align*}}x^6 + x^3 + x^3y + y &= 147^{{157}} \\ x^3 + x^3y + y^2 + y + z^9 &=
157^{{147}}\end{{align*}}.
Calculate the sum of all possible values of $x$.



Example 2

Problem:
A triangle is called a parabolic triangle if its vertices lie on a
parabola $y = x^2$ . Prove that for every nonnegative integer $n$ , there
is an odd number $m$ and a parabolic triangle with vertices at three
distinct points with integer coordinates with area $(2^nm)^2$ .

Output:
Consider parabolic triangles whose vertices lie on $y = x^2$ with integer
coordinates. Let $f(n)$ be the smallest possible value of $c$, where $(0,0)$, $(b,b
^2)$, and $(c,c^2)$ are vertices of such a triangle with area exactly $(2^n)^2$, for
 some integer b where 0 < b < c.
Find $f(4)$.



Now here is the problem you need to modify. Only output the new problem **WITH NO**
explanation or notes after it.
Again, start with the problem right away, **DO NOT** start with "Let's modify the
given problem" or anything like that.

Problem:
{problem}

Output:
```

D.6. Forum Answer Extraction

Prompt: Forum Answer Extraction

```
I will give you a series of posts from a math-related forum that contain one or
several math problems and discussion of their solutions.
I will also specify which problem I'm currently looking at (in case there are
multiple).
You task is to find an answer to the problem I'm currently looking at inside the
forum discussions.
The answer should be a numerical value or a mathematical expression.
If the answer is not available, output "Answer not found." in the last line of your
response.
You can think before stating the final answer. The final line of your response
should be "Answer: <final answer>".

Here is an example.

 First forum post with problem(s):
This problem was extra credit for my math class and I haven't gotten it back yet but
 I'm assuming
a.) Everyone handed it in
and
b.) None of you here goes/takes/will go/take my math class

Anyways:
Suppose two of the zeroes of the following fourth-degree equation are the same and
the other two zeroes are the recipricals of each other. Find a and b.

x:^4:+ax:^3:+bx:^2:+4x+4=0


It's not at all hard as it looks...a lot of work though, so I suggest orginizing as
you go along.

Problem we are looking at (it might be rephrased):
Suppose two of the zeroes of the fourth-degree equation \( x^4 + ax^3 + bx^2 + 4x +
4 = 0 \) are the same and the other two zeroes are reciprocals of each other. Find
\( a \) and \( b \).

Forum discussions:
Post 1:
Tare wrote:x:^4:+ax:^3:+bx:^2:+4x+4=0




Here's a shorter way:

[hide]Say the four roots are c, c, d, and 1/d.  Then the product of the four roots
is the constant term of the polynomial, so c^2=4.  Then c= :pm: 2.  Similarly, from
the linear term, c^2d+c^2/d+c+c=-4.  If we plug in c=2, we get d=-1,  so the roots
are 2, 2, -1, -1.  So a = -(2+2-1-1)=-2 and b = 2*2+2(-1)+2(-1)+2(-1)+2(-1)+(-1)(-1)
=-3.  If we plug in c=-2, we get 4d+4/d=0, so d+1/d=0.  Then a = -(-2-2+0)=4 and b
=(-2)(-2)+(-2)(0)+(-2)0+1=5.  So either a=-2, b=-3 or a=4,b=5.

[/hide]

Thanks Tare for catching the mistakes.



--Dan
```

```
Post 2:
Well...it didn't specify that the solution is real and also you were supposed to get
 a and b...

Output:
Seems that there is an answer at the end of the first post. Since none of the other
posts contradicts it, we can assume that the answer is correct.
Answer: a = -2, b = -3 or a = 4, b = 5



Now here are the posts from the forum that I'm currently looking at. Please find the
 answer to the problem.
Don't forget to say "Answer not found." if the answer is not available.

First forum post with problem(s):
{forum_post}

Problem we are looking at (it might be rephrased):
{problem}

Forum discussions:
{forum_discussions}

Output:
```

D.7. Forum Problem Extraction

Prompt: Forum Problem Extraction

```
I will give you a post from a math-related forum that might contain one or several
math problems.
Your task is to extract all problems or state that none are available.

Here are some guidelines you should follow
- If no problems are available, output "No problems identified."
- For each problem found, use the following format:
Problem 1: <problem statement>

Problem 2: <problem statement>

...

- For each math problem you identify, make sure to rephrase it such that it's stated
 clearly and concisely.
Remove any redundant context, personal commentary, anecdotes, or unrelated
information.
But make sure not to change the meaning of the problem and keep all necessary
mathematical or technical details.
- If multiple problems that you extract are related, make sure to include all the
context in each problem statement
as they will be looked at independently.

Here are a few examples.


Example 1

Forum post:
```

```
Countdown:

What is the remainder of 8^6+7^7+6^8 is divided by 5?

no calculator of course, paper isn't needed either, but sure.

Output:
Problem 1: What is the remainder of $8^6+7^7+6^8$ when divided by 5?



Example 2

Forum post:
Question 1:
A tetrahedron has four vertices. We can label each vertex by one of the four digits:
 $1, 2, 3, 4$. How many non-congruent ways are there to assign a different digit to
each vertex of a tetrahedron? Tetrahedra are considered congruent through rotation.
Reflections are considered different.

I'm wondering how I could approach a problem like this. I started off with $4!$ and
then divided by $4$ to take out the rotation aspect. Now I am stuck.

Note: I'd rather not do case work because I'm sure the test writers could have
easily used an icosahedron, or something equally lengthy.

Another Question along the same lines:

How many ways to color a cube using 6 colors, where each face has a unique color?

Thanks

Output:
Problem 1: How many non-congruent ways are there to assign a different digit to each
 vertex of a tetrahedron? Tetrahedra are considered congruent through rotation.
Reflections are considered different.
Problem 2: How many ways can a cube be colored using 6 colors, where each face has a
 unique color?



Example 3

Forum post:
Yes! I completely agree with what you said. It's been a tough transition for me too,
 but we'll figure it out.

Output:
No problems identified



Example 4

Forum post:
Billy Bob has fourteen different pairs of socks in his drawer. They are just thrown
around randomly in the drawer. Billy Bob once woke up in a hurry and had to get his
socks quickly.
Without switching the light on, he pulled out enough socks to know that he had at
least one pair, and then he ran out of the room. How many socks did Billy Bob pull
out
```

```
Output:
Problem 1: From a drawer containing 14 different pairs of socks, how many socks must
 be pulled out randomly to ensure at least one matching pair?



Please analyze the following forum post and extract all math problems. Here are the
guidelines one more time for your reference
- If no problems are available, output "No problems identified."
- For each problem found, use the following format:
Problem 1: <problem statement>

Problem 2: <problem statement>

...

- For each math problem you identify, make sure to rephrase it such that it's stated
 clearly and concisely.
Remove any redundant context, personal commentary, anecdotes, or unrelated
information.
But make sure not to change the meaning of the problem and keep all necessary
mathematical or technical details.
- If multiple problems that you extract are related, make sure to include all the
context in each problem statement
as they will be looked at independently.


Forum post:
{forum_post}

Output:
```

E. TIR Data Generation Prompts

E.1. Stage-0 TIR Data Generation Prompt

TIR Inference Prompt for Stage-0 Data Generation

````
You are a math problem solver that uses Python code as an integral part of your
reasoning.
In your solution you MUST strictly follow these instructions:
1. For each step requiring complex calculation write Python code.
2. For Python code use the following template:
```python
# Your Python code
```
3. Put the final answer within \boxed{{}}.

Please reason step by step, and put your final answer within \\boxed{{}}.

user: |-
Solve the following math problem using Python code for the calculations.

{problem}
````

E.2. TIR Novelty Evaluation

Prompt to evaluate TIR novelty

````
You will be given a fragment of a solution to a math problem that includes a Python
code block.
Your task is to determine the purpose of this Python code block in the solution
fragment.
In your assessment, you MUST follow these guidelines:

1. Classification:
- Verification: Python code is used to verify the correctness of the previous manual
 calculations or to confirm some results. E.g. if the result of the code execution
exists in the solution above, it is definitely a verification.
- Novel Calculation: Otherwise, if the result of code execution is not present in
ANY FORM in the solution above, it is a novel calculation.

If you are unsure about the classification of specific code block, you MUST label it
 as Verification!

2. Output Format:
- Your response MUST follow this exact format (without extra commentary or text):
    ```
    Reasoning: <a couple of sentences explaining your rationale>
    Judgement: <Verification or Novel Calculation>
    ```

---

**EXAMPLES**

1.
"""
Solution:
<Some text reasoning without code>

Wait, so the answer is 143? Let me verify this with the pow function.

```python
# Compute 7^999 mod 1000 using pow function
print(pow(7, 999, 1000))  # Should print 143
```
```output
143
```
So the answer is \\boxed{{143}}.
"""
```
Reasoning: This is for sure a verification, because the result of the code execution
 is present in the solution above. Moreover, comment in the code block explicitly
states that it should print 143 which means that the result is known in advance.
Judgement: Verification
```
---

2.
"""
Solution:
<Some text reasoning without code>

Therefore, let's proceed to compute P^5. I can use Python for this calculation to
ensure accuracy.
````

```
First, let's define the transition matrix P as a numpy array, then compute P^5, then
 extract the relevant entry, and divide by 3.

Let me import numpy and do the calculations.
```python
import numpy as np

# Define the transition matrix P
P = np.array([
    [0, 1, 0, 0],
    [1/3, 0, 2/3, 0],
    [0, 2/3, 0, 1/3],
    [0, 0, 1, 0]
])

# Compute P^5
P5 = np.linalg.matrix_power(P, 5)

# The initial state is state 0, so the distribution after 5 steps is [1, 0, 0, 0] @
P5
# But since P5 is the transition matrix after 5 steps, the entry (0,1) is the
probability of going from 0 to 1 in 5 steps.
# However, to get the distribution, we need to multiply the initial distribution by
P5.
initial_distribution = np.array([1, 0, 0, 0])
distribution_after_5_steps = initial_distribution @ P5

# The probability mass at state 1 (distance 1) after 5 steps
prob_mass_at_1 = distribution_after_5_steps[1]

# Since state 1 corresponds to 3 vertices (B, D, E), the probability of being at B
is this mass divided by 3
prob_at_B = prob_mass_at_1 / 3

prob_at_B
```
```output
np.float64(0.25102880658436205)
```
...
"""
```
Reasoning: The solution fragment describes algorithmic steps to calculate the
probability and the code block executes these steps. The result of the code
execution is not present in the solution above in any form. Therefore, this is a
novel calculation.
Judgement: Novel Calculation
```

3.
"""
Solution:
<Some text reasoning without code>

Compute C(51, 5):

51! / (5! * 46!) = ?

But maybe I should calculate it using Python to be accurate.
```python
import math
```
```

````
math.comb(51, 5)
```
```output
2349060
```
...
"""
```
Reasoning: The solution fragment describes the calculation of a combinatorial
expression and the code block executes this calculation. The result of the code
execution is not present in the solution above in any form. Therefore, this is a
novel calculation.
Judgement: Novel Calculation
```
---

4.
"""
Solution:
<Some text reasoning without code>

But let's compute these values in Python.
```python
import math

# Given dimensions
R = 4  # feet
H = 12  # feet
h = 9  # feet from the tip, so remaining height
r = (h / H) * R  # since r/R = h/H

# Original volume
V_original = (1/3) * math.pi * R**2 * H

# Remaining volume
V_remaining = (1/3) * math.pi * r**2 * h

# Volume poured out
V_poured = V_original - V_remaining

V_poured
```
```output
116.23892818282235
```
When I computed the volume manually, I obtained \( 37\pi \) cubic feet.
Approximating this as $$
37 * 3.14159 \approx 116.23
$$, it closely matches the Python result of approximately 116.2389. Therefore, the
result appears to be correct.

...
"""
```
Reasoning: The rationale right after the code block states that the manual
calculation (that happened before the code block) matches the Python result.
Therefore, code block verifies the previous manual calculations. So, this is a
verification.
Judgement: Verification
```
---

**REMINDER**
````
````

````
Focus only on the Python code block in the provided fragment and classify it as
either Verification or Novel Calculation based on whether its output appears in the
solution text before the code.
---

**YOUR TASK**

Solution fragment: {fragment}
````

E.3. TIR Significance Evaluation

Prompt to evaluate TIR significance

````
You will be given a fragment of a solution to a math problem that includes a Python
code block.
Your task is to evaluate the significance of this Python code in solving the math
problem.
In your assessment, you MUST follow these guidelines:

1. Classification:
Evaluate the significance of the code's contribution by categorizing it into one of
three levels:

- Trivial: The code performs calculations that could easily be done manually without
 significant effort (e.g., solving simple equations, doing arithmetic, applying
formulas to known variables). The code usage provides no meaningful or minor
advantage over manual calculation.

- Moderate: The code performs calculations that would be tedious, error-prone, or
time-consuming to do manually, but still technically possible (e.g., matrix
operations, numerical integration of standard functions, solving systems of
equations). The code usage provides efficiency but isn't essential.

- Significant: The code performs calculations that would be practically impossible
or extremely difficult to do manually (e.g., brute-forcing combinatorial problems,
complex simulations, solving complex differential equations, high-dimensional
optimization). The code usage creates a crucial shortcut that fundamentally enables
the solution.

2. Output Format:
- Your response MUST follow this exact format (without extra commentary or text):
    ```
    Reasoning: <a couple of sentences explaining your rationale>
    Significance: <Trivial, Moderate, or Significant>
    ```

---

**EXAMPLES**

1.
"""
Let's find the roots of the quadratic equation: 3x^2 - 5x + 2 = 0

```python
import numpy as np
from sympy import symbols, solve, Eq

x = symbols('x')
equation = 3*x**2 - 5*x + 2
````
````

````
solutions = solve(equation, x)
print(solutions)
```
```output
[2/3, 1]
```

So the solutions are x = 2/3 and x = 1.
"""
```
Reasoning: This code simply solves a basic quadratic equation that could easily be
solved manually using the quadratic formula or factoring. Finding roots of a
quadratic equation with small integer coefficients is a standard calculation that
requires minimal effort by hand.
Significance: Trivial
```
---

2.
"""
To solve this system of 4 linear equations with 4 unknowns:
3x + 2y - z + 2w = 10
x - y + 2z - w = -1
2x + y + z + 3w = 12
x + 3y - z - w = 5

I'll use Python to solve this system using matrices.

```python
import numpy as np
from scipy import linalg

# Define coefficient matrix
A = np.array([
    [3, 2, -1, 2],
    [1, -1, 2, -1],
    [2, 1, 1, 3],
    [1, 3, -1, -1]
])

# Define constants vector
b = np.array([10, -1, 12, 5])

# Solve the system
solution = linalg.solve(A, b)
print("x =", solution[0])
print("y =", solution[1])
print("z =", solution[2])
print("w =", solution[3])
```
```output
x = 0.64
y = 2.7
z = 1.6
w = 2.14
```

Therefore, the solution is x = 0.64, y = 2.7, z = 1.6, and w = 2.14.
"""
```
Reasoning: This code solves a system of 4 linear equations with 4 unknowns. While
this could be solved manually using Gaussian elimination or Cramer's rule, it would
be tedious and error-prone. The system is complex enough that computational
````

````
assistance provides significant efficiency but doesn't enable something impossible.
Significance: Moderate
```
---

3.
"""
For this traveling salesman problem with 11 cities, where the distances between
cities are given in the distance matrix below, I need to find the shortest possible
route that visits each city exactly once and returns to the starting city.

```python
import numpy as np
from itertools import permutations
import time

# Distance matrix (11x11) between cities
distances = np.array([
    [0, 29, 82, 46, 68, 52, 72, 42, 51, 55, 29],
    [29, 0, 55, 46, 42, 43, 43, 23, 23, 31, 41],
    [82, 55, 0, 68, 46, 55, 23, 43, 41, 29, 79],
    [46, 46, 68, 0, 82, 15, 72, 31, 62, 42, 21],
    [68, 42, 46, 82, 0, 74, 23, 52, 21, 46, 82],
    [52, 43, 55, 15, 74, 0, 61, 23, 55, 31, 33],
    [72, 43, 23, 72, 23, 61, 0, 42, 23, 31, 77],
    [42, 23, 43, 31, 52, 23, 42, 0, 33, 15, 37],
    [51, 23, 41, 62, 21, 55, 23, 33, 0, 29, 62],
    [55, 31, 29, 42, 46, 31, 31, 15, 29, 0, 51],
    [29, 41, 79, 21, 82, 33, 77, 37, 62, 51, 0],
])

# Brute force approach to solve TSP
def tsp_exact(distances):
    n = len(distances)
    cities = list(range(1, n))  # Start from city 0
    min_length = float('inf')
    best_route = None

    start_time = time.time()
    count = 0

    # Try all possible permutations of cities (excluding starting city)
    for perm in permutations(cities):
        route = (0,) + perm + (0,)  # Complete route starting and ending at city 0
        length = sum(distances[route[i]][route[i+1]] for i in range(len(route)-1))

        count += 1
        if length < min_length:
            min_length = length
            best_route = route

    end_time = time.time()
    return best_route, min_length, count, end_time - start_time

# Solve the TSP problem
best_route, min_length, permutations_tried, time_taken = tsp_exact(distances)

print(f"Best route: {{best_route}}")
print(f"Minimum distance: {{min_length}}")
print(f"Permutations evaluated: {{permutations_tried:,}}")
print(f"Time taken: {{time_taken:.2f}} seconds")
```
```output
````

```
Best route: (0, 1, 8, 4, 6, 2, 9, 7, 5, 3, 10, 0)
Minimum distance: 251
Permutations evaluated: 3,628,800
Time taken: 5.77 seconds
```

Therefore, the optimal route has a total distance of 291 units.
"""
```
Reasoning: This code solves a Traveling Salesman Problem with 11 cities by evaluating over 3.6M permutations - a computation that would be absolutely impossible to do manually. The brute-force approach here creates a crucial shortcut to the solution that would be practically unattainable through manual calculation, even with significant time investment.
Significance: Significant
```

4.
"""
To find all integer solutions to the Diophantine equation 17x + 23y = 3284 where both x and y are non-negative, I'll implement search in Python.

```python
def find_solutions(a, b, c):
    solutions = []

    # Find the maximum possible value of x
    max_x = c // a

    # Check all possible values of x from 0 to max_x
    for x in range(max_x + 1):
        # Calculate the corresponding y value
        remaining = c - a * x

        # If remaining is divisible by b and the result is non-negative,
        # we have a valid solution
        if remaining >= 0 and remaining % b == 0:
            y = remaining // b
            solutions.append((x, y))

    return solutions

# Given equation: 17x + 23y = 3284
a, b, c = 17, 23, 3284
solutions = find_solutions(a, b, c)

print(f"Solutions to {{a}}x + {{b}}y = {{c}}:")
for x, y in solutions:
    print(f"x = {{x}}, y = {{y}}")
    # Verify the solution
    print(f"Verification: {{a}}*{{x}} + {{b}}*{{y}} = {{a*x + b*y}}")
    print()
```
```output
Solutions to 17x + 23y = 3284:
x = 20, y = 128
Verification: 17*20 + 23*128 = 3284

x = 43, y = 111
Verification: 17*43 + 23*111 = 3284

x = 66, y = 94
```

~~~~
Verification: 17*66 + 23*94 = 3284

x = 89, y = 77
Verification: 17*89 + 23*77 = 3284

x = 112, y = 60
Verification: 17*112 + 23*60 = 3284

x = 135, y = 43
Verification: 17*135 + 23*43 = 3284

x = 158, y = 26
Verification: 17*158 + 23*26 = 3284

x = 181, y = 9
Verification: 17*181 + 23*9 = 3284

```
So the integer solutions to the Diophantine equation are x = 11, y = 1.
"""
```
Reasoning: This code finds all integer solutions to a Diophantine equation by
iterating through possible values of x and calculating the corresponding y. While
this could be done manually, the exhaustive search for non-negative integer
solutions is tedious and error-prone. The computational approach reduces the effort
and simplifies the solution process, making it more efficient. Thus it provides a
moderate level of significance.
Significance: Moderate
```
---

5.
"""
To verify my hypothesis, I need to find the probability of getting at least 3 heads
in 10 coin flips. I'll calculate this using the binomial distribution.

```python
import math

def binomial_probability(n, k, p):
    # Calculate the probability of k successes in n trials
    # with probability p of success on a single trial
    combinations = math.comb(n, k)
    return combinations * (p ** k) * ((1-p) ** (n-k))

# Calculate P(X \geq 3) when flipping a fair coin 10 times
p_at_least_3 = sum(binomial_probability(10, k, 0.5) for k in range(3, 11))

print(f"P(X \geq 3) = {{p_at_least_3:.6f}}")
print(f"Percentage: {{p_at_least_3 * 100:.2f}}%")
```
```output
P(X \geq 3) = 0.945312
Percentage: 94.53%
```

So the probability of getting at least 3 heads in 10 coin flips is approximately
94.53%.
"""
```
Reasoning: This code calculates a probability using the binomial distribution
formula. While the calculation involves combinations and powers, the mathematical
concept is straightforward and could be calculated manually by explicitly writing
~~~~

```
and reducing the terms. The code provides a minor computational convenience but
doesn't fundamentally change the nature of the solution process, making it a trivial
 use of Python code.
Significance: Trivial
```

REMINDER
When evaluating significance, consider:
1. Could this calculation reasonably be done by hand? If yes, how difficult would it
 be?
2. Does the code enable a solution approach that would otherwise be impractical?
3. Is the computational advantage merely convenience, or is it essential to the
solution?

Remember to classify as Trivial, Moderate, or Significant based on these
considerations.

YOUR TASK

Solution fragment: {fragment}
```

# F. Prompts for Different Inference Modes

## F.1. CoT Inference

**CoT Inference Prompt**

```
Solve the following math problem. Make sure to put the answer (and only answer)
inside \boxed{{}}.

{problem}
```

F.2. TIR Inference

TIR Inference Prompt

```
Solve the following math problem, integrating natural language reasoning with Python
 code executions.
You may perform up to {total_code_executions} Python code calls to assist your
reasoning.
Make sure to put the answer (and only answer) inside \boxed{{}}.

{problem}
```

F.3. GenSelect Inference

GenSelect Inference Prompt

```
You will be given a challenging math problem followed by {num_solutions} solutions.
Your task is to systematically analyze these solutions to identify the most
mathematically sound approach.
```
```

```
Input Format:
Problem: A complex mathematical word problem at advanced high school or college
level
Solutions: Detailed solutions indexed 0-{max_idx}, each concluding with an answer in
 \boxed{{}} notation

YOUR TASK

Problem: {problem}

Solutions:
{solutions}

Evaluation Process:

1. Initial Screening
- Group solutions by their final answers
- Identify and explain mathematical contradictions between different answers
- Eliminate solutions with clear mathematical errors

2. Detailed Analysis
For remaining solutions, evaluate:
- Mathematical precision and accuracy
- Logical progression of steps
- Completeness of mathematical reasoning
- Proper use of mathematical notation, including \boxed{{}}
- Handling of edge cases or special conditions
- For solutions containing and addressing errors, evaluate the error identification
and correction methodology.

3. Solution Comparison
Compare viable solutions based on:
- Efficiency of approach
- Clarity of mathematical reasoning
- Sophistication of method
- Robustness of solution (works for all cases)

Your response should include:
1. Brief analysis of conflicting answers
2. Detailed evaluation of mathematically sound solutions
3. Justification for eliminating incorrect solutions
4. Clear explanation for selecting the best approach

End your evaluation with exactly:
Judgment: [IDX]
where IDX is the index 0-{max_idx} of the best solution.
```

G. Prompts for GenSelect Data Preparation

G.1. Re-generating Comparison Summary

Prompt to re-generate comparison summary

```
I will give you a critique's response which compares multiple solutions of a math
problem. This response will explore different approaches, making mistakes along the
way, correcting them, switching around and so on. But eventually the critique
converges on its ranking of the different solutions and ends with a string "Judgment
: " followed by a number corresponding to the index of the best solution. Your task
is to write a clean version of the final critique without all the exploration and
```

```
make sure to put the final judgment at the end. Please make sure to include the
details related to arriving at the final judgment but not the tangential details.

Problem:
{problem}


Solutions:
{solutions}


Critique:
{generation}


Now write a clean version of the final critique without all the exploration. Make
sure to put in:
- The details related to arriving at the final judgment.
- Clearly writing the final judgment at the end via "Judgment: " followed by the
index of the best solution.
```

G.2. Re-generating Solution Summary

Prompt to re-generate solution summary

```
I will give you a math problem and a long solution to that problem exploring
different approaches, making mistakes along the way, correcting them, switching
around and so on. But eventually that solution gets to the right approach and solves
 the problem. Your task is to write a clean version of the final correct solution
without all the exploration. Cover all the details of the final solution.

Problem:
{problem}

Solution:
{generation}

Now write a clean version of the final correct solution without all the exploration
but cover all the details of the final solution.
```

H. Example of Re-generated Summary

Problem

Call a 9-digit number a cassowary if it uses each of the digits 1 through 9 exactly once. Compute the number of cassowaries that are prime.

Deepseek R1 Response

<think>
Okay, so I need to figure out how many 9-digit numbers, which use each digit from 1 to 9 exactly once, are prime. These numbers are called cassowaries. Hmm, let's start by understanding the problem step by step.
First, a cassowary is a 9-digit pandigital number, meaning it uses each digit 1 through 9 once. So, numbers like 123456789, but any permutation of those digits. The question is asking how many of these are prime numbers.
Alright, so the main task is to compute the count of prime numbers among all 9-digit pandigitals. Let me think about the properties of prime numbers and pandigital numbers.
First, let's recall that a prime number is a number greater than 1 that has no positive divisors other than 1 and itself. For a number to be prime, it can't be even (except for 2), and it can't end with 5. Also, there are some divisibility rules that might help here.
Since we're dealing with 9-digit numbers using all digits 1-9, let's check the sum of the digits. The digits are 1+2+3+4+5+6+7+8+9 = 45. Wait, 45 is divisible by 9, right? So any number made with all these digits will have a digit sum of 45, which is divisible by 9. Therefore, any such number is divisible by 9. But if a number is divisible by 9, it can't be prime unless the number itself is 9, which it's not here because we're dealing with 9-digit numbers. So, does this mean that all cassowaries are divisible by 9 and hence composite?
If that's the case, then there are zero prime cassowaries. But let me verify this again. The divisibility rule for 9 is that if the sum of the digits is divisible by 9, then the number itself is divisible by 9. Since 45 is divisible by 9, every permutation of the digits 1-9 will result in a number divisible by 9. Therefore, all cassowaries are divisible by 9, which means they can't be prime. So the answer should be zero.
Wait, but let me make sure I'm not missing something. For example, could the number 9 itself be considered? But the problem specifies a 9-digit number, so 9 is a single-digit number. So all cassowaries are 9-digit numbers, which are divisible by 9, hence composite. Therefore, there are no primes in the set of cassowaries. So the answer is 0.
But just to be thorough, let me check if there's any exception. For example, is 9 a prime? No, 9 is 3 squared. So even if there was a 1-digit cassowary (which there isn't, since cassowary is defined as 9-digit), it's not prime. So, yeah, all 9-digit pandigitals using 1-9 are divisible by 9, hence composite. Therefore, the number of prime cassowaries is zero.
</think> $\boxed{0}$

Qwen2.5-32b-instruct Solution Summary

To determine the number of 9-digit prime numbers (cassowaries) that use each digit from 1 to 9 exactly once, we need to analyze the properties of these numbers.
1. **Sum of the Digits**: The sum of the digits from 1 to 9 is: $1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45$.
2. **Divisibility by 9**: A number is divisible by 9 if the sum of its digits is divisible by 9. Since the sum of the digits is 45, which is divisible by 9, every 9-digit number formed using the digits 1 through 9 will be divisible by 9.
3. **Prime Numbers and Divisibility**: A prime number has no positive divisors other than 1 and itself. Since any 9-digit number formed using the digits 1 through 9 is divisible by 9, it cannot be a prime number (except for the number 9 itself, which is not a 9-digit number).
4. **Conclusion**: Therefore, no 9-digit number that uses each digit from 1 to 9 exactly once can be a prime number. Thus, the number of prime cassowaries is: $\boxed{0}$

I. TIR Output Examples

I.1. Example of a warning of the remaining code executions

TIR solution with code execution limit warning

````
<tool_call>
# Initialize sum
total_sum = 0

# Iterate over possible bases b from 10 to 49 inclusive
for b in range(10, 50):
    divisor = b + 7
    dividend = 9 * b + 7
    if dividend % divisor == 0:
        total_sum += b

total_sum
</tool_call>
```output
70
```
```system
Remaining code executions: 3. You will not be able to call code when you run out of
executions, so use it wisely. Note that you can still continue solving the problem
without code after that.
```
````

I.2. Example of a redundant code execution

Redundant code execution that does not introduce new information to the solution

````
Since 5 is a prime number and 13 is also a prime number, their gcd should indeed be
1. Just to be thorough, let me compute it using Python.

<tool_call>

import math

# Compute gcd of 5 and 13

gcd = math.gcd(5, 13)

print(gcd)

</tool_call>

```output

1

```

The gcd is 1, so an inverse does exist. Good.
````

I.3. Example of model encountering code execution limit

Example of models reaction to reaching code execution limit

```
Let me import the necessary libraries and define the equations.

<tool_call>
import numpy as np
from scipy.optimize import fsolve

# Define the system of equations
def equations(vars):
    x, y = vars
    eq1 = 24 * np.cos(x) + 26 * np.cos(y) - 13 * np.sqrt(3) * np.sin(y) + 17.5
    eq2 = 24 * np.sin(x) + 13 * np.sqrt(3) * np.cos(y) + 26 * np.sin(y) + 7 * np.
    sqrt(3)/2
    return [eq1, eq2]

# Initial guess
initial_guess = [1, 1]

# Solve the system
solution = fsolve(equations, initial_guess)
solution
</tool_call>
```output
array([-0.93580304,  2.03210427])
```
```system
You have run out of code executions! You can no longer write or execute code. Now
you should continue solving the problem by relying on your mathematical reasoning
and analytical skills.
```
Hmm, it seems I've reached the limit of code executions. Let me check the solution
provided by the code.
```

J. Related Work

J.1. Tool Integration Reasoning

Tool-augmented approaches to mathematical problem solving have advanced rapidly in recent years. A seminal contribution by Chen et al. (2023) introduced the Program of Thoughts (PoT) framework, which integrates natural language with executable code to support step-by-step reasoning through a hybrid of textual and programmatic logic. Building on this foundation, subsequent research has focused on developing both datasets and models that facilitate tool-integrated reasoning.

On the data side, OpenMathInstruct-1 (Toshniwal et al., 2024) offers 1.8M instruction-tuning examples derived from code interpreters across benchmarks such as GSM8K and MATH. Similarly, InfinityMATH (Zhang et al., 2024) introduces 100K instances of programmatic reasoning, while MARIO (Liao et al., 2024) combines model reasoning with tool outputs, accompanied by a dataset constructed from GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). These resources have significantly enriched the training landscape for tool-augmented reasoning systems.

On the modeling side, Qwen2.5 (Yang et al., 2024) introduced a series of models with strong mathematical reasoning capabilities, supporting advanced techniques like Chain-of-Thought (CoT) and Tool-Integrated Reasoning (TIR). Gao et al. (2024) proposed a two-stage method: training large language models to generate reasoning chains, and then invoking domain-specific tools to execute each step by injecting the necessary knowledge. Xiong et al. (2024) proposed a multi-turn, online, iterative direct preference learning framework tailored to this unique context. By incorporating feedback from code interpreters during the training process, their approach achieves significant performance improvements on the MATH benchmark. Wu et al. (2025) dynamically integrate web search, code execution, and structured reasoning with contextual memory to tackle complex problems that demand deep research and multistep logical deduction. Li et al. (Li et al., 2025) developed a Tool-Integrated Reinforcement Learning framework that autonomously utilizes computational tools by scaling reinforcement learning directly from base models, and demonstrate substantial improvements compared to RL without tools.