COMP 562 Final Project - Math Router

George Harris 730568263 | **Matthew Zhu** 730679894 | **Milan Dutta** 730599453 | **Robert Lewison** 730571332

1 Introduction

In recent years, the capability of LLMs to solve complex mathematical problems has rapidly increased. However, these high-performing models often come with substantial computational costs, which can be unnecessary when the complexity of a problem does not warrant such extensive resources. To address this, our work is focused on an adaptive routing mechanism that assigns math problems to either a general-purpose solver or a more resource-intensive model based on their difficulty.

We use datasets with math problems of varying difficulty (such as MATH and HARDMath) to train a lightweight binary classifier. This classifier will then act as a decision-making proxy to decide whether a problem is reliably solvable by the simpler, cheaper model, or whether the use of a reasoning model would significantly increase the chance of an accurate response.

With the modern advent of heavy, chain of thought enabled reasoning LLM models, math is increasingly solvable using AI. But often, users choose these models superfluously when the difficulty of their problem, does not require such a costly solution. These reasoning models, enabled using reinforcement learning techniques to think about their answers for significant periods of time, use external tools, and complete reflect on their past work, are sometimes up to an order of magnitude more expensive to operate than foundation LLMs. To that end, reducing the number of calls for easy problems going to these models not only improves time to solution for users, but also reduces cost for providers.

Our work plans to use a variety of techniques from data processing, LLM calling, and logistic regression to form a consistent semantic classification of problems as solvable by our base model GPT-4.1 and GPT-4.1 mini. This comes in comparison to a reasoning-enabled model like GPT-o4, both from Google. By the end of the project, our goal is a full end to end query processing and handling system which can provide consistently correct, but cost effective solutions to math based queries.

This system is intended to be modular and extensible: it does not assume a fixed set of models or a particular type of math input. As model architectures evolve, the routing strategy can be retrained or fine-tuned to accommodate new capabilities or pricing structures. The broader vision is to allow AI-based math problem solving to scale in a way that is both financially sustainable and performance-aware. In addition to cost savings, we see this work as a step toward more intelligent and dynamic model orchestration. Instead of relying on a single powerful model to handle every input, adaptive pipelines can help break down AI tasks into smarter, more efficient workflows. Math, with its clear structure and solution correctness, offers an ideal testbed for this class of optimization.

2 Related Work

Mathematical reasoning has become a central benchmark for evaluating the capabilities of large language models (LLMs), leading to the creation of large, specialized datasets and increasingly powerful reasoning architectures. While models have made significant strides in solving complex math problems, there has been comparatively less attention on optimizing these solutions for efficiency, particularly in routing problems to different models based on difficulty or computational cost.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Throughout the modern AI landscape, math consistently serves as a significant benchmark. Examples like *Humanity's Last Exam* (Chen et al., 2024) and *FrontierMath* (Anil et al., 2024) are considered high-priority tasks for measuring LLM intelligence. New LLMs are frequently released, highlighting their growing capabilities in these domains. Consequently, a variety of datasets have been developed, including HARDMath (Yu et al., 2024), the MATH Dataset (Hendrycks et al., 2021), and Omni-MATH (Gao et al., 2024). These datasets are instrumental in training models to classify math problem difficulty.

On the modeling side, there has been a wave of progress in the capability of language models to reason through increasingly complex mathematical problems. One of the most notable developments was *Minerva*, a system that demonstrated how scaling language models and pairing them with chain-of-thought prompting techniques could yield significant performance improvements on quantitative reasoning tasks (Lewkowycz et al., 2022). Minerva's key innovation was not just in its use of a larger language model, but in its ability to reason step by step, replicating a human-like process of deriving intermediate steps before arriving at a final solution. This was particularly impactful in domains like algebra, calculus, and probability, where the intermediate reasoning process is crucial to arriving at the correct answer. The model was fine-tuned on a large corpus of mathematical content, including textbooks and academic problems, enabling it to internalize domain-specific solution strategies and logical structures. Minerva's performance on benchmarks such as MATH and GSM8K established a new standard for LLM mathematical reasoning, emphasizing the value of structured thinking over direct answer prediction.

Following this trend, *AlphaCode* applied similar principles of structured reasoning to the domain of competitive programming. Unlike mathematical problem solving, which often focuses on symbolic manipulation and numerical computation, competitive programming requires converting natural language problem descriptions into syntactically correct, semantically valid code. AlphaCode's contribution lay in its ability to generate, filter, and rerank large pools of candidate solutions using both language modeling and program execution signals. By generating thousands of potential code completions and narrowing them down based on performance and diversity, AlphaCode was able to reach human-competitive performance on Codeforces problems. While its architecture was similar to large-scale transformers used in other domains, the task-specific adaptations—including a dataset of programming contest problems and a post-generation selection mechanism—highlighted the importance of task-aligned supervision and search strategies in enhancing LLM capabilities.

More recently, models like *MathGPT* have extended these ideas to more symbolic and formal branches of mathematics, such as proofs, formal logic, and equation manipulation. MathGPT places a stronger emphasis on symbolic accuracy, aiming to emulate not just the reasoning process but also the rigorous formalism of mathematical writing. This includes handling multi-line derivations, symbolic transformations, and context-sensitive notations, which are essential for higher mathematics. These models are often trained or fine-tuned on curated datasets that contain step-by-step derivations, formal solutions, and domain-specific symbols. In contrast to general-purpose LLMs, MathGPT is designed to preserve mathematical structure and correctness even in long-form answers. As a result, it represents an evolution in the specialization of LLMs toward domain-specific reasoning, where correctness is not just measured by plausibility but by strict adherence to formal rules and derivations.

Together, these models—Minerva, AlphaCode, and MathGPT—demonstrate a trajectory in which language models are becoming increasingly adept at reasoning in structured domains. Each system applies scale, supervision, and prompting differently to align with its respective domain's needs, but all share a reliance on chain-of-thought reasoning, problem-specific training data, and downstream filtering or evaluation mechanisms. Their successes underscore the growing potential of LLMs not only to retrieve or summarize information but to engage in complex, multi-step problem solving that mirrors human cognition in specialized tasks.

Our approach draws on prior work in ensemble learning and model routing. In traditional ensemble systems, selector models are trained to decide which base model should handle a given input. More recently, systems like *RouteLLM* have explored full-model selection, sending queries to different language models based on expected cost, latency, or complexity (Jiang et al., 2024). Our work follows that logic but focuses specifically on mathematics. We train a lightweight classifier to make simple binary choices between two types of solvers.

There is also a growing body of research on predicting problem difficulty. Although this topic has a long history in educational research, its application to AI systems is relatively recent. Some

approaches focus on estimating question difficulty using features or embeddings, while others propose using model uncertainty or sensitivity as signals. In our case, we rely on difficulty labels from datasets and observed solver behavior to train our classifier to anticipate when a stronger model is likely to be necessary.

While many of these prior systems have aimed to improve overall model performance or reduce latency through dynamic model selection, relatively few have focused on the specific task of cost-aware routing for mathematical reasoning. Our approach differs in its emphasis on semantic solvability, training a classifier not on general difficulty heuristics but on whether a given math problem is empirically solvable by a cheaper model. Rather than using indirect proxies like model confidence or uncertainty, we derive routing labels directly from model behavior across datasets of varying difficulty.

Additionally, whereas some prior routing methods rely on complex selectors—often transformer-based or learned through reinforcement learning—we prioritize cost and latency minimization at every step of the pipeline. This means our classifier must be fast, lightweight, and accurate enough to make a routing decision without undermining the cost savings gained by avoiding high-end models. This makes our routing system not only mathematically aware but also computationally pragmatic. While our earlier work was conducted using Gemini models, we have since transitioned to GPT-based systems and are expanding our classifier to incorporate non-linear neural architectures. The high-level design goals remain the same, which is to maximize the efficient use of LLM capabilities through smarter problem routing.

Finally, it is worth emphasizing that our system is grounded in domain-specific observations. Rather than treating all inputs as equivalent text, we treat mathematical problems as structured entities, using embedding models that preserve semantic meaning and training classifiers that can distinguish subtle shifts in problem complexity. This domain sensitivity sets our work apart from more generic routing frameworks and enables higher precision when deciding which model is appropriate for a given task.

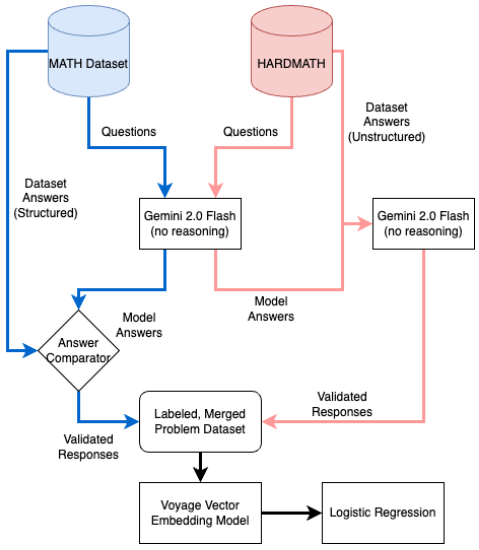

Figure 1: Old System Flowchart

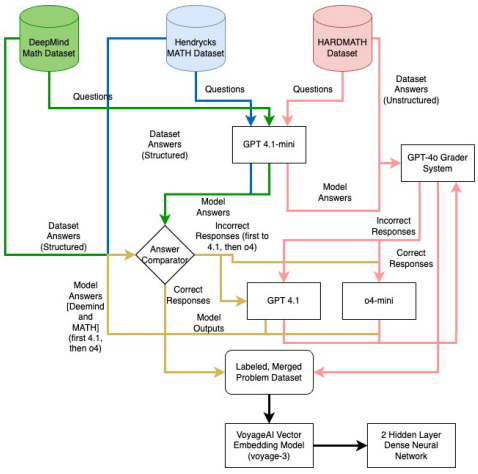

Figure 2: New System Flowchart

3 Methods

3.1 Datasets

Our goal was to create a classifier which can discriminate based on problem semantics if a problem is likely to be solvable using a mini (low parameter count), base model (1T+ parameter), or a model enabled with chain of thought reasoning. This classifier is a proof-of-concept for a model that could reduce the cost of problems by choosing a lower cost model (mini, base), which is likely to give

accurate results at the lowest latency and time. We are using data from three datasets: Dan Hendrycks' MATH dataset, which contains problems, annotated solutions, difficulty ratings, and problem category classification; the HARDMath dataset, which contains problems, annotated solutions, and problem category classification, and the Google Deepmind MATH Dataset.

3.2 Data Preprocessing and Format

Our objective was to properly format the data such that it can be used for a classifier. To that end, we processed our data using Pandas such that we were left with a dataframe including problem text, solution, model tier which solved the problem, and token counts. This is done by first sending our problems to 4.1-mini, where the model gives a response. These responses are judged, and correct responses are sent to the final dataset, alongside token counts retrieved from the OpenAI API model response objects. Then, for those problems not accurately answered by 4.1-mini, they are sent to 4.1 base, where they are solved again, re-evaluated, and split off. Finally, the problems are sent to o4-mini, where they are solved a final time. Our data follows a JSONL format as follows:

{id:"Problem_0", problem:"...", solution:"...", correct:0, 1, 2, 3, cost[input_count:0, output_count:0], ...}

The "correct" tag denoted what model, if any, was the first to solve the problem. 0 being 4.1-mini, 1 being 4.1, 2 being o4-mini, and finally 3 denoting a problem that all available models were unable to solve.

This problem, on its surface seems simple, but due to the scale of the data processing and the need for a second grader model for unparsable answers, we were required to make over 20000 calls to the OpenAI API, costing around 40 dollars total, all for data labeling and preprocessing. We were also required to write a large parsing, prompt building, and evaluation pipeline. We will detail this below.

3.3 Answer Processing

For the MATH dataset, every problem's annotated solution contained a singular discrete numerical answer wrapped by a LaTeX box. Therefore, we prompted our models to write its final answer with the same formatting, and judged its correctness by comparing extracted answers from each box with a string search. Before each comparison, we used various string replacement and parsing techniques to reduce the corresponding LaTeX to its base components to reduce the possibility of stylistic differences in answers from giving a false negative. We found that our base level non-reasoning model, GPT 4.1-mini had a very high success rate on the MATH dataset problems, around 90% (i.e. 90% of our samples received a 1 value for correct). Based on this, we concluded that to show more concrete results, we needed a more balanced split of categorical data. To achieve this, we looked at the HARDMath dataset, which contained much more difficult problems, but which did not have the solution simplicity of the MATH dataset. Similarly, we incorporate the Deepmind Math dataset (similarly difficult for the models to solve).

Due to the nature of the problems in HARDMath, the solution extraction methods that we used for the MATH and Deepmind datasets were insufficient to determine the correctness of a given solution. To address this, we used a second model in parallel to act as a "grader," which grades solutions based on rubrics derived from a problem's type. For example, if the problem asked for a proof-based response, then our grader was asked to judge a solution based on the similarity of reasoning. If the problem asked for a discrete solution, then the grader was more strict on numerical equivalence. Before we used a model as a grader, we often found that a model solution was correct, but was an alternate form of the given solution. A grader was able to determine both numerical and reasoning equivalence in order to give us our labeled data.

We also found that parsing ground-truth solutions and model solutions into Sympy and then comparing them to be very accurate at determining if a model had succeeded or not. Occasionally we needed to write special parsing methods to compare Non-Latex/Sympy format-able answers such as unordered lists.

3.3.1 Grading Model Prompt Building

Prompt engineering has one major task to achieve: lead the model into providing a solution that can be easily compared to the ground-truth solution.

For each prompt given to a model, we provided example problems and solutions, which proved to be vital for the HARDMath dataset, boosting accuracy during testing by 35%. We made sure that each query did not contain the target answer, nor did it have any context of other queries, so that the model would never input the ground-truth answer we were seeking.

Each prompt included instructions on how to format their answers, and every prompt instructed the model to encapsulate their final answer after their explanation in a LaTeX box for easier answer extraction. An example of a problem specific formatting instruction is: "write either 'True' or 'False'".

The HARDMath dataset proved especially difficult as the problems were very specialized, and we found that adding even more format specific "hints" such as "Hint: Please answer the question requiring a floating-point number with two decimal places and provide the final value" helped improve model answer extraction.

However, that was not all that was needed for HARDMath, as some of the problems did not have a singular concrete answer, often asking for formulas or abstract representations not easily parsed by tools like sympy. Therefore, we needed to use a secondary model along with a specialized grading prompt for each problem type to determine if an answer was correct.

The Hendrycks dataset and the DeepMind dataset were more straightforward. They required only small amounts of format prompting as the answers were more linear and singular in nature.

3.4 Upsampling

Unfortunately, after all of our processing and data collection, we were left with a rather weak spread of data, with Deepmind proving far too easy of a dataset, producing a split that was disappointingly as follows:

0:12391, 1:693, 2:370 3:384.

Clearly, we have a class imbalance issue, and so to fix this, we used class weighting from scikit-learn in order to take advantage of the abundance of 0 data, as well as upsampling our lower classes and downsampling the 0 classes to just 3 times the total of the rest of the classes (making sure to ensure that we avoid training on test data). Additionally, we took advantage of regularization and dropout during model training, as discussed below.

3.5 Vector Embeddings - Encoding

Once we label our dataset with LLM responses and classification targets, we need a way to encode our text into a classifiable format. While it might be better to directly train our own encoder on text using a transformer, we would need significantly more data, time, and compute to do so. So, in order to give ourselves an encoding format that is classifiable and accurately captures semantic information on the text, we employ a Voyage-AI text embedding model. This model voyage-large, a SOTA transformer based text encoder, reduces our problems to vector representations in 1024 float-64 dimensions. Under the hood, this captures the semantic meaning of our problems through a transformer, and encodes these features into a vector. These vectors are then aligned with their labels, solvable or not, from the previous data labeling steps. This decision was made because of the complicated nature of these math problems. On the surface, many of these math problems have a significant intersection in terms of words, for example things like “x”, “variable”, “integral”, etc. For this reason, a simple bag of word representation is not sufficient to capture the complex interrelations between concepts over the problems. Additionally, semantically similar vectors, and therefore vectors similar in difficulty, will be numerically close, making this an apt format for classification.

These vectors are saved and cached between usages by ID in order to reduce costs. They are then fed into our model to be used as training data alongside their labels.

3.6 Classification

Our model architecture for classification leverages a 3-hidden layer neural network. Our input layer consists of 1024 neurons matching the dimensionality of the input vectors. This is followed by our first hidden layer, a 512 neuron dense layer, with a batch normalization, and a ReLU activation. This

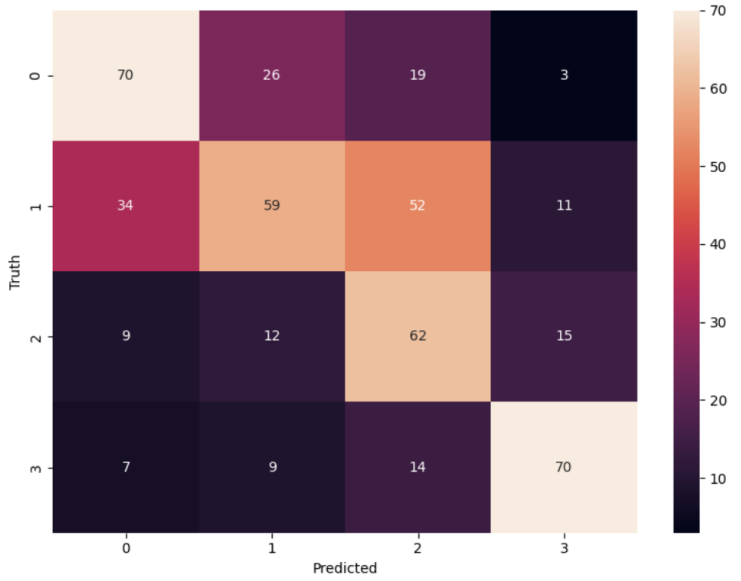

Figure 3: Confusion Matrix

is followed by 256, then 128, then 64, then 32, eventually culminating in an output layer of 4 neurons. This is then applied a softmax activation for classification, which gives our outputs as a probability. The model is trained with an Adam optimizer, using a learning rate of 0.001, and a loss function of sparse categorical cross entropy.

Our data, sparse as it is, gave us significant issues with overfitting. To that end, we went all in on trying to prevent this via the aforementioned dropout layers, regularization, and batch norms.

Dropout randomly silences a percentage of neurons during each training update, preventing complex co-dependencies between neurons by forcing the network to learn redundant representations. This technique interestingly approximates training thousands of different neural networks simultaneously and averaging their outputs, creating an implicit ensemble effect that can generalize better on new data.

Batch normalization standardizes the inputs to each layer by calculating the mean and standard deviation of the current mini-batch, then normalizing and rescaling with learnable parameters. This keeps data flowing through the network in a consistent range, preventing extremely large or small values that slow down or destabilize training.

Finally, regularization works like ridge regression by penalizing high variance in the weights of each layer, preventing overfitting be reducing variance.

3.7 Cost Analysis

The main use-case of our model is to save token costs, we benchmark our model's performance against a naive user in the whom starts with the lowest model and increases the complexity of the model on each failure.

Under the assumption that a higher model will always solve a problem if a lower model can, if our classifier model overshoots or matches the truth, our comparative cost to get the correct answer is:

$$cost(estimate) - \sum_{i=0}^{truth} cost(i)$$

If our classifier model undershoots the truth, our comparative cost is:

$$\sum_{i=estimate}^{truth} cost(i) - \sum_{i=0}^{truth} cost(i) = - \sum_{i=0}^{estimate-1} cost(i)$$

Model	Avg. Output Tokens	Cost per 1M Tokens ($)	Avg. Cost per Question ($)
GPT-4.1 Mini	417	1.60	0.0006672
GPT-4.1	549	8.00	0.003672
GPT-o4 Mini	2547	4.40	0.0112068

Table 1: Average cost per question for different GPT models

Using this cost table we can see that if our model overshoots we will gain comparative cost and therefore lose money, and if we undershoot we will always lose or match comparative cost.

Our equation however can capture complexities between models of varying scale. For example, if model 1 and model 2 together are more expensive than model 3, then overshooting 2 -> 3 will not gain cost.

4 Preliminary Results

When training a logistic regression exclusively on the MATH dataset, we achieved a misleading 91.2% accuracy by simply learning to predict the dominant class (solvable problems), rather than identifying meaningful patterns in problem complexity. We addressed this issue by incorporating the HARDMath dataset, where our base model could only solve 20% of problems, creating essential class diversity. The merged dataset produced a more robust classifier with 90.21% test accuracy and 0.3731 training loss that could effectively distinguish between problems solvable by a fast base model versus those requiring advanced reasoning.

Despite these promising results, significant limitations remained: our classifier may have learned dataset-specific patterns rather than true semantic features of problem solvability, given the stylistic differences between MATH and HARDMath; and our combined dataset lacks representation of intermediate-difficulty problems that occupy the crucial middle ground where model selection decisions are most valuable.

5 Final Results

Training our neural network model on 5692 train examples, testing on 472, with a training accuracy of 78%, and test of 58%, we get the confusion matrix above. We can see that there is a degree of overfitting still in place, with a large gap between training and testing data. Throughout the course of this project, we have learned an appreciation for how important data is to a project like this. The lack of publicly available mathematics datasets that cleanly fit classification by models like this make this issue even more pronounced. Furthermore, the large differences in formats between datasets, we presume, also affect the models ability to generalize features across problems to inform decisions that are deeper than specific words or problem styles. In other words, we believe it's possible that the model is fitting to the kinds of problems, and not some semantic understanding of the difficulty of problems.

However, while it may be true that our model was unable to **accurately** predict, in terms of absolute accuracy, the classes for most of the problems, analyzing the confusion matrix shows us that when the model misses, it doesn't miss by much. The highest error category is a false 2 when the ground truth is a 1. Evidently, the boundary in problems between 4.1 and o4 is thin. But for problems which should be solved by o4-mini, we relatively accurately predict as much. Additionally, for class 3 ground truth, we almost always see incorrect values pushed to class 2, which is the same end result for the user.

6 Conclusion

After calculating the cost analysis based on our confusion matrix and our comparative cost equations, we find that we save an average of $0.00004102 per call, or about 1/16th of a GPT-4.1-mini call.

This is not a significant difference, and can largely be attributed to o4 being incredibly expensive, meaning that overshooting to o4 nearly outweighs all of the cost saves from beneficial classification. Additionally, 4.1 is such a large and expensive model that overshooting from mini to 4.1 wipes out any cost savings.

In this project, we developed an intelligent and adaptive routing system which can accurately evaluate and direct mathematical queries a language model of sufficient solving capabilities based on the query's semantics. We began by amassing a sufficient collection of problems from mathematics datasets of varying topic and difficulty. We then turned each problem itself into a 1024-dimensional embedding and fed them into a lightweight neural network with three hidden layers and regularization in the aim to avoid overfitting.

Despite our final network's mediocre absolute accuracy on our test data, we were still pleased to find that, in reality, we gained quite a bit of cost-efficiency. These results leave us optimistic that smart routing can be effectively implemented alongside large-model reasoning development to the benefit of both accuracy and efficiency. With better data and more compute, we are confident that we would be able to create a model capable of higher accuracy.

7 Next Steps

With a lot more time, a lot more resources, and a lot more money, there are a variety of next steps we could take. First, we could scrape math contest training websites, generate problems using LLMs, RL self-play, and more. We could engage in some form of online labeling with this, where datapoints are sent, processed into a standard format, and labeled over the OpenAI API as they come in. Doing this, and improving our processing pipeline, we could likely 10x-100x our total data, which would enable our model to see the more complex boundaries in our classes.

We could abandon fragile string-difference checks and instead combine symbolic analysis with a neural-network-based rubric. Both the ground-truth solution and the candidate answer could be parsed into structured representations; otherwise, the system reverts to a GPT-driven critique prompt designed to penalize hallucinated reasoning. All results could flow through checkers so that erroneous autogrades are flagged. This could reduce the prevalence of false labels and improve data integrity.

We could train our own distilled encoder to reduce the input parameterization of the problems, and allow for a modest dense neural network to actually classify without overfitting to the data.

In an extreme case, we could even try to train with an RL policy, where making the choice to move up a model incurs a cost equal to the expected monetary cost of doing so, or something similar.

Additionally, model selection would go a long way in saving costs for an end user, as we had discussed in our conclusion a very expensive o4 model heavily distorted the results.

Finally, we could try to implement a front end solution to the problem, where users' queries are automatically routed properly to the right models and their cost savings are displayed to them directly in app.

8 References

Anil, R., Gao, B., Chen, B., et al. (2024). FrontierMath: A Benchmark for Evaluating Advanced Mathematical Reasoning in AI. arXiv:2411.04872.

Chen, B., Gao, B., Anil, R., et al. (2024). Humanity's Last Exam. arXiv:2501.14249.

Gao, B., Song, F., Yang, Z., et al. (2024). Omni-MATH: A Universal Olympiad Level Mathematic Benchmark for Large Language Models. arXiv:2410.07985.

Hendrycks, D., Burns, C., Kadavath, S., et al. (2021). Measuring Mathematical Problem Solving With the MATH Dataset. arXiv:2103.03874.

Jiang, Z., Bai, Y., Choudhury, S., et al. (2024). RouteLLM: Learning to Route LLMs with Preference Data. arXiv:2406.18665.

Lewkowycz, A., Austin, J., Zelikman, E., et al. (2022). Solving Quantitative Reasoning Problems with Language Models. arXiv:2206.14858.

Li, Y., Szlam, A., Brockschmidt, M., et al. (2022). Competition-Level Code Generation with AlphaCode. Science, 378(6624), 1092–1097.

Yu, D., Lin, Y., Liu, Z., et al. (2024). AI-Assisted Generation of Difficult Math Questions. arXiv:2407.21009.