

[← Back to Articles](#)

How NuminaMath Won the 1st AIMO Progress Prize

Published July 11, 2024

[Update on GitHub](#)


▲ Upvote 99

 +93



[yfleureau](#)
Yann Fleureau
 AI-MO



[liyongsea](#)
Li Jia
 AI-MO



[edbeeching](#)
Edward Beeching




[lewtun](#)
Lewis Tunstall



[benlipkin](#)
Ben Lipkin
 AI-MO



[romansoletskyi](#)
Roman Soletskyi
 AI-MO


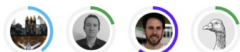















[vwxyzjn](#)
Shengyi Costa Huang



[kashif](#)
Kashif Rasul

This year, [Numina](#) and Hugging Face collaborated to compete in the 1st Progress Prize of the [AI Math Olympiad \(AIMO\)](#). This competition involved fine-tuning open LLMs to solve difficult math problems that high school students use to train for the International Math Olympiad. We're excited to share that our model — [NuminaMath 7B TIR](#) — was the winner and managed to solve 29 out of 50 problems on the private test set 🤖!

#	Team	Members	Score
1	Numina 		 29
2	CMU_MATH		 22
3	after exams		 21

4	codeinter	    		21
5	Conor #2			20

In this blog post, we introduce the Numina initiative and the technical details behind our winning solution. If you want to skip straight to testing out the model with your hardest math problems, check out our [**demo**](#).

Let's dive in!

1. [Introducing Numina - an open AI4Maths initiative](#)
2. [The AI Math Olympiad \(AIMO\) prize](#)
3. [Our winning solution for the 1st progress prize](#)
4. [The training recipe](#)
5. [Good data is all you need](#)
6. [Taming the variance with Self-Consistency and Tool Integrated Reasoning \(SC-TIR\)](#)
7. [Avoiding the curse of overfitting](#)
8. [Other ideas we tried](#)
9. [Numina's future - looking for contributors and partners!](#)
10. [Acknowledgements](#)

🔗 **Introducing Numina - an open AI4Maths initiative**

There is something very special about mathematics.


Mathematics is a domain accessible to everyone, even to children long before they can read. One of the greatest mathematicians of all time [**Srinivasa Ramanujan**](#) was self-taught, being born into a modest family in India in 1887. While for others it varies from recreation to profession and everything in between.

Mathematics is essential to humanity, being the backbone upon which we have built everything from commerce to iPhones and nuclear power plants. Yet even solving maths problems for a critical application can be a playful experience.

Pure mathematics transcends intelligence like an endless ocean only the mind can sail.

This is why when we started Numina, going open-source and open-dataset was the natural option. As for human intelligence, we believe progress in artificial intelligence for maths should be universal. If the computer is a bicycle for the mind, artificial intelligence is its engine - opening new horizons for the Ramanujans of our time.

With the initial support from Mistral AI, Numina was founded late 2023 by a collective passionate about AI and mathematics (Jia Li, Yann Fleureau, Guillaume Lample, Stan Polu and Hélène Evain), inspired by the AI Math Olympiad (AIMO) competition initiated by Alex Gerko and XTX Markets.

In early 2024, the Numina team was reinforced by two LLM fine-tuning experts from Hugging Face ( Lewis Tunstall and Ed Beeching) to tackle the 2024 AIMO progress prize. We also received additional support from General Catalyst and Answer.ai, and by March 2024, Numina had gathered a team of top talents from all around the world.

With the team in place, it was time to tackle the AIMO challenge!

🔗 The AI Math Olympiad (AIMO) prize

Every year, high school students from all around the world compete in the International Math Olympiad - a competition to solve six challenging problems across domains like algebra, geometry, and number theory. To give you a sense of the difficulty involved, here's one of last year's problems:

Problem 1. Determine all composite integers $n > 1$ that satisfy the following property: if d_1, d_2, \dots, d_k are all the positive divisors of n with $1 = d_1 < d_2 < \dots < d_k = n$, then d_i divides $d_{i+1} + d_{i+2}$ for

every $1 \leq i \leq k - 2$.

In November 2023, the **AIMO Prize** was launched to drive the open development of AI models that excel in mathematical reasoning. A grand prize of \$5M will be awarded to whoever can create an AI model that can win a gold medal in the IMO. Alongside the grand prize, AIMO has introduced a series of *progress prizes* to mark milestones toward this ultimate goal. The first progress prize was held as a **Kaggle competition**, with problems that are *less challenging* than those in the IMO but are at the level of IMO preselection. Here's an example problem, which is somewhat easier to solve than the IMO example above, but still tricky for LLMs:

“Let $k, l > 0$ be parameters. The parabola $y = kx^2 - 2kx + l$ intersects the line $y = 4$ at two points A and B . These points are distance 6 apart. What is the sum of the squares of the distances from A and B to the origin?”

The competition featured two sets of 50 problems, forming public and private leaderboards, with problems hidden from competitors. The problems, comparable in difficulty to **AMC12** and **AIME** exams, require integer outputs for verification. The private leaderboard determined the final rankings. Competitors can submit solutions twice daily, using only open-weight models released before February 23. Each submission is allocated either a P100 GPU or 2xT4 GPUs and up to 9 hours to solve the 50 problems.

Given these constraints and rules, strategic choices were essential to develop our winning solution.

🔗 Our winning solution for the 1st progress prize

After much iteration throughout the competition, our solution to the 1st Progress Prize consisted of three main components:

- A recipe to fine-tune **DeepSeekMath-Base 7B** to act as a "reasoning agent" that can solve mathematical problems via a mix of natural language reasoning and

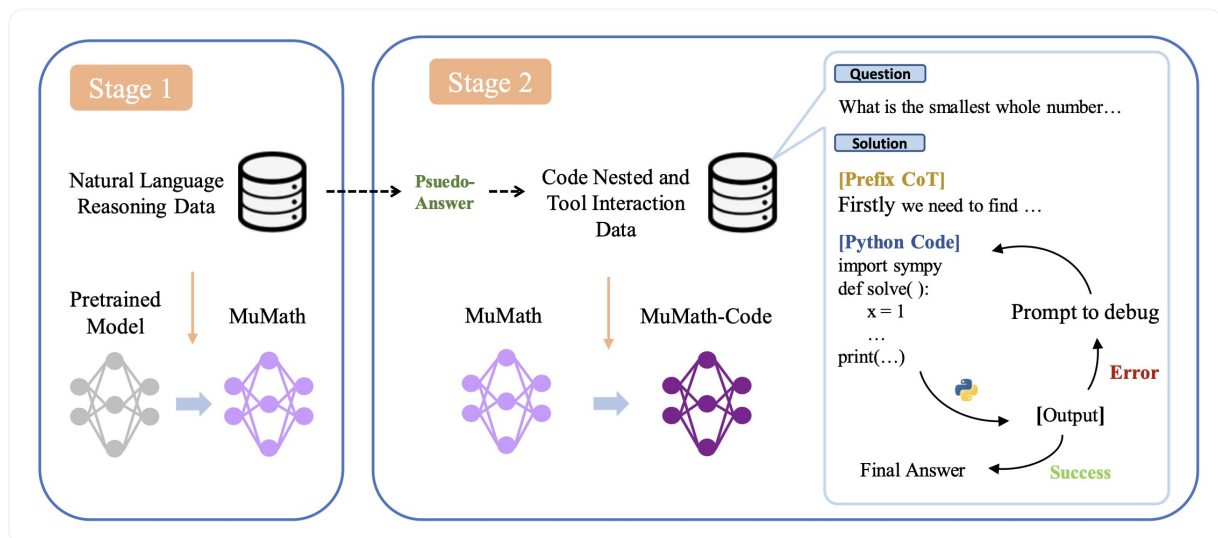
the use of the Python REPL to compute intermediate results.

- A novel decoding algorithm for tool-integrated reasoning (TIR) with code execution feedback to generate solution candidates during inference.
- A variety of internal validation sets that we used to guide model selection and avoid overfitting to the public leaderboard.

We used a mix of open-source libraries to train our models, notably [TRL](#), [PyTorch](#), [vLLM](#), and [DeepSpeed](#). On one node of 8 x H100 GPUs, our models took 10 hours to train.

🔗 The training recipe

Our fine-tuning recipe was largely based on the [MuMath-Code paper](#), which involves training the model in two stages:



Two-stage training method from the MuMath-Code paper

- **Stage 1:** Fine-tune the base model on a large, diverse dataset of natural language math problems and solutions, where each solution is templated with Chain of Thought (CoT) to facilitate reasoning.
- **Stage 2:** Fine-tune the model from Stage 1 on a synthetic dataset of tool-integrated reasoning, where each math problem is decomposed into a sequence

of rationales, Python programs, and their outputs. Here, we followed Microsoft’s [ToRA paper](#) and prompted GPT-4 to produce solutions in the ToRA format with code execution feedback. Fine-tuning on this data produces a reasoning agent that can solve mathematical problems via a mix of natural language reasoning and the use of the Python REPL to compute intermediate results (see screenshot below).

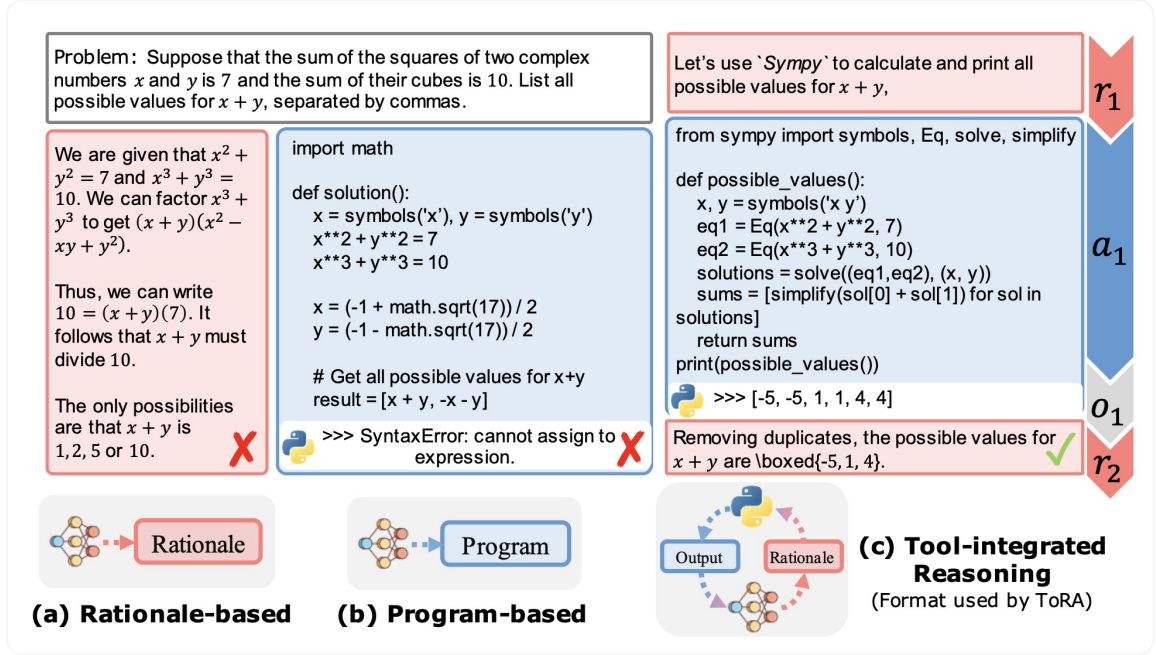


Figure from the ToRA paper on the tool-integrated reasoning format we trained our models with.

We performed “full fine-tuning” in both stages, where all model weights were updated during backpropagation. In other words, we did not use parameter-efficient techniques like LoRA or DoRA because we were not confident they could match the performance of full fine-tuning without significant experimentation. We used the “packing” feature from TRL’s `SFTTrainer` to concatenate multiple samples in a single chunk of 2048 tokens. All models were trained with gradient checkpointing and sharded with the DeepSpeed ZeRO-3 protocol to ensure the weights, gradients, and optimizer states could fit within the available VRAM. See below for the main hyperparameters we used in each stage:

	Stage 1	Stage 2
learning rate	2.0 E-5	2.0 E-5
total batch size	32	32
block size	2048	1024
num epochs	3	4
lr scheduler	cosine	cosine
warmup ratio	0.1	0.1

Our initial submissions used DeepSeek 7B models that were only fine-tuned on Stage 1, but we found the performance was quite limited, with 8/50 being our best score on the public leaderboard using maj@32. It was Abdur Rafae's public prize notebook that prompted us to take a look at integrating code execution in the training recipe. Initially, we focused on the Mix of Minimal Optimal Sets (MMOS) dataset, as described in the notebook's title. We found that using MMOS improved performance but was still capped at 16/50 on the public leaderboard with maj@32, likely due to the fact that MMOS only consists of single-turn solutions (i.e., the model only generates a single Python program, which is insufficient for hard problems). We later realized that MMOS was a misnomer and that Kaggle notebooks were actually running the DeepSeekMath 7B RL model, which is capable of multi-step reasoning and code execution.

At this point, we focused our efforts on producing a dataset similar to the one used by the DeepSeekMath Instruct / RL models, and this, together with the MuMath-Code recipe, led to significant improvements.

Let's take a look at how we built these datasets.

🔗 Good data is all you need

In terms of the dataset, we have extensively referred to DeepSeek Math and other

scholars' approaches, scaling them up significantly. This has resulted in a fine-tuned dataset of several hundred thousand problem-solution pairs, covering topics from high school mathematics to competition-level mathematics. This dataset will be fully open-sourced over the next few weeks, potentially with larger models to see how well our recipe scales. Please refer to our upcoming dataset technical report for details on the dataset construction. When it comes to the progress prize, we have built two datasets so to finetune our model.

Chain of Thought

This dataset consists of several hundred thousand problems, each with solutions written in a Chain of Thought manner. The sources of the dataset range from Chinese high school math exercises to US and international mathematics olympiad competition problems. The data were primarily collected from online exam paper PDFs and mathematics discussion forums.

The processing steps include:

1. OCR from the original PDFs.
2. Segmentation into problem-solution pairs.
3. Translation into English.
4. Realignment to produce a Chain of Thought reasoning format.
5. Final answer formatting.

Tool-integrated reasoning

Tool-integrated reasoning (TIR) plays a crucial role in this competition. However, collecting and annotating such data is both costly and time-consuming. To address this, we selected approximately 60,000 problems from the Numina dataset, focusing on those with numerical outputs, most of which are integers.

We then utilized a pipeline leveraging GPT-4 to generate TORA-like reasoning paths, executing the code and producing results until the solution was complete. We

filtered out solutions where the final answer did not match the reference and repeated this process three times to ensure accuracy and consistency. This iterative approach allowed us to generate high-quality TORA data efficiently.

As a point of reference, here is the performance of our Stage 1 model **NuminaMath-7B-CoT** and final Stage 2 model **NuminaMath-7B-TIR** on the **MATH benchmark** compared to other open and proprietary models:

Model	MATH (%)
	Chain of Thought Reasoning
GPT-4 (2023)	42.5
GPT-4o	76.6
Claude 3.5 Sonnet	71.1
DeepSeekMath-7B-Instruct	46.8
DeepSeekMath-7B-RL	51.7
NuminaMath-7B-CoT	56.3
	Tool-Integrated Reasoning
DeepSeekMath-7B-Instruct	57.4
DeepSeekMath-7B-RL	58.8
NuminaMath-7B-TIR	68.2

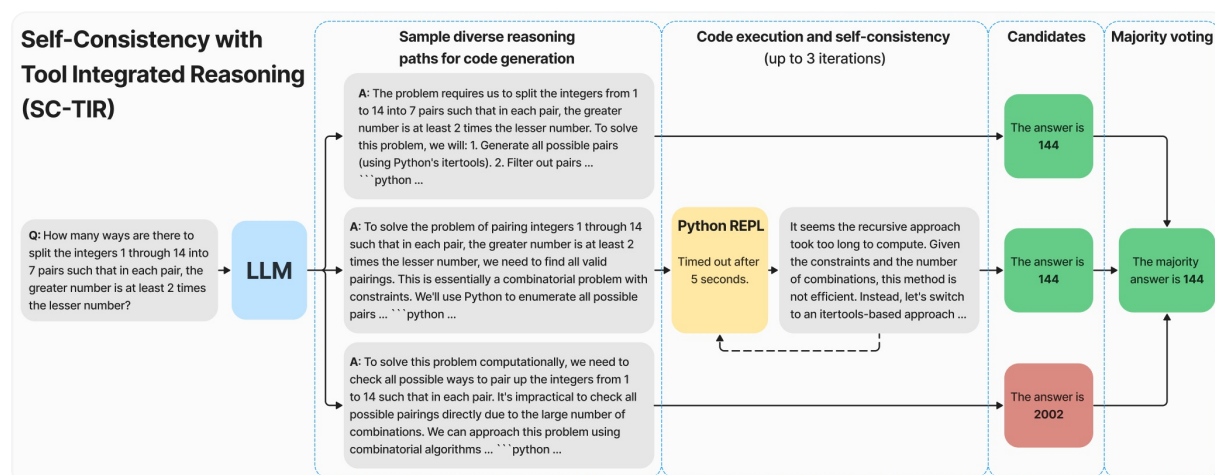
Performance on MATH benchmark. All numbers, unless explicitly stated, are obtained with zero-shot greedy decoding.

Taming the variance with Self-Consistency and Tool Integrated Reasoning (SC-TIR)

As other competitors noted, this competition posed several challenges with respect to model submission and evaluation:

- The evaluation API provides problems in random order, so tactics like early stopping produce high variance because one run may have more hard problems at the start, which leaves less time for the remainder (and vice versa)
- Most innovations in LLM inference require access to modern GPUs, so standard methods like Flash Attention 2 or torch.compile do not work on T4 GPUs. Similarly, modern data types like bfloat16 are not supported, which prompted us to explore post-training quantization methods like AWQ and GPTQ.

Initially, we used [Abdur Rafae's public notebook](#) for our submissions, but found the high variance to be problematic. To handle this, we took a different approach based on tool-integrated reasoning:

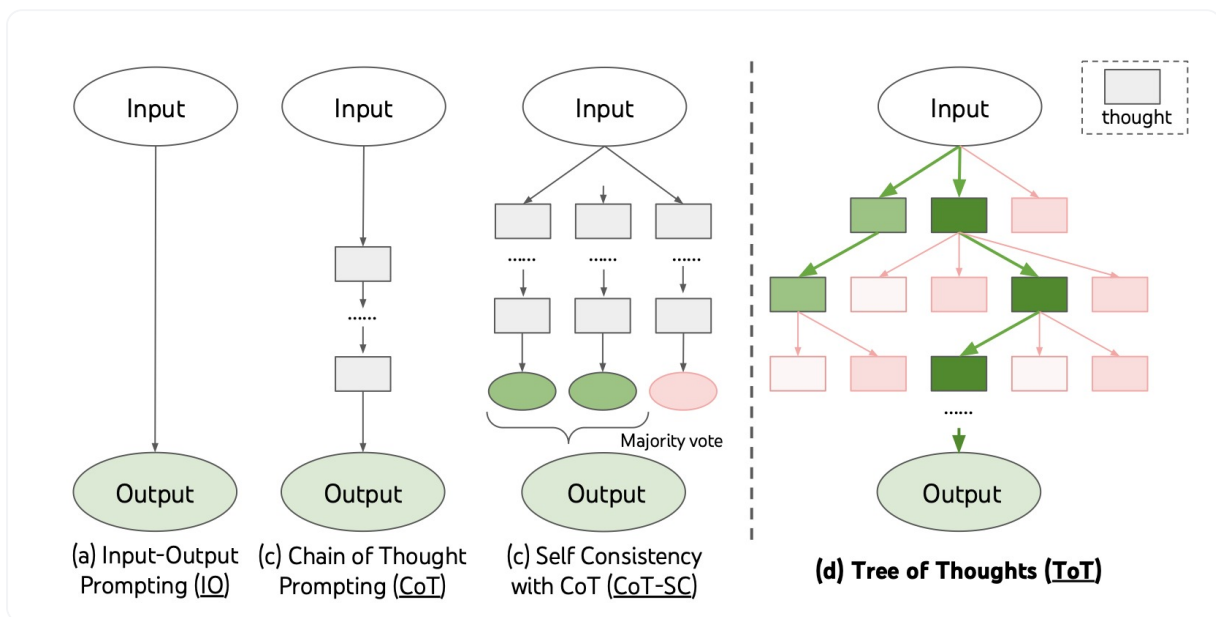


1. For each problem, copy the input N times to define the initial batch of prompts to feed vLLM. This effectively defines the number of candidates one uses for majority voting.
2. Sample N diverse completions until a complete block of Python code is produced.
3. Execute each Python block and concatenate the output, including tracebacks if they appear.
4. Repeat M times to produce a batch of generations of size N and depth M, allowing the model to self-correct code errors using the traceback. If a sample

fails to produce sensible outputs (e.g., incomplete code blocks), prune that result.

5. Postprocess the solution candidates and then apply majority voting to select the final answer

For our winning submission, we generated $N=48$ candidates with a depth of $M=4$. Increasing either parameter did not improve performance, so we took a conservative approach to stay within the time limit. In effect, this algorithm augments Self Consistency with CoT (shown below) with Tool-Integrated Reasoning.



We found that our SC-TIR algorithm produced more robust results with significantly less variance on both our internal evaluations and the public leaderboard.

One technical detail worth mentioning is that we found it helpful to quantize the models in 8-bit precision. This was for three reasons:

- It is very slow to upload models to the Kaggle Hub, and compressing the model made this step twice as fast.
- T4 GPUs do not support bfloat16, and casting to float16 leads to a degradation in model performance. Casting to float32 was not possible as that exceeded the available GPU memory.
- In addition, a 16-bit model consumes approximately 32GB VRAM just to load the weights. With 2xT4s, this would have required manipulating the KV cache

to run fast, and we found it beneficial to tradeoff model precision for speed.

We quantized our models using [AutoGPTQ](#) along with the training datasets for calibration. In practice, this led to a small drop in accuracy but provided the best compromise to accommodate the constraints imposed by evaluation on the Kaggle platform.

🔗 **Avoiding the curse of overfitting**

Overfitting to the public leaderboard is a common risk in Kaggle competitions, and even more so when the test set is just 50 problems. In addition, the rules allowed at most two submissions per day, making a robust internal validation dataset crucial for pacing our development. As specified by the AIMO team, the test problems are of intermediate difficulty, between AMC12 and AIME levels, with integer outputs.

To guide model selection, we used four internal validation sets to gauge the performance of our models on math problems of varying difficulty. To avoid potential contamination in the base model, we selected problems from AMC12 (2022, 2023) and AIME (2022, 2023, 2024) to create two internal validation datasets:

- **AMC (83 problems):** We picked all the problems from [AMC12 22](#), [AMC12 23](#) and kept those that can be converted to integer outputs. This results in a dataset of 83 problems. This validation set was designed to be representative of the private test set on Kaggle since we knew from the competition description that the problems were of this level or harder. We found our models could solve about 60-65% of these problems. To measure the variance, we ran each evaluation with 5-10 different seeds and typically saw variations of around 1-3% with our SC-TIR algorithm.
- **AIME (90 problems):** We picked all the problems from [AIME 22](#), [AIME 23](#), and [AIME 24](#) to measure how well our models could perform on difficult problems, as well as to gauge the most common failure modes. As above, we ran each evaluation with 5-10 seeds to measure variation.

Due to the small size of the AMC/AIME validation sets, model performance on these

datasets was susceptible to noise, similar to the public leaderboard. To better assess our model's performance, we also evaluated it using a subset of the MATH test set, which contains 5,000 problems. We retained only the problems with integer outputs, to simplify majority voting and mimic competition evaluation. This resulted in two additional validation sets:

- **MATH level 4 (754 problems)**
- **MATH level 5 (721 problems)**

By using these four validation sets, we were able to pick the most promising models across different training stages and narrow down the choice of hyperparameters. We found that combining small but representative validation sets with larger ones was useful in this particular competition, where each submission is subject to some stochasticity from sampling.

🔗 Other ideas we tried

As mentioned above, we tried a few approaches that were ultimately discarded in favor of the MuMath-Code recipe:

- Training a pure CoT model and using majority voting for evaluation
- Training an MMOS model to solve problems with Python in a single step

Another technique we tried was applying **Kahneman-Tversky Optimisation (KTO)** to new completions sampled from the SFT model. Here the approach was similar to **OrcaMath**, namely:

- Sample 4 completions per problem with the SFT model, using interleaved rationales and code execution. We used the SFT dataset from Stage 2 as the source of prompts.
- Extract the answer and compare it with the ground truth. If correct, label the sample as positive, else negative.
- Apply KTO to the SFT model on this dataset.

We found this form of on-policy KTO produced a slightly better model than the SFT one (a few percentage points on our internal evaluations) and scored 27/50 on the public leaderboard.

One nice feature of KTO is that you can track the implicit reward during training, and this really helps with debugging a run - for example, here's one of our successful training logs where one sees the chosen (i.e., correct solutions) rewards increase over training, while the rejected ones are suppressed.

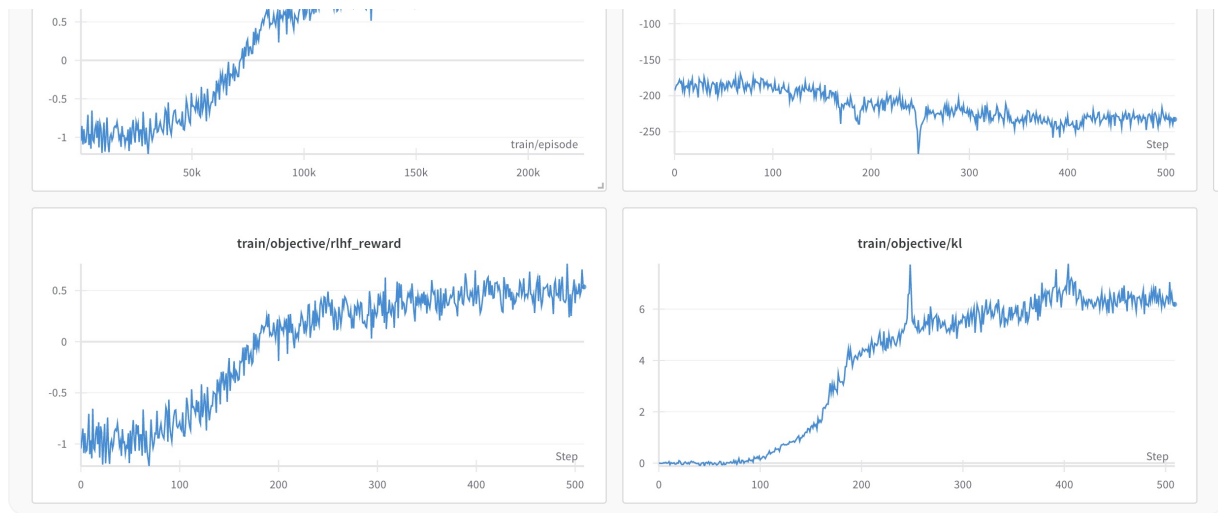


Unfortunately, we ran out of time to apply this method to our final SFT model, so it is possible we may have been able to 1-2 more problems!

We also experimented with applying our SFT recipe to larger models like InternLM-20B, CodeLlama-33B, and Mixtral-8x7B but found that (a) the DeepSeek 7B model is very hard to beat due to its continued pretraining on math, and (b) inference is very slow on 2xT4 GPUs, and we experienced a number of mysterious timeouts that we couldn't trace the root cause of.

Another failed experiment includes trying to use reinforcement learning (specifically the Proximal Policy Optimization algorithm and **REINFORCE-leave-one-out (RLOO) algorithm**) with code execution feedback and shaped rewards for writing code and getting correct/incorrect solutions. We applied this to the DeepSeekMath 7B RL model. While we saw some promising reward curves, we did not see any significant gains in performance. Given that online methods like RLOO are bottlenecked by text generation and slow to iterate with, we abandoned reinforcement learning in favor of experimenting with KTO.





On the inference side, we also experimented with:

- Using a static KV cache and torch compilation. We found we were able to speed up generation in native transformers code by 2-3x on a H100, but hit a variety of cryptic errors on the Kaggle T4s, mostly due to the lack of support for model sharding with torch compilation in accelerate.

A variety of model merging techniques like [DARE](#), [TIES](#), and [WARP](#). Here we used [mergekit](#) to merge the SFT and KTO models, or the SFT models with the public DeepSeekMath ones. Overall we found these merges led to either significant regressions on our internal evaluations and we ran out of time to explore this more deeply.

Numina's future - looking for contributors and partners!

Following the initial success of Numina at winning the AIMO 2024 progress prize, we now aim to pursue our mission of fostering the development of artificial and human intelligence in the field of mathematics. You can visit our website to know more about our projects and please always feel free to drop us a note at contact@projectnumina.ai.

Numina, like mathematics, is meant to be open to talents and supporters from all around the world who are willing to bring mathematics further with AI!

🔗 Acknowledgements

We thank Thomas Wolf and Leandro von Werra for enabling the Numina and Hugging Face collaboration. We also thank Hugo Larcher for helping make the GPUs go brrrr on the Hugging Face cluster, Colin Raffel for his advice on model merging methods, and Omar Sanseviero for feedback on the blog post.

We also wanted to express our gratitude to [Mistral.ai](#), [General Catalyst](#), [Answer.AI](#), and [Beijing International Center for Mathematical Research](#) @ Peking University who supported the project from the beginning.

Finally, we thank the AIMO Prize team for launching such an exciting and inspiring competition!



Company

[TOS](#)

[Privacy](#)

[About](#)

[Jobs](#)

Website

[Models](#)

[Datasets](#)

[Spaces](#)

[Pricing](#)

[Docs](#)

