Extended Abstract

Motivation Although Large Language Models (LLMs) have performed remarkably well when doing natural language processing (NLP) tasks, they fall short for mathematical reasoning problems. LLMs often struggle to compute mathematical equations accurately and hallucinate results in expressions. Traditionally, LLM benchmarks for problem solving do inference in isolation, preventing these models from having access to external tools. This is unrealistic in a real-world setting, where LLMs utilize tools like calculators, Python interpreters, and search engines to generate a response. With this in mind, we provide an LLM with access to a calculator tool, hoping that the model will rely on the calculator for arithmetic and focus on developing a strategy. Our goal is to see if this approach results in better performance for mathematical tasks, namely in Countdown.

Method Our method is composed of three parts. First, we fine-tune the Qwen 2.5-0.5B model Supervised Finetuning (SFT) on the Warmstart dataset for initial alignment. Then, we use the RLOO algorithm on the Prompts dataset to further improve the model by rewarding accurate solutions. Lastly, we add our calculator to the mix and perform SFT with a modified dataset. The calculator tool can deterministically evaluate simple expressions that are appended to the output by wrapping expressions in a special tag. When these tags are seen during generation, the model is stopped, and the output is appended to the model's reasoning chain.

Implementation To implement the tool use functionality, we preprocess the training data in the Warmup dataset by finding expressions and wrapping each respective component in <calculator></calculator> and <solution></solution> tags. This approach allows the model to learn how to properly access the calculator tool API. During inference, the model runs the eval function on safe expressions within the <calculator></calculator> tags and returns the result to the model in <solution></solution> tags. We perform RLOO with 16 sequence generations for each prompt and apply the reward for exact answers. The expressions are checked with a validator to accurately determine the correct answer for each Countdown prompt.

Results For SFT, the model was able to achieve a solid baseline, reaching an accuracy score of 0.43 on leaderboard1 and 0.19 on leaderboard2 after 20 epochs of training. RLOO on the model after SFT alignment further improved performance to 0.695 on leaderboard1. Our tool-use approach also improved performance on the original SFT implementation by achieving an accuracy score of 0.275 on leaderboard2. On the final leaderboard, RLOO achieved a winrate of 0.3889 and our tool-use approach achieved a winrate of 0.3403.

Discussion Our tool-augmented model shows promise in improving LLM reasoning capabilities when doing math-related tasks like Countdown. However, the gains from using tools were modest when only performing supervised fine-tuning. Although the calculator tool makes it easier for the model to evaluate expressions, it adds complexity to token generation and the decision-making process. After the initial alignment, RLOO drastically improves model performance by learning from generated samples. Further fine-tuning using RLOO or some other technique on the calculator-aligned model is likely necessary for the model to achieve substantial improvements in performance.

Conclusion Our work explores a number of methods that improve LLM performance in math reasoning tasks. We show that RLOO is necessary in order to achieve significant results with our SFT models. Additionally, external tool use can improve model capabilities for arithmetic expression evaluation. We highlight the relevance of using tools in LLM settings to perform real-world tasks and how they interact with model alignment using reinforcement learning. Future work could explore the use of a wider range of tools in mathematical tasks and expand the scope to other tasks such as coding, information retrieval, and theorem proving.

Improving LLM Mathematical Reasoning Capabilities Using External Tools

Jack Albright
Department of Computer Science
Stanford University
jacalbr@stanford.edu

Sheden Andemicael
Department of Computer Science
Stanford University
sheden@stanford.edu

Abstract

Although Large Language Models (LLMs) perform well on natural language tasks, they often struggle with mathematical reasoning, especially when required to compute exact values. Standard LLM benchmarks typically evaluate models in isolation, ignoring realistic scenarios where models can access tools like calculators or interpreters. In this work, we design a calculator tool for a fine-tuned LLM, allowing it to delegate the arithmetic workload and focus on strategic reasoning. Our goal is to see whether tool use improves performance on math tasks like Countdown. We apply three methods: supervised fine-tuning (SFT) of the Qwen 2.5-0.5B model on the WarmStart dataset, reinforcement learning via RLOO to reward correct completions, and SFT with integrated tool use through tagged calculator calls. During inference, the model pauses generation to execute arithmetic expressions enclosed in special tags, appending the result back to its output. SFT alone reached 0.43 accuracy on leaderboard1 and 0.19 on leaderboard2. RLOO significantly improved performance to 0.695 accuracy, while the calculator-augmented model achieved 0.275 on leaderboard2. On the final leaderboard, RLOO achieved a winrate of 0.3889 and our tool-use approach achieved a winrate of 0.3403. These results suggest that tool use can enhance LLM capabilities when combined with reinforcement learning. Future work should explore RLOO finetuning of tool use models, as well as broader tool use across a larger set of reasoning tasks.

1 Introduction

For this project, we use two different training methodologies to fine-tune a large language model to complete a mathematical reasoning task called the Countdown task. And for the extension, we explore how large language models can be trained to utilize external applications to approach language tasks. Specifically, for the math reasoning task, we can permit the LLM to send requests to an external calculator tool. This is designed to offload much of the computational pressure from the model, allowing it to specialize more in planning and delegation. While most model benchmarks evaluate models based on their own ability without any assistance from outside resources, this does not accurately reflect what we should expect of models deployed in real-world scenarios. Deployed language models, like us humans, will have access to tools like calculators, IDEs, and even web search. Our project extension seeks to measure how effectively a language model can utilize one of these tools.

2 Related Work

The paper by Ahmadian et al. (2024) explores the use of various reinforcement learning methods for fine-tuning LLMs to align them with human preferences on reasoning tasks. The authors compare

SFT, Reward Modeling (RM), and Reinforcement Learning from Human Feedback (RLHF) to optimize model performance on a diverse set of benchmarks. They show that using SFT provides a solid baseline model by aligning the model with ground truth outputs. Further improvements can be achieved by sampling from the model using RLHF. One such method they explored was RLOO, which biased the model towards more accurate completions. To perform this method, the authors of the paper trained a reward model using human feedback that tried to estimate the reward of the model's output. Instead of using human feedback, in our paper, we use an expression validator that provides a reward signal of one for correct answers on the Countdown task and zero otherwise. Ahmadian et al. (2024) is used as justification for training our model in the step-by-step fashion outlined in this paper.

Schick et al's paper Schick et al. (2023) introduces a model named *Toolformer* that can tackle the fundamental constraints of LLMs in performing tasks that require accurate calculations or factual inquiries by enabling them to determine how and when to use external tools. The model is designed to decide which APIs to invoke, the correct moment for these invocations, the parameters to provide, and how to fit the result in future text generations. The authors accomplish this task using a self-supervised method. The tools included in the paper are a calculator, a Q&A system, search engines, a translation tool, and a calendar. We hope to apply the methods in this paper in tackling the math reasoning task. And we plan on extending on these methods, either by changing the model structure or adding additional tools.

The paper by Jin et al. Jin et al. (2025) introduces a framework that uses RL to improve an LLM's reasoning skills via independent search queries and immediate information access. Their model, Search-R1, is trained to use search queries throughout the reasoning process, allowing it to gather external information dynamically. The model uses RL instead of prompting during inference to enhance the model's decision-making when using tools, similar to the paper above. We will apply the idea of dynamic tool use to allow our model to verify its answer during inference for the Math Reasoning task.

3 Method

Our project can be understood as having three stages. First, we perform supervised fine-tuning (SFT) on a Qwen 2.5 0.5B Base model using the WarmStart dataset. This can be understood as optimizing the SFT objective which measures the log probability of the model generating the preferred completions in the training dataset.

$$\max_{\theta} \mathbb{E}_{x,y \in D} \sum_{t=1}^{|y|} \log \pi_\theta(y_t | x, y_{<t})$$

Second, we use REINFORCE Leave-One-Out (RLOO) to fine-tune the SFT model using the Prompts dataset. Specifically, we generate $k = 16$ completions for each numbers-target pair, find the log probabilities for each completion, and weight these losses based on the completions' correctness scores according to the formula below. The purpose of this weighting is to select for correct completions and against incorrect completions. Due to the large size of the Prompts dataset, we only use a small sample of the dataset for training.

$$\frac{1}{k} \sum_{i=1}^{k} \left[R(y_{(i)}, x) - \frac{1}{k-1} \sum_{j \neq i} R(y_{(j)}, x) \right] \nabla \log \pi(y_{(i)} | x) \text{ for } y_{(1)}, ..., y_{(k)} \overset{i.i.d}{\sim} \pi_\theta(\cdot | x)$$

Finally, we repeat SFT with an external calculator tool added to the pipeline. Specifically, we add special tokens to the model's vocabulary that allow it to enclose requests to the calculator tool in HTML-like tags. If a valid request is detected, the model decoding will pause, and the request will be sent to the calculator tool. The calculator tool is a simple python script that confirms if the string only contains numbers and operations (e.g. "2*3-1*4") and calls eval() on this string. This answer will then be added to the model output in a designated answer section of the request, where it can be used by the model in future decoding steps.

For SFT, we use a modified version of the WarmStart dataset. The sample outputs in the WarmStart dataset are full of equations stated as fact — for example, "60/15 = 4". We modify the dataset by locating these equations and introducing the relevant calculator tokens. The example above would become "<calculator>60/15</calculator><solution>4</solution>". The calculator's answer logits are masked out of training, and the initial user prompt is modified to indicate how to format these requests.

4 Experimental Setup

We've used two datasets for training the model. The first dataset is the WarmStart dataset, which contains query-response pairs for the Countdown task. The dataset consists of 1000 training pairs and 200 validation pairs. The second dataset is the Prompts dataset, which contains raw numbers-target pairs for the Countdown task. The dataset consists of about 490,000 pairs in total.

We also used the provided held-out prompts for the leaderboard to calculate validation accuracy. As two different sets of prompts were provided at different points in time, some of our models are evaluated on different prompt sets. For clarity, the first set of held-out prompts will be referred to as "leaderboard1" and the second set of held-out prompts will be referred to as "leaderboard2". A model's validation accuracy is calculated by generating a response to each of the prompts in the validation dataset and scoring them based on their correctness. Specifically, a response is given a score of 1 if the answer only contains the numbers allowed and the answer evaluates to the correct target number, and 0 otherwise. This score is not used to train the model in any way, and only serves as a measurement of model performance. When relevant, we also include leaderboard win-rate scores for comparison.

When training a model with SFT, we collected training loss values based on the SFT objective. For RLOO, due to the large size of the Prompts dataset, we chose to use validation accuracy as a measurement of model progression, as any loss calculation would be comparing model performance on completely different prompt sets.

5 Results

After a single epoch of SFT on the base model, the model returned an accuracy score of 0.37 on leaderboard1. When increased to 20 epochs of SFT, the model's accuracy scores increased to 0.43 on leaderboard1. The model scored 0.19 on leaderboard2 and recorded a win-rate of 0.2674 on the final leaderboard. Applying RLOO to the SFT model increased the model's accuracy scores to 0.695 on leaderboard1 and 0.324 on leaderboard2. The win rate on the final leaderboard also increased to 0.3889. Finally, performing 20 epochs of SFT on the base model with the external calculator tool added to the pipeline resulted in an accuracy score of 0.275 on leaderboard 2 and a win rate of 0.3403 on the final leaderboard. These scores are higher than that of SFT without the calculator, but lower than that of SFT with RLOO.

Table 1: Performance Comparison

Method	leaderboard1 accuracy	leaderboard2 accuracy	Leaderboard Win-rate
SFT (1 epoch)	0.37	-	-
SFT (20 epochs)	0.43	0.19	0.2674
SFT+RLOO	0.695	0.324	0.3889
SFT (+Calculator)	-	0.275	0.3403

For SFT on its own, model training seemed to take about 3500-4000 batches (or about 15 epochs) to converge, although most of the model's performance increases occur in the very first epoch. The same is the case for model training for SFT (+Calculator).

RLOO on the SFT model seems to roughly converge after about 400 batches, where it hovers around 0.65 accuracy on the validation set.

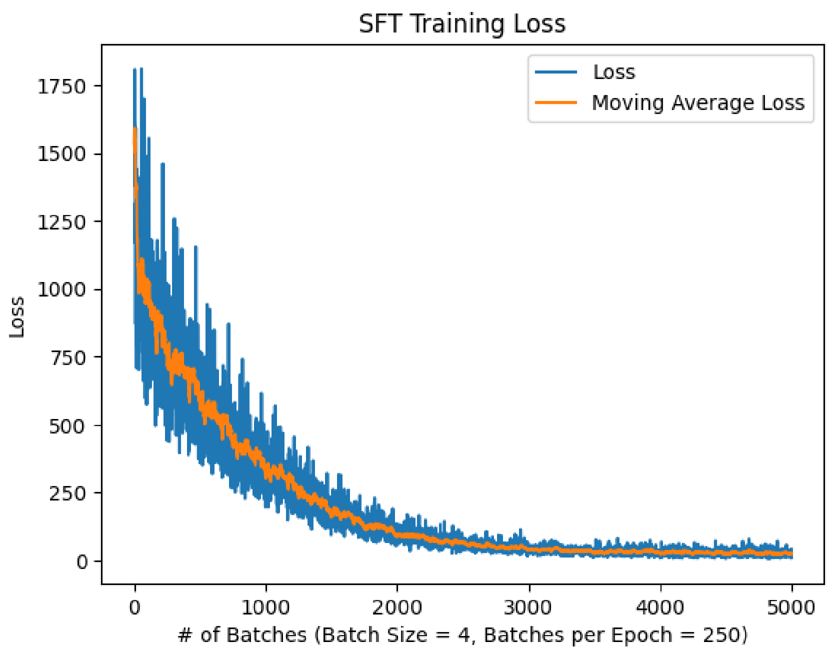

Figure 1: SFT Training Loss curve

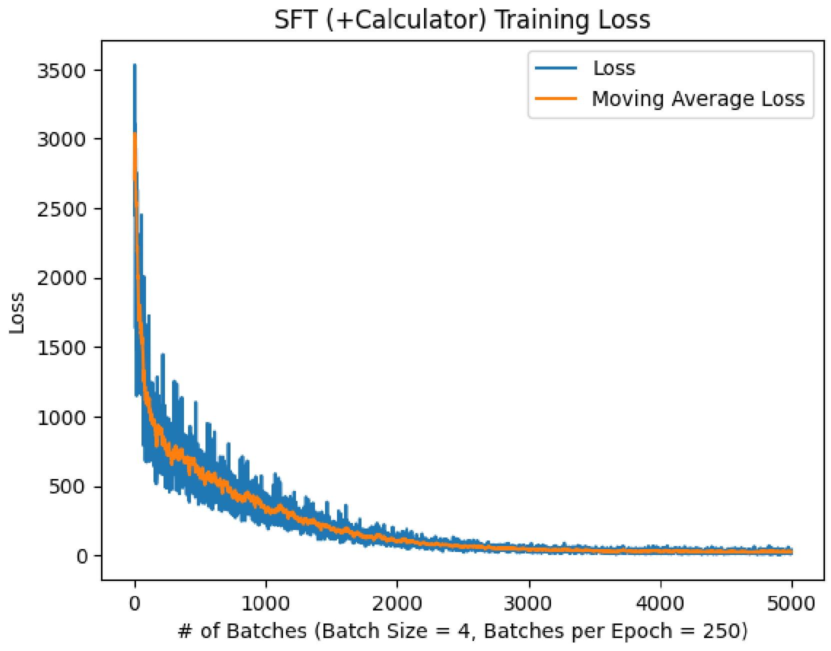

Figure 2: SFT (+Calculator) Training Loss curve

6 Discussion

While the loss curves for SFT and SFT (+Calculator) start at different locations, after about a couple epochs both models seem to follow similar loss curves. This can especially be seen in 4, suggesting that adding tool use during fine-tuning does not make the learning problem much harder or destabilize training. We see that the tool-augmented model starts at a slightly higher loss level, likely due to the fact that the base Qwen model was not used to generating this type of structured output to reference

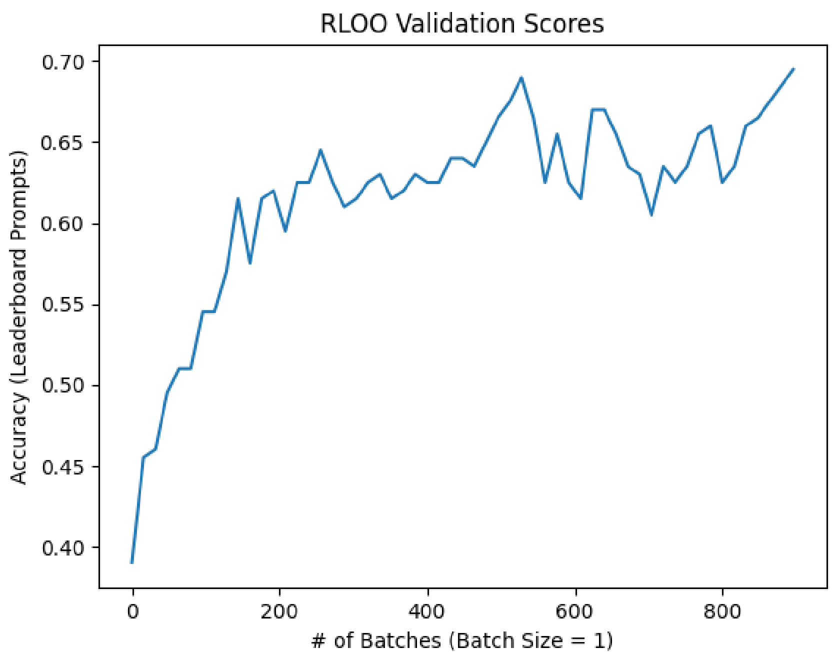

Figure 3: RLOO Validation Accuracy curve

external tools. The loss ends up becoming nearly identical quickly, indicating that the base model is well-suited to generating outputs infused with tags. The identical losses also suggest that the tool-augmented model outperformed the default SFT model because of better reasoning performance rather than differences in training.

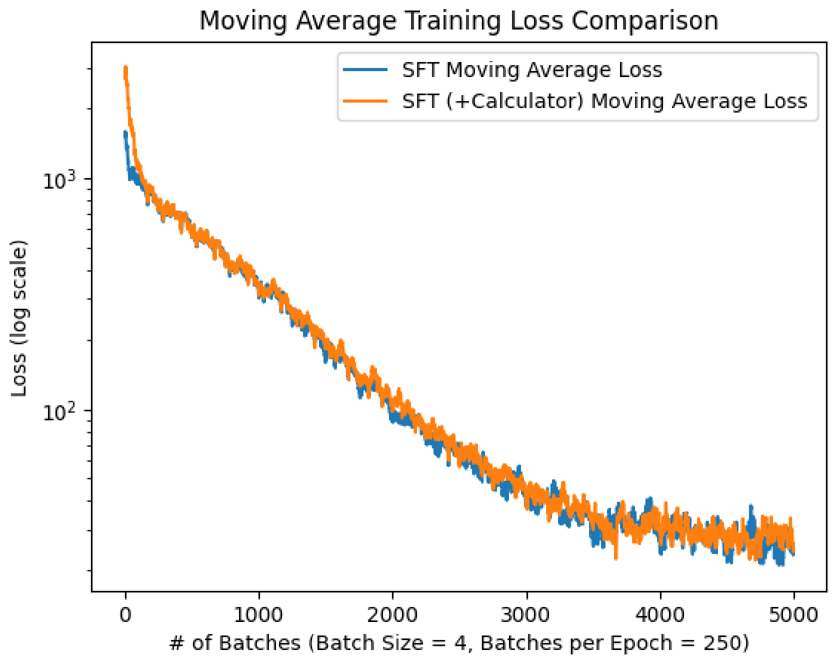

Figure 4: Log Training Loss curve of both SFT trained models

RLOO fine-tuning seems to slow down fairly early in the training process. Most of the performance increase due to RLOO occurs after only a hundred training examples, and model performance seems to grow somewhat unstable as the batches progress. Overall, we don't expect additional RLOO fine-tuning to improve the model's performance here.

Although the tool-augmented SFT model outperformed the base SFT aligned model, it is important to mention the inference time tradeoff necessary to achieve these results. The tool-augmented model experienced much slower generations during sampling because the model needed to be paused at each occurrence of <calculator></calculator> tags, evaluate the expression, append the solution to the output, and resume generating the response. This was especially harsh in the Countdown task since the model would tend to make many intermediate calculations before reaching an answer. Though training time was roughly the same as the base SFT model, this led to a significant increase in inference time for the tool-augmented model. This also prevented us from effectively applying RLOO to our tool-augmented model (beyond a few batches). Future work may explore efficient optimizations for sampling from a tool-augmented model, such as some kind of modified KV caching.

7 Conclusion

In our paper, we explored the effectiveness of combining tools and alignment using reinforcement learning to achieve improvements in mathematical reasoning tasks. We fine-tuned a Qwen 2.5-0.5B model using SFT to align the model to ground truth output and used RLOO to further improve the model from generated samples. For exploration, we fine-tuned another model using SFT that had access to a calculator tool with the goal of achieving further improvements in the Countdown task. Our tool-augmented model outperformed the base model fine-tuned with default SFT on leaderboard2, suggesting that access to external tools like a calculator can help the LLM reason by taking on the computation load and allow the LLM to focus on strategic planning. While the RLOO fine-tuned model achieved a large performance boost compared to only doing SFT, implementing the calculator tool only led to more modest improvements. This result underscores the need to perform further alignment using reinforcement learning algorithms that generate direct samples from the model. The use of tools in completing reasoning tasks has much potential in unlocking more powerful LLM capabilities. Our paper focuses on the importance of using LLMs in realistic environments where they can use external tools to complete tasks. Future work can look at using other kinds of tools, such as code interpreters and search engines, as well as explore broader reasoning domains outside of mathematics. Furthermore, the impact of RLOO on training a tool-augmented model deserves more exploration.

8 Team Contributions

We split the work for this project evenly. We both did parts of data loading and construction, method implementation, the extensions, and writing/design for the milestone, final report, and poster. No changes from the proposal to mention.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs. arXiv:2402.14740 [cs.LG] `https://arxiv.org/abs/2402.14740`

Xinshi Jin, Linyong Nan, Zihan Zhang, Colin Wei, Yiqin Yang, Steven C.H. Hoi, and Bo Li. 2025. Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning. arXiv:2503.09516 [cs.CL]

Timo Schick, Jane Dwivedi-Yu, Roberta Raileanu, Niklas Muennighoff, Yulia Tsvetkov, Kurt Shuster, and Douwe Kiela. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL]