Extended Abstract

Motivation Large Language Models (LLMs) demonstrate high performance on a wide range of natural language tasks but often fail at multistep reasoning, especially mathematical problem solving. While supervised fine-tuning helps improve baseline LLMs' ability to reason, it does not consistently teach the logical reasoning necessary to solve multi step puzzles. Reinforcement learning, with its capacity to use reward signals aligned to correctness, presents a promising avenue for fine-tuning models on this kind of task. Our goal is to determine whether RL techniques, specifically on-policy and off-policy methods, can meaningfully improve performance on arithmetic reasoning challenges.

Method We compare two reinforcement learning strategies: REINFORCE Leave-One-Out (RLOO), an on-policy approach, and Tapered Off-Policy REINFORCE (TOPR), an off policy approach. RLOO collects multiple completions from the current model per prompt and computes gradient updates using variance-reduced rewards. In contrast, TOPR operates on previously generated (prompt, output, reward) triplets and performs updates based on assigned importances. Both methods start with an SFT baselines.

Results We found that both RL (RLOO and TOPR) approaches outperformed the SFT baseline in terms of accuracy and average reward. TOPR showed faster convergence and was easier to tune. It also generated validly formatted answers more consistently. Using a simple reward penalty was good for improving the SFT model, and using a more structured reward function after that initial improvement boosted our score even further. We also noticed that problems which involved large numbers tended to perform the worst, indicating perhaps an imbalance in the training data.

Discussion The comparison between RLOO and TOPR illustrates a trade-off between adaptability and scalability. RLOO benefits from real-time exploration and feedback, enabling it to improve even in sparse reward environments. However, it is expensive to run and limited by slower generation tools. This was one challenge that ultimately set us back considerably. TOPR, on the other hand, leverages batch efficiency and can scale more easily, but suffers when its offline data lacks correct examples—especially in harder tasks. This caused TOPR to perform poorly for the more complex prompts. The design of the reward function and the quality of the reference model used to generate training data significantly affect both methods' effectiveness. We observe that supplementing off-policy training with higher-quality or more diverse positive samples could bridge this performance gap.

Conclusion Our study shows that reinforcement learning can significantly improve LLMs' performance on mathematical reasoning tasks like Countdown. While both RLOO and TOPR outperform supervised fine-tuning, each comes with its strengths and limitations. RLOO yields higher accuracy but demands more computational resources; TOPR offers faster and more stable training at the cost of data quality sensitivity. Future work could explore hybrid approaches that combine the efficiency of off-policy training with limited on-policy updates, or enriching the off policy dataset to be more robust.

Off-Policy Finetuning for LLM Math Reasoning

Althea Hudson
Department of Computer Science
Stanford University
altheah@stanford.edu

Narvin Phouksouvath
Department of Computer Science
Stanford University
narpho23@stanford.edu

Abstract

It is no secret that LLMs often struggle with multi-step arithmetic reasoning tasks; one such task where is is clearly exemplified is the CountDown task. In this project, we investigated whether reinforcement learning (RL) fine-tuning can improve model performance on mathematical reasoning. We compare an on-policy method, REINFORCE Leave-One-Out (RLOO), with an off-policy approach, Tapered Off-Policy REINFORCE (TOPR), and benchmarked both against traditional supervised fine-tuning (SFT) for a mathematical task (the countdown game). Our results show that experimenting with TOPR reward functions, data collection, and hyperparameters, offered great improvement over SFT and baseline TOPR algorithms. Our findings highlight the benefits and trade-offs of different RL paradigms for fine-tuning LLMs on discrete, logic-driven tasks.

1 Introduction

Large language models (LLMs) have demonstrated impressive capabilities in language understanding and generation, but they often struggle with multi-step mathematical computation and other complex reasoning tasks. A good example of such a task is the Countdown puzzle, where the model is given a set of numbers and a target value and must combine the numbers into an equation that when evaluated, yields the target. For example: given numbers (50, 8, 1, 7, 1, 4) and a target 165, the solver must use arithmetic operations to reach the target. Most LLMs struggle to solve the Countdown game due to their lack of step-by-step reasoning capabilities, and we aim to discover whether a model's performance can be improved with reinforcement fine-tuning.

In this project, we explore online and off-policy reinforcement learning as an effective strategy for improving math reasoning in LLMs, focusing on improving performance for the Countdown game. On policy RL samples data online from the current policy, whereas off policy RL uses a static dataset of trajectories. While both have their advantages, of -policy is appealing because we can leverage both successful solutions and failures from a prior model or dataset, and learn from both positive and negative examples. For our off policy algorithm, we implemented the **Tapered Off-Policy REINFORCE (TOPR)** algorithm, which enables off policy learning for LLMs without the need for KL penalties. Moreover, for a task with well-defined correctness criteria like math problem solving, off policy RL lets us directly optimize against an exact, rule-based reward (correct vs. incorrect) rather than a subjective preference model. We compare our TOPR implementation with RLOO (Reinforce Leave one out) (an on policy algorithm).

Our goal is to experiment with off policy RL for improving math reasoning capabilities, using Countdown as the benchmark. We use a 0.5B parameter Qwen 2.5 model as the base and finetune it with a variant of TOPR. We tried multiple different reward functions/importance algorithms for TOPR, and compare these approaches against both an SFT baseline and an on policy RL baseline (RLOO).

2 Related Work

Fine-tuning LLMs on curated datasets of human demonstrations or highly-rated outputs is a highly effective technique for alignment. Ouyang et al. (2022) showed that supervised fine-tuning on human demonstrations could significantly improve instruction following (as in InstructGPT), and further gains were achieved with RLHF using human preference models. However for domains like math, obtaining fine-grained human preferences is less straightforward, as correctness is binary and often requires domain knowledge to judge. In addition, many Direct Preference Optimization (DPO) Rafailov et al. (2024) has been proposed as an RL free alternative that converts preference data into a supervised objective, but it still relies on a preference model or labeling strategy. In math reasoning tasks where an objective ground truth is available, a verifier-based approach can be used instead of human preferences: for example, Cobbe et al. (2021) trained a verifier model to rank multiple solution candidates and pick the correct one, dramatically improving accuracy on math word problems. Such verifier-guided methods highlight the benefit of using correctness signals; however, they typically require generating many candidates and training a separate judge, and they do not directly train the model to get the answer right on the first try.

Another issue with many online reinforcement learning algorithms (such as PPO) Zheng et al. (2023), is that they are computationally and resource intensive, often requiring multiple models to be stored (policy, ref, reward, and critics). The advantage of The REINFORCE Leave-One-Out (RLOO) algorithm Ahmadian et al. (2024) is that it is slightly more efficient, requiring less memory as it does not need to load as many copies of the model into memory. Generally, RLOO is a variance-reduced policy gradient method that uses multiple sampled outputs per prompt to compute a baseline by leaving one sample out. By averaging gradients from several samples, RLOO reduces variance and was shown to outperform PPO and other baselines in aligning LLMs to preferences. RLOO essentially makes better use of each query by learning from comparisons among its own generated answers, rather than requiring an explicit value function, which in turn makes it more efficient. In our work, we use RLOO as an on-policy RL baseline for math reasoning.

Despite successes, on-policy RL methods still face challenges with discrete reasoning tasks. They require careful management of the trade-off between exploring new solutions and exploiting known good ones. They also typically discard low-reward trials (or at least do not specifically reinforce what not to do, beyond a baseline adjustment). Moreover, in tasks like Countdown, obtaining rewards does not require human feedback but rather a programmatic verifie (a function to check solution correctness). This opens the door for off-policy, verifier-based fine-tuning: one can generate a large set of model attempts, label them with a rule-based reward (correct or not, plus intermediate formatting checks), and then train the model on this data. Off-policy reinforcement learning for language models has recently gained attention as a way to reuse and leverage fixed datasets of model behavior. The TOPR algorithm by Roux et al. (2025) is particularly relevant: Tapered Off-Policy REINFORCE (TOPR) uses a importance sampling to speed up learning while maintaining stable learning dynamics, without the use of KL regularization. This method allows fine-tuning completely offline on a collection of (prompt, output, reward) samples. When demonstrated on math reasoning benchmarks (GSM8K, MATH), TOPR shows that that leveraging both correct and incorrect solutions in training can improve accuracy while also being more training-efficient.

3 Method

3.1 Task Setup and Dataset

We focus on the Countdown numbers game as our benchmark task. Each example provides a list of integers and a target number, and the model must output a sequence of arithmetic operations using the given numbers to reach the target. We formatted each prompt as a natural language instruction (e.g., "Using the numbers [50, 8, 1, 7, 1, 4], create an equation that equals 165."), and expected the model to respond with a clear, well-structured equation.

For supervised fine-tuning (SFT), we used the `cog_behav_all_strategies` dataset from Hugging-Face, which includes completions demonstrating various problem-solving strategies (e.g., systematic search, backtracking). For RL fine-tuning (RLOO and TOPR), we used a separate pool of Countdown puzzles without reference answers. These were held out from the SFT data.

3.2 Data Collection

For off-policy training, we generated a dataset of solutions using the SFT model. Each problem in this dataset was sampled multiple times ($n = 2$) with temperature-based decoding (temperature = 0.15, top-p = 0.9). We then filtered out duplicates and ran a rule-based reward function on each sample to label it. This resulted in an offline dataset consisting of tuples of the form (prompt, completion, reward). Interestingly, we found that many output answers received low or negative reward due to invalid formats or incorrect answers.

For on-policy training with RLOO, data was collected dynamically: at each training step, the current model generated multiple completions for a batch of prompts. These were scored by the reward function, and the RLOO gradient update was applied immediately based on leave-one-out baselines. This dynamic data collection allows exploration but is computationally more intensive and sensitive to reward variance.

3.3 Reward Function Design

As mentioned before, Countdown uses a specific reward function. The reward function makes several checks, which include:

- Whether the model outputs a valid arithmetic equation.
- Whether the equation uses only the given numbers, each at most once.
- Whether the result matches the target exactly.

Although the Countdown task has its reward function, the rewards are sparse and lack meaningful reward signals. So, we wanted expand on the reward function and see if small changes could improve the performance of the model.

In Reward Function A, we used the following reward scheme:

- $+1.0$: Fully correct and valid solution.
- $+0.1$: Validly formatted but numerically incorrect solution.
- $+0.1$: Invalid solution (e.g., using an unavailable number).
- $-0.0003\times$ word count: Applied to completions that failed to produce any answer.

This is largely similar to the original reward scheme, except for the length penalty for missing solutions. The length penalty is a simple heuristic meant to discourage the model from making redundant lines.

In Reward Function B, we used the following reward scheme:

- $+1.0$: Fully correct and valid solution.
- $+0$: Validly formatted but numerically incorrect solution.
- -0.1: Invalid solution (e.g., using an unavailable number).
- -0.2: Applied to completions that failed to produce any answer.

The philosophy behind this reward scheme is based on the following beliefs:

1. Completions without solutions should be penalized the most as having no solution always scores a 0. A no-solution response should not be penalized less for being longer, as an early stop without an answer is also bad.
2. An invalid solution earns the format score of 0.1, but it should not be treated as equal to a solution that has all the correct numbers.
3. A numerically correct solution is usually very close to the correct solution in that the differences can be resolved by rearranging the numbers and using the correct operators.

These reward signals are incorporated during TOPR training. The gradient from each sample is scaled accordingly to encourage behaviors that maximize correctness while discouraging invalid or empty completions.

3.4 Supervised Warm Start

Before applying reinforcement learning algorithms, we fine-tuned the base model (Qwen 2.5 - 0.5B) on the `cog_behav_all_strategies` dataset using a standard next-token prediction loss. This supervised fine-tuning (SFT) phase helps the model learn the syntax and structure of Countdown-style reasoning, serving as the initialization for both RLOO and TOPR fine-tuning. The objective loss function for SFT is given by:

$$\mathcal{L}_{\text{SFT}} = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i \mid x_i) \tag{1}$$

3.5 On-Policy RL with RLOO (Baseline)

As a baseline, we implemented REINFORCE Leave-One-Out (RLOO), a variance-reduced on-policy method. For each prompt, the model samples multiple outputs, scores them using the reward function, and computes each sample's gradient with respect to the average reward of the other completions. This allows learning from self-generated comparisons without requiring a separate value network or KL regularization. RLOO was trained concurrently with TOPR for comparison under matched hyperparameters and reward signals. The reward function used here was:

$$\hat{R}_i = R_i - \frac{1}{k-1}\sum_{j\neq i} R_j \tag{2}$$

These adjusted rewards weight the negative log-likelihood loss of each sampled trajectory:

$$\mathcal{L}_{\text{RLOO}} = -\frac{1}{k}\sum_{i=1}^{k} \hat{R}_i \log P(y_i \mid x) \tag{3}$$

3.6 Off-Policy RL with TOPR

For our core approach, we implemented Tapered Off-Policy REINFORCE (TOPR) using the SFT model. This algorithm uses importance-weighted updates based on a static offline dataset. To avoid instability from large importance weights, we used a tapering strategy: weights above a threshold are clamped, preventing gradient spikes from rare or outlier samples.

The TOPR update is computed by weighting the gradient of each sample according to its reward and importance weight relative to the current model's probability of that sample. In contrast to standard REINFORCE, which uses raw importance sampling, TOPR's tapering ensures more stable and controlled training over long sequences.

Our objective function is largely the same as in the TOPR paper, except that we compute the importance weighted loss for examples with sub-1 reward, not just for negative reward. This change is mainly due to the fact that Roux et al. (2025) used a simple 1 and -1 reward for correct and incorrect responses. Because the Countdown task uses only non-negative rewards, using the original TOPR objective function verbatim would not make sense. We modified the object so that the importance weighted computation is for examples where the score is less than 1. Our objective for TOPR is as follows:

$$\nabla J(\pi) = \mathbb{E}_{x,y:R(x,y)=1}\left[R(x,y)\log\pi(y|x)\right] + \mathbb{E}_{x,y:R(x,y)<1}\left[\left[\frac{\pi(y|x)}{\mu(y|x)}\right]_0^1 R(x,y)\log\pi(y|x)\right] \tag{4}$$

where x is a prompt, y is a completion, R is the reward function, μ is the reference policy, and π is the current policy that is being updated. On the left term, a vanilla policy gradient is computed for completions with positive rewards. On the right term, importance weights clamped between 0 and 1 are included in the policy gradient calculation. The clipping of importance weights is used to minimize gradient variance. Separating positve and negative examples into two terms allows for the model to quickly learn positive trajectories while slowly but steadily unlearn negative rewards.

Prompts and completions are synthetic data obtained from a reference model (in this case, our trained SFT model)

4 Experimental Setup

4.1 Datasets

For SFT, we used `Asap7772/cog_behav_all_strategies` dataset from HuggingFace. This dataset contains a set of query-completion pairs which we can do token-wise log-likelihood finetuning on. Although not noted anywhere, only about half the examples are correct (i.e. the solution scores a 1).

For RLOO, we used the Countdown dataset. This dataset is comprised only of target-number pairs (an integer and a list of integers), so we have to process these ground truth values into a prompt. The dataset itself contains around 47000 examples, but given the time and compute constraints, we only worked with a small subset of around 1000 examples

For TOPR, we sampled 1000 examples from the countdown dataset and ran inference on it with our SFT model to generate 1000 corresponding completions. Since we are both computing the likelihoods and the rewards of these completions, we store both the query-completion pairs and the ground truth target-numbers pairs. For the generation configuration, we used temperature = 0.15 and top-p = 0.85.

4.2 Metrics and Inference

To evaluate our models, we used the Countdown reward function, which gives a 1 for correct solutions, a 0.1 for incorrect solutions (format score), and a 0 for no solutions.

To evaluate our models during training, we took a small subset of examples from the Countdown dataset (64 examples) that was not part of the training set. Then, at validation, we would use the current model to generate completions for each prompt and calculate the return for those completions, taking the average as our validation metric. We used the recommended generation parameters of temperature = 0.6, top-p = 0.95, and top-k = 20.

For inference, we initially used a temperature of 0.15 and top-p = 0.9. In later experiments, we added a top-k parameter of 30.

4.3 Evaluation

For evaluation, we used a held-out test set of Countdown puzzles disjoint from the training pool. Each model (SFT, RLOO, and TOPR) was prompted with the same puzzles, and answers were generated using controlled sampling settings (temperature = 0.15, top-p = 0.9, top-k = 30). These outputs were scored using the same rule-based reward function used during training.

Our primary evaluation metric was accuracy – the percentage of test puzzles for which the model generated a valid equation that exactly matches the target number. In addition, we report average reward across completions, which accounts for both correctness and formatting adherence.

This dual-metric approach allows us to distinguish between models that generate better-formatted answers (higher reward) versus those that truly solve the puzzle (higher accuracy). All evaluation was done without reference solutions, relying entirely on the verifier logic to assess correctness, ensuring consistency between training and testing protocols.

4.4 Training Techniques

To simulate larger batch sizes, we used gradient accumulation. To improve stability, we also clipped gradients to a max norm of 1.0. For SFT, we included a cosine annealing with 500 warmup steps.

5 Results

5.1 Quantitative Evaluation

Our experiments showed that TOPR can be an effective way to improve upon an SFT model and be a serious alternative to RLOO. Notably, we found that the length penalty of Reward A was able to make a significant improvemnt over the SFT model. Unfortunately, Reward B proved to be too unstable and ineffective to improve SFT, quickly collapsing to 0.1 evaluation reward. Interestingly, using

Table 1: Results On Original Test Set (200 examples)

Algorithm	**Average Score**
SFT	0.3385
RLOO	0.2140
TOPR with Original Reward	0.2270
TOPR with Reward A	0.4060
TOPR with Reward A+B	0.4849

Table 2: Results On Harder Test Set (1000 examples)

Algorithm	**Average Score**
RLOO	0.129
TOPR with Reward A	0.2656
TOPR with Reward A+B	0.1458

Reward B TOPR after using Reward A led to additional improvement. We denote this algorithm as TOPR with Reward A+B.

Unfortunately, these models fared poorly against the harder Countdown test set. RLOO and TOPR with Reward A+B performed similarly. Interestingly, TOPR with Reward A performed better the best out of these three, perhaps suggesting that it has better generalization.

5.2 Learning Rates Experiment

In one experiment, we tested different learning rates for TOPR. Specifically, we experimented with learning rates 1e-4, 1e-5, and 1e-6 and kept the rest of the hyperparameters constant (effective batch size of 64,

The results of this experiment showed that using 1e-4 worked best. The other learning rates converged sooner, suggesting that using too small of a learning rate could cause the model to get stuck in a local minimum. We experimented with larger learning rates as well, such as 1e-3, but we found these larger learning rates to cause unstable training.

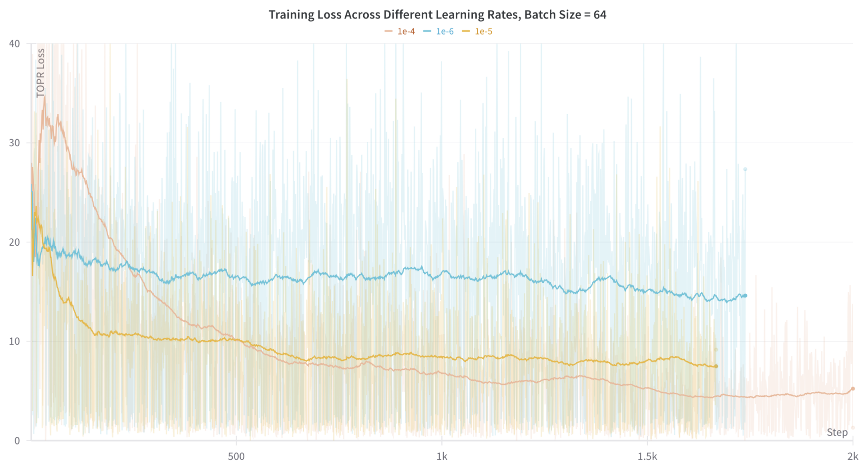

Figure 1: TOPR Training Loss Across Different Learning Rates, Smoothed

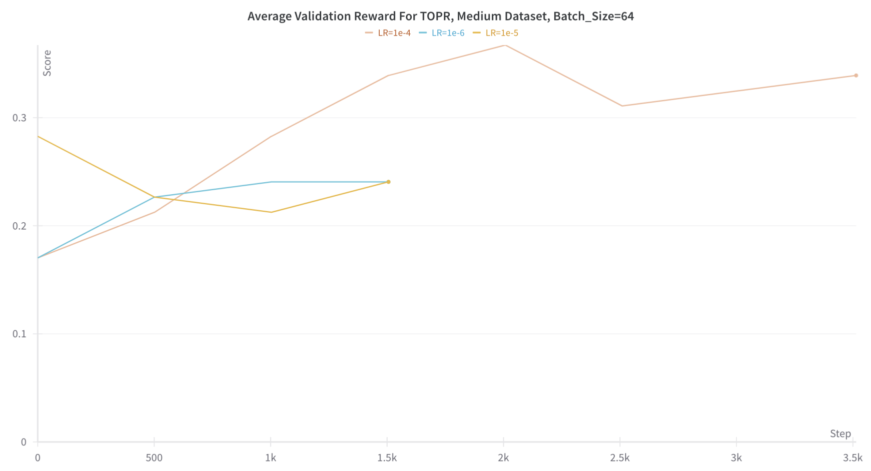

Figure 2: Rewards Across Different Learning Rates. Note, 1e-4 converges later

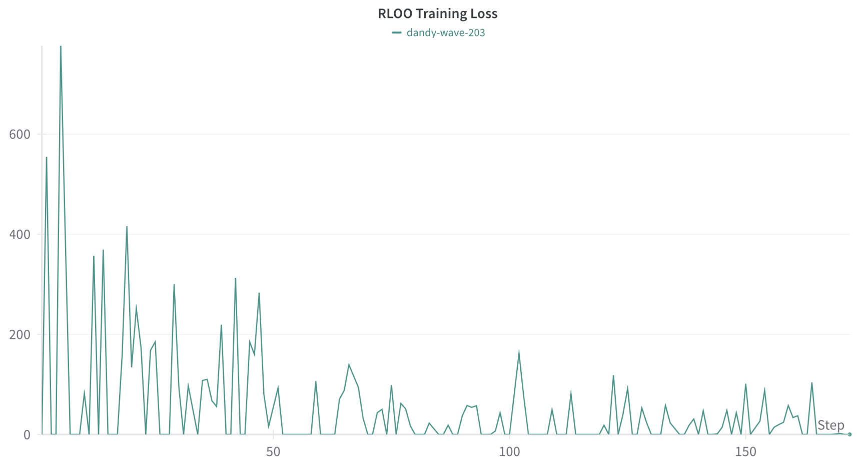

Figure 3: RLOO Training shows high variance

5.3 Qualitative Analysis

With the original heldout prompts, we found that the TOPR model was able to produce a response for every prompt. We believe this is because, with the length penalty, the model was disincentivized to waste tokens on redundant generations (e.g. repeating equations) and incentivized to output an answer, thus being able to get at least the format score. Interestingly, when we ran this model with the harder heldout prompts, this ability to answer every prompt went away. We found that the model failed to work with very large numbers, often doing the math incorrectly, and terminating midway through the generation.

After training the model on the larger dataset with Reward Function A, we found that while the model more consistently provided a solution, it still failed to answer large number questions successfully, often doing the math incorrectly and thus leading it to return solutions which were not mathematically correct. We suspect this is because the larger dataset lacks correct responses for large number examples. This is because the dataset obtains data from our SFT model, which is not able to do solve mathematical expressions with large numbers. Intuitively, this behavior makes sense. Knowing what wrong answers look like does not give you a good sense of what a right answer looks like, especially when there are a finite set of right answers and infinitely more wrong answers.

Experimenting with different effective batch sizes, we found that for TOPR specifically, it was crucial to have a high enough batch size to minimize variance during training, but a low enough batch size to improve generalization. Using an effective batch size 32 seemed to have worked the best.

6 Discussion

6.1 RLOO vs TOPR

From our experience, we found that TOPR was significantly easier to train than RLOO. It had less variance, making hyperparameter tuning easier to do. It trains and converges faster, so it was easier to debug and know when to stop training. It also was more space efficient. With RLOO, to generate $k = 16$ samples, you could only accomodate a batch size of 1 on an Nvidia l40s GPU. With TOPR, you could use vLLM or similar libraries for fast LLM inference to generate training examples. Due to the on-policy nature of RLOO, using vLLM was not practical, so you had to use the much slower Huggingface model.generate(), which took on average 20 seconds per batch generation of $k = 16$ completions.

With TOPR, the fundamental flaw is its reliance on the off-policy data. Where RLOO can continuously produce different responses that can each give unique reward signals, TOPR can only rely on what training data the model currently has. To help the model with the harder held-out countdown prompts, we tried creating responses with our reference model to use for TOPR training. However, because the reference model could not generate correct responses for the harder prompts, the training data lacked positive examples. This is problematic for two reasons:

1. There are infinitely many incorrect solutions but finitely many correct solutions. The model cannot generate a positive example with negative samples alone.
2. TOPR objective function is meant for the model to learn positive examples quickly and negative examples slowly. Thus, with data mostly being negative, convergence is very slow.

There are several possible ways to ameliorate this. One is to use a better reference model to generate data. The issue with this is using a better finetuned model to finetune a weaker model for the exact same task does not make sense and defeats the point of these experiments. Another way is to use human data. Supplementing the synthesized data with human-curated positive examples can help jumpstart the model. However, obtaining human data is expensive. A third option is to generate more examples per prompt. This is very similar to RLOO, which samples multiple completions from the same prompt to get a better estimate of the average return.

Both these models depend heavily on having a good reference model to start training. Minimally, the reference model should be able to produce correct responses. The more often it is able to produce correct responses, the better the reward signals are.

While our results for RLOO were underwhelming, we would not conclusively say that RLOO is inferior to TOPR, as our experiments on RLOO were limited, given our resource and time limitations.

6.2 Other Extensions Attempted

We briefly attempted to implement tool use, but found this to be too difficult to finish with the limited time we had between deadlines.

6.3 Possible Next Steps

Curating a larger, more diverse, and more challenging dataset is a natural next step for this project. Incorporating curriculum learning could also be a way to improve performance. Exploring even more reward functions, or using a neural network to learn one, would also be a natural continuation of our experiments.

7 Conclusion

In conclusion, we sought to understand how RL could be used to improve baseline SFT for mathematical reasoning tasks. To investigate this, we implemented one on policy (RLOO) and one off policy (TOPR) algorithm, using a supervised fine tuned model as our baseline. Much to our surprise, we found that the TOPR algorithm performed slightly better than the online algorithm, though both performed better than SFT. That said, we encountered difficulties implementing RLOO with limited compute resources, and also saw many of the shortcomings with the static nature of data in off policy algorithms. Going forward, we would look to curate a larger and more robust dataset for TOPR, and experiment with other reward functions.

8 Team Contributions

- **Narvin:** Implemented dataloader, SFT, RLOO, TOPR. Ran some experiments. Worked on written deliverables.
- **Althea:** Worked on dataloading, evaluation, and TOPR scripts. Ran experiments with RLOO and TOPR reward functions. Worked on written deliverables

Changes from Proposal Originally, we set out to do data augmentation for both ultrafeedback and countdown. After the milestone changes, we chose to do only countdown and decided on tool use. Tool use took too much time, so we went with off-policy methods.

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

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. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168 [cs.LG] https://arxiv.org/abs/2110.14168

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL] https://arxiv.org/abs/2203.02155

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] https://arxiv.org/abs/2305.18290

Nicolas Le Roux, Marc G. Bellemare, Jonathan Lebensold, Arnaud Bergeron, Joshua Greaves, Alex Fréchette, Carolyne Pelletier, Eric Thibodeau-Laufer, Sándor Toth, and Sam Work. 2025. Tapered Off-Policy REINFORCE: Stable and efficient reinforcement learning for LLMs. arXiv:2503.14286 [cs.LG] https://arxiv.org/abs/2503.14286

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. 2023. Secrets of RLHF in Large Language Models Part I: PPO. arXiv:2307.04964 [cs.CL] https://arxiv.org/abs/2307.04964