Extended Abstract

Motivation Many reasoning benchmarks for language models focus on abstract logic or symbolic manipulation, but relatively few evaluate a model's ability to solve concrete, multi-step arithmetic problems with verifiable outputs. We focus on the Countdown task, where models must combine a given list of integers using basic arithmetic operations to reach a target number. This problem requires not just numerical accuracy but also structured, interpretable reasoning and output formatting. Our goal is to explore how Qwen 2.5 .5B can be trained to solve such problems effectively by combining supervised fine-tuning, reinforcement learning, and synthetic data augmentation. Beyond improving performance on this task, we aim to better understand how different training strategies help models learn to reason, recover from failure, and generalize complex arithmetic patterns.

Method Our method consists of three stages: supervised fine-tuning (SFT), reinforcement learning using REINFORCE Leave One Out (RLOO), and synthetic data augmentation. We begin by fine-tuning Qwen 2.5 .5B on a warm start dataset of examples formatted with reasoning and answer tags, training it to mimic structured solutions in a chain-of-thought style. We then apply RLOO to further improve performance by sampling multiple completions per prompt, computing a reward based on correctness and formatting, and optimizing the model to prefer generations with higher rewards. To enhance both stages, we introduce synthetic data: examples generated by prompting GPT-3.5-Turbo with (numbers, target, expression) triplets and asking it to generate a CoT-style response which eventually reaches the expression. This includes responses that succeed immediately, pivot after a failed attempt, or iterate through multiple incorrect approaches. We additionally extract "frontier" examples, problems the current model can solve inconsistently, and mix them into RLOO training to focus learning on the boundary of the model's capabilities.

Implementation We implement our system using the Hugging Face Transformers and vLLM libraries. Supervised fine-tuning is performed on a single A100 GPU using mixed bfloat16 precision, gradient checkpointing, and dynamic padding to manage memory efficiently. We tokenize inputs using the Qwen tokenizer, masking the prompt portion of each sequence so that the model learns only from the reasoning and answer spans. For RLOO, we generate multiple rollouts per prompt using nucleus sampling and temperature scaling, compute rewards using a custom verifier that checks both format and correctness, and apply the REINFORCE update with a leave-one-out baseline to reduce variance. Synthetic data is generated by prompting GPT-3.5-Turbo with arithmetic expressions and structured instructions, then post-processed to ensure valid formatting. We use vLLM for fast inference during evaluation, stopping generation at the </answer> tag and extracting the final answer using regular expressions.

Discussion After supervised fine-tuning (SFT), the model learns to emit the required `<think>/<answer>` scaffold. Format accuracy jumps from 0.3% (raw Qwen) to 69% after five SFT epochs and stays saturated thereafter. Raw arithmetic skill, however, remains modest (exact accuracy = 0.32). A RLOO pass raises reward from .357 to .575. Blending in synthetic prompts pushes the SFT performance to .4, and when using this base model and incorporating frontier examples into RLOO we reach an average reward of .586. Exact accuracy follows the same trend, peaking at 58 % while tag fidelity stays above 70%. Frontier-style synthetic prompts consistently amplify the gains of RLOO. Countdown's reward is extremely sparse—every roll-out earns 0 or 0.1 until a single exact expression pushes the reward to 1. The leave-one-out baseline centres this spike by subtracting the within-prompt mean, yielding a low-variance gradient even when only one of the k=8 samples scores non-zero. Injecting frontier prompts increases the frequency of such informative spikes, so each mini-batch carries a stronger learning signal without additional compute.

Conclusion Our results highlight that small, mid-scale language models can attain strong, verifiable reasoning. Masked SFT lifts a the base model from near-zero performance to produce consistently formatted responses and 32% exact accuracy. Synthetic data augmentation improves exact accuracy to 37%, by exposing the model to novel operator patterns. Leave-one-out RL (RLOO) delivers the decisive leap: a few hundred training steps raises mean reward to 0.59 and pushes exact accuracy to 58 %, with tag fidelity still 70 %. Mixing in just 20 % "frontier" prompts—cases the policy previously only occasionally solved—provides an additional, albeit modest, boost during RLOO.

Enhancing Mathematical Reasoning Capabilities Through Frontier Examples

Ohm Patel
Department of Computer Science
Stanford University
ohmpatel@stanford.edu

Will Healy
Department of Computer Science
Stanford University
whealy@stanford.edu

Klara Andra-Thomas
Department of Computer Science
Stanford University
klaraat@stanford.edu

Abstract

We address Countdown – a multi-step arithmetic reasoning task in which a model must combine a small multiset of integers into an expression that exactly matches a target value. Naïve language models routinely generate ill-formed or numerically invalid answers; we therefore propose a three-stage, data-centric fine-tuning pipeline for a 500 M-parameter $Qwen - 2.5$ model: (i) Masked supervised fine-tuning (SFT): training on a warm start dataset while masking instruction tokens yields format-fluency (ii) Frontier-driven synthetic augmentation: mining prompts that the SFT model solves only sporadically, generate diverse verifier-approved solutions, and add new demonstrations (iii) Reinforcement learning with a leave-one-out (RLOO) baseline: Using the exact arithmetic checker as reward (+1 correct, +0.1 well-formed) and a low-variance advantage estimator. Overall, our results show that careful data augmentation plus a lightweight variance-reduced policy gradient can more than halve the error rate of a mid-scale model on strict mathematical reasoning, closing much of the gap to far larger systems.

1 Introduction

The Countdown dataset is a logical reasoning dataset used to test the reasoning capabilities of LLMs. The inputs of the dataset are a list of integers and a target integer. The goal is to use basic arithmetic operations (+, -, /, *) to combine the integers in a way which reaches the target. This paper investigates various fine-tuning frameworks for increasing the performance of Qwen 2.5 0.5B on the countdown dataset. Our work is composed of three sections: supervised fine-tuning, Reinforce Leave One Out (RLOO), and synthetic data augmentation.

Supervised fine-tuning is done to convert our base model into a model which understands the basic structure of the task, how to approach finding the solution in a chain-of-thought (CoT) manner, and knows the format the answer in a way which our scoring function understands. This has been shown to increase the performance of LLMs in logical reasoning tasks by teaching them to mimic how a human expert would approach the problem Gandhi et al. (2025) . We first apply SFT to the warm start dataset. This dataset has question answer pairs, where the question explains the task, gives the numbers and target, and uses a few-shot approach to explain how to format the answer. The answer then explains a coherent reasoning pattern, finds the solution, and puts the solution in the correct format. We finetune on this by minimizing the perplexity of the answers in the test set.

The next step section of our research focuses on taking this finetuned model and using reinforcement learning to improve it further. Specifically, we apply RLOO to the model to optimize it further.

The extension of our project focused on using synthetic data augmentation to attempt to improve our model further. Our method for utilizing synthetic data augmentation were heavily influenced by (Dong, Ma, 2025). We had two primary goals in our synthetic data augmentation. The first is to enrich the warm start dataset with a more diverse set of responses. Specifically, we wanted the model to become better at trying one approach, failing, and learning from its mistake to better inform its next approach. The second goal was to extract frontier examples. Frontier examples are examples which the model is capable of answering correctly but still struggles with. Our hope was that by identifying these frontier examples and incorporating them into RLOO, it would be possible to push the model to the edge of its capabilities, enabling it to learn step by step rather than wasting time on overly easy or overly difficult tasks.

By experimenting with different approaches of incorporating synthetic data augmentation in both SFT and RLOO, we hope to gain a better understanding of how to generate, use, and interpret synthetic data intelligently.

2 Related Work

We lean on existing research in both reinforcement learning (RL) and natural language processing (NLP). To motivate our SFT approach, we take inspiration from (12 in spec). They show that models with existing reasoning capabilities improve more from RL than those which don't. They go on to show the behaviors exhibited by the model (how it approaches a problem) matter more than their baseline performance. This highlights the efficacy of using SFT on a dataset which emphasizes Chain of Thought (CoT) reasoning. The methods we use for synthetic data generation were in line with research (Puerto et al., 2025). This paper produces evidence that finetuning LLMs on diverse reasoning chains enables them to learn more general reasoning skills and memorize specific patterns of the correct answers it sees less.

Bai et al. (2022) address these challenges by generating model-driven critiques and revisions of outputs according to predefined principles, augmenting the training data without additional human preference labels. Similarly, Lee et al. (2024) synthesize preference labels through AI feedback, scaling reward modeling efforts without relying on human comparisons. Both works focus on augmenting feedback signals through critiques and synthetic preferences, rather than expanding the diversity of prompts or tasks themselves.

Our approach for incorporating our synthetic data into RLOO was heavily influenced by (Dong and Ma, 2025). This paper aimed to increase the ability of LLMs to prove mathematical conjectures by training it on increasingly difficult conjectures. The difficulty of the conjectures was measured by the frequency with which the current model could prove them. Thus, they iteratively extracted frontier examples based on the current model performance, used them for training, and then re-extracted frontier examples.

3 Method

Our method consists of three stages: a SFT stage, a RLOO stage, and a synthetic data augmentation stage. We evaluate the success of our models on a held out countdown test set. Our main metric rewards 1 point for a correct answer, .1 points for an correctly formatted wrong answer, and 0 points for an incorrectly formatted answer.

3.1 Basic SFT

To warm start our model, we run supervised fine tuning on the warm start dataset. We initialize from the 500 M-parameter **Qwen-2.5-0.5B** language model and teach it to produce fully worked solutions

to Countdown puzzles by minimizing token-level cross entropy. Each example is linearised as

$$\texttt{<bos>}\ \underbrace{\texttt{prompt}}_{\text{numbers, target}}\ \texttt{<sep>}\ \underbrace{\texttt{solution}}_{\texttt{<think>...</think> <answer>...</answer>}}\ \texttt{<eos>},$$

where <sep> is a newly added delimiter token. During training all tokens up to and including <sep> are masked in the loss vector, steering the model to imitate *only* the reasoning chain and final expression while leaving the instruction prefix untouched. In practice the procedure gives the model "format fluency" after a single epoch—it almost never omits, re-orders, or mis-spells the required tags—while preserving its general-language competence.

Given a prompt–solution pair, we minimise cross-entropy on the solution span. Supervised fine-tuning maximises the likelihood of the solution tokens while ignoring the prompt:

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

Equation (1) is the standard SFT / Pref-FT objective used in RLHF pipelines. In practice we realise the masking by assigning the label -100 to every token up to and including the delimiter <sep>; the optimiser therefore receives no gradient signal for the instruction prefix. Sequences are truncated at 2 048 tokens. We optimise (1) for five epochs with AdamW (lr $= 1\times10^{-5}$, batch = 2) using mixed bfloat16 precision on a single A100-40 GB GPU. Dynamic padding, alignment to multiples of eight tokens, and gradient checkpointing keep peak memory below 24 GB. Perplexity on a 2 % validation split is logged every 100 updates. The result is a model that preserves its pre-trained linguistic competence while concentrating its capacity on emitting syntactically valid solutions. Within one epoch the model already achieves *format fluency*, i.e. it rarely omits or mis-orders the required <think> / <answer> tags.

3.2 RLOO

We aim to improve the performance of our model by taking our fine tuned model and applying REINFORCE Leave One Out. Starting from the SFT checkpoint π_{SFT} we treat expression generation as an on-policy RL problem. We draw prompts from the Countdown-Tasks-3to4 corpus. Given a prompt x we sample k continuations $y_1, \ldots, y_k \sim \pi_\theta(\cdot\,|x)$ and evaluate them with the verifier reward

$$R(x,y) = \begin{cases} 1.0 & \text{if the parsed <answer> expression equals the target} \\ 0.1 & \text{else if the output obeys the required "<think>/<answer>" format} \\ 0 & \text{otherwise.} \end{cases}$$

To reduce gradient variance we adopt the leave-one-out baseline: for the i^{th} sample in a mini-batch the advantage is

$$A_i = R_i - \frac{1}{k-1}\sum_{\substack{j=1\\ j\neq i}}^{k} R_j.$$

The RLOO surrogate loss therefore becomes

$$\mathcal{L}_{\text{RLOO}}(\theta) = -\frac{1}{k}\sum_{i=1}^{k}\mathbb{E}_{y_i\sim\pi_\theta}\left[A_i \log \pi_\theta(y_i \mid x)\right],$$

which we minimise with AdamW. Gradients are accumulated over 8 forward–backward passes to obtain an effective batch size of 8 × 1 = 8 prompts.

We instantiate π_θ with the 500 M-parameter **Qwen-2.5-0.5B** checkpoint. We load weights in bfloat16 and enable PyTorch automatic mixed precision. The original Qwen tokenizer is used, padding on the left; the <eos> token serves as <pad> when absent. Prompts are truncated at 256 tokens and continuations at 1 024 tokens. We use the AdamW optimizer with learning-rate 1×10^{-5} and $\varepsilon = 10^{-8}$. No scheduler or weight decay is applied. Mini-batches contain one prompt; gradients are accumulated over eight steps (*effective* batch $= 8$) before each optimiser update. Training roll-outs use temperature 0.3 and nucleus sampling with $p = 0.95$; log-probabilities are divided by the same temperature before softmax. Metrics (loss, reward moments, token log-probability) are appended to CSV every five optimisation steps. Weights and tokenizer snapshots are written every 50 steps.

3.3 Synthetic Data Augmentation

3.3.1 Generation

As an extension to our project, we attempt to improve performance by generating our own data. As a first step, we generate our (numbers, target, expression) triples randomly. We do this by randomly sampling integers between 0 and 100, randomly sampling arithmetic expressions, combining the numbers in the way specified by sampling the arithmetic expressions, and then recording the outcput as our target. In line with the countdown dataset, we discard any examples where the answer is greater than 100, less than 0, or not an integer. We then go on to textify this dataset. In line with Puerto et al. (2025), we create an extended version of the warm start dataset which incorporates a more diverse set of responses. Specifically, we inject diversity into the number of paths which the response explores before reaching the correct answer. That is, for the 1050 triples which we generate, we prompt the LLM (GPT-3.5-Turbo) with the triple, a randomly sampled response from the warm start dataset, and instructions on how to create a sample answer. We append one of the following strings to the prompts, each with probability of $\frac{1}{3}$

1. IMPORTANT: Solve this problem directly and efficiently on the first attempt. Show clear, straightforward reasoning that leads directly to the correct answer without any false starts or corrections.
2. IMPORTANT: Start with one approach that seems reasonable but leads to an incorrect result. Then realize the mistake, correct your approach, and arrive at the right answer on the second attempt. Show this process of initial error and subsequent correction clearly.
3. IMPORTANT: Try two different reasonable approaches that both lead to incorrect results, showing your reasoning for each failed attempt. Continue trying and reach the correct answer on the third attempt. In your response, you should explain how the results of your failed attempts influence your new approaches. Don't label these attempts, it is supposed to mimic someone thinking out loud. This is to create a dataset for finetuning. Explain your reasoning step by step, you are supposed to teach the model how to try, fail, and iterate. Make sure to leave the answer in the <answer> </answer> tags.

We take the outputs of the LLM to be our model textual answers.

3.3.2 Enhanced SFT

This synthetic textified dataset is meant to inject diversity into the SFT data. The standard deviations of the response length in the warm start and synthetic datasets are 297.5, and 313.9 respectively. In addition to diversity in the length, the hope is that there is also increased diversity in how often the answer pivots from one approach to another.

We concatanate this synthetic dataset with the warm start dataset and re-run SFT on this elongated dataset for 5 epochs.

3.3.3 Enhanced RLOO

We then pivot to enriching the model further by incorporating synthetic data into RLOO and using our enhanced SFT model as our starting point. We mimic the methods of (Dong and Ma, 2025) (on a smaller scale) to generate frontier examples. In that paper, the researchers iteratively generate increasingly difficult problems for their model to solve, attempting to improve its capabilities little by little, not wasting time on questions which are too easy for the model to learn and questions which are to difficult for the model to reasonably get. They run several inferences on questions, and hand pick the ones which their model got right a small but non-zero amount of the time. We aim to reproduce a minature version of this as follows.

We take our model which was fine tuned on only the warm start dataset and evaluate it on a random subset of our synthetic data points. For each data point, we generate k inferences using this model.

We keep only the data points which the model got correct one or two times, of which there were 129. We assume that if the model got it correct most times, it already has the capability to solve this problem with good accuracy. We also assume that if the model got the question right 0 times, then it is likely too hard for this iteration of the model and our compute is better spent training on easier questions [1]. We then run RLOO on a dataset which consists of 80% randomly sampled countdown examples and 20% the frontier examples which we generated. We compare this with an RLOO model which was trained using the raw data with no frontier examples injected.

4 Experimental Setup

To test our models, we run inferences on unseen examples of the Countdown dataset. Since the countdown dataset by default consists of only the numbers and target, we embed the numbers and target into a prompt matching the warm start dataset and feed this to our model. At evaluation time, we set the temperature to .1, top p to .85 and max new tokens to 1024. We then track the number of correct, correctly formatted but incorrect, and incorrectly formatted examples. Our primary metric is our reward, which assigns value to both correct and correctly formatted incorrect examples, with weights of 1 and .1 respectively.

5 Results

We run two versions of RLOO: One which trains on a dataset of 80% normal examples and 20% frontier examples and one which trains on 100% regular examples. We run this for 100 steps, and evaluate the performance on the 200 held out examples. The results of the two methods are seen here:

Category	**Score**
No frontier RLOO (synthetic SFT base)	0.380
frontier RLOO (synthetic SFT base)	0.427

Table 1: Comparison of RLOO performance with and without synthetic data

Since RLOO with the frontier examples appeared to perform better, we went on to train further using this 80:20 combination of regular and frontier examples. We evaluate the model's performance on a held out dataset of 200 examples at each 100 step increment.

Setting	**Avg. Score**	**Score = 0.0**	**Score = 0.1**	**Score = 1.0**
RLOO synthetic + real 100	0.427	93 (46.5%)	24 (12.0%)	83 (41.5%)
RLOO synthetic + real 200	0.571	76 (38.0%)	11 (5.5%)	113 (56.5%)
RLOO synthetic + real 300	0.571	75 (37.5%)	12 (6.0%)	113 (56.5%)
RLOO synthetic + real 400	0.586	72 (36.0%)	12 (6.0%)	116 (58.0%)

Table 2: Evaluation of RLOO-trained models with synthetic + real data

After each step of our methods is added to our model, we experiment on unseen Countdown examples and evaluate using the two stage reward for Countdown tasks. The performance on each part of the reward system is documented below. For SFT models, perplexity has been included.

[1] RLOO is more effective with larger sizes of generated samples due to variance being minimized and more stable gradients with more samples. Experimenting with $k = 4$ and $k = 8$, we noticed better results with a larger k.

Setting	Reward	Validation Perplexity	Format Accuracy	Final Accuracy
Qwen	0.003	-	0.003	0.0
SFT	0.357	1.65	0.690	0.320
SFT+Synthetic	0.400	1.84	0.700	0.370
RLOO-on-SFT	0.575	-	0.704	0.538
Best RLOO-with-Synthetic (Table 2)	0.586	-	0.640	0.580

Table 3: Model Performance on Milestone Countdown Dataset

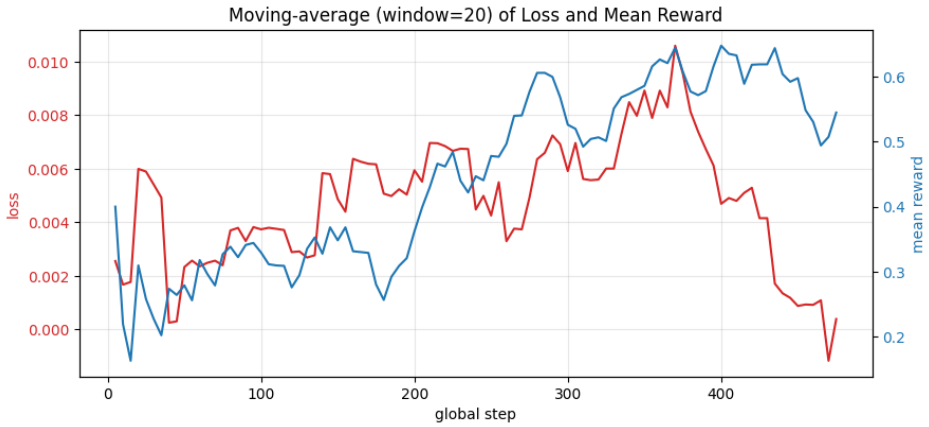

Figure 1: Loss and Reward on RLOO training using Synthetic Data Augmentation

6 Discussion

We will analyze the results based on the improvements and changes in the model behavior as each layer of the research was added. As the methods outline, the approach was to fine-tune the baseline Qwen model using supervised fine-tuning and apply RL using RLOO. In our extension, we tested the effects of incorporating synthetic data enhancement at different steps of the process.

6.1 Supervised Fine-tuning

The most noticeable change in model behavior from the baseline to the SFT stage is a dramatic improvement in the model's ability to follow the expected output structure, specifically generating responses with well-formed <think>...</think> and <answer>...</answer> tags. This shift is reflected in the sharp increase in format accuracy from the base Qwen model to the SFT model, as shown in Table 3. In contrast, subsequent enhancements such as synthetic data augmentation and RLOO provide minimal further gains in formatting, suggesting that basic format compliance is largely learned during supervised fine-tuning. The formatting score ceiling may also be due to the more advanced models simply running out of tokens.

When fine-tuning the model with the synthetic data incorporated in training, we see a jump in final accuracy. When SFT was ran using synthetic and real data, the size of the training set was twice the size of just the real data. While this makes it difficult to analyze how much of the improvement was due to the diversity we injected and how much is due to simply having additional examples, it is clear that there was a strong improvement in out-of-distribution performance from incorporating these synthetic examples.

6.2 RLOO

Building on top of these models with RLOO leads to significant increases in final accuracy, and upholding the trend of a plateauing format accuracy. RLOO proves particularly effective for the countdown task because it tackles two of the hardest challenges of the task at once: the sparsity of

the reward signal and the need for exact arithmetic correctness. Countdown scores remain flat at 0 or 0.1 until the model produces a fully valid equation that evaluates to the exact target, at which point the reward abruptly jumps to 1.0. A vanilla REINFORCE update therefore receives a high-variance gradient: most roll-outs contribute nothing, while an occasional perfect roll-out contributes a large spike. In contrast, the leave-one-out baseline in RLOO subtracts the mean reward of the other $k-1$ samples drawn from the same prompt, so each advantage term is centered around zero. This simple baseline removes much of the within-prompt variance without requiring a learned critic, giving the optimizer a far cleaner signal and allowing it to take stable gradient steps even when only one or two samples in a batch achieve a non-zero score.

The gains from RLOO compound when the policy is warm-started with synthetic frontier data. Synthetic prompts are generated to emphasize strategies and values that rarely appear in the original set. Supervised fine-tuning on these examples teaches the model to imitate those maneuvers, but it does not teach it the logic behind when to apply them. Thus, many of the tricks still fail to hit the target at inference time. In effect, SFT on synthetic data broadens the policy's search region, while RLOO acts as a fast ranking mechanism that can explore different paths in training and effectively learn best strategies to mathematically get to the correct answer more consistently.

Empirically this is clear in Table 3. Synthetic data alone lifts final accuracy from 0.32 to 0.37 by supplying higher-quality demonstrations, but the real jump occurs after on-policy optimization: RLOO on the pure SFT model adds a further 45% relative gain in mathematical accuracy and a 44% boost in reward, pushing the model to a final accuracy of 0.538 while format accuracy remains essentially saturated. In short, supervised fine-tuning teaches the model to speak the Countdown "dialect," and RLOO turns those strategies into consistent, exact solutions.

Looking at the plot of loss versus mean reward during RLOO training, we can see how the learning process unfolds step by step, reflecting the different components of the model's improvement. In the early stages, loss is very volatile, but is generally below the 0.004 mark. As this loss trends upwards, we see the mean reward increase as well. This makes sense, as in the early stage the model has poor performance, nothing different than a simple SFT model. Thus, consistently receiving rewards of 0.0 or 0.1 make the loss minimal, but not vanishing due to multiple samples being generated per example. As the model begins to learn, reward increases more and more. Once it reaches its "peak", we see the same mean reward and diminishing variance, so the loss computation trends to 0, by definition of RLOO.

6.3 Synthetic Data Augmentation

We see clear benefits of synthetic data augmentation in our SFT performance. We also see a subtle increase in performance due to the incorporation of frontier examples. The increased data in SFT is useful due to the limited size of the warm start dataset. By allowing the model to take more steps while keeping the data diverse enough to not overfit, we allow our model to learn more general patterns without memorizing question-specific results.

The results from incorporating frontier examples into RLOO are less dramatic but still meaningful. These examples push the model to train on problems it is close to solving but not yet consistent on, which is where gradient signal is most useful. This aligns with the intuition from curriculum learning and frontier sampling literature: learning is most efficient when focused near the edge of a model's current capabilities.

The ceiling reached by RLOO was slightly higher when synthetic data was incorporated into both the SFT and RLOO stages. The final model trained with synthetic-augmented SFT and frontier-enhanced RLOO achieved the highest overall reward and accuracy, suggesting that synthetic data plays a critical role not just in improving initial reasoning behavior, but also in shaping the regions of the search space that reinforcement learning can effectively explore. However, this performance boost cannot be attributed solely to the frontier filtering step, since the synthetic RLOO model also benefits from starting from a stronger SFT checkpoint. To isolate the contribution of the frontier examples, we ran an additional RLOO experiment using the same SFT plus synthetic base model, but with no frontier examples, instead relying solely on real examples. We ran this for 100

steps and evaluated both models. Table 1 shows that even under this setup, we observed a gain in reward over RLOO trained without the frontier examples, indicating that the benefits of synthetic examples generalize beyond SFT.

7 Conclusion

We present a compute-light pipeline—masked SFT, modest synthetic augmentation, and a single RLOO pass that trains a 500 M-parameter Qwen model to solve Countdown problems with 58 % exact accuracy while retaining 70 % tag fidelity. SFT brings the model from near-zero to format fluency and 32 % accuracy; the extra synthetic demonstrations provide a small but consistent bump to 37 %, mainly by exposing the model to previously unseen operator orders. The bulk of the improvement, however, comes from RLOO: one epoch with a 20 % blend of "frontier" prompts nearly doubles the reward and supplies the exact arithmetic competence that SFT - even with synthetic data -cannot deliver.

8 Team Contributions

- **Ohm:** Ohm led the RLOO implementation and experimentation in the project. He also helped Will and Klara in early stages of SFT and data-loading pipeline, and evaluating different models and methods. helped in SFT, data loaders, RLOO implementation, helped in evaluation scripts
- **Will:** Will led the synthetic augmentation portion of the project. He designed the pipeline to generate synthetic data, textify it with diversity, and extract frontier examples. He also helped in writing the SFT and evaluation scripts.
- **Klara:** Klara worked on and tuned supervised–fine-tuning (SFT) stage. supervised–fine-tuning (SFT) stage. Conducted SFT hyper-parameter sweeps and evaluation toolkit.

References

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022. Constitutional AI: Harmlessness from AI Feedback. arXiv:2212.08073 [cs.CL] `https://arxiv.org/abs/2212.08073`

Kefan Dong and Tengyu Ma. 2025. STP: Self-play LLM Theorem Provers with Iterative Conjecturing and Proving. arXiv:2502.00212 [cs.LG] `https://arxiv.org/abs/2502.00212`

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. 2025. Cognitive Behaviors that Enable Self-Improving Reasoners, or, Four Habits of Highly Effective STaRs. *arXiv preprint arXiv:2503.01307* (2025). arXiv:2503.01307 [cs.CL]

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. 2024. RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback. arXiv:2309.00267 [cs.CL] `https://arxiv.org/abs/2309.00267`

Haritz Puerto, Tilek Chubakov, Xiaodan Zhu, Harish Tayyar Madabushi, and Iryna Gurevych. 2025. Fine-Tuning on Diverse Reasoning Chains Drives Within-Inference CoT Refinement in LLMs. arXiv:2407.03181 [cs.CL] `https://arxiv.org/abs/2407.03181`