CountUP: Improving LLM Reasoning with Reinforcement Learning and Synthetic Data

Team Members: Bruno de Moraes Dumont, Ethan Goodhart

Emails: bdumont@stanford.edu, goodhart@stanford.edu

1 Extended Abstract

Language models have made significant advances in natural language generation and reasoning, yet they continue to underperform on structured, multi-step symbolic problems such as arithmetic reasoning. Existing methods like Reinforcement Learning from Human Feedback (RLHF) often require costly preference data and suffer from reward sparsity and training instability. This project addresses these challenges by exploring whether synthetic data and lightweight reinforcement learning can improve model performance on the Countdown task, a math benchmark requiring compositional arithmetic expressions using a limited set of numbers and operations.

We propose a training pipeline built around four core components: Supervised Fine-Tuning (SFT) on 1,000 high-quality step-by-step examples, a novel synthetic data generation (SD) pipeline, REINFORCE with Leave-One-Out (RLOO) policy gradients trained on the Countdown dataset, and a lightweight Improved Generation (IG) decoding strategy. The synthetic data pipeline is the centerpiece of our approach: it automatically constructs 7,000 verified reasoning traces using symbolic solvers, backward planning, and heuristic distractors, closely mimicking human-authored solutions in both structure and tone. This augmentation addresses the data scarcity issue in mathematical reasoning while scaling supervision without human effort.

Our experiments show that adding synthetic data to SFT leads to a dramatic performance increase, nearly doubling the model's average score on a held-out evaluation set. IG further boosts performance by sampling and reranking completions using a rule-based verifier, often recovering correct answers that are missed by greedy decoding. While RLOO training does not surpass the gains from SFT + SD, it validates the potential of using verifier-based rewards for direct optimization without learned reward models.

These findings suggest that scalable, automated synthetic supervision, combined with targeted training and strategic decoding, can unlock substantial reasoning improvements in small LLMs. Our approach offers a robust alternative to fragile reinforcement learning pipelines and highlights the power of data-centric interventions for advancing symbolic problem-solving. Future work may explore how to better support more complex reasoning steps, but even in its current form, our method demonstrates strong, reliable improvements in compositional arithmetic tasks.

2 Abstract

We investigate how synthetic data and lightweight reinforcement learning can improve arithmetic reasoning in language models, focusing on the Countdown task. Our approach combines supervised fine-tuning, a novel synthetic data pipeline that generates verified step-by-step solutions, reinforcement learning via REINFORCE with Leave-One-Out (RLOO), and an inference-time reranking method called Improved Generation (IG). Results show that synthetic data nearly doubles performance over fine-tuning alone, and IG yields further gains by selecting high-quality completions. While RLOO offers limited additional benefit, our findings demonstrate that scalable synthetic supervision and strategic decoding are effective, low-cost alternatives to complex RL pipelines for symbolic problem-solving.

3 Introduction

Large Language Models (LLMs) have transformed natural language processing, but they continue to show limitations in structured, multi-step reasoning tasks—especially in domains requiring symbolic manipulation, such as mathematics. These shortcomings become particularly apparent when models are evaluated on benchmarks like the Countdown task, which demands precise arithmetic operations under syntactic and semantic constraints. Traditional fine-tuning approaches, including supervised learning and Reinforcement Learning from Human Feedback (RLHF), have been explored to improve LLMs in such domains. However, these methods face challenges such as limited data diversity, sparse reward signals, and unstable training dynamics.

This project addresses these issues by investigating whether synthetic data generation, combined with reinforcement learning and inference-time selection techniques, can significantly improve LLM performance on math reasoning tasks. Our primary objective is to develop a training pipeline that enhances the step-by-step problem-solving capabilities of a small language model (Qwen 2.5 0.5B), using scalable and automated tools. We explore three main strategies: (1) generating synthetic examples that mimic human reasoning; (2) applying REINFORCE with Leave-One-Out (RLOO) baselines to directly optimize task rewards; and (3) introducing an inference-time reranking technique we call Improved Generation (IG).

The central research questions guiding our study are:

- Can synthetic data augmentation significantly improve model performance in symbolic reasoning tasks beyond small, manually curated datasets?
- How effective is RLOO in optimizing correct step-by-step solutions under sparse reward conditions?
- To what extent can inference-time sampling and reranking improve model reliability without additional training?

By tackling these questions, we aim to shed light on data and inference-centric alternatives to reinforcement learning for improving LLM reasoning in constrained, verifiable domains.

To do that, we focused the project on the Countdown problem. The task is a symbolic arithmetic reasoning challenge where the model is given a list of integers $\{x_1, x_2, \ldots, x_n\}$ and a target T The objective is to construct an arithmetic expression using each number at most once and basic operations $\{+, -, \times, \div\}$ such that the expression evaluates exactly to T. Intermediate results must be integers. This task tests the model's ability to reason compositionally over multiple steps, making it a natural fit for exploring structured LLM training techniques.

4 Related Work

Our project builds on a growing body of work exploring reinforcement learning and synthetic supervision to improve language model reasoning, particularly in symbolic and mathematical domains.

STP: Self-play LLM Theorem Provers (Dong and Ma, 2025) introduced a self-play framework for symbolic theorem proving, where an LLM iteratively generated theorems and attempted proofs while using rule-based verification to guide learning. The key insight was that tasks with deterministic verifiability, such as formal logic or mathematics, allow for scalable, automated supervision. STP demonstrated that models can benefit from high-quality synthetic data generated without human labeling. Our project draws heavily from this principle, using a brute-force solver and rule-based verifier to generate and validate synthetic examples for the Countdown arithmetic task. However, unlike STP, which focused on theorem proving in formal logic systems, our task involves free-form natural language reasoning traces for arithmetic problems, requiring additional heuristics to mimic human-like explanations. Moreover, we do not use an iterative self-play mechanism; our data generation is one-shot and backward-guided.

DeepSeek-R1 (DeepSeek-AI et al., 2025) presented a pipeline for enhancing reasoning capabilities in LLMs through reinforcement learning using synthetic chain-of-thought (CoT) traces. These traces were distilled from expert models and used to train smaller models via reward modeling and RL optimization. DeepSeek-R1 showed that high-quality intermediate steps, even when synthetic, could significantly improve downstream reasoning tasks. Inspired by this, we also create synthetic reasoning traces, but instead of distilling them from larger models, we use brute-force symbolic solvers and heuristics to construct reasoning paths. Additionally, our project investigates the use of a lightweight, rule-based reward function (rather than learned preference models) and explores inference-time improvements (IG), diverging from the training-intensive DeepSeek-R1 framework.

5 Methods

Our project explores a hybrid approach to improving arithmetic reasoning in LLMs by combining supervised fine-tuning, reinforcement learning, test-time optimization, and synthetic data. In this section, we describe the core training components used throughout the project. All training was conducted using the Qwen 2.5 0.5B model checkpoint, implemented via HuggingFace Transformers and PyTorch. Unless otherwise noted, all experiments used the AdamW optimizer, a batch size of 1 (due to GPU memory constraints), and a max new tokens limit of 1024. Prompts were masked during training to prevent models from being rewarded for simply copying the input. A light hyperparameter sweep was conducted to tune learning rates and training durations for each stage.

5.1 Supervised Fine Tuning

The supervised fine-tuning (SFT) phase serves as the initialization step for downstream reinforcement learning. It leverages standard teacher-forcing with a next-token prediction loss on paired (prompt, response) examples. Given a natural language prompt x and an ideal completion y, the model is trained to maximize the log-likelihood of the correct response tokens, conditioned on the prompt and the previously generated tokens.

We apply the loss only to the completion tokens; that is, gradients are masked over the prompt to ensure that the model does not learn to copy or overfit to prompt phrasing. This aligns with the standard SFT objective used in language model pretraining. The objective is given by:

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

For this task we used the warmstart dataset, consisting of 1k pairs of queries and completions containing an explanation of the countdown problem and a human-generated step-by-step solution to the specific problem. For this phase, we trained for 5 epochs with a learning rate of $1e-5$ on the entire dataset using a held-out subset of the Countdown dataset for validation.

5.2 REINFORECE leave one out

To build on the supervised initialization and directly optimize the model for correct reasoning, we implemented REINFORCE with Leave-One-Out (RLOO) — an on-policy gradient estimator that uses a per-batch baseline to reduce variance. The key idea is to generate multiple completions for a given prompt x, assign each one a reward using a rule-based verifier, and compute the policy gradient with respect to the advantage over the average reward of other samples in the batch.

Formally, the update rule for RLOO is given by:

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

For this task, we trained on the countdown dataset containing 490k pairs of numbers and targets without a prompt or answer. Since the countdown dataset is massive we trained for only one epoch on a small subset of the data using a learning rate of $1e-6$. Sampling was conducted with $\text{top}_k = 50$. Values for the temperature were tested between 0.7 and 1 to promote trajectory diversity, and the maximum number of completions per prompt K were also varied between 4, 8, and 16. The reward function combined correctness verification with formatting constraints, ensuring that only syntactically valid and numerically correct completions were rewarded.

5.3 Extention: Synthetic Data

A central contribution of this project is the development of a scalable, automated pipeline for generating synthetic Countdown examples that resemble the structure and style of the manually curated warmstart dataset. The core objective of this synthetic data generation process is not only to increase training volume, but also to emulate the reasoning heuristics and linguistic patterns exhibited in human-written examples. This section outlines our procedure in detail, including the solution discovery algorithm, the backward reasoning generation strategy, and the techniques used to inject realistic variability and naturalness into the outputs.

5.3.1 Problem Selection and Solvability Filtering

We begin by sampling Countdown problems from the large Countdown dataset, excluding any prompts used in supervised fine-tuning or evaluation. For each selected instance, defined by a list of integers and a target number, we first attempt to determine whether the problem is solvable within a restricted computational budget. The goal is not to exhaustively search all valid expressions, but to identify a correct solution pathway that can be explained step-by-step in a natural format.

To test for solvability, we use a hierarchical brute-force algorithm that categorizes problems into one of three complexity classes — addition/subtraction only, requires one division, or requires one multiplication — while ensuring tractability. The process operates as follows:

- **Pure Addition/Subtraction Check:** Each number in the list is multiplied by either $+1$ or -1 and summed. We enumerate all 2^n combinations (where n is the number of input numbers) and check if any combination yields the target. If successful, the problem is classified under the addition category.
- **Single Division Check:** If no additive combination works, we test whether the target can be reached using exactly one division. We iterate over all pairs of numbers that are divisible and create a new number from their quotient. The two original numbers are removed and the quotient is added to the number list. We then reapply the addition-only brute-force check to this modified list. If any combination succeeds, the problem is added to the division category.
- **Single Multiplication Check:** If division fails, we repeat a similar procedure for one multiplication operation. A pair of numbers is multiplied and replaced with their product, after which the modified list is subjected to the addition-only brute-force solver. If this succeeds, the example is placed in the multiplication category.
- **Discarding Unsolvable Examples:** Any example that cannot be solved using at most one multiplication or division followed by addition/subtraction is discarded to maintain both tractability and clarity in solution traces. This restricted formulation allowed us to solve over 75

5.3.2 Heuristic Reasoning Trace Construction

Once a valid solution is found, we do not simply translate the final expression into natural language. Instead, we construct a plausible reasoning sequence that approximates how a human might arrive at the correct answer. This is achieved by working backwards from the solution and incrementally building an explanation that satisfies two competing objectives: (1) semantic coherence and (2) alignment with observed warmstart patterns. The reasoning trace generation is split into two stages: **Constructing Multiplication/Division Reasoning Steps**: If the solution required a multiplication or division step, the reasoning trace begins by simulating exploratory operations over the number list. Specifically, we:

- Sample two random pairs of numbers (that are not divided/multiplied in the final solution) and attempt to divide or multiply them.

- If a division yields a non-integer result or a multiplication yields a value exceeding 3× the target, the operation is marked as not useful. Otherwise, it's marked as potentially useful
- Then we divide/multiply the numbers we used on the final solution and mark that as very useful.

Constructing Addition/Subtraction Reasoning Steps: After the core multiplicative or divisive transformation, the problem reduces to a sequence of additions and subtractions. At this point, we:

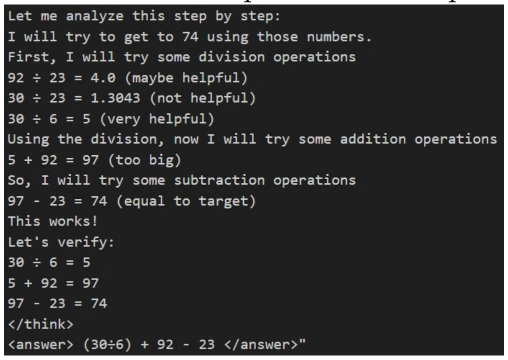
- Initialize a running total by selecting one of the positive components of the final solution.
- If the current value is greater than the target, we try subtracting a number from the list (we subtract a number that is subtracted on the final solution). If its less than or equal to the target we we attempt an addition.
- After all operations we evaluate if the new running total is too big or too small

5.3.3 Finalizing the output

Once the complete reasoning sequence is constructed, we perform a recap phase, where we summarize the key operations and present the final result in a coherent, grammatically polished paragraph. To enhance readability and naturalness, we include transitional phrases whenever switching operation types, and we start the solution with a small introduction. These linguistic cues were modeled after repeated discourse patterns found in the warmstart dataset.

By avoiding rigid templates and instead using structured heuristics, this generation process yields natural, step-by-step explanations that simulate authentic problem-solving behavior while ensuring correctness through verifier-backed construction. Importantly, the variation in phrasing, exploratory steps, and transitional logic mirrors the stylistic diversity of warmstart data without requiring human authorship.

Below is an example of a full response



```
Let me analyze this step by step:
I will try to get to 74 using those numbers.
First, I will try some division operations
92 ÷ 23 = 4.0 (maybe helpful)
30 ÷ 23 = 1.3043 (not helpful)
30 ÷ 6 = 5 (very helpful)
Using the division, now I will try some addition operations
5 + 92 = 97 (too big)
So, I will try some subtraction operations
97 - 23 = 74 (equal to target)
This works!
Let's verify:
30 ÷ 6 = 5
5 + 92 = 97
97 - 23 = 74
</think>
<answer> (30÷6) + 92 - 23 </answer>"
```

5.4 Improved Generation (IG)

Improved Generation (IG) is a decoding-time strategy designed to enhance performance without modifying model parameters. It leverages the observation that language models often generate near-correct responses, and that sampling multiple completions can reveal valid solutions missed by greedy decoding. By combining stochastic sampling with reward-based selection, IG increases accuracy through inference-time reranking.

The process is simple yet effective: for each prompt, the model generates multiple candidate responses using temperature = 1 sampling. Each output is scored using the same compute score function provided.

6 Experimental Setup

We conducted a sequence of controlled experiments to evaluate the effectiveness of our techniques in improving language model reasoning on the Countdown task. Our core goal was to assess how different stages in the training pipeline, Supervised Fine-Tuning (SFT), Synthetic Data (SD), Reinforcement Learning with Leave-One-Out (RLOO), and Improved Generation (IG), contributed to overall performance. To ensure a fair and consistent evaluation, we used a standardized scoring metric and analyzed results across multiple levels of task difficulty and operation categories.

All models were evaluated on both the held-out leaderboard datasets using the official compute score function from the Countdown benchmark. This function assigns a binary reward of 1.0 for perfectly correct completions and a score of 0.1 for syntactically valid expressions that are numerically incorrect. For each model, we compute the average reward across all examples in the test set. To ensure reproducibility, we use consistent prompts across runs, picking the prompt template from the warmstart dataset for all of them. All experiments only used one generation per prompt, except for the ones with the improved generation.

7 Result and Analysis

7.1 First Leaderboard Test Set

The primary experiment evaluates the performance of four model variants on the leaderboard dataset:

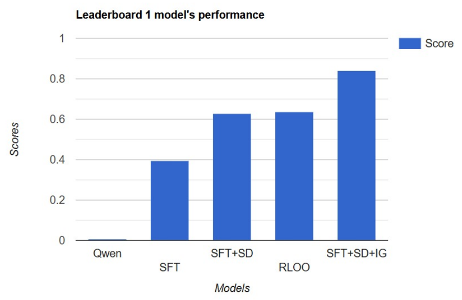


Table 1: Models performance on leaderboard 1

Model	Test Score
Qwen	0.0065
SFT	0.3970
SFT + SD	0.6310
RLOO	0.6380
SFT + SD + IG	0.8425

The base Qwen 2.5 0.5B model performed poorly on the Countdown task, as expected, since it lacks exposure to structured symbolic reasoning during pretraining. Applying SFT using just 1,000 warmstart examples resulted in a large performance jump, validating that high-quality, human-authored reasoning traces can significantly improve arithmetic performance, even in small quantities.

Adding the synthetic dataset to the SFT training nearly doubled the average reward. This highlights the effectiveness of our synthetic generation pipeline: even though the data is machine-generated, its structure and linguistic variation closely mimic the warmstart examples. The diversity and correctness of these 7,000 examples likely helped the model generalize better to new arithmetic prompts.

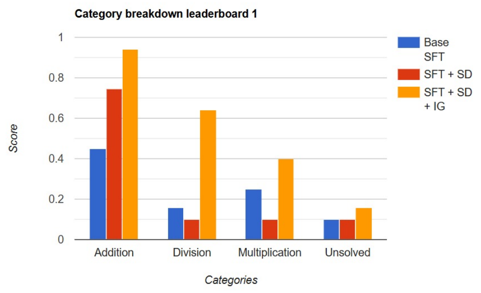
Applying RLOO reinforcement learning after SFT + SD did not yield significant further gains. We attribute this to reward sparsity and optimization instability. Because most outputs earn a score of 0, the learning signal is too weak to improve the model beyond what was already captured during supervised training. Additionally, since we trained RLOO on only a small subset of examples for one epoch, the model likely lacked sufficient exploration to benefit from on-policy updates.

Finally, Improved Generation (IG) showed the most consistent gains across the board. Without modifying the model weights, IG increased accuracy by reranking multiple completions using our verifier. This suggests that many of the model's incorrect greedy completions were close to valid solutions and could be rescued through simple sampling and selection. IG's model, agnostic nature and low overhead make it a highly practical enhancement in settings where RL is fragile or expensive.

7.2 Breakdown by Operation Type

To better understand where performance gains occur, we grouped the leaderboard examples by the operation category required to solve them. These matched the same categories used during synthetic data generation: addition only, requires division, and requires multiplication.

For each model, we computed the average reward within each category. Our results show that:



We observed that SFT + SD dramatically improved accuracy on addition-only problems, likely because these examples dominate both the warmstart and synthetic datasets. Addition steps are also easier to express in natural language and follow more predictable reasoning patterns, making them easier for the model to learn.

However, performance on multiplication and division problems stagnated or even declined. This suggests that our synthetic data may not have captured the complexity of these operations as effectively. Multiplicative and divisive reasoning often involves non-intuitive transformations and more numerical variability, which the model may struggle to internalize from limited, heuristically generated examples. Since the test dataset is composed of 90% addition problems, this discrepancy might also be due to limited data on division and multiplication problems.

By contrast, Improved Generation increased performance in all operation categories, showing its robustness to problem type. Since IG selects completions based on outcome validity rather than process form, it is able to recover correct answers even in harder categories where the model fails under greedy decoding.

7.3 Second Leaderboard

We did the same for the leaderboard 2

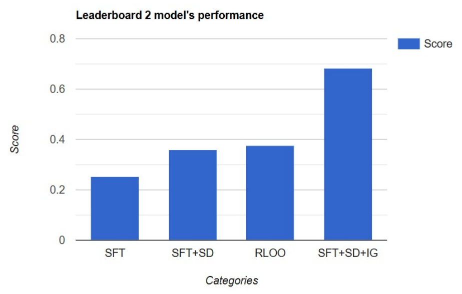
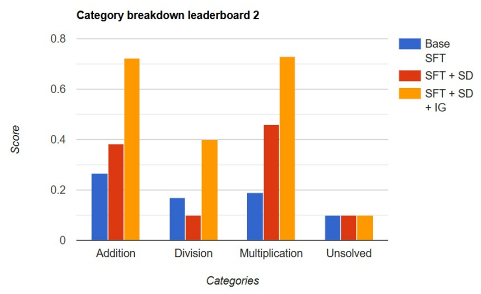


Table 2: Models performance on leaderboard 2

Model	Test Score
SFT	0.2548
SFT + SD	0.3592
RLOO	0.3754
SFT + SD + IG	0.6841



On the second held-out leaderboard, we observed the same pattern as the first. The scores were smaller across the board due to a higher level of difficulty of the dataset, but the findings were consistent with the first set of results.

8 Discussion

This project demonstrated that combining synthetic data with inference-time reranking can significantly improve arithmetic reasoning in LLMs, but several limitations remain. Most gains were concentrated in addition-only problems, with poorer generalization to multiplication and division, likely due to bias in the synthetic dataset and the complexity of these operations. Reinforcement learning via RLOO yielded limited improvements, possibly due to sparse rewards and short training runs. We also faced compute constraints, which restricted batch sizes, training duration, and the number of experiments. Despite these challenges, our results highlight the promise of scalable, verifier-backed synthetic data and decoding strategies as practical alternatives to heavy RL pipelines. However, care must be taken to ensure that synthetic examples promote generalizable reasoning rather than reinforce brittle patterns.

9 Conclusion

Our project explored how synthetic data and decoding-time reranking can enhance the mathematical reasoning capabilities of small language models without the instability or overhead of complex reinforcement learning. By designing a scalable synthetic generation pipeline and leveraging a simple reward-based selection strategy, we achieved strong performance improvements on a challenging arithmetic benchmark. The key takeaway is that well-targeted data and smarter inference can often outperform fragile RL methods in low-resource settings. Looking forward, future work could improve performance on harder operations like multiplication and division by refining synthetic reasoning traces, incorporating learned reward models, or extending our inference strategies with ensemble sampling and self-improving generation loops.

10 Team Contributions

- **Bruno de Moraes Dumont**: Coded both SFT and RLOO, designed and developed the synthetic data pipeline, wrote and presented the poster, wrote the final report, designed and ran all experiments.
- **Ethan Goodhart**: Brainstormed ideas, wrote part of the project proposal, and attempted to increase the performance of the RLOO model.

References

[1] DeepSeek-AI et al. (2025). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv:2501.12948 [cs.CL].*

[2] Dong, K. and Ma, T. (2025). STP: Self-play LLM Theorem Provers with Iterative Conjecturing and Proving. *arXiv:2502.00212 [cs.LG].*

[3] Ouyang, L., Wu, J., Jiang, X. et al. (2022). Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]. *https://arxiv.org/abs/2203.02155*

[4] Wei, J., Wang, X., Schuurmans, D. et al. (2022). Chain of Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]. *https://arxiv.org/abs/2201.11903*