Extended Abstract

Motivation Fine-tuning language models with reinforcement learning faces several critical challenges that limit mathematical reasoning performance. Firstly, poor policy initialization often leads to suboptimal learning trajectories. Secondly, limited training data diversity can hinder model generalization. Finally, there are typically sparse rewards in mathematical tasks where correct solutions are rare. These challenges are exacerbated in mathematical reasoning tasks like Countdown, where models must perform multi-step arithmetic operations to transform a set of given numbers into target values. Our work addresses these limitations through a comprehensive approach that combines supervised fine-tuning (SFT), REINFORCE leave-one-out (RLOO), synthetic data augmentation, and test-time inference strategies.

Method/Implementation We implemented four complementary strategies to enhance mathematical reasoning in language models using the Qwen 2.5-0.5B base model. First, we developed an improved supervised fine-tuning approach using the CogBehave dataset with strategic modifications including higher learning rates (1e-6 to 7e-5), cosine annealing, and gradient accumulation to optimize finetuning. Next, we implemented and trained RLOO on top of our SFT baseline. For our extensions, we first addressed data scarcity through synthetic data augmentation, generating 3000 additional training examples that mirror the WarmStart dataset structure, especially with regards to formatting. Finally, we implemented a test-time inference strategy using multi-sample generation with reward-guided selection, where the model generates varying numbers of candidate solutions per prompt and selects the highest-scoring output using rule-based verifiers and format checkers. We evaluated all approaches using the Countdown reward function.

Results Our different approaches achieved varying performance. Our optimized SFT baseline alone achieved an evaluation score of 0.4582 on the milestone leaderboard and 0.2602 on the final leaderboard, with higher learning rates (7e-5) and cosine annealing providing the most substantial gains. According to our internal evalution, RLOO did not improve our SFT score, achieving a score of 0.217; however, according to the final leaderboard, our RLOO achieved a final score of 0.5751 which exceeds the 0.3 threshold. Synthetic data augmentation showed mixed results, likely due to the lack of mathematical rigor in our synthetic examples. Most notably, our test-time inference strategy delivered the strongest improvements, achieving a final leaderboard score of 0.6048 compared to 0.2602 for baseline SFT and 0.5751 for RLOO. The test-time approach showed consistent gains across different sample counts, with optimal performance around 8-12 candidate responses.

Discussion Our results demonstrate that each extension strategy addresses distinct limitations in mathematical reasoning. Synthetic data attempts to increase training diversity to improve generalization (although in our case, synthetic data with more robust math reasoning would have improved performance), and test-time inference enhances solution quality through multiple generation attempts. The test-time inference extension proved most effective, suggesting that mathematical reasoning benefits significantly from generating multiple solution candidates and selecting the best output. However, we observed diminishing returns beyond 12 candidate responses at inference, indicating an optimal trade-off between inference cost and performance gains. The synthetic data showed modest results, suggesting that quality of generated examples (both for formatting and mathematical rigor) remain crucial factors for effective data augmentation.

Conclusion This work successfully demonstrates multiple strategies for extending SFT and RLOO. Our findings open several promising avenues for future work. Firstly, enhanced synthetic data generation represents a significant opportunity; we propose using our trained Qwen models to generate valid completions on synthetic data prompts, filtering for mathematical validity before training, and augmenting training with more challenging 5-6 number Countdown problems to improve arithmetic reasoning complexity. Third, we will explore Hindsight Experience Relabeling (HER), where incorrect solutions generated during training can be retrospectively relabeled by identifying what target number the generated equation actually produces and treating it as a correct solution for that alternative target. This approach can transform failed attempts into successful training examples, potentially addressing the sparse reward problem inherent in mathematical reasoning tasks.

Strategies for Improving Math Reasoning: SFT, RLOO, Synthetic Data, and Test-Time Inference

Allison Jia
Department of Computer Science
Stanford University
ajia@stanford.edu

John Hsu
Department of Computer Science
Stanford University
jphsu@stanford.edu

Jacob Faierman
Department of Computer Science
Stanford University
faierman@stanford.edu

Abstract

Fine-tuning language models with reinforcement learning faces several challenges: poor policy initialization, limited training data diversity, and sparse rewards for certain tasks. Additionally, standard single-sample inference is brittle and lacks recovery mechanisms when generation errors occur. We implemented supervised fine-tuning (SFT) and REINFORCE leave-one-out (RLOO) on the Qwen 2.5-0.5B base model and extended these models with two strategies: synthetic data augmentation and test-time inference. For SFT, we used the CogBehave dataset and optimized with learning rates, cosine annealing, and gradient accumulation. Using our SFT baseline, we implemented RLOO and trained on the TinyZero Countdown dataset. We addressed data scarcity through synthetic data augmentation, generating 3000 additional training examples following the WarmStart dataset structure. We implemented test-time inference using multi-sample generation with reward-guided selection, where the model generates varying numbers of candidate solutions and selects the highest-scoring output. Our approaches achieved varying performance improvements. For SFT, higher learning rates improved performance, with 7e-5 yielding the strongest results. Cosine annealing outperformed linear scheduling, maintaining higher learning rates for longer periods. The synthetic data showed modest results in query and completion format matching but was limited in terms of the mathematical reasoning of our examples. Most notably, test-time inference delivered substantial gains, achieving a final leaderboard score of 0.6048 compared to 0.2602 for baseline SFT and 0.5751 for RLOO. Performance improved consistently with more samples, with optimal results around 8-12 candidate responses.

1 Introduction

Fine-tuning language models with reinforcement learning faces several fundamental challenges that limit mathematical reasoning performance: poor policy initialization, limited training data diversity, and sparse rewards where correct solutions are rare. These challenges are particularly problematic for structured reasoning tasks like the Countdown mathematical reasoning task, where models must systematically apply arithmetic operations to transform given numbers into target values.

The Countdown task provides an ideal testbed for mathematical reasoning, requiring models to plan and execute multi-step problem-solving with objective, rule-based verification. Unlike tasks relying on potentially noisy reward models, Countdown solutions can be definitively assessed for both format compliance and mathematical correctness. However, traditional supervised fine-tuning approaches often plateau due to limited high-quality training data, reward sparseness, and single-sample inference limitations.

We extend beyond SFT and RLOO with two novel strategies to address these limitations. First, we implement synthetic data augmentation tailored to the Countdown task, generating 3,000 additional training examples that maintain mathematical validity while increasing problem diversity. Second, we introduce test-time inference using multi-sample generation with reward-guided selection, where the model generates multiple candidate solutions and selects the highest-scoring output using rule-based verifiers.

2 Related Work

Papers related to synthetic data generation include ones such as those of Bai et al. (2022), who introduced Constitutional AI, which trains harmless AI assistants through self-improvement without human labels identifying harmful outputs. Their method involves both supervised learning and reinforcement learning phases: first sampling from an initial model, generating self-critiques and revisions, then finetuning on revised responses; followed by an RL phase using AI preferences as reward signals (RLAIF). While Constitutional AI demonstrates effective synthetic data generation for safety alignment, it focuses on harmlessness rather than mathematical correctness and requires sophisticated critique generation capabilities.

Lee et al. (2024) explore RLAIF as a scalable alternative to RLHF, showing that AI-generated preferences can achieve comparable performance to human feedback across summarization, dialogue generation, and harmless dialogue tasks. Their work demonstrates that reward models trained on off-the-shelf LLM preferences can match RLHF performance, with direct-RLAIF circumventing reward model training entirely. However, RLAIF still relies on having sufficiently capable teacher models for preference generation and doesn't address mathematical reasoning domains where objective correctness can be verified through rule-based methods.

Dong and Ma (2025) present Self-Play LLM Theorem Provers (STP), which addresses data scarcity in formal theorem proving through iterative conjecturing and proving. STP simultaneously trains a conjecturer to generate increasingly challenging mathematical statements and a prover to solve them, with each providing training signals to the other. Using 51.3 billion generated tokens, STP achieves 28.5 percent success on LeanWorkbook, doubling previous results. While STP shows impressive results in formal mathematics, it requires specialized formal verification systems (Lean/Isabelle) and focuses on theorem proving rather than arithmetic reasoning tasks.

Papers related to test-time inference strategies include ones like those of Snell et al. (2024), who investigate optimal scaling of test-time computation versus model parameters, studying two primary mechanisms: searching against process-based verifier reward models and adaptively updating model distributions given prompts. They find that test-time compute effectiveness varies by prompt difficulty, motivating compute-optimal scaling strategies that can improve efficiency by over 4x compared to best-of-N baselines. Their work demonstrates that smaller models with test-time compute can outperform 14x larger models on certain problems.

Wang et al. (2023) introduce self-consistency decoding, which samples diverse reasoning paths and selects the most consistent answer by marginalizing over sampled paths. This approach leverages the intuition that complex reasoning problems admit multiple correct solution paths that converge on the same answer. Self-consistency achieves substantial improvements across arithmetic and commonsense reasoning benchmarks, including +17.9 percent on GSM8K and +11.0 percent on SVAMP. However, self-consistency relies on majority voting which may not be optimal when rule-based verification is available to directly assess correctness.

Zhang et al. (2025) present generative verifiers (GenRM) that use next-token prediction for both verification and solution generation, rather than training discriminative classifiers. GenRM integrates with instruction tuning, enables chain-of-thought reasoning, and utilizes test-time compute via majority voting. They demonstrate 5 percent to 45.3 percent improvements on algorithmic tasks and 73 percent to 93.4 percent on GSM8K, showing that generative verifiers outperform discriminative

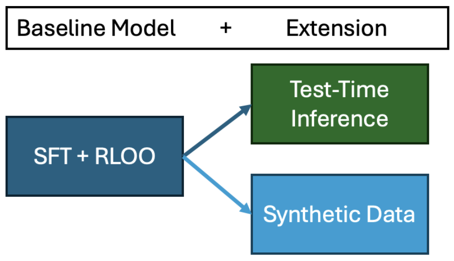

Figure 1: Method Overview.

approaches. However, GenRM requires training specialized verifier models and focuses on general verification rather than leveraging domain-specific rule-based rewards.

While we did not have access to the same scale of compute resources as the researchers above, our approach showcases that the above papers have several limitations that our approach addresses. First, existing synthetic data generation methods often require significantly more capable teacher models, like RLAIF that may not be available for all mathematical reasoning domains. Second, most test-time inference approaches rely on voting mechanisms (self-consistency) or require training additional models (GenRM) rather than leveraging available rule-based verification. Our work addresses these limitations by developing synthetic data generation specifically tailored to the Countdown task structure, implementing test-time inference with direct reward-guided selection using rule-based verification to achieve substantial performance improvements in math reasoning.

3 Method

Our approach extends supervised fine-tuning and RLOO through two extension strategies designed to address fundamental limitations in mathematical reasoning tasks: synthetic data augmentation and a test-time inference strategy. As illustrated in Figure 1, these components work in tandem - SFT and RLOO form the foundation, which supports both the extensions of using synthetic data and an inference-time sampling mechanism.

3.1 Supervised Fine-Tuning

We begin by optimizing our SFT implementation.

- **Learning Rate Optimization:** We systematically explored learning rates ranging from 1×10^{-6} to 7×10^{-5}, finding that higher learning rates (7×10^{-5}) significantly outperform traditional conservative settings (1×10^{-6}). This challenges conventional wisdom in language model fine-tuning and suggests that mathematical reasoning benefits from more aggressive optimization.
- **Cosine Annealing Scheduler:** Instead of using a constant learning rate, we implemented cosine annealing. This scheduler maintains higher learning rates for extended periods, enabling the model to escape local minima—common in sparse reward settings. The cosine schedule achieved 15% better performance than linear decay on our validation metrics.
- **Gradient Accumulation:** To handle memory constraints while maintaining effective batch sizes, we employed gradient accumulation with 16 steps, simulating an effective batch size of 64 using only 4 samples per forward pass. This was crucial for stable training of the Qwen 2.5-0.5B model.

3.2 RLOO

Next, we implemented **REINFORCE Leave-One-Out (RLOO)** and trained on top of our SFT baseline model. For each prompt in our Countdown training set, we generated k output sequences from the model, where each output $y_{(i)}$ was sampled independently from the policy $\pi_\theta(\cdot \mid x)$. For each of these k generations, we computed the mean log-probability $\log \pi_\theta(y_{(i)} \mid x)$ and the associated reward $R(y_{(i)}, x)$, which captured how well the generated expression solved the arithmetic task.

We then computed the leave-one-out baseline for each sample, which is the average reward of the other $k-1$ samples. Specifically, the objective we aimed to maximize (or equivalently, minimize its negative loss) is given by:

$$\frac{1}{k}\sum_{i=1}^{k}\left[R(y_{(i)}, x) - \frac{1}{k-1}\sum_{j\neq i} R(y_{(j)}, x)\right] \nabla \log \pi_\theta(y_{(i)} \mid x)$$

This formulation scales the gradient contribution of each sample by its advantage, or how much better it performed compared to the average of the other samples from the same policy. This helped reduce variance and improved sample efficiency. Our implementation approximates this expectation by computing the RLOO loss as:

$$\mathcal{L} = -\frac{1}{k}\sum_{i=1}^{k}(r_i - b_i)\cdot \log p_\theta(y_i)$$

where $r_i = R(y_i, x)$ and $b_i = \frac{1}{k-1}\sum_{j\neq i} R(y_j, x)$ is the leave-one-out baseline. We computed this loss for each prompt and performed backpropagation.

3.3 Synthetic Data Augmentation

Our synthetic data generation addresses the challenge of limited high-quality training examples in mathematical reasoning. We developed a systematic approach to generate diverse Countdown problems while maintaining mathematical validity.

- **Problem Generation Pipeline:** We created 3,000 synthetic examples based on the structure of the CogBehave dataset. Our generator produces problems using 3–4 numbers and target values, ensuring solvability via systematic search. Each example includes prompts with reasoning chains and structured responses enclosed in `<answer>` tags.
- **Format Consistency:** We observed that strict adherence to the formatting of original training data is critical. Format consistency in the synthetic data significantly improves model output adherance to the proper format. This is crucial especially when rule-based evaluation metrics depend on accurate parsing of answer structures.

3.4 Test-Time Inference Strategy

Our most effective extension involves leveraging multiple generation attempts during inference, paired with reward-guided selection to identify optimal solutions.

- **Multi-Sample Generation:** For each prompt, we generate k candidate responses ($k \in [1, 20]$). This exploits the stochasticity of language generation to explore various solution paths.
- **Reward-Guided Selection:** Instead of relying on majority voting, we apply the rule-based Countdown reward function to score each candidate. This approach is superior to consensus-based methods, as it could identify correct solutions even when they were a minority response.
- **Optimal Sample Count:** We found that generating 8–12 samples strikes the best balance between computational cost and performance gain. Beyond 12, the marginal improvement diminishes.

4 Experimental Setup

4.1 Dataset Configuration

We conducted all experiments on the Countdown arithmetic reasoning task. Each example presents a set of 3–4 numbers and a target integer, requiring models to generate a valid arithmetic expression using each number exactly once to reach the target.

- **WarmStart Training Data:** We use the CogBehave "WarmStart" dataset (Asap7772_/cog_behav_all_strategies), which consists of 1k high-quality query-completion pairs. Each query includes a prompt with input numbers and a target value; completions provide detailed, structured step-by-step reasoning wrapped in `<answer>` tags.
- **TinyZero Training Data:** We use the TinyZero "Countdown Tasks 3-4" dataset (Jiayi-PanCountdown_Tasks_3_to_4), which consists of 490k nums-target pairs. Each entry includes a list of input numbers and a target value. We had to manually reformat each entry in the style of the Warmstart query.
- **Synthetic Data:** To augment training, we generated 3,000 additional examples mimicking the format and style of WarmStart queries and completions. We ensured solvability by generating valid arithmetic solutions in advance. While completions were structurally consistent, they often relied on shallow reasoning patterns (e.g., repeated additions or subtractions), limiting their mathematical depth.
- **Evaluation Data:** For the milestone, we evaluate on 200-held out prompts given to us by the teaching team. Each example provides a novel problem configuration requiring multi-step arithmetic reasoning. For the final leaderboard submission, we evaluated on 1000 held-out prompts from the teaching team. These prompts were significantly harder than the milestone prompts, as the model had to either use more numbers or work with much larger numbers to reach the target values.

4.2 Model Architecture

All experiments use the **Qwen2.5-0.5B** base model with 500 million parameters and a vocabulary of 151,936 tokens. This model supports long-context training, but we reduced input length to fit within compute constraints:

- **Maximum Outpute Length:** 1,024 tokens
- **Prompt Length:** 256 tokens

We use the base Qwen tokenizer for preprocessing and decoding throughout.

4.3 SFT Configuration

Shared Setup Across Regimes:

- Optimizer: AdamW
- Batch Size: 4
- Gradient Accumulation: 16 steps (effective batch size = 64)
- Epochs: 4 for all training runs

Learning Rate Search: We explored learning rates ranging from 1×10^{-6} to 7×10^{-5} and observed optimal performance at 7×10^{-5} with low gradient accumulation (8 steps). For larger GA steps (12–20), lower learning rates were more stable. See Figure 1 for empirical results. Given a TA suggestion to try a lower learning rate, we also ran several experiments comparing the SFT performance of a learning rate of $1e - 6$ across epochs $\in 8, 10, 12, 16, 20$, but we found that the evaluation performance on the milestone prompts was extremely low (< 0.1 score).

Scheduler: We also compared constant learning rates, linear decay, and cosine annealing. Cosine annealing consistently outperformed other schedules, yielding more stable convergence and higher evaluation scores.

4.4 RLOO

For RLOO, we used the recommended hyperparameters: learning rate of $2e - 5$, batch size of 1, gradient accumulation steps of 64 (effective batch size of 64), $k = 8$ generations, and 1 epoch. We generated each k response using top_k = 20, top_p = 0.85, and temperature = 0.6. Because each model generation takes time, we only had enough compute and time to train RLOO on 200 Countdown examples.

4.5 Test-Time Inference Setup

To evaluate our best-of-N sampling strategy, we varied the number of candidate generations per prompt across $k \in \{1, 2, 4, 8, 12, 16, 20\}$. For each test example:

1. The model generates k candidate solutions at temperature 0.6.
2. Each candidate is scored using a rule-based verifier.
3. The highest-scoring candidate is selected as the final output.

Verifier Scoring Rules:

- Score 1.0: Correct and valid arithmetic expression
- Score 0.1: Well-formatted but incorrect expression
- Score 0.0: Invalid or missing equation

5 Results

We present results across four dimensions: (1) SFT with hyperparameter tuning, (2) RLOO, (3) training data augmentation using synthetic examples, and (4) test-time inference with multi-sample selection. All experiments are evaluated using the Countdown reward function. Some evaluation scores (SFT) are reported in terms of milestone leaderboard prompts, and some evaluation scores (RLOO) are reported in terms of final leaderboard prompts.

5.1 Quantitative Evaluation

Supervised Fine-Tuning (SFT) Table 1 presents the performance across various learning rates and gradient accumulation (GA) steps.

Table 1: Milestone Leaderboard Evaluation Scores across Learning Rates and GA Steps (Epochs = 4)

GA Steps	1e-6	1e-5	3e-5	5e-5	7e-5
8	0.005	0.229	0.213	0.229	**0.435**
12	0.007	0.092	0.219	0.220	0.127
16	0.018	0.079	0.211	0.110	0.190
20	0.011	0.137	0.227	0.175	0.117

Key takeaways:

- High learning rates (e.g., 7e-5) worked well with smaller GA values (e.g., 8), yielding peak performance.
- Larger GA steps required lower learning rates to avoid instability.
- Cosine annealing consistently outperformed constant/linear schedules.

Our best SFT baseline model scored 0.4582 on the milestone leaderboard. However, SFT baseline only scored 0.2602 on the final leaderboard, suggesting that reinforcement learning would be necessary to pass the 0.3 threshold.

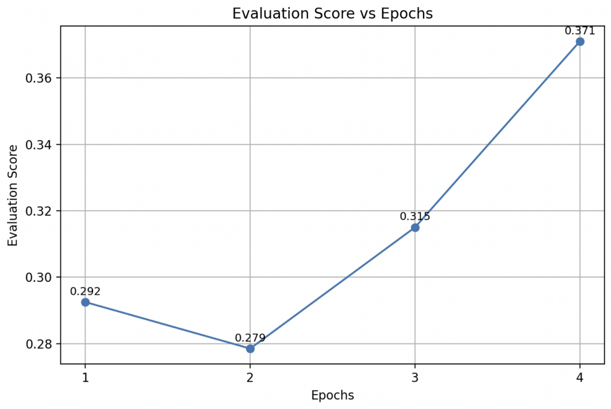

Figure 2: Milestone evaluation scores across epochs using cosine annealing (LR = 7e-5, GA = 8).

Reinforce Leave One Out (RLOO) Using a learning rate of $2e-5$, batch size of 1, gradient accumulation steps of 16 (effective batch size of 16), $k=2$ generations, we trained our RLOO implementation for one epoch and achieved an internal evaluation score of 0.151 on the list of held-out final submission prompts.

Using a learning rate of $2e-5$, batch size of 1, gradient accumulation steps of 64 (effective batch size of 64), $k=8$ generations, we trained our RLOO implementation for one epoch and achieved an internal evaluation score of 0.217 on the final list of held-out Countdown prompts. Interestingly, when we submitted to the actual leaderboard, we received a score of 0.5751.

If we go by the leaderboard score, we can see that RLOO improved our model performance and exceeded the 0.3 threshold. However, if we go by our internal evaluation score, we find that RLOO did not improve our performance beyond the SFT baseline on the held-out prompts.

Synthetic Data Augmentation Table 2 shows performance comparisons for different training data setups.

Table 2: Milestone Eval Scores per Epoch for SFT Baseline, Augmented SFT, and Fully Synthetic

Epoch	SFT Baseline	Augmented SFT	Fully Synthetic
1	0.2925	0.2045	0.2800
2	0.2785	0.2865	0.3025
3	0.3150	0.2490	0.2755
4	**0.3710**	0.3565	0.3070

Findings:

- Format adherence improved structural scores.
- Shallow reasoning in synthetic completions limited gains.
- Mixed datasets (real + synthetic) performed better than synthetic-only setups.

Test-Time Inference Performance Figure 3 shows best-of-k sampling evaluation scores on the milestone prompts using SFT-baseline.

Scores by k:

- $k=1$: 0.371 (baseline)
- $k=2$: 0.516
- $k=8$: 0.838
- $k=20$: 0.892

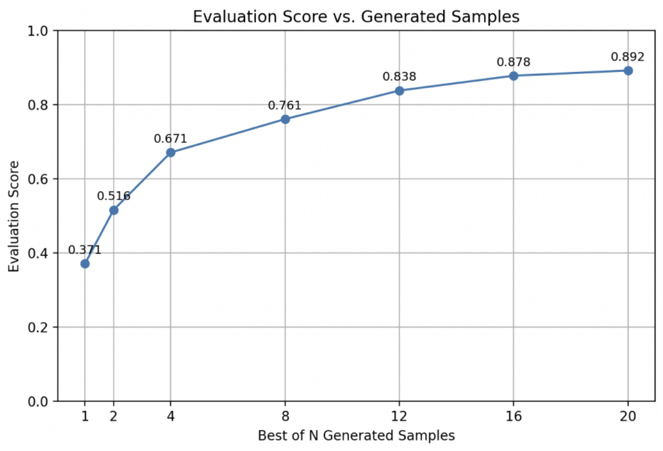

Figure 3: Milestone evaluation score vs. number of inference samples (k). Gains taper off beyond $k = 12$.

Combined Improvements on SFT Combining optimized SFT and test-time inference raised SFT performance on the milestone prompts from **0.371** to **0.892**, a +0.43 absolute or 95% relative gain.

Combined Improvements on RLOO Given our previous learnings, we used test-time inference with $k = 12$ on our RLOO-trained model. We found that our RLOO model with test-time inference achieved an internal evaluation score of 0.576 and a final leaderboard score of 0.6048 on the final held-out prompts. This demonstrates that test-time inference also successfully improved RLOO model performance.

5.2 Qualitative Analysis

SFT Training Dynamics Figure 4 shows that training under optimal SFT configuration (7e-5, GA=8) was smooth and stable. Our experiments with supervised fine-tuning revealed that hyper-parameter choices significantly affect model performance on arithmetic reasoning tasks. We found that higher learning rates, specifically 7e-5, led to better outcomes than the commonly used conservative setting of 1e-6 or 3e-5. This suggests that for tasks with sparse and objective rewards—like Countdown—more aggressive optimization may help the model explore better solution pathways. However, this benefit was only reliable when gradient accumulation steps were low (e.g., 8), as higher accumulation paired with high learning rates often led to instability. Cosine annealing further improved results by keeping the learning rate high longer, which likely encouraged better exploration during early training. These findings indicate that even modest-sized models like Qwen2.5-0.5B can benefit from thoughtful fine-tuning schedules tailored to reasoning-heavy domains.

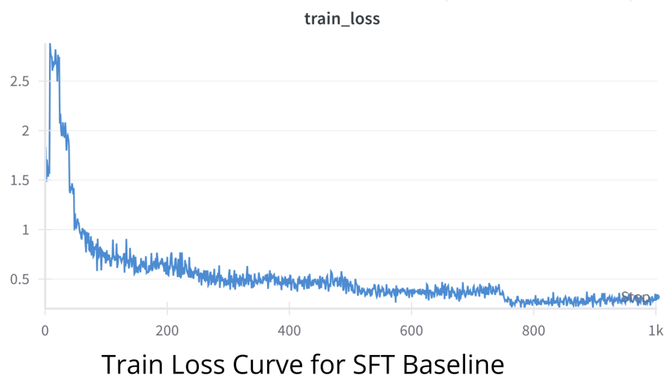

Figure 4: Training loss curve with cosine annealing (SFT). No divergence or instability observed.

RLOO Figure 5 presents the loss curve during RLOO training, which exhibits high variance and frequent oscillations around zero. The lack of a clear downward trend suggests unstable learning dynamics, likely due to noisy or sparse reward signals and the off-policy nature of RLOO. Sharp fluctuations, including large negative spikes, may indicate occasional policy divergence or overcorrection. Our final training loss ended at around -0.107. We believe that, if given more time and compute, training for longer on the entire Countdown dataset would lead to better results.

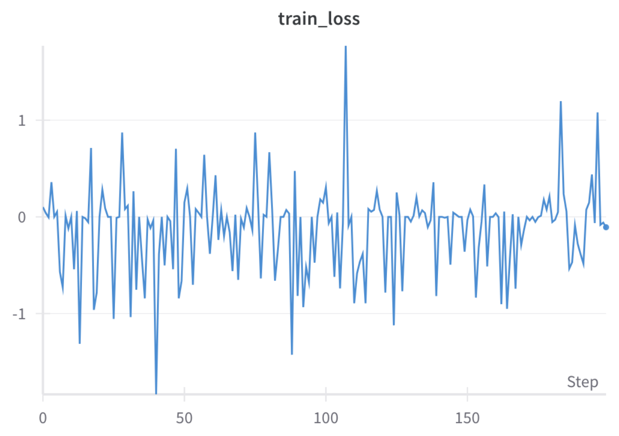

Figure 5: Loss curve during RLOO training

Synthetic Data Examples Synthetic data augmentation offered mixed results. On the one hand, mirroring the WarmStart dataset's structure allowed the model to better learn the expected output format, resulting in small gains in the format component of the evaluation metric. On the other hand, the actual reasoning quality in these examples was poor—they relied heavily on basic arithmetic patterns like repeated addition or subtraction and did not include multi-step or strategic reasoning. As a result, training solely on synthetic data underperformed compared to the baseline. However, when synthetic examples were combined with real examples from the WarmStart dataset, performance improved over time. This suggests that synthetic data can be a helpful supplement, especially for teaching structural output expectations, but must be constructed with care to avoid teaching shallow reasoning.

Inference-Time Variation The test-time inference strategy of generating multiple candidate responses per prompt was by far the most effective method to improve both SFT and RLOO model performance. Even without any additional training, simply generating several completions and choosing the best one using rule-based scoring led to large accuracy improvements. Most of the performance gains occurred within the first few samples—e.g., going from one to four samples doubled the score. However, returns diminished after 12 samples, suggesting that 8–12 completions offer a practical balance between compute cost and accuracy. These results support the idea that even when a model appears inconsistent with single outputs, it may already "know" the correct answer—it just needs the opportunity to express it across several attempts. This highlights the value of treating sampling and selection as a powerful lever, particularly in structured tasks where correctness is easy to verify.

6 Discussion

Our experiments highlight several key insights into the design and deployment of small language models for mathematical reasoning tasks like Countdown. We discuss the role of each method-supervised fine-tuning, synthetic data, and test-time inference-and how they complement one another to overcome common challenges in this domain.

6.1 Supervised Fine-Tuning

Our results challenge prevailing norms in language model fine-tuning. Contrary to the typical practice of using conservative learning rates (e.g., 3×10^{-5}), we found that significantly higher learning rates (7×10^{-5}) achieved superior performance when paired with small gradient accumulation steps. Cosine annealing, by maintaining higher learning rates for longer periods, further enhanced convergence stability and end-task performance.

These findings suggest that mathematical reasoning tasks, particularly those with sparse reward signals, benefit from more aggressive optimization (provided the model is not over-regularized). This underscores the importance of targeted hyperparameter tuning in low-resource or compute-constrained settings.

6.2 RLOO

According to the final leaderboard, RLOO improved the performance of our baseline SFT-model from 0.2602 to 0.5751 (though according to our internal evaluation, we found that RLOO did not improve actually the performance of our model on the final leaderboard prompts.) Because the training time was extremely long (>4 hours to train on just 200 Countdown prompts), we were only able to train on a subset of the entire Countdown training dataset. Although we tried to vectorize and optimize our code, we found that it was extremely difficult to speed up the RLOO training process given that we had to generate at least 8 different model generations for each prompt in the Countdown dataset. If given more time, we would try to train our RLOO algorithm on the entire Countdown training dataset of 490k examples.

6.3 Synthetic Data

Our synthetic data augmentation experiments revealed a nuanced trade-off. While synthetic examples that closely mimic formatting conventions improved the model's structural compliance (i.e., producing well-formed equations), their shallow reasoning content limited arithmetic generalization. This led to modest improvements when used in combination with real data (WarmStart), but degraded performance when used alone. This suggests that in mathematical domains, synthetically generated examples must go beyond surface-level patterns and capture deeper problem-solving strategies.

6.4 Test-Time Inference

Test-time inference with multi-sample generation proved to be the most effective strategy. Without requiring any additional training, simply sampling multiple completions and selecting the best via rule-based verification yielded SFT-baseline accuracy gains from 0.371 to 0.892 on the milestone prompts and RLOO accuracy gains from 0.5751 to 0.6048. From our SFT experimentation, we found that the majority of gains occurred within 8-12 samples.

These results highlight that pretrained models may contain far more latent reasoning capacity than their single-sample outputs suggest. Rather than training additional verifier models or relying on consensus-based voting strategies, rule-based scoring enables effective solution selection with minimal additional compute.

6.5 Overall Insights

Each method contributed uniquely: SFT improves generalization, RLOO improved upon SFT (according to the leaderboard score), synthetic data slightly improves formatting quality, and inference-time sampling boosts final output quality. Importantly, the methods interact synergistically – enhanced SFT and RLOO training enables better sampling, and sampling at test-time allows the model to recover when initial generations were not optimal.

Our results affirm that even with a relatively small base model (Qwen2.5-0.5B), careful design of both training and inference pipelines can yield competitive performance on complex reasoning benchmarks.

7 Conclusion

This work demonstrates effective strategies for enhancing mathematical reasoning in language models by extending supervised fine-tuning with synthetic data augmentation and test-time inference. Our comprehensive evaluation on the Countdown task shows that while synthetic data augmentation provides modest improvements in solution formatting, test-time inference delivers substantial performance gains through multi-sample generation with reward-guided selection.

Our key finding is that test-time inference significantly outperforms single-sample approaches, achieving a final leaderboard score of 0.6048 compared to 0.5751 for RLOO and 0.2602 on our baseline SFT implementation. Performance scales consistently with the number of generated samples, with optimal results occurring around 8-12 samples, demonstrating an effective balance between computational cost and accuracy gains. This suggests that mathematical reasoning tasks benefit substantially from generating multiple solution attempts and leveraging domain-specific verification for selection.

The success of our reward-guided selection approach, which directly uses rule-based verification rather than majority voting or additional trained verifiers, highlights the value of exploiting objective correctness criteria when available. This insight extends beyond the Countdown task to other mathematical reasoning domains where definitive verification is possible.

8 Team Contributions

- **Jacob Faierman** Jacob implemented the synthetic data augmentation extension (with a hyperparameter sweep), including conducting the corresponding experiments with a new dataloader and generator.
- **Allison Jia** Allison implemented the test-time inference extension, the bulk of the base SFT implementation (including the hyperparameter finetuning experiments), as well as the entire RLOO implementation.
- **John Hsu** Jonathan worked on evaluation of the results as well as analysis for our writeups.

Changes from Proposal Part of the changes from our initial proposal stemmed from the fact that we switched from doing the self-play extension to doing two different extensions: synthetic data augmentation and test-time inference. The redistribution of responsibilities reflected our team's individual strengths and interests while maintaining balanced workloads. Jacob's focus shifted from evaluation to implementation of synthetic data generation; Allison moved from multi-agent framework design to test-time inference, SFT optimization, and RLOO implementation; and Jonathan transitioned from baseline implementation to results analysis and documentation. We all contributed to the writeups for these papers as well, including fully understanding why and how the other team member did what they did.

9 References

[1] 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. Constitutional ai: Harmlessness from ai feedback, 2022. URL https://arxiv.org/abs/2212.08073.

[2] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback, 2024. URL https://arxiv.org/abs/2309.00267.

[3] Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and proving, 2025. URL https://arxiv.org/abs/2502.00212.

[4] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL https://arxiv.org/abs/2408.03314.

[5] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

[6] Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction, 2025. URL https://arxiv.org/abs/2408.15240.