Extended Abstract

Motivation Large language models (LLMs) are very sensitive when it comes to arithmetic tasks: a single mis-applied operator can turn a seemingly correct line of thinking into a wrong answer. The Countdown benchmark challenges this arithmetic reasoning directly by aligning success with an exact numeric match under strict formatting constraints. Prior reinforcement-based methods (such as RLOO) improve accuracy but are sample-inefficient and prone to reward collapse. We ask: can we reach comparable or better gains by learning from mistakes offline, without expensive online rollouts or human feedback via synthetic hard negatives? RL literature has primarily focused on synthetic data in a positive reinforcing manner, but negatives offer another unique approach for suppressing common errors. Additionally, what types of negatives have the most effect on boosting model performance?

Method We propose Group Relative Policy Optimization (GRPO) with offline synthetic hard negatives to tackle the challenge of arithmetic reasoning. Our pipeline consists of three stages:

- **WarmStart SFT and RLOO:** Fine tune Qwen-2.5-0.5B on correct Countdown traces to give the model a legal-syntax basis.
- **Offline Synthetic Negative Generation**: Synthetically generate five error buckets (`near_miss`, `arithmetic_slip`, `duplicate_reuse`, `premature_stop`, and `sign_flip`) by prompting a larger model to make invalid traces according to respective failures.
- **GRPO**: For each problem, draw a group of candidate solutions from the current policy, score them with the reward model, and rank them relative to each other. Then adjust the policy to boost higher-ranked traces and suppress lower-ranked ones.

Implementation We built our pipeline on Qwen-2.5-0.5B using PyTorch and HuggingFace transformers, running all stages in FP16 on a single NVIDIA L4. First, we fine-tuned on the WarmStart dataset for one epoch (LR = 1e-6, batch size = 4). Next, we performed a single-epoch RLOO pass sampling $k = 8$ trajectories per prompt (batch size = 2, max length = 1024). Finally, for GRPO we sampled $k = 8$ candidates per prompt, computed token-level similarity against each synthetic negative, and applied a clipped policy-gradient update (clip = 0.2, KL penalty $\beta = 0.1$, LR = 1e-6). Synthetic negatives (200 per bucket) were generated offline via GPT-3.5-Turbo (temp = 0.9).

Results The main results that we got from this project show us that training on hard negatives provide some benefit. Specifically, we trained on the most common hard negative, which was a near miss. This allowed us to try to correct the "majority" or the most common source of errors. Overall, we were able to get a lot of improvement from supervised fine tuning of the model to RLOO. Then, we got some improvement, but not as much improvement from RLOO to the model after the extension.

Discussion First, we saw a lot of improvement from the base Qwen model to the fine tuned version of the Qwen model on the Countdown task using the Warmstart dataset. The GRPO on hard negatives method specifically addressed model weaknesses by suppressing some generation outputs. While RLOO already improved over SFT, we found that GRPO training on structured negative examples helped performance by reducing format errors and improving success. The nature of the hard negatives made training more understandable in terms of what we are solving for, and the offline generation ensured stability without the need for online rollouts.

Conclusion We developed a two-step method to make the Qwen 2.5 0.5B language model better at the Countdown task. Instead of expensive online training, we used three techniques: supervised fine-tuning with WarmStart, online RLOO (Reinforce Leave One Out), and Group Relative Policy Optimization (GRPO). For the hard negatives, our main idea was to create fake "wrong answers" that look like the mistakes the model actually makes. Training on these examples helps the model become more reliable without needing complex reward systems or real-time feedback. Unlike previous methods that require reward models or multiple attempts, our GRPO approach is simpler but still improves accuracy and makes the model's reasoning clearer. We found that certain types of wrong answers, like almost correct solutions with small errors, work especially well for reducing mistakes.

Improving Math Reasoning via GRPO on Offline Synthetic Hard Negative Generation

Arun Moorthy
Stanford University
amoorth2@stanford.edu

Alan Ma
Stanford University
alanpma@stanford.edu

Abstract

Large language models (LLMs) struggle with multi-step arithmetic because a single mis-applied operator can invalidate the entire solution. We tackle this challenge on the Countdown benchmark using Group Relative Policy Optimization (GRPO) trained offline on five buckets of synthetic hard-negative solutions, (`arithmetic_slip`,`duplicate_reuse`, `near_miss`, `premature_stop`, `sign_flip_finale`). Starting from a supervised fine tuned (SFT) and Reinforce Leave-Only-One (RLOO) model, we generate bucket-specific negatives, apply GRPO to negative behaviors. This method of training on hard negatives boosts accuracy on the Countdown task. These findings show the usefulness of targeted hard-negative examples for mathematical reasoning and LLM reasoning at large.

1 Introduction

Despite signficant improvement in language and instruction comprehension of modern LLMs, current models often struggle with multi-step arithmetic reasoning. Under a next-token loss, an error like $8 \times 7 = 54$ only causes a small penalty, so incorrect steps persist. The Countdown problem magnifies this weakness: it requires arriving at an exact numeric target under strict formatting rules, using each of six given numbers exactly once with only addition, subtraction, multiplication, or division, and respecting standard operator precedence. This setup exposes how even sophisticated models can produce chains of reasoning that look convincing but ultimately fail to compute the correct result.

The premise of Countdown serves as a diagnostic for arithmetic reasoning: given numbers $\{x_i\}$ and target T, use each x_i exactly once with any of $\{+, -, \times, \div, \}$, enforcing standard precedence. To help bootstrap performance, the WarmStart dataset provides high-quality chain-of-thought examples for supervised fine-tuning, giving models a strong knowledge base in the correct syntax and basic operation sequencing.

Reinforcement approaches such as RLHF, PPO, or RLOO improve accuracy but require online roll-outs, delicate reward models, and often collapse when rewards are sparse. GRPO has grown popular as a compute-wise cheap solution at boosting model performance. In this project, we ask: can we achieve comparable gains by teaching the model what **not** to do via GRPO on synthetic negatives?

Our answer is an offline fine-tuning pipeline that diagnoses common failure types, synthesizes hard negatives for each of these types, and applies GRPO to prefer correct over incorrect traces. We show that structured hard negatives yield marked improvements while being fully interpretable.

Our approach addresses the need for interpretable and targeted feedback in training. Each synthetic negative is crafted to mirror a realistic mistake (e.g. an arithmetic slip or a premature end to the solution), so the model can learn to avoid those specific errors. This contrasts with black-box reward optimization methods like RLOO or PPO that implicitly learn from negative outcomes but do not

explain what the mistakes were. By structuring the training around known error categories, we achieve both improved performance and better insight into why the model is getting better.

2 Motivation

Large language models (LLMs) can often mess up arithmetic tasks because a single mis-applied operator can turn a seemingly sound derivation into a wrong answer. The Countdown benchmark is a key example of this challenge because success requires an exact numeric match under strict formatting constraints. Prior reinforcement-based methods (such as RLOO) improve accuracy but are sample-inefficient and prone to reward collapse (Zou et al.). RL literature has primarily focused on synthetic data in a positive reinforcing manner, yet negatives offer a unique angle of attack at "suppressing" common errors. Additionally, what types of negatives have the most effect on making model performance better?

3 Related Work

3.1 Reinforcement Learning for LLM Reasoning

Reinforcement learning from feedback has emerged as a key technique for fine-tuning LLMs on complex reasoning and alignment tasks. Proximal Policy Optimization (PPO) is widely used in this context to optimize a policy with a reward signal, often provided by a task-specific evaluator or human preference model. PPO has shown benefits for tasks like mathematical problem solving, as it can directly optimize correctness and format criteria beyond what supervised data provides. However, PPO requires training a separate critic network and can introduce significant overhead and hyperparameter sensitivity.

To simplify RL for LLMs, several recent approaches revisit the basic REINFORCE algorithm. Group Relative Policy Optimization (GRPO) removes the need for a learned value function by using group-based advantage estimation: multiple responses are sampled per prompt and their rewards are compared to compute advantages, which are then used in a clipped-policy update similar to PPO. GRPO has been empirically successful in boosting math reasoning – as seen with the DeepSeek-R1 model – and is noted as an essential algorithm in recent RL training setups (Guo et al.). These methods share the aim of using somewhat binary signals to improve the model. However, a consistent finding is that while such RL methods can improve immediate accuracy, they might not expand the model's fundamental reasoning ability or coverage of problem types. In fact, using only outcome rewards can cause the model to exploit frequent solution patterns and ignore less common reasoning paths. This is known as reward hacking from the models. This potential drawback motivates our exploration of training methods that incorporate more informative feedback than a pure success/fail reward.

3.2 SFT for Math Reasoning

Supervised fine-tuning (SFT) on human-written or high-quality solutions has been a primary approach to enhance LLMs' reasoning. These methods significantly improve performance by exposing the model to the desired chain-of-thought format and correct mathematical workflows. However, the amount of explicit math reasoning data in pretraining corpora for Qwen2.5-0.5B is relatively small and often not easily "learnable". So, domain-specific fine-tuning, especially in this case, helps the model acquire mathematical problem-solving skills. SFT alone can take a model to a decent accuracy level, but it eventually plateaus, often still making mistakes on more complex or uncommon problem instances. Importantly, SFT usually uses only positive examples (correct solutions), as shown with the Warmstart dataset. This means the model learns what correct solutions look like but not explicitly how to avoid producing incorrect reasoning. Our work starts with an SFT WarmStart to reach a reasonable base performance, then addresses this gap by introducing explicit negative examples in further training.

3.3 Hard-Negative Mining in Machine Learning

The concept of hard-negative mining is well-established in areas like representation learning and classification. The idea is to identify the most challenging negative examples—those that the current

model gets wrong or finds confusing—and use them to further train and sharpen the decision boundary. For example, in image classification, one might add misclassified images that looked very similar to the target class as hard negatives so the model learns to tell them apart. Hard negatives are useful because they force the model to confront its weaknesses rather than just reinforcing what it already knows.

In our context, a "negative" is a flawed solution to a math problem. A hard negative would be an incorrect solution that looks correct or logical at first glance, rather than an attempted solution that is clearly incorrect. By training on hard negatives, we push the model to develop more precise reasoning discrimination—it must learn exactly why the correct solution is right and the tempting incorrect one is wrong. We take inspiration from hard-negative mining literature, where the typical strategy is to select negatives that the model currently would mistakenly classify as positive. In other words, we choose errors that the model is likely to make or accept. By adding these to training, the model can correct those specific weaknesses (Xuan et al.). Our work systematically generates such negatives offline. We note that in the same vein, researchers also found that using all negative samples blindly is not necessarily beneficial, and that filtering out extremely bad samples can improve training stability. This underlines the importance of carefully choosing which negative examples to use—precisely what hard-negative mining accomplishes.

4 Method

Our training approach consists of two stages: (1) SFT and RLOO methods to initialize the model's reasoning ability, and (2) an offline GRPO fine-tuning that uses synthetic hard negatives. In this section, we describe the model and data setup, the creation of hard-negative solution buckets, the GRPO algorithm details, and the overall loss and training procedure.

4.1 Model Pipeline

- We begin by fine-tuning Qwen-2.5-0.5B on the Countdown WarmStart dataset. This dataset contains correct chain-of-thought examples that reach a target number using arithmetic operations on a fixed set of integers. Training on this corpus gives the model a solid foundation in generating legal and well-formed math expressions.
- For creating the synthetic hard negatives, we generate examples for a specific type of common error (called "error buckets"). These are created by prompting a larger model (GPT-3.5-Turbo) to intentionally introduce targeted flaws like arithmetic slips or duplicate usage. A checker verifies that each example is both syntactically valid and belongs to the intended error category.
- During GRPO training, the model generates a group of candidate solutions per prompt. These samples are ranked using a reward model or verifier. The policy is then updated to favor higher-ranked responses and discourage lower-quality ones. This relative scoring helps the model learn without relying on absolute rewards or unstable rollouts.

4.2 Why Hard Negatives Matter

In traditional fine-tuning, models are mostly shown correct examples, with little guidance on how to avoid plausible but incorrect responses. As a result, language models often produce outputs that seem valid on the surface—using proper syntax and fluent reasoning—but contain subtle mistakes that undermine correctness, especially in tasks like Countdown where every step of arithmetic must be exact.

Hard negatives directly address this gap. These are intentionally incorrect answers that look correct, designed to mimic very targeted flaws. Unlike random wrong answers, hard negatives challenge the model by pushing it to learn fine-grained distinctions: what makes a particular trace incorrect, and why it shouldn't be selected.

Post-SFT analysis of 200 sampled outputs (run on milestone countdown dataset) from the baseline model identified specific recurring reasoning mistakes. We systematically categorized these errors to form structured "hard-negative" examples, scientifically tabulated as follows:

Bucket	Model failure behavior	What the hard negative teaches
`arithmetic_slip` (most common)	Single fact error such as $8 \times 7 = 54$ passes token-level checks.	Memorizes arithmetic facts rather than pattern-match.
`duplicate_reuse`	Model re-uses a high-probability number twice.	Strict accounting of resource usage.
`near_miss`	Output is within ± 5 of target; gradients are weak.	Prioritize exactness over proximity.
`premature_stop`	Model stops after 2–3 ops—shorter texts have higher likelihood.	Penalize length-based optimization that harms correctness.
`sign_flip_finale` (rarest)	Final operator flip is a one-token change, easily missed in reasoning chains.	Sensitivity to tail-position operator semantics.

Table 1: Error buckets (ordered by frequency) and their role in training.

4.3 Baseline Error Distribution Across Hard Negative Buckets

Among the 200 error output samples, the bucket categorizations were as follows:

Bucket	Failure Share (%)
`arithmetic_slip`	41
`duplicate_reuse`	22
`near_miss`	21
`premature_stop`	9
`sign_flip_finale`	7

Table 2: Share of each error type in the *pre-GRPO*. Values are placeholder estimates pending final counts.

Overall, the error distribution reflects the kinds of mistakes real models make when left to generalize from positive-only supervision. By converting these observed patterns into structured, adversarial training data, we create a feedback loop that is both targeted and scalable. These hard negatives allow the model to internalize what not to do, improving robustness, especially in high-precision reasoning tasks.

4.4 Hard Negative Generation

Our synthetic negative generator takes a correct chain-of-thought trace plus a target error bucket and rewrites the reasoning to introduce exactly that flaw while preserving overall syntax. This teaches the model to spot and avoid specific mistake patterns.

Crucially, these negatives simulate real but subtle model failure modes that would not be effectively penalized under standard next-token loss or even sparse verifier-based rewards. By targeting common error types, such as arithmetic slips or premature stops, this method helps the policy discriminate between correct and slightly wrong solutions, aligning with the Countdown challenge's objective of robust math reasoning under structural constraints. The offline generation also avoids the instability of online policy sampling, making it a stable and efficient extension for large-scale RL fine-tuning.

Prompt Template:

This is the base prompt given to GPT-3.5-Turbo to generate flawed responses. It specifies the desired format (step-by-step reasoning in <think> tags and a final answer in <answer> tags) and asks the model to make a deliberate computational mistake while maintaining plausible reasoning structure.

```
"prompt": "A conversation between User and Assistant. The user asks a question, and the
    Assistant solves it step by step with detailed mathematical reasoning but makes a
    computational mistake. Assistant first thinks through the math, then provides the final
```

```
    answer.\n\nUser: Using the numbers {numbers_str}, create an equation that equals {
    target}. You can use +, -, *, / and each number only once. Show your work in
    <think></think> tags. Return the final equation in <answer></answer>, e.g.
    <answer>(1+2)/3</answer>.\nAssistant:"
```

Response Format:

This illustrates the expected structure of a generated response. The model encloses reasoning steps in <think> tags and the final error-ridden equation in <answer> tags, ensuring the output remains well-formed and parseable for downstream training.

```
"response": "<think>\n...detailed but flawed chain-of-thought...\n</think>\n<answer>...final
    (incorrect) equation...</answer>"
```

Example Synthetic Negative (duplicate_reuse):

A concrete example of a generated hard negative for the duplicate_reuse bucket. The model computes intermediate steps correctly but introduces a duplicate use of 23, leading to a valid-looking yet incorrect final expression. This teaches the model to recognize and correct such subtle numerical inaccuracies.

```
{
  "index": 6,
  "prompt": "A conversation between User and Assistant. The user asks a question, and the
      Assistant solves it step by step with detailed mathematical reasoning but makes a
      computational mistake. The assistant first thinks about the reasoning process and then
      provides the final answer.\nUser: Using the numbers [23, 20, 19], create an equation
      that equals 82. You can use basic arithmetic operations (+, -, *, /) and each number
      can only be used once. Show your work in <think></think> tags. And return the final
      answer equation in <answer></answer> tags, for example <answer> (1 + 2) / 3
      </answer>.\nAssistant: Let me solve this step by step.",
  "response": "<think>\nLet me work through this systematically, using mathematical
      expressions: First, I'll calculate 23 + 23 = 46. Next, I'll divide this by 2: 46 / 2 =
      23. Then, I'll multiply by 20: 23 * 20 = 460. Finally, I'll add 19: 460 + 19 = 479
      .\n</think>\n<answer>(23 + 23) / 2 * 20 + 19</answer>"
}
```

For our GRPO fine tuning process, We utilized 1000 such examples of these negatives, 200 per error.

4.5 GRPO Fine-Tuning on Structured Hard Negatives (Refined)

To fine tune the model's arithmetic reasoning capabilities, we apply Group Relative Policy Optimization (GRPO) that focuses solely on structured hard negatives. Unlike traditional reinforcement learning pipelines that reward correct answers, our method penalizes proximity to these known hard negative outputs. This way of doing GRPO is designed to teach the model what not to do, by suppressing generation outputs that closely resemble the negative examples.

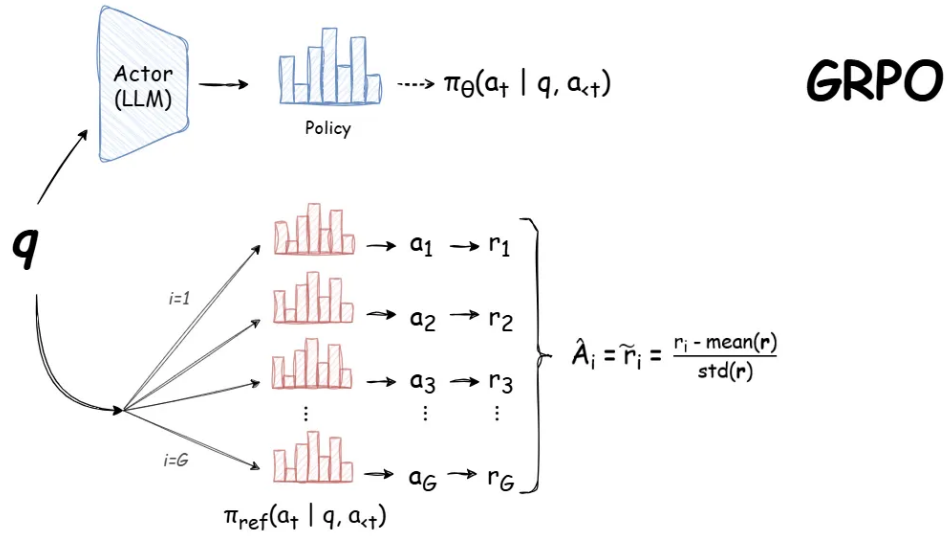

Figure 1: Standard GRPO pipeline (Liang and Hong 2024)

Training Procedure: For each of the 1000 (200 per error bucket) hard negative Countdown prompts q, we query the model to create a group of 8 generated outputs. We then compute a token-level similarity score between each candidate and the negative example that it corresponds to it, producing a ranking from most to least similar. Rather than relying on any ground-truth answer or learned reward model, we use this rank ordering to drive GRPO's policy-gradient update, penalizing the output that most closely mirrors the mistake by minimizing its log-probability under the model.

This negative-only GRPO schema matters because many flawed arithmetic traces look superficially plausible. So, they may be numerically close to the target or follow syntactically valid steps, but they still have subtle errors that supervised fine-tuning won't catch. By explicitly discouraging outputs that exhibit arithmetic inconsistencies, operator misuse, number repetitions, or incomplete reasoning chains, we force Qwen 2.5 0.5B to learn the important boundary between correct and plausible, but wrong responses. This boosts its verifier success without online policies or custom reward functions for the task at hand.

5 Experimental Setup

5.1 Model and Training

We use the Qwen-2.5-0.5B model—a compact 0.5B parameter variant of Qwen2.5—as the backbone for all training. Both supervised fine-tuning, RLOO, and GRPO stages are conducted in full-precision float16. Synthetic data generation is handled externally via OpenAI's GPT-3.5-Turbo API.

Stage 1: WarmStart SFT. The first step of this project was fine tuning the base Qwen 0.5B model on the Warmstart dataset. This corpus consists of verified correct solutions with detailed step-by-step reasoning. The goal is to give the model an initial prior over legal arithmetic sequences and valid syntax. Here, we give the Qwen model examples of great outputs, given a Countdown task prompt.

Stage 2: Online Policy RLOO The next step of the project was the RLOO step, which is the Reinforce Leave One Out method. Here, we take a prompt and give query the model k amount of times. We found that a value of 8 for k was ideal because it was a great balance for memory issues and enough responses for the model to learn.

Stage 3: GRPO. The third step of our experimental setup was running GRPO on the final model after SFT and RLOO. This GRPO was done with hard negatives, meaning that the model is queried with a prompt k times. Then, in the GRPO framework, our reward model is GPT 4o mini. Specifically, with

few shot prompting, we prompt GPT 4o mini to output a similarity score between the outputs that the model produces and the negative example that is in the dataset. Then, we try to minimize this similarity, which is our loss function. So, we run GRPO to try to maximize the difference between the generated outputs from the model and the hard negative examples that are in the dataset.

Hyperparameters Summary:

- **SFT:** 1 epochs, learning rate 1×10^{-6}, batch size 4
- **RLOO:** 1 epoch, 150 samples, batch size = 2, k samples = 8, and the max token response length = 1024.
- **GRPO:** 150 samples, learning rate 1×10^{-6}, KL penalty $\beta = 0.1$, k samples = 8, token limit = 1024, temperature = 0.7.
- **Synthetic Generation:** GPT-3.5-Turbo with temperature = 0.9 to create 200 example per error bucket.
- When querying the model for sample generations, a temperature of 0.2 was used to limit variability amongst outputs.

5.2 Training Configuration

All experiments were conducted on a single NVIDIA L4 GPU with 24GB VRAM. To accommodate memory constraints, some parameters such as the maximum tokens per response had to be changed. Additionally, running it with $k = 16$ could not work because of CUDA memory issues. Overall, this allowed us to fully fine tune Qwen2.5-0.5B within a 24GB memory budget, while maintaining strong downstream performance.

We tracked loss and accuracy metrics using WandB, and saved intermediate checkpoints every 200 steps to evaluate training stability.

The **WarmStart dataset** used during SFT is publicly available and contains one thousand legal reasoning traces, specifically curated for the Countdown math task. These include diverse reasoning strategies (e.g., backtracking, greedy, and chaining), which help bias the initial model toward valid, multi-step arithmetic composition.

5.3 Evaluation Protocol

To assess improvement from fine-tuning, we compared model outputs from the base Qwen-2.5-0.5B model and the fine tuned version using the 200 prompts from the dataset that was given for the milestone. Each prompt was evaluated by correctness of the return value in the response field. Another key thing in this part was creating the parsing script. So, taking a response from the model and converting it to the format that was requested by the leaderboard was really important. Additionally, this caused some trouble, as we had parsing errors in the parsing script. That resulted in a few extra submissions on the leaderboard. Overall, the evalutation protocol was mainly done from the held out prompts.

6 Results

6.1 Quantitative Improvements

Table 3: Countdown Performance: Baseline vs. SFT+RLOO vs. GRPO

Model	Verifier Score (%)
SFT Baseline	37.9 (Milestone Dataset)
SFT + RLOO	30.88 (Updated Dataset)
GRPO (combined categories)	$\approx 31\%$ (Updated Dataset)

GRPO on hard negatives was helpful in terms of eliminating some errors that the model made, as shown in the table. Overall, the GRPO caused a slight increase in accuracy when testing on the held

out prompts for the Countdown task. Compared to the ablation study following this section, we suspect these more difficult prompts had impacts on our synthetic hard negative efficacy.

In terms of the loss functions, the ending loss for the Supervised Fine tuning step was around 0.445. For RLOO, the loss function ended at -0.01. For the GRPO step, the loss function was slightly different because we are minimizing the likelihood of the model producing responses that are similar to the negative examples. Under this setup, the model wants to get as far away from the hard negative outputs as possible, but due to the hard negative outputs being derived from arithmetically sound responses, this may not always be optimal as seen in the verifier score.

6.2 Ablation: Individual Buckets

Training with only one bucket's negatives on *the original milestone dataset* reveals:

- `arithmetic_slip` alone raises exact-match to 41.2%.
- `duplicate_reuse` alone raises to 36.2%.
- `near_miss` alone raises to 42.2%.
- `premature_stop` alone raises to 32.6%.
- `sign_flip_finale` alone raises to 33.8%.

The above improvements for each of the error buckets was calculated on 200 Countdown prompts. Between runs on the original milestone dataset and the newer held-out dataset, we observe that combining the synthetic negatives in GRPO fine tuning yields the best overall gains to model performance.

7 Discussion

Our results demonstrate that introducing specific failure types via GRPO with synthetic hard negatives bolsters mathematical reasoning. However, there are a few different areas where we can improve this model.

7.1 Expanding Negative Buckets

While our five synthetic negative categories captured common mistakes, arithmetic slips, duplicate reuse, near misses, premature stops, and mis-bracketing—other error types still persist. For example, the model may make reasoning leaps (skipping intermediate logical steps), misinterpret problem statements (confusing which numbers to use), or commit complex algebraic slips (such as mishandling nested terms). Incorporating these additional buckets could help address residual errors. One promising direction is iterative hard-negative mining. In other words, as the model corrects simpler errors, we could automatically create new hard negatives through an online generation and add them as fresh negative examples with a given prompt. Over successive iterations, this process would target increasingly subtle mistakes, progressively making the model more robust and less prone to certain errors. Additionally, trying to directly apply response failures during evaluation of the SFT and RLOO models would be another direction we hope to take.

7.2 Integration with Instruction Tuning

Our fine-tuning process was domain-specific and constrained to the math reasoning format. For making this method more broad in terms of application, we must integrate this specialized model back into a general instruction-following framework. A good approach for this is to perform lightweight instruction tuning on a mixed dataset containing both our math-specific fine-tuning data and general instruction-following examples. By including a small weight on general prompts during a secondary fine-tuning phase and leveraging the KL penalty already in place to preserve pre-trained language behaviors, we can help the model maintain its math proficiency while also gaining open-domain instruction following skills. Of course, empirical evaluation will be needed to confirm that hard-negative training does not degrade performance on these other non-math tasks.

7.3 Going Broad: AI Safety

Although our work was on math, the notion of synthetic hard negatives can extend to safety and alignment training. For example, one could generate "hard negative" responses that are subtly toxic or biased and train the model to avoid those. This would be more interpretable than just a reward model that scores toxicity. We see our approach as a form of rule-based RLHF, where the rules are embedded in the training data explicitly. Future work might explore applying this to ensure that models follow ethical guidelines, by creating scenarios where a model might barely violate a guideline and teaching it to do the right thing instead.

8 Conclusion

We present an offline GRPO pipeline leveraging synthetic hard negatives for Qwen2.5-0.5B on Countdown. Compared to SFT (37.9% on the milestone dataset) and SFT+RLOO (30.88% on the final dataset), our combined synthetic negative method achieves (31% on the final dataset), while eliminating many format errors. By targeting common errors via five buckets and using GPT-3.5-Turbo to generate realistic negatives, the model learns to avoid plausible yet incorrect solutions without requiring online rollouts.

Our results suggest that explicitly correcting particular failure types is a good way to extend the model's capabilities, rather than relying on a global reward signal alone. Our results show a slight increase, but we would love to continue this project and explore other error types that these models are making, especially with a task like the Countdown task. One reason our approach is effective is that it sidesteps some difficulties of RLHF. Specifically, we did not need a separate reward model or active human feedback, and training was stable since the data was fixed. GRPO served as a great training framework for implementing this preference-based update, leveraging relative rewards within a group.

The success on math tasks raises the question: can this approach generalize to other domains of reasoning? We believe that it can. A similar pipeline could be applied to logical reasoning, code generation, or even dialogue—first identify common errors (logical fallacies, bugs, toxic replies, etc.), then generate synthetic hard negatives, and finally fine tune with a preference / RL algorithm to avoid those negative examples. This could yield models that are more robust and trustworthy in those domains as well.

9 Team Contributions

- **Arun Moorthy**
  - Implemented the SFT and RLOO fine-tuning pipeline, including reward model training and online policy gradient optimization.
  - Developed online sampling infrastructure for hard negative generation.
  - Ran RLOO diagnostics and assisted with asynchronous negative refreshing, mining schedule tuning, and safety monitoring.
  - Worked with Alan on the project poster and project report.
- **Alan Ma**
  - Created structured hard negative categories tailored for math reasoning (e.g., arithmetic slips, mis-brackets).
  - Designed the contrastive reward signal for Group Relative Policy Optimization (GRPO).
  - Conducted evaluation, ablation studies, and error analysis on bucketed outputs.
  - Worked with Arun on the project poster and project report.

Changes from Proposal Our primary shift was choosing Group Relative Policy Optimization (GRPO) over contrastive Supervised Fine-Tuning (SFT) for training on synthetic hard negatives. While contrastive SFT initially appeared promising for enforcing distinctions between correct and flawed outputs, we found that GRPO provided a more direct and fine-grained reward mechanism. GRPO's groupwise ranking approach allowed us to decrease the log probability of some common reasoning failures without relying on handcrafted contrastive pairs or an arbitrary loss function, ultimately resulting in a more interpretable training loop.

10 References

References

[1] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z.F̃. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, Han Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J.L̃. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R.J̃. Chen, R.L̃. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S.S̃. Li, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. Submitted January 22, 2025.

[2] Janghoon Han, Dongkyu Lee, Joongbo Shin, Hyunkyung Bae, Jeesoo Bang, Seonghwan Kim, Stanley Jungkyu Choi, and Honglak Lee. Efficient dynamic hard negative sampling for dialogue selection. In *Proceedings of the 6th Workshop on NLP for Conversational AI (NLP4ConvAI 2024)*, August 2024.

[3] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023. EMNLP 2023.

[4] Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alexander Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. *Findings of ACL 2023*, 2023. Accepted at Findings of ACL 2023.

[5] Xiao Liu, Xixuan Song, Yuxiao Dong, and Jie Tang. Extensive self-contrast enables feedback-free language model alignment. *arXiv preprint arXiv:2404.00604*, 2024.

[6] Alize Pace, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. West-of-n: Synthetic preferences for self-improving reward models. *arXiv preprint arXiv:2401.12086*, 2024.

[7] Ebrahim Pichka. Group relative policy optimization (grpo) illustrated breakdown & explanation. https://pub.towardsai.net/group-relative-policy-optimization-grpo-illustrated-breakdown-explanation-684e71b8a3f2, 2025. Accessed June 2025.

[8] Haoran Xu, Amr Sharaf, Yunmo Chen, Weiting Tan, Lingfeng Shen, Benjamin Van Durme, Kenton Murray, and Young Jin Kim. Contrastive preference optimization: Pushing the boundaries of llm performance in machine translation. *arXiv preprint arXiv:2401.08417*, 2024. Accepted at ICML 2024.

[9] Hong Xuan, Abby Stylianou, Xiaotong Liu, and Robert Pless. Hard negative mining for multi-view 3d shape retrieval. In *European Conference on Computer Vision (ECCV) Workshops*, volume 12359 of *Lecture Notes in Computer Science*. Springer, 2020.

[10] Xiandong Zou, Wanyu Lin, Yuchen Li, and Pan Zhou. Hard preference sampling for human preference alignment. *arXiv preprint arXiv:2502.14400*, 2025. Version v3; Submitted February 2025.