Week 10: Deep Dive into RLVR with nano-aha-moment

BEH Chuen Yang

Abstract

This report explores the use of RLVR for base model post-training in two parts. Firstly, we explore how RLVR is implemented in the nano-aha-moment repository, and see it in action on a simple task called countdown. Secondly, we adapt the code to train on the GSM8K dataset, a grade school level math problem dataset. We detail any insights and challenges encountered during the process, and present the results of our experiments.

1 Motivation

Much like how pre-training looks deceptively easy (Beh (2025b)), Reinforcement Learning with Verifiable Rewards (RLVR) is another algorithm which promises amazing reasoning capabilities just by training a model to do math/computing/logic problems (which lend themselves well to uncomplicated verification mechanisms) (Shao et al. (2024); DeepSeek-AI et al. (2025); Lambert et al. (2025)).

Indeed, as with pre-training (Beh (2025b)), this overly-simplistic perspective hides a lot of the ugly details that make RLVR work. In this report, we will explore the nano-aha-moment (Kazemnejad et al. (2025)) repository, and in doing so, explore qualitatively how RLVR works (and doesn't) in practice.

2 nano-aha-moment

Apparently continuing in the nanoGPT (Karpathy (2022)) tradition, the nano-aha-moment (Kazemnejad et al. (2025)) repository provides a minimalistic implementation of RLVR using Group Relative Policy Optimization (GRPO) (Shao et al. (2024)). In particular, it follows the setup of DeepSeek-R1 and R1-Zero (DeepSeek-AI et al. (2025)), which uses GRPO with relatively simple mechanisms to train language models on reasoning tasks.

While similar minimalistic repositories for R1-style RLVR exist such as TinyZero (Pan et al. (2025)), nano-aha-moment (Kazemnejad et al. (2025)) distinguishes itself in the following ways [1]:

- It does not offload the RLVR algorithm implementation to complex libraries such as veRL (Sheng et al. (2024); Zhang et al. (2024)) and HuggingFace's trl (von Werra et al. (2020)), instead implementing the GRPO algorithm directly in PyTorch.
- It has a small codebase ($\sim$ 1.5k lines of decently-commented code), which makes it easy to understand how RLVR with GRPO works.
- It is designed to be run on a single GPU.
- As with nanoGPT (Karpathy (2022)), WandB (Biewald (2020)) integration for convenient logging.

These features make it a good analogue to nanoGPT (Karpathy (2022)) for RLVR using GRPO, serving as a pedagogical tool for understanding RLVR concepts, as well as for experimenting with RLVR on (very) small language models.

[1] Unfortunately the repository itself is not updated with robust quality checks, which resulted in a few headscratchers before experiments could be run. We detail some of these issues in Appendix A.

2.1 Training Loop

Unlike in nanoGPT (Karpathy (2022)), where most features have to be implemented from scratch, Kazemnejad et al. (2025) use the HuggingFace (HuggingFace (2025)) library for model training, inference, tokenization, and dataset loading in nano-aha-moment. As such, perhaps the only salient feature of nano-aha-moment (Kazemnejad et al. (2025)) is the training loop, which can be found implemented here (Countdown-3-to-4) and here (GSM8K).

We defer a detailed discussion to the next section, where we first discuss the GRPO algorithm used by Kazemnejad et al. (2025), and then provide a sketch of the training loop used in our experiments in Algorithm 1.

3 More on GRPO

Since the RLVR algorithm implemented in nano-aha-moment (Kazemnejad et al. (2025)) is based on R1-Zero (DeepSeek-AI et al. (2025)), we will first provide a brief overview of the GRPO algorithm (Shao et al. (2024)) used in R1-Zero (DeepSeek-AI et al. (2025)).

3.1 GRPO vs PPO

Consider a policy network π_θ and critic network V_ϕ in a single-state, single-action armed bandit environment (Sutton & Barto (2018); Langford & Zhang (2008)) with state s and action $a = (o_1, \ldots, o_n)$, where o_k is the k-th output token of the language model.

Both PPO and GRPO (Schulman et al. (2017); Shao et al. (2024)) maximize this objective function[2]:

$$\mathcal{J}(\theta) \quad = \mathbb{E}_{(s,o_1,\ldots,o_n)\sim\pi_\theta}\left[\frac{1}{n}\sum_{k=1}^{n}\min\left\{\frac{\pi_\theta(o_k|s,o_{<k})}{\pi_{\theta_{old}}(o_k|s,o_{<k})}A(s,a), g(s,a)\right\}\right] - \lambda D_{KL}(\pi_\theta||\pi_{\theta_{old}}) \tag{1}$$

where $g(s,a) = clip\left(\frac{\pi_\theta(o_k|s,o_{<k})}{\pi_{\theta_{old}}(o_k|s,o_{<k})}, 1-\epsilon, 1+\epsilon\right)A(s,a)$, and $\pi_{\theta_{old}}$ is the policy network from the previous GRPO step.

However, where in PPO with Generalized Advantage Estimation (GAE) (Schulman et al. (2017; 2018)), $A(s,a) = r(s,a) - V_\phi(s)$ is the one-step advantage of the action a in state s, in GRPO (Shao et al. (2024)), $A(s,a) = \frac{r(s,a)-\mu(s,a)}{\sigma(s,a)}$, where $\mu(s,a)$ and $\sigma(s,a)$ are the mean and standard deviation of the rewards for a given batch of state-action pairs $(s_1,a_1),\ldots,(s_r,a_r)$.

Note how in both cases, the advantage $A(s,a)$ is the same regardless of the output token o_k. This is because the entire output is considered a single, composite action.

3.2 Why GRPO?

Where policy gradient algorithms are concerned (Weng (2018)), Proximal Policy Optimization (PPO) (Schulman et al. (2017)) has been a standard choice as it contains many tricks and improvements that serve to greatly stabilize the high-variance training process inherent to RL. Namely, these are:

- Using a value function baseline, approximated by a critic network, to reduce variance in the policy gradient
- Using Generalized Advantage Estimation (GAE) to compute advantages

[2]This is not PPO as used in RLHF by Ouyang et al. (2022), rather the one used in RLVR by Shao et al. (2024). Notably, the KL Divergence term is incorporated into the objective instead of the reward function. We show this formulation of PPO to highlight the fact that, ultimately, GRPO is just a realization of PPO with a different way of computing the advantage $A(s,a)$.

- Using a clipped surrogate objective to prevent large updates to the policy

However, when hundreds of millions, or even billions, of parameters are involved, it can become prohibitive to host the language model, let alone increase the memory footprint by adding a separate critic network. [3]

It is in the context of these compute constraints that Shao et al. (2024) proposed GRPO. Unsurprisingly, GRPO is very similar to PPO (Schulman et al. (2017)) in most respects, except that it does not require a critic network. Instead, Shao et al. (2024) use batch reward statistics to estimate the relative advantage of certain actions (i.e. outputs) over others.

Conceptually speaking, this is a counterintuitive idea. GRPO (Shao et al. (2024)) effectively eschews a high-quality value function baseline in favor of an extremely local and coarse state-value estimate, which should result in destabilized training.

However, Shao et al. (2024) are able to deliver first-rate empirical results when used to train large language models, with DeepSeekMath-7B achieving better Top-1 scores on the MATH benchmark than models up to 10x larger, such as Qwen2-72B (Yang et al. (2024)) and WizardMath-70B (Luo et al. (2023)). [4]

Coupled with the fact that GRPO (Shao et al. (2024)) does not require a critic network and thus allows for reduced memory footprint, it has become a good choice for RL post-training of large language models specifically.

3.3 GRPO in nano-aha-moment

While we do not have access to the source code used in the training of DeepSeek-R1 (DeepSeek-AI et al. (2025)), the nano-aha-moment (Kazemnejad et al. (2025)) repository does provide an implementation which appears to closely follow the methodology described in DeepSeek-AI et al. (2025).

In fact, directly inspecting the source code reveals that the gradient $\nabla\mathcal{J}(\theta)$ is

$$\begin{aligned}\nabla\mathcal{J}(\theta) \quad &= \mathbb{E}_{(s,o_1,\ldots,o_n)\sim\pi_\theta}\left[\frac{1}{n}\sum_{k=1}^{n}\min\left\{\frac{\pi_\theta(o_k|s,o_{<k})}{\pi_{\theta_{old}}(o_k|s,o_{<k})}A(s,a), g(s,a)\right\}\right] \\ &\quad -\lambda\left[\log\left(\frac{\pi_{\theta_{ref}}(o_k|s,o_{<k})}{\pi_\theta(o_k|s,o_{<k})}\right) - \frac{\pi_{\theta_{ref}}(o_k|s,o_{<k})}{\pi_\theta(o_k|s,o_{<k})} - 1\right]\end{aligned} \tag{2}$$

where $g(s,a)$ is as in Equation 1, and $\pi_{\theta_{ref}}$ is the original, pre-trained policy network, and the second line is an unbiased estimator of the KL divergence $D_{KL}(\pi_\theta||\pi_{\theta_{ref}})$ (Schulman (2020)). This is almost identical to the gradient shown by Shao et al. (2024), which strengthens our belief that nano-aha-moment (Kazemnejad et al. (2025)) is a faithful implementation of GRPO (Shao et al. (2024)).

We give a sketch of the algorithm in Algorithm 1, based on directly inspecting its source code. This algorithm will be used in all our experiments in the next section.

4 Experiments

As part of exploring how RLVR works, we want to test just how good nano-aha-moment (Kazemnejad et al. (2025)) is at training language models on reasoning and math tasks.

As such, we will train models on two different tasks before evaluating their Pass@1 performance on the tasks.

[3] Parameter sharing between the policy and critic networks is of course possible. However, this results in a network that tries to optimize for two different objectives at once, which *can* lead to suboptimal performance.

[4] For some hypotheses why, see Beh (2025a).

Algorithm 1 GRPO in Kazemnejad et al. (2025)

Policy network π_θ, pre-trained policy network $\pi_{\theta_{ref}}$, hyperparameters, questions $\mathcal{S}$
Initialize $\theta \leftarrow \theta_{ref}$, $\theta_{old} \leftarrow \theta_{ref}$
while not converged **do**
    Sample m questions $\mathcal{S}_m = \{s_i\}_{i=1}^{m} \subseteq \mathcal{S}$
    Autoregressively generate n answers each for $\mathcal{S}_m$, using π_θ. Define the answers as $\mathcal{A}_{mn} = \{\alpha_{ij}\}_{i=1,j=1}^{i=m,j=n}$
    Compute rewards $r(s_i, \alpha_{ij})$ for each $(s_i, \alpha_{ij}) \in \mathcal{S}_m \times \mathcal{A}_{mn}$
    Compute batch statistics $\mu(s_i, \alpha_{i,\cdot}), \sigma(s_i, \alpha_{i,\cdot})$ for rewards $r(s_i, \alpha_{ij})$
    Compute advantages $A(s_i, \alpha_{ij}) = \frac{r(s_i,\alpha_{ij}) - \mu(s_i,\alpha_{i,\cdot})}{\sigma(s_i,\alpha_{i,\cdot})}$
    Calculate logprobs of each token t_j of α_{ij} under $\pi_{\theta_{ref}}$
    Update $\theta_{old} \leftarrow \theta$
    Update $\theta \leftarrow \theta + \alpha \nabla \mathcal{J}(\theta)$ (Equation 2)
end while
Output: GRPO-trained policy π_θ

4.1 Tasks

Both tasks were split into train and test sets, with the test set consisting of 500 randomly selected samples from the original dataset. All other samples were used for training.

4.1.1 Countdown-3-to-4

Countdown-3-to-4 (Pan (2025)) is a simple task where the model is given three or four numbers, as well as a target number, and must yield a mathematical expression in +, -, *, / that evaluates to the target, using each number at most once.

Though in a vacuum, this task can be generated by a simple program, Pan (2025) presents the task as a dataset of 490,364 such problems, all of whose answers are programmatically verified using Python's `eval` function.

Although this task can be quickly solved using tree search methods, the need (in the worst case) to exhaustively evaluate all possible expressions means that the task should be pretty difficult for a language model to solve. This would be less of a problem if the language model has good numerical intuition and reasoning capabilities, but it makes the task harder in general.

4.1.2 GSM8K

GSM8K (Cobbe et al. (2021)) is a grade school level math problem dataset. It consists of 8,792 math problems, each with a single integer answer.

The problems are designed to be solvable by a grade school student, and cover a wide range of topics such as arithmetic, algebra, and geometry.

Due to the natural language format, this task is a lot harder to solve algorithmically, and it tests models' ability to parse diverse natural language math problems, reason about numbers, and perform arithmetic operations correctly (Cobbe et al. (2021)).

4.2 Experimental Configurations

Due to limited compute budget, each model and task combination was only tested with one run This also means that hyperparameters were tuned with expert advice and a desire to balance the training epoch count and training throughput, rather than extensive hyperparameter sweeping.

Training was conducted on a single NVIDIA 4090 GPU with 48GB of VRAM [5], and all runs took a combined $\sim$ 33 hours of wall-clock time to complete.

Hyperparameter	**Countdown-3-to-4**		**GSM8K**	
Model Family	Qwen2.5 (Qwen et al. (2025))			
Model Scale	0.5B	1.5B	0.5B	1.5B
Batch Size	16			
Learning Rate	1e-6			
Max. Response Length	512			
Completions per Task	8			
Episodes per GRPO Iteration	1024	1024	16	16
KL Divergence Coefficient λ	0			
Random Seed	42			

Table 1: Hyperparameters used in our experiments.

The hyperparameters in Table 1 were used for both experiments. They were mostly kept the same for parity, though they vary quite a bit from the defaults in nano-aha-moment (Kazemnejad et al. (2025)).

A notable exception is the `eps_per_iteration` parameter, which was set to 1024 for Countdown-3-to-4 (Pan (2025)), and 16 for GSM8K (Cobbe et al. (2021)). The GSM8K (Cobbe et al. (2021)) task has much fewer samples ($\sim$ 8k samples) than Countdown-3-to-4 ($\sim$490k, Pan (2025)), and thus fewer samples are required to achieve the same number of training epochs. In order to leverage the full computing capacity of the GPU at 1.5B scale (thus maximizing training throughput), we set the batch size to 16, necessitating a minimum episodes per GRPO iteration of 16.

4.3 Reward Modelling

As with most RL setups, the reward function design is crucial to the success of the RLVR algorithm.

In creating R1 and R1-Zero, DeepSeek-AI et al. (2025) appeared for a rather simple reward function. While the exact details of the reward function are not disclosed [6], it is said to have 2 principal components:

- Format Reward: The model must pen its thoughts in between `<think>` and `</think>` tags.
- Accuracy Reward: The model must output the correct answer to the problem in a specified format. This is enforced either with rule-based methods, or with external tools like a compiler.

We speculate that these rewards allow the model's responses to be evaluated most conveniently (hence the moniker "verifiable rewards"), and leave little room for undesirable optimizations such as reward hacking, for example, by outputting the same answer over and over again in hopes of getting some rewards.

This design pattern is also followed in nano-aha-moment (Kazemnejad et al. (2025)), and the reward function is solely comprised of the same two components, with the inclusion of an additional 0.5 reward if the model outputs an answer in the correct format, but its response is otherwise malformed (to encourage answering of questions).

For brevity, we will not include the full implementations here. They can instead be found in here (Countdown-3-to-4) and here (GSM8K).

[5] Thanks Chann.

[6] The most probable explanation is that the reward functions vary from task to task, and take too much space to document comprehensively.

4.4 Evaluation

The evaluation of the models is done using the Pass@1 metric across the entire test set. Specifically, we measure how many of the model's answers to the test set problems are verifiably correct, given only one attempt to answer each problem.

This is the least taxing evaluation metric, as it does not require the model to generate multiple answers to the same problem, and is thus the most suitable for our (compute-constrained) experiments.

4.5 Hypotheses

Before discussing the result, we want to make some hypotheses about the performance of RLVR on the two tasks, based on what we know about the tasks and their difficulty for language models.

Despite anecdotal observations that tasks like Countdown-3-to-4 (Pan (2025)) would be harder for humans to solve as compared to GSM8K (Cobbe et al. (2021)), we nevertheless hypothesize on grounds of algorithmic tractability that RLVR will perform better on Countdown-3-to-4 (Pan (2025)) than on GSM8K (Cobbe et al. (2021)).

Namely:

- RLVR will increase the Pass@1 score of Countdown-3-to-4 and GSM8K tasks across the board.
- Countdown-3-to-4 will require fewer training epochs to achieve good performance than GSM8K.
- After training, Countdown-3-to-4 will be solved with a higher Pass@1 score than GSM8K.

5 Results

5.1 Countdown-3-to-4

The results of the RLVR training on Countdown-3-to-4 (Pan (2025)) can be found in Figure 1.

At the end of 1000 training steps, the Pass@1 scores of the 0.5B and 1.5B models are quite similar, hovering somewhere around 0.35.

Though this appears better than random guessing, qualitative analyses of the model's reasoning traces reveal failures to engage in any meaningful thinking. Consider this excerpt from Qwen2.5-1.5B (Qwen et al. (2025)):

```
(Numbers: 17, 54, 13. Target: 50.)
Answer: <think> I need to use the numbers 17, 54 and 13 to make
the equation 50. </think> <answer>((54) - (17) + (13))</answer>
```

This answer is correct, but probably a fluke, as the model's reasoning trace simply repeats the question context, and does not engage in any meaningful reasoning.

Indeed, it uses the exact same equation for another question (which it gets wrong):

```
(Numbers: 54, 33, 31. Target: 27.)
Answer: <think> I need to use the numbers 54, 33 and 31 to make
the equation 27. </think> <answer>((54) - (33) + (31))</answer>
```

Even more egregious is Qwen2.5-0.5B (Qwen et al. (2025)), which is even more enthusiastic about using the same equation form for every question:

```
(Numbers: 49, 48, 40, 42. Target: 81.)
Answer:  <think> Given the numbers [49, 48, 40, 42] </think>
<answer>(49 - 40 + 48 - 42)</answer>
```

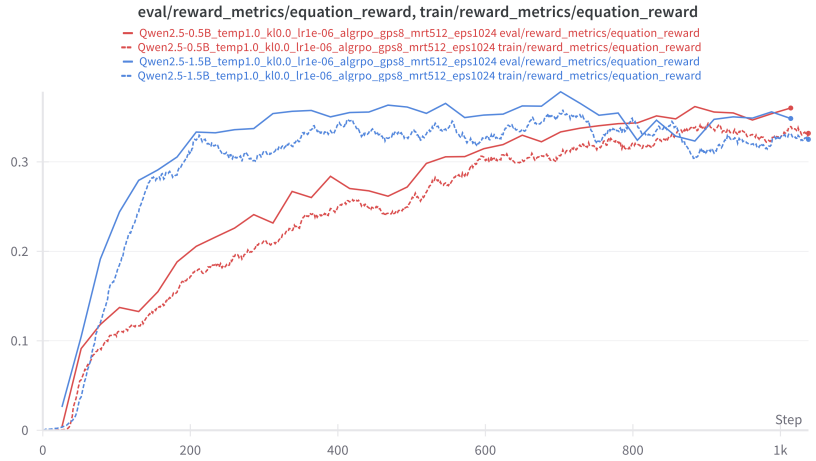

Figure 1: Performance of RLVR on Countdown-3-to-4 task. Dotted lines are training curves, solid lines are Pass@1 scores on the test set. (Red) Qwen2.5-0.5B, (Blue) Qwen2.5-1.5B. Curves are smoothed using Time-Weighted Exponential Moving Average (TWEMA) with $\alpha = 0.95$.

```
(Numbers: 40, 7, 93, 2. Target: 94.)
Answer: <think> Given the numbers [40, 7, 93, 2] </think>
<answer>(93 - 2 + 40 - 7)</answer>

(Numbers: 20, 27, 96, 46. Target: 97.)
Answer: Given the numbers [20, 27, 96, 46] </think> <answer>(96 -
20 + 46 - 27)</answer>
```

However, Qwen2.5-1.5B (Qwen et al. (2025)) is able to learn the task much faster, achieving that 0.35 Pass@1 score in just 200 training steps, while Qwen2.5-0.5B (Qwen et al. (2025)) takes about 800 training steps to reach the same score.

Somewhat surprisingly, even though Qwen2.5-0.5B does not experience sustained decreases in performance, Qwen2.5-1.5B's Pass@1 reaches a peak nearing 0.4 at around 700 training steps, before slowly declining to the final score of 0.35.

5.2 GSM8K

The results of the RLVR training on the GSM8K (Cobbe et al. (2021)) task can be found in Figure 2.

Unlike on the Countdown-3-to-4 task (Pan (2025)), Qwen2.5-1.5B achieves a much higher Pass@1 score than Qwen2.5-0.5B after 1000 training steps, with Qwen2.5-1.5B achieving a Pass@1 score above 0.8, and Qwen2.5-0.5B achieving a Pass@1 score of around 0.5.

Interestingly, also unlike on the Countdown-3-to-4 task (Pan (2025)), both models do NOT start out with a Pass@1 score of 0.0, with both hovering rather safely above that threshold at the start of training.

On GSM8K (Cobbe et al. (2021)), both models learnt at approximately the same pace, with both experiencing rapid increases in Pass@1 score in the first 200 training steps, before slowly but surely converging to their respective Pass@1 scores at the final step.

Interestingly, both models are quite cogent in their reasoning traces, with Qwen2.5-1.5B producing answers such as:

```
Question: Mason is trying to download a 880 MB game to his
phone. After downloading 310 MB, his Internet connection slows
```

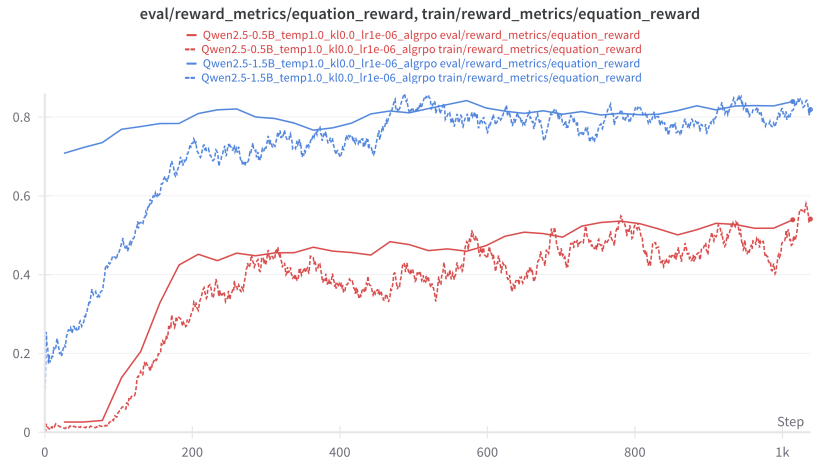

Figure 2: Performance of RLVR on GSM8K task. Dotted lines are training curves, solid lines are Pass@1 scores on the test set. (Red) Qwen2.5-0.5B, (Blue) Qwen2.5-1.5B. Curves are smoothed using Time-Weighted Exponential Moving Average (TWEMA) with $\alpha = 0.95$.

```
to 3 MB/minute. How many more minutes will it take him to
download the game? Show your work in <think> </think> tags, and
return the final answer as a single number in <answer> </answer>
tags, for example <answer>5</answer>, not <answer>the answer is
5</answer>.<|im_end|>
Answer: <think> First, we need to determine how much of the game
Mason has already downloaded, which is 310 MB. Then, we need to
find out how much more data he needs to download, which is the total
game size minus the downloaded size: 880 MB - 310 MB = 570 MB Now,
we know that his internet connection slows down to 3 MB/minute.
To find out how many more minutes it will take to download the
remaining 570 MB, we divide the remaining data by the download
speed: 570 MB / 3 MB/minute = 190 minutes So, it will take Mason an
additional 190 minutes to download the game. </think> <answer>190
</answer><|endoftext|>
```

and Qwen2.5-0.5B adopting a similar reasoning style:

```
(Same question as above.)
Answer: <think> Mason has a total download of 880 MB - 310
MB = 570 MB left to download. His Internet connection slows
to 3 MB/minute, so the time it will take to download the
remaining 570 MB is 570 MB / 3 MB/minute = 190 minutes. </think>
<answer>190</answer><|endoftext|>
```

5.3 Checking Our Hypotheses

We can now check our hypotheses from Section 4.5 against the results we have obtained.

- RLVR will increase the Pass@1 score of Countdown-3-to-4 (Pan (2025)) and GSM8K (Cobbe et al. (2021)) tasks across the board.
  - **True.** All models end training with a better Pass@1 score than they started with.
- Countdown-3-to-4 will require fewer training epochs to achieve good performance than GSM8K.

  - **False.** Qwen2.5-1.5B on GSM8K achieves a Pass@1 score of 0.7 after 200 training steps, while all models on Countdown-3-to-4 fail to ever break the 0.5 Pass@1 score.

- After training, Countdown-3-to-4 will be solved with a higher Pass@1 score than GSM8K.
  - **False.** The 1.5B model on GSM8K achieves a Pass@1 score above 0.8, while the 1.5B model on Countdown-3-to-4 only achieves a Pass@1 score of 0.35. Moreover, the 0.5B model on GSM8K achieves a Pass@1 score of 0.5, compared to a similar Pass@1 score of 0.35 for the 0.5B model on Countdown-3-to-4.

These results are overall rather surprising, and they are points of interest to consider when reflecting on how RLVR works in practice.

6 Discussion

6.1 Necessary Caveats

Before elaborating on the results, we have to point out some limitations of our experiments, which are primarily borne out of limited compute budget:

Firstly, our results may not be statistically significant, as each experimental configuration was only run once. It is very plausible that our results are as good / bad as they are purely due to random chance, and that the true performance of RLVR on these tasks is much better / worse than what we report here.

Secondly, our results are not "optimized" in the sense that no hyperparameter sweeping or prompt engineering was done. Any choice of hyperparameters/prompts was made with reference to the defaults in nano-aha-moment (Kazemnejad et al. (2025)), as well as expert consultations. This means that our results may not be the best possible performance of RLVR on these tasks, and that with more compute budget, we could have achieved better results.

6.2 GSM8K May Be Easier Than Countdown-3-to-4 For LLMs

One way to rationalise our results is that GSM8K (Cobbe et al. (2021)), by nature, could be a much easier task than Countdown-3-to-4 (Pan (2025)). Rather than having to autonomously enumerate and evaluate all possible expressions to find the correct answer, GSM8K problems often come with a lot of context and natural language cues that could hint to language models how they should use mathematical techniques to solve the problem.

6.3 Prompt Engineering May Be Required to Make RLVR Work

We did not do any prompt engineering for Countdown-3-to-4, as compared to GSM8K.

This is because we believed that the default prompt template was already optimized for the task by Kazemnejad et al. (2025), whereas there was no such default prompt for GSM8K, and we had to create one ourselves to be as specific as possible about what we want the model to do.

Somewhat naively, this leads us to believe that performing similar prompt engineering on the Countdown-3-to-4 task could lead to better results (Chen et al. (2025)), and that the default prompt template is not sufficient to make RLVR work on this task.

This appears to be a rather contentious point where RLVR is concerned, depending on which models you use. We know at least one instance in which not having a prompt template at all reportedly significantly boosted Qwen models' performance on RLVR tasks (Liu et al. (2025)).

6.4 RLVR May Be Eliciting Existing Capabilities Rather Than Enhancing Them

It is also possible that due to Pan et al. (2025)'s remarks that 0.5B models fail to develop reasoning capabilities, Qwen2.5-0.5B (Qwen et al. (2025))'s surprising ability to achieve a passing grade on GSM8K is a product of the model having seen enough math problems during pre-training to be able to answer them correctly.

This is a phenomenon that Shao et al. (2024) and Yue et al. (2025) have observed in their experiments, where RLVR was found to be increasing the output likelihood of responses determined correct by the verifier (via Maj@k improvement (Shao et al. (2024))), rather than increasing other metrics indicative of growth in reasoning capabilities, such as (ΔPass@k - ΔPass@1) (Yue et al. (2025)).

Such a notion is also supported by our experiments, where the same model (Qwen2.5-0.5B) fails quite spectacularly on Countdown-3-to-4, showing signs of mode collapse and outputting the same equation form over and over again without thought.

6.5 RLVR Could Require A Certain Model Scale to Work

All other speculations failing, it is very possible that the models we tested simply do not have enough parameters to benefit from RLVR. For one, we have been using models that are at most 1.5B parameters in size, as compared to the models used for RLVR experiments by Shao et al. (2024) (7B) and Liu et al. (2025) (1.5B, 7B), which are larger and could possess more learning capacity as a result.

However, given our current compute resources, we cannot test this hypothesis.

References

Chuen Yang Beh. Week 9: Why rl x llm works, 2025a. URL ../wk9/wk9.pdf.

Chuen Yang Beh. Week 8: Pre-training with nanogpt, 2025b. URL ../wk8/wk8.pdf.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. Unleashing the potential of prompt engineering for large language models. *Patterns*, 6(6):101260, June 2025. ISSN 2666-3899. doi: 10.1016/j.patter.2025.101260. URL http://dx.doi.org/10.1016/j.patter.2025.101260.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI, 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. F. 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, H. 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. L. 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. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu,

Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

HuggingFace. Hugging face, 2025. URL https://huggingface.co/.

Andrej Karpathy. nanogpt, 2022. URL https://github.com/karpathy/nanoGPT.

Amirhossein Kazemnejad, Milad Aghajohari, Alessandro Sordoni, Aaron Courville, and Siva Reddy. Nano aha! moment: Single file "rl for llm" library. https://github.com/McGill-NLP/nano-aha-moment, 2025. GitHub repository.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

John Langford and Tong Zhang. The epoch-greedy algorithm for contextual multi-armed bandits. 2008.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct, 2023. URL https://arxiv.org/abs/2308.09583.

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

Jiayi Pan. Countdown-tasks-3to4, 2025. URL https://huggingface.co/datasets/Jiayi-Pan/Countdown-Tasks-3to4.

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. Tinyzero. https://github.com/Jiayi-Pan/TinyZero, 2025. Accessed: 2025-01-24.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

John Schulman. Approximating kl divergence, 2020. URL http://joschu.net/blog/kl-approx.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. URL https://arxiv.org/abs/1506.02438.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

L. Weng. Policy gradient algorithms, 2018. URL https://lilianweng.github.io/posts/2018-04-08-policy-gradient/.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?, 2025. URL https://arxiv.org/abs/2504.13837.

Chi Zhang, Guangming Sheng, Siyao Liu, Jiahao Li, Ziyuan Feng, Zherui Liu, Xin Liu, Xiaoying Jia, Yanghua Peng, Haibin Lin, and Chuan Wu. A framework for training large language models for code generation via proximal policy optimization. *arXiv preprint arXiv: 2409.19256*, 2024.

A Some Issues with nano-aha-moment

We encountered a few issues which prevented us from running training code from the nano-aha-moment (Kazemnejad et al. (2025)) repository as-is, or otherwise slowed down our experiments.

- The logging code responsible for logging entropy and zero-advantages was misshapen as the authors neglected to remove the first advantage score from within the tensor.

- Within their notebook, it is highly likely the authors did not preprocess the HuggingFace dataset correctly, as the processed dataset did not include label masks, resulting in a KeyError during training.
- The authors' probably did not test their code on multiple GPUs, as the multi-GPU version of the code experienced deadlocks when trying to spawn multiple processes.
- The authors' tested their code on a single A100-80G with Qwen2.5-3B, which could be too big to run on consumer-grade hardware. As such, we had to scale down the model to Qwen2.5-0.5B, and later to Qwen2.5-1.5B, which can reasonably fit on a single 4090 GPU and not take too long to train.