CS 336: Assignment 5

Brandon Snider

June 06, 2025

Contents

3 Measuring Zero-Shot MATH Performance

Problem (`math_baseline`): 4 points

a) See `cs336_alignment/math_baseline.py`

b)

Format Reward	Answer Reward	n (of 5,000)
1	1	80
1	0	769
0	0	4,151

Table 1: Formatting and answer rewards, zero-shot baseline

The parser seems generally correct, but imperfect. In both sets of examples (format 1 answer 0, and format 0 answer 0), there are occasional instances of incorrect reward assignment, detailed below.

Observations from 10 cases in which format reward is 0:

In general, these examples show the model failing to follow the format. There are many failure modes: not outputting the closing `</think>` tag, not outputting one of the two answer tags, not outputting any tags at all, using non-XML syntax for the tags, putting the answer in an attribute on the opening `<answer>` tag, etc.

The parser seems to handle `\frac` strangely. See the first example below in which the model includes a closing `</think>` tag and includes the correct final answer between `<answer>` tags, but still receives a 0 format reward:

Completion that appears correct, but receives a 0 format reward:

```
Completion:

// We can write $\frac{1}{31} \times 93$.
</think>
<answer> <math>p=\frac{1}{31}\times 93=\frac{1\times 93}{31}=3</math>, as $\frac{31\times 3}{31}=3$</answer>

Ground truth:

3
```

Completion without a `</think>` tag:

```
We need to perform the multiplication of the three numbers 10.0000198, 5.9999985401, and 6.9999852, and then
round the result to the nearest whole number. Let's do the calculation first.
The product of these three numbers is approximately 459.999999999998.
<answer> 46 </answer>
```

Completion with answer as an attribute on `<answer`:

```
We can just count these, or we can think that the set of possible rolls is 6 x 6 = 36, so 36 / 9 = 4.
</think>
<a answer="1/6">We can also see that the probability is 1/6ths, because, in a 1/36 chance of each possible roll, the
probability of rolling a 9 is 4 different rolls out of a possible 36, and 4 / 36 = 1/9.</answer>
```

Observations from 10 cases in which format reward is 1, but answer reward is 0:

In most of these cases, the model just gets the answer wrong. However, there was one observed case in which a correct answer received 0 answer reward, shown below::

```
Completion:

To simplify the expression, we first combine like terms. There are four "a" terms, which when combined give 4a.
The constant terms are 1, minus 2, plus 3, minus 4. When we add these together, we get -2, as 1 - 2 + 3 - 4 = -2. The
reasoning process involves recognizing like terms and adding or subtracting their coefficients. </think> <answer>
$4a - 2$ </answer>

Ground truth:

4a-2
```

c) The zero-shot baseline performs poorly on MATH, getting 80 of 5,000 examples correct (1.6%).

Problem (`tokenize_prompt_and_output`): 2 points

See `cs336_alignment/utils.py`

Problem (`compute_entropy`): 1 point

See `cs336_alignment/utils.py`

Problem (`get_response_log_probs`): 2 points

See `cs336_alignment/utils.py`

Problem (`masked_normalize`): 1 point

See `cs336_alignment/utils.py`

Problem (`sft_microbatch_train_step`): 3 points

See `cs336_alignment/sft_exp/sft_microbatch_train_step.py`

Problem (`log_generations`): 1 point

See `log_generations` in `cs336_alignment/utils.py`

Problem (`sft_experiment`): 2 points

a)

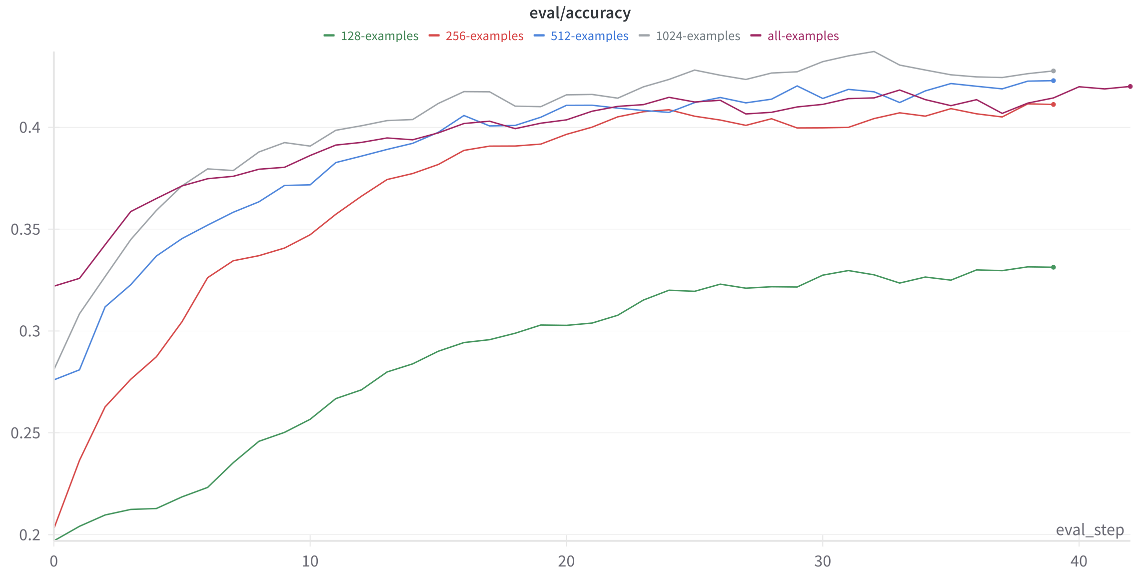

Figure 1: SFT validation accurary curves, varying unique examples (correct and incorrect)

b) Number of unique correct examples: **1,308**

Below I show (i) the validation accuracy curve when training on only correct examples, and (ii) all validation accuracy curves plotted together.

The final validation accuracy when training only on correct examples is **0.4416**. This provides a bump over training on all examples (for which the run with the best final accuracy achieves **0.4338**), though it's a much smaller improvement than I expected.

I also note that increasing the number of unique examples yielded negligible improvements beyond the smallest set of 128 examples, when the number of training epochs was fixed at 20. Of course, 20 epochs on a set of 256 examples is a much shorter training run than 20 epochs on 1024 examples, so it's interesting and surprising that the final validation accuracy is similar. This may suggest that the model is quite sample-efficient at learning whatever it can from the SFT examples.

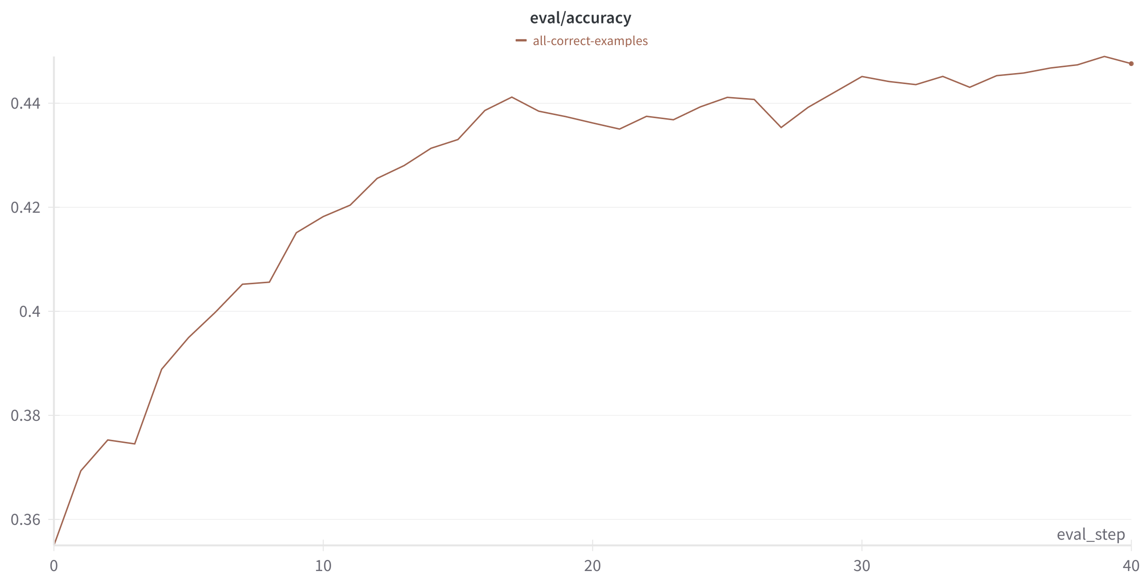

Figure 2: SFT validation accurary curve (correct examples only)

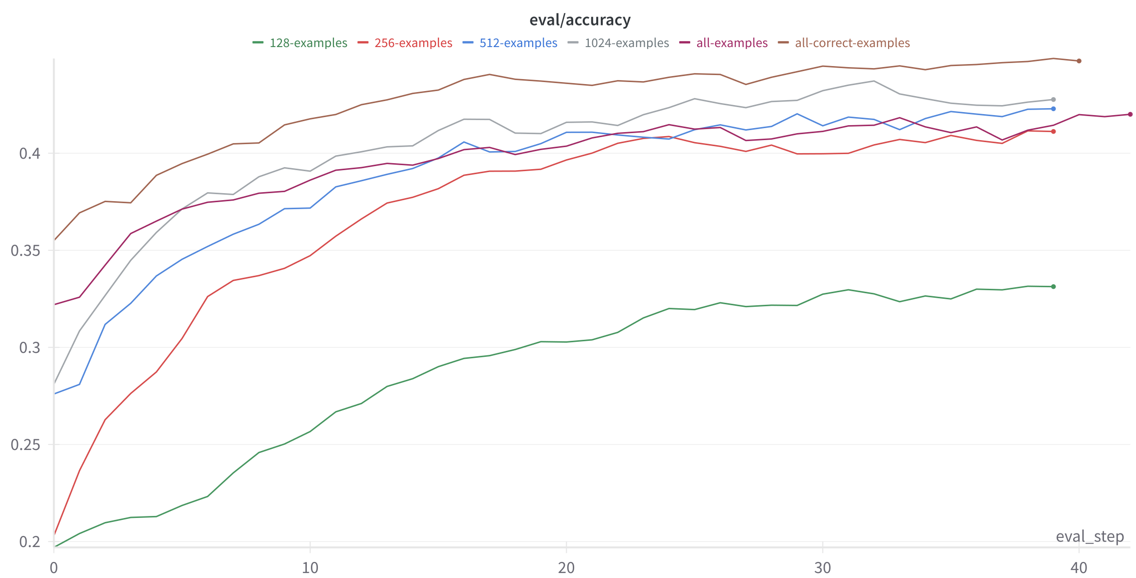

Figure 3: SFT validation accurary curves (all runs)

5 Expert Iteration for MATH

Problem (`expert_iteration_experiment`): 2 points

I find a significant improvement in the best validation accuracy using expert iteration over the best validation accuracy using SFT (0.494 vs. 0.4416). However, this was achieved over a run of almost two hours, whereas the SFT result was achieved in roughly 20 minutes.

The shape of the improvement curve is quite similar to the SFT improvement curve, though it appears that expert iteration has a higher ceiling. The SFT curves were almost exactly flat by the end of each run, whereas expert iteration continued to gradually improve.

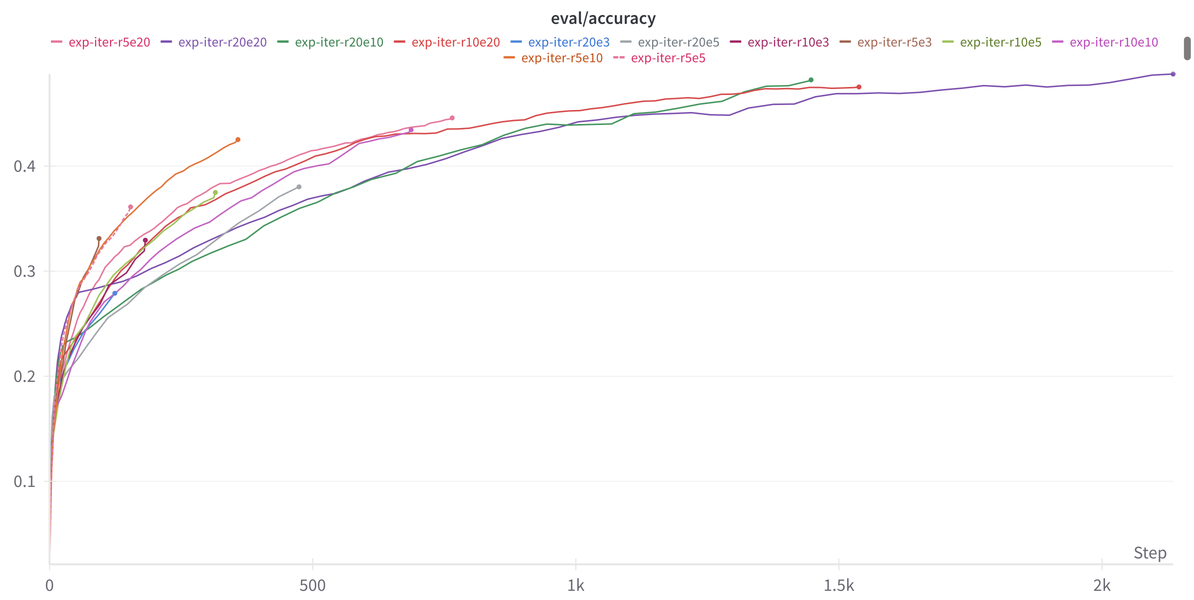

Figure 4: Expert iteration validation accuracy curves

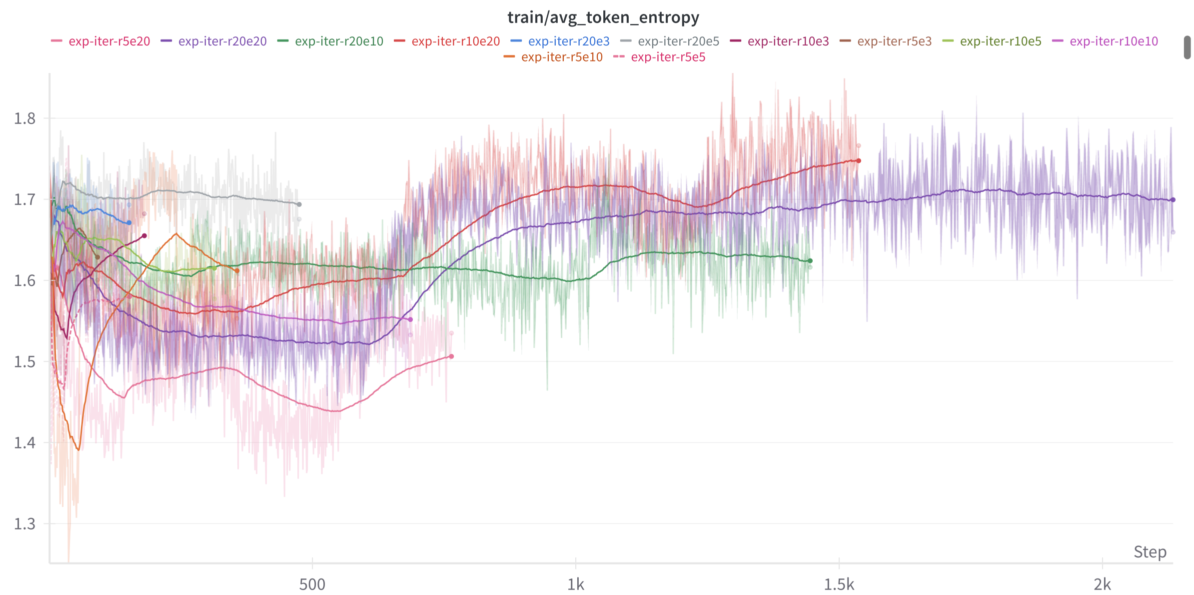

Figure 5: Expert iteration token entropy curves

7 Group Relative Policy Optimization

Problem (`compute_group_normalized_rewards`): 2 points

See `cs336_alignment/grpo_utils.py`

Problem (`compute_naive_policy_gradient_loss`): 1 point

See `cs336_alignment/grpo_utils.py`

Problem (`compute_grpo_clip_loss`): 2 points

See `cs336_alignment/grpo_utils.py`

Problem (`compute_policy_gradient_loss`): 1 point

See `cs336_alignment/grpo_utils.py`

Problem (`masked_mean`): 1 point

See `cs336_alignment/grpo_utils.py`

Problem (`grpo_microbatch_train_step`): 3 points

See `cs336_alignment/grpo/grpo_microbatch_train_step.py`

Problem (`grpo_train_loop`): 5 points

See `cs336_alignment/grpo/train.py`

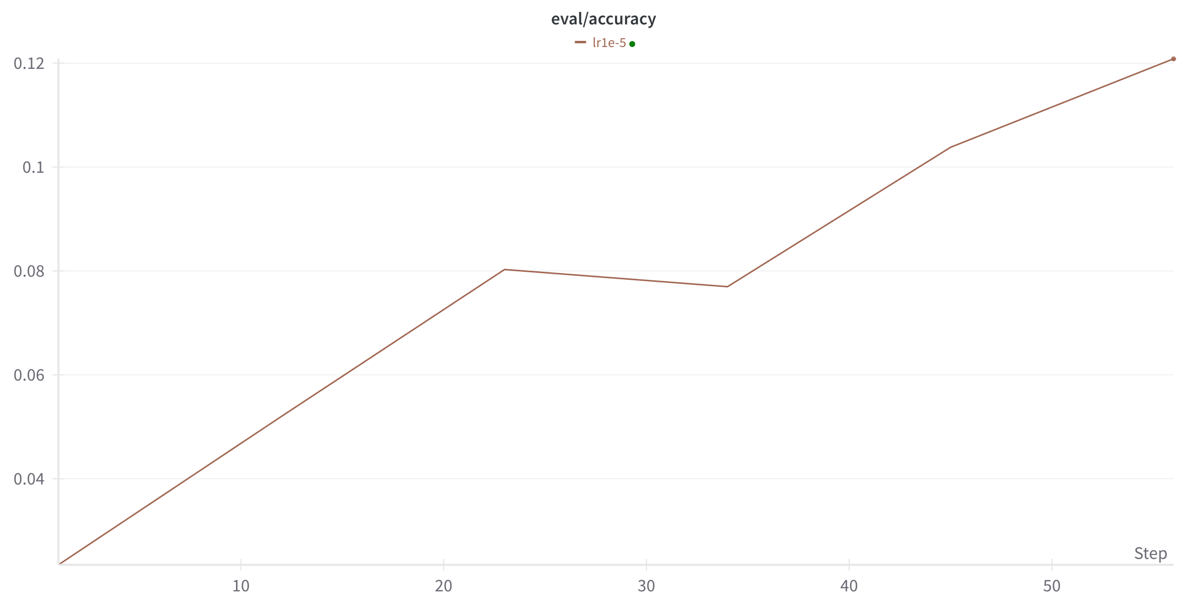

Figure 6: GRPO train loop: validation accuracy (first 50 steps)

Typical rollout from step 0 (multiple `<think>` tags and incorrect answer):

{"prompt":"A conversation between User and Assistant. The User asks a question, and the Assistant solves it. The Assistant first thinks about the reasoning process in the mind and then provides the User with the answer. The reasoning process is enclosed within <think> </think> and answer is enclosed within <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>.\nUser: What is the remainder when 5462 is divided by 9?\nAssistant: <think>","completion":" The units digit of 5462 is 2. And there is an odd number of digits in this number. Then the sum of the digits of the obtained number by dividing 5462 by 9 results in

<think> 2 </think>. The remainder in this case also results in <think> 2 </think>.</think> <answer> The remainder is 2 </answer>","ground_truth":"8","rewards":{}}

Example rollout from step 50 (correct format, incorrect answer):

{"prompt":"A conversation between User and Assistant. The User asks a question, and the Assistant solves it. The Assistant first thinks about the reasoning process in the mind and then provides the User with the answer. The reasoning process is enclosed within <think> </think> and answer is enclosed within <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>.\nUser: What is the smallest possible perimeter, in units, of a triangle whose side-length measures are consecutive integer values?\nAssistant: <think>","completion":"To find the smallest possible perimeter, we need to consider the properties of triangles and the fact that the side lengths are consecutive integers. </think> <answer>6 </answer>","ground_truth":"9","rewards":{}}

Example rollout from step 50 (correct format, correct answer):

{"prompt":"A conversation between User and Assistant. The User asks a question, and the Assistant solves it. The Assistant first thinks about the reasoning process in the mind and then provides the User with the answer. The reasoning process is enclosed within <think> </think> and answer is enclosed within <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>.\nUser: An assortment of 200 pencils is sold through a catalog for $\\$19.90$. Shipping is an additional $\\$6.95$. Including the charges for both the pencils and the shipping, what is the average cost, in cents, for each pencil? Express your answer rounded to the nearest whole number.\nAssistant: <think>","completion":"The cost for the pencils is \\$19.90 for 200 pencils, which is a cost of \\$0.0995 per pencil. Adding the shipping cost of \\$0.03475 per pencil, the total cost per pencil is \\$0.13425. Multiplying by 100 to convert to cents, the average cost per pencil is 13.425 cents. Rounding to the nearest whole number, the average cost per pencil is 13 cents.</think> <answer>13</answer>","ground_truth":"13","rewards":{}}

Problem (`grpo_learning_rate`): 2 points

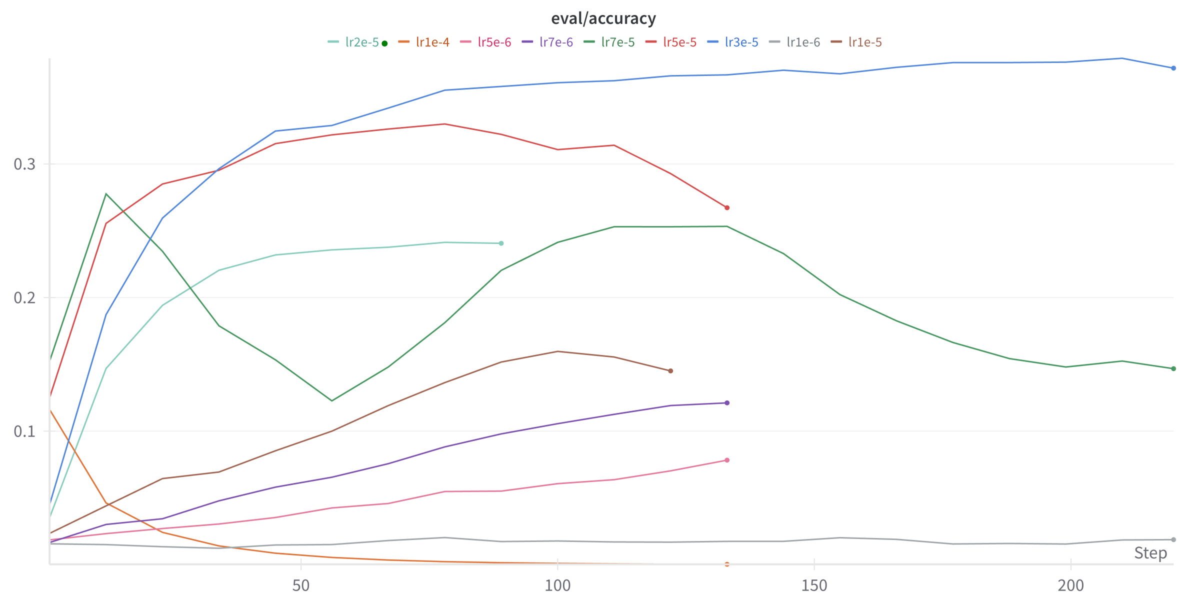

Figure 7: GRPO learning rate sweep

The optimal learning rate in my sweep was `3e-5`, achieving a final validation accuracy of 37%.

Above that, I saw significant instability (accuracy fluctuating throughout the run, or decreasing roughly monotonically).

At `1e-6` I saw no improvement. At learning rates between `5e-6` and `3e-5`, I generally saw improving accuracy and stable training, though not always (even changing the random seed seemed to have a significant effect). I generally truncated these runs around 130 steps because they looked unlikely to outperform `3e-5`. After 130 steps, validation accuracy was between 7% (at `5e-6`) and 20% (at `2e-5`).

I frequently observed entropy collapse. Entropy would start around 1.6, briefly shoot up after some number of steps, and then nearly vanish.

My intuition is that entropy grows while the model is doing some exploration, and then collapses as the model settles in some local optimum. Of course, after entropy collapsees, the model is no longer exploring, so accuracy stops improving. Intuitively this is suboptimal given that it happens even when the model clearly has headroom to perform better on the task. For example, at `2e-5` I saw entropy collapse and improvement cease at an accuracy of ~25%, but I know from the fun at `3e-5` that 37% is possible at minimum. It seems like shaping the reward to encourage continued exploration could be useful.

Problem (`grpo_baselines`): 2 points

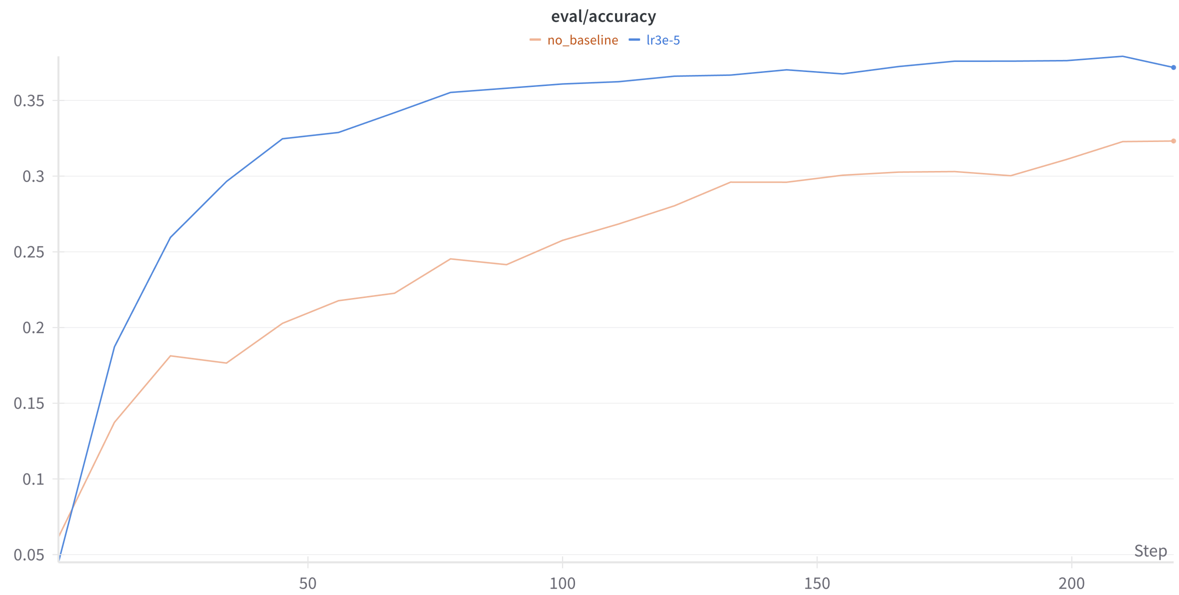

Figure 8: Validation accuracy curves with and without baselining

Using a baseline improved validation accuracy, and resulted in significantly different training dynamics. *With* a baseline, gradients and training loss were vastly larger and noisier. Entropy also followed a very different curve, spiking early and then collapsing, rather than staying roughly constant as it did without a baseline.

Figure 9: Metrics with and without baselining

Problem (`think_about_length_normalization`): 1 point

Normalizing by sequence length makes the per-sequence loss and per-sequence gradient contribution independent of the sequence length. Removing this normalization makes the loss and gradient contributions scale with the sequence length, such that the model is incentivized to produce longer correct answers (to maximize reward) and shorter incorrect answers (to minimize negative reward).

Normalizing by sequence length can help with stability because every trajectory is renormalized to unit weight. Gradient norms stay in a narrower range even when sequence lengths vary widely.

However, when we have outcome rewards and particularly want the policy to (i) value the extra reasoning tokens for correct answers (ii) de-vealue reasoning tokens that produce incorrect answers, constant normalizatioin (rather than sequence length normalization) makes sense.

Problem (`grpo_length_normalization`): 2 points

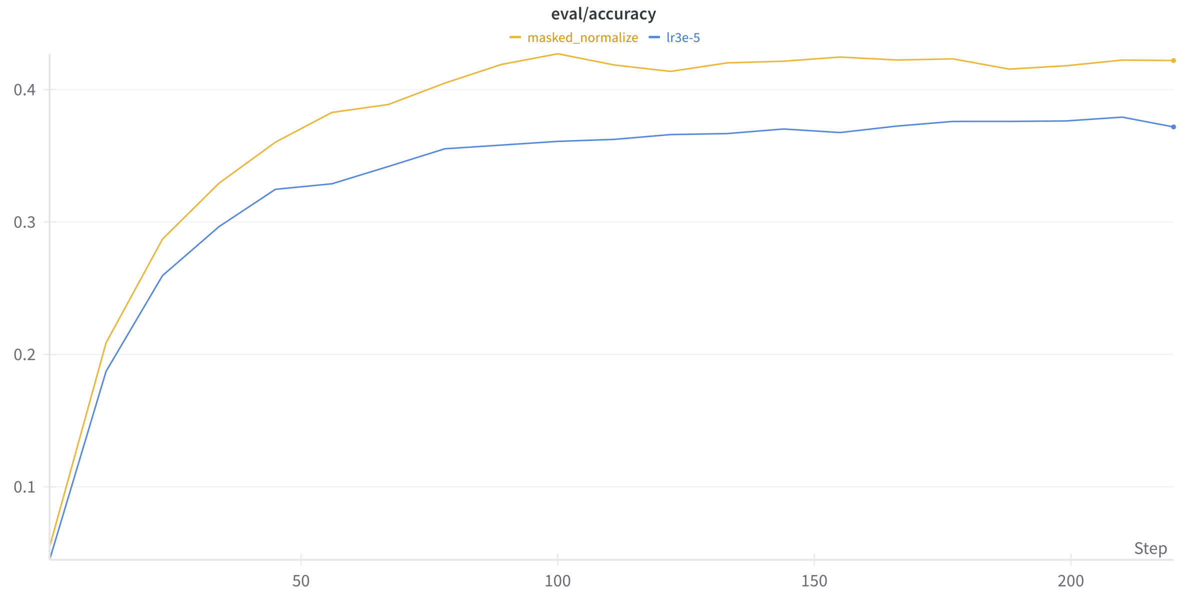

Figure 10: Validation accuracy: normalization by sequence length vs. normalization by constant

Dividing by a constant (the length of the longest generation) produces significantly better validation accuracy than normalizing by sequence length. The entropy curves also look different: normalizing by sequence length produces a larger early spike and deeper subsequenc collapse.

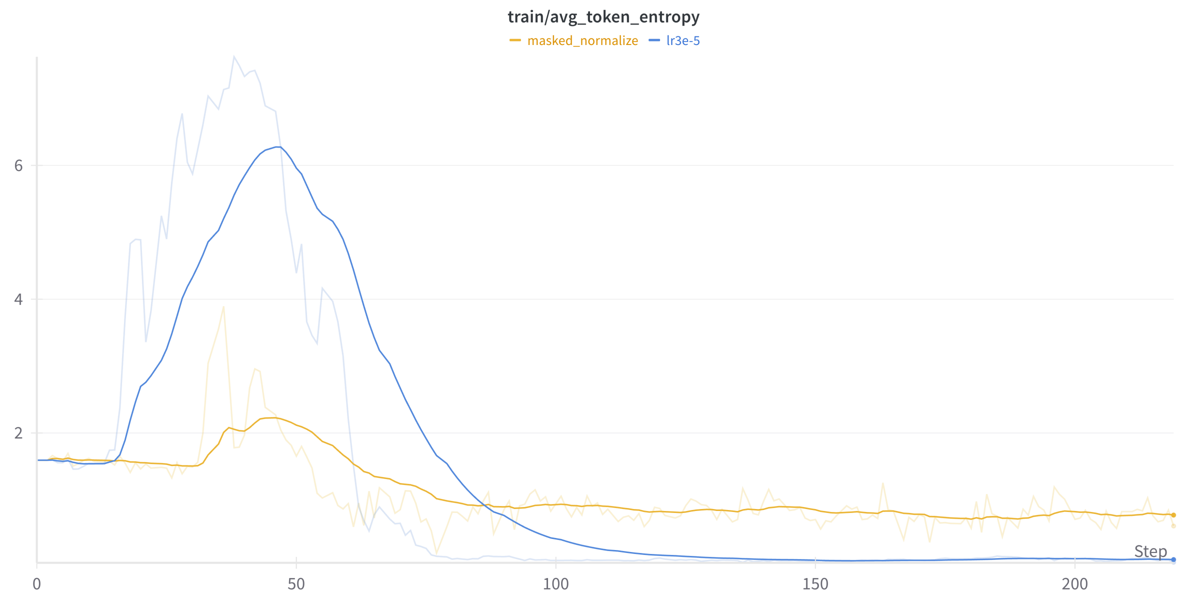

Figure 11: Entropy: normalization by sequence length vs. normalization by constant

Problem (`grpo_group_standard_deviation`): 2 points

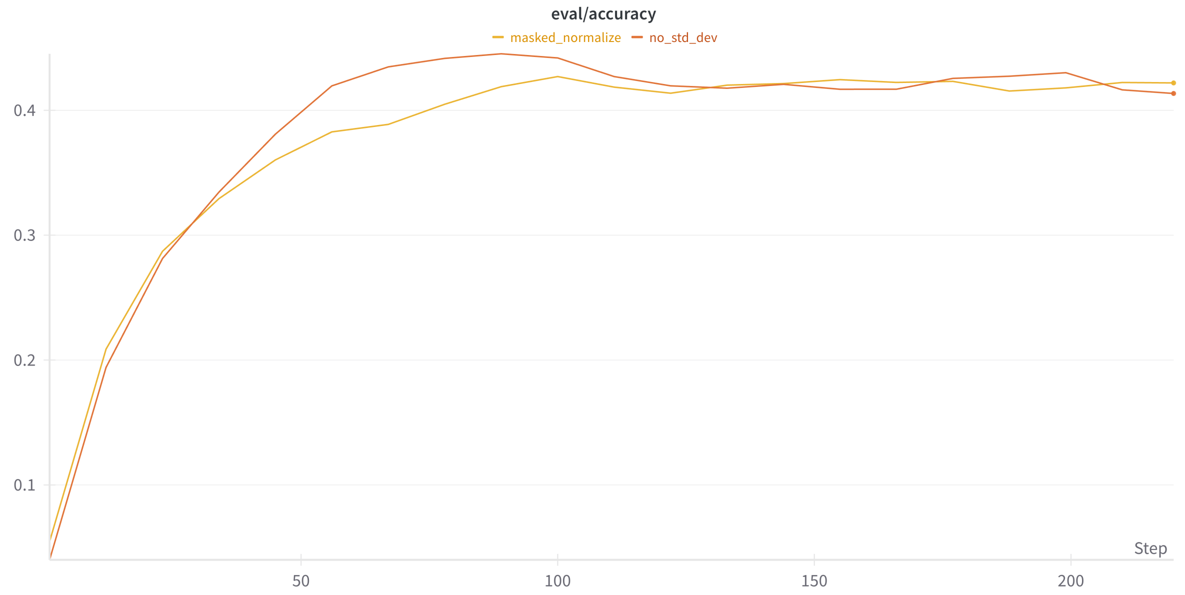

Figure 12: Validation accuracy with and without std. dev. normalization

Normalizing the advantage by the standard deviation of the group's rewards made almost no difference to validaton accuracy. It did make some difference to gradient norms and entropy. Without std. dev. normalization, gradient norms were substantially smaller; entropy spiked significantly further, and subsequently collapsed significantly more deeply.

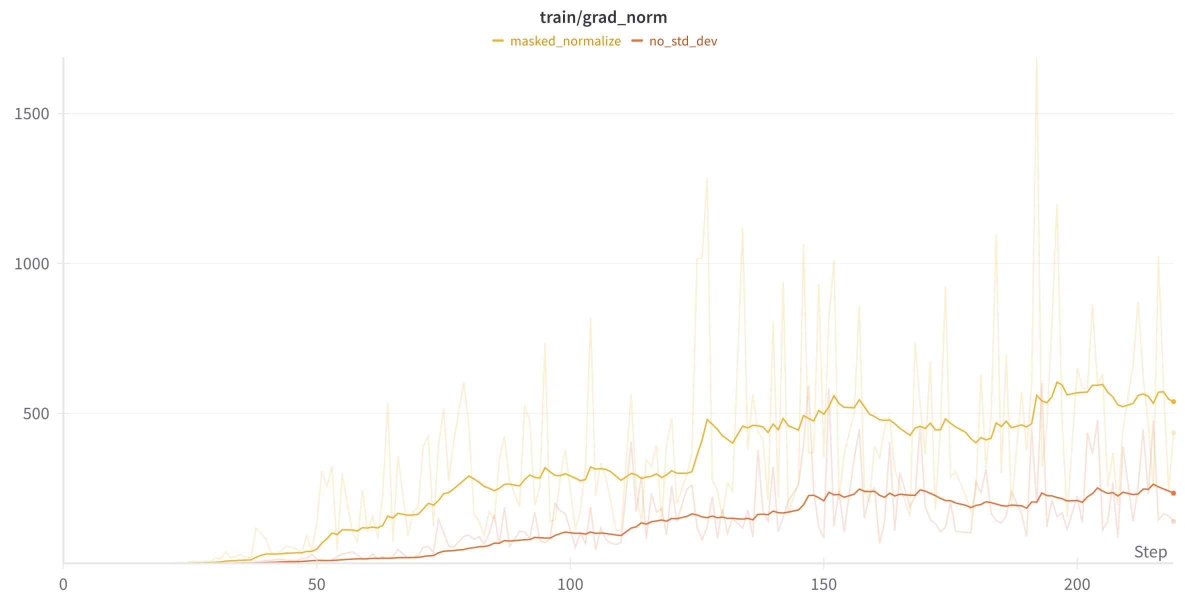

Figure 13: Entropy with and without std. dev. normalization

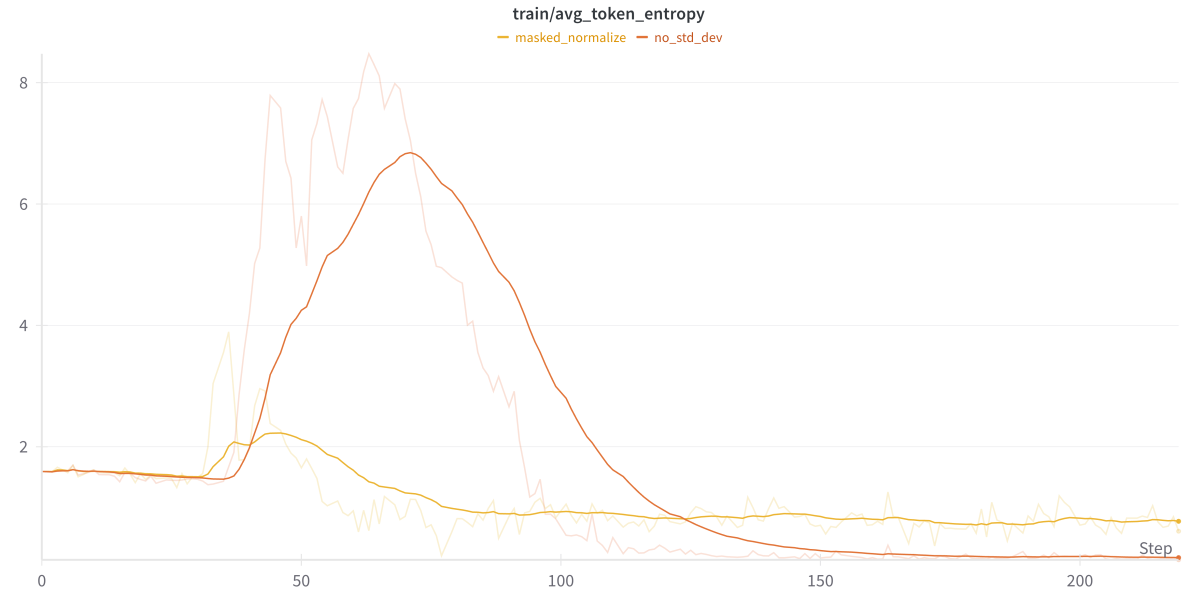

Figure 14: Entropy with and without std. dev. normalization

Problem (`grpo_off_policy_sweep`): 4 points

For the short runs (50 steps), I initially swept over 1, 2, and 4 epochs, with batch sizes of 16, 32, 64, 128, and 256, stopping early whenever it became clear that a configuration was suboptimal (e.g. divergence). I was not able to improve on the default on-policy configuration of 1 epoch with a batch size of 256.

I frequently saw divergence with batch sizes of 16 and 32. For all the runs that used more than 1 epoch, I saw slower improvements in accuracy. I was not able to discern any reliable pattern in entropy curves except that, in all cases, entropy shrank over the course of the run. In some cases, entropy grew early in the run and then shrank; in others, it shrank immediately, but this is not obviously correlated with either the number of epochs or the train batch size in my experiments.

Per eval step, the original configuration (1 epoch, batch size 256) significantly outperformed all others. Against wall clock time, using a smaller batch size (still 1 epoch) came close, but was still suboptimal.

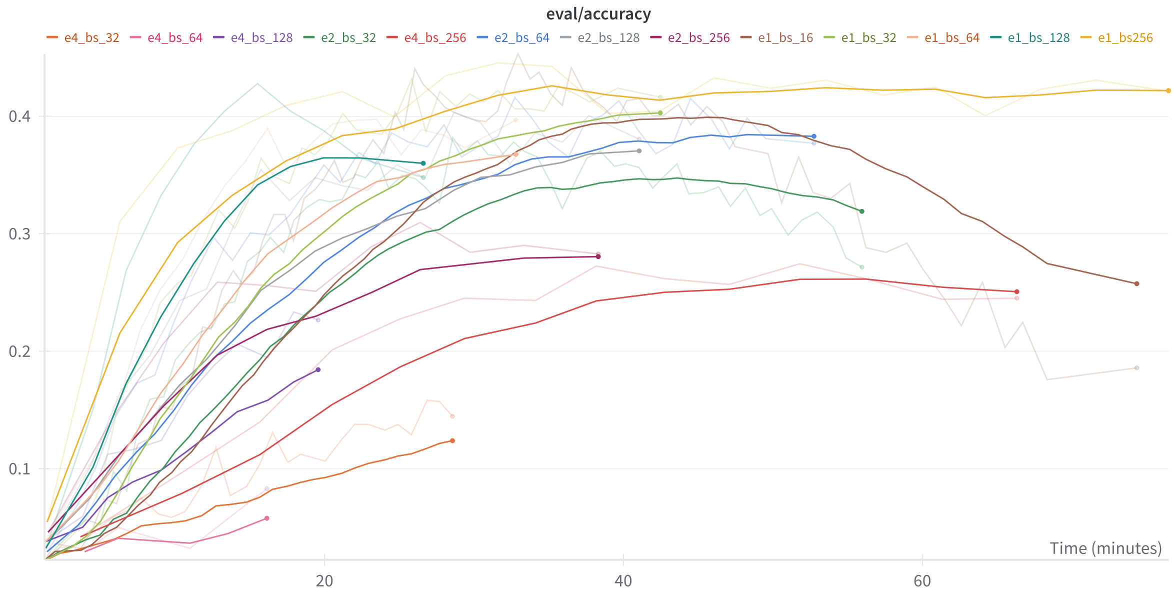

Figure 15: GRPO off-policy sweep: validation accuracy vs. wall clock time

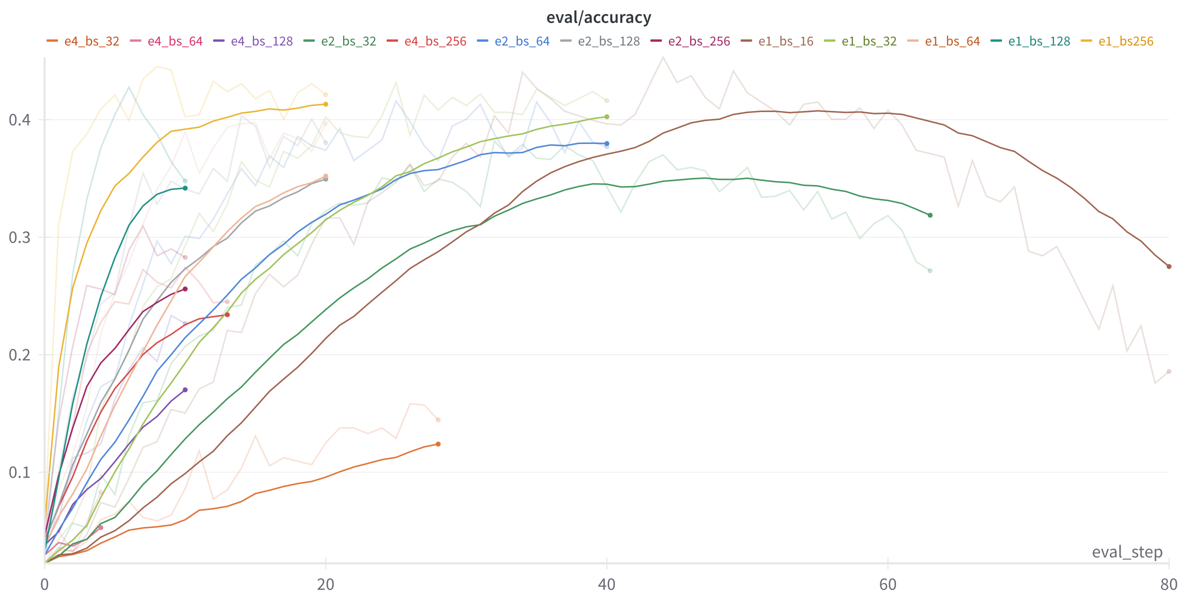

Figure 16: GRPO off-policy sweep: validation accuracy vs. eval step

Problem (`grpo_off_policy_clip_ablation`): 2 points

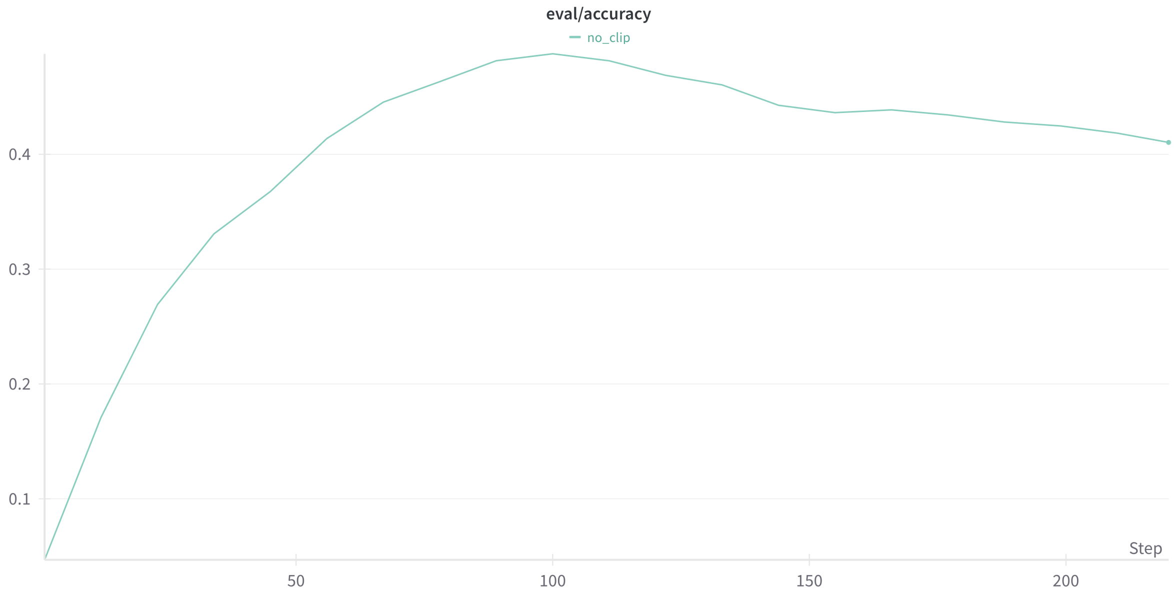

Figure 17: GRPO off-policy without clipping

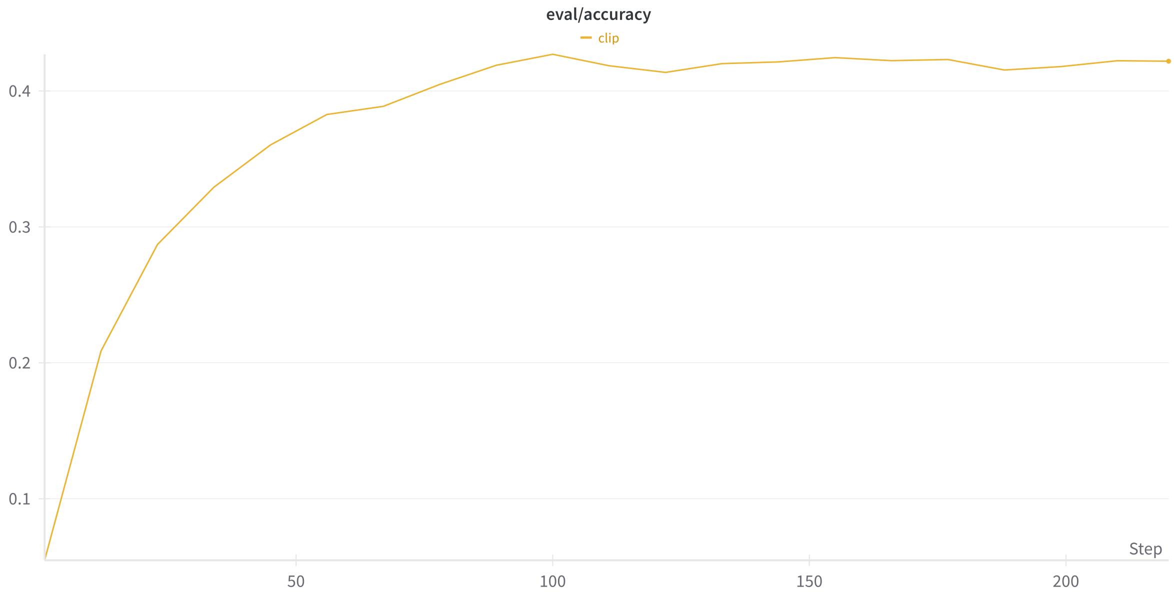

Figure 18: GRPO off-policy with clipping

Without clipping, I found training to be less stable. The gradient norms blew up after a few dozen steps, and there were occasional loss spikes. Entropy fluctuated, but seemed to grow over the course of the run when not clipping.

Problem (`grpo_prompt_ablation`): 2 points

The question-only prompt significantly outperforms the R1-Zero prompt. The answer reward starts higher, but also continues to improve throughout training rather than plateauing after fast initial improvement, which happens with the R1-Zero prompt.

Training with the question-only prompt also seems much more stable. Gradient norms are much smaller, entropy shrinks very slowly (rather than spiking and/or collapsing), and the loss remains relatively stable throughout training.

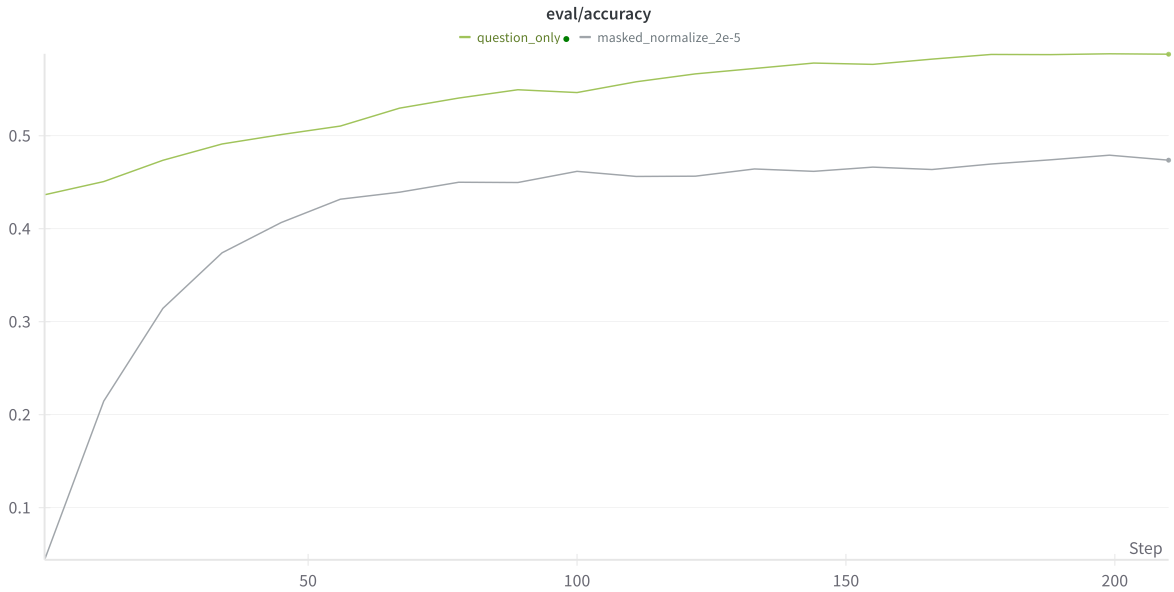

Figure 19: GRPO prompt ablation: validation accuracy

Problem (`leaderboard`): 16 points

WandB Report°

I swept over various combinations of loss types and baselines with and without std. dev. normalization, tuning learning rates in each case to try to get stability.

I found the best results with:

- Loss: `reinforce_with_baseline`
- Baseline: constant normalization by 1,024 (max. response length)
- Learning rate: `2e-5`
- Std. dev. normalization: `true`

Final validation accuracy: **48%**

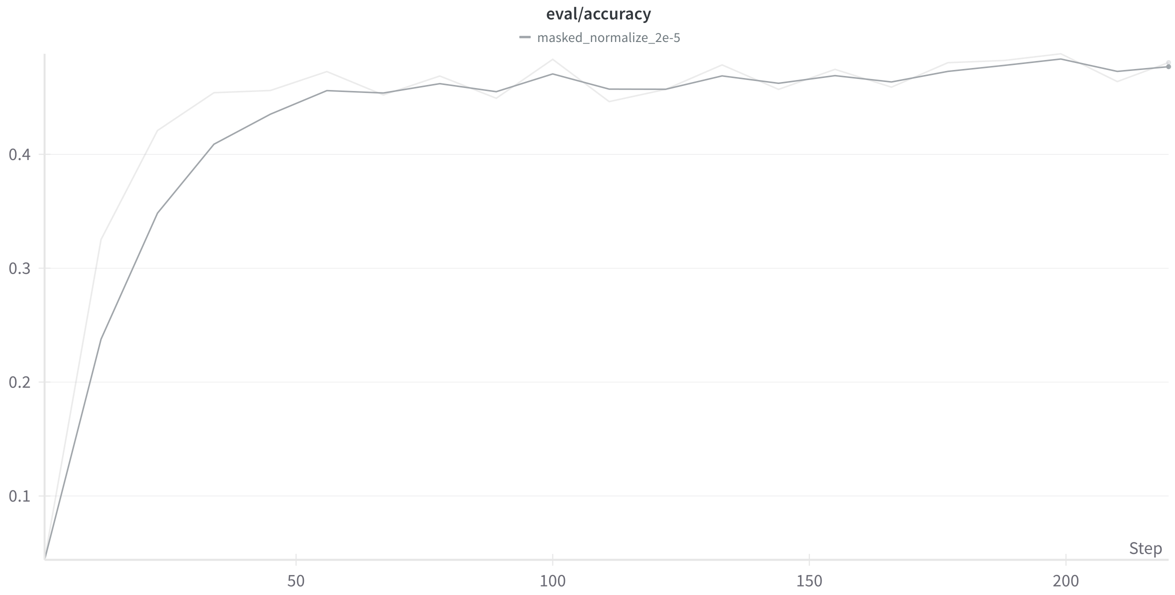

Figure 20: GRPO leaderboard run: validation accuracy

Bibliography

Index of Figures

Index of Tables