Eliciting Reasoning in LLMs using Logprob-based Rewards

Group Name: RL Barcelona
Authors: Orhun Ege Celik, Ibrahim Karaca, Semih Zaman, Arkadiusz Rozmarynowicz, David Grishchuk. Supervised by: Auguste Poiroux, EPFL LARA

Background

Log-probability-based rewards provide smooth and continuous feedback in RL for language models—especially useful when correctness is not binary. Language models assign probabilities to tokens. The log-probability reflects the model's confidence in predicting each token.

- **SEQUENCE SCORING:** Summing the log-probs of all tokens gives the total log-likelihood of a generated sequence.
- **USAGE AS REWARD:** In RL, the log-likelihood of the correct answer a given the prompt p and model-generated reasoning r can be used as reward:

We took the Qwen3B-Instruct non-reasoning model and trained it to be a reasoning model. Then, we also trained it to be a reasoning model but with a reward of checking exact matching. The model initially had only a 4% accuracy. We also took advantage of GSM8K math dataset.

Objectives

This project aims to explore reinforcement learning (RL) techniques to enhance reasoning capabilities in language models, focusing on reproducing and extending the DeepSeek-R1 approach.

- We implement and compared rule-based and log-probability-based reward mechanisms.
- Reproduce DeepSeek-R1 with rule-based rewards on a small scale (3B parameter model).
- Implement and evaluate logprob-based rewards.

Methods

LOG-PROBABILITY REWARD SIGNAL

- We discard the answer generated by the LLM and replace it with the official answer a. The reward is then the log-probability of a given prompt p and reasoning trace r:

$$R = \log P(a \mid p, r) = \sum_{t=1}^{T} \log \pi_\theta\big(a_t \mid a_{<t}, p, r\big)$$

BASELINE (GROUP AVERAGE) AND ADVANTAGE CALCULATION

- For a batch of N generated outputs $\{r_i, a_i\}_{i=1}^{N}$, compute the baseline as the average log-probability:

$$b = \frac{1}{N}\sum_{i=1}^{N} R_i = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i} \log \pi_\theta\big(a_{i,t} \mid a_{i,<t}, p, r_i\big)$$

- The advantage of a given output is the difference between its reward and the batch average:

$$A_i = R_i - b$$

POLICY GRADIENT UPDATE

- Use the standard policy gradient objective (REINFORCE) plus a KL-divergence penalty to prevent catastrophic drift.

NORMALIZATION & STABILITY TECHNIQUES

- *Length normalization*: counteract bias for shorter answers by dividing reward by answer length T_i:

$$\widetilde{R}_i = \frac{R_i}{T_i^{\alpha}}$$

- *Advantage normalization*: normalize A_i across the batch to zero mean and unit variance:

$$\hat{A}_i = \frac{A_i - \text{mean}(A)}{\sqrt{\text{Var}(A)} + \varepsilon}$$

- *KL-penalty tuning*: select λ and η (learning rate) to maintain stable reward trajectories and prevent collapse.

GRPO - GROUP RELATIVE POLICY OPTIMIZATION
BATCH SAMPLING (REPEAT-SAMPLER)

- Each prompt repeated multiple $\rightarrow$ multiple completions $\Rightarrow$ consistent group rewards.

COMPLETION GENERATION AND REWARD COMPUTATION

- Uses HuggingFace `.generate()` or vLLM.
- Truncates completions at the first `<eos>` token and computes attention masks.

ADVANTAGE CALCULATION

- For each group of G completions: $A_i = R_i - \frac{1}{G}\sum_{j=1}^{G} R_j$

LOSS COMPUTATION

- Using the clipped policy objective:

$$L = -\min\big(r_t\, A_t,\ \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\, A_t\big), \quad r_t = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)}$$

Discussions

It is important to note one detail about switching from rule-based to logprob-based rewards:

- Suppose the model receives a prompt i, generates a reasoning r, and outputs a prediction a_p. Let a_g denote the ground-truth answer. In rule-based evaluation, the reward is determined by whether $a_p = a_g$. In contrast, a logprob-based reward is defined as $\log P(a_g \mid i, r)$.
- In standard training, the model computes gradients based on the full sequence (i, r, a_p). This is valid for rule-based rewards because a_p and a_g align. However, in the logprob setting, $a_p \neq a_g$ is common, and computing gradients on a_p misaligns the learning signal with the intended reward.
- To address this, gradients should ideally be computed with respect to (i, r, a_g) so the model is trained to increase the likelihood of the correct answer. In some cases, using only (i, r) during backpropagation can reduce overfitting by decoupling reasoning optimization from direct answer memorization.

Implementation: The log-prob approach

The log-probability-based reward function calculates how likely the model thinks the correct (gold) answer is, given its generated reasoning. This reward is computed by estimating the conditional probability of the correct answer appearing after the model has already seen the prompt and generated the reasoning portion of its output. To do this, the function first constructs a full sequence consisting of:

1: The prompt (e.g., a math question),

2: The generated reasoning (text before <answer>), and

3: The gold answer (wrapped in <answer>...</answer> tags).

The combined sequence is tokenized and passed into the model to produce logits—raw scores for each possible token at every position. These logits are used to compute the log-probabilities of generating each correct token in the gold answer, conditioned on the previous tokens (prompt + reasoning).

Implementation: The rule-based approach

In the rule-based approach, we hand-craft five deterministic reward functions (rules) that inspect each generated completion and return a scalar score. All five functions are passed in as `reward_funcs` to the `GRPOTrainer`. During GRPO fine-tuning, these fixed rules steer the model toward:

- Producing well-formed XML wrappers (`<reasoning>…</reasoning><answer>…</answer>`)
- Giving numeric, concise answers
- Matching the gold answer exactly when possible

THE FIVE RULE BASED REWARD FUNCTIONS

correctness_reward_func(prompts, completions, answer, **kwargs) -> list[float]:

- **Goal:** Reward exact match of the generated answer to the gold answer.

1 Extract the completion content ⇒ Log question, gold answer, and model output to `responses.txt`.
2 Return 2.0 if the extracted answer equals the gold answer; otherwise 0.0.

int_reward_func(completions, **kwargs) -> list[float]:

- **Goal:** Encourage purely numeric answers.

1 Extract the `<answer>` string ⇒ return 0.5 if the answer is purely digits (`r.isdigit()`); otherwise 0.0.

strict_format_reward_func(completions, **kwargs) -> list[float]:

- **Goal:** Enforce an exact XML format with precise line breaks.

1 Define a regex: `<reasoning>\n.*?\n</reasoning>\n<answer>\n.*?\n</answer>\n`
2 Match each completion against this pattern ⇒ return 0.5 for a full match; otherwise 0.0.

soft_format_reward_func(completions, **kwargs) -> list[float]:

- **Goal:** Encourage the use of `<reasoning>` and `<answer>` tags without strict newline constraints.

1 Use a permissive regex: `<reasoning>.*?</reasoning>\s*<answer>.*?</answer>`
2 Return 0.5 if the tags appear in order; otherwise 0.0.

xmlcount_reward_func(completions, **kwargs) -> list[float]:

- **Goal:** Batch-apply `count_xml` to all completions.

1 Extract each completion's content ⇒ return a list of `count_xml(content)` values.

Results

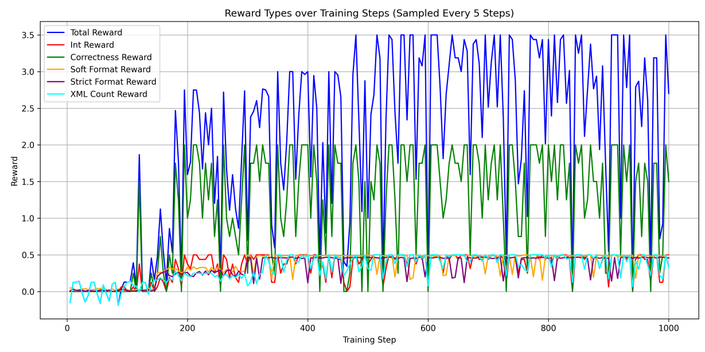

Figure 1: Rewards obtained by the rule-based approach over number of steps.

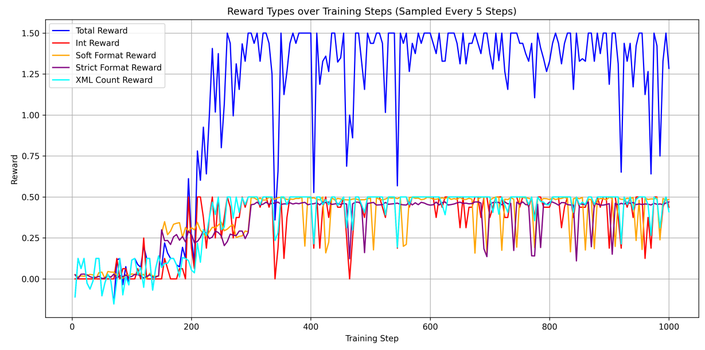

Figure 2: Rewards obtained by our baseline (a reasoning model) over number of steps

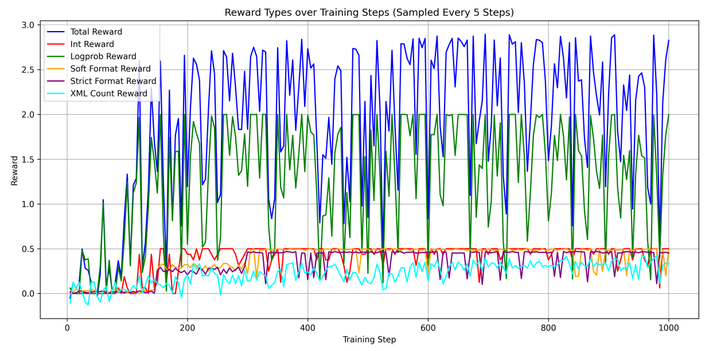

Figure 3: Rewards obtained by log-prob approach.

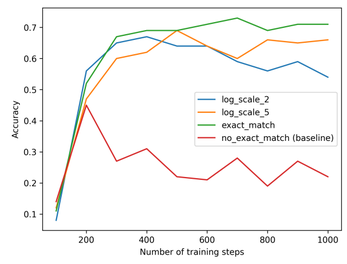

Figure 4: Comparison of the reasoning models trained with reward functions.

Rule-based reward performs best, peaking at 71% accuracy, while log-probability rewards (scaled between 0–2 and 0–5) come second—scaling to 0–5 yields better results with 67% accuracy. The baseline model lags behind with 25–30% accuracy, and shows the impact of meaningful reward signals.

Conclusion

The Qwen3B-Instruct model trained only with formatting rewards reached **25%** accuracy. Adding rule-based exact-match rewards boosted accuracy to **71%**, while logprob rewards with scale 5 and 2 reached up to **67%**, showing the effectiveness of both rewarding answers based on the ground truth and rewarding the probability of generating truth tokens conditioned with the reasoning trace.

With better tuning (e.g., dynamic scaling), the logprob-based method could potentially surpass exact matching, as it offers smoother gradients and more generalizable reward signals.

References: DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, by Zhihong Shao, Peiyi Wang, 2024.
Kimi k1.5: Scaling Reinforcement Learning with LLM, by Kimi Team, 2025
DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, autors too numerous to mention individually: research@deepseek.com, 2025.
GSM8K dataset, qwen3b-instruct model