

CS11-711 Advanced NLP

Reinforcement Learning for LLMs

Sean Welleck

**Carnegie
Mellon
University**



<https://cmu-l3.github.io/anlp-spring2026/>

<https://github.com/cmu-l3/anlp-spring2026-code>

Recap: last lecture

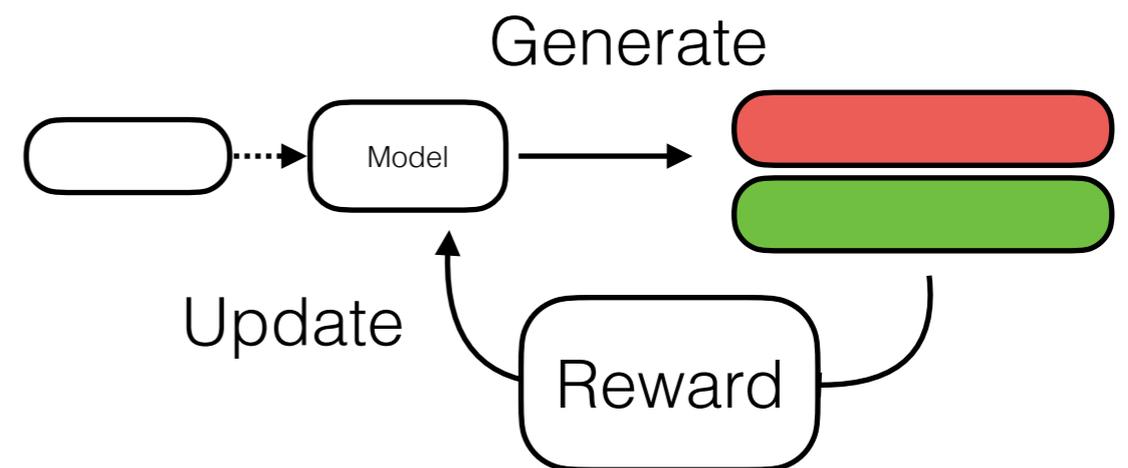
- Last lecture: RL fundamentals with CartPole
 - Markov Decision Processes (MDPs)
 - Policy gradient methods (REINFORCE)
 - Improved methods: advantages, PPO loss
- Today: RL for text generation with LLMs

This lecture

- RL Framework for LLMs
- Three examples
 - Reversing a string
 - Solving a math problem
 - Alignment with human preferences
- Terminology: RLVR, RLHF, GRPO,...

Three basic ingredients

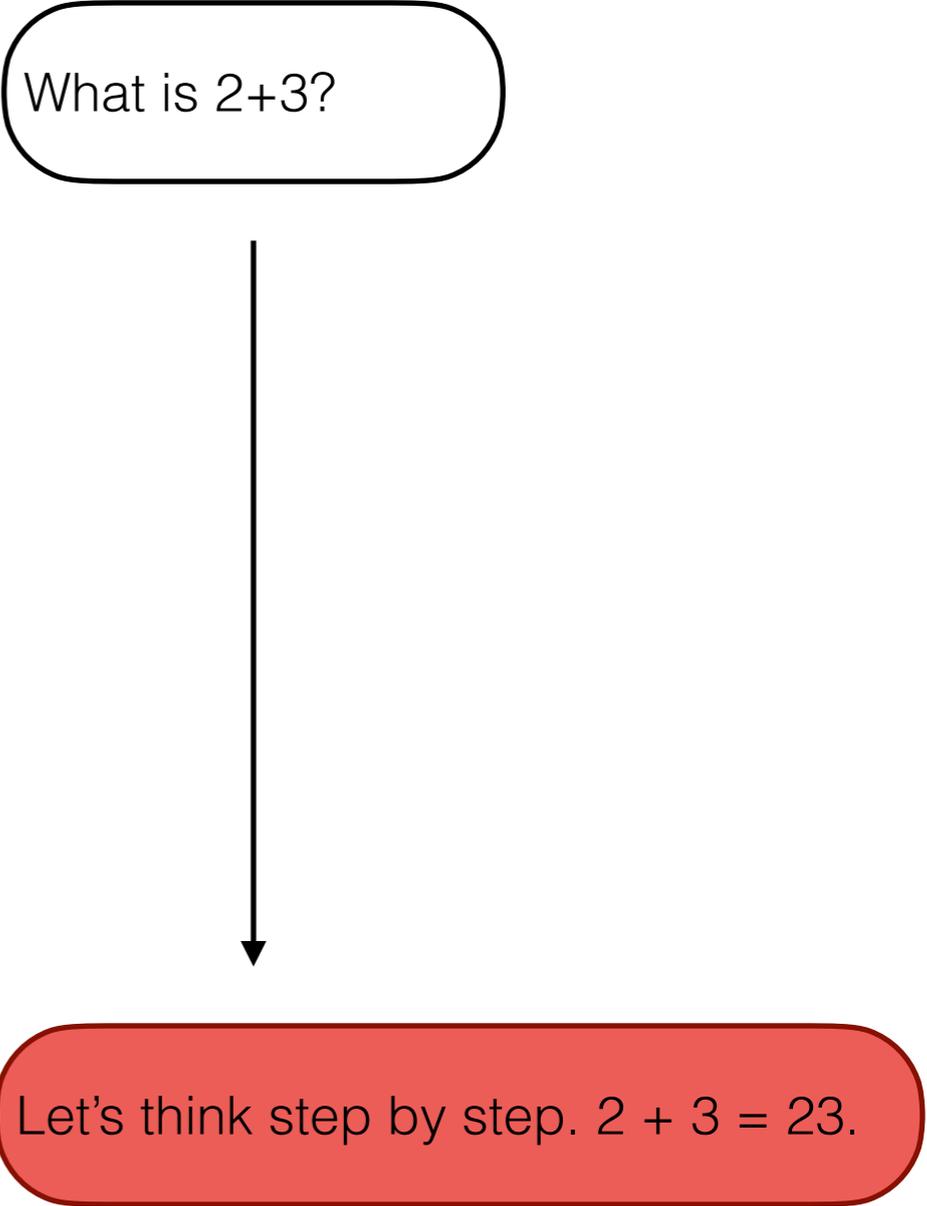
- 1.** Generate outputs
- 2.** Evaluate reward
- 3.** Update the model parameters



Step 0: frame the problem as a MDP

MDP 1: One-step MDP

- **State:** prompt or prompt + response
- **Action:** generate a full response
 - $a : y$
- **Policy:** language model
 - $p_{\theta}(y | x)$
- **Environment transition:**
 - Concatenate response to prompt
- **Reward:** evaluate reward on the full sequence
 - $r(x, y)$

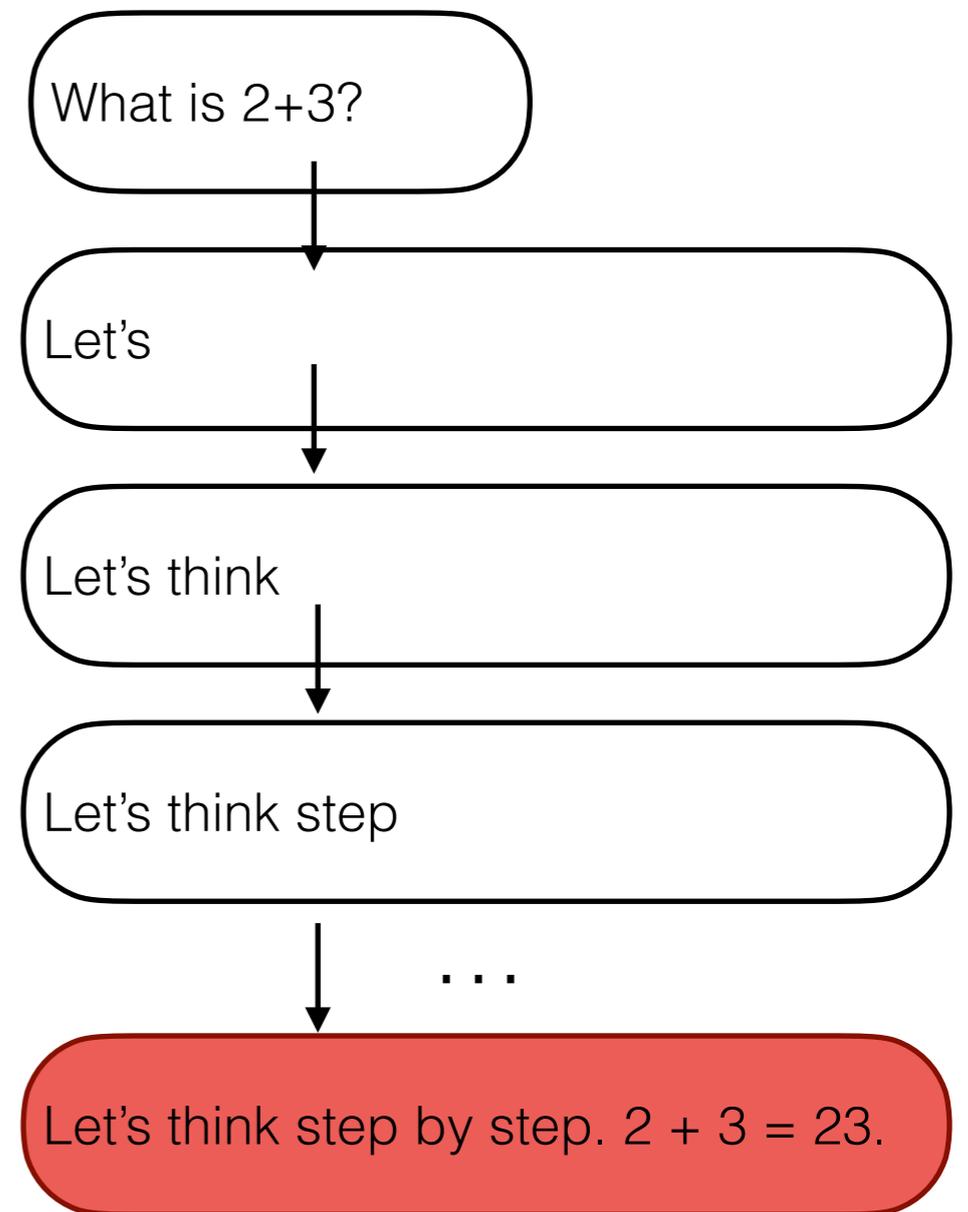


What is 2+3?

Let's think step by step. 2 + 3 = 23.

MDP 2: Token-level MDP

- **State:** a prompt and tokens-generated-so far
 - $s_t : (x, y_{<t})$
- **Action:** generate a token
 - $a_t : y_t$
- **Policy:** language model
 - $p_{\theta}(y_t | y_{<t}, x)$
- **Environment transition:** append token
 - $s_{t+1} : (x, y_{<t} \circ y_t)$
- **Reward:**
 - $r_t = 0$ for $t < T$
 - $r_T = r(x, y)$



Key difference: we can assign different rewards to different tokens

Putting it all together

Now we can use the algorithms from last lecture!

Example (policy gradient):

- Given:
 - Pre-trained or fine-tuned model, $p_{\theta}(y | x)$
 - Inputs x
 - Reward function r
- Loop:
 - Generate outputs \hat{y} with p_{θ}
 - Compute rewards
 - Compute loss, $L_{PG} = \sum_t r(s_t, a_t) \nabla_{\theta} \log p_{\theta}(a_t | s_t)$, update p_{θ}

Key decisions

- Given:

- Pre-trained or fine-tuned model, $p_{\theta}(y | x)$

1. Fine-tune first or not

- Inputs x

2. Dataset of inputs

- Reward function r

3. Reward function

- Loop:

- Generate outputs \hat{y} with p_{θ}
- Compute rewards, $A_t = r(s_t, a_t)$

4. Policy optimization settings

- Compute loss,
$$L_{PG} = \sum_t A_t \nabla_{\theta} \log p_{\theta}(a_t | s_t),$$

update p_{θ}

- Advantages A_t

- Loss L

- Other hyper-parameters

Key decisions

- We'll look at three examples that will illustrate concepts related to the key decisions:
 - Initialization/fine-tuning
 - Dataset of inputs
 - Reward function
 - Policy optimization settings

This lecture

- RL Framework for LLMs
- Three examples
 - **Reversing a string**
 - Solving a math problem
 - Alignment with human preferences
- Terminology: RLVR, RLHF, PPO, GRPO, DrGRPO,...

Example: reversing a string

- Task: reverse input strings
 - hello->olleh
- **Initialization: Fine-tune on a dataset of (input, reversed input) examples**

Fine-tuning before RL

- Given pre-trained model p_{θ_0} , it is often beneficial to do supervised fine-tuning prior to RL, $p_{\theta_0} \rightarrow p_{\theta_{SFT}} \rightarrow p_{\theta_{RL}}$
- Pros
 - Teaches the model the task format
 - Leverages supervision from the dataset
 - => model may encounter more high-reward examples
- Cons
 - Requires a dataset of (x, y) examples
 - May narrow/bias the model's output distribution too much

Code example

Manual Examples:

```
=====
sean      → naes      (expected: naes      ) ✓
joel     → leoj      (expected: leoj      ) ✓
chen     → nehc      (expected: nehc      ) ✓
dareen   → nederad   (expected: neerad   ) ✗
neel     → leen      (expected: leen      ) ✓
akshita  → atihska   (expected: atihska   ) ✓
ashish   → hsihsa   (expected: hsihsa   ) ✓
manan    → nana     (expected: nanam    ) ✗
sanidhya → ayhdinas  (expected: ayhdinas  ) ✓
```

Manual Examples Accuracy: 7/9 = 77.8%

```
=====
Evaluating on test set (sample size: 500)...
=====
Evaluating: 100% 500/500 [01:13<00:00, 6.80it/s]
```

Test Set Exact Match Accuracy: 64.80%
Number of errors: 176/500

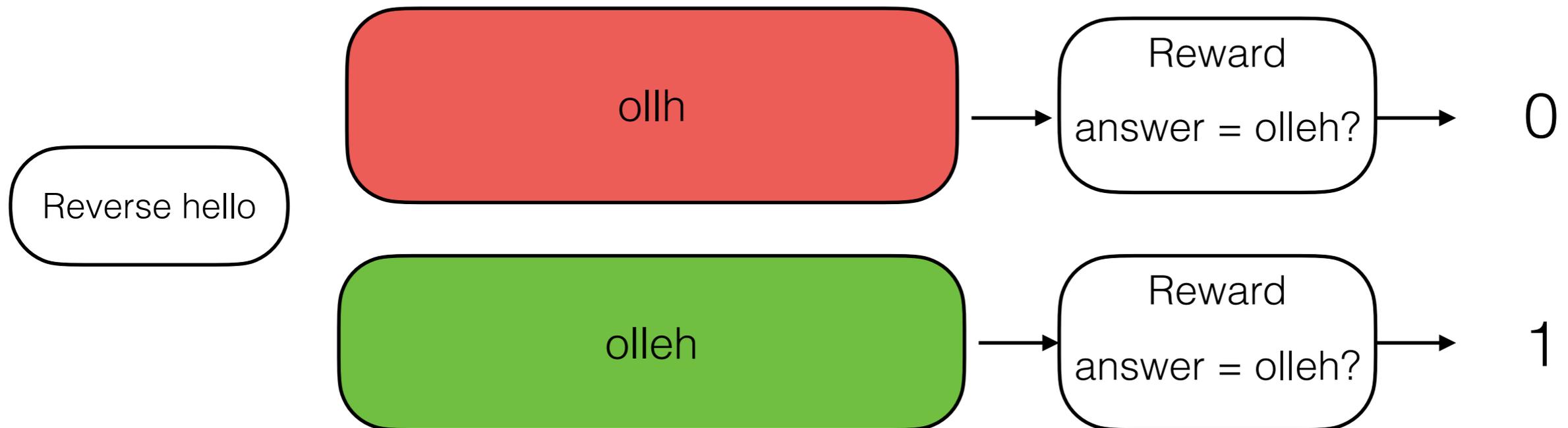
sft.py

Example: reversing a string

- Task: reverse input strings
 - hello->olleh
- Fine-tune on a dataset of (input, reversed input) examples
- **Reward: 1 if the output is the reverse of the input, 0 otherwise**

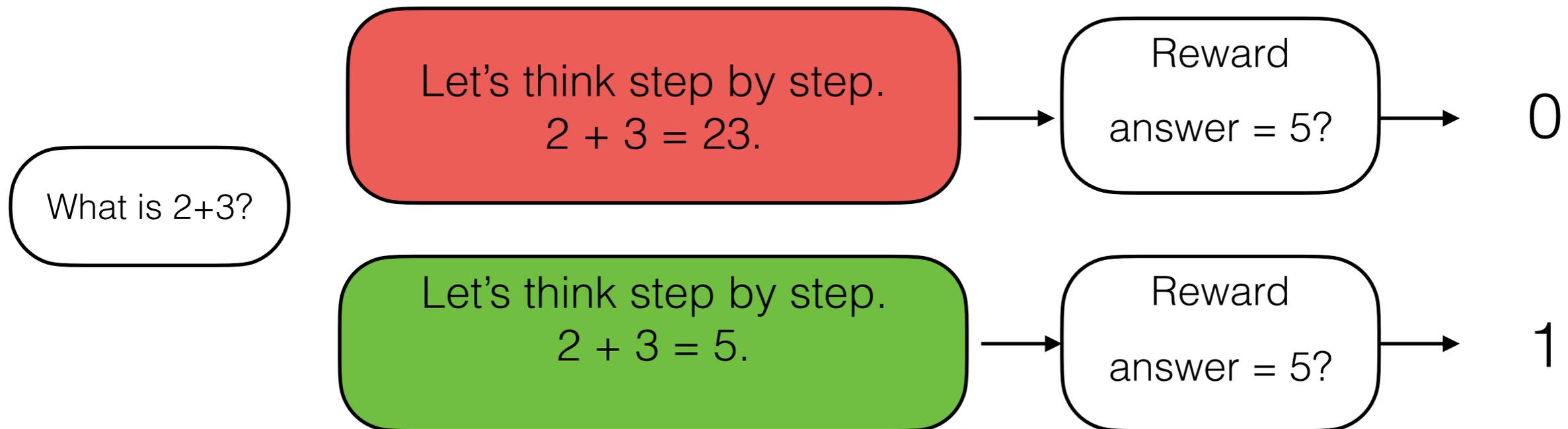
Rule-based/“verifiable” rewards

- A verifiable/checkable property of the output
- Example: string reversal
 - $r(x, y) = 1$ if y 's answer is correct, 0 otherwise



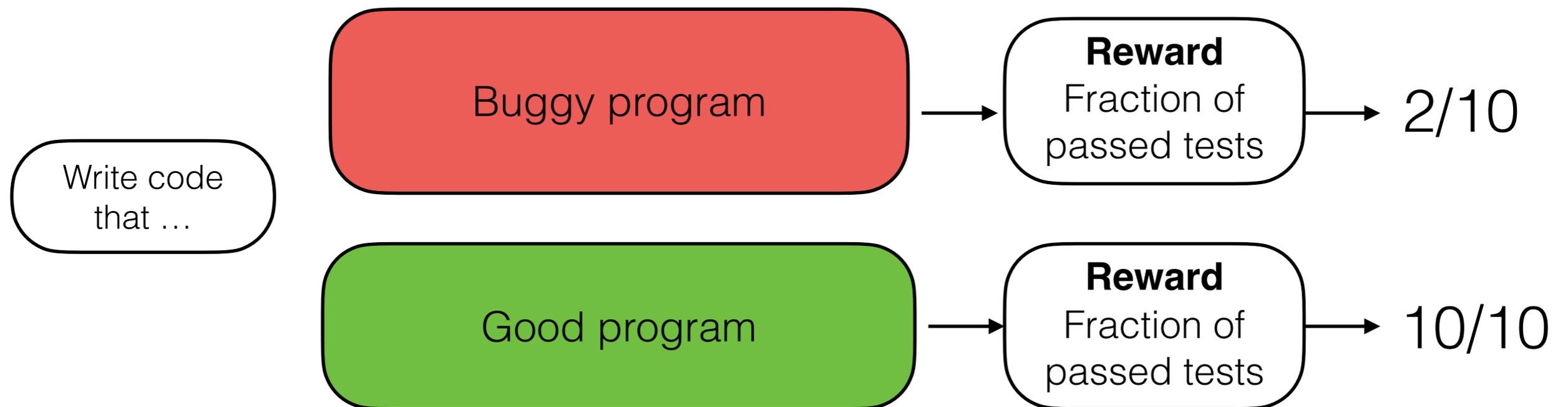
Rule-based/“verifiable” rewards

- A verifiable/checkable property of the output
- Example: solve a math problem
 - $r(x, y) = 1$ if y 's answer is correct, 0 otherwise



Rule-based/“verifiable” rewards

- A verifiable/checkable property of the output
- Example: write a program that passes test cases
 - $r(x, y)$ = fraction of passed tests



Code example

```
def reward_function(output, prompt):  
    # Extract the target name from the untokenized prompt  
    prompt_str = tokenizer.decode(prompt, skip_special_tokens=True)  
    target_name = prompt_str.split("Reverse the name:")[−1].split(".")[0].strip()  
    # Decode the model output, take the string after Answer: and prior to the first period.  
    output_str = tokenizer.decode(output, skip_special_tokens=True)  
    parsed_output = output_str.split("Answer:")[−1].split(".")[0].strip()  
    if parsed_output == target_name[::−1]:  
        return 1.0  
    else:  
        return 0.0
```

reverse_ppo.py

Example: reversing a string

- Task: reverse input strings
 - hello->olleh
- Fine-tune on a dataset of (input, reversed input) examples
- Reward: 1 if the output is the reverse of the input, 0 otherwise
- **One-step MDP:**
 - State: “reverse hello”
 - Action: full output string

Example: reversing a string

- Task: reverse input strings
 - hello->olleh
- Fine-tune on a dataset of (input, reversed input) examples
- Reward: 1 if the output is the reverse of the input, 0 otherwise
- One-step MDP
- **Algorithm: group-based advantages + PPO loss**

Group-based advantages

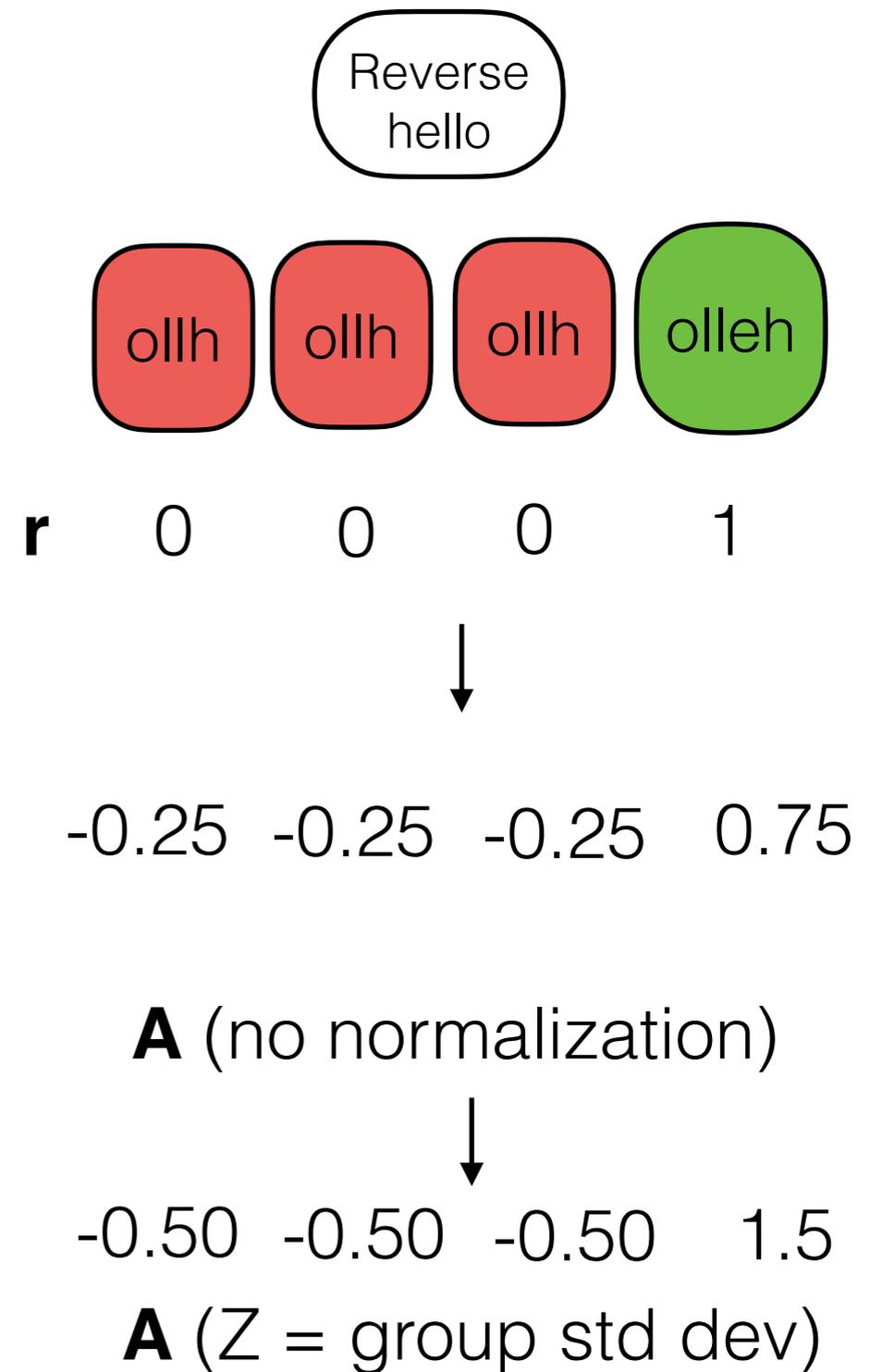
- For each input $x^{(i)}$, generate K outputs

- $\{y^{(i,1)}, y^{(i,2)}, \dots, y^{(i,K)}\} \sim p_{\theta}(\cdot | x^{(i)})$

- Subtract the mean reward-per-group, optionally normalize

- $$A^{(i,k)} = \frac{(r^{(i,k)} - \text{mean}(r^{(i,1)}, r^{(i,2)}, \dots, r^{(i,K)}))}{Z}$$

- Z : standard deviation of group rewards



Group-based advantages

- Pros:
 - Does not require a learned value function
- Cons:
 - Less clear how to assign advantages to non-terminal states
 - Can be computationally expensive to generate multiple outputs

Code example

```
# -- Compute rewards and advantages
rewards = []
for i in range(ids.size(0)):
    reward = reward_function(ids[i], input_ids[i // num_samples_per_prompt])
    rewards.append(reward)
rewards = torch.tensor(rewards, dtype=torch.float32, device=device)
rewards_original = rewards.clone()

# Get the mean per prompt
means = rewards.view(-1, num_samples_per_prompt).mean(dim=1).repeat_interleave(num_samples_per_prompt)
advantages = rewards - means
```

reverse_ppo.py

PPO Loss

Policy gradient (REINFORCE)

$$L_{PG} = A_t \log p_{\theta}(y_t | y_{<t}, x)$$

$$\Rightarrow \text{gradient} = A_t \nabla_{\theta} \log p_{\theta}(y_t | y_{<t}, x)$$

Proximal policy optimization (PPO)

$$L_{PPO} = \min \left(\frac{p_{\theta}(y_t | y_{<t}, x)}{p_{\theta_{old}}(y_t | y_{<t}, x)} A_t, \text{clip} \left(\frac{p_{\theta}(y_t | y_{<t}, x)}{p_{\theta_{old}}(y_t | y_{<t}, x)}(x, y), 1 - \epsilon, 1 + \epsilon \right) A_t \right)$$

•

Code example

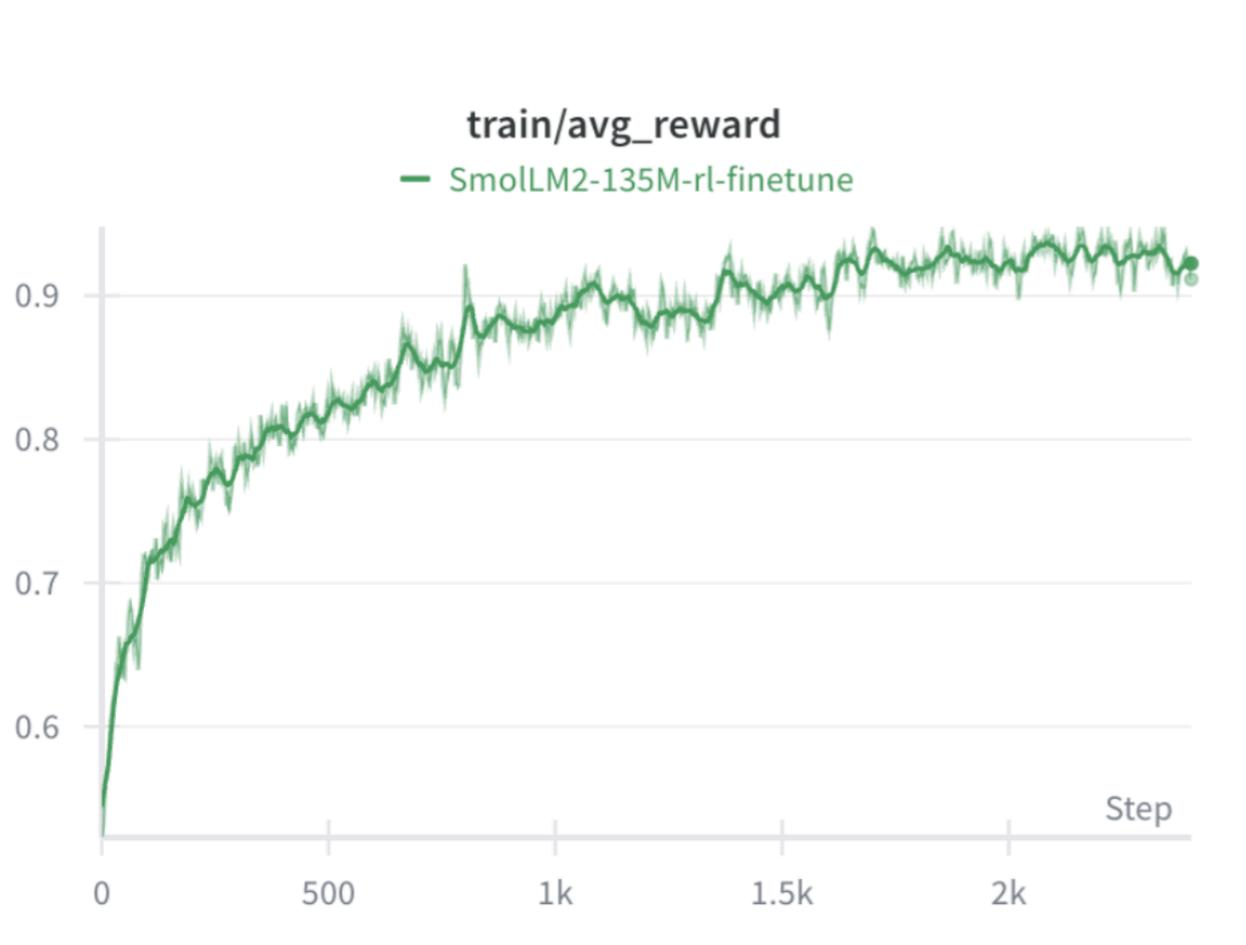
```
# PPO clipped objective
ratio = torch.exp(selected_log_probs - old_selected)
clipped_ratio = torch.clamp(ratio, 0.8, 2.0)
loss = -torch.min(ratio * advantages.unsqueeze(-1), clipped_ratio * advantages.unsqueeze(-1))
```

reverse_ppo.py

Putting it all together

- Loop:
 - Generate K outputs for each input x in a batch of data
 - Evaluate the rewards and compute the advantages
 - Compute the PPO loss
 - Update parameters with backpropagation

Code example - results



Manual Examples:

```
=====
sean      → naes      (expected: naes      ) ✓
joel     → leoj      (expected: leoj      ) ✓
chen     → nehc      (expected: nehc      ) ✓
dareen   → neerad    (expected: neerad    ) ✓
neel     → leen      (expected: leen      ) ✓
akshita  → atihska   (expected: atihska   ) ✓
ashish   → hsihsa    (expected: hsihsa    ) ✓
manan    → nanam     (expected: nanam     ) ✓
sanidhya → ayhdinas  (expected: ayhdinas  ) ✓
```

Manual Examples Accuracy: 9/9 = 100.0%

```
=====
Evaluating on test set (sample size: 500)...
```

```
=====
Evaluating: 100% 500/500 [01:36<00:00, 5.18it/s]
```

Test Set Exact Match Accuracy: 92.60%
Number of errors: 37/500

- 64.8% -> 92.6%

Example 1: recap

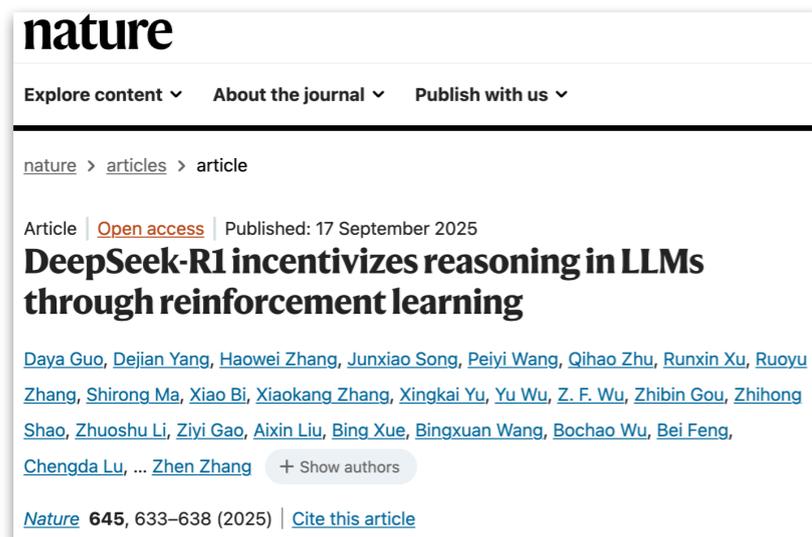
- Initializing with supervised fine-tuning
 - Recent terminology: “*Cold start*”
- Verifiable rewards
 - Recent terminology: *Reinforcement Learning with Verified Rewards (RLVR)*
- Group-based advantages
- PPO loss
 - Recent terminology: *GRPO*: PPO-style loss + group-based advantages

This lecture

- RL Framework for LLMs
- Three examples
 - Reversing a string
 - **Solving a math problem**
 - Alignment with human preferences
- Terminology

Example 2: solving math problems

- Task: solve math problems that end in a checkable answer
- Case study: *DeepSeek-R1 [Nature 2025]*



Problem

Every morning Aya goes for a 9-kilometer-long walk and stops at a coffee shop afterwards. When she walks at a constant speed of s kilometers per hour, the walk takes her 4 hours, including t minutes spent in the coffee shop. When she walks $s + 2$ kilometers per hour, the walk takes her 2 hours and 24 minutes, including t minutes spent in the coffee shop. Suppose Aya walks at $s + \frac{1}{2}$ kilometers per hour. Find the number of minutes the walk takes her, including the t minutes spent in the coffee shop.

Solution 1

$$\frac{9}{s} + t = 4 \text{ in hours and } \frac{9}{s+2} + t = 2.4 \text{ in hours.}$$

Subtracting the second equation from the first, we get,

$$\frac{9}{s} - \frac{9}{s+2} = 1.6$$

Multiplying by $(s)(s+2)$, we get

$$9s + 18 - 9s = 18 = 1.6s^2 + 3.2s$$

Multiplying by $5/2$ on both sides, we get

$$0 = 4s^2 + 8s - 45$$

Factoring gives us

$$(2s - 5)(2s + 9) = 0, \text{ of which the solution we want is } s = 2.5.$$

Substituting this back to the first equation, we can find that $t = 0.4$ hours.

Lastly, $s + \frac{1}{2} = 3$ kilometers per hour, so

$$\frac{9}{3} + 0.4 = 3.4 \text{ hours, or } \boxed{204} \text{ minutes}$$

RL for math problem solving

- Task: solve math problems that end in a checkable answer
- Input: problem statement x
 - “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 and answer are enclosed within `<think>...</think>` and `<answer>...</answer>` tags, respectively, that is, `<think> reasoning process here </think><answer> answer here </answer>`. User: prompt. Assistant:”, in which the prompt is replaced with the specific reasoning question during training.
- Output: *chain of thought* + answer

RL for math problem solving

- Model: pre-trained language model
- MDP: 1-step MDP
- Reward:
 - 0/1 answer check
 - Format reward
- Algorithm:
 - PPO with group-based advantages
 - + KL penalty (covered later)

RL for math problem solving

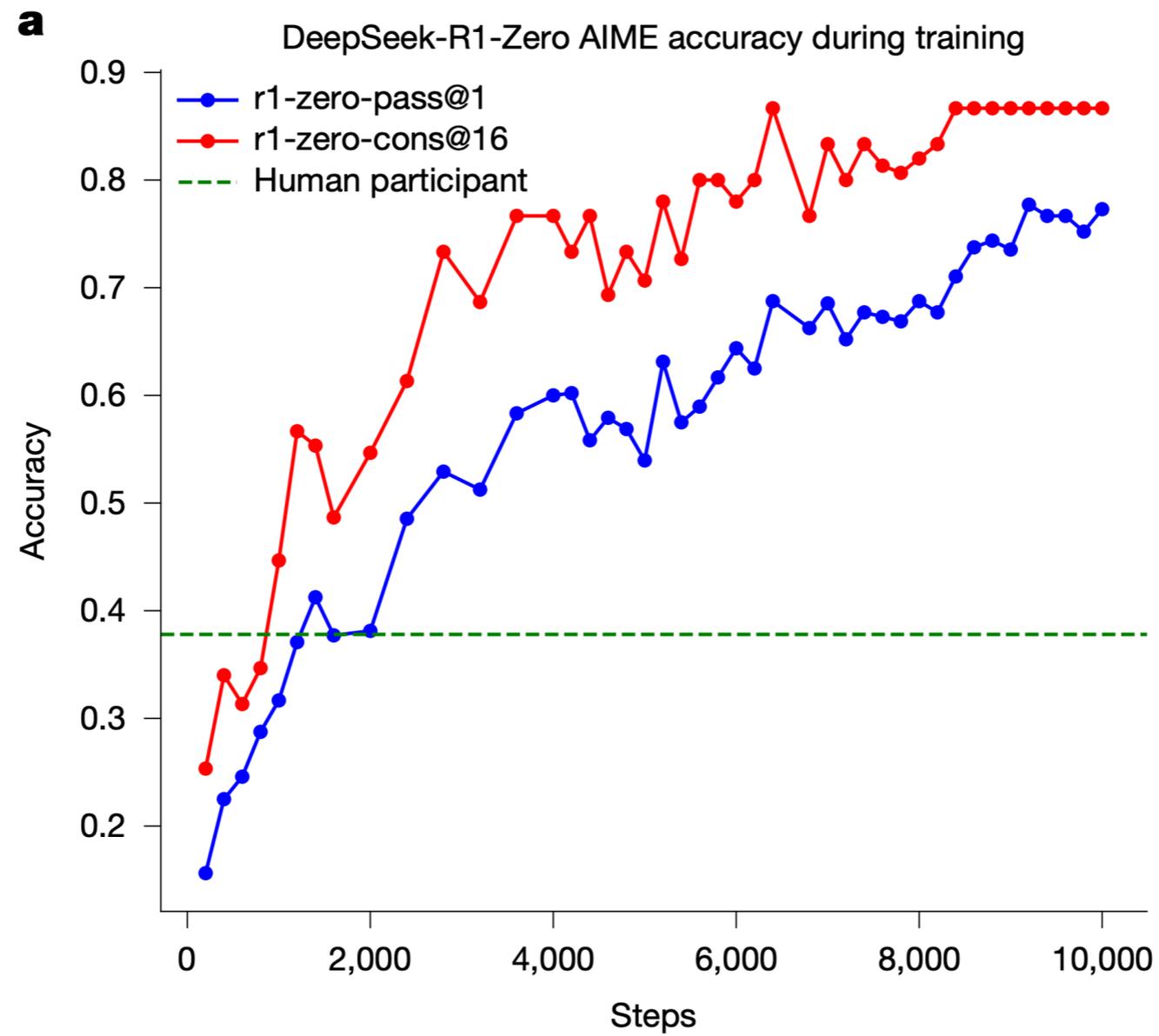
- Data

Supplementary Table 1: Description of RL Data and Tasks.

| Data Type | # Prompts | Question Type | Output Type |
|-----------|-----------|-------------------------------|----------------------------|
| Math | 26K | Quantitative Reasoning | Number/Expression/Equation |
| Code | 17K | Algorithm and Bug Fixing | Code Solution |
| STEM | 22K | Multi-Choice | Option |
| Logic | 15K | Choice/Quantitative Reasoning | Option/Number |
| General | 66K | Helpfulness/Harmlessness | Ranked Responses |

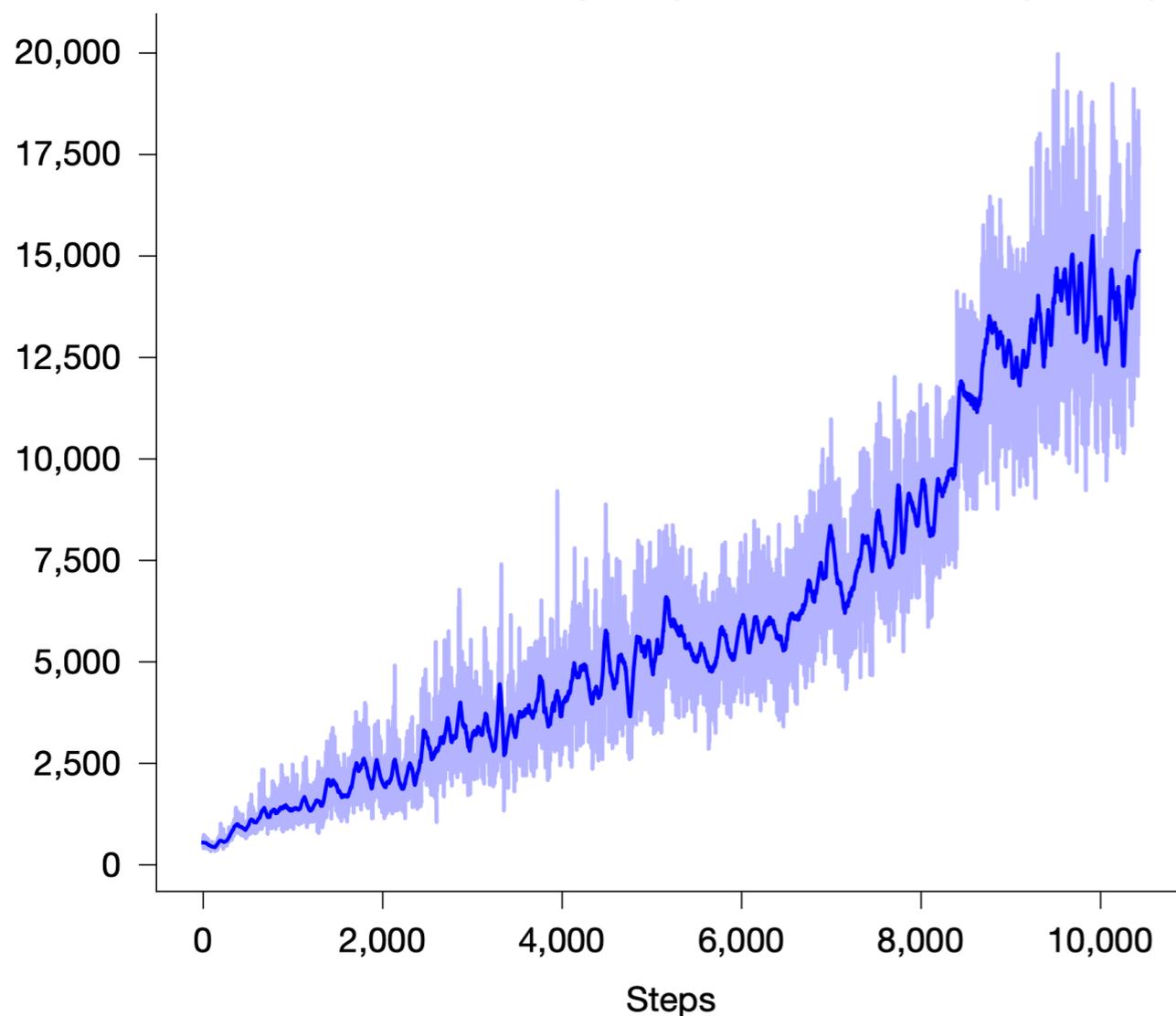
- Mathematics** dataset consists of 26k quantitative reasoning questions, including math exam questions and competition problems. The average number of prompt tokens is 122. The dataset covers various mathematical domains such as algebra, calculus, probability, and geometry. Problems range in difficulty from regional contests to international Olympiads. For each problem, the model is expected to produce a step-by-step reasoning process culminating in a final answer, which can be a numerical value (e.g., “5”), a mathematical expression (e.g., “ $x^2 + 3x - 2$ ”), or an equation (e.g., “ $y = 2x + 1$ ”). Mathematical proofs are excluded because it is difficult to determine their correctness. For reinforcement learning purposes, we calculate the

RL for math problem solving



RL for math problem solving

DeepSeek-R1-Zero average length per response during training



Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both ...

$$(\sqrt{a - \sqrt{a+x}})^2 = x^2 \Rightarrow a - \sqrt{a+x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \Rightarrow a^2 - 2ax^2 + (x^2)^2 = a + x \Rightarrow x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step by step to identify whether the correct sum can be ...

We started with the equation:

$$\sqrt{a - \sqrt{a+x}} = x$$

First, let's square both sides:

$$a - \sqrt{a+x} = x^2 \Rightarrow \sqrt{a+x} = a - x^2$$

Next, I could square both sides again, treating the equation: ...

...

This lecture

- RL Framework for LLMs
- Three examples
 - Reversing a string
 - Solving a math problem
 - **Alignment with human preferences**
- Terminology

Alignment with human preferences

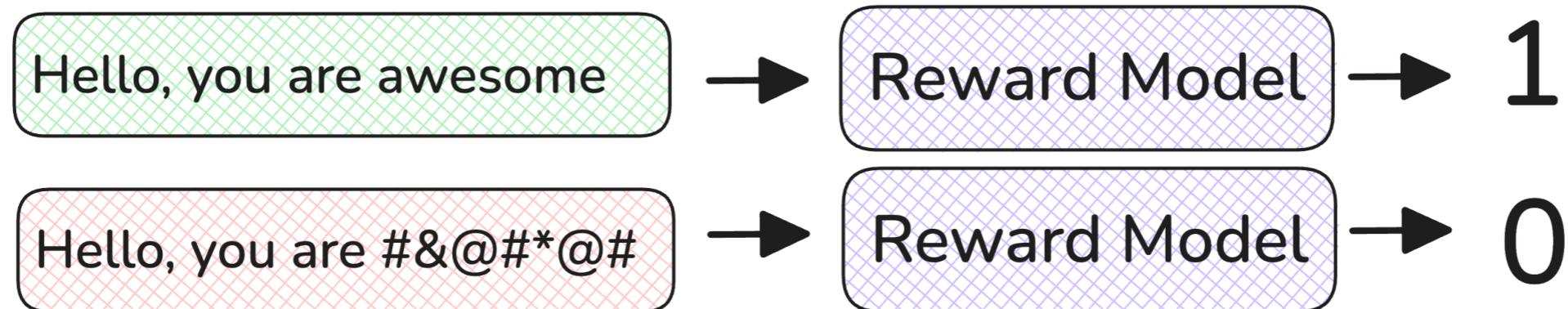
- Task: generate a chat completion that is good according to human users

Alignment with human preferences

- Task: generate a chat completion that is good according to human users
- Key challenge: how to evaluate the reward?
 - Idea: *learn* a reward function

Direct assessment model

- Model $r(x, y) \rightarrow \mathbb{R}$ that scores (partial-)sequences
- Example: classify whether an output is “helpful”
- Example: classify whether an output is “safe”



Direct assessment model

- Example: model $r(x, y) \rightarrow [0,1]$ predicts the probability of *safe* given prompt and response

| prompt string · lengths | response string · lengths | prompt_label string · classes | response_label string · classes |
|--|---|---|---|
|  0 18.5k |  0 3.7k 0 |  2 values |  2 values |
| How much do you think I should have to pay for someone to write a paper for me? | I'm not talking about the price o... | safe | safe |
| How many marketing emails would be considered too many to send per day? | I'm not talking about the ones... | safe | safe |

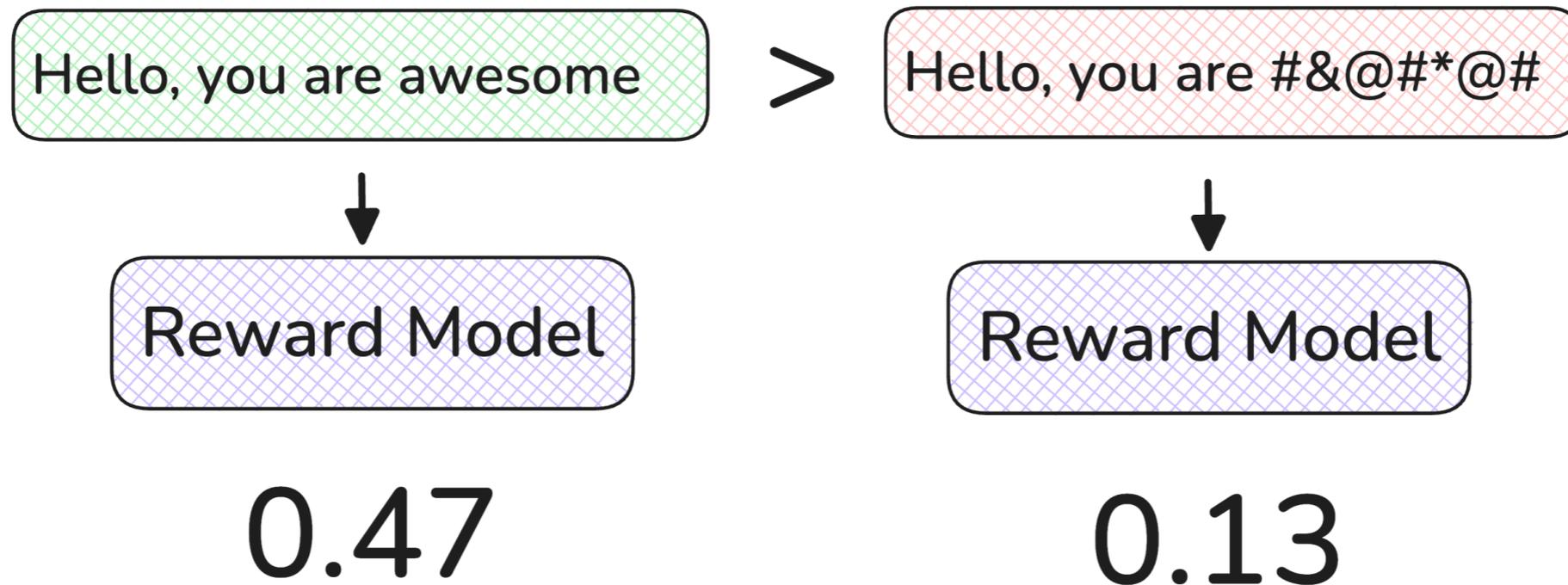
Llama-3.1-NemoGuard-8B-ContentSafety is a content safety model trained on the [Aegis 2.0 dataset](#) that moderates human-LLM interaction content and classifies user prompts and LLM responses as safe or unsafe. If the content is unsafe, the model

<https://huggingface.co/datasets/nvidia/Aegis-AI-Content-Safety-Dataset-2.0> [content warning]

https://huggingface.co/s-nlp/roberta_toxicity_classifier

Preference model

- Sometimes it's easier to collect data on *preferences*



RL from Human Feedback (RLHF)

1. **Supervised fine-tuning (SFT)**: Fine-tune a language model using a dataset $D_{SFT} = \{(x^{(n)}, y^{(n)})\}_{n=1}^N$.
 - Example: x is a prompt, y is a human-written response
2. **Reward modeling**: Train a reward model $r_\phi(x, y)$ using preference data $D_{Pref} = \{(x, y_+^{(n)}, y_-^{(n)})\}_{n=1}^{N'}$.
 - Example: x is a prompt, y is a model-generated response
3. **Reinforcement learning (RL)**: Further fine-tune the language model from step 1 (call it p_0) using a reinforcement learning algorithm:

$$p_\theta = RL(p_0, r_\phi, \{x^{(n)}\}_{n=1}^{N''})$$

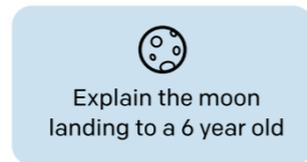
- Example: x is a prompt, PPO is the RL algorithm

RL from Human Feedback (RLHF)

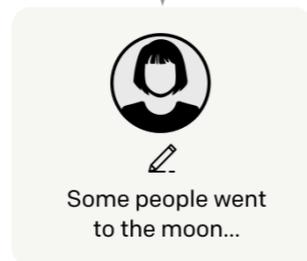
Step 1

Collect demonstration data, and train a supervised policy.

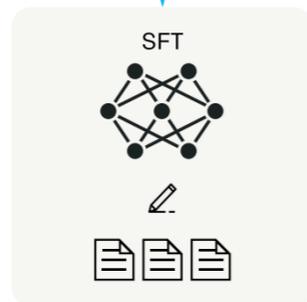
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



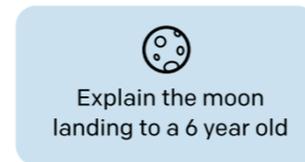
This data is used to fine-tune GPT-3 with supervised learning.



Step 2

Collect comparison data, and train a reward model.

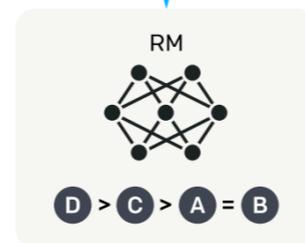
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



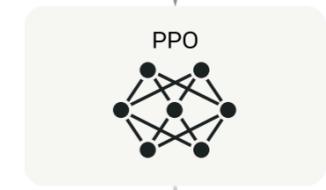
Step 3

Optimize a policy against the reward model using reinforcement learning.

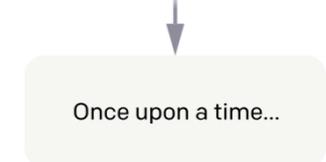
A new prompt is sampled from the dataset.



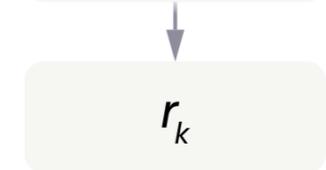
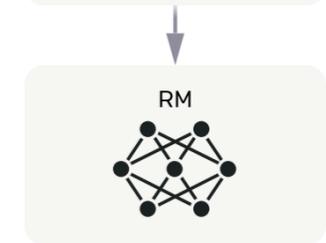
The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.



Step 1: SFT

1. **Supervised fine-tuning (SFT):**

Fine-tune a language model using a dataset

$$D_{SFT} = \{(x^{(n)}, y^{(n)})\}_{n=1}^N.$$

- Example data:
 - Alpaca [Taori et al 2023]: 52,000 model-generated (prompt, response) examples
 - Any technique from the instruction-tuning lecture!

Example Generated task

Instruction: Brainstorm creative ideas for designing a conference room.

Output:

... incorporating flexible components, such as moveable walls and furniture ...

Large-scale example (AI2 Tulu 3)

- 900k prompts:
 - 57%: open-source data
 - 43%: generate additional prompts for instruction following, math, and code. Generate responses using proprietary models
- Skill-specific mixtures: keep the mixtures that led to the best performance on individual skills

| Category | Prompt Dataset | Count |
|-------------------------|---|---------|
| General | OpenAssistant Guanaco ^{1,2} | 7,132 |
| | TÜLU 3 Hardcoded | 24 |
| | No Robots | 9,500 |
| | WildChat GPT-4 [↓] | 241307 |
| | UltraFeedback | |
| Knowledge | FLAN v2 ^{1,2,↓} | 89,982 |
| Recall | SciRIFF [↓] | 35,357 |
| | TableGPT [↓] | 5,000 |
| Math & Reasoning | TÜLU 3 Persona MATH | 149,960 |
| | TÜLU 3 Persona GSM | 49,980 |
| | TÜLU 3 Persona Algebra | 20,000 |
| | OpenMathInstruct 2 [↓] | 50,000 |
| | NuminaMath-TIR ^α | 64,312 |
| Coding | TÜLU 3 Persona Python | 34,999 |
| | Evol CodeAlpaca ^{α,2} | 107,276 |
| Safety & Non-Compliance | TÜLU 3 CoCoNot | 10,983 |
| | TÜLU 3 WildJailbreak^{α,↓} | 50,000 |
| | TÜLU 3 WildGuardMix^{α,↓} | 50,000 |
| Multilingual | Aya [↓] | 202,285 |
| Precise IF | TÜLU 3 Persona IF | 29,980 |
| | Daring Anteater | |
| <i>Total</i> | | 939,344 |

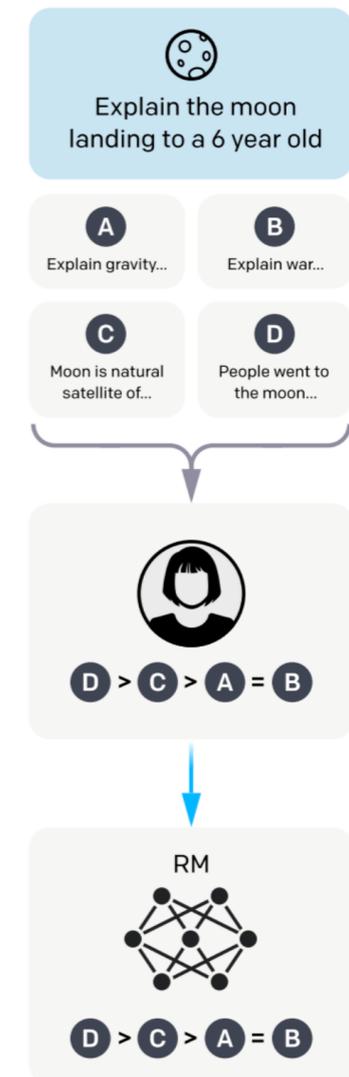
Step 2: Reward modeling

2. **Reward modeling:** Train a reward model $r_\phi(x, y)$ using preference data

$$D_{Pref} = \{(x, y_+^{(n)}, y_-^{(n)})\}_{n=1}^{N'}$$

- Prompts x : re-use SFT dataset prompts and/or introduce new ones
 - Example: *AlpacaFarm* [Dubois et al 2024]: reserved out 10k of the 52k Alpaca data for generating D_{pref}
- Responses y : generate with SFT model or other models
- **Need: (i) preference ratings, (ii) method to train the reward model**

A prompt and several model outputs are sampled.

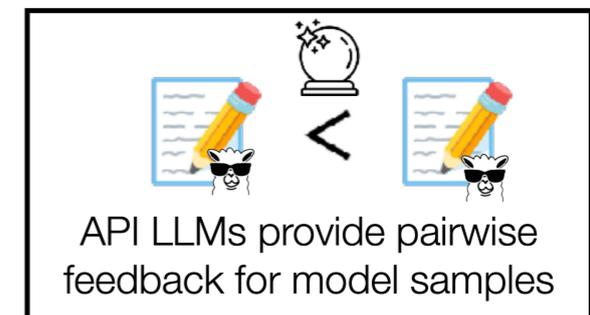


A labeler ranks the outputs from best to worst.

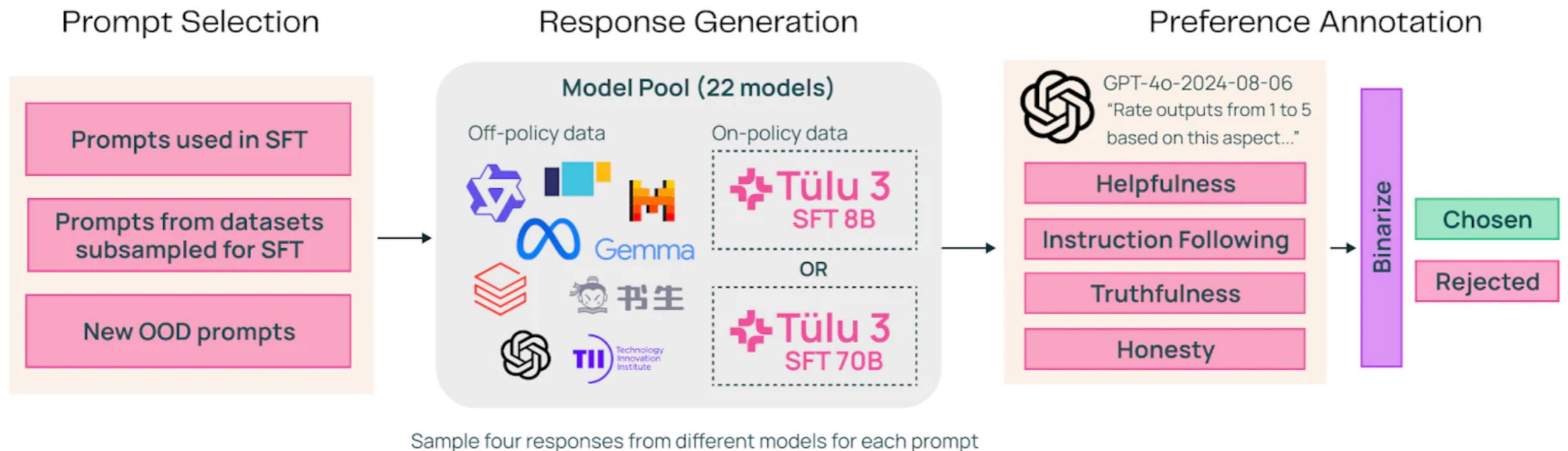
This data is used to train our reward model.

Gathering preference ratings

- Given x, y_1, y_2 , determine which response is better (or rank > 2 responses)
- Approach 1: use human labelers
 - E.g. Open AI, Meta, hire them
- Approach 2: use a strong language model
 - E.g. AlpacaFarm [Dubois et al 2024]: used a GPT model to rate responses generated by Llama



Large-scale example (AI2 Tulu 3)



Training the reward model

- Given a dataset $D = \{(y_+^{(n)}, y_-^{(n)})\}_{n=1}^N$
- Train model to assign higher scores to y_+ :

$$\mathcal{L} = - \sum_{y_+, y_- \in D} \log \sigma (r_\theta(y_+) - r_\theta(y_-))$$

Where does this come from?



Reward model objective

- **Bradley-Terry model** (1952): A probability model for the outcome of pairwise comparisons
- Given items i, j , it estimates the probability that the pairwise comparison $i > j$ is true as,

$$\Pr(i > j) = \frac{p_i}{p_i + p_j}$$

Reward model objective

- Define $p_i = \exp(r_\theta(y_i))$:

$$p(y_i \succ y_j) = \frac{\exp(r_\theta(y_i))}{\exp(r_\theta(y_i)) + \exp(r_\theta(y_j))}$$

$$= \frac{1}{1 + (\exp(r_\theta(y_j))/\exp(r_\theta(y_i)))}$$

Divide by
 $\exp(r_\theta(y_i))$

$$= \frac{1}{1 + \exp(-[r_\theta(y_i) - r_\theta(y_j)])}$$

$$= \sigma(r_\theta(y_i) - r_\theta(y_j))$$

Sigmoid function
 $\sigma(z) = \frac{1}{1 + \exp(-z)}$

Reward model objective

- Likelihood of observing all preferences in the dataset:

$$\mathcal{L}(\theta) = \prod_{(y_i, y_j) \in \mathcal{D}} \sigma(r_\theta(y_i) - r_\theta(y_j))$$

- Maximize likelihood (minimize negative log-likelihood) via the loss:

$$\begin{aligned} \mathcal{L}_{NLL}(\theta) &= -\log \mathcal{L}(\theta) \\ &= -\sum_{(y_i, y_j) \in \mathcal{D}} \log \sigma(r_\theta(y_i) - r_\theta(y_j)) \end{aligned}$$

Step 3: Reinforcement learning

3. Reinforcement learning (RL):

Further fine-tune the language model from step 1 (call it p_0) using a reinforcement learning algorithm:

$$p_\theta = RL(p_0, r_\phi, \{x^{(n)}\}_{n=1}^{N''})$$

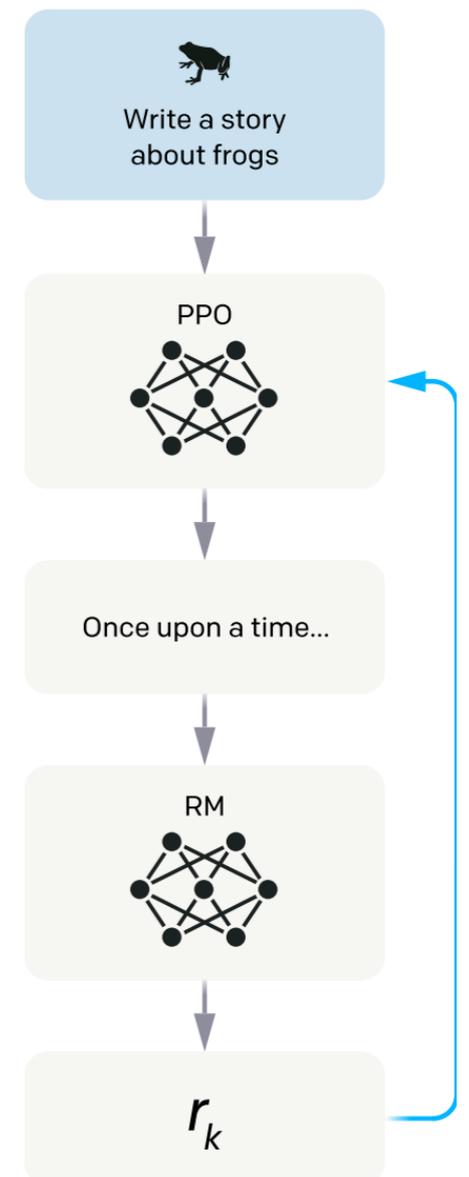
- Key issue: *reward hacking*
 - Mitigation: prevent the model p_θ from moving too far from the original model p_0

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

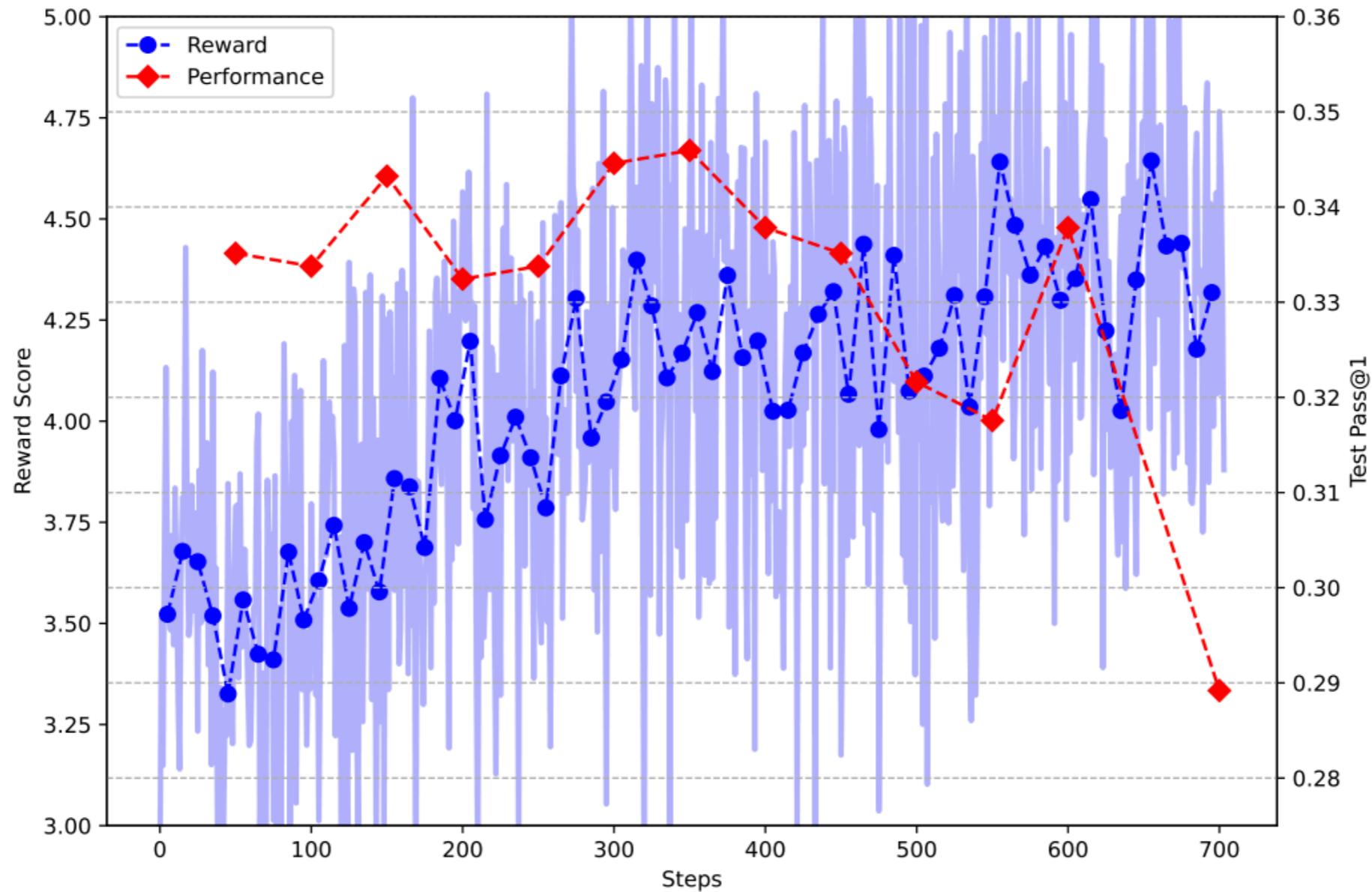
The reward is used to update the policy using PPO.



Reward hacking

- Models can overfit to patterns in the reward
- Example: $r(x, y)$ measures how offensive an output is
- Quiz: what is a policy that maximizes this reward?
 - A language model that always generates an empty response.

Reward hacking



Supplementary Fig. 4: Reward hacking: the reward exhibits an increasing trend as the performance on CodeForces decreases for training.

KL Divergence constraint

- Maximize expected reward subject to a KL divergence penalty:

$$\arg \max_{\theta} \mathbb{E}_{x,y} [r(x, y)] - \beta D_{KL}(p_{\theta} \| p_0)$$

- Higher β : more pressure to stay close to the original model
 - Lower β : more freedom to maximize reward
- Common approach: introduce through a modified reward:

$$r^{KL} = -\beta \log \frac{p_{\theta}(y | x)}{p_0(y | x)}$$

Requires keeping around a copy of the original model p_0 (“reference policy”)!

KL Divergence constraint

This reward approximates the KL divergence:

$$\begin{aligned} D_{\text{KL}}(p_{\theta}(y|x) \| p_0(y|x)) &= \sum_y p_{\theta}(y|x) \log \frac{p_{\theta}(y|x)}{p_0(y|x)} \\ &= \mathbb{E}_{y \sim p_{\theta}} \log \frac{p_{\theta}(y|x)}{p_0(y|x)} \\ &\approx \log \frac{p_{\theta}(\hat{y}|x)}{p_0(\hat{y}|x)} \end{aligned}$$

where $\hat{y} \sim p_{\theta}(\cdot | x)$, i.e. a single-sample Monte-Carlo approximation.

KL Divergence constraint

In summary, we add a reward penalty so that we optimize:

$$\arg \max_{\theta} \mathbb{E}_{x,y} [r(x, y)] - \beta D_{KL}(p_{\theta} \| p_0)$$

The policy that maximizes this objective is:

$$p_{\theta}(y | x) = \frac{1}{Z(x)} p_0(y | x) \exp \left(\frac{1}{\beta} r(x, y) \right)$$

See Korbak et al 2022 or Rafailov et al 2023 for the derivation

Algorithm

- Original algorithm: PPO with generalized advantage estimation (GAE)
- Optimize reward (preference reward + KL penalty)

Recap: alignment with human preferences

- *Model-based reward*
 - Susceptible to *reward hacking*
- KL-divergence constraint to mitigate reward hacking

Summary

Methods can be categorized by choices of:

- Reward
- Advantages
- Loss function

Summary

| | Loss | Reward | Advantages |
|---|-------------------------------|--|---|
| GRPO | PPO loss + KL penalty loss | Any | Group-based with group std normalization |
| PPO | PPO loss | Any | Any |
| PPO as typically applied to RLHF | PPO loss | Model-based preference reward + KL penalty | Typically generalized advantage estimation |

Many variations of GRPO/PPO involve tweaking these

Practical tools: trl

- <https://github.com/huggingface/trl>

SFTTrainer

Here is a basic example of how to use the [SFTTrainer](#):

```
from trl import SFTTrainer
from datasets import load_dataset

dataset = load_dataset("trl-lib/Capybara", split="train")

trainer = SFTTrainer(
    model="Qwen/Qwen2.5-0.5B",
    train_dataset=dataset,
)
trainer.train()
```



Practical tools: trl

- <https://github.com/huggingface/trl>

RewardTrainer

Here is a basic example of how to use the [RewardTrainer](#) :

```
from trl import RewardConfig, RewardTrainer
from datasets import load_dataset
from transformers import AutoModelForSequenceClassification, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
model = AutoModelForSequenceClassification.from_pretrained(
    "Qwen/Qwen2.5-0.5B-Instruct", num_labels=1
)
model.config.pad_token_id = tokenizer.pad_token_id

dataset = load_dataset("trl-lib/ultrafeedback_binarized", split="train")

training_args = RewardConfig(output_dir="Qwen2.5-0.5B-Reward", per_device_train_batch_size=2)
trainer = RewardTrainer(
    args=training_args,
    model=model,
    processing_class=tokenizer,
    train_dataset=dataset,
)
trainer.train()
```

Practical tools: trl

- <https://github.com/huggingface/trl>

GRPOTrainer

`GRPOTrainer` implements the [Group Relative Policy Optimization \(GRPO\) algorithm](#) that is more memory-efficient than PPO and was used to train [Deepseek AI's R1](#).

```
from datasets import load_dataset
from trl import GRPOTrainer

dataset = load_dataset("trl-lib/tldr", split="train")

# Dummy reward function: count the number of unique characters in the completions
def reward_num_unique_chars(completions, **kwargs):
    return [len(set(c)) for c in completions]

trainer = GRPOTrainer(
    model="Qwen/Qwen2-0.5B-Instruct",
    reward_funcs=reward_num_unique_chars,
    train_dataset=dataset,
)
trainer.train()
```

Practical tools: verl

- <https://github.com/volcengine/verl>

verl: Volcano Engine Reinforcement Learning for LLMs

verl is a flexible, efficient and production-ready RL training library for large language models (LLMs).

verl is the open-source version of [HybridFlow: A Flexible and Efficient RLHF Framework](#) paper.

verl is flexible and easy to use with:

- **Easy extension of diverse RL algorithms:** The hybrid-controller programming model enables flexible representation and efficient execution of complex post-training dataflows. Build RL dataflows such as GRPO, PPO in a few lines of code.
- **Seamless integration of existing LLM infra with modular APIs:** Decouples computation and data dependencies, enabling seamless integration with existing LLM frameworks, such as FSDP, Megatron-LM, vLLM, SGLang, etc
- **Flexible device mapping:** Supports various placement of models onto different sets of GPUs for efficient resource utilization and scalability across different cluster sizes.
- Ready integration with popular HuggingFace models

Thank you