CSE 574 Planning and Learning Methods in AI

Ransalu Senanayake

Week 8

GPT-3

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

“these models can also generate outputs that are untruthful, toxic, or reflect harmful sentiments.” - OpenAI

Language Models are Few-Shot Learners

Tom B. Brown* Benjamin Mann* Nick Ryder* Melanie Subbiah*

Jared Kaplan† Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry

Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan

Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter

Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray

Benjamin Chess Jack Clark Christopher Berner

Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei

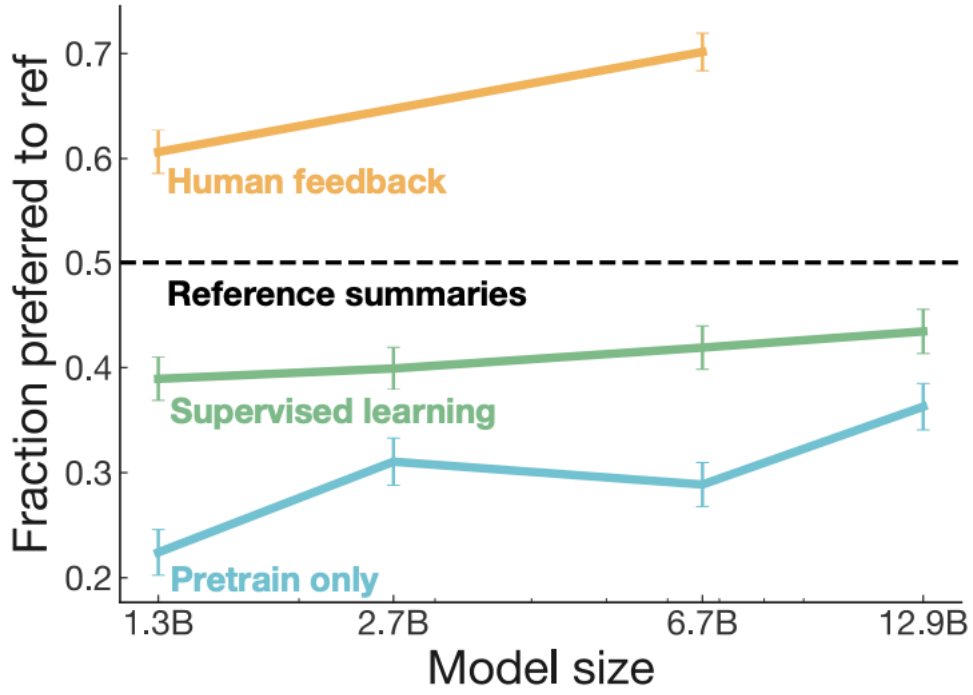
OpenAI

NeurIPS’20

How to align outputs with human preferences?

Learning to summarize from human feedback

Nisan Stiennon* Long Ouyang* Jeff Wu* Daniel M. Ziegler* Ryan Lowe*
Chelsea Voss* Alec Radford Dario Amodei Paul Christiano*
OpenAI



NeurIPS'20

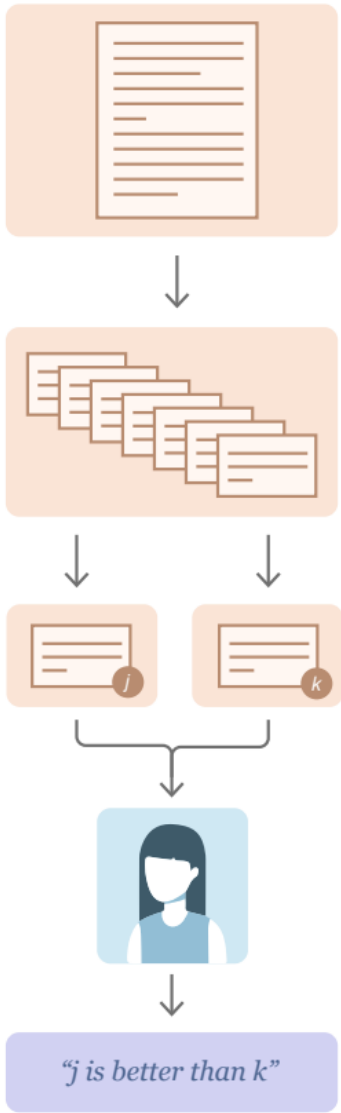
❶ Collect human feedback

A Reddit post is sampled from the Reddit TL;DR dataset.

Various policies are used to sample a set of summaries.

Two summaries are selected for evaluation.

A human judges which is a better summary of the post.

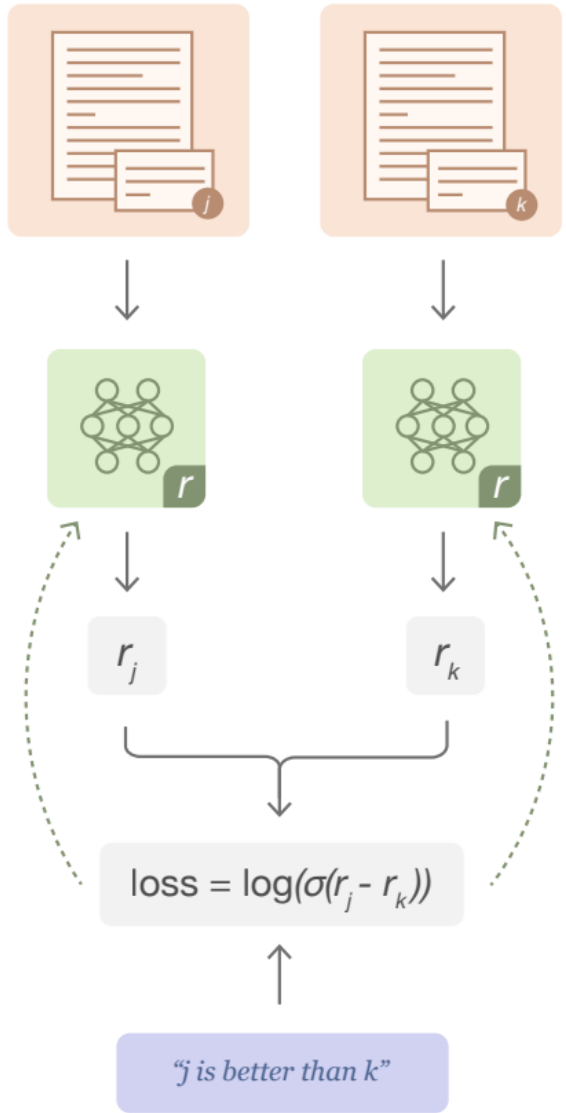


❷ Train reward model

One post with two summaries judged by a human are fed to the reward model.

The reward model calculates a reward *r* for each summary.

The loss is calculated based on the rewards and human label, and is used to update the reward model.



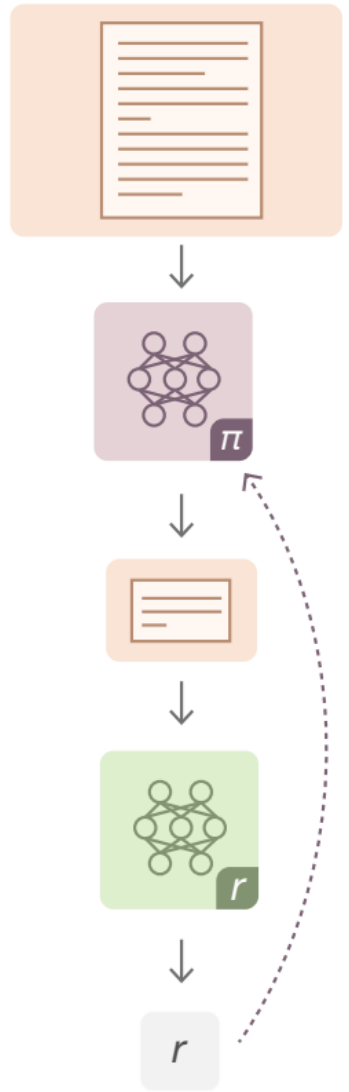
❸ Train policy with PPO

A new post is sampled from the dataset.

The policy π generates a summary for the post.

The reward model calculates a reward for the summary.

The reward is used to update the policy via PPO.



InstructGPT

Training language models to follow instructions with human feedback

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* **Ryan Lowe***

OpenAI

NeurIPS'22

Supervised Fine Tuning (SFT)

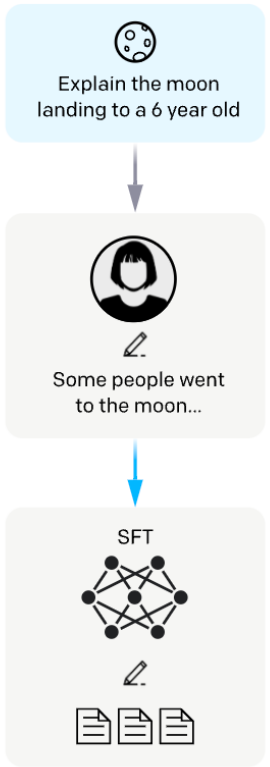
Step 1

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.



Reward Modeling (RM)

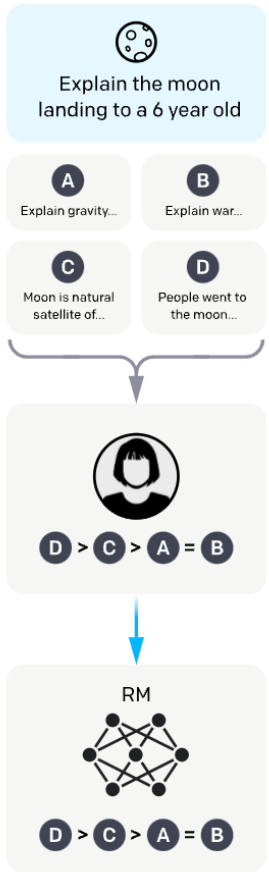
Step 2

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.



Reinforcement Learning (RL)

Step 3

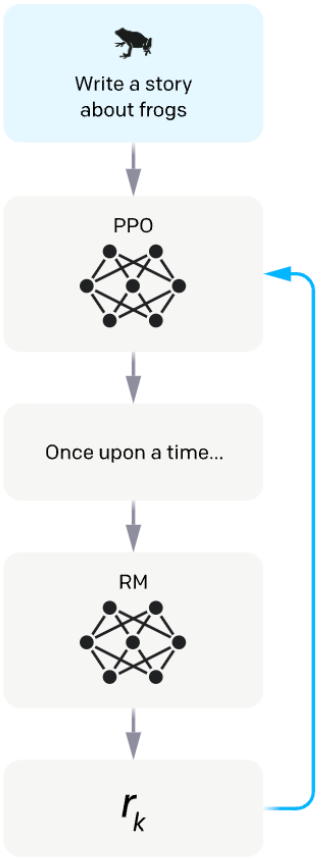
Optimize a policy against the reward model using reinforcement learning.

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.



https://openai.com/research/instruction-following

Policy Optimization Methods

1. Policy iteration
   - Algorithm: Given a policy, estimate the value function. Then, improve the policy. Repeat until convergence.
   - Note: It's model-based. So, we need to know the model! Not for big state-spaces.

```
1. Policy evaluation
2. Policy improvement
3. Repeat
```

2. Gradient-free
   - Algorithm: Explore the policy space using BO, CEM, etc. to maximize the total reward, considering it's a black box
   - Note: It's model free. It can sometimes be sample inefficient and sensitive to hyperparameters.

```
1. Consider a parameterized distrib
   for θ
2. Get samples and evaluate the obj
3. Keep the elite samples (large
   obj)
4. Re-estimate the parameters of
   the distrib and repeat
```

3. **Policy gradient**
   - Algorithm: Get the gradient of the expected rewards w.r.t. policy parameters
   - Note: It's model free. It can be sample inefficient. Variance of the gradient estimations needs to be controlled.

```
1. Do gradient update ∇ log π_θ (a_t|s_t) Â_t
2. Repeat
```

Policy Gradient Methods

1. Vanilla Policy Gradient
2. Trust Region Policy Optimization (TRPO)
3. Proximal Policy Optimization (PPO 1)
4. **Proximal Policy Optimization-CLIP (PPO 2)**

Vanilla Policy Gradient (VPG)

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

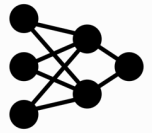
7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

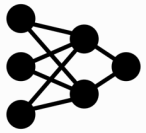
or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**



Policy network



Value network/Critic (optional)

REINFORCE (Monte Carlo Policy Gradient)

High variance in policies

$$A(s,a) = r + \gamma V(s') - V(s)$$

https://spinningup.openai.com/

Derivation

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{\mathrm{E}}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) && \text{Expand expectation} \\
&= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) && \text{Bring gradient under integral} \\
&= \int_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) && \text{Log-derivative trick} \\
&= \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\nabla_\theta \log P(\tau|\theta) R(\tau)] && \text{Return to expectation form} \\
\therefore \nabla_\theta J(\pi_\theta) &= \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right] && \text{Expression for grad-log-prob}
\end{aligned}
$$

https://spinningup.openai.com/

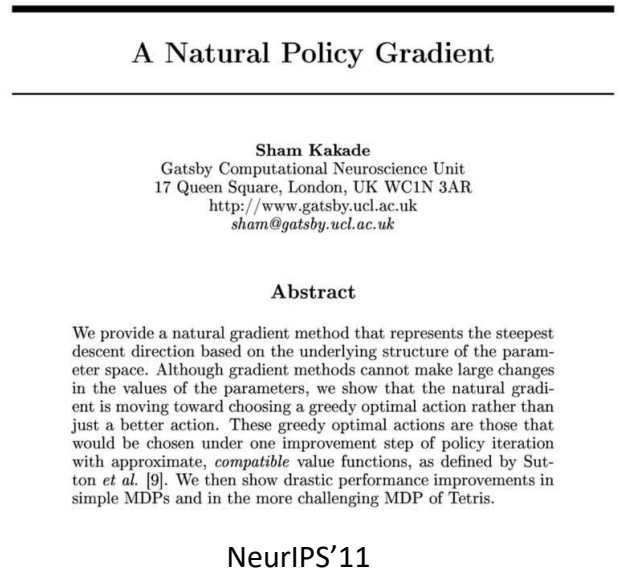
Trust Region Policy Optimization (TRPO)

$$\max_{\pi} \mathbb{E}_{\pi_{old}} \left[\frac{\pi(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s,a) \right]$$

s.t. $\mathbb{E}_{\pi_{old}}[KL[\pi||\pi_{old}]] \leq \epsilon$

Conjugate gradients, need to compute the *natural gradients*/Hessians, inefficient

$$\nabla_{\theta}^{\text{natural}} \mathcal{L}(f(\theta)) = F^{-1} \nabla_{\theta} \mathcal{L}(f(\theta))$$



A Natural Policy Gradient

Sham Kakade
Gatsby Computational Neuroscience Unit
17 Queen Square, London, UK WC1N 3AR
http://www.gatsby.ucl.ac.uk
sham@gatsby.ucl.ac.uk

Abstract

We provide a natural gradient method that represents the steepest descent direction based on the underlying structure of the parameter space. Although gradient methods cannot make large changes in the values of the parameters, we show that the natural gradient is moving toward choosing a greedy optimal action rather than just a better action. These greedy optimal actions are those that would be chosen under one improvement step of policy iteration with approximate, *compatible* value functions, as defined by Sutton *et al.* [9]. We then show drastic performance improvements in simple MDPs and in the more challenging MDP of Tetris.

NeurIPS'11

Recall: Importance sampling

$$\mathbb{E}_{x\sim p}[f(x)] = \int p(x) f(x)\, dx$$

$$= \int p(x) \frac{q(x)}{q(x)} f(x)\, dx$$

$$= \int q(x) \left(\frac{p(x)}{q(x)} f(x)\right) dx$$

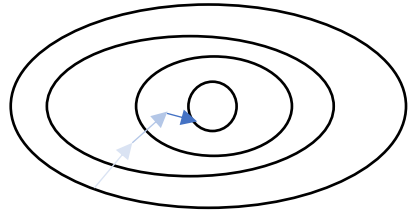
$$= \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)} f(x)\right]$$

Estimating the expectation by sampling from a different distribution.

https://github.com/RansML/RAS-Robust-Autonomous-Systems/blob/master/2_importance_sampling.ipynb

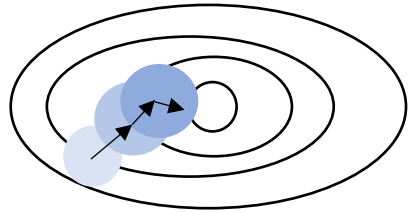
Recall: Iterative gradient-based optimization

- Line search methods: Find the direction of improvement (steepest descent). Then decide the step size along that direction.
  - If f has a simple analytical form for the obj, finding the best step size in each step is possible
  - Otherwise, decide an appropriate step size



$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \nabla_\theta f(\theta_t)$$

- **Trust region methods**: Select the trust region (i.e., max step size). Then find a point (i.e., direction) of improvement.



Approximate the trust region using second order methods

Proximal Policy Optimization (PPO)

VPG

$$\max_{\pi} \pi(a|s) A^{\pi_{old}}(s,a)$$

Sutton, NeurIPS'99

TRPO

$$\max_{\pi} \mathbb{E}_{\pi_{old}} \left[\frac{\pi(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s,a) \right]$$

$$\text{s.t. } \mathbb{E}_{\pi_{old}}[KL[\pi||\pi_{old}]] \leq \epsilon$$

Schulman et al. ICML'15

PPO 1

$$\max_{\pi} \mathbb{E}_{\pi_{old}} \left[\frac{\pi(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s,a) \right] - \beta \left(\mathbb{E}_{\pi_{old}}[KL[\pi||\pi_{old}]] - \epsilon \right)$$

Schulman et al. arXiv'17

PPO 2

$$\max_{\pi} \mathbb{E}_{\pi_{old}} \left[\min \left(\frac{\pi(a|s)}{\pi_{old}(a|s)} A^{\pi_{old}}(s,a), clip \left(\frac{\pi(a|s)}{\pi_{old}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{old}}(s,a) \right) \right]$$

Proximal Policy Optimization (PPO)

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

https://spinningup.openai.com/

Supervised Fine Tuning (SFT)

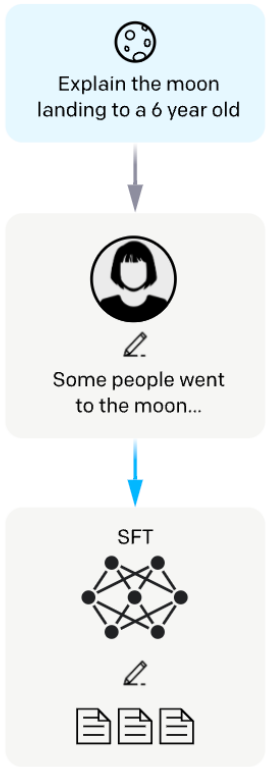
Step 1

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.



Reward Modeling (RM)

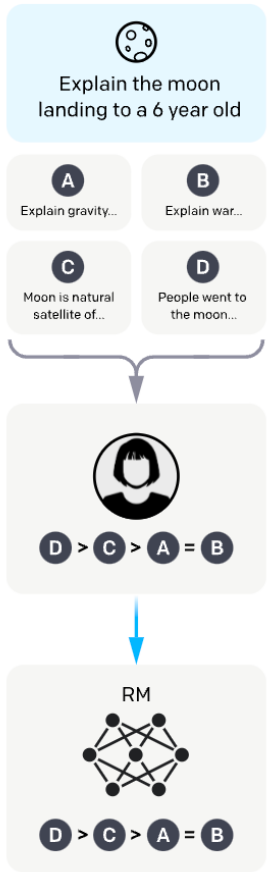
Step 2

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.



Reinforcement Learning (RL)

Step 3

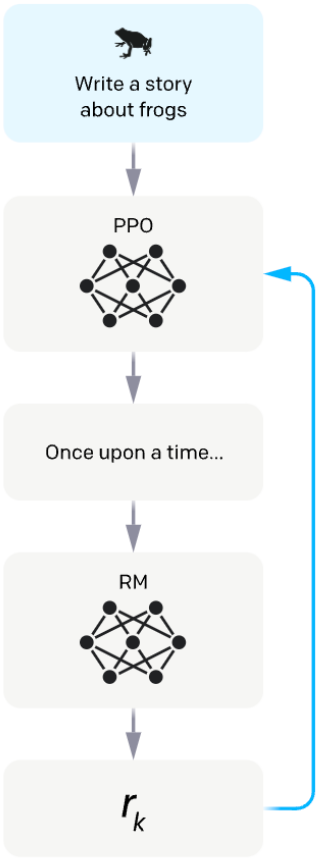
Optimize a policy against the reward model using reinforcement learning.

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.



https://openai.com/research/instruction-following