

# RL for LLMs

## From Parrots to Agents

10-301/601 Guest Lecture

Alex Xie

6/12/2025



# Roadmap

## Why RL for LLMs?

Some RL background

- Policy gradients

- Actor-critic

Proximal Policy Optimization & RLHF

Group-Relative Policy Optimization

# Recap: Pretraining + Fine-tuning

- We begin by **pretraining** LLMs with a language modeling objective on vast amounts of text scraped from the web
  - ➡ Resulting model is *knowledgable* but not yet *helpful*
- Afterwards, we teach the model to accomplish specific tasks/behaviors by doing **supervised fine-tuning (SFT)** on a smaller set of high quality data
  - Where does this data come from?
    - Human demonstrations — resource & time intensive
    - Other LLMs — convenient, assumes existence of better model, legally questionable
    - Itself (rejection fine-tuning) — assumes model is already pretty good at the task
  - SFT teaches a model to **imitate** (humans, better models, or it's own successes)

# Why train LLMs with RL? (or, why is SFT not sufficient?)

- RL allows us to directly optimize for (proxies of) desired constructs that are not differentiable
  - Human preference, factuality, safety, alignment with human values, correctness, etc.
- RL allows the model to explore multiple ways to solve a problem
  - Helps with math, coding, reasoning
  - Also allows the model to learn from both good and bad trajectories
- SFT encourages/cannot fix hallucination (and other pathologies)
  - If we teach the model a new fact in SFT, we're actually teaching it how to make up more new "facts" at test time
  - RL can teach models to abstain based on what they know

# Text generation

Autoregressive  
token-by-token  
generation

My favorite profess or is Henry



# Text generation as RL

State

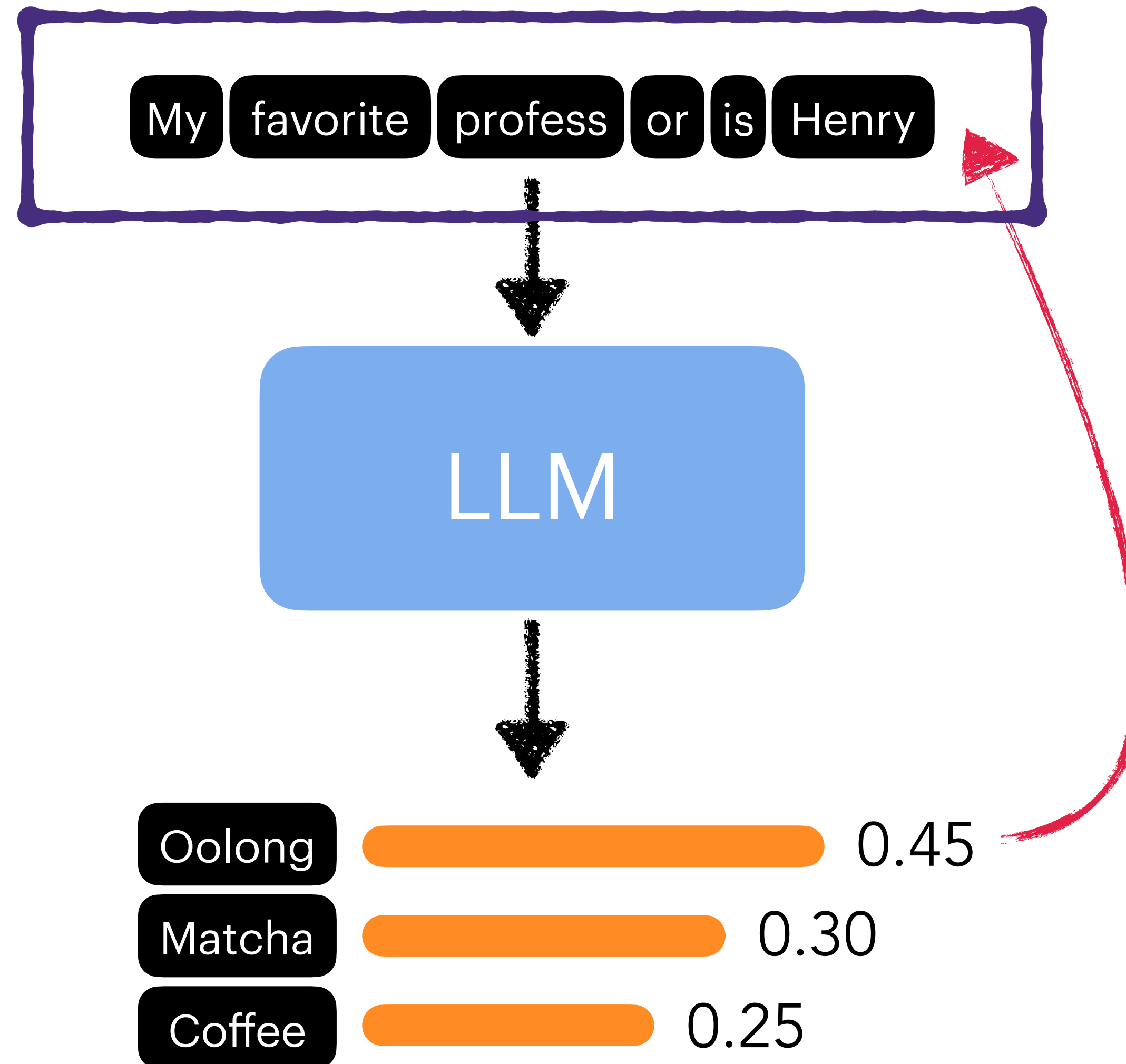
Action

Transition

Reward

Policy

Value



We use the current text prefix as the state

# Text generation as RL

State

**Action**

Transition

Reward

Policy

Value

My favorite profess or is Henry

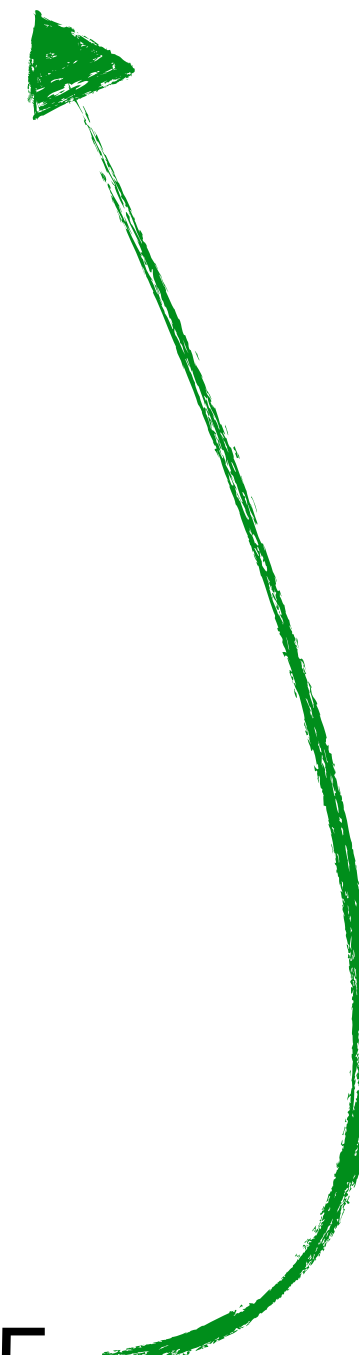


The chosen next token is the action

# Text generation as RL

State  
Action  
**Transition**  
Reward  
  
Policy  
Value

My favorite profess or is Henry



Transitions are trivially deterministic here; we just append the next token to the prefix.



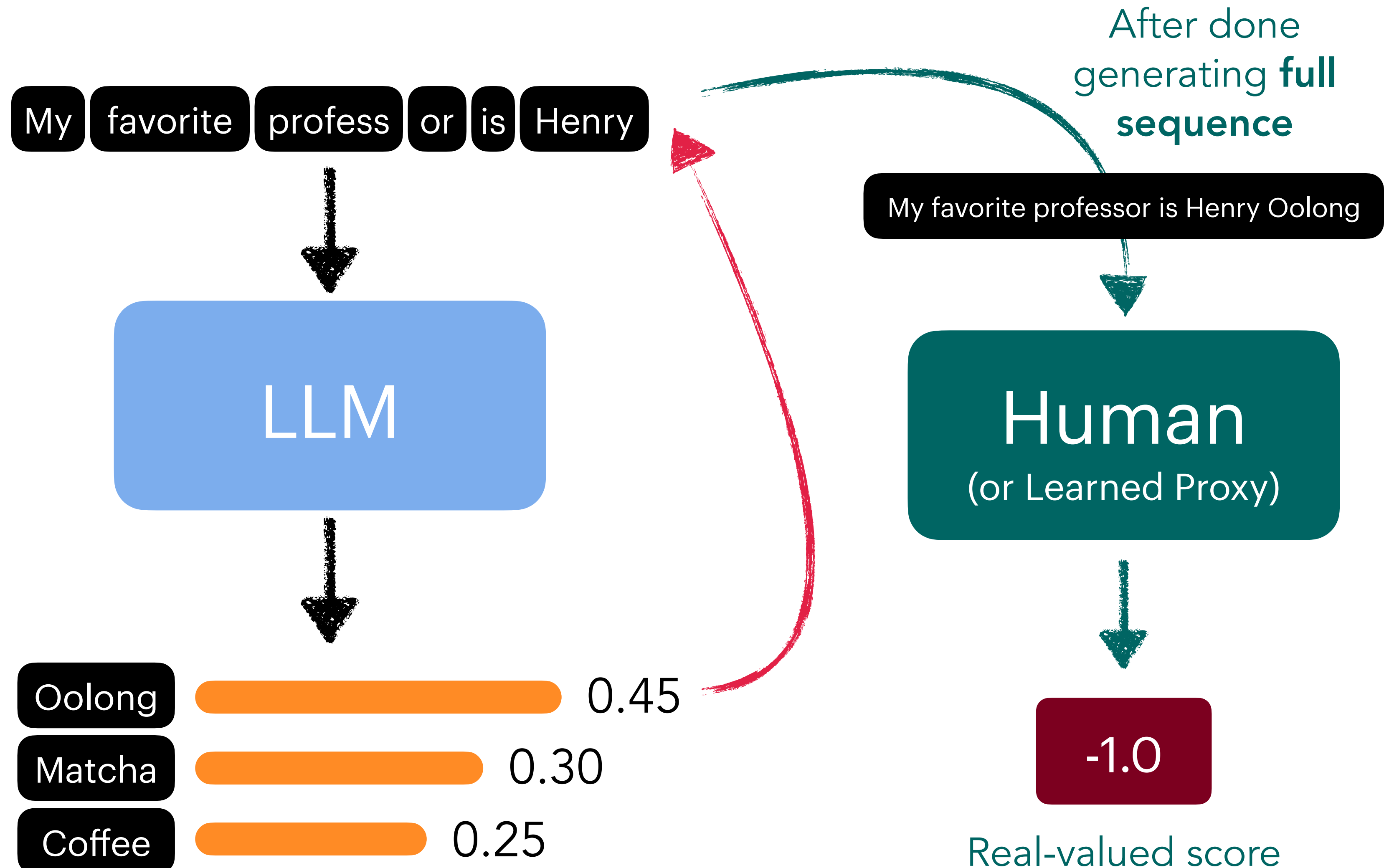
# Text generation as RL

State  
Action  
Transition  
**Reward**

Policy  
Value

In this environment, we only get one reward at the very end.

In some cases we may have intermediate *process* rewards, but we won't consider those today.



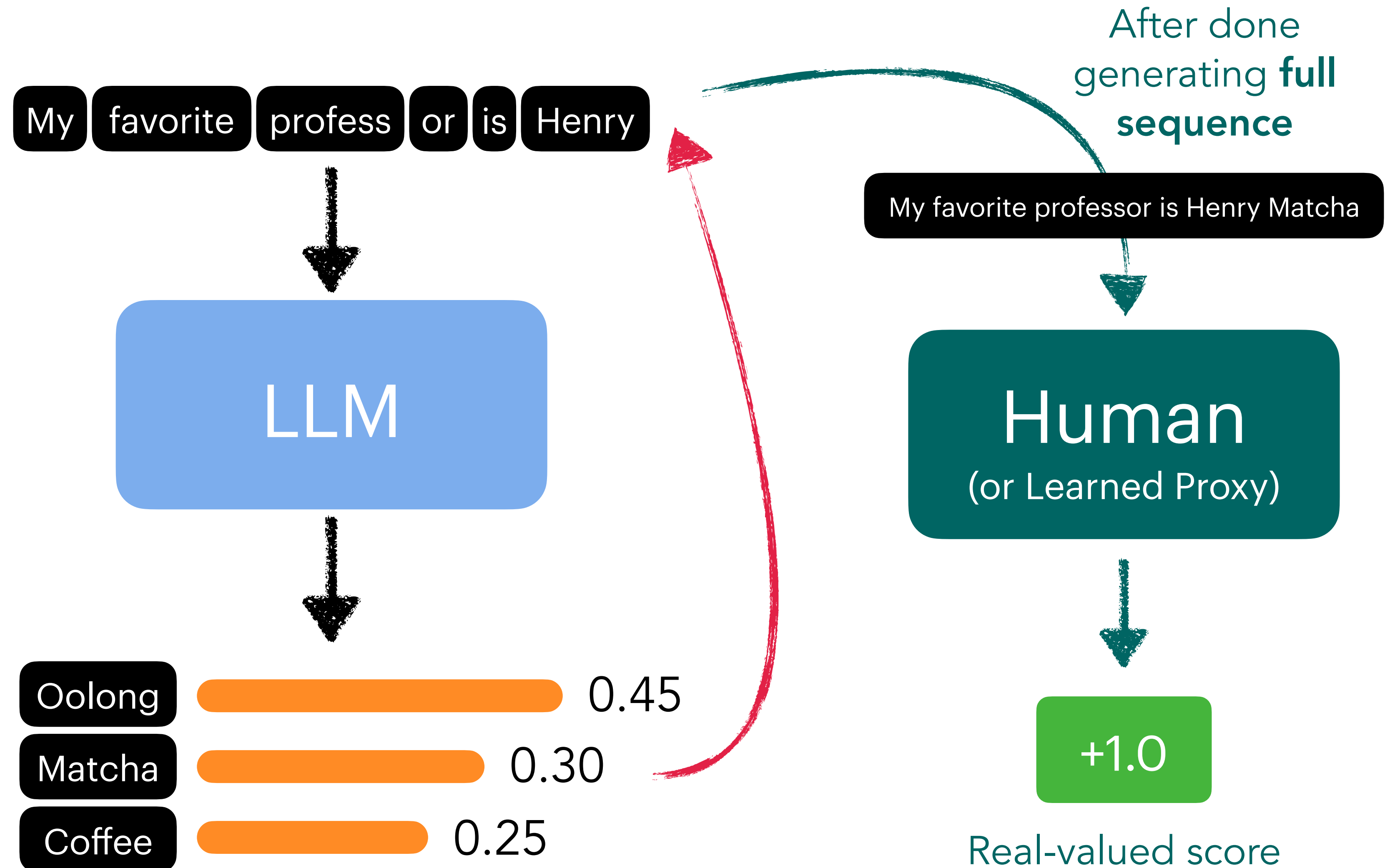
# Text generation as RL

State  
Action  
Transition  
**Reward**

Policy  
Value

Different trajectories (i.e. full sequences) will have different rewards.

(the annotator really likes matcha)



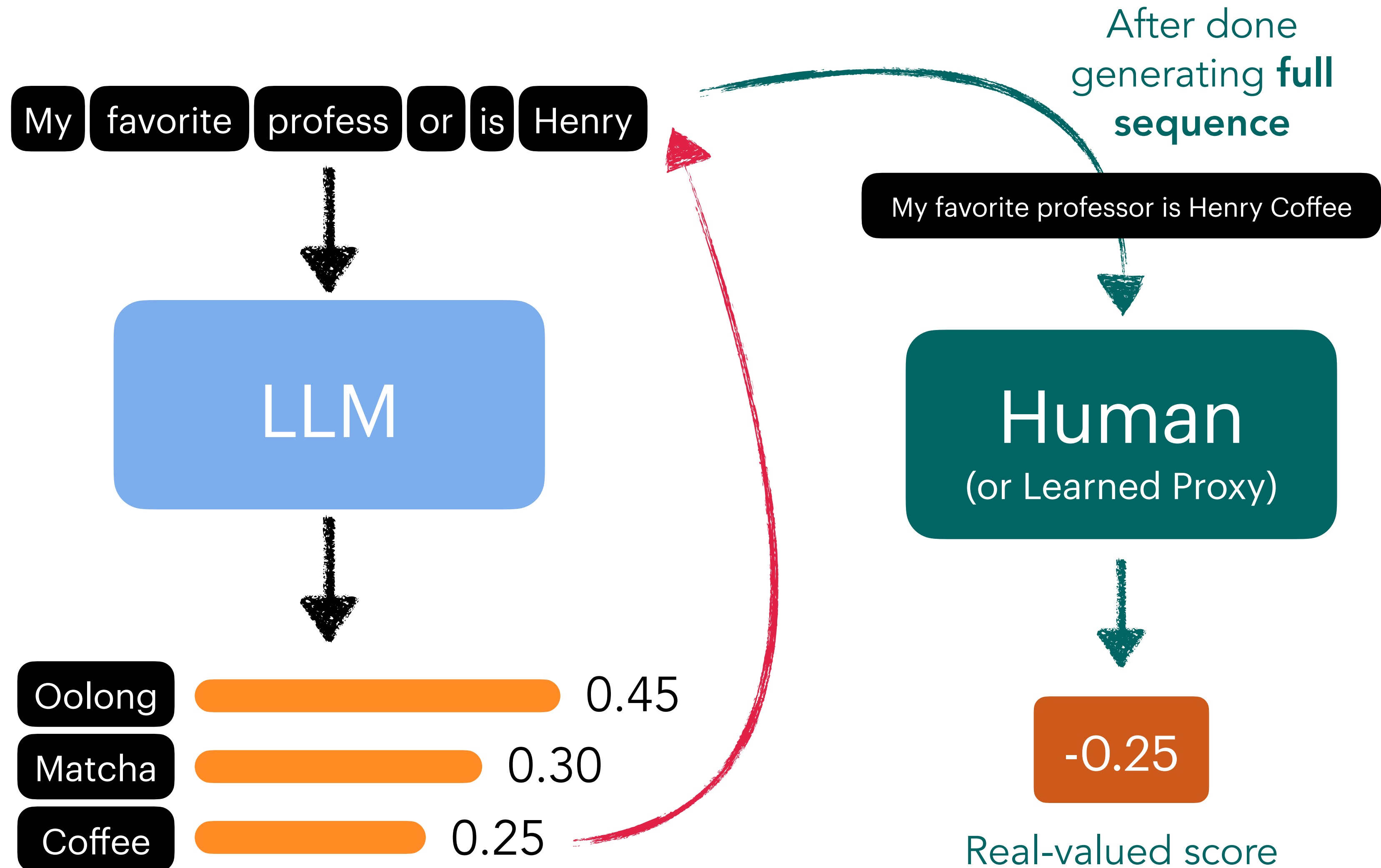
# Text generation as RL

State  
Action  
Transition  
**Reward**

Policy  
Value

Different trajectories will have different rewards!

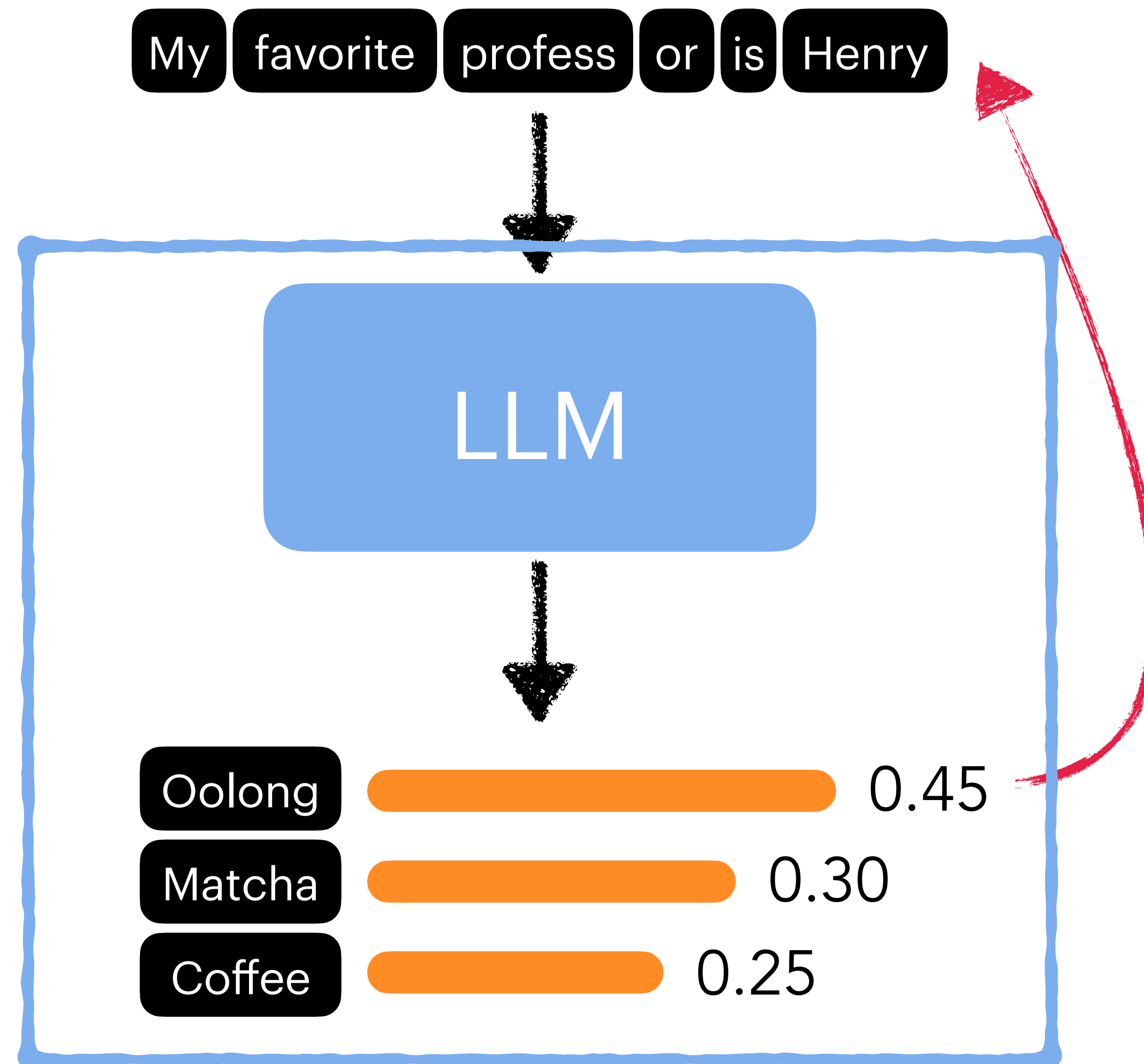
(and they're lukewarm about coffee)



# Text generation as RL

State  
Action  
Transition  
Reward

Policy  
Value



Unlike past lectures, we're going to be considering a **stochastic** policy, which yields a **distribution over next actions**.

# Text generation as RL

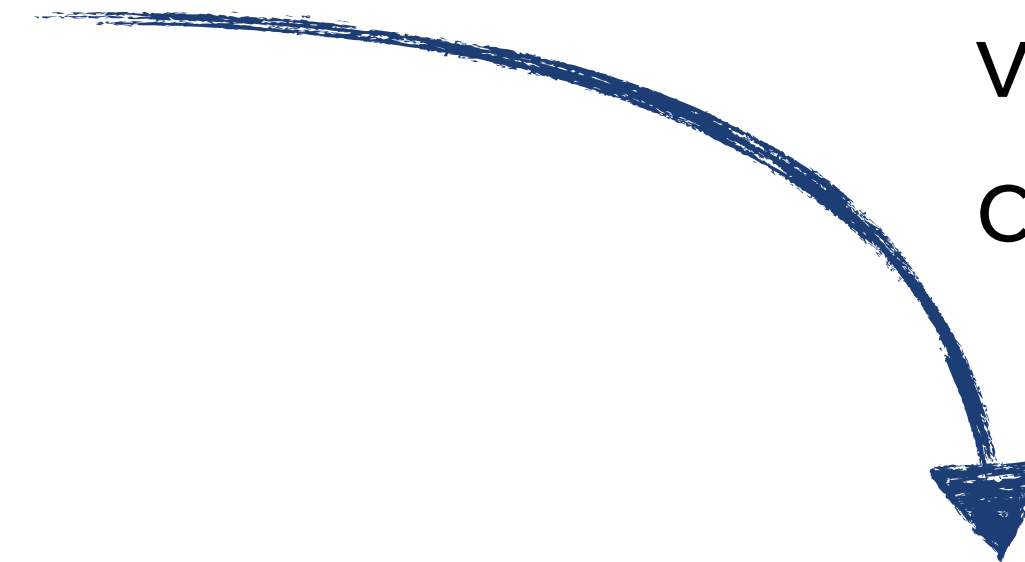
State  
Action  
Transition  
Reward

Policy  
**Value**

My favorite profess or is Henry



We can estimate  
value for the  
current state



0.7

# Roadmap

Why RL for LLMs?

Some RL background

## **Policy gradients**

Actor-critic

Proximal Policy Optimization & RLHF

Group-Relative Policy Optimization

# Policy Gradients

Previous approaches we've discussed focused on estimating certain quantities (i.e. values, q-values) and deriving a policy from them.

What if we directly learn a policy to maximize the reward?

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} [R(\tau)]$$

# Notation

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} [R(\tau)]$$

Let's assume we have a **stochastic** policy  $\pi$  parameterized by  $\theta$ . Our aim is to learn parameters  $\theta^*$  to optimize the reward of trajectories sampled from our policy (and the environment)

Trajectory (or episode)

$$\tau = \langle (s_1, a_1), \dots, (s_T, a_T) \rangle$$

Reward of entire trajectory

$$R(\tau) = \sum_{i=1}^T R(s_i, a_i)$$

Probability of trajectory

$$P_{\theta}(\tau) = \prod_{t=1}^T \underbrace{P(s_{t+1} \mid s_t, a_t)}_{\text{Environment transitions}} \underbrace{\pi_{\theta}(a_t \mid s_t)}_{\text{Policy}}$$



# Why don't we just differentiate the reward?

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} [R(\tau)]$$

The natural first step here is just to take the gradient of the expectation w.r.t.  $\theta$ .

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} [R(\tau)] = \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} \left[ \frac{\partial}{\partial \theta} R(\tau) \right]$$



Usually, the next step is to bring the gradient operator inside the expectation.

But now, the quantity we're differentiating by is also part of the expectation — is that still allowed?

no!

# Let's open up the expectation

Recall the definition of expectation:

$$\mathbb{E}_{x \sim P(\cdot)}[f(x)] = \sum_x f(x)P(x)$$

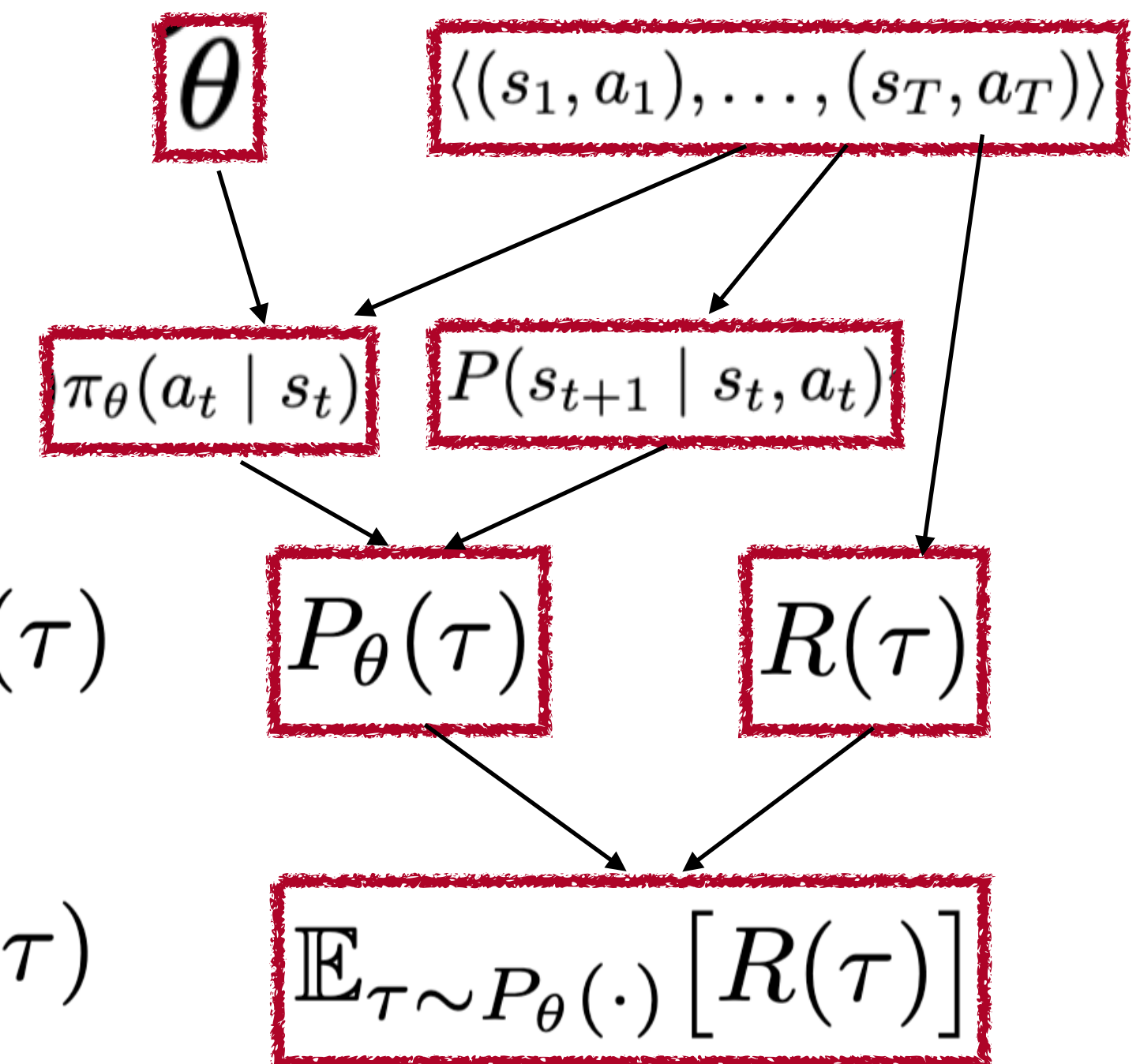
Applying it to our gradient expression, we get

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathbb{E}_{\tau \sim P_{\theta}(\cdot)}[R(\tau)] &= \frac{\partial}{\partial \theta} \sum_{\tau} P_{\theta}(\tau) \cdot R(\tau) \\ &= \sum_{\tau} R(\tau) \cdot \frac{\partial}{\partial \theta} P_{\theta}(\tau) \end{aligned}$$

$$R(\tau) = \sum_{i=1}^T R(s_i, a_i)$$

Has no explicit dependency on  $\theta$ ,  
so we can treat it as a constant.

The states and actions are derived from the policy/env via **sampling**, which is not a differentiable operation. Thus, we treat them as **constants**.



# From sum to expectation

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} [R(\tau)] = \sum_{\tau} R(\tau) \cdot \frac{\partial}{\partial \theta} P_{\theta}(\tau)$$

This is better, but still intractable due to the infinite sum over trajectories. What if we turn it into an expectation over trajectories?

$$= \sum_{\tau} R(\tau) \cdot \frac{\partial}{\partial \theta} P_{\theta}(\tau) \cdot \overbrace{\frac{1}{P_{\theta}(\tau)} \cdot P_{\theta}(\tau)}^1$$

$$= \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} \left[ R(\tau) \cdot \frac{\partial}{\partial \theta} P_{\theta}(\tau) \cdot \frac{1}{P_{\theta}(\tau)} \right]$$

Why is this a better way to express the policy gradient?

# Sidenote: Monte Carlo estimation

We can estimate expected value of a random variable by drawing a bunch of samples and taking their mean — this is a **Monte Carlo** estimate.

Monte Carlo methods trade off between speed and accuracy; taking more samples gets the estimate closer to the true value, but can be very costly (esp. for LLMs!)

$$\mathbb{E}_{\tau \sim P_{\theta}(\cdot)} \left[ R(\tau) \cdot \frac{\partial}{\partial \theta} P_{\theta}(\tau) \cdot \frac{1}{P_{\theta}(\tau)} \right]$$

To estimate the policy gradient, we can draw  $N$  trajectories from  $P_{\theta}$ , compute the quantity in brackets for each of them, and take the average.

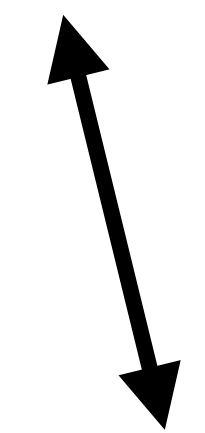
This is very similar to what we did with (mini-batch) stochastic gradient descent to estimate the average gradient over the *entire* dataset.



# The log derivative trick

$$\mathbb{E}_{\tau \sim P_{\theta}(\cdot)} \left[ R(\tau) \cdot \frac{\partial}{\partial \theta} P_{\theta}(\tau) \cdot \frac{1}{P_{\theta}(\tau)} \right]$$

Let's do something clever here! Observe that:

$$\frac{\partial}{\partial \theta} \log P_{\theta}(\tau) = \frac{\partial}{\partial \theta} P_{\theta}(\tau) \cdot \frac{1}{P_{\theta}(\tau)}$$


Which lands us here:

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] = \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} \left[ R(\tau) \cdot \frac{\partial}{\partial \theta} \log P_{\theta}(\tau) \right]$$

# Policy Gradients

Let's plug in the expression for the probability of trajectory, giving us the following final-ish expression.

Note that we're sampling  $N$  trajectories for the Monte Carlo estimate.

$$\begin{aligned} P_{\theta}(\tau) &= \prod_{t=1}^T P(s_{t+1} \mid s_t, a_t) \pi_{\theta}(a_t \mid s_t) \\ \log P_{\theta}(\tau) &= \sum_{t=1}^T \log P(s_{t+1} \mid s_t, a_t) + \log \pi_{\theta}(a_t \mid s_t) \\ \frac{\partial}{\partial \theta} \log P_{\theta}(\tau) &= \sum_{t=1}^T \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t \mid s_t) \end{aligned}$$

Contains no  $\theta$  terms

$$\begin{aligned} \frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] &= \mathbb{E}_{\tau \sim P_{\theta}(\cdot)} \left[ R(\tau) \cdot \sum_{t=1}^T \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t \mid s_t) \right] \\ &\approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T R(\tau^{(i)}) \cdot \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t^{(i)} \mid s_t^{(i)}) \end{aligned}$$

# Credit Assignment

Let's inspect our final expression:

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \underbrace{R(\tau^{(i)})}_{\text{Credit for } a_t} \cdot \underbrace{\frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t^{(i)} \mid s_t^{(i)})}_{\text{Direction to increase probability of } a_t}$$

Suppose the agent commits a big mistake and gets a huge negative reward at  $T = 1$ , but is perfect for the rest of the trajectory. However, the overall trajectory has a large negative  $R(\tau)$ . Should we punish later actions for rewards obtained at earlier steps?

**No! Let's only scale by rewards *caused* by the action.**

# Better Credit Assignment

Instead of scaling each gradient by the total reward, let's scale only by *future* rewards:

$$G_t^{(i)} = \sum_{j=t}^T R(s_j^{(i)}, a_j^{(i)})$$

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T G_t^{(i)} \cdot \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t^{(i)} \mid s_t^{(i)})$$

This gives us the REINFORCE policy gradient.



# REINFORCE

1. Sample trajectories  $\tau_1, \dots, \tau_N$  from the policy and environment
2. Compute policy gradients

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T G_t^{(i)} \cdot \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t^{(i)} \mid s_t^{(i)})$$

3. Apply gradient updates

$$\theta \leftarrow \theta + \alpha \cdot \frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)]$$

# Roadmap

Why RL for LLMs?

Some RL background

Policy gradients

**Actor-critic**

Proximal Policy Optimization & RLHF

Group-Relative Policy Optimization

# Limitations of REINFORCE

Naive REINFORCE usually doesn't work too well — why?

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T G_t^{(i)} \cdot \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t^{(i)} \mid s_t^{(i)})$$

## Variance!

The rewards may vary wildly depending on the states in the trajectory, meaning that for stability, we often need a very large  $N$ . —> This results in unstable gradient updates!

How can we reduce variance in our gradient estimator?

# Accounting for Variance

Recall our interpretation of the policy gradient:

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \underbrace{G_t^{(i)}}_{\text{Credit for } a_t} \cdot \underbrace{\frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t^{(i)} \mid s_t^{(i)})}_{\text{Direction to increase probability of } a_t}$$
$$G_t^{(i)} = \sum_{j=t}^T R(s_j^{(i)}, a_j^{(i)})$$

Observe that we don't need the credit term to be exactly the rewards — it just has to be anything that gives some signal for which actions are better/worse *in a given state*.

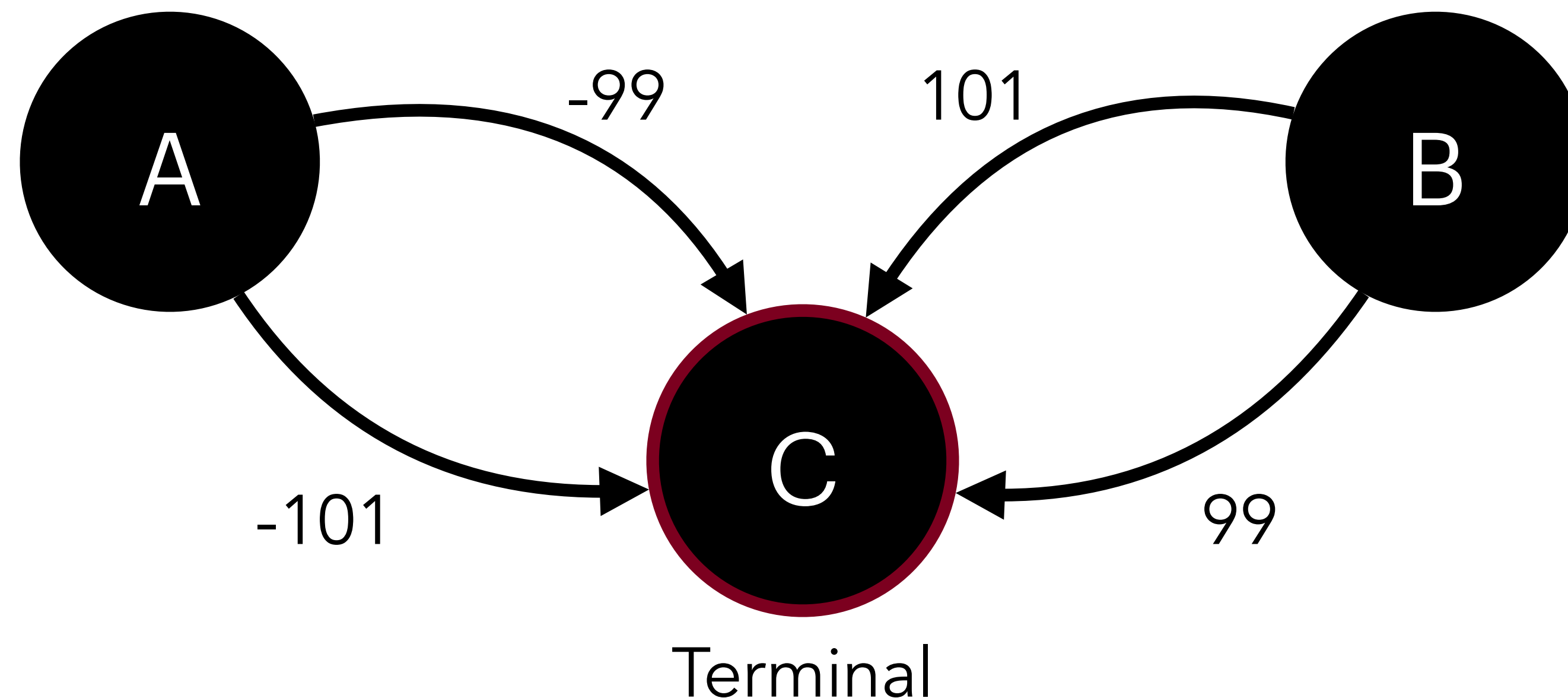
**Solution:** For each state, subtract a specific *baseline* value from all the rewards to remove the variance across states.

# Why state-dependent baselines?

We don't need the credit term to be exactly the rewards — it just has to be anything that gives some signal for which actions are better/worse *in a given state*.

**Solution:** For each state, subtract a specific *baseline* value to remove the variance in rewards across states.

Observe that the **bad action** in **B** will have a greater gradient than a **good action** in **A**.

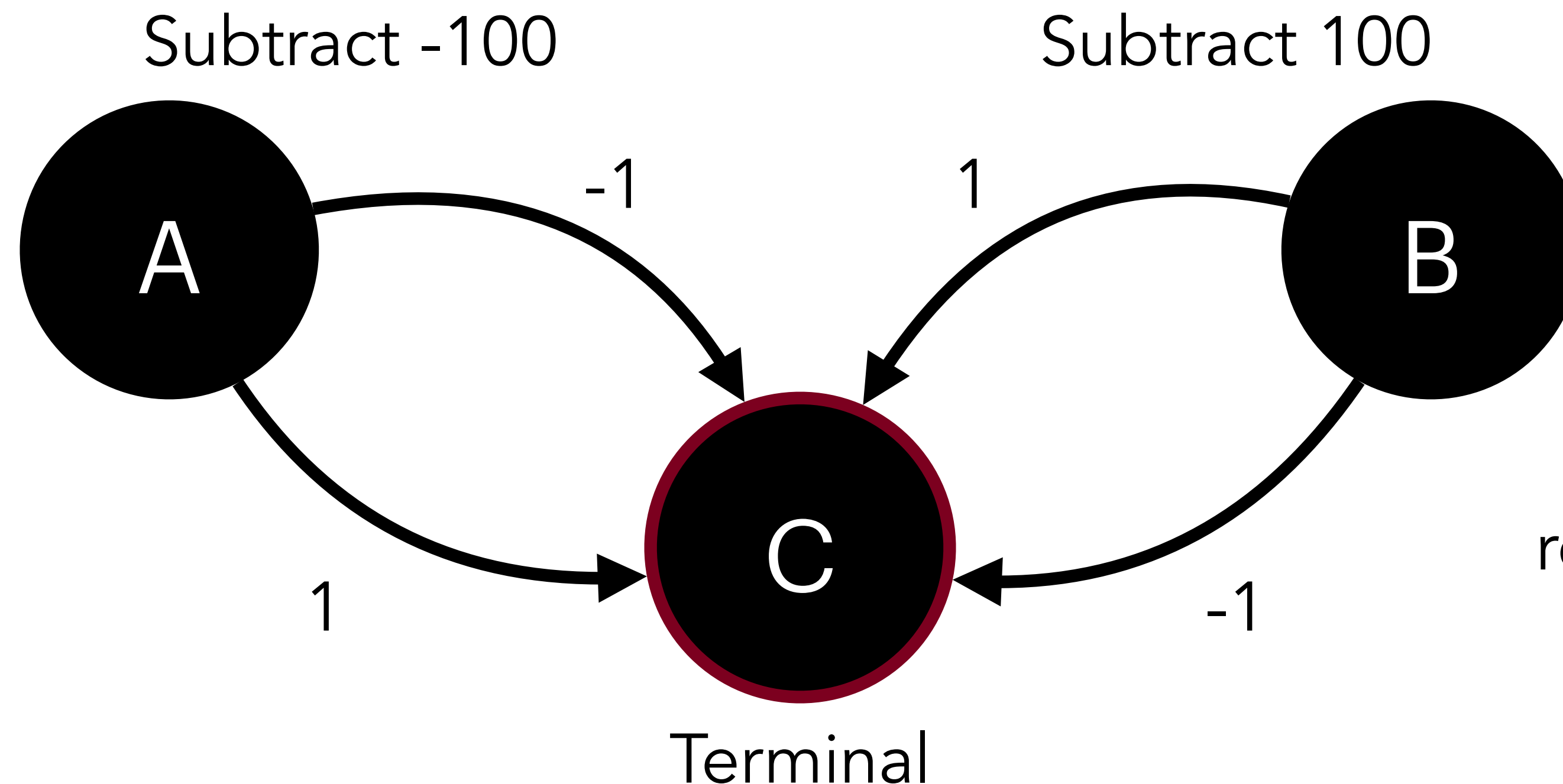


# Why state-dependent baselines?

We don't need the credit term to be exactly the rewards — it just has to be anything that gives some signal for which actions are better/worse *in a given state*.

**Solution:** For each state, subtract a specific *baseline* value to remove the variance in rewards across states.

But if we subtract away a baseline value for each state, we will have the desired gradient behavior.



We'll call these remaining quantities the **advantages**.

# Learning baselines

To normalize the rewards, we'll subtract the mean observed rewards  $G_t$  for each state.

If the number of states is small we can use a table (like tabular q-learning), but if there are lots of states (which there are for LLMs) then we can learn a **critic model**.

$$\mathcal{L}(\tau_1, \dots, \tau_N; \phi) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \left( V_{\phi}(s_t) - \sum_{j=t}^T R(s_j^{(i)}, a_j^{(i)}) \right)^2$$

Then, in the gradient, instead of using the raw reward  $G_t$ , we'll use the **advantage**  $A_t$

$$G_t^{(i)} = \sum_{j=t}^T R(s_j^{(i)}, a_j^{(i)}) \longrightarrow A_t^{(i)} = G_t^{(i)} - V(s_t^{(i)})$$



# Q-learning returns

Recall the definition of values and Q-values (assuming deterministic transitions)

$$\begin{aligned} V(s_t) &= \mathbb{E}_{a_t \sim \pi(\cdot | s_t)} [Q(s_t, a_t)] \\ Q(s_t, a_t) &= R(s_t, a_t) + \gamma V(s_{t+1}) \\ &= R(s_t, a_t) + \mathbb{E}_{\pi} [\gamma R(s_{t+1}, a_{t+1}) + \gamma^2 V(s_{t+2})] \\ &= R(s_t, a_t) + \mathbb{E}_{\pi} [\gamma R(s_{t+1}, a_{t+1}) + \gamma^2 R(s_{t+2}, a_{t+2}) + \gamma^3 V(s_{t+3})] \\ &= \mathbb{E}_{\pi} \left[ \sum_{j=t}^T \gamma^{j-t} R(s_j, a_j) \right] \end{aligned}$$

Note that this is no longer a max since our policy is stochastic

So our  $G_t$  quantities are q-value estimates, our critic is estimating values, and our advantages are gaps between q-value and value!

$$G_t^{(i)} = \sum_{j=t}^T R(s_j^{(i)}, a_j^{(i)})$$



# Actor-Critic

1. Sample trajectories  $\tau_1, \dots, \tau_N$  from the policy and environment
2. Update critic

$$\mathcal{L}(\tau_1, \dots, \tau_N; \phi) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \left( V_{\phi}(s_t) - \sum_{j=t}^T R(s_j^{(i)}, a_j^{(i)}) \right)^2$$

3. Compute policy gradients using advantages.

$$\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T A_t^{(i)} \cdot \frac{\partial}{\partial \theta} \log \pi_{\theta}(a_t^{(i)} \mid s_t^{(i)}) \quad A_t^{(i)} = G_t^{(i)} - V(s_t^{(i)})$$

4. Update actor

$$\theta \leftarrow \theta + \alpha \cdot \frac{\partial}{\partial \theta} \mathbb{E}_{\tau \in P_{\theta}(\cdot)} [R(\tau)]$$

# Roadmap

Why RL for LLMs?

Some RL background

- Policy gradients

- Actor-critic

## **Proximal Policy Optimization & RLHF**

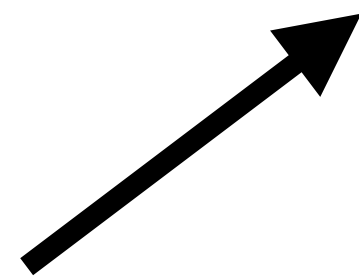
Group-Relative Policy Optimization

# Proximal Policy Optimization

Modifying the policy too quickly can cause instability in training.

Ideally, we'd like to limit the amount that the **policy** distribution changes in each update, i.e.

$$\theta_{k+1}^* = \arg \max_{\text{KL}(\pi_{\theta_k} \parallel \pi_{\theta_{k+1}}) < \epsilon} \mathbb{E}_{\tau \sim P_{\theta_{k+1}}}(\cdot) [R(\tau)]$$



KL divergence is a measure of the “distance” between two distributions; here, we want the distance between the old and updated policies to be under some threshold.

# Proximal Policy Optimization

Modifying the policy too quickly can cause instability in training.

Ideally, we'd like to limit the amount that the **policy** distribution changes in each update, i.e.

$$\theta_{k+1}^* = \arg \max_{\text{KL}(\pi_{\theta_k} \parallel \pi_{\theta_{k+1}}) < \epsilon} \mathbb{E}_{\tau \sim P_{\theta_{k+1}}(\cdot)} [R(\tau)]$$

However, limiting update size in this manner is difficult and expensive to implement. Instead, PPO optimizes the following proxy objective:

$$\mathbb{E}_{\tau \sim \pi_{\theta}(\cdot)} \left[ \sum_{t=1}^T \min \left( \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{ref}}}(a_t | s_t)} A_t, \text{clip} \left( \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{ref}}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right]$$

Note that this is the clipped version of the objective for language models.

# Making sense of PPO

As in actor-critic, we're optimizing the advantage-adjusted reward  $A_t$ . However, we're also scaling the advantage by some odd-looking terms.

$$\mathbb{E}_{\tau \sim \pi_{\theta}(\cdot)} \left[ \sum_{t=1}^T \min \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)} A_t, \text{clip} \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right]$$

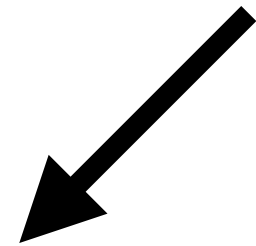
Likelihood ratio  $r$  between  
our **current** policy vs. policy  
at the **start of training**

Limits the quantity to be  
between  $1-\epsilon$  and  $1+\epsilon$

hyperparameter

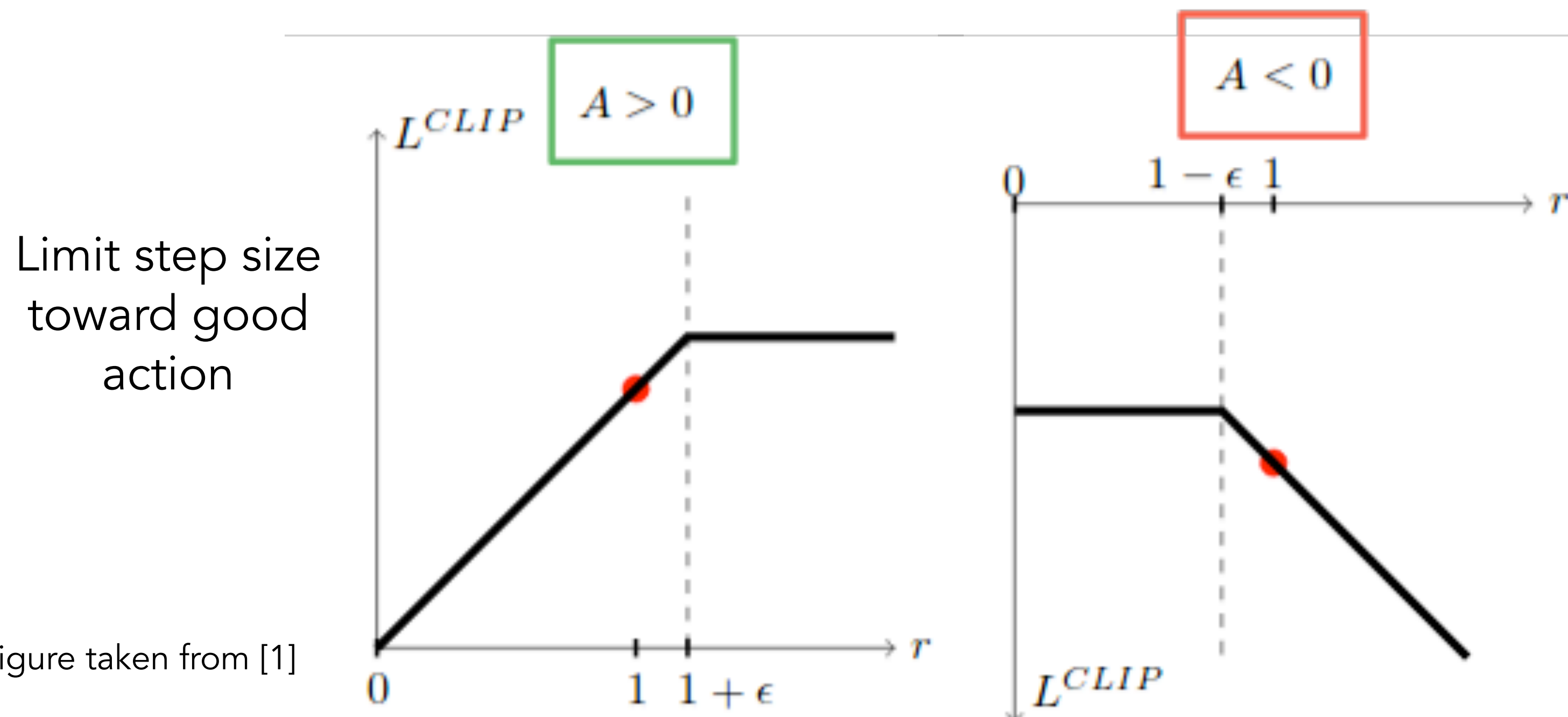
# Making sense of PPO

$$\mathbb{E}_{\tau \sim \pi_{\theta}(\cdot)} \left[ \sum_{t=1}^T \min \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)} A_t, \text{clip} \left( \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\text{ref}}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right]$$

Likelihood ratio  $r$  

If the action was **good**....

If the action was **bad**....



Limit step size  
toward good  
action

When bad action is more  
probable under current  
policy, make step large so  
we can **revert back to  
original policy**.

**TLDR:** new policy is not  
allowed to move far away  
from original policy

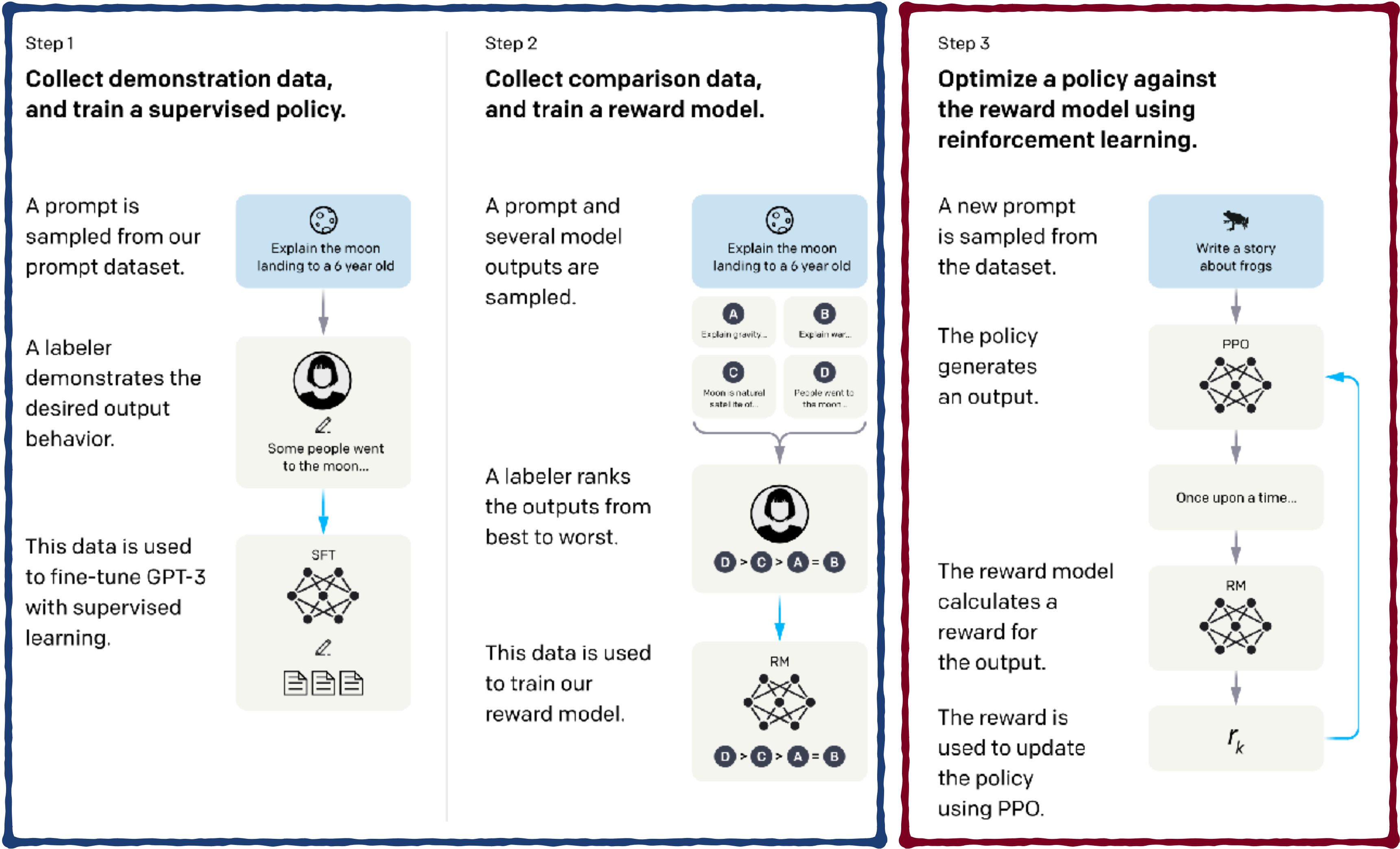
Figure taken from [1]



# RL from Human Feedback

SFT + learning a reward model based on human preferences

Figure from [2]



PPO!

This is what ChatGPT did (circa 2022)

<sup>2</sup> Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang et al. "Training language models to follow instructions with human feedback." *Advances in neural information processing systems* 35 (2022): 27730-27744.

# Roadmap

Why RL for LLMs?

Some RL background

- Policy gradients

- Actor-critic

Proximal Policy Optimization & RLHF

**Group-Relative Policy Optimization**



# PPO is expensive!

To do PPO, we need to keep in memory:

- Policy weights
- Reference model weights (same size as policy, frozen)
- Value/critic model weights (same size as policy, learned)
- Reward model weights (same size as policy, frozen)

The weights that are not frozen need another ~2x additional memory for an optimizer like Adam!

For an **8B** parameter LLM (16GB with fp16):

$$\begin{aligned} & 3 * 16\text{GB policy} + 16\text{GB reference} + 3 * 16\text{GB critic} + 16\text{GB reward} \\ & = 8 * 16\text{GB} = 128\text{GB} \end{aligned}$$

(pessimistic estimate)

# Why do we need the value model anyway?

Recall that we use the value model to compute advantages, which reduces variance.

$$A_t^{(i)} = G_t^{(i)} - V(s_t^{(i)})$$

Variance in rewards is a major issue for a lot of RL tasks, but it turns out that for LLMs, rewards are usually very well-behaved.

Ten students come to lecture. Half leave out of boredom, then another falls asleep. How many are still paying attention to lecture?

Trajectory		Reward
$(10 - 1) / 2 = 9 / 2 = 4.5$		0
$10 / 2 - 1 = 5 - 1 = 4$		1
$10 / 2 - 1 = 10 / 1 = 10$		0

# GRPO: seizing the means of PPO<sub>R</sub>duction

For each prompt, sample  $G$  outputs  $o_1, o_2, \dots, o_G$  and get their sequence-level rewards  $r_1, r_2, \dots, r_G$

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]$$

Taken from [3]

$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \underbrace{\min \left[ \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left( \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1 - \varepsilon, 1 + \varepsilon \right) \hat{A}_{i,t} \right]}_{\text{PPO objective}} - \underbrace{\beta \mathbb{D}_{KL} [\pi_{\theta} || \pi_{ref}]}_{\text{KL penalty (regularization)}} \right\}$$

where advantages are obtained by simply normalizing the rewards

$$\hat{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$$

<sup>3</sup> Shao, Zhihong, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models." arXiv preprint arXiv:2402.03300 (2024).

# GRPO: seizing the means of PPO<sub>R</sub>duction

For each prompt, sample  $G$  outputs  $o_1, o_2, \dots, o_G$  and get their sequence-level rewards  $r_1, r_2, \dots, r_G$

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]$$

Taken from [3]

$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \underbrace{\min \left[ \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left( \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1 - \varepsilon, 1 + \varepsilon \right) \hat{A}_{i,t} \right]}_{\text{PPO objective}} - \underbrace{\beta \mathbb{D}_{KL} [\pi_{\theta} || \pi_{ref}]}_{\text{KL penalty (regularization)}} \right\}$$

**New memory footprint for 8B LLM:**

$$3 * 16\text{GB policy} + 16\text{GB reference} + \text{~~3 * 16GB critic~~} + 16\text{GB reward} \\ = 5 * 16\text{GB} = 80\text{GB}$$

<sup>3</sup> Shao, Zhihong, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models." arXiv preprint arXiv:2402.03300 (2024).

# Roadmap

Why RL for LLMs?

Some RL background

- Policy gradients

- Actor-critic

Proximal Policy Optimization & RLHF

Group-Relative Policy Optimization

**Direct Policy Optimization**

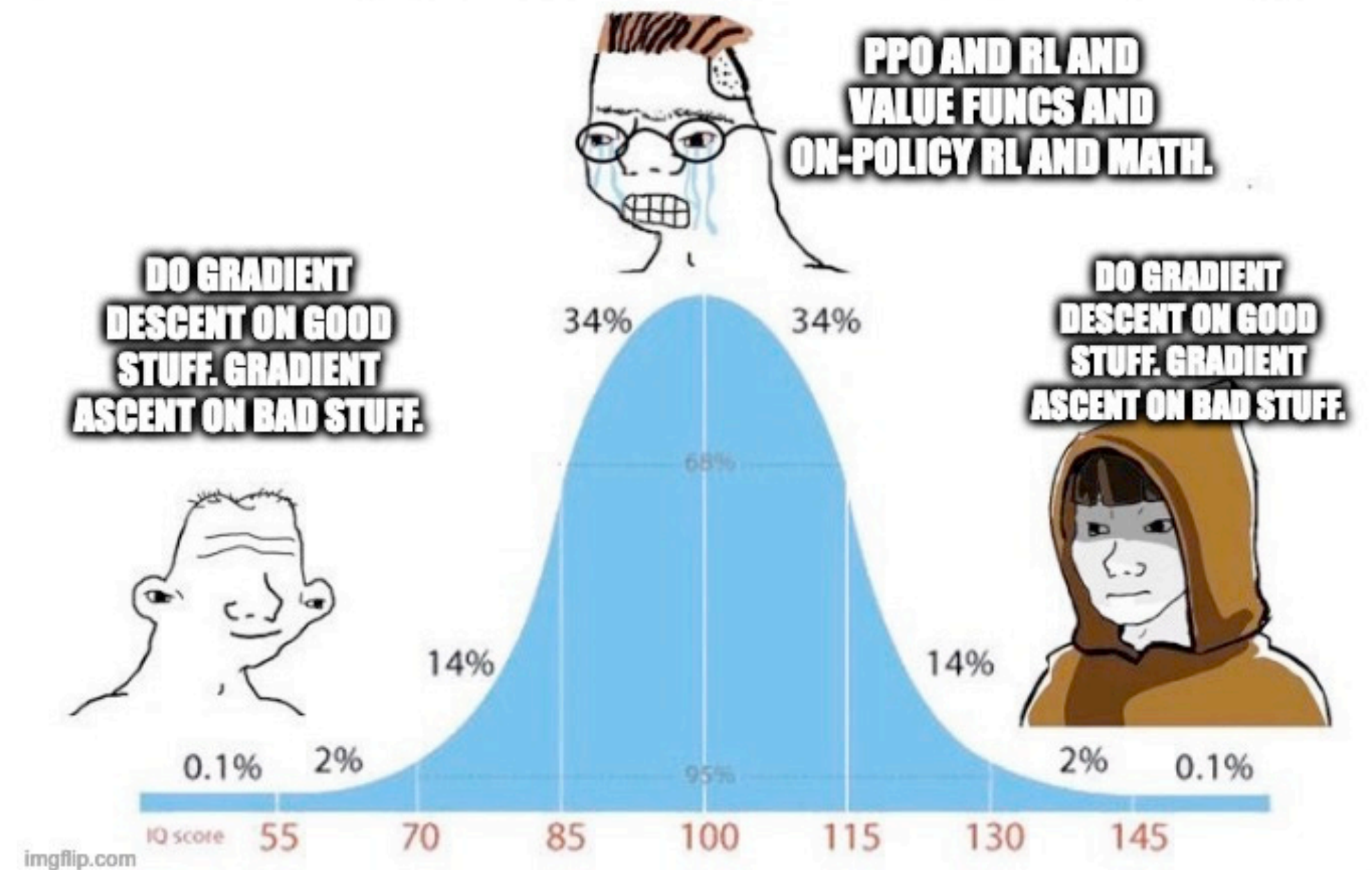


# Do we need RL anyway?

RL lets us learn from both good and bad outputs, but is RL the only way to do this?

What if we just do SFT, but with a loss that increases the likelihood of "good" things and decreases the likelihood of "bad" things?

## LEARNING FROM HUMAN FEEDBACK





# Direct Preference Optimization

Suppose we have a dataset of tuples  $(x, y_w, y_l)$  where  $x$  is a prompt,  $y_w$  is a preferred response, and  $y_l$  is a dispreferred response.

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[ \log \sigma \left( \underbrace{\beta \log \frac{\pi_\theta(y_w | x)}{\pi_{\text{ref}}(y_w | x)}}_{\text{Likelihood ratio for good response}} - \underbrace{\beta \log \frac{\pi_\theta(y_l | x)}{\pi_{\text{ref}}(y_l | x)}}_{\text{Likelihood ratio for bad response}} \right) \right]$$

**Intuition:** use the difference between the LLM's probabilities of the good & bad responses as the reward (DPO doesn't exactly do this, but another algo, SimPO, does).

Note that this increases the gap between good and bad responses, but doesn't guarantee that the probability of good responses will go up!

# Is DPO still RL?

$$\mathcal{L}_{\text{DPO}}(\pi_{\theta}; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[ \log \sigma \left( \beta \log \frac{\pi_{\theta}(y_w | x)}{\pi_{\text{ref}}(y_w | x)} - \beta \log \frac{\pi_{\theta}(y_l | x)}{\pi_{\text{ref}}(y_l | x)} \right) \right]$$

We can view DPO as an *offline* RL algorithm since we often sample the preferred/dispreferred responses from the original LLM (you can also use response pairs from another model, but it usually doesn't work as well) — this is similar to doing Q-learning with a replay buffer consisting only of trajectories from the initial policy.

DPO significantly cuts downs on memory/computation costs, but loses the exploration aspect of online RL algorithms.