# Post-training (cont'd)

## CS 6804: Frontier AI Systems

*Spring 2026*

https://tuvllms.github.io/ai-seminar-spring-2026/

**Tu Vu**

VIRGINIA TECH

# Logistics

- Homework assignments
  - Homework 0 & 1, due 3/3 & 3/10
    - 5% extra credits each

# Reinforcement learning from human feedback (RLHF)

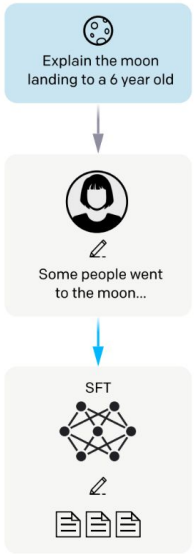
Step 1

**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.



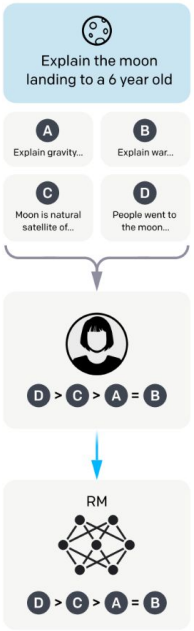

Step 2

**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

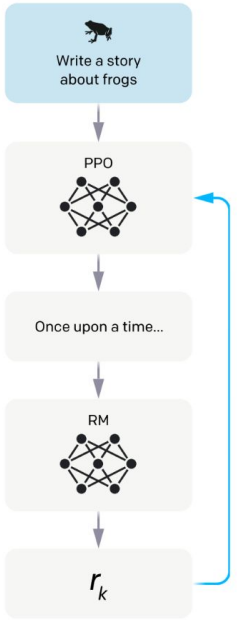
**Optimize a policy against the reward model using reinforcement learning.**

A new prompt is sampled from the dataset.

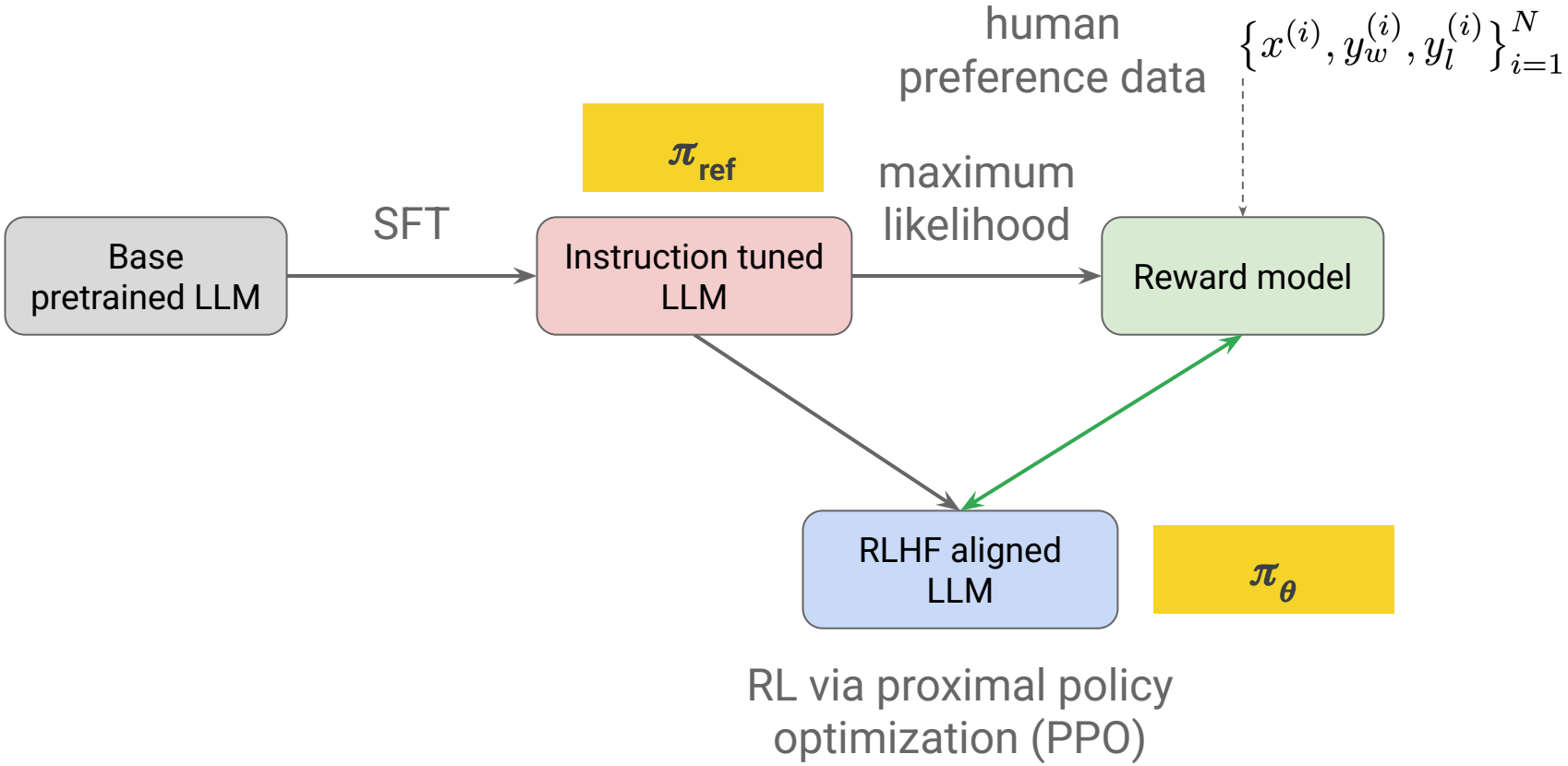
The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



# RLHF pipeline: putting it all together



# Step 3: RL fine-tuning

**The second term prevents the model from deviating too far from the distribution on which the reward model is accurate.**

**y = $\pi_\theta$(x)**

$$\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(y|x)} \left[r_\phi(x,y)\right] - \beta D_{KL}\left[\pi_\theta(y|x) \,||\, \pi_{\text{ref}}(y|x)\right]$$

where $\beta$ is a parameter controlling the deviation from the base reference policy $\pi_{\text{ref}}$, namely the initial SFT model $\pi^{SFT}$. In practice, the language model policy $\pi_\theta$ is also initialized to $\pi^{SFT}$.

# Reinforcement Learning

# Policy Gradient Algorithms

**Goal:** we want a language model (the policy) that produces completions that get high reward.

- Policy $\pi_\theta(y|x)$: a distribution over completions given a prompt.
- Reward $R(x, y)$: a scalar that scores the full completion.
- We optimize expected reward:

$$J(\theta) = \mathbb{E}_{y \sim \pi_\theta(\cdot|x)}[R(x, y)].$$

**Key framing for LMs:** the completion is often treated as one action (bandit view), even though we write

$$J(\theta) = \sum_y \pi_\theta(y|x)\, R(x, y).$$

# Policy Gradient Algorithms

**Goal:** we want a language model (the policy) that produces completions that get high reward.

- Policy $\pi_\theta(y|x)$: a distribution over completions given a prompt.
- Reward $R(x, y)$: a scalar that scores the full completion.
- We optimize expected reward:

$$J(\theta) = \mathbb{E}_{y \sim \pi_\theta(\cdot|x)}[R(x, y)].$$

$$J(\theta) = \sum_y \pi_\theta(y|x)\, R(x, y).$$

# Policy Gradient

$$J(\theta) = \sum_y \pi_\theta(y|x)\, R(x,y).$$

$$d(\log x) = \frac{1}{x}\, dx$$

$$\nabla_\theta J(\theta) = \sum_y \nabla_\theta \pi_\theta(y|x)\, R(x,y).$$

$$\nabla_\theta \pi_\theta(y|x) = \pi_\theta(y|x)\, \nabla_\theta \log \pi_\theta(y|x).$$

$$dx = x\, d(\log x)$$

Log-derivative trick

$$\nabla_\theta J(\theta) = \sum_y \pi_\theta(y|x)\, \nabla_\theta \log \pi_\theta(y|x)\, R(x,y).$$

$$\boxed{\nabla_\theta J(\theta) = \mathbb{E}_{y\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(y|x)\, R(x,y)\right]}$$

# Policy Gradient (cont'd)

$$\boxed{\nabla_\theta J(\theta) = \mathbb{E}_{y \sim \pi_\theta} \left[ \nabla_\theta \log \pi_\theta(y|x)\, R(x, y) \right]}$$

A sequence probability factorizes:

$$\log \pi_\theta(y|x) = \sum_t \log \pi_\theta(y_t|x, y_{<t}).$$

**Each sampled action contributes a gradient, scaled by how good the outcome was**

$$\nabla_\theta J(\theta) = \mathbb{E}\left[ \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\, R \right].$$

# Policy Gradient (cont'd)

We usually replace raw reward with something lower-variance:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\, \Psi_t\right]$$

Where $\Psi_t$ can be:

- total return $G_t$
- return-to-go
- advantage $A_t$
- TD residual
- GAE estimate

# Advantage form (most important)

In modern algorithms:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\, A_t\right]$$

$A_t$ **(or** $\Psi_t$**)**

**Learning signal.**

- Positive → action better than expected → reinforce.
- Negative → worse → suppress.
- Zero → no update.

**Policy gradient updates the model by increasing the log-probability of actions that produced better-than-expected outcomes and decreasing those that produced worse outcomes.**

# Advantage form (most important)

In modern algorithms:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\, A_t\right]$$

$A_t$ **(or** $\Psi_t$**)**

**Learning signal.**

- Positive → action better than expected → reinforce.
- Negative → worse → suppress.
- Zero → no update.

**Policy gradient updates the model by increasing the log-probability of actions that produced better-than-expected outcomes and decreasing those that produced worse outcomes.**

# High-level framework

1. **Sample**: pick prompts $x$, sample completions $y \sim \pi_\theta(\cdot|x)$.
2. **Score**: compute reward $R(x, y)$ (optionally add KL penalty).
3. **Assign credit**: produce a per-token learning signal (this is where algorithms differ).
4. **Update**: increase probability of good tokens/sequences, decrease probability of bad ones.

**Core identity (intuition):**

We can push probability mass toward actions that led to high reward by using

$$\nabla_\theta \log \pi_\theta(\text{chosen action})$$

as the "direction" and reward/advantage as the "scale."

**Policy gradients update the model by changing the log-probabilities of what it sampled, where the change is scaled by how good the outcome was.**

# High-level framework (cont'd)

- **Return** $G_t$: "how good the outcome was from time $t$ onward."
  - In single-final-reward RLHF, $G_t = R$ for all tokens.
- **Baseline** $b_t$: something that you subtract to reduce variance.
  - Baseline does not change the expected gradient, but it reduces noise.
- **Advantage** $A_t$: "better-than-expected"

$$A_t = G_t - b_t.$$

**Key takeaway:** in single-final-reward RLHF, most differences come from *how you choose* $b_t$ and *how you constrain updates*.

**Think of $G_t$ as the score from your teacher for each essay you write**

**Think of $b_t$ as your own internal expectation ("I usually get an 80 on essays like this")**

# Return

**General discounted return**

$$G_t \;=\; R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \;=\; \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

If the episode terminates at time $T$, this becomes

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T.$$

**When $\gamma = 1$**

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T.$$

**In RLHF with a single terminal reward**

$$R_1 = 0,\ \ldots,\ R_{T-1} = 0, \qquad R_T = R.$$

So for every timestep $t$,

$$G_t = R.$$

# KL penalty per token (reward shaping)

RLHF usually adds a per-token KL penalty to keep the policy close to a reference model.

So per-token reward becomes:

$$r_t = -\beta\,\mathrm{KL}_t, \quad r_T = R(x,y) - \sum_t \beta\,\mathrm{KL}_t.$$

# KL penalty per token (reward shaping)

$$R_{\text{shaped}} = R_{\text{model}} - \beta \sum_{t} \text{KL}_t$$

So the KL is computed **per token**, then summed over the sequence.

# Policy Gradient Algorithms (cont'd)

- REINFORCE
- RLOO (REINFORCE Leave One Out)
- PPO (Proximal Policy Optimization)

| Method | $G_t$ | $b_t$ | Advantage shape |
|---|---|---|---|
| REINFORCE | $R$ | batch mean | same for all tokens |
| RLOO | $R$ | group mean | same for all tokens |
| PPO | $R$ | $V(s_t)$ | different per token |

- **REINFORCE** → $A_t = G_t - b$
- **RLOO** → $A_t = R - \text{leave-one-out baseline}$
- **PPO** → $A_t$ from value function / GAE + clipped ratio

# REINFORCE

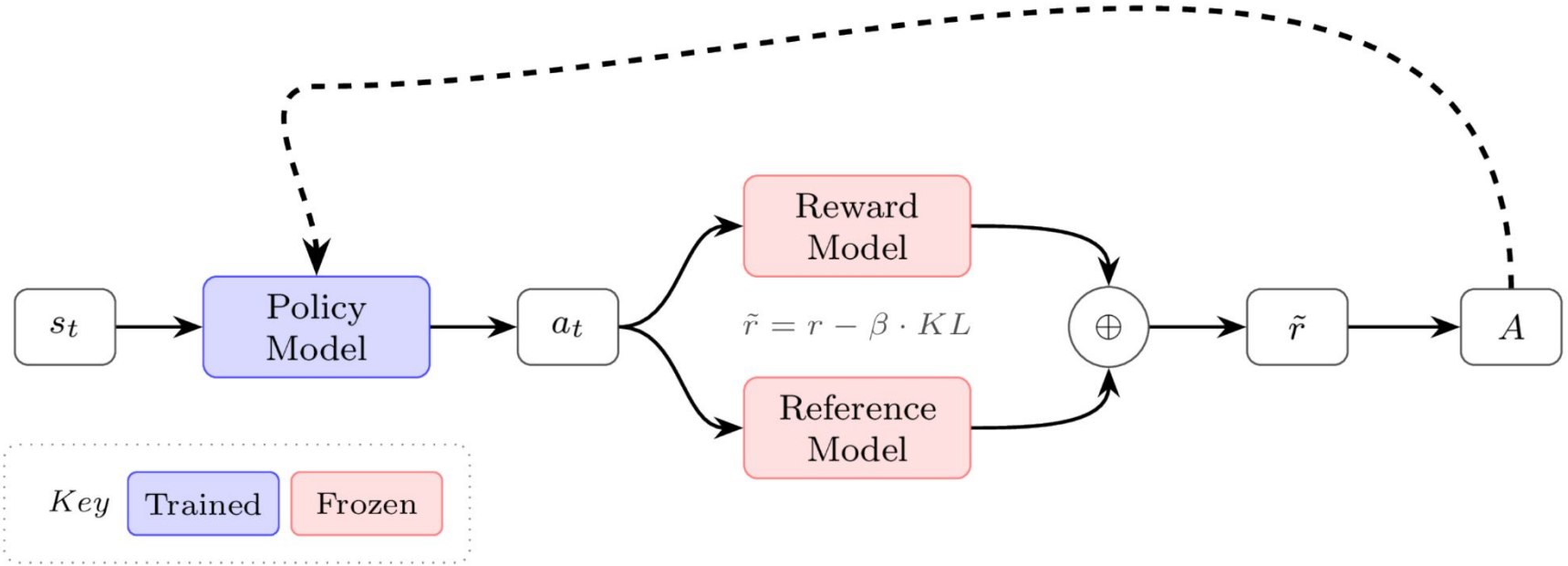



Figure 17: Basic REINFORCE architecture for language models. The shaped reward combines the reward model score with a KL penalty from the reference model. We build on this structure throughout the chapter.

# REINFORCE (cont'd)

After computing the shaped reward (a single scalar), REINFORCE uses:

$$A_t = R_{\text{shaped}} - b.$$

So:

- Even though KL was per-token,
- It is **aggregated into one scalar reward**,
- Then that scalar is broadcast to all tokens.

# REINFORCE (cont'd)

$$b_t = \text{mean reward over batch.}$$

So if your batch has rewards $[R_1, R_2, R_3]$,

$$b_t = \frac{R_1 + R_2 + R_3}{3}.$$

This baseline is the same for all tokens in that sequence.

Then:

$$G_t - b_t = R - \text{batch mean.}$$

So every token in that completion gets the same advantage.

# REINFORCE (cont'd)

## Intuition

- Sample a completion.
- If reward is high, reinforce every token in that completion.
- If reward is low, suppress every token in that completion.

## Update shape (minimal form)

$$\sum_t \nabla_\theta \log \pi_\theta(y_t|x, y_{<t}) \, (R - b)$$

# REINFORCE (cont’d)

It is a **Monte Carlo** gradient estimator: it uses sampled outcomes.

With a single final reward, credit is **broadcast** to all tokens.

Main weakness: **high variance**, because one scalar reward must explain many tokens.

# RLOO

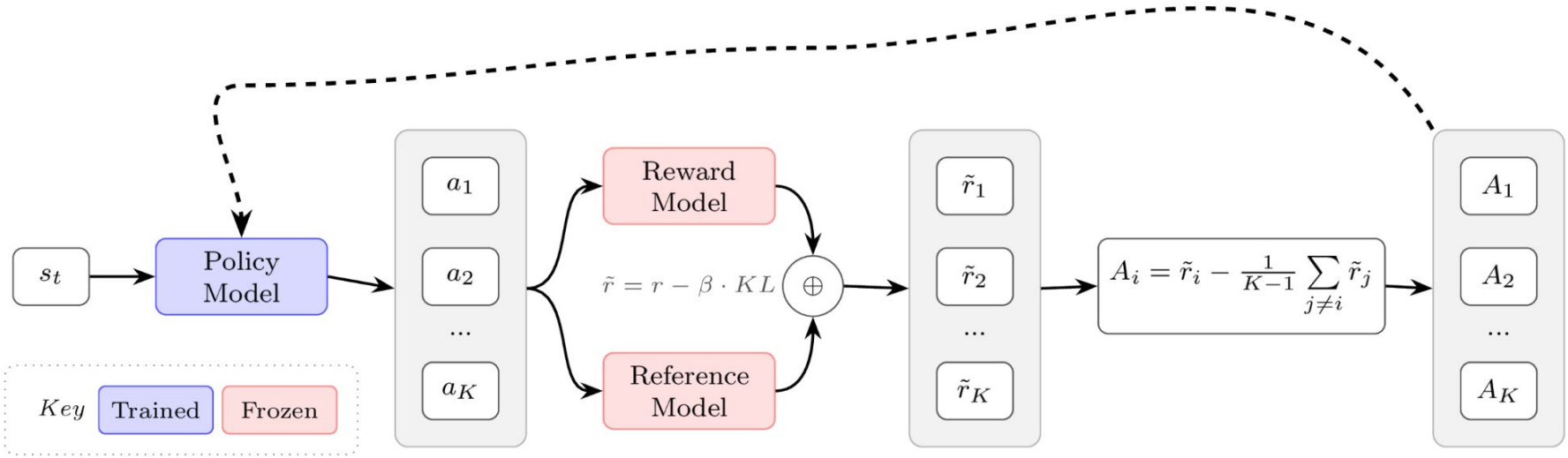



Figure 18: REINFORCE Leave-One-Out (RLOO) architecture. Multiple completions per prompt provide a leave-one-out baseline for advantage estimation without learning a value function.

# RLOO (cont'd)

If you generate multiple completions per prompt:

$$b = \text{mean reward of other completions.}$$

Still one scalar per completion.

Still identical across tokens.

# RLOO (cont'd)

## Motivation

REINFORCE is noisy because the baseline is crude (often a batch average). RLOO improves the baseline by making it **prompt-conditional**.

## Setup

- For one prompt $x$, sample $K$ completions $y^{(1)}, \ldots, y^{(K)}$.
- Score each completion: $R^{(k)} = R(x, y^{(k)})$.

## Baseline

For completion $k$, the baseline is the mean reward of the *other* completions:

$$b^{(k)} = \frac{1}{K-1} \sum_{i \neq k} R^{(i)}.$$

## Advantage

$$A^{(k)} = R^{(k)} - b^{(k)}.$$

# RLOO (cont'd)

RLOO computes:

$$A^{(k)} = R^{(k)}_{\text{shaped}} - \text{mean of other rewards.}$$

Again:

- Shaped reward includes summed KL.
- But the advantage is one scalar per completion.
- That scalar is broadcast to all tokens.

So still no token-level differentiation.

- RLOO keeps the simplicity of REINFORCE.
- It reduces variance because it compares completions **within the same prompt**, which removes prompt difficulty effects.
- It still assigns a **sequence-level** advantage that is broadcast to tokens.

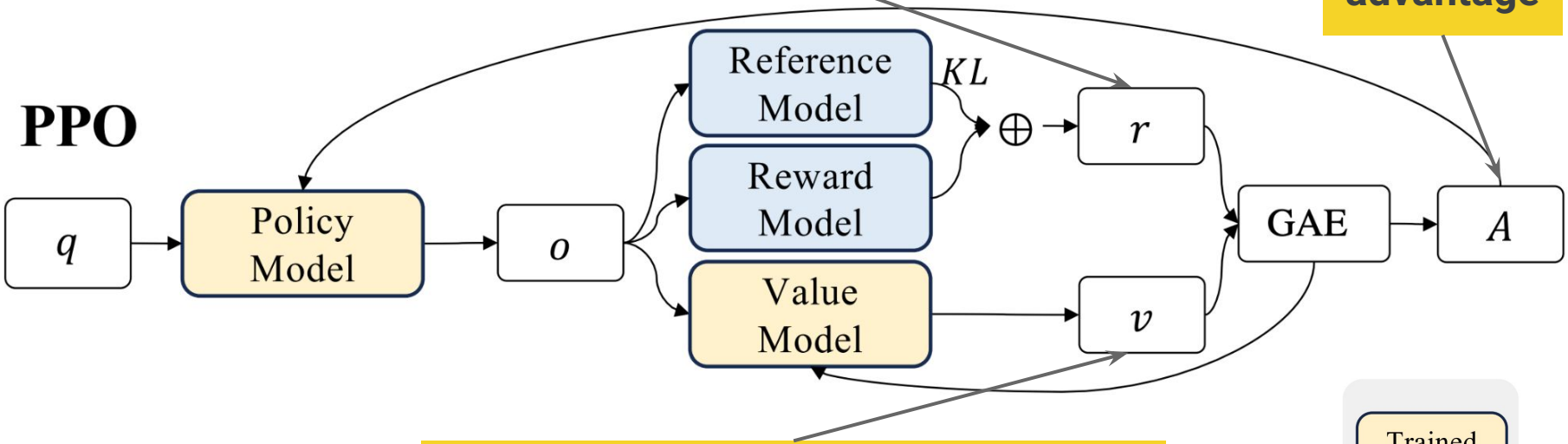
## Quick intuition line

RLOO turns “is this good?” into “is this better than the other tries for the same question?”

# PPO

**Think of the reward model as a teacher who grades each essay you write**

**advantage**




**The critic is like your own internal expectation ("I usually get an 80 on essays like this")**

# PPO (stable updates + token-level credit)

## A. PPO idea #1: do not move the policy too far (the ratio + clipping)

Define the ratio:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)} = \exp\left(\log \pi_\theta - \log \pi_{\text{old}}\right).$$

PPO optimizes a clipped objective that prevents $\rho_t$ from leaving $[1-\epsilon,\ 1+\epsilon]$ in a way that would create overly large updates.

- $\pi_{\text{old}}$ is the policy that generated the batch.
- You can do multiple gradient steps on the same batch because importance ratios correct for drift.
- Clipping creates a trust region that stabilizes training.

## B. PPO idea #2: a learned value function gives token-level baselines

Value model predicts:

$$V(s_t) \approx \mathbb{E}[G_t \mid s_t].$$

Then advantage can be computed with TD or GAE (you can pick one to focus on):

- **TD residual:**

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

- **GAE:**

$$A_t^{GAE} = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots$$

- In single-final-reward RLHF, the final reward enters at the end, and the value function propagates it backward through advantages.
- PPO has two losses: **policy loss** (clipped) and **value loss** (fit $V$).

# PPO with Temporal-Difference (TD) error

## Final advantage

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

This differs across tokens.

---

## Meaning

> Was this specific action better or worse than expected at this timestep?

- Positive → reinforce this token
- Negative → suppress this token

# PPO with Generalized Advantage Estimation (GAE)

### Final advantage

$$A_t = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

---

### Meaning

> A smoothed measure of how much better this action was, accounting for both immediate and future prediction errors.

- λ controls how far the reward propagates backward.
- Provides smoother and lower-variance credit assignment.

# Core PPO policy objective (clipped)

Define the ratio:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)}.$$

The PPO clipped objective is:

$$\boxed{L^{\text{policy}}(\theta) = \mathbb{E}\left[\min\left(\rho_t(\theta)A_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\,A_t\right)\right]}$$

This is what we maximize.

# Thank you!