ECE7115 ~~Multimodal VLM~~ LLM

13. RLHF

Spring 2026

Namhyuk Ahn, Inha University

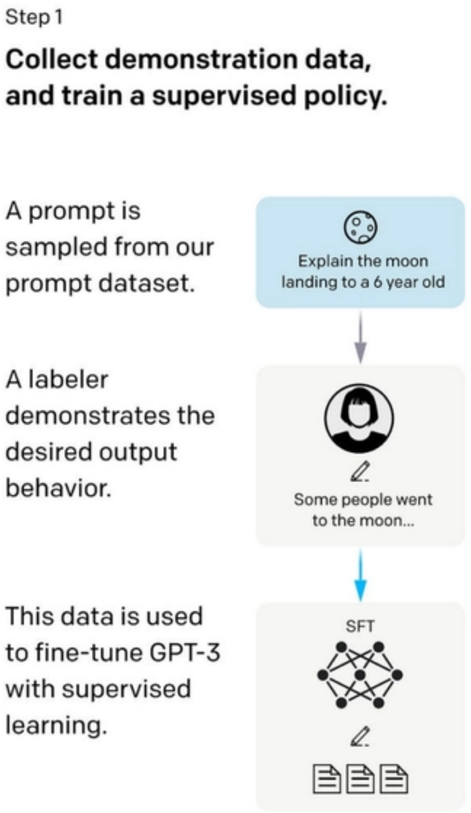
Last Week: Dataset

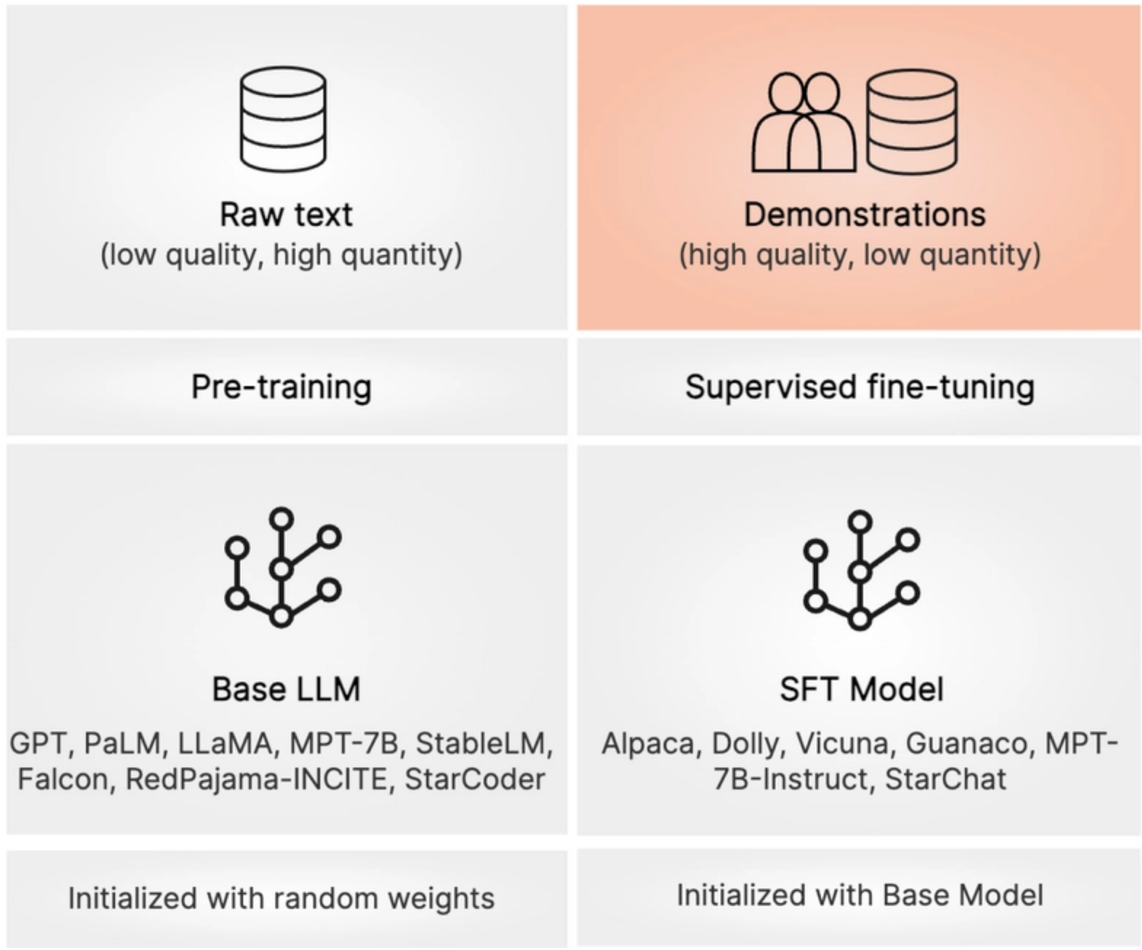
- Key lesson: Data does not fall from the sky. You have to work to get it
- and data is the key ingredient that differentiates language models

- Standard tools
- Filtering: KenLM / fastText / DSIR
- Deduplication: Bloom Filter / MinHash / LSH
- Much of this pipeline is heuristic, many opportunities to improve

- Recently many models use synthetic dataset
- Beware of legal and ethical issues

Last Week: SFT





Prompt:
Should I add chorizo to my paella?

Feedback (completion):
Absolutely! Chorizo is a popular ingredient in many paella recipes

Image credit: https://klu.ai/glossary/supervised-fine-tuning

Lecture Overview

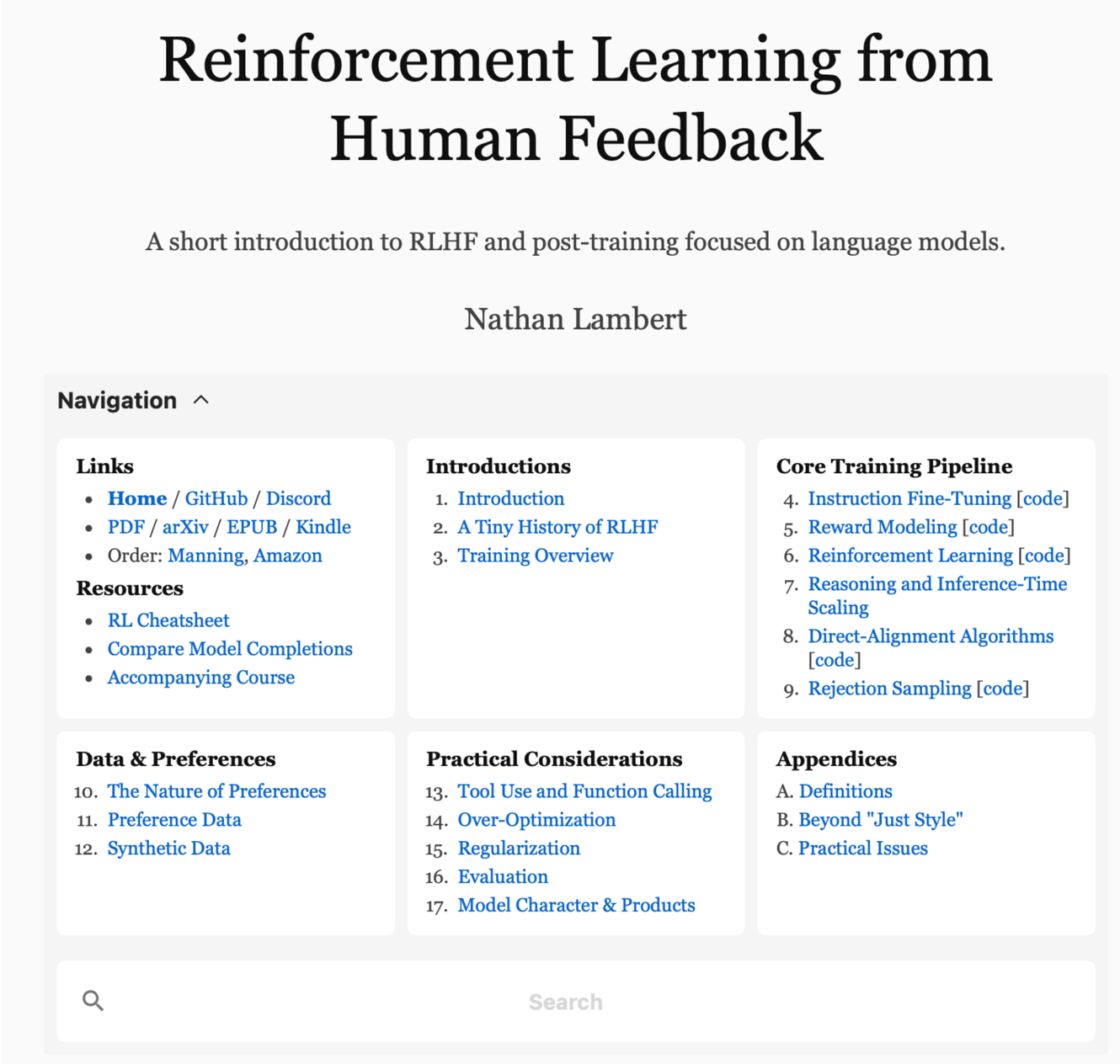
- Why RL When Training LLM?

- From SFT to PPO
  - REINFORCE
  - A2C
  - PPO

- RL from Human Feedback
  - RLHF
  - DPO

RLHFBook

- https://rlhfbook.com/
- [Lambert+ 2026]



Reinforcement Learning from Human Feedback

A short introduction to RLHF and post-training focused on language models.

Nathan Lambert

Navigation

Links
- **Home** / GitHub / Discord
- PDF / arXiv / EPUB / Kindle
- Order: Manning, Amazon

Resources
- RL Cheatsheet
- Compare Model Completions
- Accompanying Course

Introductions
1. Introduction
2. A Tiny History of RLHF
3. Training Overview

Core Training Pipeline
4. Instruction Fine-Tuning [code]
5. Reward Modeling [code]
6. Reinforcement Learning [code]
7. Reasoning and Inference-Time Scaling
8. Direct-Alignment Algorithms [code]
9. Rejection Sampling [code]

Data & Preferences
10. The Nature of Preferences
11. Preference Data
12. Synthetic Data

Practical Considerations
13. Tool Use and Function Calling
14. Over-Optimization
15. Regularization
16. Evaluation
17. Model Character & Products

Appendices
A. Definitions
B. Beyond "Just Style"
C. Practical Issues

Search

Why RL?

SFT Imitates a Target Answer

- SFT data is usually a prompt-response pair $(x, y^\star)$
  - where x is an user prompt and $y^\star$ is a target response
- SFT minimizes $L_{\text{SFT}}(\theta) = -\sum_{t=1}^{T} \log \pi_\theta \left(y_t^\star \mid x, y_{<t}^\star\right).$
- It increases the probability of the tokens in the demonstrated answer
  - "Imitation learning" in RL

Problem: No Single Correct Answer

- Consider the prompts:
  - "Explain policy gradient to a student learning RL for the first time."
  - Many answers can be good
    - short intuitive answer, or equation-heavy answer

- SFT increases the likelihood of the target response
  - SFT doesn't see the preference among many possible answers
  - Hence, this usually does not directly see the rejected alternatives

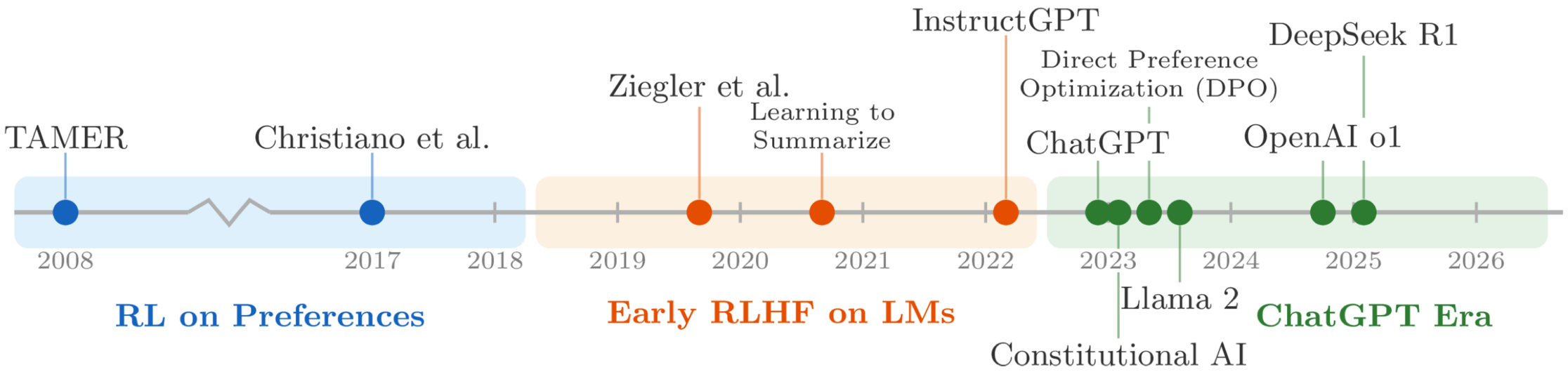
TL;DR: Use Reinforcement Learning

- Instead of using labeled text, we use human preference dataset
  - Reinforcement learning from human feedback (RLHF)
  - We want to make LLM to maximize human preference
  - How to train the model? use RL

- In RL, we try to maximize the expected reward function $\max_{\theta} \mathbb{E}[R]$

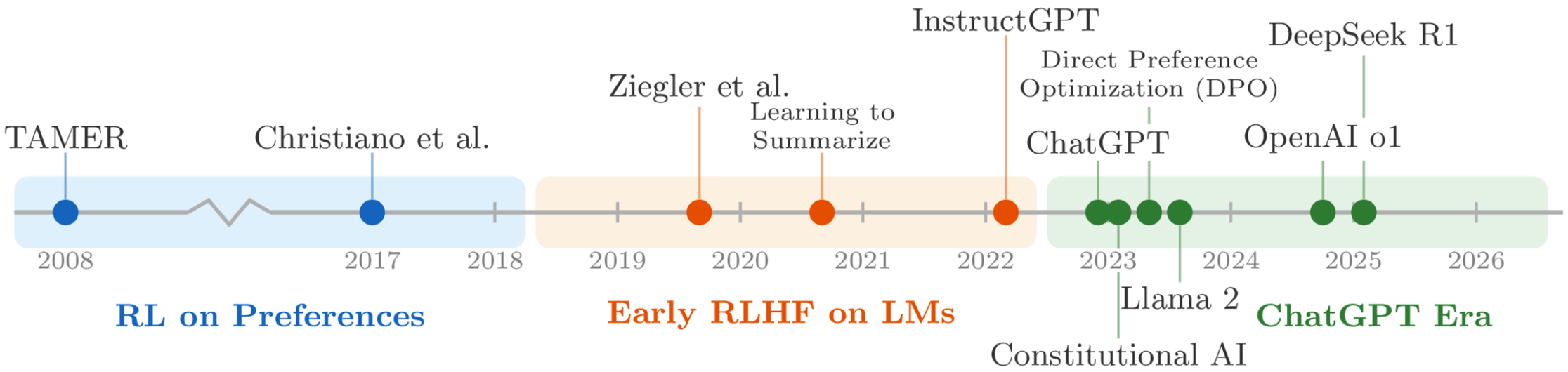
- Reward is the objective function and we can learn the reward
  - By training the reward model using the human preference dataset
  - So, reward model predicts the human preference
  - RL-trained LLM maximizes reward from the reward model

The Path to Modern RLHF



- [Ziegler+ 2019]: first RLHF on LMs
- InstructGPT [Ouyang+ 2022]: the canonical RLHF recipe behind ChatGPT
- Constitutional AI [Bai+ 2022]: Early methods for AI feedback in Claude

The Path to Modern RLHF



- DPO [Rafailov+ 2023]: RLHF without a reward model
- LLaMA3 [Grattafiori+ 2024] and Tulu 3 [Lambert+ 2024]: modern multi-stage recipes
- DeeSeek R1 [Guo+ 2025]: popularized RLVR

What RLHF Data Looks Like

- A human (or AI) annotator sees two responses to the same prompt and picks the better one
- This preference pair becomes training data for the reward model

USER
Explain why the sky is blue in one sentence.

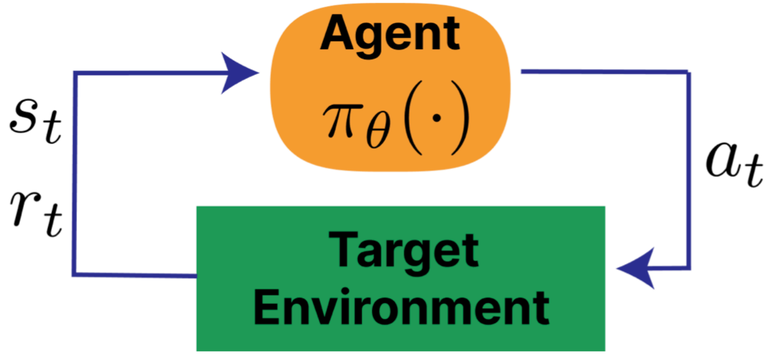
ASSISTANT (Response A ✓)
The sky is blue due to Rayleigh scattering, where shorter blue wavelengths of sunlight are scattered more by atmospheric molecules than longer wavelengths.

USER
Explain why the sky is blue in one sentence.

ASSISTANT (Response B)
The sky appears blue because of the way light interacts with the atmosphere and stuff, it's basically just physics.

Classical Reinforcement Learning (RL)

- A reinforcement learning problem is often written as a Markov decision process (MDP)
  - State space $\mathcal{S}$, action space $\mathcal{A}$
  - Transition dynamics $P(s_{t+1} \mid s_t, a_t)$
  - Reward function $r(s_t, a_t)$ and discount γ
  - Policy $\pi_\theta(a_t \mid s_t)$

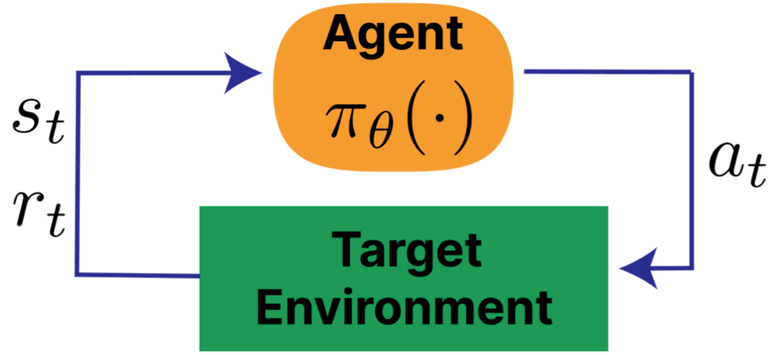


  - We optimize cumulative return over a trajectory

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right]$$

Classical Reinforcement Learning (RL)

- RL is trial-and-error learning, balancing exploration and exploitation across long-term rewards



- State: the current situation the agent is in
- Action: what the agent does next
- Reward: the signal for how good that action was
- Policy: the strategy for choosing actions
- Trajectory: a sequence of states, actions, and rewards
  - $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$
- Trajectory probability: $P(\tau \mid \pi) = p(s_0) \prod_{t=0}^{T} \pi(a_t \mid s_t)\, p(s_{t+1} \mid s_t, a_t)$

A Simple RL Example: Thermostat

- The agent learns over many episodes when to turn the heater on or off
  - State: the current room temperature
  - Action: turn the heater on or off
  - Reward: positive when the room stays near the target temperature
  - Policy: the rule for deciding what to do next
  - Example policy: $\pi(a_t = \text{on} \mid s_t) = \begin{cases} 1 & \text{if } s_t < 70^\circ\text{F} \\ 0 & \text{otherwise} \end{cases}$

A Standard RL Example: CartPole

- State: cart position, velocity, pole angle, angular velocity

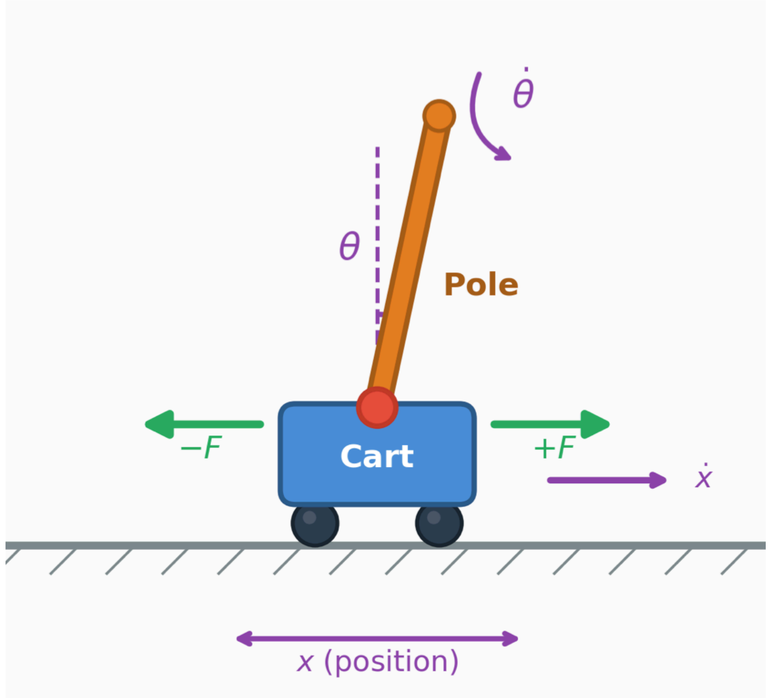
$$s_t = (x_t,\ \dot{x}_t,\ \theta_t,\ \dot{\theta}_t)$$

- Action: push the cart left or right

$$a_t \in \{\text{left},\ \text{right}\}$$

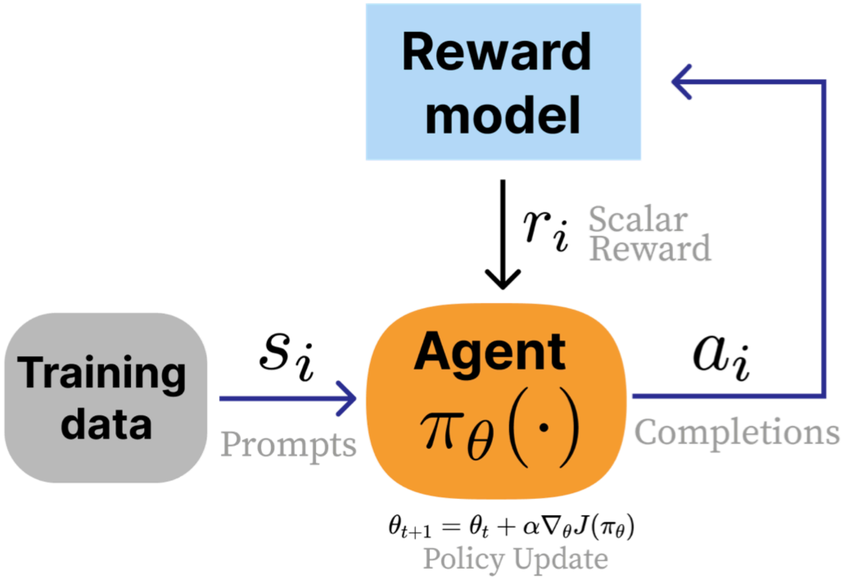
- Reward: +1 for every step the pole stays upright

$$r(s_t, a_t) = \begin{cases} 1 & \text{if } |\theta_t| < 12^\circ \text{ and } |x_t| < 2 \\ 0 & \text{otherwise (episode ends)} \end{cases}$$

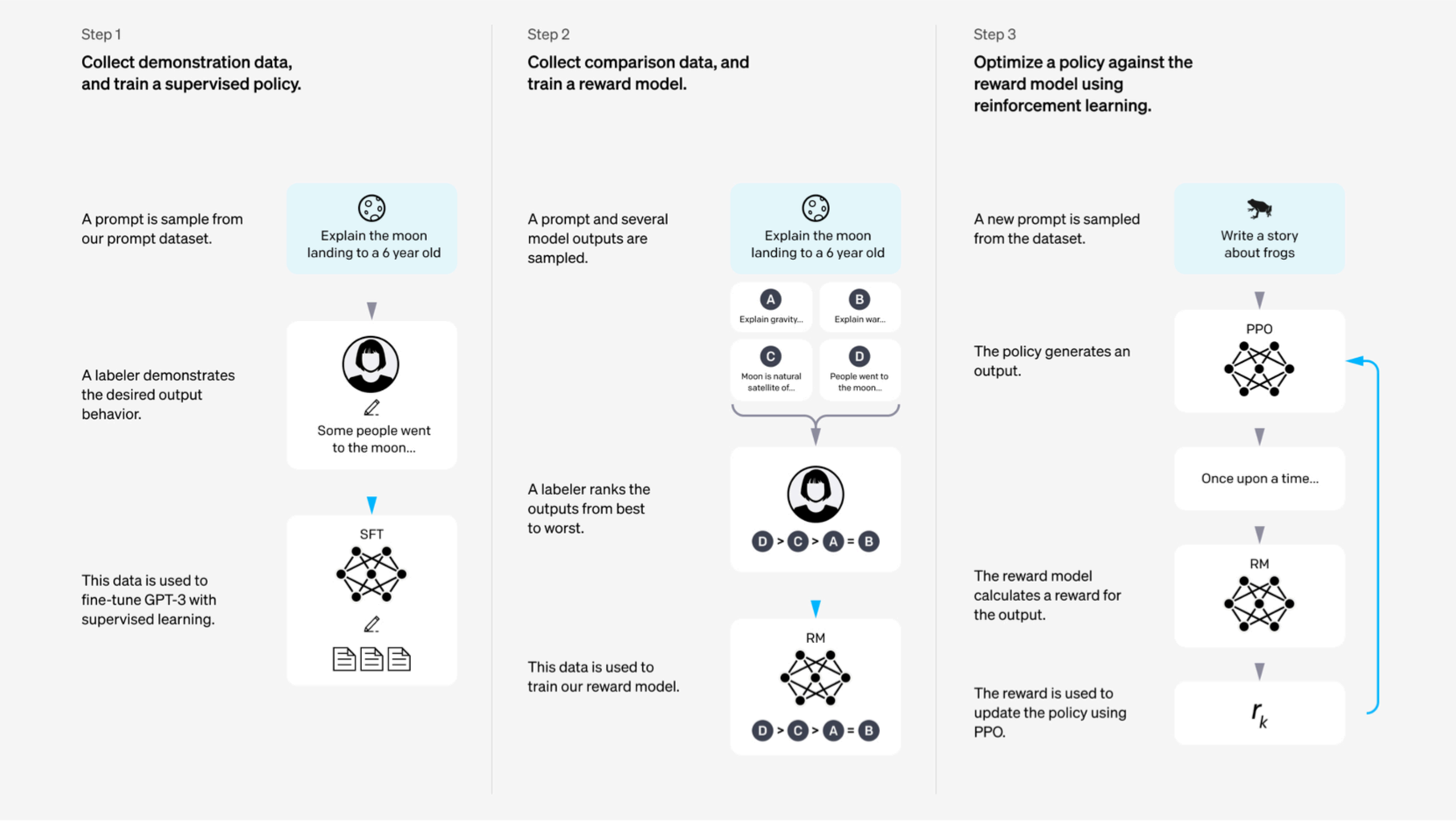


Classical RL vs. RL in LLM

- For a prompt x, LLM generates token as $y = (y_1, \ldots, y_T)$
  - Environment: None
  - State: prompt + previous tokens $s_t = (x, y_{<t})$
  - Action: next token $a_t = y_t$
  - Reward: score for the response
  - Policy: LLM $\pi_\theta(a_t \mid s_t) = \pi_\theta(y_t \mid x, y_{<t})$.
  - Trajectory: full response $\tau = (x, y_1, \ldots, y_T)$



InstructGPT's RLHF Recipe [Ouyang+ 2022]



InstructGPT's RLHF Recipe [Ouyang+ 2022]

Reinforcement learning (RL). Once again following Stiennon et al. (2020), we fine-tuned the SFT model on our environment using PPO (Schulman et al., 2017). The environment is a bandit environment which presents a random customer prompt and expects a response to the prompt. Given the prompt and response, it produces a reward determined by the reward model and ends the episode. In addition, we add a per-token KL penalty from the SFT model at each token to mitigate over-optimization of the reward model. The value function is initialized from the RM. We call these models "PPO."

We also experiment with mixing the pretraining gradients into the PPO gradients, in order to fix the performance regressions on public NLP datasets. We call these models "PPO-ptx." We maximize the following combined objective function in RL training:

$$\begin{aligned}\text{objective}\,(\phi) =& E_{(x,y)\sim D_{\pi_\phi^{\mathrm{RL}}}}\left[r_\theta(x,y) - \beta \log\left(\pi_\phi^{\mathrm{RL}}(y \mid x)/\pi^{\mathrm{SFT}}(y \mid x)\right)\right] + \\ & \gamma E_{x\sim D_{\text{pretrain}}}\left[\log(\pi_\phi^{\mathrm{RL}}(x))\right]\end{aligned} \tag{2}$$

where π_ϕ^{RL} is the learned RL policy, π^{SFT} is the supervised trained model, and D_{pretrain} is the pretraining distribution. The KL reward coefficient, β, and the pretraining loss coefficient, γ, control the strength of the KL penalty and pretraining gradients respectively. For "PPO" models, γ is set to 0. Unless otherwise specified, in this paper InstructGPT refers to the PPO-ptx models.

From SFT to PPO

From Imitation to Optimization

- Imitation (SFT): Fit $L_{\text{SFT}}(\theta) = -\sum_{t=1}^{T} \log \pi_\theta \left(y_t^\star \mid x, y_{<t}^\star\right).$
  - Pure generative modeling perspective
  - Requires samples from reference policy

- Optimization (RLHF): Find $y \sim \pi_\theta(\cdot \mid x)$ such that maximize

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)} \left[r_\phi(x, y)\right]$$

  - Maximize some reward function that we can measure

Policy Gradient

- Goal: makes high-reward samples more likely
  - If a sampled response gets high reward → increase its log- prob
  - If it gets low reward → decrease or avoid increasing its log-prob

- Expected reward that we want to maximize: $J(\theta) = \mathbb{E}_{y \sim \pi_\theta(\cdot|x)}\left[R(y)\right]$
  - In practice, we estimate this expectation with sampled rollouts

$$J(\theta) = \sum_y \pi_\theta(y \mid x) R(y)$$

  - The derivate is:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_y \pi_\theta(y \mid x) R(y) \; = \sum_y \nabla_\theta \pi_\theta(y \mid x) R(y).$$

Policy Gradient - Log Trick

- From $\nabla_\theta \log \pi_\theta(y \mid x) = \dfrac{\nabla_\theta \pi_\theta(y \mid x)}{\pi_\theta(y \mid x)}$, multiply both sides by $\pi_\theta(y \mid x)$
  - Now we get $\nabla_\theta \pi_\theta(y \mid x) = \pi_\theta(y \mid x) \nabla_\theta \log \pi_\theta(y \mid x)$
  - Substitute this into $\nabla_\theta J(\theta) = \sum_y \nabla_\theta \pi_\theta(y \mid x) R(y)$.
- $\nabla_\theta J(\theta) = \sum_y \pi_\theta(y \mid x) \nabla_\theta \log \pi_\theta(y \mid x) R(y)$
  - and since $\sum_y \pi_\theta(y \mid x)[\cdots]$ is expectation, we can rewrite this as:
  - REINFORCE: $\nabla_\theta J(\theta) = \mathbb{E}_{y \sim \pi_\theta(\cdot \mid x)} \left[R(y) \nabla_\theta \log \pi_\theta(y \mid x) \right].$

REINFORCE

- Gradient ascent on reward can be written as minimizing

$$L_{\text{REINFORCE}}(\theta) = -R(y) \log \pi_\theta(y \mid x)$$

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

- The core idea is that we sample over trials and estimate the gradient
- For a sampled answer:
  - If the return is high, it will push up the probabilities of the response
  - If the return is low, it will push down the probabilities of the response

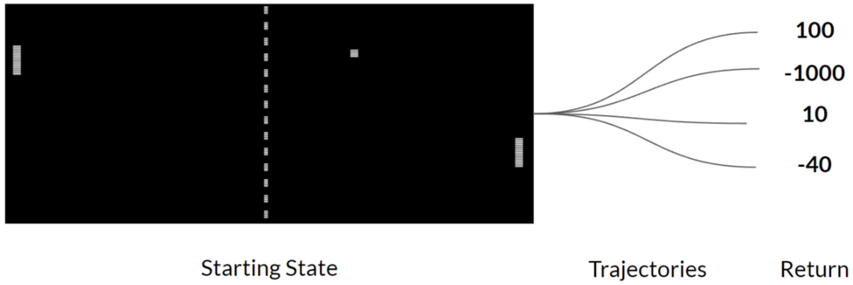
- Token-level REINFORCE: $L_{\text{REINFORCE}}(\theta) = -R(x, y) \sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t}).$

SFT vs. REINFORCE

- SFT: $L_{\text{SFT}}(\theta) = -\sum_{t=1}^{T} \log \pi_\theta \left(y_t^\star \mid x, y_{<t}^\star\right).$
  - Training text: human/model demonstration
  - Weight: always target likelihood (+1 for the target)
- REINFORCE: $L_{\text{REINFORCE}}(\theta) = -R(x,y) \sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t}).$
  - Training text: policy sample
  - Weight: reward-weighted likelihood

The Variance Problem

- REINFORCE signals are unbiased but noisy (high variance)
- Why?
  - One scalar reward is assigned to the whole response
    - It is unclear which token caused the reward

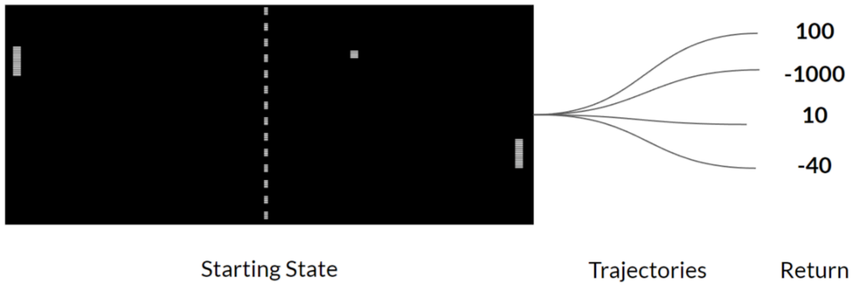


Using only the final game score, can we determine which control affects winning the game?

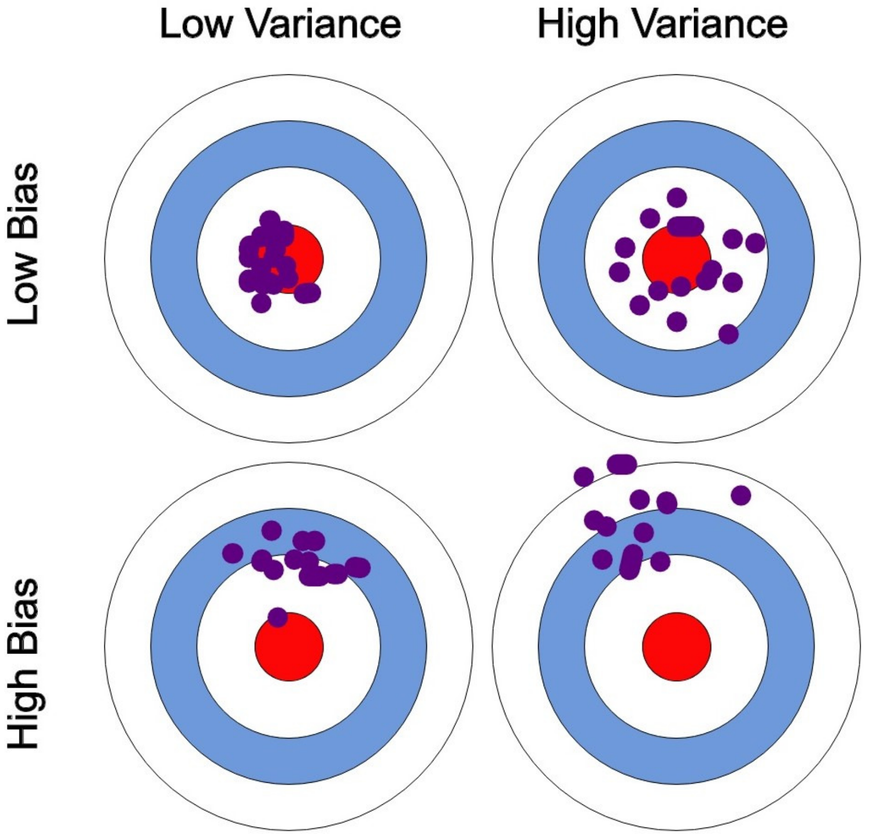
  - The absolute reward value depends on prompt difficulty
    - If reward of the completion is +5, is this good?
    - What if the average reward for this prompt is +10?

The Variance Problem

- REINFORCE signals are unbiased but noisy (h
- Why?
  - One scalar reward is assigned to the whole
    - It is unclear which token caused the rewa



Usir
can
affe



  - The absolute reward value depends on prompt difficulty
    - If reward of the completion is +5, is this good?
    - What if the average reward for this prompt is +10?

Baselines: Compare Against an Expectation

- Instead of using raw reward, subtract a reference value from reward

$$\nabla_\theta J(\theta) = \mathbb{E}\left[(R(y) - b)\nabla_\theta \log \pi_\theta(y \mid x)\right]$$

baseline

- $R(y) - b(x) > 0$: better than expected
- $R(y) - b(x) < 0$: worse than expected
- $R(y) - b(x) = 0$: average

- Note: baseline does not bias the gradient $\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[b(s)\nabla_\theta \log \pi_\theta(a \mid s)\right] = 0$.
  - A baseline that does not depend on the sampled action
  - Can reduce variance without changing the expected gradient

Baselines: Zero-Expectation

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[b(s)\nabla_\theta \log \pi_\theta(a \mid s)\right] = \sum_a \pi_\theta(a \mid s) b(s) \nabla_\theta \log \pi_\theta(a \mid s)$$

baseline that does not depend on the sampled action

$$= b(s) \sum_a \pi_\theta(a \mid s) \frac{\nabla_\theta \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}$$

$$= b(s) \sum_a \nabla_\theta \pi_\theta(a \mid s)$$

$$= b(s) \nabla_\theta \sum_a \pi_\theta(a \mid s)$$

policy is random dist. so sum of prob of action is one

$$= b(s) \nabla_\theta 1$$

$$= 0.$$

Baselines $\nabla_\theta J(\theta) = \mathbb{E}\left[(R(y) - b)\nabla_\theta \log \pi_\theta(y \mid x)\right]$

- Common baselines:
  - Average reward over the batch (simplest)

```
# rewards: (B,) – one reward per sequence
seq_log_probs = (token_log_probs * completion_mask).sum(dim=-1)

# Baseline: average reward in the batch
baseline = rewards.mean()
advantages = rewards - baseline

# REINFORCE loss (negative because we minimize)
loss = -(advantages * seq_log_probs).mean()
```

  - Moving average of the recent rewards
  - Learned value function

Value Function

- Okay, but which baseline is good?
  - Expected future return from the current state $V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[G_t \mid s_t\right]$
  - This is called as a value function
  - $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, a sum of discounted reward
    - But in LLM, if we get reward after the final completion token, we can assume $\gamma = 1$
- Hence, $V^{\pi}(s_t)$ is a expected future return from the current prefix $s_t = (x, y_{<t})$, when we generate completion with policy π

  i.e. How likely is this prefix to lead to a good response in the future (with our policy)?

Value Function

- In practice, we cannot know exact $V^{\pi}(s_t)$ hence approximate with NN
  - We will denote as $V_{\psi}(s_t)$

- Normally, we train $V_{\psi}(s_t)$ using MSE loss with RL rollout return G_t

$$L_V(\psi) = \mathbb{E}_t \left[(V_{\psi}(s_t) - G_t)^2 \right]$$

- Note: value function depends on the policy

Note: Value Function vs. Reward Model

- Do not confuse these two model!

- Reward model evaluates the quality of the completed response
  - "How preferable is the response to humans (given the prompt)?"
  - Usually, trained with preference dataset

- Value model estimates expected future return for this prefix
  - "From this partial answer, what reward should I expect?"
  - This is depends on the policy model
  - Input is prompt + prefix, not prompt + full response

Action-Value and Advantage

- Value function $V^{\pi}(s_t)$: expected future return from state s
- Action-value $Q^{\pi}(s_t, a_t)$: expected return after taking action a in state s
  - In LLM, expected return when we choose y_t, given prefix $(x, y_{<t})$

- Advantage $A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$
  - Is this *action better than expected* in this state?
  - $A^{\pi}(s_t, a_t) > 0$: this action is above average; increase probability
  - $A^{\pi}(s_t, a_t) < 0$: this action is below average; decrease probability
  - $A^{\pi}(s_t, a_t) \approx 0$: this action is average

Actor-Critic View

- Two models or heads are trained together
  - Actor: the LLM policy π_θ
    - Generates responses and receives policy gradient updates
  - Critic: the value model V_ψ
    - Criticize actor's action or state by predicting expected return
    - We can reduce variance through advantage estimates

What $R(y)$ Can be

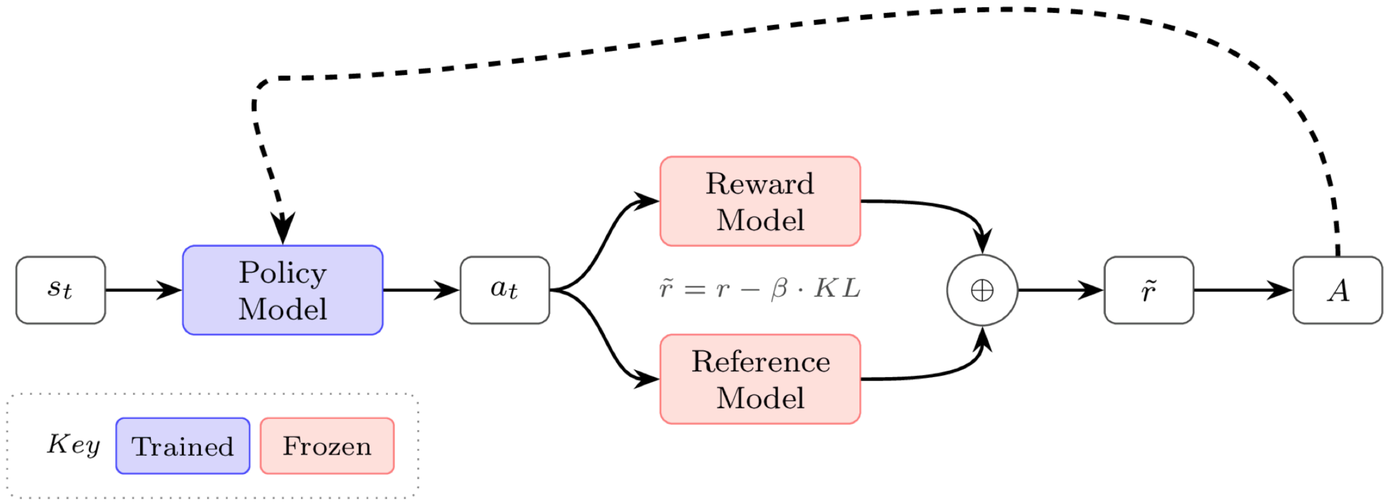
- Popular choices for $R(y)$ in $\nabla_\theta J(\theta) = \mathbb{E}_{y \sim \pi_\theta(\cdot|x)} \left[R(y) \nabla_\theta \log \pi_\theta(y \mid x) \right].$

	Description	Variance	Bias
$R(\tau) = \sum_{t=0}^{T} r_t$	Total trajectory reward	High	None
$G_t - b(s_t)$	Baselined return	Lower	None
$Q^\pi(s_t, a_t)$	State-action value function	Med	Depends
$A^\pi(s_t, a_t) = Q - V$	Advantage function	Low	None
$r_t + \gamma V(s_{t+1}) - V(s_t)$	TD residual	Low	Some

Summary So Far

- Imitation (SFT): Update the model by cloning the target distribution

- Optimization (REINFORCE): $L_{\text{REINFORCE}}(\theta) = -R(x, y) \sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t}).$
  - Updates the policy (LLM) using returns
  - Actor-critic estimates the advantage with a value function to produce more stable updates (low variance)



Basic REINFORCE architecture for language models. The shaped reward combines the reward model score with a KL penalty from the reference model.

Proximal Policy Optimization (PPO) [Schulman+ 2017]

- If REINFORCE is so simple, why PPO?
  - All the algorithms discussed so far are on-policy

  - On-policy: train on samples from the current policy
    - e.g. Generate fresh rollouts from the current policy, then update the policy using those rollouts

  - Off-policy: train on samples from another policy
    - Previous policy, other policy models, etc

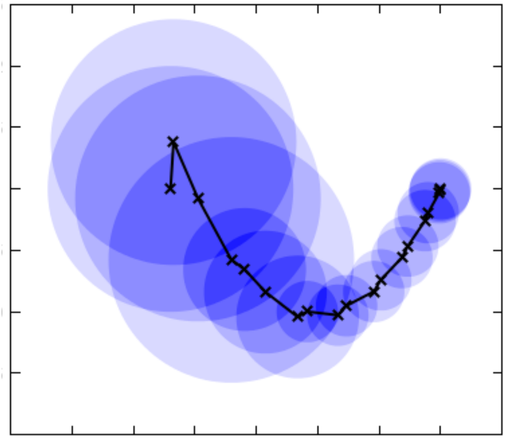
Proximal Policy Optimization (PPO) [Schulman+ 2017]

- If REINFORCE is so simple, why PPO?
  - Problem 1: Large policy updates can be unstable
    - Too large an update and the policy can collapse
    - Too small and training is painfully slow
  - Problem 2: Poor data efficiency
    - We would like to reuse collected rollouts for multiple updates
    - However, if the policy changes too much, those rollouts become increasingly off-policy
    - The data was generated by an older policy, making repeated updates less reliable

Proximal Policy Optimization (PPO) [Schulman+ 2017]

- Core idea 1: constrained updates
  - Large gradient steps can destroy the policy (instability, over-optimization, etc.)

- Trust regions limit how far the policy can move in a single update
  - TRPO [Schulman+ 2015] solved this with a hard trust-region constraint, but required expensive second-order optimization (and very hard to understand and implement)



Proximal Policy Optimization (PPO) [Schulman+ 2017]

- Core idea 2: importance sampling
  - We want to take multiple gradient steps on a batch, but the data came from an old policy $\pi_{\theta_{\text{old}}}$

- Define the importance sampling ratio $\rho_t(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)}$

  - If $\rho_t = 1$: current and old policy agree on this action
  - If $\rho_t > 1$: current policy assigns higher probability than old
  - If $\rho_t < 1$: current policy assigns lower probability than old

- This re-weights old-policy samples to estimate new-policy gradients

Proximal Policy Optimization (PPO) [Schulman+ 2017]

- Using importance sampling, the policy gradient becomes

  surrogate objective: $L^{\text{PG}}(\theta) = \mathbb{E}_t \left[\rho_t(\theta) A_t \right]$

  - A_t is advantage estimate of the rollout from the old policy

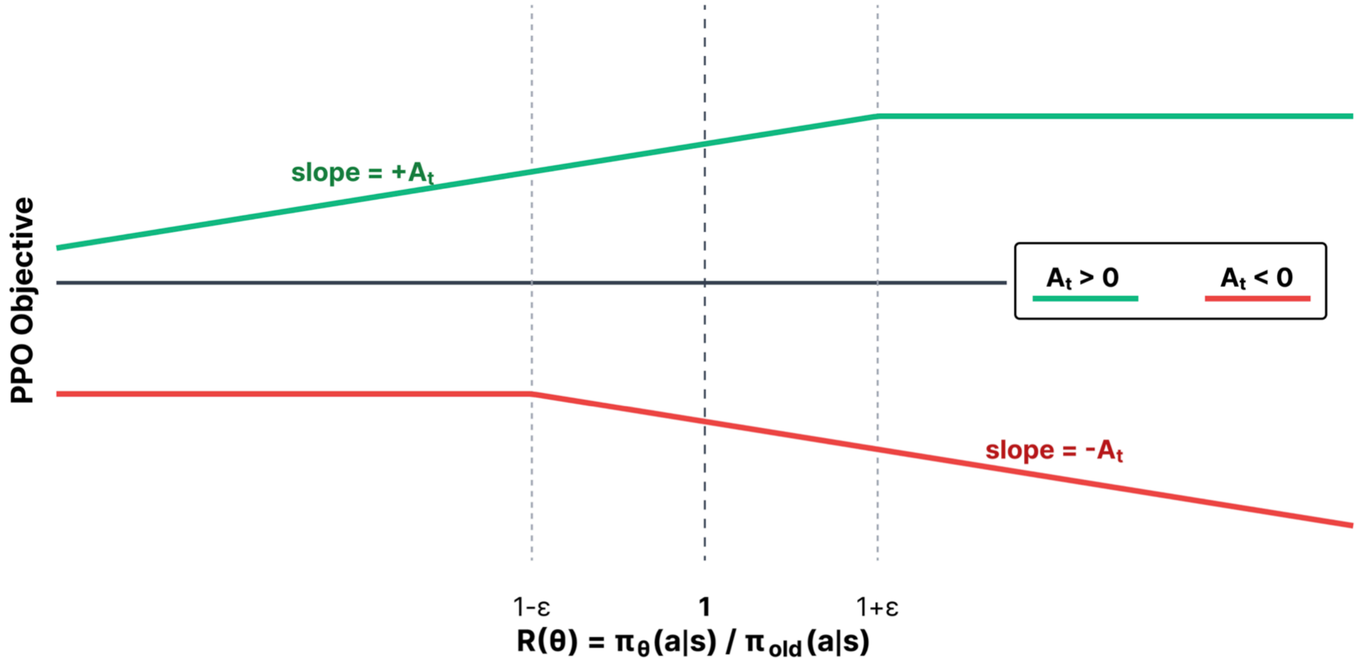
- Intermediate problem: without constraints, maximizing this can take arbitrarily large steps
  - The ratic ρ_t can diverge far from 1, making the estimate unreliable
  - TL;DR: Don't change too much

PPO Clipped Objective [Schulman+ 2017]

- PPO clips the ratio ρ_t to prevent large updates
  - A practical surrogate inspired by trust-region ideas

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

use ϵ = 0.1-0.2 in practice



PPO Clipped Objective [Schulman+ 2017]

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

- Positive advantage ($A_t > 0$)
  - Action is better than average so we want to increase its likelihood
  - $\rho_t < 1 - \epsilon$: Action is **less likely** under new policy
    - Objective: $\rho_t(\theta)A_t$ - **normal gradient**, push probability up
  - $1 - \epsilon \leq \rho_t \leq 1 + \epsilon$: Action is **roughly equally likely**
    - Objective: $\rho_t(\theta)A_t$ - **normal gradient**, push probability up
  - $\rho_t > 1 + \epsilon$: Action is **already more likely** under new policy
    - Objective: $(1+\epsilon)\,A_t$ - **gradient is zero**, no update needed (Clipped)

PPO Clipped Objective [Schulman+ 2017]

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

- Negative advantage ($A_t < 0$)
  - Action is worse than average so we want to decrease its likelihood
  - $\rho_t < 1 - \epsilon$: Action is less likely under new policy
    - Objective: $(1+\epsilon)\, A_t$ - gradient is zero, no update needed (Clipped)
  - $1 - \epsilon \leq \rho_t \leq 1 + \epsilon$: Action is roughly equally likely
    - Objective: $\rho_t(\theta)A_t$ - normal gradient, push probability down
  - $\rho_t > 1 + \epsilon$: Action is already more likely under new policy
    - Objective: $\rho_t(\theta)A_t$ - normal gradient, push probability down

PPO Clipped Objective [Schulman+ 2017]

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

- Zero advantage
  - The action was exactly as good as expected
  - The loss is zero so no update

PPO Clipped Objective [Schulman+ 2017]

- In all cases, clipping stops the gradient when the policy has already moved enough in the right direction

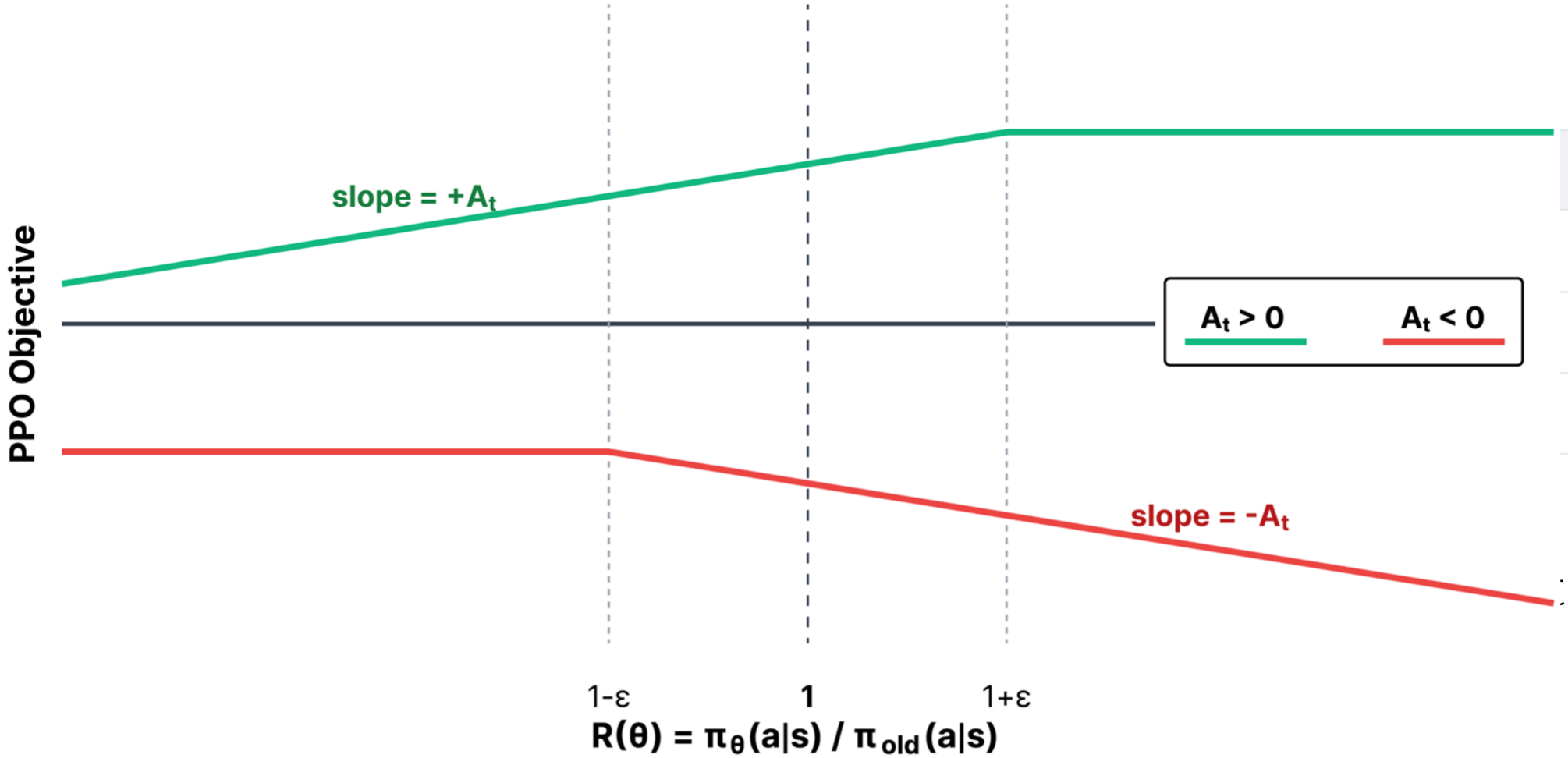
Advantage	Within trust region	Outside trust region
$\hat{A}_t > 0$	Normal gradient (reinforce)	Zero gradient if $\rho_t > 1 + \varepsilon$
$\hat{A}_t < 0$	Normal gradient (suppress)	Zero gradient if $\rho_t < 1 - \varepsilon$
$\hat{A}_t = 0$	No update	No update

- The clipping is one-sided per advantage sign
  - It caps movement when the policy has already moved enough in the beneficial direction, but never blocks correction in the detrimental direction

PPO Clipped Objective [Schulman+ 2017]

- In a
  mc

Ac
$\hat{A}_t$
$\hat{A}_t$
$\hat{A}_t$



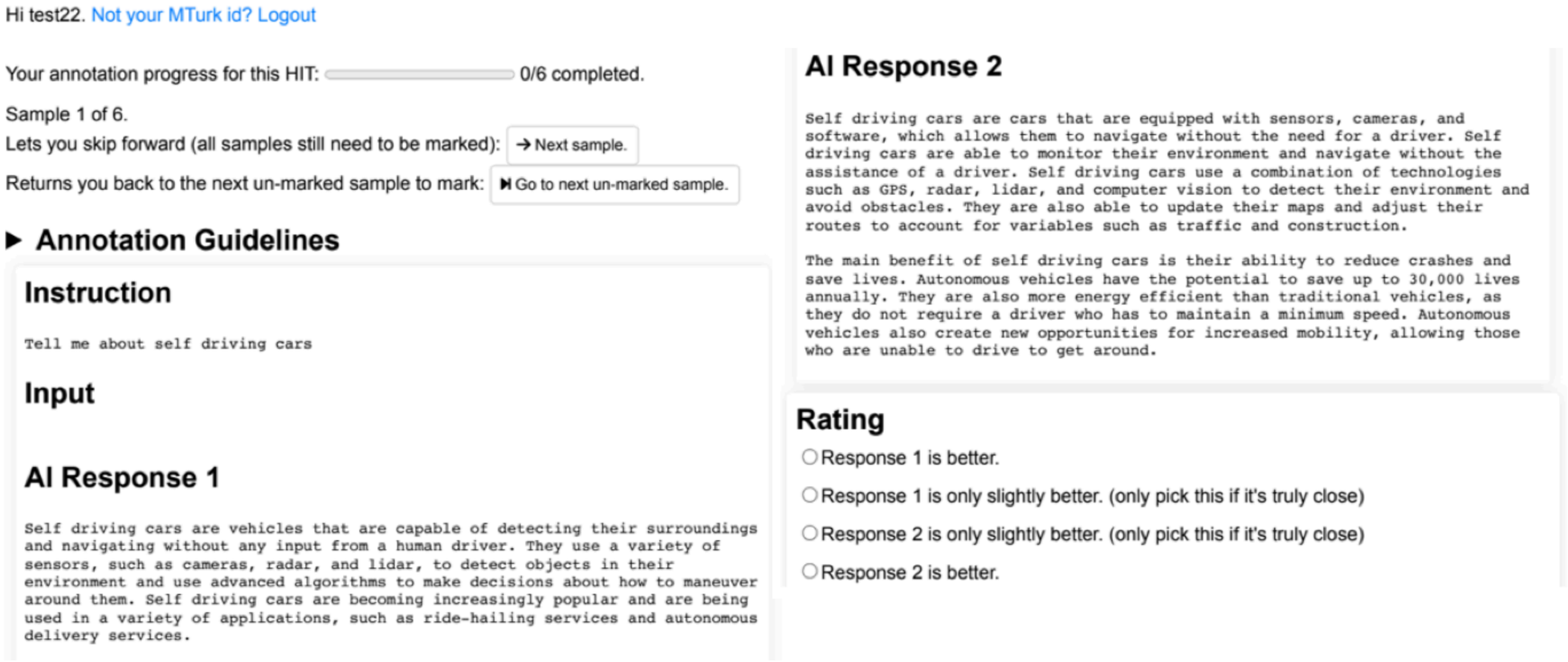
- The
  - I
    k
    c

:he

RLHF

RLHF Dataset

- How do we get (good) pairwise feedback?
- Standard pairwise feedback setup



RLHF Dataset

- How do we get (good) pairwise feedback?
- Labeling instruction of InstructGPT

Excerpt of labeling instructions on the API prompt distribution

You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog."). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story).

You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are **helpful**, **truthful**, and **harmless**. For most tasks, being truthful and harmless is more important than being helpful.

By **helpful**, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior:

- Writing in clear language.
- Answering the question they meant to ask, even if they mis-asked it.
- Being sensitive to internationality (e.g. "football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president).
- If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing.
- Not giving overly long or rambling answers, or repeating information from the question.
- Not assuming extraneous extra context outside of what's given (besides things like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body" the output shouldn't assume "I can't make it this time, but am free next weekend." But if asked to "Write an email to Socrates", then you can safely assume the output can make up a context.

By **truthful**, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior:

- On tasks like summarization, where the output should only use information from input, not making up details that are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.

By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmless behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

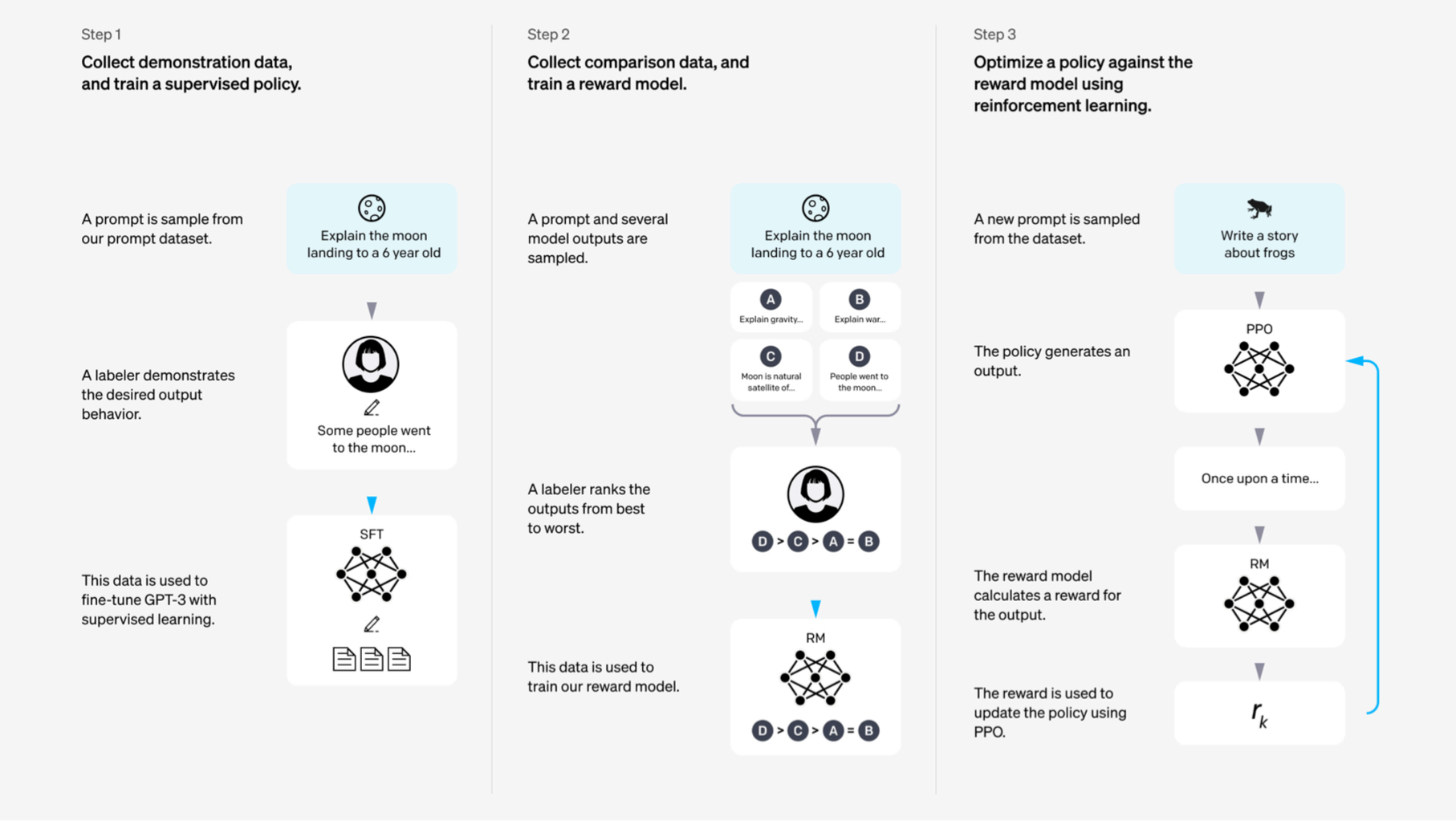
Evaluating model outputs may involve making trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be in a "high stakes domain" (e.g. loan applications, therapy, medical or legal advice, etc.); then rate the more helpful output higher. When choosing between outputs that are similarly helpful but are untruthful or harmful in different ways, ask: which output is more likely to cause harm to an end user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

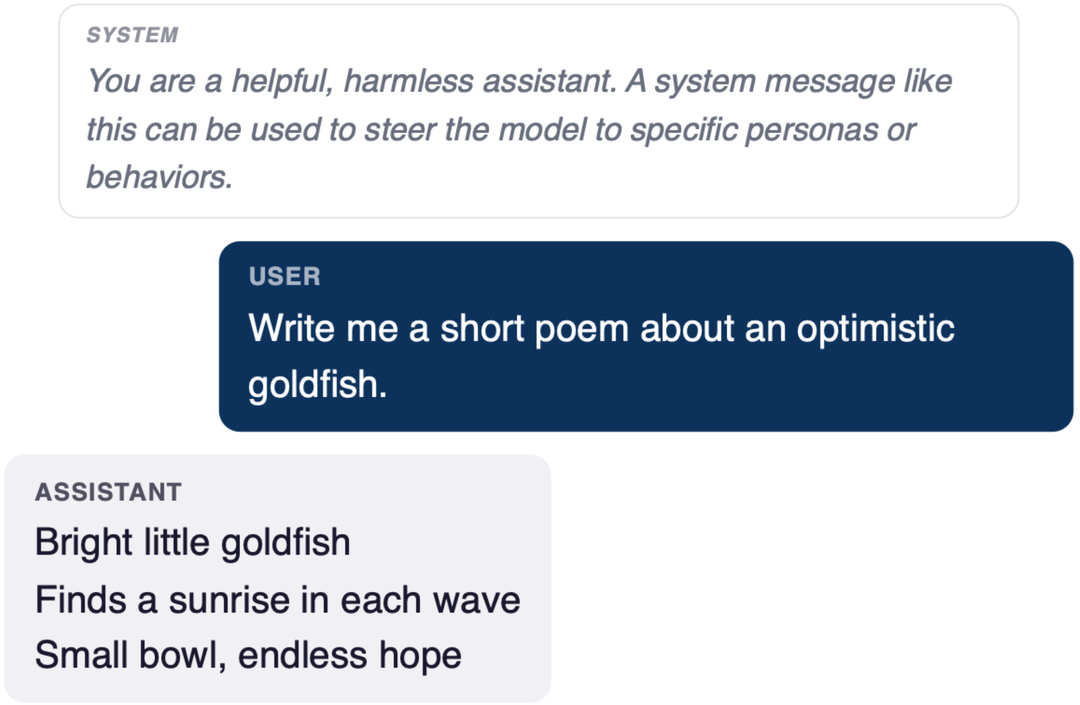
InstructGPT's RLHF Recipe [Ouyang+ 2022]



Step 1/3: SFT

- Start from pre-trained LMs
- Collect demonstrations of desired assistant behavior
- Train with standard supervised learning on prompt-response pairs

$$L_{\text{SFT}}(\theta) = -\sum_{t=1}^{T} \log \pi_\theta \left(y_t^\star \mid x, y_{<t}^\star\right).$$



Step 2/3: Reward Modeling

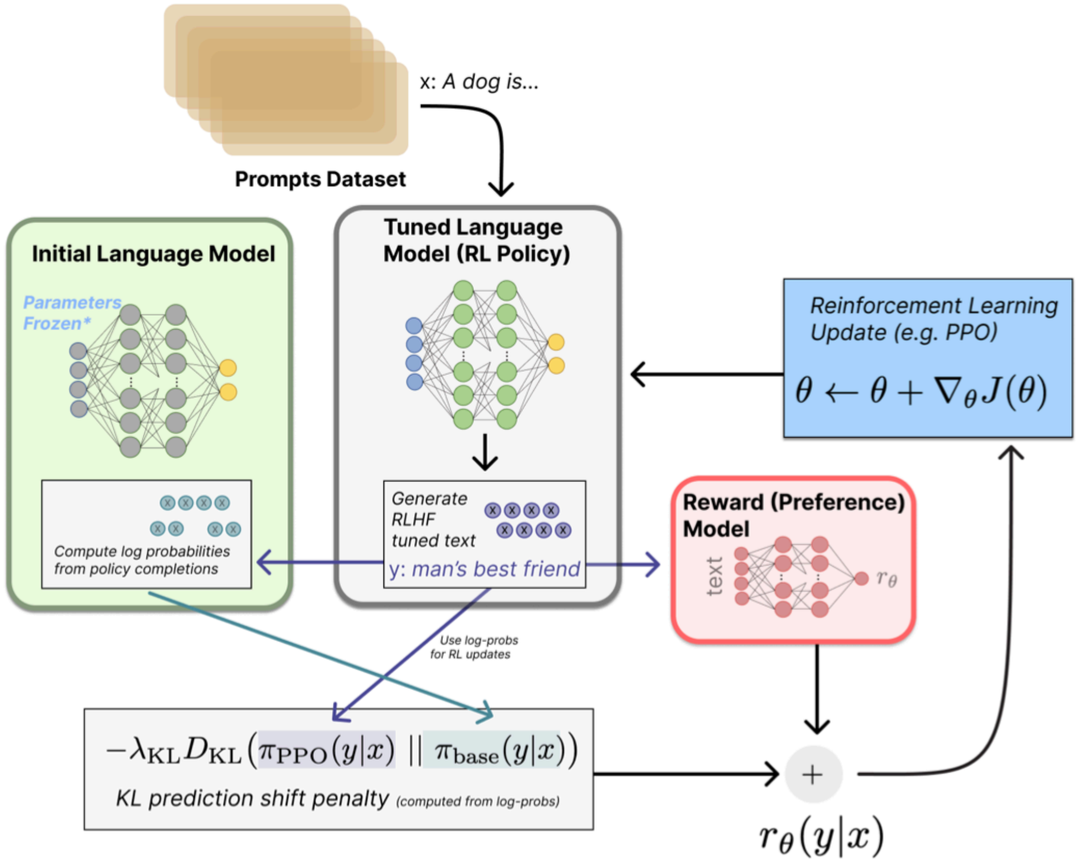
- Core idea: The reward used in RLHF is the model predicts the probability that a given piece of text would be the "winning" or "chosen" completion in a pair/batch
  - Why? there is no practical way to collect human preference on the fly during training… → approximate with a reward model!

- Collect comparisons for the same prompt (x, y_w, y_l)
  - RLHF gets its name from collecting human feedback between completions, but today much of it is AI feedback
- Train a reward model r_ϕ to score preferred completions higher

Step 3/3: RL Against RM

- Where everything comes together
- Sample a batch of prompts $x \sim \mathcal{D}$
- Generate completions $y_i \sim \pi_\theta(\cdot \mid x_i)$
- Score with the reward model $r_\phi(x_i, y_i)$
- Apply PPO or other RL algorithms
  - Add a KL penalty so the policy stays close to the SFT/reference model

$$J(\pi) = \mathbb{E}[r_\phi(x, y)] - \beta D_{\mathrm{KL}}(\pi \,\|\, \pi_{\mathrm{ref}})$$



Recall: What RLHF Data Looks Like

- A human (or AI) annotator sees two responses to the same prompt and picks the better one
- This preference pair becomes training data for the reward model
  - (x, y_w, y_l) : input prompt / selected response / rejected response

USER
Explain why the sky is blue in one sentence.

ASSISTANT (Response A ✓)
The sky is blue due to Rayleigh scattering, where shorter blue wavelengths of sunlight are scattered more by atmospheric molecules than longer wavelengths.

USER
Explain why the sky is blue in one sentence.

ASSISTANT (Response B)
The sky appears blue because of the way light interacts with the atmosphere and stuff, it's basically just physics.

The Bradley-Terry Model [Bradley & Terry 1952]

- A classical model for the outcome of the pairwise comparison
- Given two items i, j, the probability that a judge prefers i over j:

$$P(i \succ j) = \frac{p_i}{p_i + p_j}.$$

- Each item has a latent strength $p_i > 0$, so reparameterizing with $p_i = e^{r_i}$

$$P(i \succ j) = \frac{\exp(r_i)}{\exp(r_i) + \exp(r_j)} = \sigma(r_i - r_j).$$

- Only score differences matter
  - Adding the same constant to all scores leaves preferences unchanged

From Bradley-Terry to a Reward Model

- Given a prompt x, completions (y_w, y_l), $r_\phi(x, y_l)$ is an output that the reward model (RM) produce. By Bradley-Terry model,

$$P_\phi(y_w \succ y_l \mid x) = \sigma\left(r_\phi(x, y_w) - r_\phi(x, y_l)\right)$$

- We want to find the RM that maximizes this probability

- Then, we can get RM loss (see next slide for derivation)

$$L_{\text{RM}}(\phi) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}_{\text{pref}}}\left[\log \sigma\left(r_\phi(x, y_w) - r_\phi(x, y_l)\right)\right]$$

  - This loss is very similar to the binary classification loss
  - The RM is learning to classify which completion was preferred

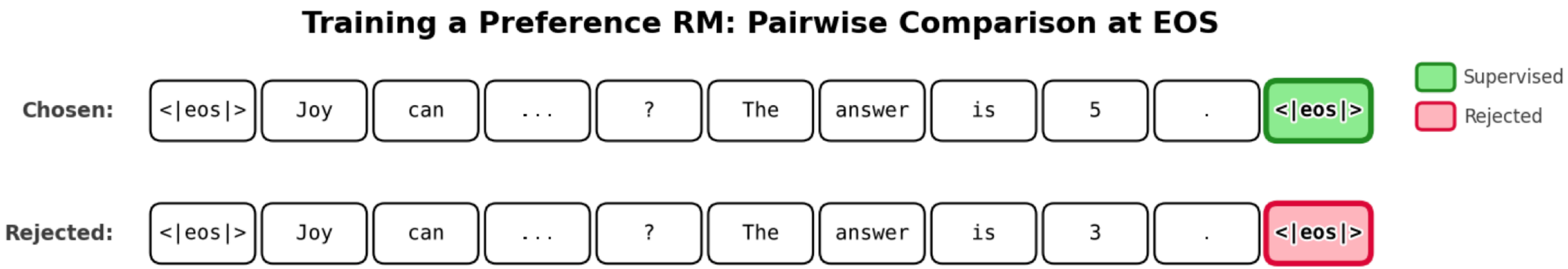
From Bradley-Terry to a Reward Model

- Derivation (from RLHFbook):

$$
\begin{aligned}
\theta^* = \arg\max_\theta P(y_c > y_r \mid x) &= \arg\max_\theta \frac{\exp\left(r_\theta(y_c \mid x)\right)}{\exp\left(r_\theta(y_c \mid x)\right) + \exp\left(r_\theta(y_r \mid x)\right)} \\
&= \arg\max_\theta \frac{\exp\left(r_\theta(y_c \mid x)\right)}{\exp\left(r_\theta(y_c \mid x)\right)\left(1 + \frac{\exp(r_\theta(y_r|x))}{\exp(r_\theta(y_c|x))}\right)} \\
&= \arg\max_\theta \frac{1}{1 + \frac{\exp(r_\theta(y_r|x))}{\exp(r_\theta(y_c|x))}} \\
&= \arg\max_\theta \frac{1}{1 + \exp\left(-(r_\theta(y_c \mid x) - r_\theta(y_r \mid x))\right)} \\
&= \arg\max_\theta \sigma\left(r_\theta(y_c \mid x) - r_\theta(y_r \mid x)\right) \\
&= \arg\min_\theta -\log\left(\sigma\left(r_\theta(y_c \mid x) - r_\theta(y_r \mid x)\right)\right)
\end{aligned}
$$

Reward Model Training

- From the pre-trained (or SFT) LLM, we train RM with RM loss



- RM computes a scalar score at the EOS token for each completion
- The most common implementation: append a linear head to a LLM that outputs a single scalar

Reward Model Training

```python
class BradleyTerryRewardModel(nn.Module):
    def __init__(self, base_lm):
        super().__init__()
        self.lm = base_lm
        self.head = nn.Linear(self.lm.config.hidden_size, 1)

    def forward(self, input_ids, attention_mask):
        hidden = self.lm(
            input_ids=input_ids, attention_mask=attention_mask,
            output_hidden_states=True, return_dict=True,
        ).hidden_states[-1]
        lengths = attention_mask.sum(dim=1) - 1
        batch_idx = torch.arange(hidden.size(0), device=hidden.device)
        seq_repr = hidden[batch_idx, lengths]
        return self.head(seq_repr).squeeze(-1)
```

RM are typically trained for only 1 epoch to avoid overfitting to the preference data

```python
rewards_chosen = model(**inputs_chosen)
rewards_rejected = model(**inputs_rejected)

loss = -nn.functional.logsigmoid(rewards_chosen - rewards_rejected).mean()
```

The Simplest Thing with RM [Gilks & Wild 1992]

- Rejection sampling is a popular baseline for preference fine-tuning
- The idea is that
  - 1. Generate many candidate completions
  - 2. Score them with RM
  - 3. Keep only the best response
  - 4. Fine-tune on the filtered responses

- No policy gradients, no online RL; just filtered supervised learning

Rejection Sampling Fine-Tuning (RSFT)

- The four stages:
  - 1. Select prompts and RM (reuse SFT prompts or curate new ones)
  - 2. Generate N completions per prompt from the current model
  - 3. Score all completions with RM
  - 4. Fine-tune on the top completions (same SFT loss)

- This is still just offline data curation: generate first, then train on the filtered outputs
- Used in WebGPT [Nakano+ 2021], Anthropic's helpful and harmless [Bai+ 2022], LLaMA2 Chat [Touvron+ 2023] and other LLMs

Rejection Sampling

- Given M prompts and N completions each:

$$X = [x_1, x_2, \ldots, x_M]$$

$$Y = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,N} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{M,1} & y_{M,2} & \cdots & y_{M,N} \end{bmatrix} \quad R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,N} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{M,1} & r_{M,2} & \cdots & r_{M,N} \end{bmatrix}$$

- Each reward: $r_{i,j} = \mathcal{R}(y_{i,j} \mid x_i)$

Rejection Sampling

- Selection method 1: Top per prompt
  - Select the best completion for each prompt independently

$$S(R) = [\arg\max_j r_{1,j},\ \arg\max_j r_{2,j},\ \ldots,\ \arg\max_j r_{M,j}]$$

- Example (5 prompts, 4 completions):

$$R = \begin{bmatrix} \mathbf{0.7} & 0.3 & 0.5 & 0.2 \\ 0.4 & \mathbf{0.8} & 0.6 & 0.5 \\ \mathbf{0.9} & 0.3 & 0.4 & 0.7 \\ 0.2 & 0.5 & \mathbf{0.8} & 0.6 \\ 0.5 & 0.4 & 0.3 & \mathbf{0.6} \end{bmatrix} \qquad S(R) = [1, 2, 1, 3, 4]$$

- Result: one completion per prompt, every prompt represented

Rejection Sampling

- Selection method 2: Top overall pairs
  - Flatten the reward matrix and select the top K pairs globally

$$S_K(R_{\text{flat}}) = \text{argsort}(R_{\text{flat}})[-K:]$$

- Example (5 prompts, 4 completions):

$$R = \begin{bmatrix} \mathbf{0.7} & 0.3 & 0.5 & 0.2 \\ 0.4 & \mathbf{0.8} & 0.6 & 0.5 \\ \mathbf{0.9} & 0.3 & 0.4 & \mathbf{0.7} \\ 0.2 & 0.5 & \mathbf{0.8} & 0.6 \\ 0.5 & 0.4 & 0.3 & 0.6 \end{bmatrix}$$

Prompt 3 gets two completions (0.9 and 0.7), while prompt 5 gets none

- This optimizes for absolute quality but can bias toward easy prompts.

Best-of-N Sampling [Liu+ 2024]

- Best-of-N (BoN) follows the rejection sampling, but skips fine-tuning step

$$S(R) = \arg\max_{j \in [1,N]} r_j$$

  - Used at inference time to pick the best completion
  - Does not modify the model. it's a sampling technique
  - Often used as a baseline comparison for online RL methods

- BoN is the simplest possible reward-guided method
  - Generate more, pick the best
  - Can also be done with verification / LLM-as-a-judge instead of traditional reward models

The Full Pipeline (without RL)

- A simple path from pre-trained model to preference-tuned model

- 1. SFT the pre-trained model → learns the chat format
- 2. Collect preference data
- 3. Train RM on preference data → learns to score quality
- 4. Rejection sampling → generate completions, filtered these, fine-tune

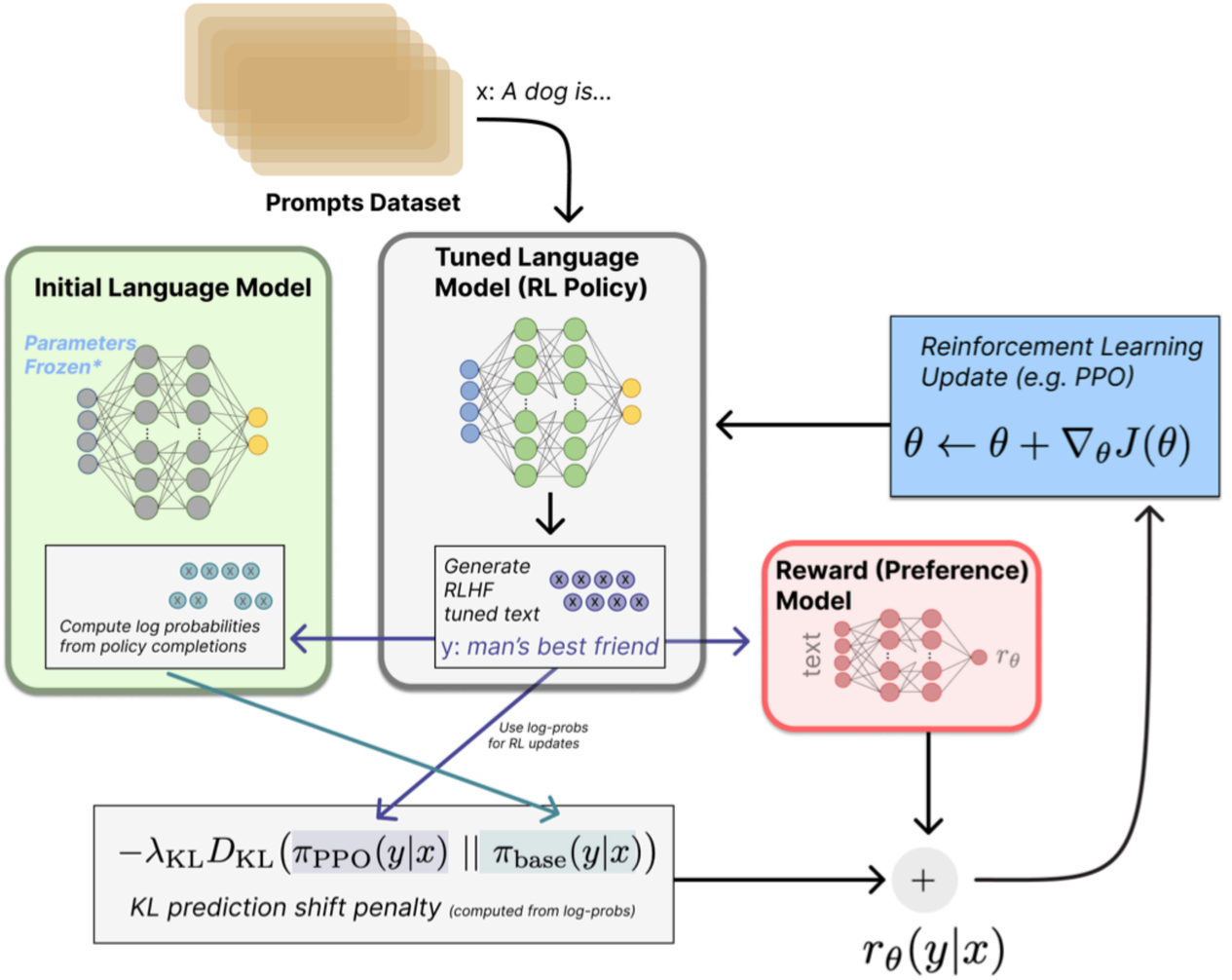
- This is a strong baseline and often used along with RL training

3. DeepSeek-R1

Although DeepSeek-R1-Zero exhibits strong reasoning capabilities, it faces several issues. DeepSeek-R1-Zero struggles with challenges like poor readability, and language mixing, as DeepSeek-V3-Base is trained on multiple languages, especially English and Chinese. To address these issues, we develop DeepSeek-R1, whose pipeline is illustrated in Figure 2.

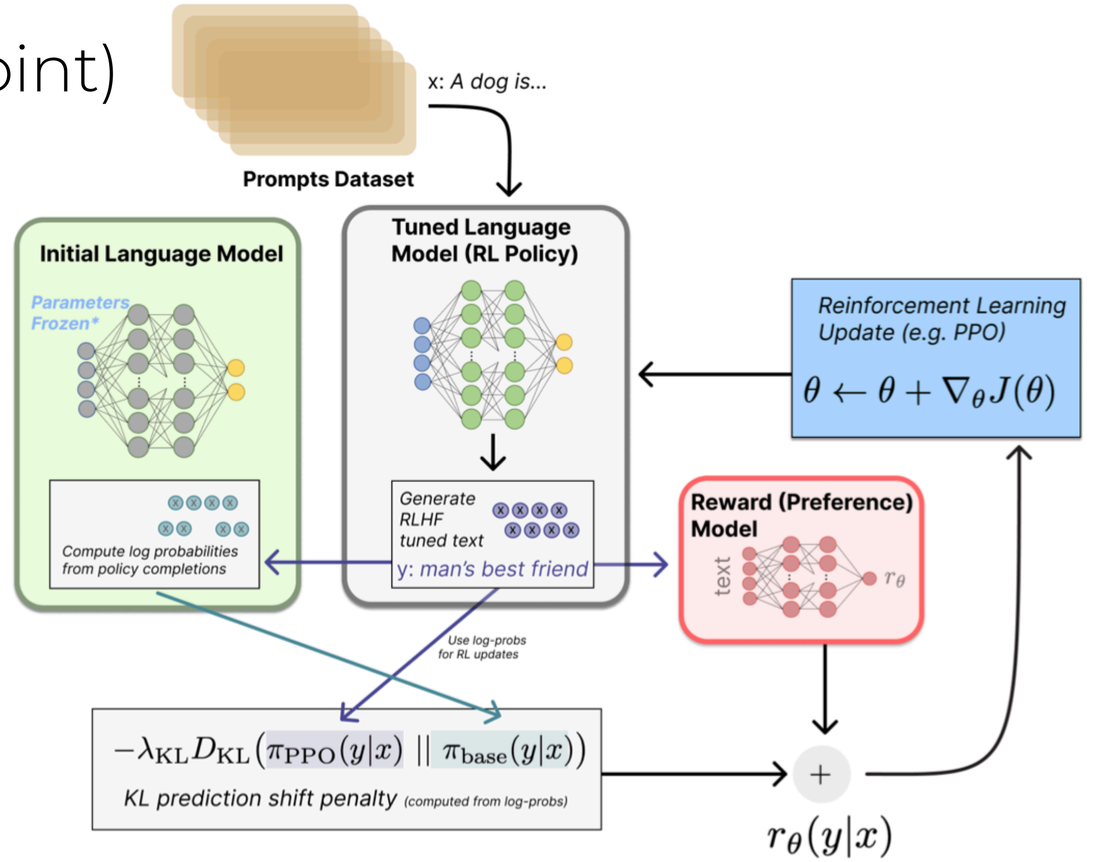
In the initial stage, we collect thousands of cold-start data that exhibits a conversational, human-aligned thinking process. RL training is then applied to improve the model performance with the conversational thinking process and language consistency. Subsequently, we apply rejection sampling and SFT once more. This stage incorporates both reasoning and non-reasoning datasets into the SFT process, enabling the model to not only excel in reasoning tasks but also demonstrate advanced writing capabilities. To further align the model with human preferences, we implement a secondary RL stage designed to enhance the model's helpfulness and harmlessness while simultaneously refining its reasoning capabilities.

Revisit the RLHF Training Loop



Reference Model

- The reference model π_{ref} is a frozen copy of the policy (typically SFT checkpoint)
- It serves one purpose: anchor the policy so it doesn't drift too far during RLHF optimization
- Without it, the policy can exploit the RM and find high-scoring outputs that are degenerate or repetitive
  - Reward hacking

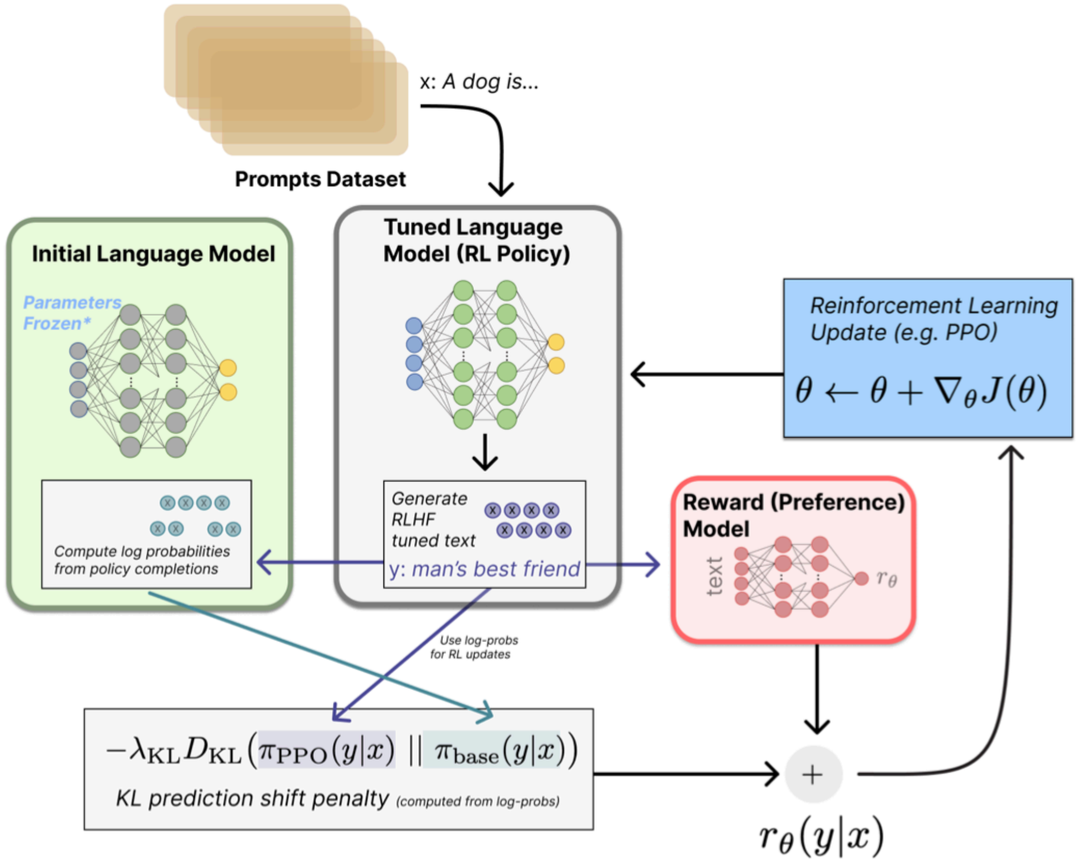


KL Divergence as Regularization

- The KL penalty measures how far the current policy has drifted from π_{ref} at each token

$$\mathbf{KL}_t = \log \pi_\theta(a_t \mid s_t) - \log \pi_{\text{ref}}(a_t \mid s_t)$$

- In practice:
  - 1. Generate a completion from π_θ
  - 2. Compute log-probs of those same tokens under both π_θ and π_{ref}
  - 3. Subtract to get the per-token KL



Note: Two Old Policies

- Don't confuse these!

	$\pi_{\theta_{\text{old}}}$	π_{ref}
What	Policy at last rollout	Policy at start of RL training (SFT checkpoint)
Updates	Every batch (or every K steps)	Never (frozen)
Used for	Importance-sampling ratio ρ_t	KL penalty
If dropped	Must use 1 gradient step per batch (on-policy)	Risk of reward hacking

Value Function

- PPO trains a value function $V_\phi(s)$ alongside the policy

$$V_\phi(s_t) = \mathbb{E}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} \,\middle|\, s_t\right]$$

expected future return from state s_t

  - Separate parameters ϕ (often initialized from the RM or SFT model)
  - Predicts post-KL shaped future return at each token position
  - Trained via MSE against post-KL returns: $\mathcal{L}_V = \frac{1}{2}(V_\phi(s_t) - G_t)^2$

- Value function serves as a learned baseline for advantage estimation

Token-Level Returns

- RLHF often starts with one scalar reward for the whole completion
- How does that become a per-token training signal?

$$r_t = \begin{cases} R(x, y) - \beta\,\mathrm{KL}_t & \text{if } t = T \text{ (final token)} \\ -\beta\,\mathrm{KL}_t & \text{otherwise} \end{cases}$$

  - where $\mathrm{KL}_t = \log \pi_\theta(a_t \mid s_t) - \log \pi_{\text{ref}}(a_t \mid s_t)$

- These per-token rewards feed into GAE, which propagates credit backward to assign per-token advantages

Recap: Advantage Estimation

- The simplest advantage is $\hat{A}_t = G_t - V_\phi(s_t)$ where $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$
- Why advantages help?
  - Centering reduces variance
    - We're asking "how much better is this action than average?" rather than "how good was the total return?"
  - Credit assignment
    - With per-token values, each token gets its own advantage signal

- Monte Carlo estimate is simple and unbiased, but it can be high variance
- Temporal Difference (TD) methods and Generalized Advantage Estimation (GAE) trade some bias for lower variance

Temporal Difference (TD) Residual

- The temporal difference (TD) residual measures how much the actual reward exceeded the value prediction

$$\delta_t^V = R_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

- This is the 1-step advantage estimate so low variance (uses learned value function) but potentially high bias (if value function is inaccurate)

Generalized Advantage Estimation (GAE)

- We can extend to K steps:

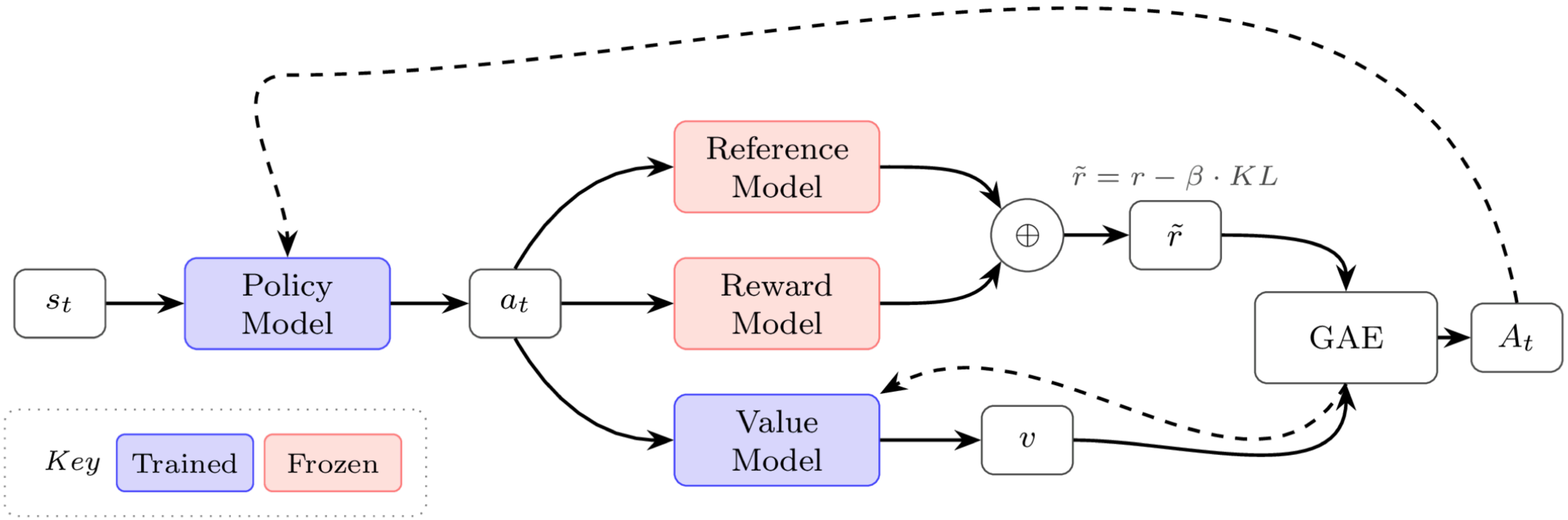
$$\hat{A}_t^{(1)} = \delta_t^V \qquad \textbf{(1-step: low variance, high bias)}$$

$$\hat{A}_t^{(2)} = \delta_t^V + \gamma\delta_{t+1}^V \qquad \textbf{(2-step)}$$

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V \qquad \textbf{(k-step: more variance, less bias)}$$

- As $k \to \infty$, we recover the full Monte Carlo advantage

PPO Architecture



PPO architecture for language models. Adds a trained value model and GAE for per-token advantage estimation.

RLHF Training with PPO

- A schematic high-level view of PPO-RLHF combines the clipped PPO objective with a KL regularizer:

$$J(\theta) \approx L^{CLIP}(\theta) - \beta\, D_{\mathrm{KL}}[\pi_\theta \| \pi_{\mathrm{ref}}]$$

- Two layers of regularization:
  - Clipping: limits how far the policy moves per batch (trust region)
  - KL penalty: limits total drift from the reference policy across training

- These serve different purposes and are not redundant

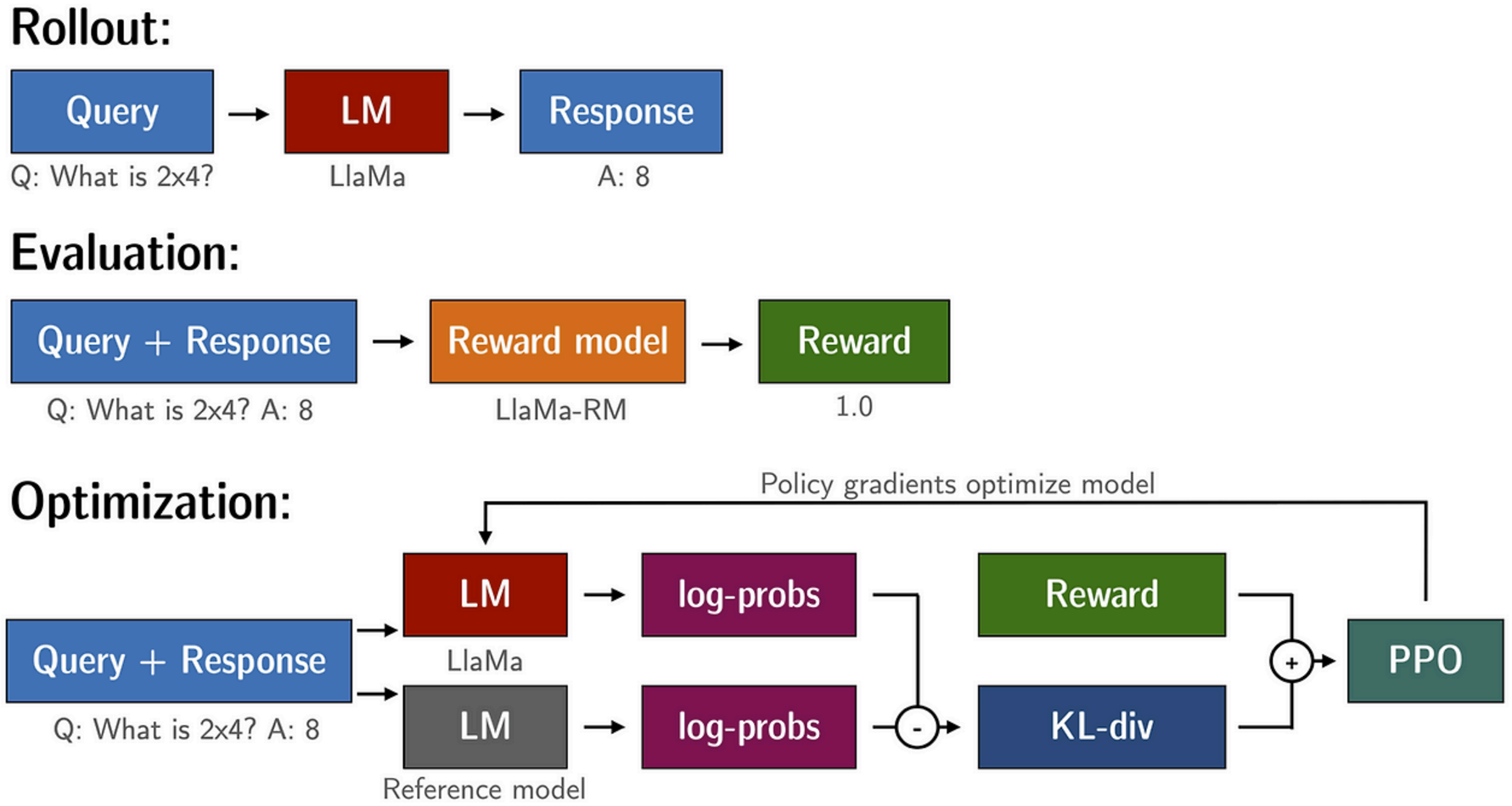
RLHF Training with PPO

- The PPO training loop (simple version)

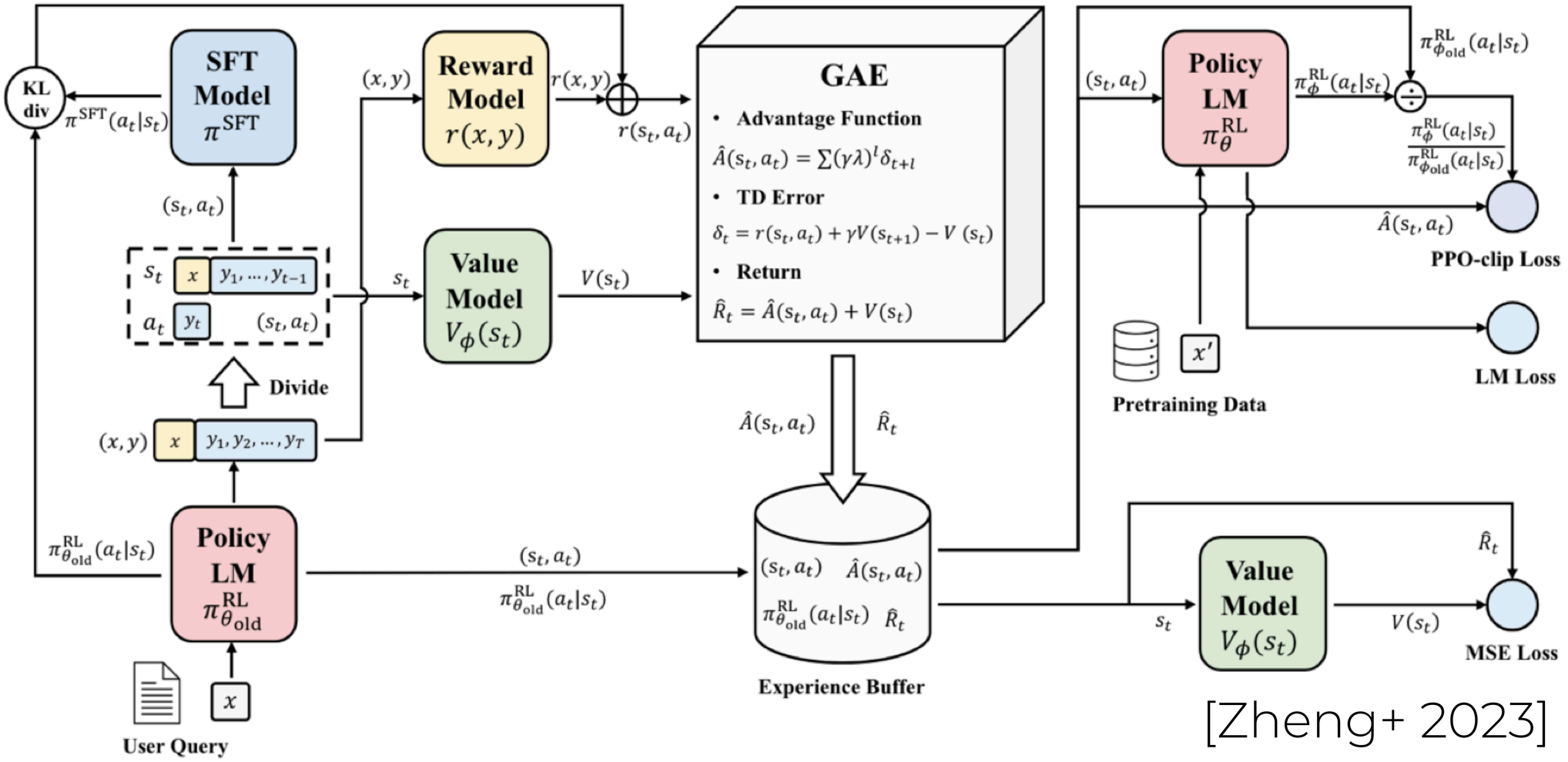
1. Generate: sample prompts, generate completions with π_θ
2. Score: compute rewards with RM + KL penalty from π_{ref}
3. Estimate advantages: compute GAE using learned value function V_ϕ
4. Update (K epochs):
   - Clipped policy gradient + value function loss on the same batch
5. Repeat: new batch, sync $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$

- Typical: K = 2-4 gradient steps per batch before re-rollout

RLHF Training with PPO



RLHF Training with PPO



[Zheng+ 2023]

RLHF Training with PPO

- Which model do we need with PPO?
- For a 7B model with fp16,

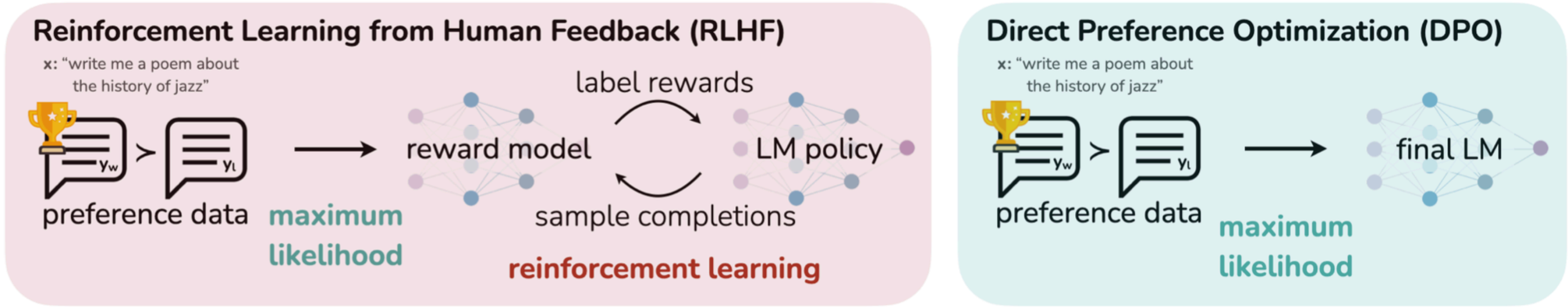
Model	Size	Purpose
Policy π_θ	~14 GB	Being trained
Value function V_ϕ	~14 GB	Learned critic
Reference policy π_{ref}	~14 GB	KL anchor (frozen)
Reward model r_ψ	~14 GB	Scoring (frozen)

- ~56 GB VRAM just for model weights before optimizer states, activations, or gradients!
- Can we simplify PPO?

DPO: Learning Directly from Preference

- Try to simplify PPO by..
  - Getting rid of RM, value function
  - Getting rid of any on-policy stuff (rollouts, outer loops etc)

- Instead, DPO (direct preference optimization) [Rafailov+ 2023]
  - Take gradient steps on log-loss of good stuff
  - Take negative gradient steps on bad stuff (appropriately weighted)



DPO: Derivation from the RLHF

- KL-regularized RLHF optimizes $\max_{\pi} \mathbb{E}_{y\sim\pi(\cdot|x)}\left[r(x,y)\right] - \beta D_{\mathsf{KL}}\left(\pi(\cdot \mid x) \,\|\, \pi_{\mathsf{ref}}(\cdot \mid x)\right)$
  - DPO starts from this same objective, then removes the explicit RM
  - Rewrite this as $\max_{\pi} \sum_{y} \pi(y \mid x) r(x,y) - \beta \sum_{y} \pi(y \mid x) \log \frac{\pi(y \mid x)}{\pi_{\mathsf{ref}}(y \mid x)}$ and $\sum_{y} \pi(y \mid x) = 1$
  - Then, the optimal policy has form:

$$\pi^{\star}(y \mid x) = \frac{1}{Z(x)} \pi_{\mathsf{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y)\right) \quad Z(x) = \sum_{y} \pi_{\mathsf{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y)\right)$$

    - The optimal policy re-weights the reference policy by exponentially increasing the probability of high-reward responses

DPO: Derivation from the RLHF

- From $\pi^\star(y \mid x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x, y)\right)$, divide both sides by $\pi_{\text{ref}}(y \mid x)$ and take the log of both sides
- We get $\log \frac{\pi^\star(y \mid x)}{\pi_{\text{ref}}(y \mid x)} = -\log Z(x) + \frac{1}{\beta} r(x, y)$
- Hence, we can write the reward as $r(x, y) = \beta \log \frac{\pi^\star(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$
  - For pairwise comparisons, Z(x) cancels out
  - So preference learning can use policy log-ratios directly
  - This is called the implicit reward

$$r_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \text{constant depending on } x$$

DPO parameterizes the optimal policy π^* directly with π_θ

DPO: Derivation from the RLHF

- Combine the Bradley-Terry model with the implicit reward

$$P(y_w \succ y_l \mid x) = \sigma\left(r(x, y_w) - r(x, y_l)\right) \quad r_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + C(x)$$

- Hence, the preference probability becomes

$$P_\theta(y_w \succ y_l \mid x) = \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)$$

- This gives the DPO loss function

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

DPO Loss

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

- DPO increases the probability of chosen responses relative to the reference model, and decreases that of rejected responses
- DPO loss gradient is

$$z_\theta = \beta\left[\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right]$$

$$\nabla_\theta L_{\text{DPO}} = -\beta\,\sigma(-z_\theta)\,[\nabla_\theta\log\pi_\theta(y_w \mid x) - \nabla_\theta\log\pi_\theta(y_l \mid x)]$$

higher weight when (implicit) reward estimate is wrong

If policy already prefers y_w over y_l, less gradient

increase likelihood of y_w

decrease likelihood of y_l

- DPO works like adaptively weighted preference matching

PPO vs. DPO

Aspect	PPO-RLHF	DPO
Feedback	RM score	preference pair
Reward model	explicit	implicit
Sampling during update	yes	no, usually offline
Main regularizer	KL to reference	reference log-ratio
Optimization	RL loop	supervised-style loss

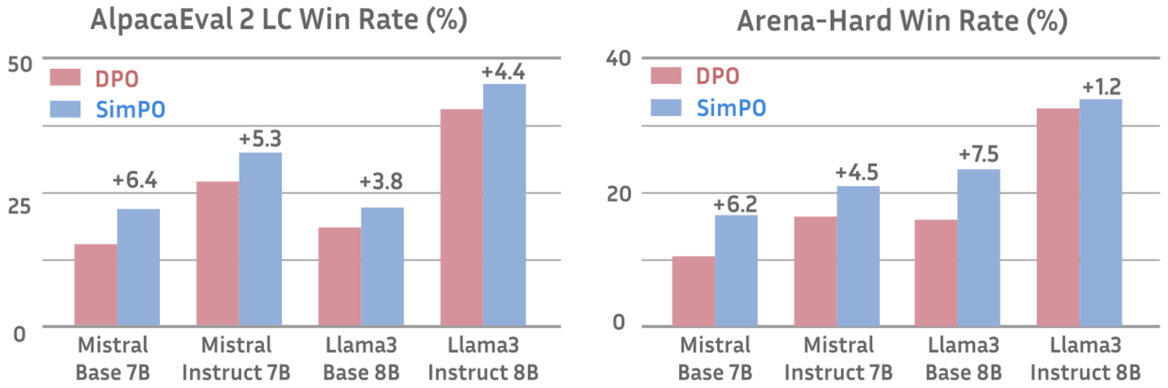
- PPO requires four models: policy, reference, value function, RM
- DPO requires two models: policy, reference

DPO Variants

- SimPO [Meng+ 2024]: reference-free DPO

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

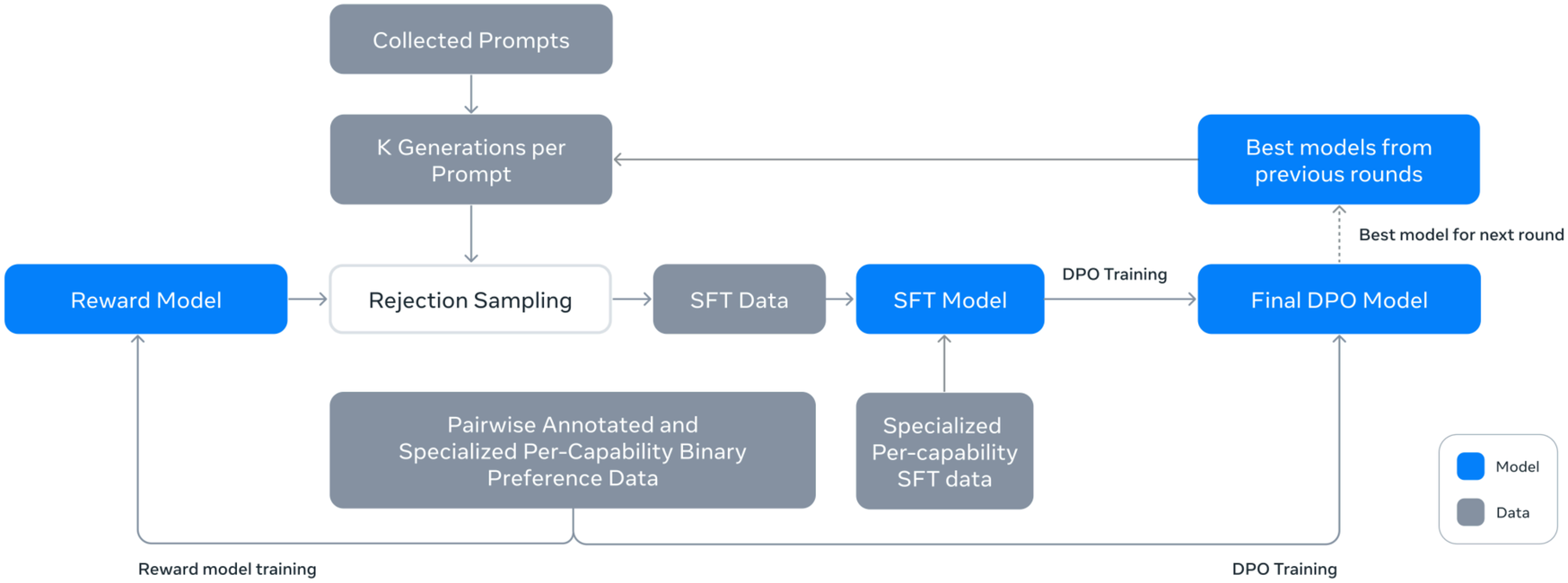
$$\mathcal{L}_{\text{SimPO}}(\pi_\theta) = -\mathbb{E}\left[\log \sigma\left(\frac{\beta}{|y_w|} \log \pi_\theta(y_w \mid x) - \frac{\beta}{|y_l|} \log \pi_\theta(y_l \mid x) - \gamma\right)\right]$$



- Length normalized rewards to reduce length bias
  - DPO use $\sum_t \log \pi_\theta(y_t \mid x, y_{<t})$, so response length affects a lot
- Target reward margin γ

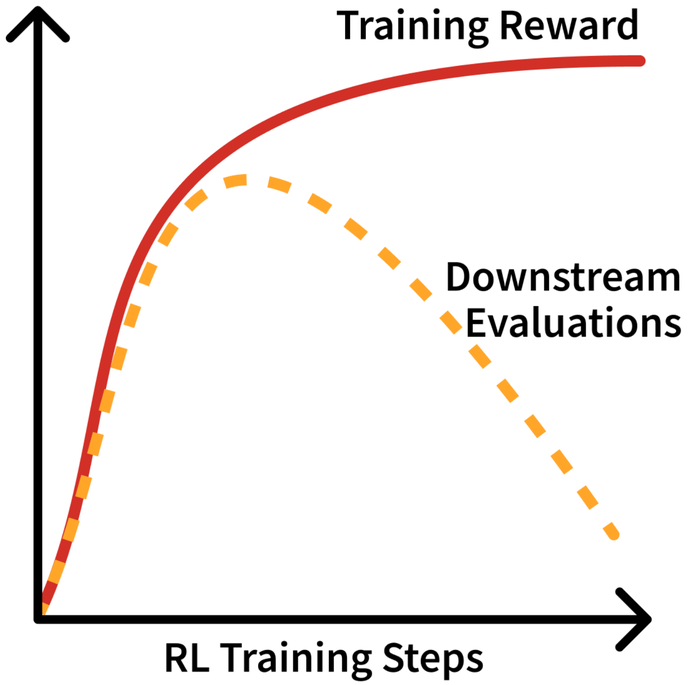
DPO Variants

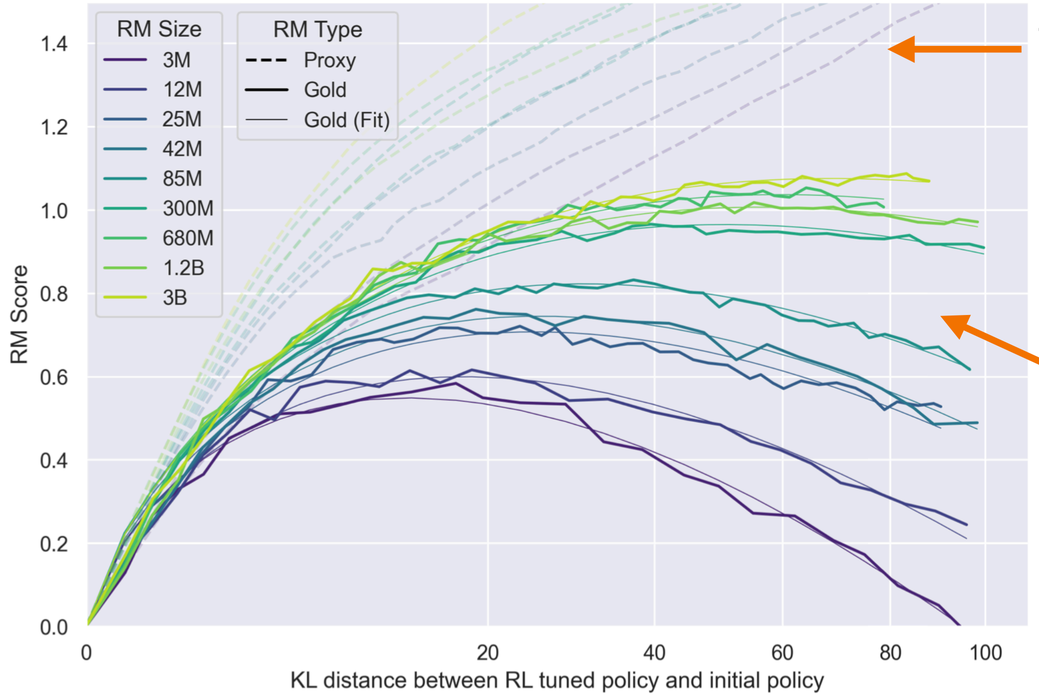
- Iterative DPO: used in LLaMA 3 and Tulu 3
  - Vanilla DPO use the fixed preference dataset
  - But the policy becomes far from the original model
  - Iterative DPO repeatedly generate, label, and optimize new data



Things to Watch Out for RLHF

- Over-optimization
  - If we optimize it too aggressively, the policy learns to exploit the reward model rather than satisfy humans (Goodhart's law?)





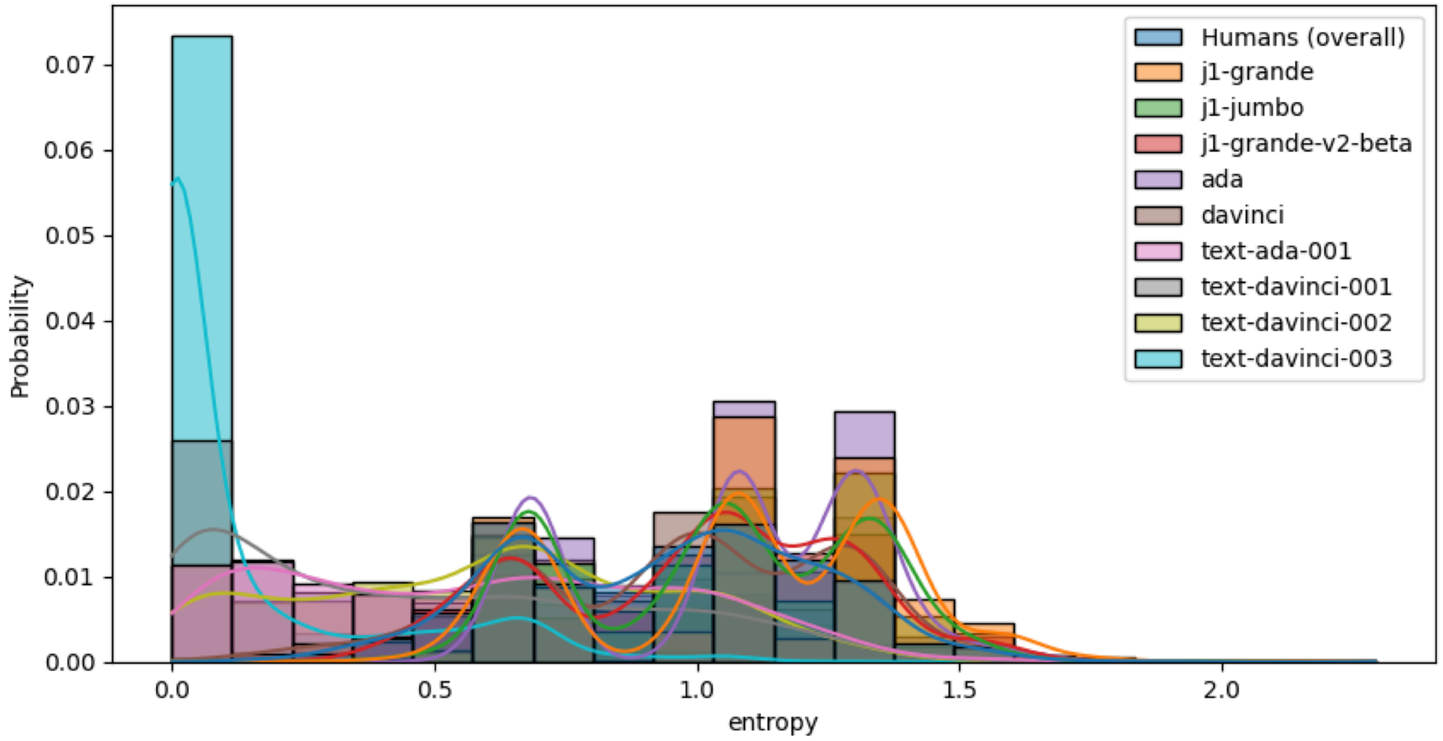
Trained reward

GT reward

[Gao+ 2022]

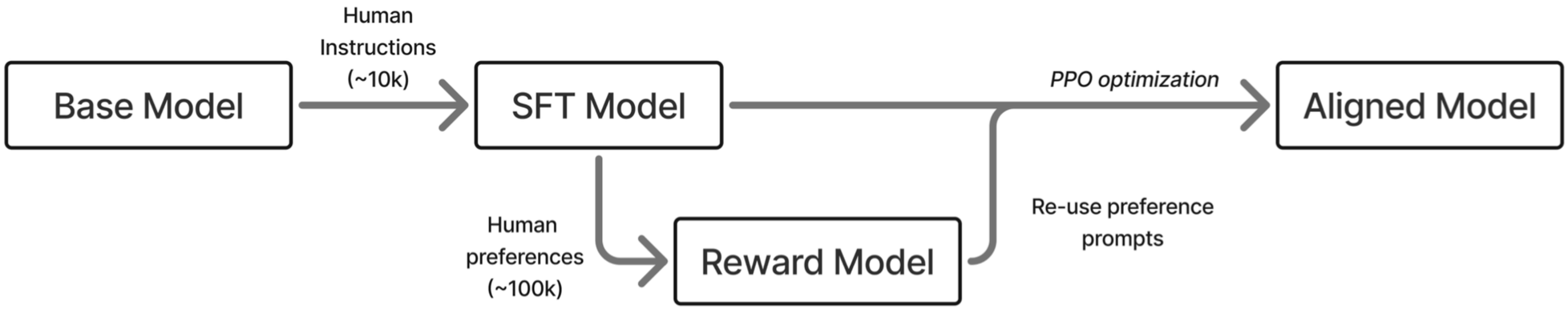
Things to Watch Out for RLHF

- Mode collapse
  - RLHF makes the model more aligned on average, but if optimized too aggressively, it can make the model less diverse



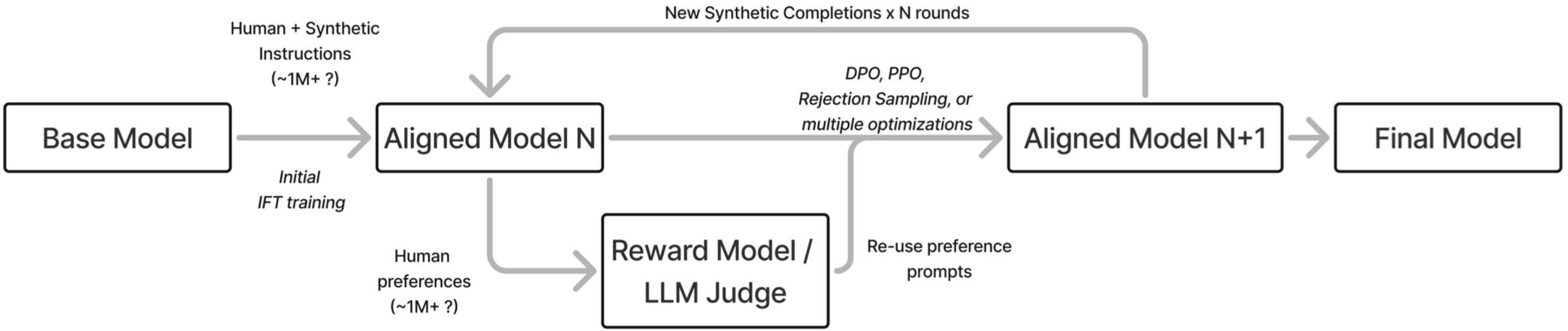
[Santurkar+ 2023]

The Early Days: InstructGPT [Ouyang+ 2022]



- Early on, RLHF had a well-documented, simple enough approach
  - InstructGPT made the classic three-stage recipe canonical:
    - SFT, reward modeling, then RL against the reward model
  - OpenAI even hinted that the original ChatGPT used this!
- This became the intellectual template for much of modern post-training

From RLHF to Post-Training



- What began as an RLHF recipe evolved into a complex series of steps to get the final, best model
  - Modern systems keep the same core idea but add more stages, more dataset, and more filtering

From RLHF to Post-Training

- As time has passed since ChatGPT,
  the field has gone through multiple distinct phases (roughly):
  - 2023: Simple SFT for better chatbots and reproducing RLHF fundamentals (Alpaca, Vicuna, etc.)
  - 2024: DPO dominates open models and training stages expand
  - 2025: RLVR, complex recipes (DS-R1, Tülu 3, OLmo 3, etc.)
  - 2026: Agentic training, multi-turn RL, etc.