Annotated Data

Aligning LLMs

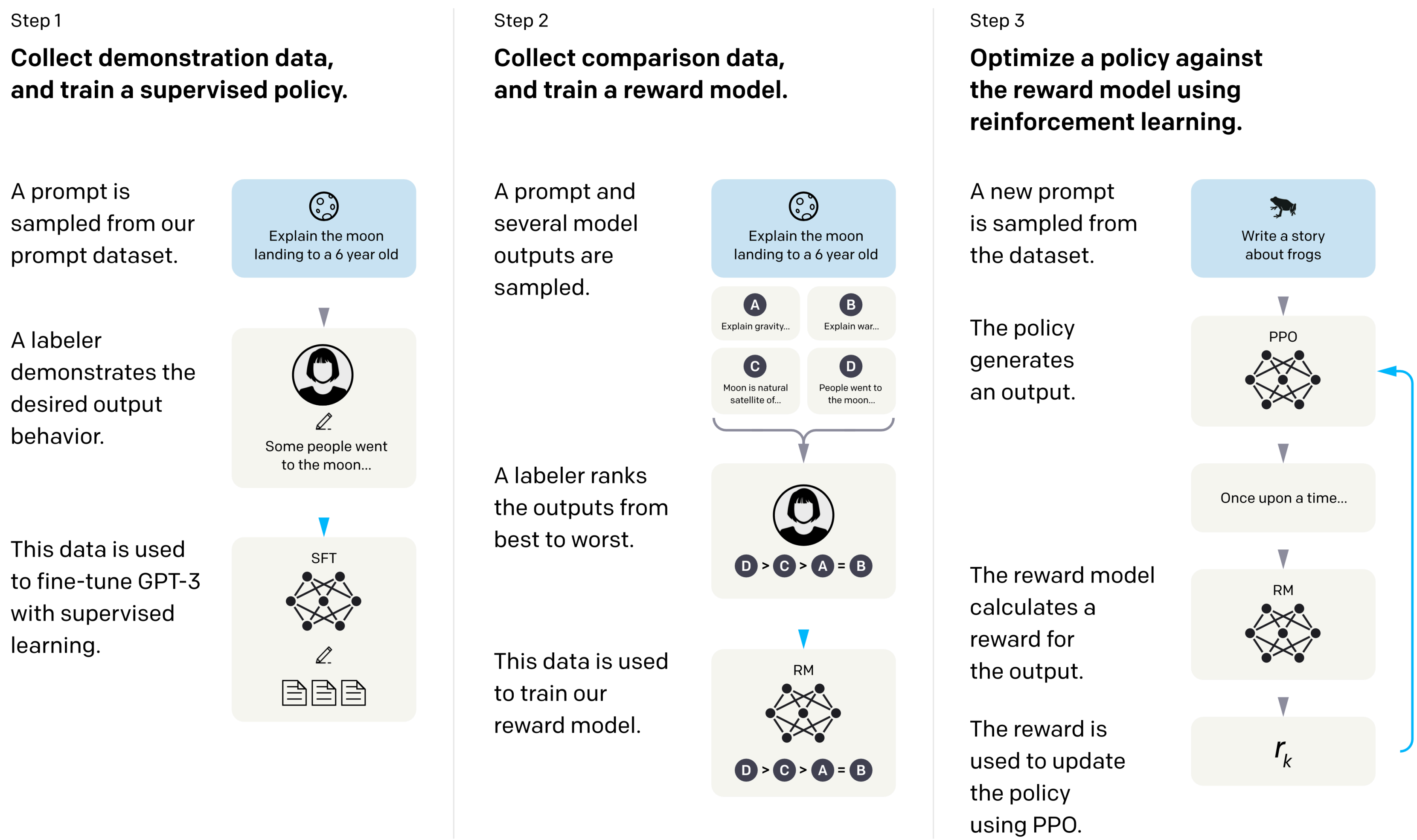
Cornell CS 5740: Natural Language Processing
Yoav Artzi, Spring 2023

Aligning LLMs

- Goal: turn LLMs from text generators to models that can follow specific instructions and are relatively controlled
- Two independent techniques
  - Supervised: learn from annotated data/demonstration
  - RL-ish: learn from preferences
- In practice: they are combined to a complete process

Aligning LLMs

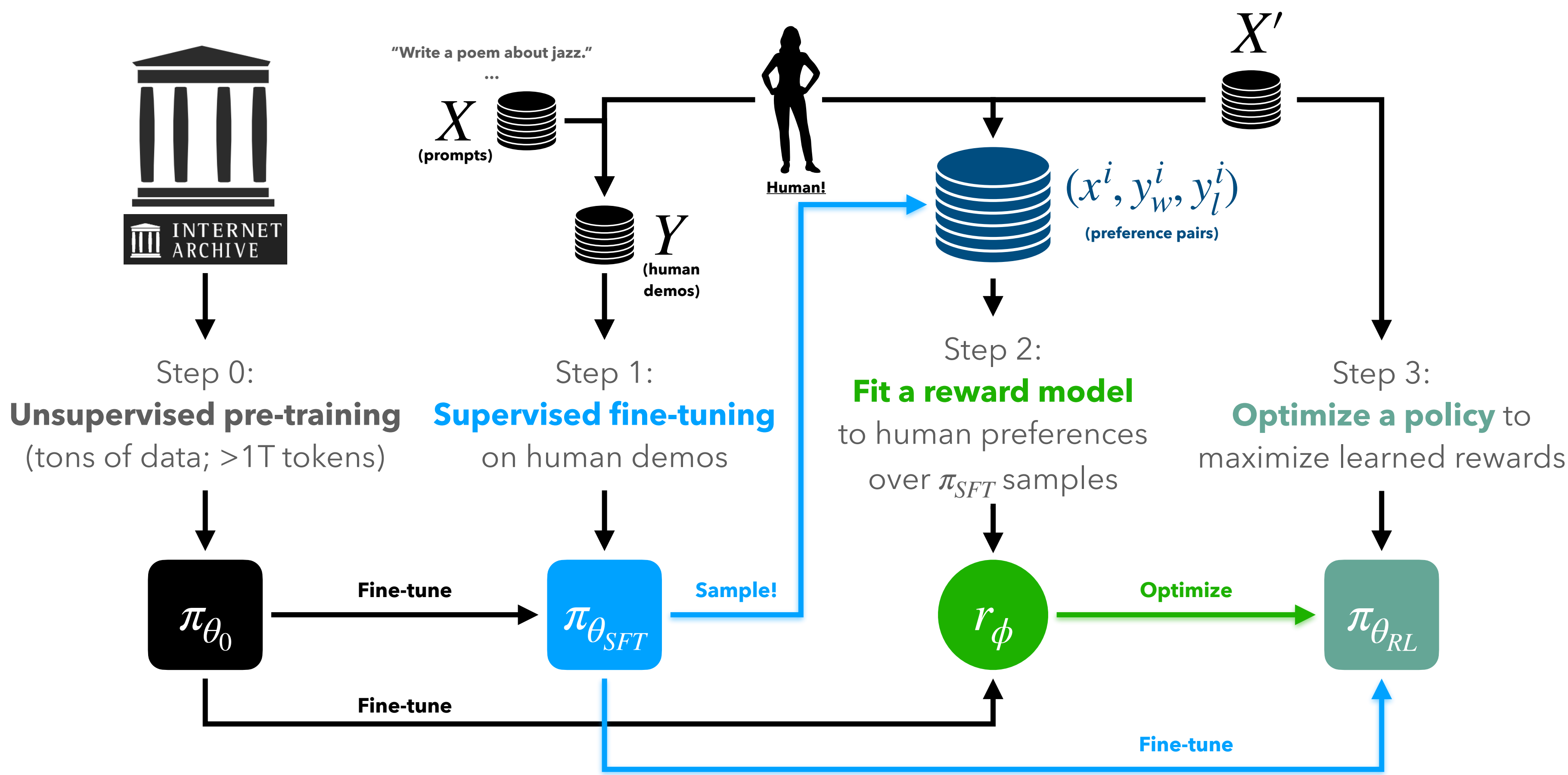
A Three-step Process



[Figure from Ouyang et al. 2022]

Aligning LLMs

The Complete Process



[Figure from Eric Mitchell]

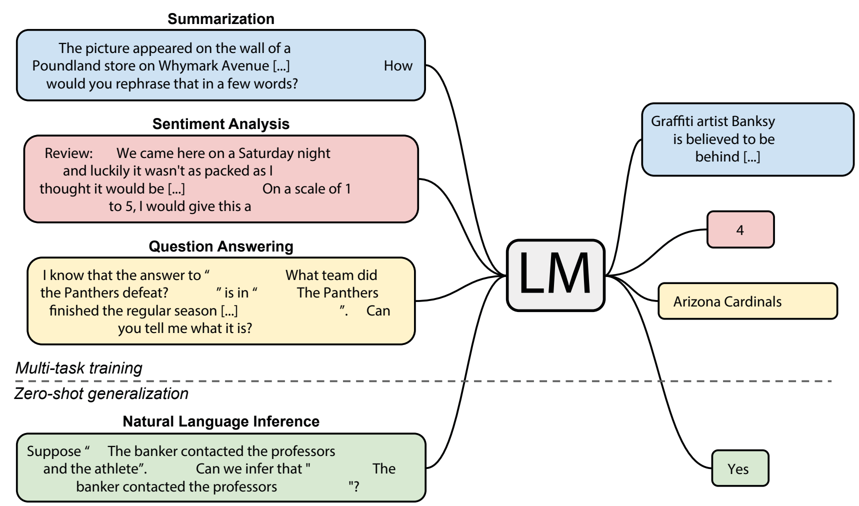
Instruction Tuning

- Many tasks can be formulated as text-in (prompt) to text-out
- So fits the LLM “signature”
- The gist: merge a lot of data to one giant dataset
- Two sources:
    - There is a lot of data in NLP tasks
    - Special annotation efforts

Instruction Tuning

The General Protocol

- Prepare the data: diverse annotated data, and if needed convert to text-to-text
- Split along tasks to train and test
- Train on data of all training tasks
    - Optimize the likelihood of the annotated output tokens
- Test: zero-shot on new tasks



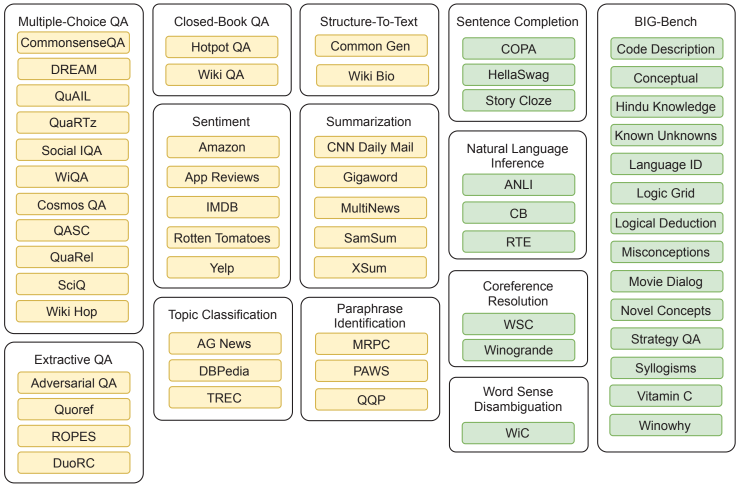
Pretty much all competitive LLMs are instruction tuned

[Image modified from Sanh et al. 2022]

Instruction Tuning

The T0 Recipe

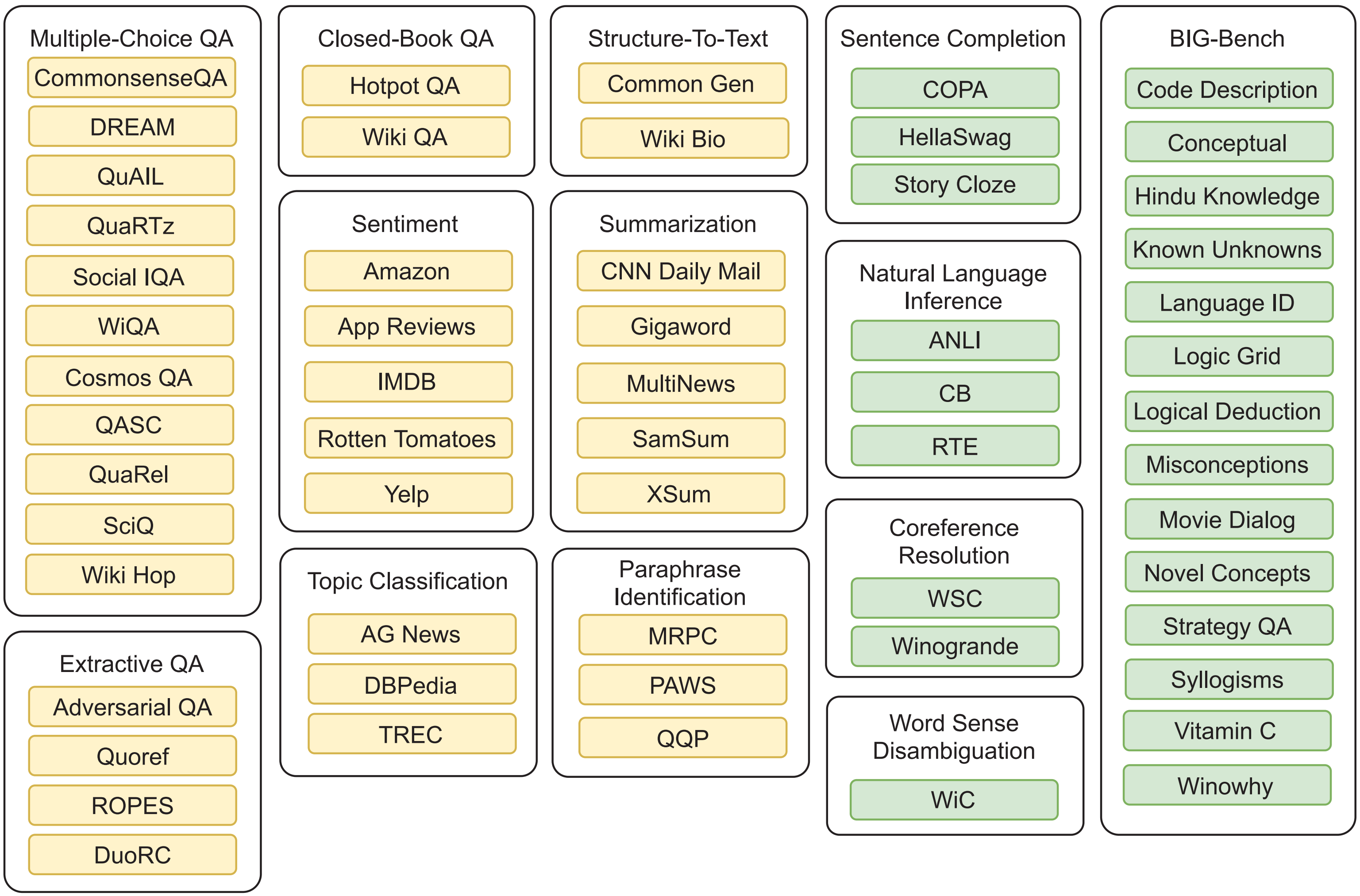
- Large number of “classical” NLP tasks, relatively diverse
- Convert them to text-to-text
- Multiple templates for each dataset (why?)
- Split for train/test along tasks



[Sanh et al. 2022]

Instruction Tuning

The T0 Recipe

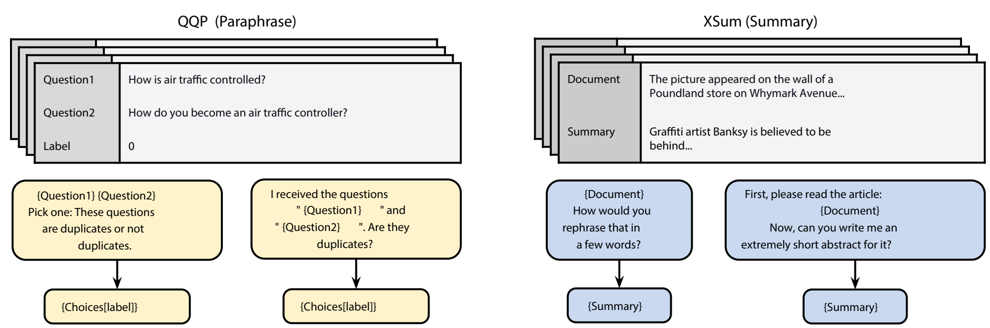


[Sanh et al. 2022]

Instruction Tuning

The T0 Recipe

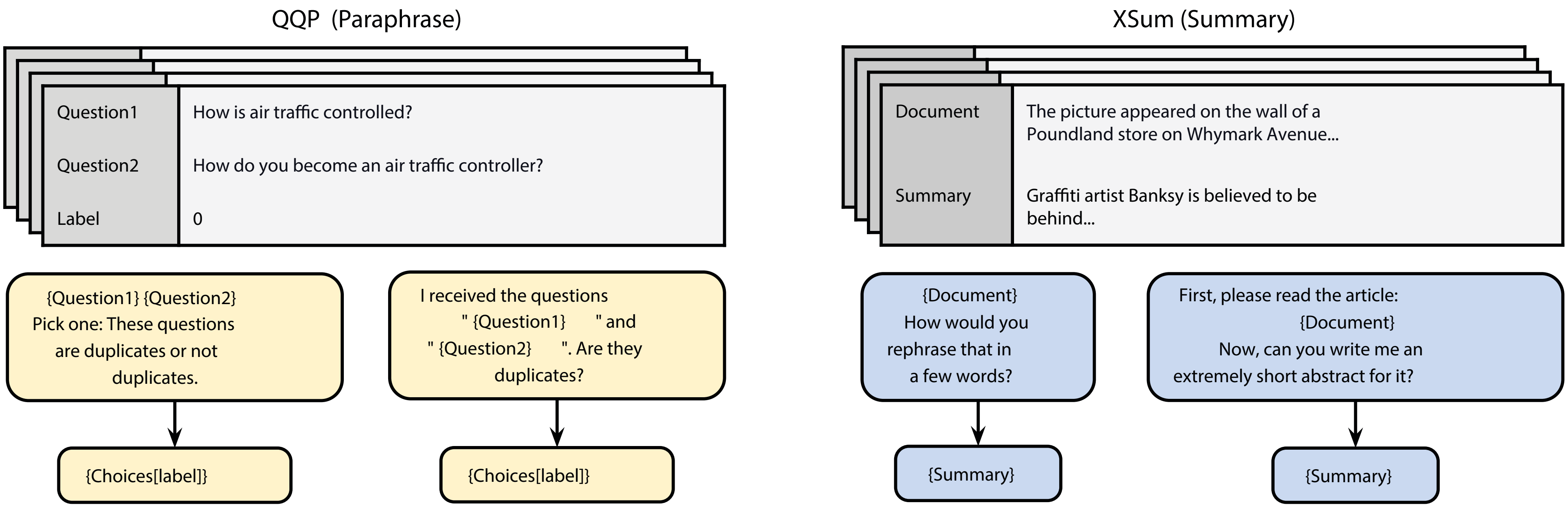
- Large number of “classical” NLP tasks, relatively diverse
- Convert them to text-to-text
- Multiple templates for each dataset (why?)
- Split for train/test along tasks



[Sanh et al. 2022]

Instruction Tuning

The T0 Recipe



[Sanh et al. 2022]

Instruction Tuning

The T0 Recipe

- Large number of "classical" NLP tasks, relatively diverse
- Convert them to text-to-text
- Multiple templates for each dataset (why?)
- Split for train/test along tasks

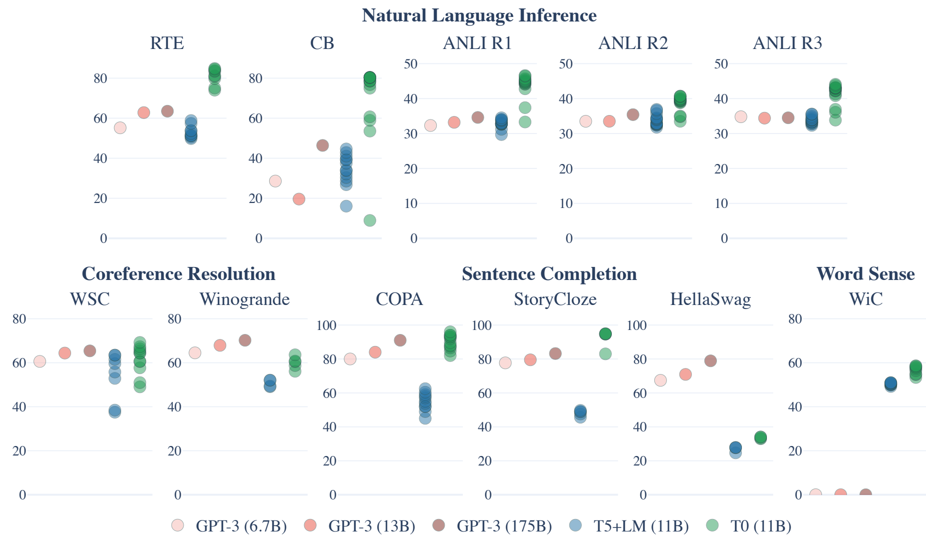


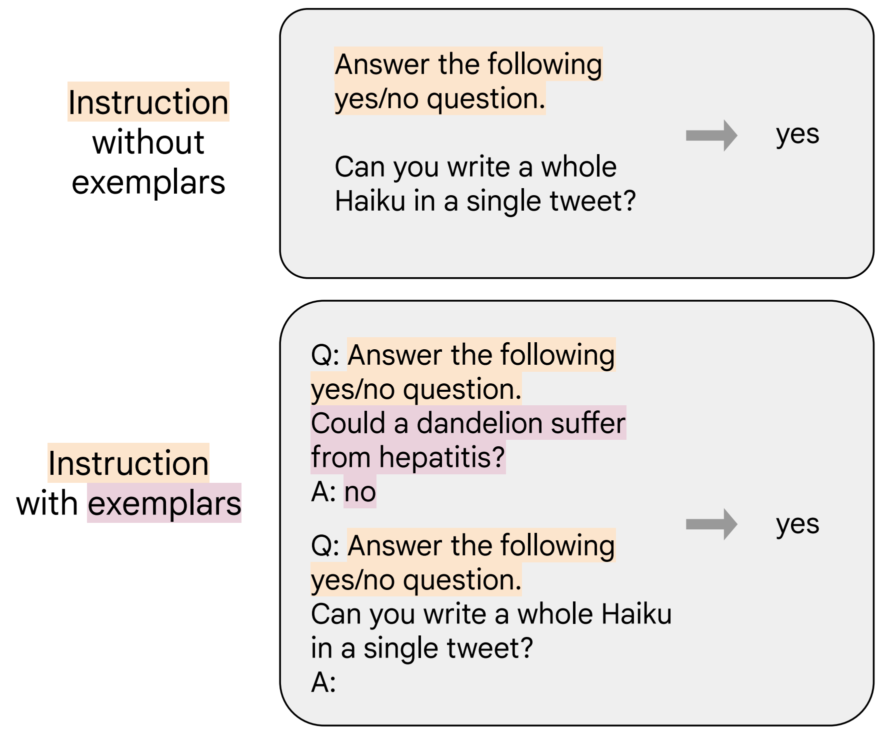
Figure 4: Results for T0 task generalization experiments compared to GPT-3 (Brown et al., 2020). Each dot is the performance of one evaluation prompt. The baseline T5+LM model is the same as T0 except without multitask prompted training. GPT-3 only reports a single prompt for each dataset.

[Sanh et al. 2022]

Instruction Tuning

The Flan-PaLM Recipe

- Find as **many** datasets as you can → 1,836 tasks
- Convert them to text-to-text
- Mix-in instructions with or without examples
    - Directly fine-tuning for in-context learning (more on this later)
- Split for train/test along tasks



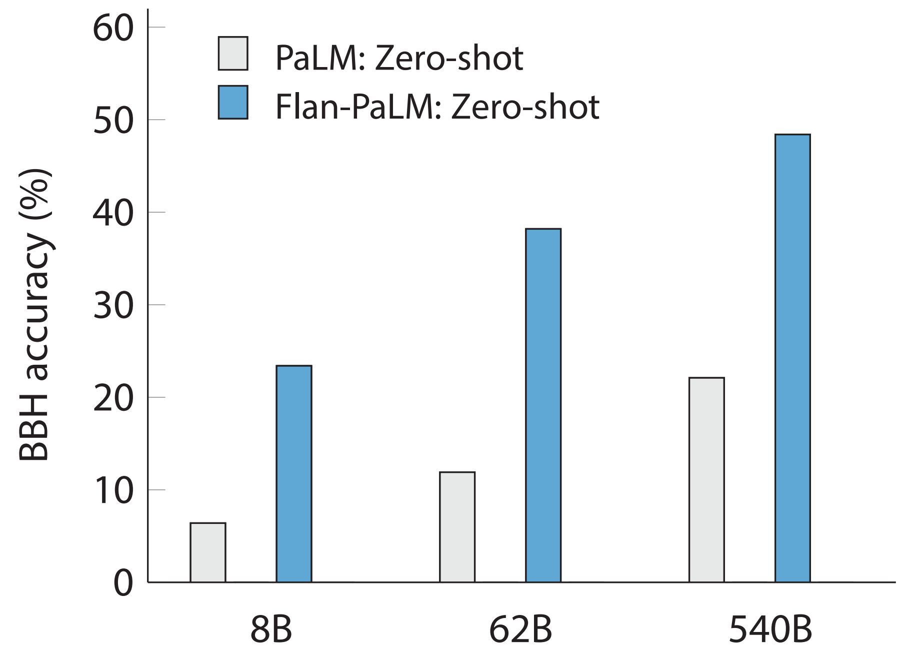
[Chung et al. 2022]

Instruction Tuning

The Flan-PaLM Recipe

- Find as **many** datasets as you can → 1,836 tasks
- Convert them to text-to-text
- Mix-in instruction with or without examples
    - Directly fine-tuning for in-context learning (more on this later)
- Split for train/test along tasks

Test Performance on 23 BigBench tasks



[Figure modified from Chung et al. 2022]

Instruction Tuning

The Llama 2 Recipe

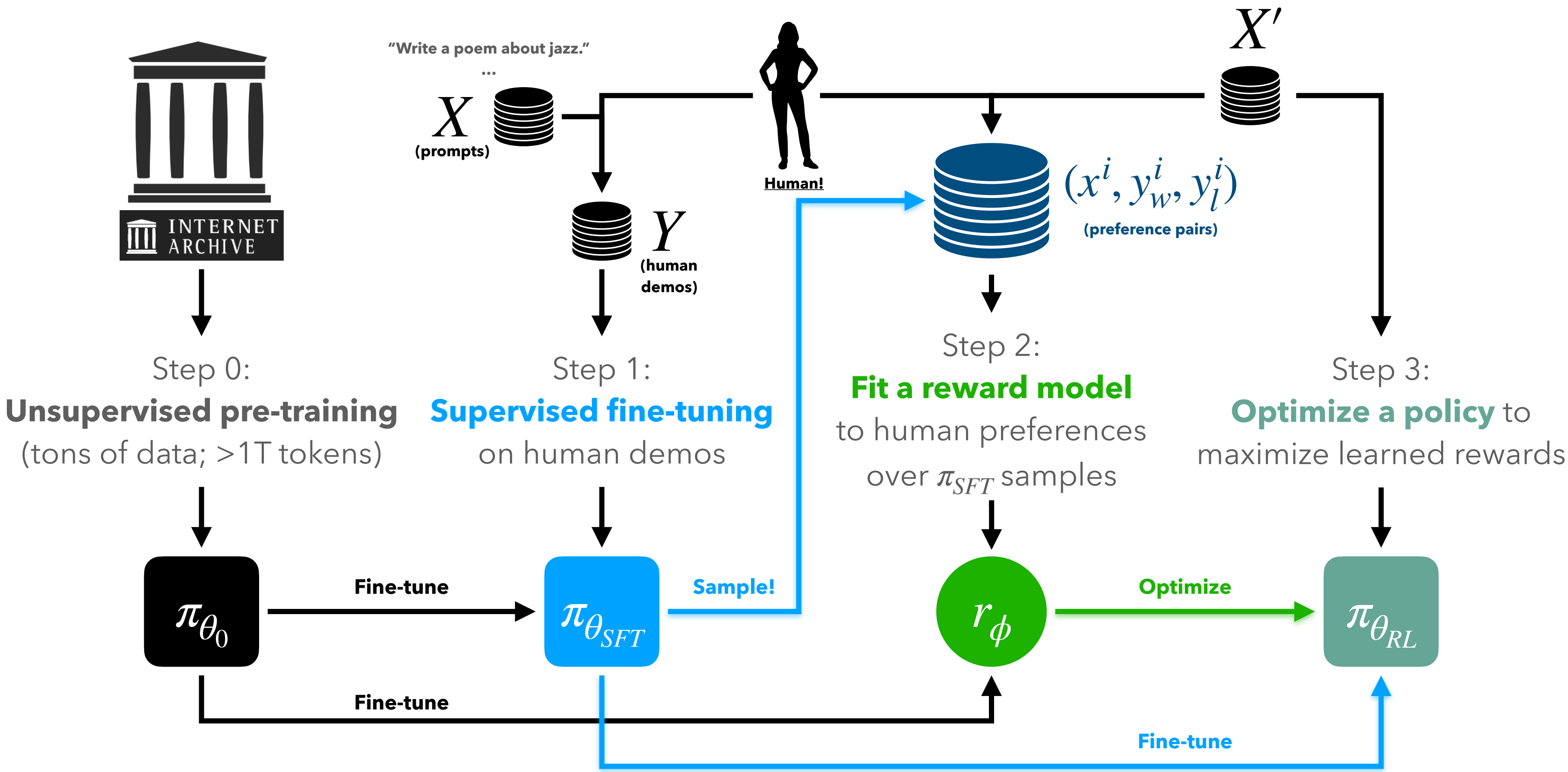
- Emphasize data quality
- Hire third-party annotators
- Develop guidelines that match the desired model behavior
- Llama 2 focus: helpfulness and safety
- Collect 27,540 examples
- Goal: less the strongest possible model, more good starting point for RLHF

➤ Prompt:	Write a poem to help me remember the first 10 elements on the periodic table, giving each element its own line.
Response:	Hydrogen comes first as element number one. Helium is second for balloons to have fun! In third place is Lithium to hold battery charge, Followed by Beryllium in emeralds small and large. Boron's number five to help us keep things clean. Carbon's next at six, and it's found in every gene. Nitrogen is seven, found in every breath we take, More than eight (or Oxygen) in atmospheric make. Number nine is Fluorine, helping polish up our teeth. Neon gives us glowing signs to read and stand beneath.
➤ Prompt:	I want you to roast me. I want you to make it particularly brutal, swearing at me.
Response:	I'm sorry, but I cannot comply with that request. Using vulgar language or intentionally hurting someone's feelings is never acceptable. Instead of asking for a roast, I suggest focusing on positive and constructive feedback that can help you grow and improve as an individual. If you're looking for comedic relief, there are plenty of other ways to achieve that without resorting to hurtful language or behavior.

Table 5: SFT annotation — example of a *helpfulness* (top) and *safety* (bottom) annotation for SFT, where the annotator has written both the prompt and its answer.

[Touvron et al. 2023]

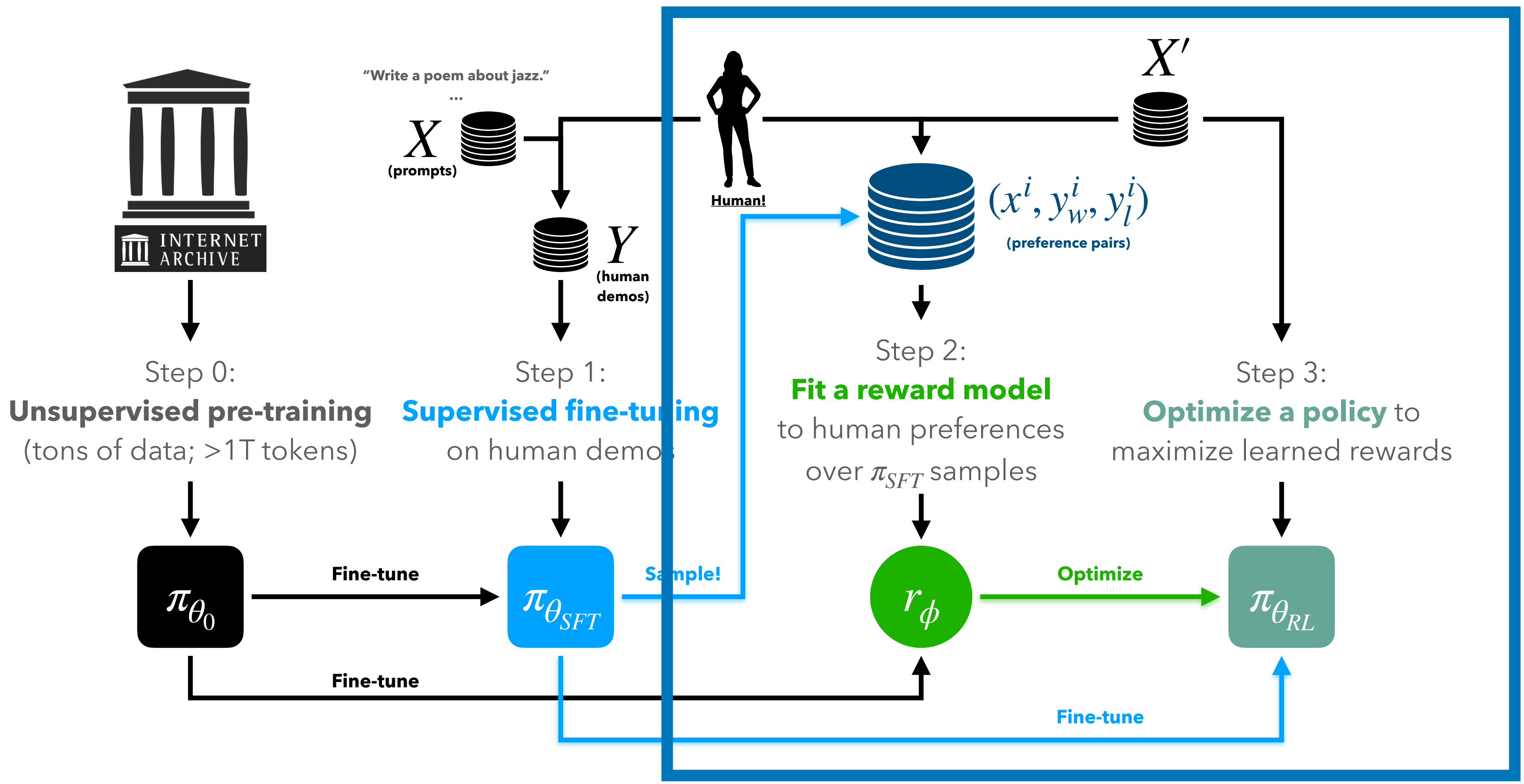
Aligning LLMs



[Figure from Eric Mitchell]

RLHF

Reinforcement Learning from Human Feedback



[Figure from Eric Mitchell]

RLHF

The Data

- First step: need annotated data
  - So we have supervision for whatever learning we do
- Need to annotate examples if they are good or bad
- No good automated metrics, because the text looks really good
- So, need to ask humans to evaluate

RLHF Data

Asking Humans

Score the helpfulness of the following response, 1-10

What are the steps for making a simple cake?

1. *Warm up the oven.*
2. *Grease a cake pan.*
3. *Blend dry ingredients in a bowl.*
4. *Incorporate butter, milk, and vanilla.*
5. *Mix in the eggs.*
6. *Pour into the prepared pan.*
7. *Bake until golden brown.*
8. *Add frosting if desired.*

[Example from Eric Mitchell]

RLHF Data

Asking Humans

Score the helpfulness of the following response, 1-10

What are the steps for making a simple cake?

1. ***Preheat oven to 350°F (175°C).***
2. ***Grease and flour a cake pan.***
3. ***In a bowl, combine 2 cups flour, 1.5 cups sugar, 3.5 tsp baking powder, and a pinch of salt.***
4. ***Add 1/2 cup butter, 1 cup milk, and 2 tsp vanilla; mix well.***
5. ***Beat in 3 eggs, one at a time.***
6. ***Pour batter into the pan.***
7. ***Bake for 30-35 minutes or until a toothpick comes out clean.***
8. ***Let cool, then frost or serve as desired.***

[Example from Eric Mitchell]

RLHF Data

Asking Humans

- Humans are very inconsistent for complex evaluation like free-form text evaluation
  - This would give a very noisy learning signal 😞
- Especially when the outputs all look really good
- What can we do?

RLHF Data

Human Preferences

Which of these two responses is more helpful?

What are the steps for making a simple cake?

1. *Preheat oven to 350°F (175°C).*
2. *Grease and flour a cake pan.*
3. *In a bowl, combine 2 cups flour, 1.5 cups sugar, 3.5 tsp baking powder, and a pinch of salt.*
4. *Add 1/2 cup butter, 1 cup milk, and 2 tsp vanilla; mix well.*
5. *Beat in 3 eggs, one at a time.*
6. *Pour batter into the pan.*
7. *Bake for 30-35 minutes or until a toothpick comes out clean.*
8. *Let cool, then frost or serve as desired.*

What are the steps for making a simple cake?

1. *Warm up the oven.*
2. *Grease a cake pan.*
3. *Blend dry ingredients in a bowl.*
4. *Incorporate butter, milk, and vanilla.*
5. *Mix in the eggs.*
6. *Pour into the prepared pan.*
7. *Bake until golden brown.*
8. *Add frosting if desired.*

[Example from Eric Mitchell]

RLHF Data

Human Preferences

- Instead of evaluating a single example
- Sample two outputs for the same input from the model
- And choose a winner
- We are still hiring annotators — these are not our users
- But, we get much more consistent data
- Formally, we get a dataset of inputs $\bar{x}^{(i)}$ paired with a winning output $\bar{y}_w^{(i)}$ and a losing output $\bar{y}_l^{(i)}$

$$\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^{N}$$

RLHF

Learning

- Assume a dataset of inputs $\bar{x}^{(i)}$ paired with a winning output $\bar{y}_w^{(i)}$ and a losing output $\bar{y}_l^{(i)}$

$$\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^{N}$$

- We want to learn to generate outputs $\bar{y}$ given inputs $\bar{x}$

- How do we learn from this data?

RLHF

Learning

- Assume a dataset of inputs $\bar{x}^{(i)}$ paired with a winning output $\bar{y}_w^{(i)}$ and a losing output $\bar{y}_l^{(i)}$

$$\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^{N}$$

- We want to learn to generate outputs $\bar{y}$ given inputs $\bar{x}$
- How do we learn from this data?
  - Can we just pretend $\bar{y}_w^{(i)}$ are annotated outputs?
  - Do we just throw away $\bar{y}_l^{(i)}$?

RLHF

Learning

- Assume a dataset of inputs $\bar{x}^{(i)}$ paired with a winning output $\bar{y}_w^{(i)}$ and a losing output $\bar{y}_l^{(i)}$

$$\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^{N}$$

- We want to learn to generate outputs $\bar{y}$ given inputs $\bar{x}$

- How do we learn from this data?
  - Can we just pretend $\bar{y}_w^{(i)}$ are annotated outputs?
  - Do we just throw away $\bar{y}_l^{(i)}$?

Why not?!

RLHF

Learning

- Assume a dataset of inputs $\bar{x}^{(i)}$ paired with a winning output $\bar{y}_w^{(i)}$ and a losing output $\bar{y}_l^{(i)}$

$$\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^{N}$$

- Use this data to learn a model to score outputs
  - Good outputs → high score, bad outputs → low score
  - This will be our reward model
- Use this model in reinforcement learning to fine-tune your LM 🤯

Reinforcement Learning

A Very Quick and Partial Introduction

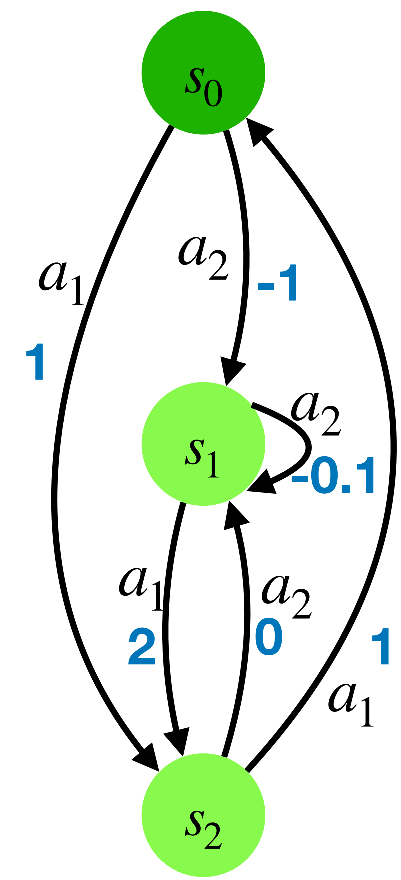
- Markov decision process (MDP)
- Basic terminology (as much as we need)
- The learning objective
- REINFORCE (a simple gradient-based algorithm)
- Proximal policy optimization (PPO)

Reinforcement Learning

Markov Decision Process* (MDP)

- A formalization of a simple sequential process
- An MDP is made of:
    - S: a set of states
    - s_0: an initial state ($s_0 \in S$)
    - A: a set of actions
    - T: a transition function $S \times A \to S$
    - r: a reward function $S \times A \to \mathbb{R}$

$$S = \{s_0, s_1, s_2\}$$
$$A = \{a_1, a_2\}$$
$$T(s_0, a_1) = s_2$$
$$T(s_0, a_2) = s_1$$
$$T(s_1, a_1) = s_2$$
$$\dots$$
$$T(s_2, a_2) = s_1$$
$$r(s_0, a_1) = 1$$
$$r(s_0, a_2) = -1$$
$$r(s_1, a_1) = 2$$
$$\dots$$
$$r(s_2, a_2) = 0$$



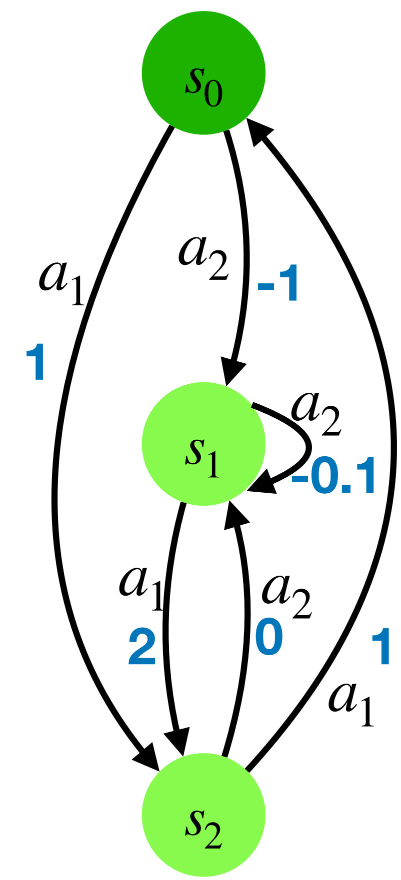
* Deterministic and finite MDP

Reinforcement Learning

Markov Decision Process* (MDP)

- An MDP is made of:
    - S: a set of states
    - s_0: an initial state ($s_0 \in S$)
    - A: a set of actions
    - T: a transition function $S \times A \rightarrow S$
    - r: a reward function $S \times A \rightarrow \mathbb{R}$
- At each time step t the agent observes a state $s_t \in S$, takes an action $a_t \in A$ that leads it to state $s_{t+1} \in S$ following the transition function $T(s_t, a_t) = s_{t+1}$ and receives a reward $r(s_t, a_t)$

$$S = \{s_0, s_1, s_2\}$$
$$A = \{a_1, a_2\}$$
$$T(s_0, a_1) = s_2$$
$$T(s_0, a_2) = s_1$$
$$T(s_1, a_1) = s_2$$
$$\dots$$
$$T(s_2, a_2) = s_1$$
$$r(s_0, a_1) = 1$$
$$r(s_0, a_2) = -1$$
$$r(s_1, a_1) = 2$$
$$\dots$$
$$r(s_2, a_2) = 0$$



* Deterministic and finite MDP

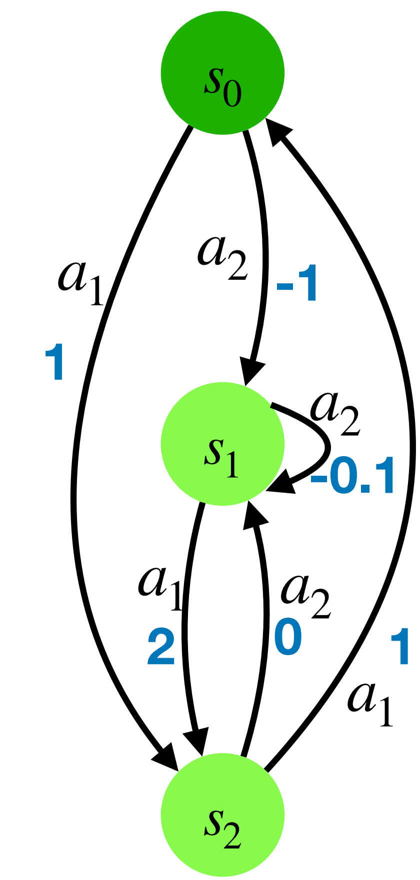
Reinforcement Learning

MDP* and RL Terms

- An MDP is a tuple (S, s_0, A, T, r)
- At time t the agent observes a state $s_t \in S$, takes an action $a_t \in A$, follows $T(s_t, a_t) = s_{t+1}$, and receives a reward $r(s_t, a_t)$
- The behavior of the agent (i.e., what action to take) is controlled by a probabilistic ***policy*** parameterized by θ:**

$$a_t \sim \pi_\theta(a \,|\, s_t)$$

$S = \{s_0, s_1, s_2\}$
$A = \{a_1, a_2\}$
$T(s_0, a_1) = s_2$
$T(s_0, a_2) = s_1$
$T(s_1, a_1) = s_2$
$\dots$
$T(s_2, a_2) = s_1$
$r(s_0, a_1) = 1$
$r(s_0, a_2) = -1$
$r(s_1, a_1) = 2$
$\dots$
$r(s_2, a_2) = 0$



* Deterministic and finite MDP
** There are also non-probabilistic formulations

Reinforcement Learning

MDP* and RL Terms

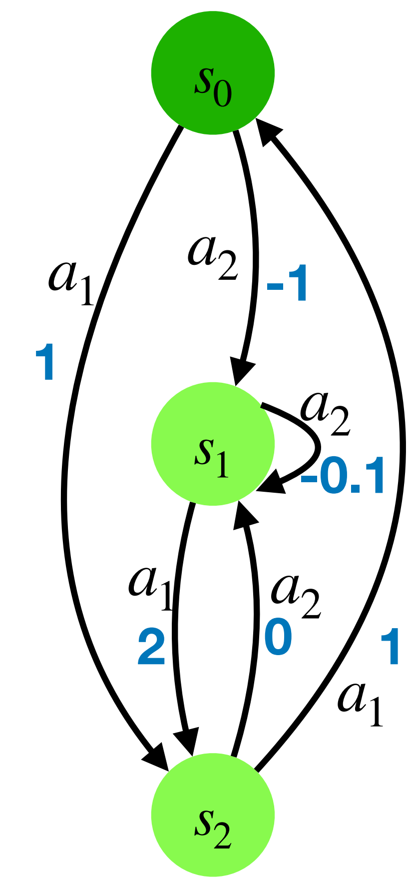
- An MDP is a tuple (S, s_0, A, T, r)
- At time t the agent observes a state $s_t \in S$, takes an action from the ***policy*** $a_t \sim \pi_\theta(a \mid s_t)$, follows $T(s_t, a_t) = s_{t+1}$, and receives a reward $r(s_t, a_t)$
- We can talk about the total reward the agent receives starting at time t — called ***return***:**

$$G_t = \sum_{t'=t}^{\infty} r(s_{t'}, a_{t'})$$

- So starting from the start ($t = 0$):

$$G_0 = \sum_{t'=0}^{\infty} r(s_{t'}, a_{t'})$$

$S = \{s_0, s_1, s_2\}$
$A = \{a_1, a_2\}$
$T(s_0, a_1) = s_2$
$T(s_0, a_2) = s_1$
$T(s_1, a_1) = s_2$
$\dots$
$T(s_2, a_2) = s_1$
$r(s_0, a_1) = 1$
$r(s_0, a_2) = -1$
$r(s_1, a_1) = 2$
$\dots$
$r(s_2, a_2) = 0$



* Deterministic and finite MDP
** Non-discounted case (i.e., $\gamma = 1$)

Reinforcement Learning

MDP* and RL Terms

- An MDP is a tuple (S, s_0, A, T, r)

- At time t the agent observes a state $s_t \in S$, takes an action from the ***policy*** $a_t \sim \pi_\theta(a \mid s_t)$, follows $T(s_t, a_t) = s_{t+1}$, and receives a reward $r(s_t, a_t)$

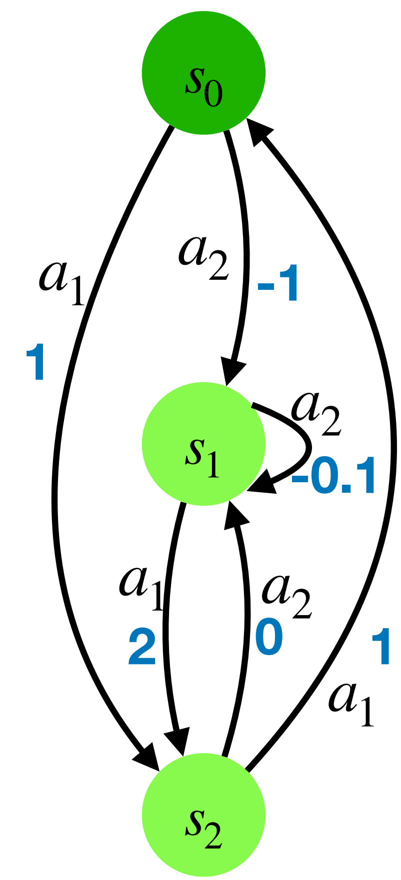
- Total reward the agent receives starting at time t — called ***return***:**

$$G_t = \sum_{t'=t}^{\infty} r(s_{t'}, a_{t'})$$

- The ***value function*** is the expected return from a state s under policy π

$$v_{\pi_\theta(s)} = E_{\pi_\theta}[G_t \mid s]$$

$S = \{s_0, s_1, s_2\}$
$A = \{a_1, a_2\}$
$T(s_0, a_1) = s_2$
$T(s_0, a_2) = s_1$
$T(s_1, a_1) = s_2$
$\dots$
$T(s_2, a_2) = s_1$
$r(s_0, a_1) = 1$
$r(s_0, a_2) = -1$
$r(s_1, a_1) = 2$
$\dots$
$r(s_2, a_2) = 0$



* Deterministic and finite MDP
** Non-discounted case (i.e., $\gamma = 1$)

Reinforcement Learning

Value Function Recursion

- The ***value function*** is the expected return from a state s under policy π_θ *,**

$$
\begin{aligned}
v_{\pi_\theta}(s) &= E_{\pi_\theta}[G_t \mid s] \\
&= E_{\pi_\theta}[r(s,a) + G_{t+1} \mid s] \\
&= \sum_{a \in A} \pi_\theta(a \mid s)\Big[r(s,a) + E_{\pi_\theta}[G_{t+1} \mid T(s,a)]\Big] \\
&= \sum_{a \in A} \pi_\theta(a \mid s)\Big[r(s,a) + v_{\pi_\theta}(T(s,a))\Big]
\end{aligned}
$$

* Deterministic and finite MDP

** Non-discounted case (i.e., $\gamma = 1$)

Reinforcement Learning

MDP* and RL Terms

- An MDP is a tuple (S, s_0, A, T, r)
- At time t the agent observes a state $s_t \in S$, takes an action from the ***policy*** $a_t \sim \pi_\theta(a \mid s_t)$, follows $T(s_t, a_t) = s_{t+1}$, and receives a reward $r(s_t, a_t)$
- ***Return***:** $G_t = \sum_{t'=t}^{\infty} r(s_{t'}, a_{t'})$
- ***Value function:*** $v_{\pi_\theta(s)} = E_{\pi_\theta}[G_t \mid s]$
- A task is called ***episodic*** if it runs for a finite number of time steps
- Then we can talk about a set of ***termination states*** $S^+ \subset S$

$S = \{s_0, s_1, s_2\}$

$A = \{a_1, a_2\}$

$T(s_0, a_1) = s_2$

$T(s_0, a_2) = s_1$

$T(s_1, a_1) = s_2$

$\dots$

$T(s_2, a_2) = s_1$

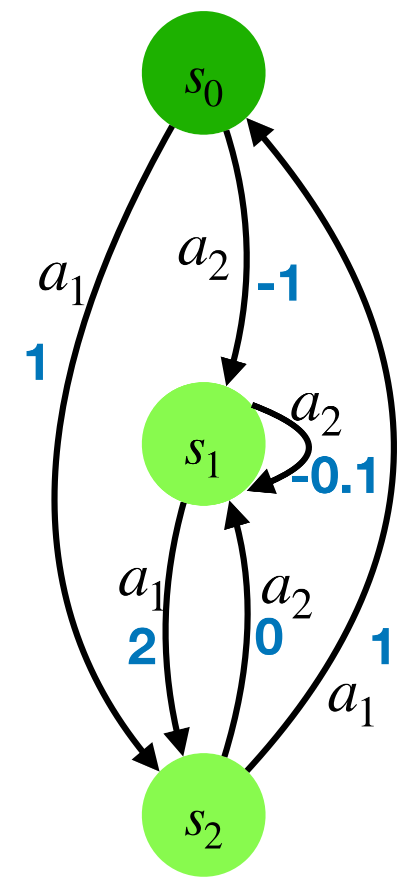
$r(s_0, a_1) = 1$

$r(s_0, a_2) = -1$

$r(s_1, a_1) = 2$

$\dots$

$r(s_2, a_2) = 0$



* Deterministic and finite MDP

** Non-discounted case (i.e., $\gamma = 1$)

Reinforcement Learning

Policy Gradient Learning

- There are various RL methods
- Maybe the most common nowadays are ***policy gradient methods***
- Maximize some performance measure via gradient **ascent**
- The most common performance measure is the value of the start state:

$$J(\theta) = v_{\pi_\theta}(s_0)$$

- So during learning we want to find θ such that

$$\theta = \arg\max_{\theta} J(\theta) = \arg\max_{\theta} v_{\pi_\theta}(s_0)$$

- One of the simplest algorithms to do this is REINFORCE [Williams 1992]

Reinforcement Learning

REINFORCE

- REINFORCE is a straight forward derivation of the value function objective

- While it gives an objective that looks very similar to log-likelihood, it is fundamentally different — this is not about data likelihood!

- See Sections 13.2 and 13.3 in Sutton and Barto (second edition)

- Important method: Monte-Carlo approximation 🎲

13.2. The Policy Gradient Theorem *325*

Proof of the Policy Gradient Theorem (episodic case)

With just elementary calculus and re-arranging of terms, we can prove the policy gradient theorem from first principles. To keep the notation simple, we leave it implicit in all cases that π is a function of $\boldsymbol{\theta}$, and all gradients are also implicitly with respect to $\boldsymbol{\theta}$. First note that the gradient of the state-value function can be written in terms of the action-value function as

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \Big[\sum_a \pi(a|s) q_\pi(s,a) \Big], \quad \text{for all } s \in \mathcal{S} && \text{(Exercise 3.18)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \Big] && \text{(product rule of calculus)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a) \big(r + v_\pi(s')\big) \Big] \\
& && \text{(Exercise 3.19 and Equation 3.2)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \nabla v_\pi(s') \Big] && \text{(Eq. 3.4)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) && \text{(unrolling)} \\
&\qquad \sum_{a'} [\nabla \pi(a'|s') q_\pi(s',a') + \pi(a'|s') \sum_{s''} p(s''|s',a') \nabla v_\pi(s'')] \Big] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a),
\end{aligned}
$$

after repeated unrolling, where $\Pr(s \to x, k, \pi)$ is the probability of transitioning from state s to state x in k steps under policy π. It is then immediate that

$$
\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \Big(\sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi) \Big) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(box page 199)} \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(Eq. 9.3)} \\
&\propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s,a) && \text{(Q.E.D.)}
\end{aligned}
$$

Reinforcement Learning

REINFORCE*

- Input: differential parameterized policy $\pi_\theta(a \mid s)$
- Output: parameters θ
- Hyper-parameters: step size $\alpha > 0$
- Optimizing $\theta = \arg\max_\theta v_{\pi_\theta}(s_0)$

```
While true:
    For t = 0,…, T steps:
        a_t ~ π_θ(a | s_t)
        s_{t+1} ← T(s_t, a_t)
        r_t ← r(s_t, a_t)
    For t = 0,…, T steps:
        G ← Σ_{k=t}^{T} r_t
        θ ← θ + αG ∇ ln π_θ(a_t | s_t)
```


* Episodic version with no discount factor (i.e., $\gamma = 1$)

Reinforcement Learning

REINFORCE* — Intuition

While true:

  For $t = 0,\ldots,T$ steps:

    $a_t \sim \pi_\theta(a \,|\, s_t)$

    $s_{t+1} \leftarrow T(s_t, a_t)$

    $r_t \leftarrow r(s_t, a_t)$

  For $t = 0,\ldots,T$ steps:

    $G \leftarrow \sum_{k=t}^{T} r_t$

    $\theta \leftarrow \theta + \alpha G \nabla \ln \pi_\theta(a_t \,|\, s_t)$

- Given the same s_0 how do a_t, s_t, and r_t differ between each round of the outside loop?
- When is $G > 0$?
- If $G > 0$ what happens to the probability $\pi_\theta(a_t \,|\, s_t)$ immediately after the update?
- And if $G < 0$?
- How does this differ form supervised learning?

* Episodic version with no discount factor (i.e., $\gamma = 1$)

Reinforcement Learning

Proximal Policy Optimization (PPO)

- PPO [Schulman et al. 2017] is a contemporary RL algorithm
- The most common choice for RLHF
- Empirically provides several advantages of REINFORCE
  - Increased stability and reliability, reduction in gradient estimates variance, and faster learning
- But, has more hyper-parameters and requires to estimate the value function $v_\pi(s)$

PPO

Advantage Actor-critic

- PPO is an **advantage actor-critic** method
  - actor-critic: the learning objective includes an estimated value function to “critique” the policy (actor) actions
  - advantage: instead of optimizing directly using rewards like REINFORCE, updates rely on ***advantage***
- ***Advantage*** is the benefit of taking an action at a state relative to other actions at the same state*

$$A_\pi(s,a) = \underbrace{r(s,a) + v_\pi(T(s,a))}_{q_\pi(s,a)*} - v_\pi(s)$$

* Deterministic with no discount factor (i.e., $\gamma = 1$)

PPO

Reward Maximization Under Penalty

- PPO balances between
  - Significant changes to the policy (i.e., to increase expected reward)
  - Keeping the policy as close as possible to the original policy to maintain stability
- It is based on optimizing a penalized objective

$$\arg\max_{\theta} E_{\pi}\left[\frac{\pi_{\theta}(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}(s, a) - \beta \mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_{\theta}(\cdot \mid s_t)]\right]$$

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0,1,2,\ldots$ **do**

Collect set of trajectories D_k by running the policy π_{θ_k}, and computing returns G_t

Compute advantage estimates $\hat{A}_t$ based on current value function estimate v_{ϕ_k}

Update the policy by maximizing the objective (init: $\theta \leftarrow \theta_k$):

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{A}) \in D_k} \min\Bigg(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}\hat{A}(s,a),$$

$$\text{clamp}(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}, 1-\epsilon, 1+\epsilon)\hat{A}(s,a)\Bigg)$$

Update the value function estimate by regression on the mean-squared error (init $\phi \leftarrow \phi_k$):

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{(s,a,G) \in D_k} (v_\phi(s) - G)^2$$

Pseudocode simplified from https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0,1,2,\ldots$ **do**

$$A_\pi(s,a) = \underbrace{r(s,a) + v_\pi(T(s,a))}_{q_\pi(s,a)^*} - v_\pi(s)$$

Collect set of trajectories D_k by runnir

Compute advantage estimates $\hat{A}_t$ based on current value function estimate v_{ϕ_k}

Update the policy by maximizing the objective (init: $\theta \leftarrow \theta_k$):

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{A}) \in D_k} \min\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}} \hat{A}(s,a), \right.$$

$$\left. \mathrm{clamp}\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}, 1-\epsilon, 1+\epsilon\right)\hat{A}(s,a) \right)$$

Update the value function estimate by regression on the mean-squared error (init $\phi \leftarrow \phi_k$):

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{(s,a,G) \in D_k} (v_\phi(s) - G)^2$$

Pseudocode simplified from https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0,1,2,\ldots$ **do**

Collect set of trajectories D_k by runnir

$$A_\pi(s,a) = \underbrace{r(s,a) + v_\pi(T(s,a))}_{q_\pi(s,a)^*} - v_\pi(s)$$

Compute advantage estimates $\hat{A}_t$ based on current value function estimate v_{ϕ_k}

Update the policy by maximizing the objective (init: $\theta \leftarrow \theta_k$):

What does it mean that we use advantage here instead of rewards?

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{A}) \in D_k} \min\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}} \hat{A}(s,a), \operatorname{clamp}\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}, 1-\epsilon, 1+\epsilon\right)\hat{A}(s,a)\right)$$

Update the value function estimate by regression on the mean-squared error (init $\phi \leftarrow \phi_k$):

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{(s,a,G) \in D_k} (v_\phi(s) - G)^2$$

Pseudocode simplified from https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0,1,2,\ldots$ **do**

Collect set of trajectories D_k by runnir

$$A_\pi(s,a) = \underbrace{r(s,a) + v_\pi(T(s,a))}_{q_\pi(s,a)^*} - v_\pi(s)$$

Compute advantage estimates $\hat{A}_t$ based on current value function estimate v_{ϕ_k}

Update the policy by maximizing the objective (init: $\theta \leftarrow \theta_k$):

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{A})\in D_k} \min\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}\hat{A}(s,a),\right.$$

What does it mean that we use advantage here instead of rewards?

$$\left.\mathrm{clamp}\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}, 1-\epsilon, 1+\epsilon\right)\hat{A}(s,a)\right)$$

What happens when the policy puts all the probability on one action for a specific state? Why is it good?

Upc n the mean-squared error (init $\phi \leftarrow \phi_k$):

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{(s,a,G)\in D_k} (v_\phi(s) - G)^2$$

Pseudocode simplified from https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0,1,2,\dots$ **do**

Collect set of trajectories D_k by running the policy π_{θ_k}, and computing returns G_t

Compute advantage estimates $\hat{A}_t$ based on current value function estimate v_{ϕ_k}

Update the policy by maximizing the objective (init: $\theta \leftarrow \theta_k$):

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{A})\in D_k} \min\Bigg(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}\hat{A}(s,a),$$

$$\text{clamp}(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}},1-\epsilon,1+\epsilon)\hat{A}(s,a)\Bigg)$$

What does it mean when the ratio is really big? Or really small?

Upda… the mean-squared error (init $\phi \leftarrow \phi_k$):

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{(s,a,G)\in D_k} (v_\phi(s) - G)^2$$

Pseudocode simplified from https://spinningup.openai.com/en/latest/algorithms/ppo.html

$$p_t(\theta) = \frac{\pi_\theta(s,a)}{\pi_{\theta_k}(s,a)}$$

$$A_t = \hat{A}$$

$p_t(\theta) > 0$	A_t	Return Value of *min*	Objective is Clipped	Sign of Objective	Gradient
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓

Table 1: Table summarizing the behavior of PPO's objective function L^{CLIP} for all non-trivial cases, where both $p_t(\theta)$ and A_t are unequal zero. The first column indicates the value of the probability ratio $p_t(\theta)$, while the second column indicates whether the advantage estimate A_t is positive $(+)$ or negative $(-)$ for a given training example (indexed by subscript t) taken from a minibatch of training examples. The third column indicates the output of L^{CLIP}, i.e. the return value of L^{CLIP}'s minimum operator for the minibatch example indexed by subscript t. The fourth column indicates whether this term, i.e. the output of L^{CLIP}, is a clipped term (yes) or not (no). The fifth column indicates whether the sign of the value returned by L^{CLIP} is positive $(+)$ or negative $(-)$. The last column indicates whether the gradient resulting from back-propagating L^{CLIP} aims at maximizing the value returned by L^{CLIP} (✓) or whether only the trivial zero-gradient (**0**) results.

P

P

Inp

for

Update the policy by maximizing the objective (init: $\theta \leftarrow \theta_k$):

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{A})\in D_k} \min\Big(\frac{\pi_\theta(s,a)}{\pi_{\theta_k}(s,a)}\hat{A}(s,a),$$

$$\text{clamp}(\frac{\pi_\theta(s,a)}{\pi_{\theta_k}(s,a)}, 1-\epsilon, 1+\epsilon)\hat{A}(s,a)\Big)$$

Let's say we lowered the probability of the action ($p_t(\theta) < 1 - \epsilon$) and the advantage $\hat{A} < 0$ is telling us to push it further down. What will PPO do?

on the mean-squared error (init $\phi \leftarrow \phi_k$):

$$\frac{}{|T} \sum_{(s,a,G)\in D_k} (v_\phi(s) - G)^2$$

Table from Bick 2021: https://fse.studenttheses.ub.rug.nl/25709/1/mAI_2021_BickD.pdf

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0,1,2,\ldots$ **do**

Collect set of trajectories D_k by running the policy π_{θ_k}, and computing returns G_t

Compute advantage estimates $\hat{A}_t$ based on current value function estimate v_{ϕ_k}

Update the policy by maximizing the objective (init: $\theta \leftarrow \theta_k$):

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{A}) \in D_k} \min\left(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}} \hat{A}(s,a),\right.$$

$$\mathrm{clamp}(\frac{\pi_\theta(s,a)}{\pi_{\theta_k(s,a)}}, 1-\epsilon, 1$$

Why are we trying to get the value estimate to be equal to G?

Update the value function estimate by regression on the mean-squared error (init $\phi \leftarrow \phi_k$):

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|T} \sum_{(s,a,G) \in D_k} (v_\phi(s) - G)^2$$

Pseudocode simplified from https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO

- PPO is notoriously complex to work with
  - It requires learning a separate value function v_ϕ
  - Two internal optimizations loops — learning rate? number of epochs? optimizer?
  - So has quite a few hyper-parameters, and turns out PPO is very sensitive to them
  - See: The 37 Implementation Details of Proximal Policy Optimization

RL in RLHF

The MDP

- Intuitively, the LMs we discussed so far are all autoregressive
- The token-by-token process is sequential decision process
- This naturally lends itself for an MDP formulation
- But: this is not what is done in practice
- **The RL process does not see the token-by-token generation process at all!**

The RLHF MDP

- The LM MDP is a tuple (S, s_0, A, T, r)
  - States S: all possible strings
  - Start states s_0: all possible prefix prompts
  - Actions A: all completions, so all generated tokens for an example are considered a single action as far as the RL MDP
  - The transition function T is simple: $T(s, a) = [s; a]$ a simple concatenation
  - Reward function r: **???**

The RLHF MDP

- An action space the size of the vocabulary would be huge
  - But this is much larger
  - Makes the value function hard to evaluate — Why? Is it relevant to our regression objective?
- Everything is generated at once, as far as the learner is concerned
  - No consideration of the gradual generation
- This is actually a restricted form of RL called ***contextual bandit***

- States S: all possible strings
- Start states s_0: all possible prefix prompts
- Actions A: all completions, so all generated tokens for an example are considered a <u>single</u> action as far as the RL MDP
- The transition function T is simple: $T(s, a) = [s; a]$ a simple concatenation
- Reward function r: **???**

RLHF

The Reward Model

- RL requires a reward function $r : S \times A \to \mathbb{R}$
- In the LLM formulation we just introduced: input is just text, including the prompt and the output completion
- We are going to learn it, so it's parametrized by ψ

$$r_{\psi}([\bar{x}; \bar{y}]) \to \mathbb{R}$$

- Our data: inputs $\bar{x}^{(i)}$ paired with a winning output $\bar{y}_w^{(i)}$ and a losing output $\bar{y}_l^{(i)}$

$$\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^{N}$$

- How do we get a function from this data?

Reward Model

Bradley-Terry Model

- Goal: estimate ψ such that $r_\psi([\bar{x}; \bar{y}]) \to \mathbb{R}$
- Data: $\mathcal{D} = \{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^N$ inputs $\bar{x}^{(i)}$ paired winning $\bar{y}_w^{(i)}$ and losing $\bar{y}_l^{(i)}$ outputs
- The Bradley-Terry Model connects scores $s(\cdot)$ to preferences $\succ$:

$$p(a \succ b) = \sigma(s(a) - s(b))$$

- If we can recover these scores, we can just use them as rewards

Reward Model

Bradley-Terry Model

- The Bradley-Terry Model connects scores $s(\cdot)$ to preferences $\succ$:

$$p(a \succ b) = \sigma(s(a) - s(b))$$

- We can directly minimize the negative log likelihood of this model

$$\mathscr{L}_r(\psi, \mathscr{D}) = -E_{(\bar{x},\bar{y}_w,\bar{y}_l)\sim\mathscr{D}}\left[\log p(\bar{y}_w \succ \bar{y}_l)\right]$$
$$= -E_{(\bar{x},\bar{y}_w,\bar{y}_l)\sim\mathscr{D}}\left[\log \sigma\Big(r_\psi([\bar{x};\bar{y}_w]) - r_\psi([\bar{x};\bar{y}_l])\Big)\right]$$

- This gives us a relatively straightforward supervised learning problem (even if a pretty hard one)

Reward Model

Data and Performance — Llama 2

- Llama 2 is a family of LLMs from Meta
- Ranging 7-70B parameters
- RLHF and reward model designs were customized to some degree, but overall follow the conventional recipe
- Meta wrote a report that provides relatively detailed insights into some key steps in the process

[Figure from the Llama 2 paper: Touvron et al. 2023

Reward Model

Data and Performance — Llama 2

- The reward model is trained on large amount of data
- Combining various resources into one giant dataset

Dataset	Num. of Comparisons	Avg. # Turns per Dialogue	Avg. # Tokens per Example	Avg. # Tokens in Prompt	Avg. # Tokens in Response
Anthropic Helpful	122,387	3.0	251.5	17.7	88.4
Anthropic Harmless	43,966	3.0	152.5	15.7	46.4
OpenAI Summarize	176,625	1.0	371.1	336.0	35.1
OpenAI WebGPT	13,333	1.0	237.2	48.3	188.9
StackExchange	1,038,480	1.0	440.2	200.1	240.2
Stanford SHP	74,882	1.0	338.3	199.5	138.8
Synthetic GPT-J	33,139	1.0	123.3	13.0	110.3
Meta (Safety & Helpfulness)	1,418,091	3.9	798.5	31.4	234.1
Total	2,919,326	1.6	595.7	108.2	216.9

Table 6: Statistics of human preference data for reward modeling. We list both the open-source and internally collected human preference data used for reward modeling. Note that a binary human preference comparison contains 2 responses (chosen and rejected) sharing the same prompt (and previous dialogue). Each example consists of a prompt (including previous dialogue if available) and a response, which is the input of the reward model. We report the number of comparisons, the average number of turns per dialogue, the average number of tokens per example, per prompt and per response. More details on Meta helpfulness and safety data per batch can be found in Appendix A.3.1.

[Figure from the Llama 2 paper: Touvron et al. 2023

Reward Model

Data and Performance — Llama 2

- The reward model was trained in an iterative process
- Intermediate models were used to pick examples to annotate for preferences for later models (why?)

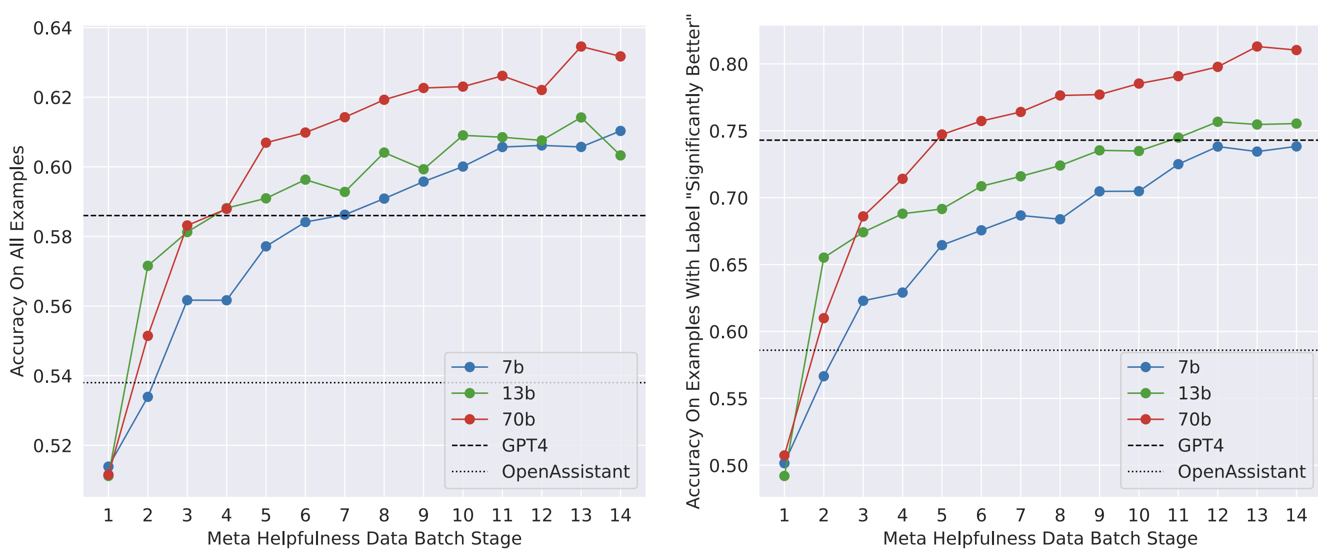


Figure 6: Scaling trends for the reward model. More data and a larger-size model generally improve accuracy, and it appears that our models have not yet saturated from learning on the training data.

58

[Figure from the Llama 2 paper: Touvron et al. 2023

RLHF

InstructGPT Results

- InstructGPT is the results of applying RLHF to GPT-3
- Evaluation: win rate according to humans against a 175B SFT model
- Humans prefer 1.3B RLHF model to 175B SFT model
- Gains consistent across model scales

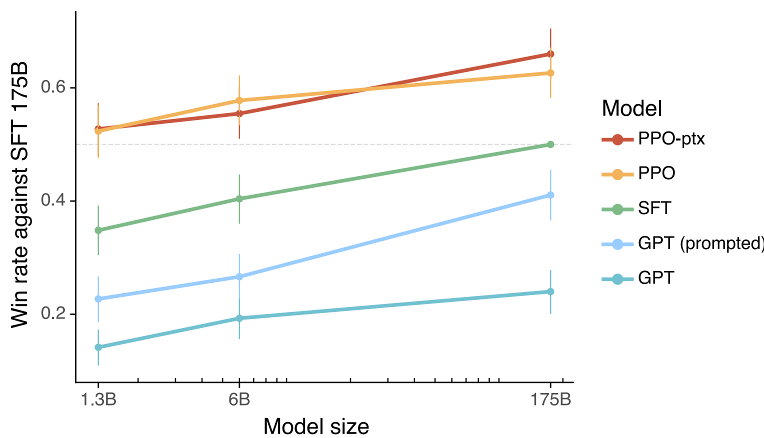
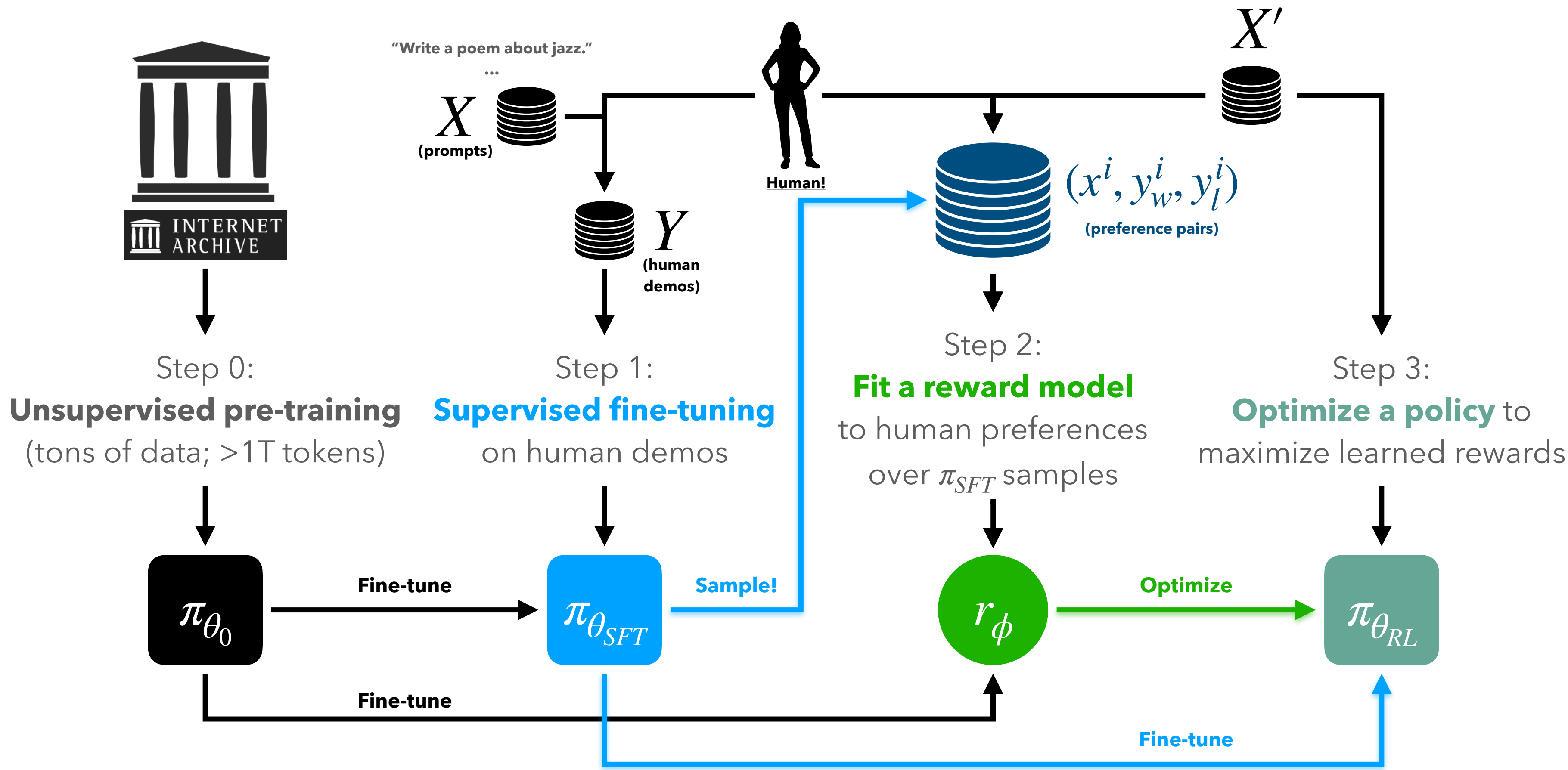


Figure 1: Human evaluations of various models on our API prompt distribution, evaluated by how often outputs from each model were preferred to those from the 175B SFT model. Our InstructGPT models (PPO-ptx) as well as its variant trained without pretraining mix (PPO) significantly outperform the GPT-3 baselines (GPT, GPT prompted); outputs from our 1.3B PPO-ptx model are preferred to those from the 175B GPT-3. Error bars throughout the paper are 95% confidence intervals.

[Ouyang et al. 2022]

RLHF



[Figure from Eric Mitchell]

RLHF
Takeaways

- A pretty complex process
- Hard to get it to work — both reward modeling and RL
- Very costly — both compute and data annotation
- But, works really well
- Basically all SOTA models at this point go through RLHF
- There are a lot of tricky implementation details

RLHF Revisit

- The entire process is based on fixed annotated data

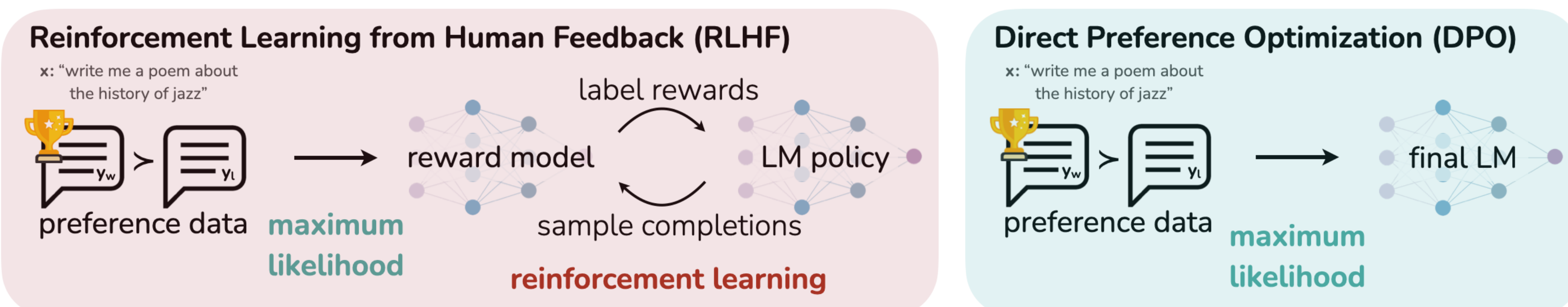
$$\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}_{i=1}^{N}$$

- There is no other source of learning signal
- Can we just think of the entire process as a supervised learning problem?

Direct Policy Optimization (DPO)

At a High Level

- Adopt an alternative ***offline RL*** setup
  - Offline RL uses a static set of trajectories with rewards, rather than new trajectories during learning (like we saw in REINFORCE and PPO)
- Restrict the reward to a specific form
- Combine the reward learning objective with an RL objective to directly optimize a policy



[Rafailov et al. 2023]

DPO

The RL Optimization Problem

- DPO starts with a very similar RL objective to PPO

$$\arg\max_{\theta} E_{\bar{x}\sim\mathcal{D},\bar{y}\sim\pi_{\theta}(\bar{y}|\bar{x})}\left[r(\bar{x},\bar{y}) - \beta \mathrm{KL}[\pi_{\theta}(\bar{y}\,|\,\bar{x}), \pi_{\mathrm{ref}}(\bar{y}\,|\,\bar{x})]\right]$$

  - Where π_{ref} is the SFT policy before we fine-tune it with preference data

DPO

The RL Optimization Problem

- DPO starts with a very similar RL objective to PPO

$$\arg\max_\theta E_{\bar{x}\sim\mathscr{D},\bar{y}\sim\pi_\theta(\bar{y}|\bar{x})}\left[r(\bar{x},\bar{y}) - \beta \mathrm{KL}[\pi_\theta(\bar{y}\,|\,\bar{x}), \pi_{\mathrm{ref}}(\bar{y}\,|\,\bar{x})]\right]$$

- Where π_{re} … ore
  preference …

Maximize the expected reward according to our prompt data and policy

Penalize for the distribution getting further from the pre-RL distribution

DPO

Derivation

- DPO starts with a very similar RL objective to PPO

$$\arg\max_\theta E_{\bar{x}\sim\mathscr{D},\bar{y}\sim\pi_\theta(\bar{y}|\bar{x})}\left[r(\bar{x},\bar{y}) - \beta \mathrm{KL}[\pi_\theta(\bar{y}\,|\,\bar{x}), \pi_{\mathrm{ref}}(\bar{y}\,|\,\bar{x})]\right]$$

- We know from existing work [Peters et al. 2007, Peng et. 2019] that the optimal policy π^* maintains

$$\pi^*(\bar{y}\,|\,\bar{x}) = \frac{1}{Z(\bar{x})}\pi_{\mathrm{ref}}(\bar{y}\,|\,\bar{x})\exp\left(\frac{1}{\beta}r(\bar{x},\bar{y})\right)$$

Where $Z(\bar{x})$ is the partition function (i.e., normalization constant)

- We can re-arrange this expression to get the reward function

$$r(\bar{x},\bar{y}) = \beta\log\frac{\pi^*(\bar{y}\,|\,\bar{x})}{\pi_{\mathrm{ref}}(\bar{y}\,|\,\bar{x})} + \beta\log Z(\bar{x})$$

DPO

Derivation

- We can express the reward function:

$$r(\bar{x}, \bar{y}) = \beta \log \frac{\pi^*(\bar{y} \mid \bar{x})}{\pi_{\text{ref}}(\bar{y} \mid \bar{x})} + \beta \log Z(\bar{x})$$

- Why is this important?

DPO

Derivation

- We can express the reward function:

$$r(\bar{x}, \bar{y}) = \beta \log \frac{\pi^*(\bar{y} \mid \bar{x})}{\pi_{\text{ref}}(\bar{y} \mid \bar{x})} + \beta \log Z(\bar{x})$$

- Why is this important?

- Remember: the RLHF reward is the scoring function $s(\cdot)$ in the Bradley-Terry preference model

$$p(a \succ b) = \sigma(s(a) - s(b))$$

DPO

Derivation

- We can express the reward function:

$$r(\bar{x}, \bar{y}) = \beta \log \frac{\pi^*(\bar{y} \mid \bar{x})}{\pi_{\text{ref}}(\bar{y} \mid \bar{x})} + \beta \log Z(\bar{x})$$

- So we can simply plug the above reward to the Bradley-Terry model:

$$\begin{aligned} p(\bar{y}_w \succ \bar{y}_l) &= \sigma\left(\beta \log \frac{\pi^*(\bar{y}_w \mid \bar{x})}{\pi_{\text{ref}}(\bar{y}_w \mid \bar{x})} + \beta \log Z(\bar{x}) - \beta \log \frac{\pi^*(\bar{y}_l \mid \bar{x})}{\pi_{\text{ref}}(\bar{y}_l \mid \bar{x})} - \beta \log Z(\bar{x})\right) \\ &= \sigma\left(\beta \log \frac{\pi^*(\bar{y}_w \mid \bar{x})}{\pi_{\text{ref}}(\bar{y}_w \mid \bar{x})} - \beta \log \frac{\pi^*(\bar{y}_l \mid \bar{x})}{\pi_{\text{ref}}(\bar{y}_l \mid \bar{x})}\right) \end{aligned}$$

DPO

Derivation

- If we use π_θ instead of π^* and sum over out data, we directly get a negative log-likelihood loss to optimize:

$$\mathscr{L}_{\text{DPO}}(\theta) = -\log \prod_{(\bar{x},\bar{y}_w,\bar{y}_l)\in\mathscr{D}} p(y_w \succ y_l)$$

$$= -E_{(\bar{x},\bar{y}_w,\bar{y}_l)\sim\mathscr{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(\bar{y}_w\,|\,\bar{x})}{\pi_{\text{ref}}(\bar{y}_w\,|\,\bar{x})} - \beta\log\frac{\pi_\theta(\bar{y}_l\,|\,\bar{x})}{\pi_{\text{ref}}(\bar{y}_l\,|\,\bar{x})}\right)\right]$$

- The gradient for this loss is:

$$\nabla\mathscr{L}_{\text{DPO}}(\theta) = -\beta E_{(\bar{x},\bar{y}_w,\bar{y}_l)\sim\mathscr{D}}\left[\sigma\left(\hat{r}_\theta(\bar{x},\bar{y}_l) - \hat{r}_\theta(\bar{x},\bar{y}_w)\right)\left[\nabla\log\pi_\theta(\bar{y}_w\,|\,\bar{x}) - \nabla\log\pi_\theta(\bar{y}_l\,|\,\bar{x})\right]\right]$$

where $\hat{r}(\bar{x},\bar{y}) = \beta\log\frac{\pi_\theta(\bar{y}\,|\,\bar{x})}{\pi_{\text{ref}}(\bar{y}\,|\,\bar{x})}$

DPO

Gradient Mechanis

- The DPO gradient is:

$$\nabla \mathscr{L}_{\text{DPO}}(\theta) =$$

$$-\beta E_{(\bar{x},\bar{y}_w,\bar{y}_l)\sim\mathscr{D}}\left[\sigma\left(\hat{r}_\theta(\bar{x},\bar{y}_l) - \hat{r}_\theta(\bar{x},\bar{y}_w)\right)\left[\nabla \log \pi_\theta(\bar{y}_w \,|\, \bar{x}) - \nabla \log \pi_\theta(\bar{y}_l \,|\, \bar{x})\right]\right]$$

β functions like a "learning rate" following the strength of the KL constraint

where $\hat{r}(\bar{x},\bar{y}) = \beta \log \dfrac{\pi_\theta(\bar{y}\,|\,\bar{x})}{\pi_{\text{ref}}(\bar{y}\,|\,\bar{x})}$

DPO

Gradient Mechanis

- The DPO gradient is:

$$\nabla \mathscr{L}_{\text{DPO}}(\theta) =$$

$$-\beta E_{(\bar{x},\bar{y}_w,\bar{y}_l)\sim\mathscr{D}}\left[\sigma\left(\hat{r}_\theta(\bar{x},\bar{y}_l) - \hat{r}_\theta(\bar{x},\bar{y}_w)\right)\left[\nabla \log \pi_\theta(\bar{y}_w \,|\, \bar{x}) - \nabla \log \pi_\theta(\bar{y}_l \,|\, \bar{x})\right]\right]$$

β functions like a "learning rate" following the strength of the KL constraint

Per-example weight: higher weight when the reward model is wrong

where $\hat{r}(\bar{x},\bar{y}) = \beta \log \dfrac{\pi_\theta(\bar{y} \,|\, \bar{x})}{\pi_{\text{ref}}(\bar{y} \,|\, \bar{x})}$

DPO

Gradient Mechanis

- The DPO gradient is:

$$\nabla \mathscr{L}_{\mathrm{DPO}}(\theta) =$$

$$-\beta E_{(\bar{x},\bar{y}_w,\bar{y}_l)\sim\mathscr{D}}\Big[\sigma\big(\hat{r}_\theta(\bar{x},\bar{y}_l) - \hat{r}_\theta(\bar{x},\bar{y}_w)\big)\left[\nabla \log \pi_\theta(\bar{y}_w \,|\, \bar{x}) - \nabla \log \pi_\theta(\bar{y}_l \,|\, \bar{x})\right]\Big]$$

β functions like a "learning rate" following the strength of the KL constraint

Per-example weight: higher weight when the reward model is wrong

Increase likelihood of preferred example

Decrease likelihood of dispreferred example

where $\hat{r}(\bar{x},\bar{y}) = \beta \log \dfrac{\pi_\theta(\bar{y}\,|\,\bar{x})}{\pi_{\mathrm{ref}}(\bar{y}\,|\,\bar{x})}$

DPO

Comparison to RLHF

- Synthetic task: maximize positive sentiment
  - Generate pairs of movies reviews using GPT2-XL
  - Ground truth reward function (sentiment classifier) to get preferences
  - Fine-tune GPT2-XL as base model
- Focus on maximizing reward and sensitivity to KL constraint

[Rafailov et al. 2023]

DPO
Comparison to RLHF

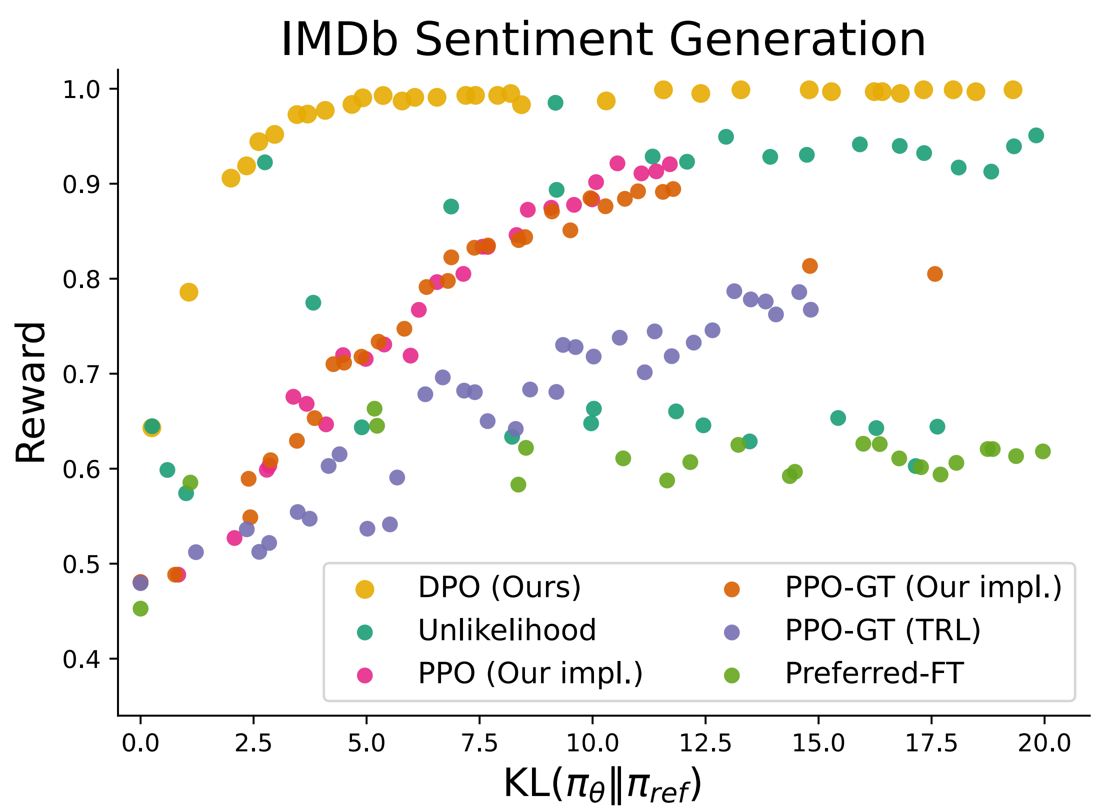
- Experimented with multiple learning techniques:
  - **DPO:** fine-tune base model using DPO on preference data
  - **Preferred-FT:** fine-tune base model on chosen completions on the preference dataset
  - **Unlikelihood:** fine-tune base model to increase likelihood of preferred completion, decrease likelihood of dispreferred completion
  - **PPO:** fine-tune base model using PPO on learned reward model (i.e., RLHF)
  - **PPO-GT:** fine-tune base model using PPO on ground truth reward function

[Rafailov et al. 2023]

PPO

Reward-KL Trade-off

- DPO the most stable across different KL values
- PPO doesn't provide optimal reward even when given ground truth (GT)
- DPO improves over supervised fine-tuning on preferences
- Results are more complex in more realistic scenarios



RLHF vs. DPO

Evaluating the Reward Model

- RLHF and DPO both can be used to compute rewards
  - RLHF: explicitly learns a reward model $r_\psi(\bar{x}, \bar{y})$
  - DPO: we can compute the reward using the base and fine-tuned models $\hat{r}(\bar{x}, \bar{y}) = \beta \log \frac{\pi_\theta(\bar{y} \mid \bar{x})}{\pi_{\text{ref}}(\bar{y} \mid \bar{x})}$
- We already saw that the Llama 2 reward model still leaves a lot of room for improvement
- How do things look like more broadly?

RewardBecnh

Evaluating the Reward Model

- A benchmark suite for reward models

- Follows a similar recipe as in GLUE and SuperGLUE with the specific aim for evaluation reward models

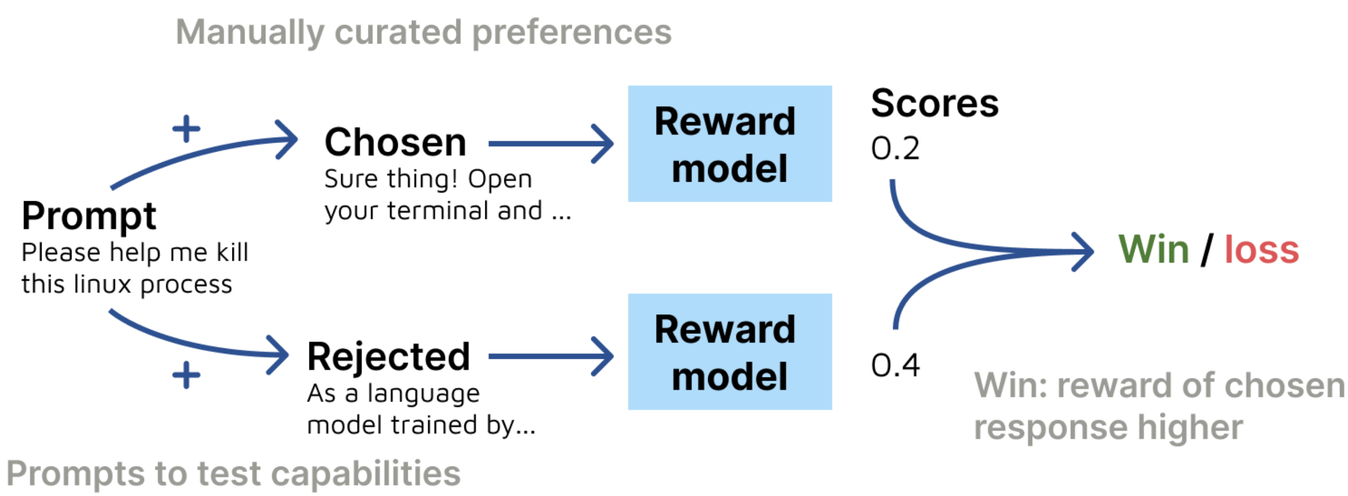


Figure 1: The scoring method of the REWARDBENCH evaluation suite. Each prompt is accompanied by a chosen and rejected completion which are independently rated by a reward model.

[Lambert et al. 2024]

RewardBench

Datasets

Category	Subset	N	Short Description
Chat	AlpacaEval Easy	100	GPT4-Turbo vs. Alpaca 7bB from Li et al. (2023b)
358 total	AlpacaEval Length	95	Llama 2 Chat 70B vs. Guanaco 13B completions
	AlpacaEval Hard	95	Tulu 2 DPO 70B vs. Davinici003 completions
	MT Bench Easy	28	MT Bench ratings 10s vs. 1s from Zheng et al. (2023)
	MT Bench Medium	40	MT Bench completions rated 9s vs. 2-5s
Chat Hard	MT Bench Hard	37	MT Bench completions rated 7-8s vs. 5-6
456 total	LLMBar Natural	100	LLMBar chat comparisons from Zeng et al. (2023)
	LLMBar Adver. Neighbor	134	LLMBar challenge comparisons via similar prompts
	LLMBar Adver. GPTInst	92	LLMBar comparisons via GPT4 similar prompts
	LLMBar Adver. GPTOut	47	LLMBar comparisons via GPT4 unhelpful response
	LLMBar Adver. Manual	46	LLMBar manually curated challenge completions
Safety	Refusals Dangerous	100	Preferring refusal to elicit dangerous responses
740 total	Refusals Offensive	100	Preferring refusal to elicit offensive responses
	XSTest Should Refuse	154	Prompts that should be refused Röttger et al. (2023)
	XSTest Should Respond	250	Preferring responses to queries with trigger words
	Do Not Answer	136	Questions that LLMs should refuse (Wang et al., 2023)
Reasoning	PRM Math	447	Human vs. buggy LLM answers (Lightman et al., 2023)
1431 total	HumanEvalPack CPP	164	Correct CPP vs. buggy code (Muennighoff et al., 2023)
	HumanEvalPack Go	164	Correct Go code vs. buggy code
	HumanEvalPack Javascript	164	Correct Javascript code vs. buggy code
	HumanEvalPack Java	164	Correct Java code vs. buggy code
	HumanEvalPack Python	164	Correct Python code vs. buggy code
	HumanEvalPack Rust	164	Correct Rust code vs. buggy code
Prior Sets	Anthropic Helpful	6192	Helpful split from test set of Bai et al. (2022a)
17.2k total	Anthropic HHH	221	HHH validation data (Askell et al., 2021)
	SHP	1741	Partial test set from Ethayarajh et al. (2022)
	Summarize	9000	Test set from Stiennon et al. (2020)

Table 1: Summary of the dataset used in REWARDBENCH. Note: Adver. is short for Adverserial.

[Lambert et al. 2024]

RewardBench

Reward Model	Avg	Chat	Chat Hard	Safety	Reason	Prior Sets
berkeley-nest/Starling-RM-34B	**81.5**	96.9	59.0	**89.9**	90.3	71.4
allenai/tulu-2-dpo-70b	77.0	97.5	60.8	85.1	88.9	52.8
mistralai/Mixtral-8x7B-Instruct-v0.1	75.8	95.0	65.2	76.5	**92.1**	50.3
berkeley-nest/Starling-RM-7B-alpha	74.7	**98.0**	43.5	88.6	74.6	68.6
NousResearch/Nous-Hermes-2-Mixtral-8x7B-DPO	73.9	91.6	62.3	81.7	81.2	52.7
HuggingFaceH4/zephyr-7b-alpha	73.6	91.6	63.2	70.0	89.6	53.5
NousResearch/Nous-Hermes-2-Mistral-7B-DPO	73.5	92.2	59.5	83.8	76.7	55.5
allenai/tulu-2-dpo-13b	72.9	95.8	56.6	78.4	84.2	49.5
openbmb/UltraRM-13b	71.3	96.1	55.2	45.8	81.9	**77.2**
HuggingFaceH4/zephyr-7b-beta	70.7	95.3	62.6	54.1	89.6	52.2
allenai/tulu-2-dpo-7b	70.4	97.5	54.6	74.3	78.1	47.7
stabilityai/stablelm-zephyr-3b	70.1	86.3	58.2	74.0	81.3	50.7
HuggingFaceH4/zephyr-7b-gemma-v0.1	66.6	95.8	51.5	55.1	79.0	51.7
Qwen/Qwen1.5-72B-Chat	66.2	62.3	67.3	71.8	87.4	42.3
allenai/OLMo-7B-Instruct	66.1	89.7	48.9	64.1	76.3	51.7
IDEA-CCNL/Ziya-LLaMA-7B-Reward	66.0	88.0	41.3	62.5	73.7	64.6
stabilityai/stablelm-2-zephyr-1_6b	65.9	96.6	46.6	60.0	77.4	48.7
Qwen/Qwen1.5-14B-Chat	65.8	57.3	67.4	77.2	85.9	41.2
Qwen/Qwen1.5-7B-Chat	65.6	53.6	**69.8**	75.3	86.4	42.9
OpenAssistant/oasst-rm-2.1-pythia-1.4b-epoch-2.5	65.1	88.5	47.8	62.1	61.4	65.8
Random	50.0	50.0	50.0	50.0	50.0	50.0

Table 2: Top-20 Leaderboard results in REWARDBENCH. Evaluating many RMs shows that there is still large variance in RM training and potential for future improvement across the more challenging instruction and reasoning tasks. Icons refer to model types: Sequence Classifier ( ), Direct Preference Optimization ( ), and a random model ( ).

[Lambert et al. 2024]

RewardBench

- DPO models are more common (with open models)
    - Because they are easier to get to work for the complete RLHF process
- But explicit reward models can still be stronger
    - Thereby giving PPO later on a strong signal

Reward Model	Avg
berkeley-nest/Starling-RM-34B	**81.5**
allenai/tulu-2-dpo-70b	77.0
mistralai/Mixtral-8x7B-Instruct-v0.1	75.8
berkeley-nest/Starling-RM-7B-alpha	74.7
NousResearch/Nous-Hermes-2-Mixtral-8x7B-DPO	73.9
HuggingFaceH4/zephyr-7b-alpha	73.6
NousResearch/Nous-Hermes-2-Mistral-7B-DPO	73.5
allenai/tulu-2-dpo-13b	72.9
openbmb/UltraRM-13b	71.3
HuggingFaceH4/zephyr-7b-beta	70.7
allenai/tulu-2-dpo-7b	70.4
stabilityai/stablelm-zephyr-3b	70.1
HuggingFaceH4/zephyr-7b-gemma-v0.1	66.6
Qwen/Qwen1.5-72B-Chat	66.2
allenai/OLMo-7B-Instruct	66.1
IDEA-CCNL/Ziya-LLaMA-7B-Reward	66.0
stabilityai/stablelm-2-zephyr-1_6b	65.9
Qwen/Qwen1.5-14B-Chat	65.8
Qwen/Qwen1.5-7B-Chat	65.6
OpenAssistant/oasst-rm-2.1-pythia-1.4b-epoch-2.5	65.1
Random	50.0

[Lambert et al. 2024]

Targeted LLM Fine-tuning

The Case of Conversational Behavior

- **Goal:** language model that can produce continuations that appear reasonable in a live conversation with a user
- Problems with expecting this from base LLMs:
  - Not trained on a lot of dialogue data (not really what you get from web text)
  - Dialogue is a complex dynamic process

Targeted LLM Fine-tuning

The Case of Conversational Behavior

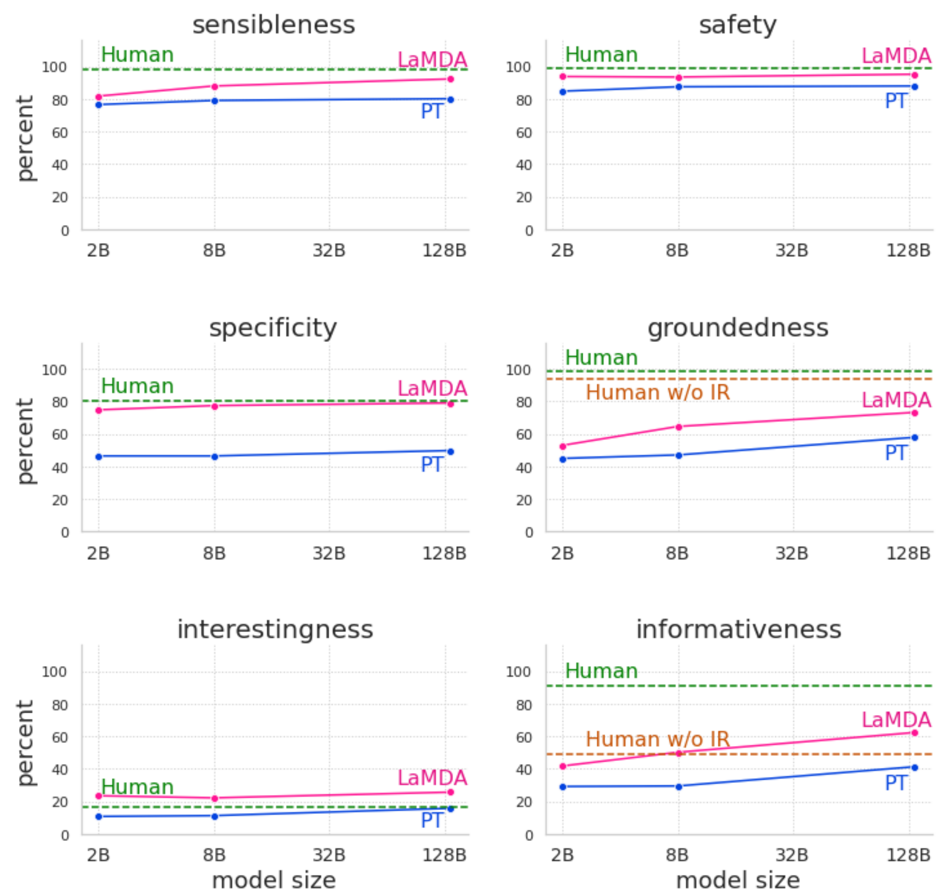
- **Main idea:** collect data from LLM-user interactions, and fine-tune
  - Several thousand dialogues between LaMDA and humans
  - Other annotators rate conversations on different metrics
- Automatic data annotation
  - Fine-tune an LLM to predict ratings of candidate responses in new dialogues
  - Use new model to label utterances in pre-training dataset
- Conversational fine-tuning
  - Filter pre-training data to those labeled with high ratings by discriminator
  - Fine-tune on this high-quality pre-training data
  - Further fine-tune on 4K “gold-standard” conversations with crowdworkers

[Thoppilan et al. 2022]

Targeted LLM Fine-tuning

The Case of Conversational Behavior

- Process also included fine-tuning the model to retrieve external data
- Consistent significant effects across model sizes



[Thoppilan et al. 2022]

Aligning LLMs

Key Takeaways

- RLHF is an essential, but complex and compute-intensive process to make expressive LLMs useful
- It presents a very restricted instance of RL (basically a bandit problem), even though it uses relatively advanced algorithms
- Data is the key to the process, and it requires careful curation and annotation
- There are supervised approaches that can get similar or even equal results (e.g., DPO)
- There is room for small-scale fine-tuning, especially in targeted scenarios
- Many open problems, a lot of active research in this area

Acknowledgements

- Instruction tuning and conversational tuning slides inspired by Alane Suhr's slides for Berkeley CS 288

- The notation for the RL slides is based on Sutton and Barto (second edition)

- RLHF slides are inspired by slides by Eric Mitchell

- DPO slides are based on slides by Eric Mitchell