Annotated Data
Aligning LLMs
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

**Step 1**
Collect demonstration data, and train a supervised policy.

- A prompt is sampled from our prompt dataset.
- A labeler demonstrates the desired output behavior.
- This data is used to fine-tune GPT-3 with supervised learning.

**Step 2**
Collect comparison data, and train a reward model.

- A prompt and several model outputs are sampled.
- A labeler ranks the outputs from best to worst.
- This data is used to train our reward model.

**Step 3**
Optimize a policy against the reward model using reinforcement learning.

- A new prompt is sampled from the dataset.
- The policy generates an output.
- The reward model calculates a reward for the output.
- The reward is used to update the policy using PPO.

[Figure from Ouyang et al. 2022]
Aligning LLMs
The Complete Process

Step 0: Unsupervised pre-training (tons of data; >1T tokens)

Step 1: Supervised fine-tuning on human demos

Step 2: Fit a reward model to human preferences over $\pi_{SFT}$ samples

Step 3: Optimize a policy to maximize learned rewards

"Write a poem about jazz."
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
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

- Multiple-Choice QA
  - CommonsenseQA
  - DREAM
  - QuAII
  - QuaRTz
  - Social IQA
  - WiQA
  - Cosmos QA
  - QASC
  - QuaRel
  - SciQ
  - Wiki Hop
- Closed-Book QA
  - Hotpot QA
  - Wiki QA
- Structure-To-Text
  - Common Gen
  - Wiki Bio
- Sentiment
  - Amazon
  - App Reviews
  - IMDB
  - Rotten Tomatoes
  - Yelp
- Summarization
  - CNN Daily Mail
  - Gigaword
  - MultiNews
  - SamSum
  - XSum
- Topic Classification
  - AG News
  - DBPedia
  - TREC
- Paraphrase Identification
  - MRPC
  - PAWS
  - QQP
- Coreference Resolution
  - WSC
  - Winogrande
- Natural Language Inference
  - ANLI
  - CB
  - RTE
- Sentence Completion
  - COPA
  - HellaSwag
  - Story Cloze
- Word Sense Disambiguation
  - WiC

BIG-Bench
- Code Description
- Conceptual
- Hindu Knowledge
- Known Unknowns
- Language ID
- Logic Grid
- Logical Deduction
- Misconceptions
- Movie Dialog
- Novel Concepts
- Strategy QA
- Syllogisms
- Vitamin C
- Winowhy

[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
Instruction Tuning
The T0 Recipe

QQP (Paraphrase)

<table>
<thead>
<tr>
<th>Question1</th>
<th>How is air traffic controlled?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question2</td>
<td>How do you become an air traffic controller?</td>
</tr>
<tr>
<td>Label</td>
<td>0</td>
</tr>
</tbody>
</table>

I received the questions "(Question1)" and "(Question2)". Are they duplicates?

XSum (Summary)

| Document | The picture appeared on the wall of a Poundland store on Whymark Avenue... |
| Summary  | Graffiti artist Banksy is believed to be behind... |

How would you rephrase that in a few words?

First, please read the article: (Document) Now, can you write me an extremely short abstract for it?

Label 0

{Choices[label]}
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.
Instruction Tuning
The Flan-PaLM Recipe

• Find as many datasets as you can $\rightarrow$ 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

<table>
<thead>
<tr>
<th>Prompt</th>
<th>Write a poem to help me remember the first 10 elements on the periodic table, giving each element its own line.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>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.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prompt</th>
<th>I want you to roast me. I want you to make it particularly brutal, swearing at me.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.
Aligning LLMs

Step 0: Unsupervised pre-training (tons of data; >1T tokens)

Step 1: Supervised fine-tuning on human demos

Step 2: Fit a reward model to human preferences over $\pi_{SFT}$ samples

Step 3: Optimize a policy to maximize learned rewards

"Write a poem about jazz."

$X'$ (prompts)

$X$ (human demos)

$(x^i, y^i_w, y^i_l)$ (preference pairs)

$r_\phi$

$\pi_{\theta_0}$

$\pi_{\theta_{SFT}}$

$\pi_{\theta_{RL}}$

[Figure from Eric Mitchell]
Reinforcement Learning from Human Feedback (RLHF)

Step 0: Unsupervised pre-training
(tons of data; >1T tokens)

Step 1: Supervised fine-tuning
on human demos

Step 2: Fit a reward model
to human preferences over $\pi_{SFT}$ samples

Step 3: Optimize a policy
to maximize learned rewards

[Figure from Eric Mitchell]
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
What are the steps for making a simple cake?

1. Warm up the oven.
2. Grease a cake pan.
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.
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.
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?
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.
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]
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 $\bar{y}^{(i)}_w$ and a losing output $\bar{y}^{(i)}_l$

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

\[
\{(\bar{x}^{(i)}, \bar{y}_w^{(i)}, \bar{y}_l^{(i)})\}^N_{i=1}
\]

• 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 $\rightarrow$ high score, bad outputs $\rightarrow$ 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)
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 \( S \times A \to \mathbb{R} \)

\[
S = \{s_0, s_1, s_2\} \\
A = \{a_1, a_2\} \\
T(s_0, a_1) = s_2 \\
T(s_0, a_2) = s_1 \\
T(s_1, a_1) = s_2 \\
T(s_2, a_2) = s_1 \\
r(s_0, a_1) = 1 \\
r(s_0, a_2) = -1 \\
r(s_1, a_1) = 2 \\
r(s_2, a_2) = 0
\]
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 $r(s_t, a_t)$.

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

$A = \{a_1, a_2\}$

$T(s_0, a_1) = s_2$

$T(s_0, a_2) = s_1$

$T(s_1, a_1) = s_2$

$T(s_1, a_2) = s_1$

$T(s_2, a_2) = s_1$

$r(s_0, a_1) = 1$

$r(s_0, a_2) = -1$

$r(s_1, a_1) = 2$

$r(s_1, a_2) = 0$

$r(s_2, a_2) = 0$

* Deterministic and finite MDP
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 \(\theta\):

\[
a_t \sim \pi_\theta(a \mid s_t)
\]

\[
\begin{align*}
S &= \{s_0, s_1, s_2\} \\
A &= \{a_1, a_2\} \\
T(s_0, a_1) &= s_2 \\
T(s_0, a_2) &= s_1 \\
T(s_1, a_1) &= s_2 \\
T(s_2, a_2) &= s_1 \\
r(s_0, a_1) &= 1 \\
r(s_0, a_2) &= -1 \\
r(s_1, a_1) &= 2 \\
r(s_2, a_2) &= 0
\end{align*}
\]

* Deterministic and finite MDP

** There are also non-probabilistic formulations
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 \((t = 0)\):

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

\(S = \{s_0, s_1, s_2\}\)

\(A = \{a_1, a_2\}\)

\(T(s_0, a_1) = s_2\)

\(T(s_0, a_2) = s_1\)

\(T(s_1, a_1) = s_2\)

\(...\)

\(T(s_2, a_2) = s_1\)

\(r(s_0, a_1) = 1\)

\(r(s_0, a_2) = -1\)

\(r(s_1, a_1) = 2\)

\(...\)

\(r(s_2, a_2) = 0\)

* Deterministic and finite MDP

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

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

\[S = \{s_0, s_1, s_2\}\]
\[A = \{a_1, a_2\}\]
\[T(s_0, a_1) = s_2\]
\[T(s_0, a_2) = s_1\]
\[T(s_1, a_1) = s_2\]
\[T(s_1, a_2) = s_1\]
\[
other...
\]
\[T(s_2, a_2) = s_1\]
\[r(s_0, a_1) = 1\]
\[r(s_0, a_2) = -1\]
\[r(s_1, a_1) = 2\]
\[r(s_1, a_2) = 0\]

* 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 \( \pi_\theta \)\(^*,**

\[
v_{\pi_\theta}(s) = E_{\pi_\theta}[G_t \mid s] = E_{\pi_\theta}[r(s, a) + G_{t+1} \mid s]
\]

\[
= \sum_{a \in A} \pi_\theta(a \mid s) \left[ r(s, a) + E_{\pi_\theta}[G_{t+1} \mid T(s, a)] \right]
\]

\[
= \sum_{a \in A} \pi_\theta(a \mid s) \left[ r(s, a) + v_{\pi_\theta}(T(s, a)) \right]
\]

\* 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** \(S^+ \subset S\)

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

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

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 θ, and all gradients are also implicit with respect to θ. First note that the gradient of the state-value function can be written in terms of the action-value function as

\[ \nabla v_\pi(s) = \nabla \left[ \sum_a \tau(a|s) q_\pi(s,a) \right] \quad \text{for all } s \in S \]  
(Exercise 3.18)

\[ = -\sum_a \nabla \tau(a|s) q_\pi(s,a) + \tau(a|s) \nabla q_\pi(s,a) \]  
(product rule of calculus)

\[ = -\sum_a \nabla \tau(a|s) q_\pi(s,a) + \tau(a|s) \sum_{s'} \left( r(s',a)(r + v_\pi(s')) \right) \]  
(Exercise 3.19 and Equation 3.2)

\[ = -\sum_a \nabla \tau(a|s) q_\pi(s,a) + \tau(a|s) \sum_{s'} \left( r(s',a) + \tau(a|s) \sum_{s''} p(s''|s,a) \nabla v_\pi(s'') \right) \]  
(unrolling)

\[ = -\sum_{s',k} \sum_a \Pr(s \rightarrow s', k \pi) \sum_{s''} \nabla \tau(a|s) q_\pi(s,a) \]  
(Exercise 3.18 and Equation 3.2)

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

\[ \nabla J(\theta) = \nabla v_\pi(s_0) \]

\[ = -\sum_a \left( \sum_{k=1}^\infty \sum_{s'} \Pr(s_0 \rightarrow s, k \pi) \right) \sum_{s''} \nabla \tau(a|s) q_\pi(s,a) \]  
(box page 199)

\[ = -\sum_s \left( \sum_{s'} p(s'|s) \nabla \tau(a'|s) q_\pi(s,a) \right) \nabla q_\pi(s,a) \]  
(Eq. 9.3)

\[ = \sum_s p(s) \sum_{s'} \nabla \tau(a|s) q_\pi(s,a) \]  
(Q.E.D.)
Reinforcement Learning

**REINFORCE**

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

While true:

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

- $a_t \sim \pi_\theta(a \mid s_t)$
- $s_{t+1} \leftarrow T(s_t, a_t)$
- $r_t \leftarrow r(s_t, a_t)$

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

- $G \leftarrow \sum_{k=t}^{T} r_t$
- $\theta \leftarrow \theta + \alpha G \nabla \ln \pi_\theta(a_t \mid s_t)$

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

REINFORCE* — Intuition

While true:

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

$$a_t \sim \pi_\theta(a \mid s_t)$$

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

$$r_t \leftarrow r(s_t, a_t)$$

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

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

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

- 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 $\pi_\theta(a_t \mid s_t)$ 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

\[
A_\pi(s, a) = r(s, a) + v_\pi(T(s, a)) - v_\pi(s)
\]

\[
q_\pi(s, a)^*
\]

* 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

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

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$

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

Collect set of trajectories $D_k$ by running the policy $\pi_{\theta_k}$, and computing returns $G_t$

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

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

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

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

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

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

**PPO**

**PPO-Clip Pseudocode (simplified)**

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$

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

Collect set of trajectories $D_k$ by running the policy

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

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

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

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

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

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$

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

Collect set of trajectories $D_k$ by running the policy

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

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

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

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

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

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

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

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$

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

Collect set of trajectories $D_k$ by running the policy

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

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

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

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

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

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

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

PPO

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$

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

Collect set of trajectories $D_k$ by running the policy $\pi_{\theta_k}$, and computing returns $G_t$

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

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

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

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

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

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

PPO-Clip

\[ p_k(\theta) = \frac{\pi_\theta(s,a)}{\pi_\theta(s,a)} \]

\[ A_t = \hat{\Delta} \]

<table>
<thead>
<tr>
<th>( p_k(\theta) )</th>
<th>( A_t )</th>
<th>Return Value of ( \text{min} )</th>
<th>Objective is Clipped</th>
<th>Sign of Objective</th>
<th>Gradient</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_k(\theta) &gt; 0 )</td>
<td>( A_t )</td>
<td>+</td>
<td>no</td>
<td>+</td>
<td>✓</td>
</tr>
<tr>
<td>( p_k(\theta) \in [1-\epsilon, 1+\epsilon] )</td>
<td>+</td>
<td>no</td>
<td>+</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>( p_k(\theta) \in [1-\epsilon, 1+\epsilon] )</td>
<td>-</td>
<td>no</td>
<td>-</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>( p_k(\theta) &lt; 1-\epsilon )</td>
<td>+</td>
<td>no</td>
<td>+</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>( p_k(\theta) &lt; 1-\epsilon )</td>
<td>-</td>
<td>yes</td>
<td>-</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( p_k(\theta) &gt; 1+\epsilon )</td>
<td>+</td>
<td>yes</td>
<td>+</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( p_k(\theta) &gt; 1+\epsilon )</td>
<td>-</td>
<td>no</td>
<td>-</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

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

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

\[ \theta_{k+1} = \arg \max_\theta \frac{1}{|D_k|T} \sum_{(a,s,\hat{\Delta}) \in D_k} \min \left( \frac{\pi_\theta(s,a)}{\pi_\theta(s,a)}, \frac{\pi_\theta(s,a)}{\pi_\theta(s,a)} \right) \]

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

PPO-Clip Pseudocode (simplified)

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$

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

    Collect set of trajectories $D_k$ by running the policy $\pi_{\theta_k}$, and computing returns $G_t$

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

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

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

    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 $\phi \leftarrow \phi_k$):

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

PPO

- PPO is notoriously complex to work with
  - It requires learning a separate value function $v_\phi$
  - 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 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 *single* action as far as the RL MDP

• The transition function $T$ is simple: $T(s, a) = [s; a]$ a simple concatenation

• Reward function $r$: ???
RLHF
The Reward Model

- RL requires a reward function $r : S \times A \rightarrow \mathbb{R}$
- 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 $\psi$

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

- 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 $\psi$ 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 $\succ$:

\[ p(a > b) = \sigma(s(a) - s(b)) \]

• 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 $>:\n  \quad p(a > b) = \sigma(s(a) - s(b))$

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

  $$\mathcal{L}_r(\psi, \mathcal{D}) = -E_{(\bar{x}, \bar{y}_w, \bar{y}_l) \sim \mathcal{D}} \left[ \log p(\bar{y}_w > \bar{y}_l) \right]$$

  $$= -E_{(\bar{x}, \bar{y}_w, \bar{y}_l) \sim \mathcal{D}} \left[ \log \sigma(r_{\psi}([\bar{x}; \bar{y}_w]) - r_{\psi}([\bar{x}; \bar{y}_l])) \right]$$

- 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
Rearwrd Model

Data and Performance — Llama 2

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

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

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.
The reward model was trained in an iterative process.

Intermediate models were used to pick examples to annotate for preferences for later models (why?)

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.

We report the results in terms of accuracy in Table 7. As expected, our own reward models perform the best on our internal test sets collected based on \( L \)-C, with the Helpfulness reward model performing best on the Meta Helpfulness test set, and similarly the Safety reward model performing best on the Meta Safety test set. Overall, our reward models outperform all of the baselines, including GPT-4. Interestingly, GPT-4 performs better than other non-Meta reward models, despite not being trained directly nor targeting specifically this reward modeling task.

The fact that helpfulness and safety performed the best on their own domain is potentially due to the tension between the two objectives (i.e., being as helpful as possible versus refusing unsafe prompts when necessary), which may confuse the reward model during training. In order for a single model to perform well on both dimensions, it needs to not only learn to select the better response given a prompt but also to distinguish adversarial prompts from safe ones. As a result, optimizing two separate models eases the reward modeling task. More detailed analysis on this tension between safety and helpfulness can be found in Appendix A.4.1.

When we group the scores by preference rating in Table 8, we can see that the accuracy is superior for the “significantly better” test set and degrades gradually as comparison pairs become more similar (e.g., “slightly better”). It is expected that learning to model human preferences becomes challenging when deciding between two similar model responses, due to annotator subjectivity and their reliance on nuanced details that may differentiate responses.

We emphasize that the accuracy on more distinct responses matters the most to improve \( L \)-C performance. The human preference annotation agreement rate is also higher on more distinct responses than similar pairs.

**Scaling Trends.** We study the scaling trends in terms of data and model size for the reward model, fine-tuning different model sizes on an increasing amount of the reward model data collected each week (see the details on volume per batch in Table 26). Figure 6 reports these trends, showing the expected result that larger models obtain higher performance for a similar volume of data. More importantly, the scaling performance has not yet plateaued given the existing volume of data annotation used for training, a signal that there is room for more improvement with more annotations. We note that reward model accuracy is one of the most important proxies for the final performance of \( L \)-C. While best practices for comprehensively evaluating a generative model is an open research question, the ranking task of the reward has no ambiguity. Therefore, everything else being equal, an improvement of the reward model can be directly translated into an improvement for \( L \)-C.

**3.2.3 Iterative Fine-Tuning**

As we received more batches of human preference data annotation, we were able to train better reward models and collect more prompts. We therefore trained successive versions for RLHF models, referred to here as RLHF-V1, ..., RLHF-V5.

We explored RLHF fine-tuning with two main algorithms:

---

**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.

[Figure from the Llama 2 paper: Touvron et al. 2023]
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.
Step 0: **Unsupervised pre-training** (tons of data; >1T tokens)

Step 1: **Supervised fine-tuning** on human demos

Step 2: **Fit a reward model** to human preferences over $\pi_{SFT}$ samples

Step 3: **Optimize a policy** to maximize learned rewards

"Write a poem about jazz."
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

\[ \{(\vec{x}^{(i)}, \vec{y}_{w}^{(i)}, \vec{y}_{l}^{(i)})\}_{i=1}^{N} \]

- 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

---

**Reinforcement Learning from Human Feedback (RLHF)**

- Use preference data
- Sample completions
- Label rewards
- Reward model
- LM policy

**Direct Preference Optimization (DPO)**

- Use preference data
- Final LM

---

[Rafailov et al. 2023]
DPO
The RL Optimization Problem

• DPO starts with a very similar RL objective to PPO

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

- Where $\pi_{\text{ref}}$ 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

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

- Where \( \pi_{\text{ref}} \) is set to the pre-RL distribution

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

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

• We know from existing work [Peters et al. 2007, Peng et al. 2019] that the optimal policy \( \pi^* \) maintains

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

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

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

\[
r(\bar{x}, \bar{y}) = \beta \log \frac{\pi^*(\bar{y} | \bar{x})}{\pi_{\text{ref}}(\bar{y} | \bar{x})} + \beta \log Z(\bar{x})
\]
DPO

Derivation

• We can express the reward function:

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

• Why is this important?
We can express the reward function:

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

Why is this important?

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

\[
p(a > b) = \sigma(s(a) - s(b))
\]
DPO

Derivation

• We can express the reward function:

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

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

\[ p(\bar{y}_w > \bar{y}_l) = \sigma \left( \beta \log \frac{\pi^*(\bar{y}_w | \bar{x})}{\pi_{\text{ref}}(\bar{y}_w | \bar{x})} + \beta \log Z(\bar{x}) - \beta \log \frac{\pi^*(\bar{y}_l | \bar{x})}{\pi_{\text{ref}}(\bar{y}_l | \bar{x})} - \beta \log Z(\bar{x}) \right) \]

\[ = \sigma \left( \beta \log \frac{\pi^*(\bar{y}_w | \bar{x})}{\pi_{\text{ref}}(\bar{y}_w | \bar{x})} - \beta \log \frac{\pi^*(\bar{y}_l | \bar{x})}{\pi_{\text{ref}}(\bar{y}_l | \bar{x})} \right) \]
If we use $\pi_\theta$ instead of $\pi^*$ and sum over our data, we directly get a negative log-likelihood loss to optimize:

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

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

The gradient for this loss is:

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

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 Mechanisms

- The DPO gradient is:

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

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

$\beta$ functions like a “learning rate” following the strength of the KL constraint

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 Mechanisms

• The DPO gradient is:

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

$\beta$ functions like a "learning rate" following the strength of the KL constraint

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

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

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

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

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

- We already saw that the Llama 2 reward model still leaves a lot of room for improvement

- How do things look like more broadly?
RewardBench
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.

[1] Lambert et al. 2024
Datasets

<table>
<thead>
<tr>
<th>Category</th>
<th>Subset</th>
<th>N</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chat</td>
<td>AlpacaEval Easy</td>
<td>100</td>
<td>GPT4-Turbo vs. Alpaca 7bB from Li et al. (2023b)</td>
</tr>
<tr>
<td></td>
<td>358 total</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AlpacaEval Length</td>
<td>95</td>
<td>Llama 2 Chat 70B vs. Guanaco 13B completions</td>
</tr>
<tr>
<td></td>
<td>AlpacaEval Hard</td>
<td>95</td>
<td>Tulu 2 DPO 70B vs. Davinci003 completions</td>
</tr>
<tr>
<td></td>
<td>MT Bench Easy</td>
<td>28</td>
<td>MT Bench ratings 10s vs. 1s from Zheng et al. (2023)</td>
</tr>
<tr>
<td></td>
<td>MT Bench Medium</td>
<td>40</td>
<td>MT Bench completions rated 9s vs. 2-5s</td>
</tr>
<tr>
<td>Chat Hard</td>
<td>MT Bench Hard</td>
<td>37</td>
<td>MT Bench completions rated 7-8s vs. 5-6</td>
</tr>
<tr>
<td></td>
<td>LLMBar Natural</td>
<td>100</td>
<td>LLMBar chat comparisons from Zeng et al. (2023)</td>
</tr>
<tr>
<td></td>
<td>456 total</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LLMBar Adver. Neighbor</td>
<td>134</td>
<td>LLMBar challenge comparisons via similar prompts</td>
</tr>
<tr>
<td></td>
<td>LLMBar Adver. GPTInst</td>
<td>92</td>
<td>LLMBar comparisons via GPT4 similar prompts</td>
</tr>
<tr>
<td></td>
<td>LLMBar Adver. GPTOut</td>
<td>47</td>
<td>LLMBar comparisons via GPT4 unhelpful response</td>
</tr>
<tr>
<td></td>
<td>LLMBar Adver. Manual</td>
<td>46</td>
<td>LLMBar manually curated challenge completions</td>
</tr>
<tr>
<td>Safety</td>
<td>Refusals Dangerous</td>
<td>100</td>
<td>Preferring refusal to elicit dangerous responses</td>
</tr>
<tr>
<td></td>
<td>740 total</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Refusals Offensive</td>
<td>100</td>
<td>Preferring refusal to elicit offensive responses</td>
</tr>
<tr>
<td></td>
<td>XTest Should Refuse</td>
<td>154</td>
<td>Prompts that should be refused Röttger et al. (2023)</td>
</tr>
<tr>
<td></td>
<td>XTest Should Respond</td>
<td>250</td>
<td>Preferring responses to queries with trigger words</td>
</tr>
<tr>
<td></td>
<td>Do Not Answer</td>
<td>136</td>
<td>Questions that LLMs should refuse (Wang et al., 2023)</td>
</tr>
<tr>
<td>Reasoning</td>
<td>PRM Math</td>
<td>447</td>
<td>Human vs. buggy LLM answers (Lightman et al., 2023)</td>
</tr>
<tr>
<td></td>
<td>HumanEvalPack CPP</td>
<td>164</td>
<td>Correct CPP vs. buggy code (Muennighoff et al., 2023)</td>
</tr>
<tr>
<td></td>
<td>HumanEvalPack Go</td>
<td>164</td>
<td>Correct Go code vs. buggy code</td>
</tr>
<tr>
<td></td>
<td>HumanEvalPack Javascript</td>
<td>164</td>
<td>Correct Javascript code vs. buggy code</td>
</tr>
<tr>
<td></td>
<td>HumanEvalPack Java</td>
<td>164</td>
<td>Correct Java code vs. buggy code</td>
</tr>
<tr>
<td></td>
<td>HumanEvalPack Python</td>
<td>164</td>
<td>Correct Python code vs. buggy code</td>
</tr>
<tr>
<td></td>
<td>HumanEvalPack Rust</td>
<td>164</td>
<td>Correct Rust code vs. buggy code</td>
</tr>
<tr>
<td></td>
<td>1431 total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prior Sets</td>
<td>Anthropic Helpful</td>
<td>6192</td>
<td>Helpful split from test set of Bai et al. (2022a)</td>
</tr>
<tr>
<td></td>
<td>17.2k total</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Anthropic HHH</td>
<td>221</td>
<td>HHH validation data (Askell et al., 2021)</td>
</tr>
<tr>
<td></td>
<td>SHP</td>
<td>1741</td>
<td>Partial test set from Ethayarajh et al. (2022)</td>
</tr>
<tr>
<td></td>
<td>Summarize</td>
<td>9000</td>
<td>Test set from Stiennon et al. (2020)</td>
</tr>
</tbody>
</table>

Table 1: Summary of the dataset used in REWARDBENCH. Note: Adver. is short for Adverserial.
Table 2: Top-20 Leaderboard results in REWARD BENCH. 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 ( ).
• 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

<table>
<thead>
<tr>
<th>Reward Model</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>berkeley-nest/Starling-RM-34B</td>
<td>81.5</td>
</tr>
<tr>
<td>allenai/tulu-2-dpo-70b</td>
<td>77.0</td>
</tr>
<tr>
<td>mistralai/Mixtral-8x7B-Instruct-v0.1</td>
<td>75.8</td>
</tr>
<tr>
<td>berkeley-nest/Starling-RM-7B-alpha</td>
<td>74.7</td>
</tr>
<tr>
<td>NousResearch/Nous-Hermes-2-Mixtral-8x7B-DPO</td>
<td>73.9</td>
</tr>
<tr>
<td>HuggingFaceH4/zephyr-7b-alpha</td>
<td>73.6</td>
</tr>
<tr>
<td>NousResearch/Nous-Hermes-2-Mistral-7B-DPO</td>
<td>73.5</td>
</tr>
<tr>
<td>allenai/tulu-2-dpo-13b</td>
<td>72.9</td>
</tr>
<tr>
<td>openbmb/UltraRM-13b</td>
<td>71.3</td>
</tr>
<tr>
<td>HuggingFaceH4/zephyr-7b-beta</td>
<td>70.7</td>
</tr>
<tr>
<td>allenai/tulu-2-dpo-7b</td>
<td>70.4</td>
</tr>
<tr>
<td>stabilityai/stablelm-zephyr-3b</td>
<td>70.1</td>
</tr>
<tr>
<td>HuggingFaceH4/zephyr-7b-gemma-v0.1</td>
<td>66.6</td>
</tr>
<tr>
<td>Qwen/Qwen1.5-72B-Chat</td>
<td>66.2</td>
</tr>
<tr>
<td>allenai/OLMo-7B-Instruct</td>
<td>66.1</td>
</tr>
<tr>
<td>IDEA-CCNL/Ziya-LLaMA-7B-Reward</td>
<td>66.0</td>
</tr>
<tr>
<td>stabilityai/stablelm-2-zephyr-1.6b</td>
<td>65.9</td>
</tr>
<tr>
<td>Qwen/Qwen1.5-14B-Chat</td>
<td>65.8</td>
</tr>
<tr>
<td>Qwen/Qwen1.5-7B-Chat</td>
<td>65.6</td>
</tr>
<tr>
<td>OpenAssistant/oasst-ram-2.1-pythia-1.4b-epoch-2.5</td>
<td>65.1</td>
</tr>
<tr>
<td>Random</td>
<td>50.0</td>
</tr>
</tbody>
</table>

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 (.)

2. Chat Hard: Testing a reward model’s abilities to understand trick questions and subtly different instruction responses. Prompts and chosen, rejected pairs are selected from MT Bench examples with similar ratings and adversarial data specifically for fooling LLM-as-a-judge tools from LLMBar’s evaluation set (Zeng et al., 2023) (reformatted for RMs).

3. Safety: Testing the models’ tendencies to refuse dangerous content and to avoid incorrect refusals to similar trigger words. Prompts and chosen, rejected pairs are selected from custom versions of the datasets XSTest (Röttger et al., 2023), Do-Not-Answer (Wang et al., 2023), and examples from an in-development refusals dataset at AI2, where the chosen response is a refusal and the rejected is harmful text of either dangerous or offensive nature.

4. Reasoning: Evaluating the models’ code and reasoning abilities. Code prompts are created by reformatting HumanEvalPack examples with correct code as chosen and rejected as one with bugs (Muennighoff et al., 2023). Reasoning prompts pair reference answers with incorrect model generations from the PRM800k dataset (Lightman et al., 2023).

5. Prior Sets: For consistency with recent work on training reward models, we average performance over test sets from existing preference datasets. We use the Anthropic Helpful split (Bai et al., 2022a) (the only multi-turn data), the Anthropic HHH subset of BIG-Bench (Askell et al., 2021), a curated subset of the test set from the Stanford Human Preferences (SHP) Dataset (Ethayarajh et al., 2022), and OpenAI’s Learning to Summarize Dataset (Stiennon et al., 2020).

For more test sets and details, visit: https://huggingface.co/datasets/allenai/preference-test-sets
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
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