

From Language Models to Assistants

Pawan Goyal

CSE, IIT Kharagpur

CS60010

GPT Assistant training pipeline



SFT Dataset

Base model
SFT model

Prompt:

"Can you write a short introduction about the relevance of the term "monopsony" in economics? Please use examples related to potential monopsonies in the labour market and cite relevant research."

Response:

""Monopsony" refers to a market structure where there is only one buyer for a particular good or service. In economics, this term is particularly relevant in the labor market, where a monopsony employer has significant power over the wages and working conditions of their employees. The presence of a monopsony can result in lower wages and reduced employment opportunities for workers, as the employer has little incentive to increase wages or provide better working conditions. Recent research has identified potential monopsonies in industries such as retail and fast food, where a few large companies control a significant portion of the market (Bivens & Mishel, 2013). In these industries, workers often face low wages, limited benefits, and reduced bargaining power, leading to a situation where they are dependent on the employer for their livelihood. This dependence can result in further suppression of wages and a decline in working conditions. Overall, the concept of monopsony is essential to understanding the dynamics of labor markets and the impact of market power on workers. Further research is needed to understand the extent and impact of monopsonies on the economy and to develop policies to address this issue. References: Bivens, J., & Mishel, L. (2013). The Pay of Corporate Executives and Financial Professionals as Evidence of Rents in Top 1 Percent Incomes. Journal of Economic Perspectives, 27(3), 57-78."

[example from OpenAssistant Conversations Dataset (OASST1)]

Labeling instructions

Excerpt of labeling instructions on the API prompt distribution

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

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

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

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

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

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

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

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

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

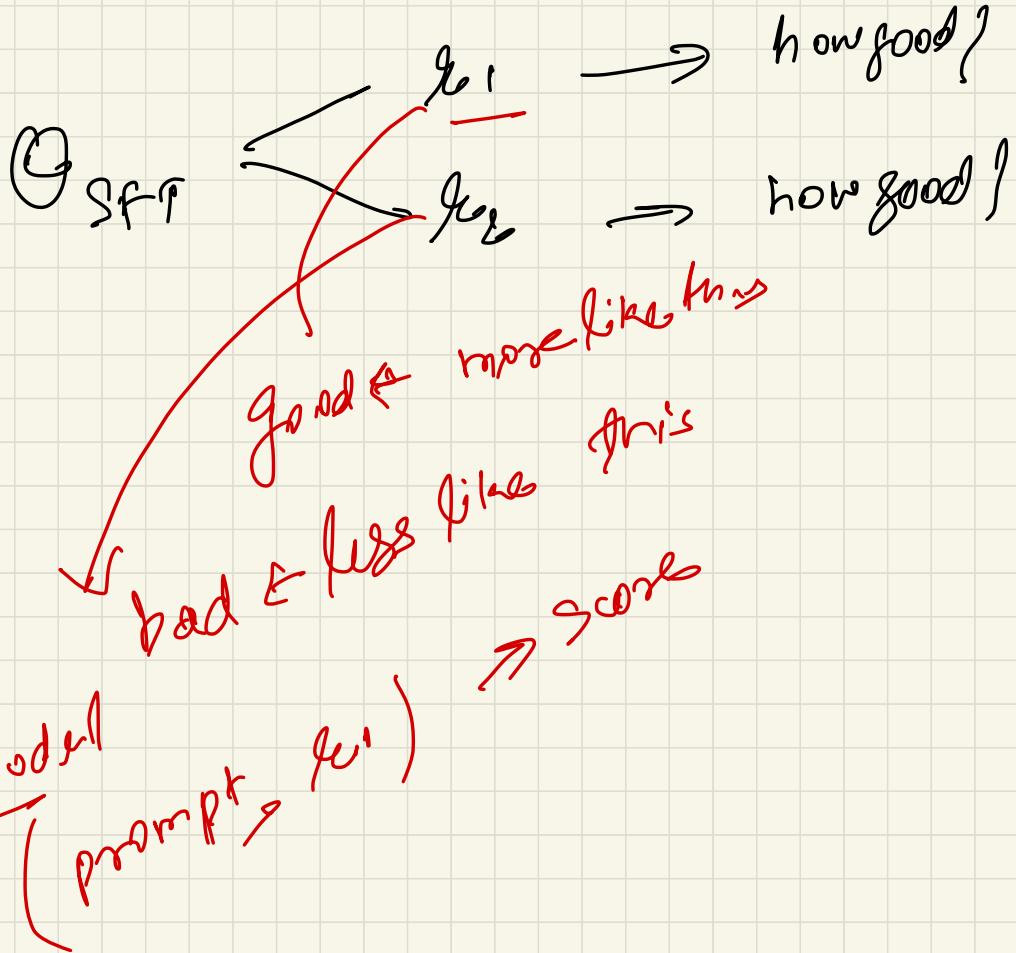
For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful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

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

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

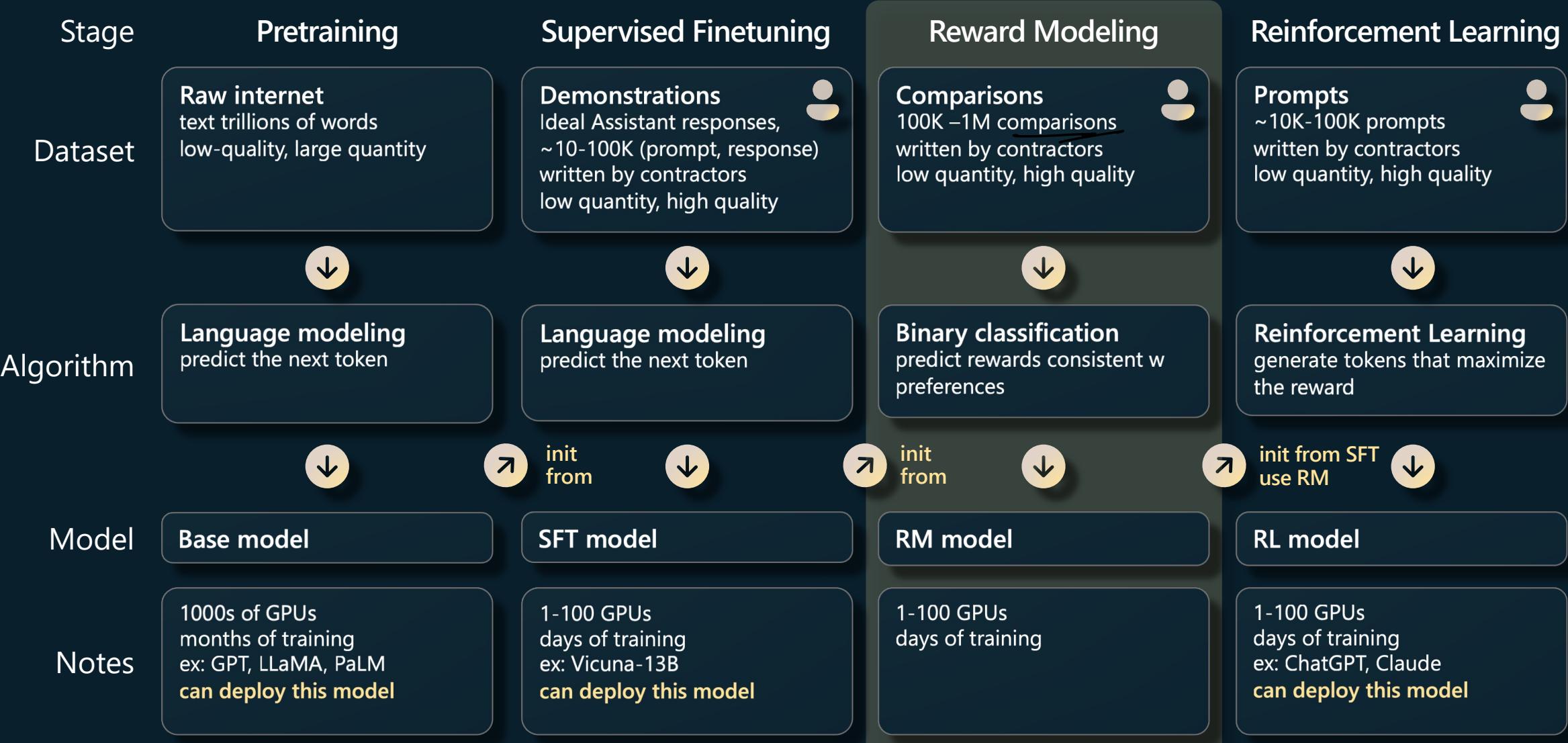
[InstructGPT]

Prompt



How?
Reward model
(prompt, g_1)

GPT Assistant training pipeline



RM Dataset

You
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Here's how you can use the function:

```
python
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing (`[::-1]`). If the word is equal to its reverse, it is a palindrome, and the function returns `True`. Otherwise, it returns `False`.

You
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns `True`, indicating that the string is a palindrome. Otherwise, it returns `False`.

Here's an example of how you can use the function:

```
python
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.)
```

Output:

```
csharp
radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.

You
Write a Python function that checks if a given string is a palindrome.

ChatGPT
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing (`[::-1]` creates a reversed copy of the string). If they are equal, the string is a palindrome, and the function returns `True`; otherwise, it returns `False`.

Here's an example usage of the function:

```
python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "hello world"
print(is_palindrome(string3)) # False
```

In the above example, `is_palindrome` is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns `True`. For the non-palindrome "hello world," it returns `False`.

RM Dataset

Write a Python function that checks if a given string is a palindrome.

ChatGPT Certainly! Here's a Python function that checks if a given string is a palindrome.

```
python
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```

Here's how you can use the function:

```
python
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing (`[::-1]`). If the word is equal to its reverse, it is a palindrome, and the function returns `True`. Otherwise, it returns `False`.

3

Write a Python function that checks if a given string is a palindrome.

ChatGPT Certainly! Here's a Python function that checks if a given string is a palindrome.

```
python
def is_palindrome(string):
    # Convert the string to lowercase and remove spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns `True`, indicating that the string is a palindrome. Otherwise, it returns `False`.

Here's an example of how you can use the function:

```
python
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.)
```

Output:

```
csharp
radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.

2

Write a Python function that checks if a given string is a palindrome.

ChatGPT Certainly! Here's a Python function that checks if a given string is a palindrome.

```
python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join(char.lower() for char in s if char.isalnum())

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```

In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing (`[::-1]` creates a reversed copy of the string). If they are equal, the string is a palindrome, and the function returns `True`; otherwise, it returns `False`.

Here's an example usage of the function:

```
python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

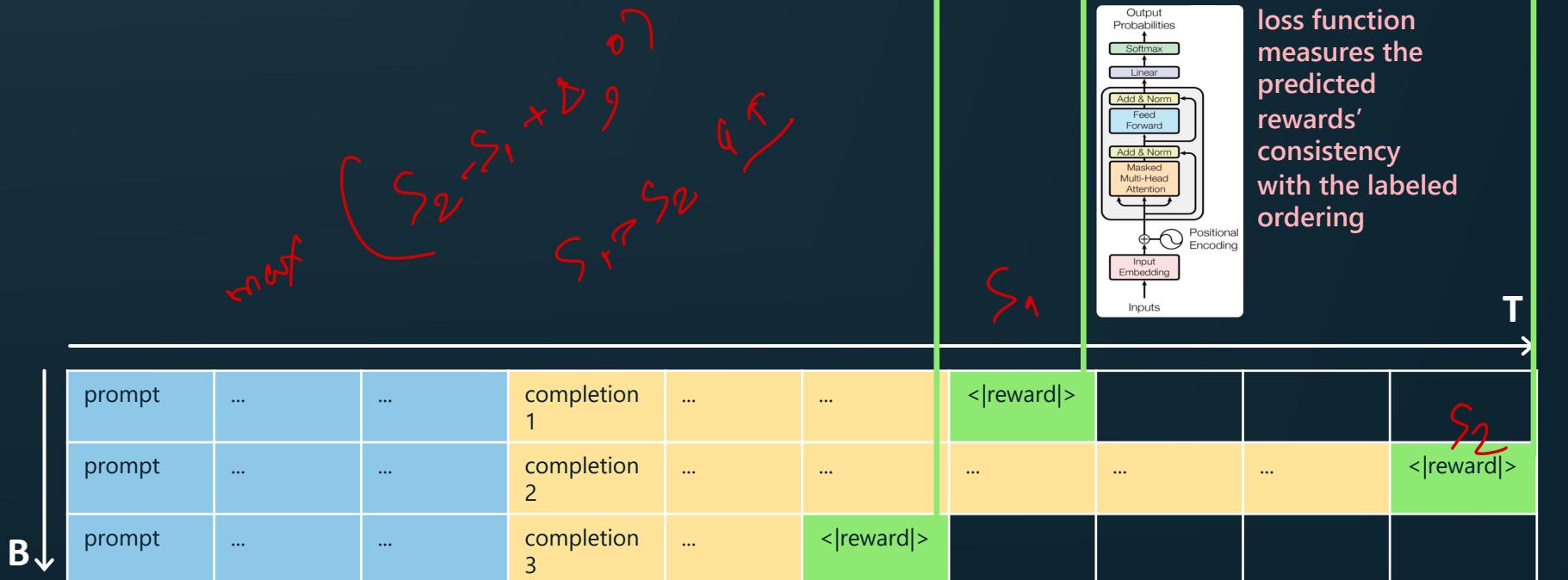
string3 = "hello world"
print(is_palindrome(string3)) # False
```

In the above example, `is_palindrome` is called with three different strings. The function correctly identifies the palindromes ("A man, a plan, a canal: Panama" and "race car") and returns `True`. For the non-palindrome "hello world," it returns `False`.

1

RM Training

Blue are the prompt tokens, identical across rows
Yellow are completion tokens, different in each row
Green is the special <|reward|> token "readout"
Only the outputs at the green cells is used, the rest are ignored



GPT Assistant training pipeline



RL Training

Blue are the prompt tokens, identical across rows

Yellow are completion tokens by the model (initialized with SFT model)

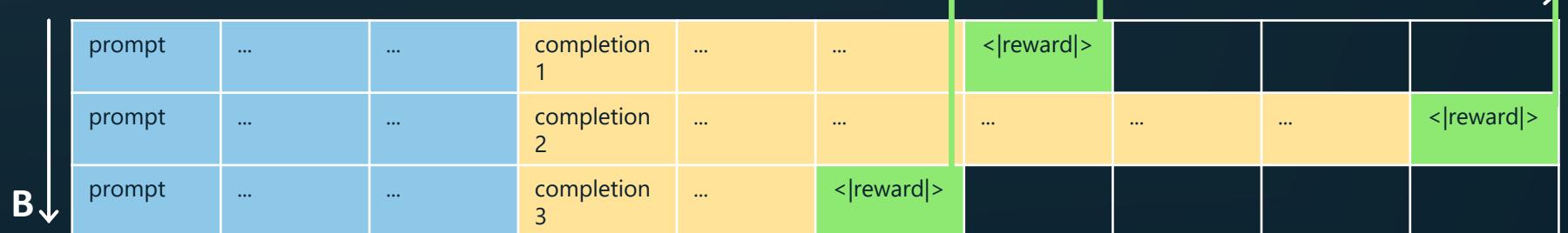
Green is the special `<|reward|>` token “readout”, RM now predicts these

Only the yellow cells are trained on, the rest are ignored.

The sampled tokens become labels, but the training objective is weighted by the “advantage” (normalized rewards)

In this example:

- Row #1 tokens were great. These get their probabilities boosted.
- Row #2 tokens were bad. These get their probabilities decreased.
- Row #3 tokens were ~ok. These get their probabilities slightly boosted.



[Proximal Policy Optimization Algorithms, Schulman et al. 2017]

Optimizing for Human Preferences

- Let's say we were training a language model on some task (e.g. summarization).
- For each LM sample s , imagine we had a way to obtain a *human reward* of that summary: $R(s) \in \mathbb{R}$, higher is better.

SAN FRANCISCO,
California (CNN) --
A magnitude 4.2
earthquake shook the
San Francisco

...
overturn unstable
objects.

An earthquake hit
San Francisco.
There was minor
property damage,
but no injuries.

$$s_1 \\ R(s_1) = 8.0$$

The Bay Area has
good weather but is
prone to
earthquakes and
wildfires.

$$s_2 \\ R(s_2) = 1.2$$

- Now we want to maximize the expected reward of samples from our LM:

$$E_{\hat{s} \sim p_\theta(s)}[R(\hat{s})]$$

RLHF Pipeline

Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

Explain the moon landing to a 6 year old

A labeler demonstrates the desired output behavior.



Some people went to the moon...

This data is used to fine-tune GPT-3 with supervised learning.



initial set

Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

Explain the moon landing to a 6 year old

- A Explain gravity...
- B Explain why...
- C Moon is natural satellite of...
- D People went to the moon...

A labeler ranks the outputs from best to worst.



D > C > A = B

This data is used to train our reward model.



D > C > A = B

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

Write a story about frogs



Once upon a time...

The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.

Optimizing for Human Preferences

§

- How do we actually change our LM parameters θ to maximize this?

$$\mathbb{E}_{\hat{s} \sim p_{\theta}(s)}[R(\hat{s})]$$

- Let's try doing gradient ascent!

$$\theta_{t+1} := \theta_t + \alpha \nabla_{\theta_t} \mathbb{E}_{\hat{s} \sim p_{\theta_t}(s)}[R(\hat{s})]$$

How do we estimate
this expectation??



What if our reward
function is non-
differentiable??

Using policy gradient / REINFORCE (1992)

- We want to obtain

(defn. of expectation) (linearity of gradient)

$$\nabla_{\theta} \mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s})] = \nabla_{\theta} \sum_s R(s) p_{\theta}(s) = \sum_s R(s) \nabla_{\theta} p_{\theta}(s)$$



- Here we'll use a very handy trick known as the **log-derivative trick**. Let's try taking the gradient of $\log p_{\theta}(s)$

$$\nabla_{\theta} \log p_{\theta}(s) = \frac{1}{p_{\theta}(s)} \nabla_{\theta} p_{\theta}(s) \Rightarrow \nabla_{\theta} p_{\theta}(s) = p_{\theta}(s) \nabla_{\theta} \log p_{\theta}(s)$$

(chain rule)

This is an
expectation of this

- Plug back in:

$$\sum_s R(s) \nabla_{\theta} p_{\theta}(s) = \sum_s p_{\theta}(s) R(s) \nabla_{\theta} \log p_{\theta}(s)$$



$$= \mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s}) \nabla_{\theta} \log p_{\theta}(\hat{s})]$$

Using policy gradient / REINFORCE

- Now we have put the gradient “inside” the expectation, we can approximate this objective with Monte Carlo samples:

$$\nabla_{\theta} \mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s})] = \mathbb{E}_{\hat{s} \sim p_{\theta}(s)} [R(\hat{s}) \nabla_{\theta} \log p_{\theta}(\hat{s})] \approx \frac{1}{m} \sum_{i=1}^m R(s_i) \nabla_{\theta} \log p_{\theta}(s_i)$$

This is why it's called “**reinforcement learning**”: we **reinforce** good actions, increasing the chance they happen again.

- Giving us the update rule:

$$\theta_{t+1} := \theta_t + \alpha \frac{1}{m} \sum_{i=1}^m R(s_i) \nabla_{\theta_t} \log p_{\theta_t}(s_i)$$

~~$\nabla_{\theta_t} \log p_{\theta_t}(s_i)$~~

If R is $++$ Take gradient steps to maximize $p_{\theta}(s_i)$

If R is $--$ Take steps to minimize $p_{\theta}(s_i)$

$\log p_{\theta_t}(s_i) = \sum_{w_j \in s_i} \log p_{\theta_t}(w_j)$

But how do you get the rewards?

For any arbitrary, non-differentiable reward function $R(s)$, we can train our language model to maximize expected reward.

But it has problems in practice!

Problem 1: human-in-the-loop is expensive!

- **Solution:** instead of directly asking humans for preferences, **model their preferences** as a separate (NLP) problem! [[Knox and Stone, 2009](#)]

An earthquake hit
San Francisco.
There was minor
property damage,
but no injuries.

$$S_1 \\ R(S_1) = 8.0$$


The Bay Area has
good weather but is
prone to
earthquakes and
wildfires.

$$S_2 \\ R(S_2) = 1.2$$


Train an LM $RM_\phi(s)$ to
predict human
preferences from an
annotated dataset, then
optimize for RM_ϕ instead.

But how do we even calibrate human preferences?



- **Problem 2:** human judgments are noisy and miscalibrated!
- **Solution:** instead of asking for direct ratings, ask for **pairwise comparisons**, which can be more reliable [[Phelps et al., 2015](#); [Clark et al., 2018](#)]

A 4.2 magnitude earthquake hit San Francisco, resulting in massive damage.

s_3

$$R(s_3) = \text{4.1? } \text{6.6? } \text{3.2?}$$



Training a reward model

$$R(s^w) > R(s^l)$$
$$s^w \succ s^l$$
$$R(s^w) < R(s^r)$$
$$s^w \prec s^r$$

Problem 2: human judgments are noisy and miscalibrated!

Solution: instead of asking for direct ratings, ask for **pairwise comparisons**, which can be more reliable [Phelps et al., 2015; Clark et al., 2018]

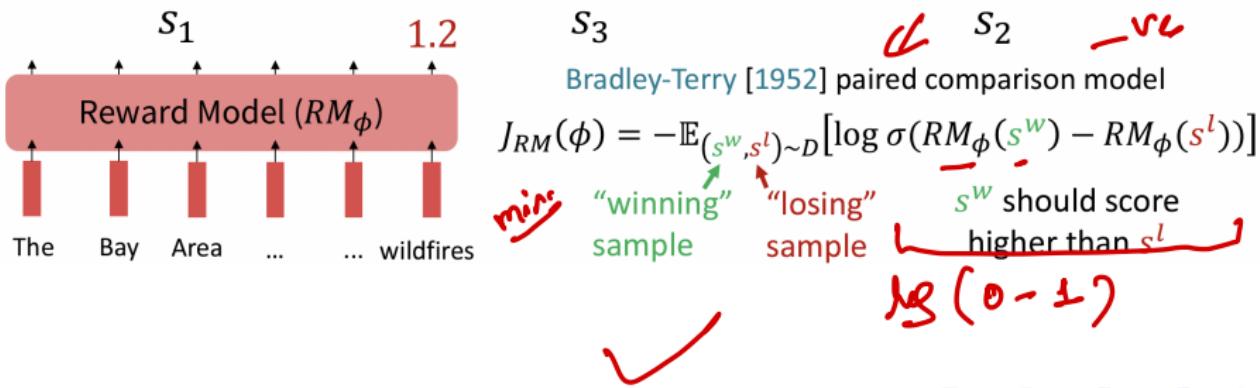
An earthquake hit
San Francisco.
There was minor
property damage,
but no injuries.

>

A 4.2 magnitude
earthquake hit
San Francisco,
resulting in
massive damage.

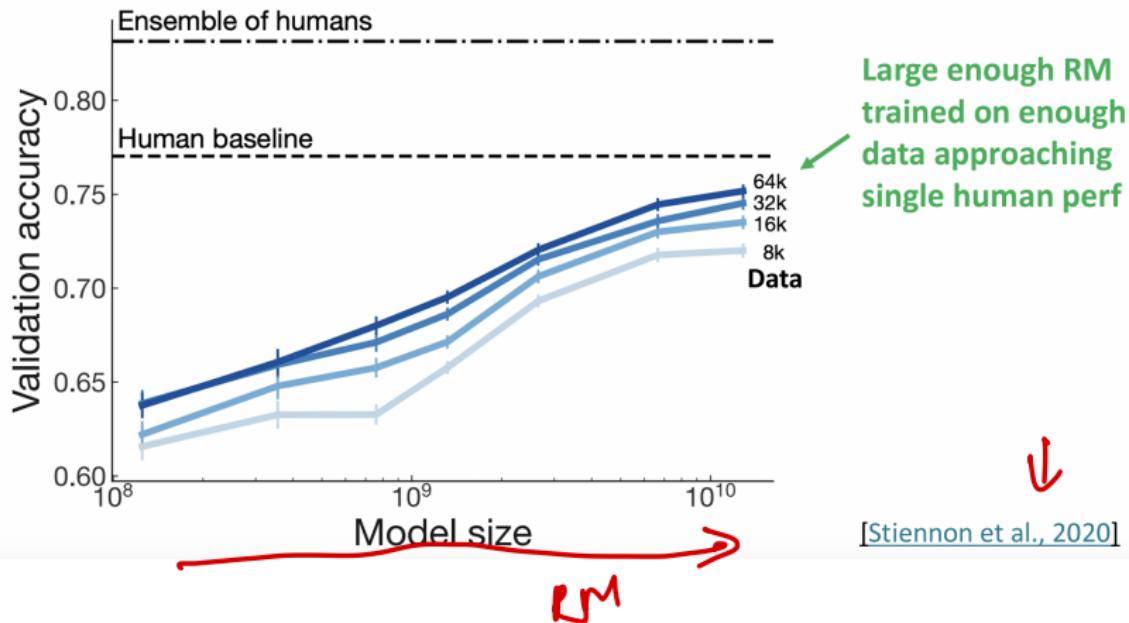
>

The Bay Area has
good weather but is
prone to
earthquakes and
wildfires.



Can a reward model replace a human?

Evaluate RM on predicting outcome of held-out human judgments



RLHF: Putting it all together

- Finally, we have everything we need:

- A pretrained (possibly instruction-finetuned) LM $p^{PT}(s)$
- A reward model $RM_\phi(s)$ that produces scalar rewards for LM outputs, trained on a dataset of human comparisons
- A method for optimizing LM parameters towards an arbitrary reward function.

- Now to do RLHF:

- Initialize a copy of the model $p_\theta^{RL}(s)$, with parameters θ we would like to optimize
- Optimize the following reward with RL:

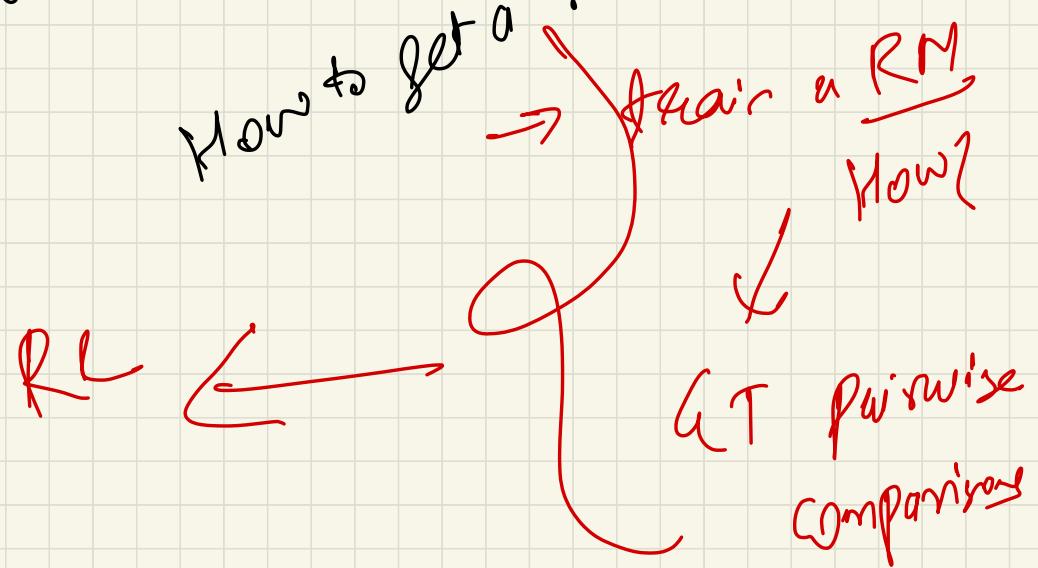
$$R(s) = RM_\phi(s) - \beta \log \left(\frac{p_\theta^{RL}(s)}{p^{PT}(s)} \right)$$

Pay a price when $p_\theta^{RL}(s) > p^{PT}(s)$

This is a penalty which prevents us from diverging too far from the pretrained model. In expectation, it is known as the Kullback-Leibler (KL) divergence between $p_\theta^{RL}(s)$ and $p^{PT}(s)$.

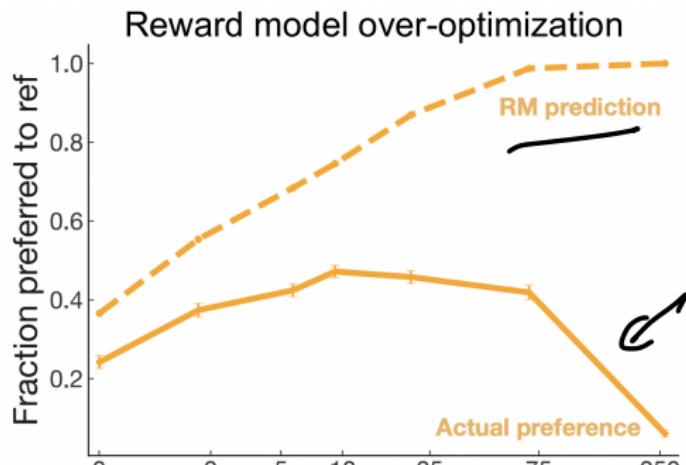
Arbitrary Reward Models

Action: next token predictions
Policy is the model decided by reward



RLHF: use of KL divergence

Optimal θ



$$R(s) = \overbrace{RM_\phi(s)} - \beta \log \left(\frac{p_\theta^{RL}(s)}{p^{PT}(s)} \right)$$



InstructGPT: Scaling up RLHF

30k
tasks!

Step 1

Collect demonstration data,
and train a supervised policy.

A prompt is sampled from our prompt dataset.

Explain the moon landing to a 6 year old

A labeler demonstrates the desired output behavior.

Some people went to the moon...

This data is used to fine-tune GPT-3 with supervised learning.

SFT

✍



Step 2

Collect comparison data,
and train a reward model.

A prompt and several model outputs are sampled.

Explain the moon landing to a 6 year old

A Explain gravity...
B Explain war...

C Moon is natural satellite of...

D People went to the moon...

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

D > C > A = B

RM

✍

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

Write a story about frogs

The policy generates an output.

PPO

Once upon a time...

The reward model calculates a reward for the output.

RM

r_k

[Ouyang et al., 2022]

InstructGPT: Scaling up RLHF

Tasks collected from labelers:

- **Plain:** We simply ask the labelers to come up with an arbitrary task, while ensuring the tasks had sufficient diversity.
- **Few-shot:** We ask the labelers to come up with an instruction, and multiple query/response pairs for that instruction.
- **User-based:** We had a number of use-cases stated in waitlist applications to the OpenAI API. We asked labelers to come up with prompts corresponding to these use cases.

Use-case	Prompt
Brainstorming	List five ideas for how to regain enthusiasm for my career
Generation	Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.

InstructGPT

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION GPT-3

Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.

InstructGPT

People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.

InstructGPT

PROMPT *Write a short poem about a wise frog.*

COMPLETION GPT-3

Write a short story in which a character has two different names.

Write a short story in which you try to get something back that you have lost.

Write a short story in which a character has a bad dream.

InstructGPT

The frog is a symbol of wisdom
He knows all the secrets of the world
He is a master of disguise
And a great teacher of life
He is a symbol of transformation
And the bringer of change
He is the frog who has seen it all
And knows the meaning of it all

ChatGPT: Optimizing Language Models for Dialogue

Note: OpenAI (and similar companies) are keeping more details secret about ChatGPT training (including data, training parameters, model size)—perhaps to keep a competitive edge...

Methods

We trained this model using Reinforcement Learning from Human Feedback (RLHF), using the same methods as InstructGPT, but with slight differences in the data collection setup. We trained an initial model using supervised fine-tuning: human AI trainers provided conversations in which they played both sides—the user and an AI assistant. We gave the trainers access to model-written suggestions to help them compose their responses. We mixed this new dialogue dataset with the InstructGPT dataset, which we transformed into a dialogue format.

(Instruction finetuning!)

ChatGPT: Optimizing Language Models for Dialogue

Note: OpenAI (and similar companies) are keeping more details secret about ChatGPT training (including data, training parameters, model size)—perhaps to keep a competitive edge...

Methods

To create a reward model for reinforcement learning, we needed to collect comparison data, which consisted of two or more model responses ranked by quality. To collect this data, we took conversations that AI trainers had with the chatbot. We randomly selected a model-written message, sampled several alternative completions, and had AI trainers rank them. Using these reward models, we can fine-tune the model using Proximal Policy Optimization. We performed several iterations of this process.

(RLHF!)

Removing 'RL' from RLHF

Neurips'23 Best Paper Review Up

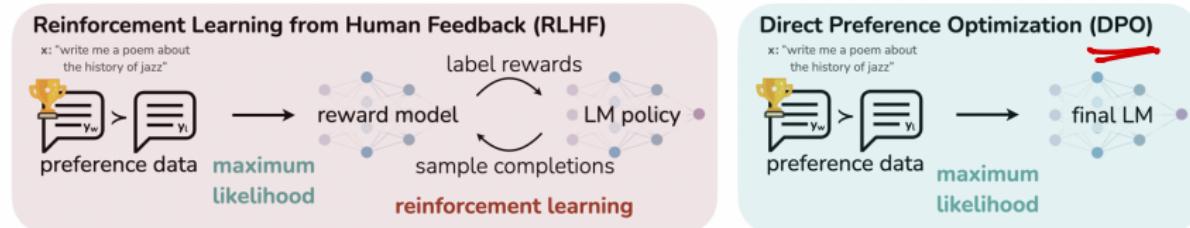


Figure 1: **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, fitting an *implicit* reward model whose corresponding optimal policy can be extracted in closed form.

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

Annotations in red:

- A bracket under the equation is labeled prompt, winner, and loser.
- A bracket on the right side of the equation is labeled π_θ : DPO and π_{ref} : PR.

How to use LLMs

