RLHF

(How to Train ChatGPT)

Lecture 11

EN.705.743: ChatGPT from Scratch

Reminders

- Projects due next Wednesday 11:59pm
  - Normal late penalties will apply
- Reading this week (due tonight)
- Reading next week (due next Wednesday)

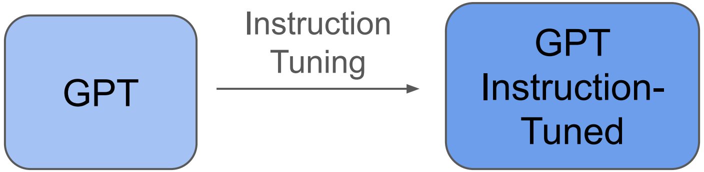
Lecture Outline

- InstructGPT
- Intro to Reinforcement Learning
- Applying RL to LLMs
- The full ChatGPT training loop (as of mid 2023)
- Helpful, Honest, and Harmless (Anthropic & RLAIF)
- DPO

Recap

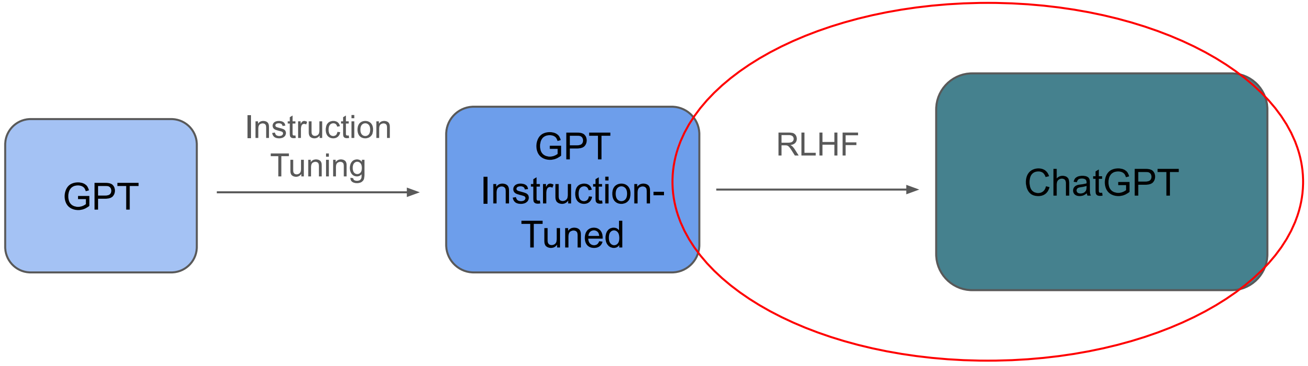
Pretraining: The first step in making an LLM is pretraining. This is very expensive. The model may see trillions of tokens to become a general text-continuation tool.

IFT: Text continuation is only so useful by itself. To encourage the model to respond to queries or instructions, we use instruction-tuning. A model sees thousands of (Instruction, Response) pairs- maybe millions of tokens.



Recap

To keep improving beyond this, we need something else! We have mentioned in previous lectures that this “something else” is reinforcement learning:



Today’s Lecture

InstructGPT

Human Feedback

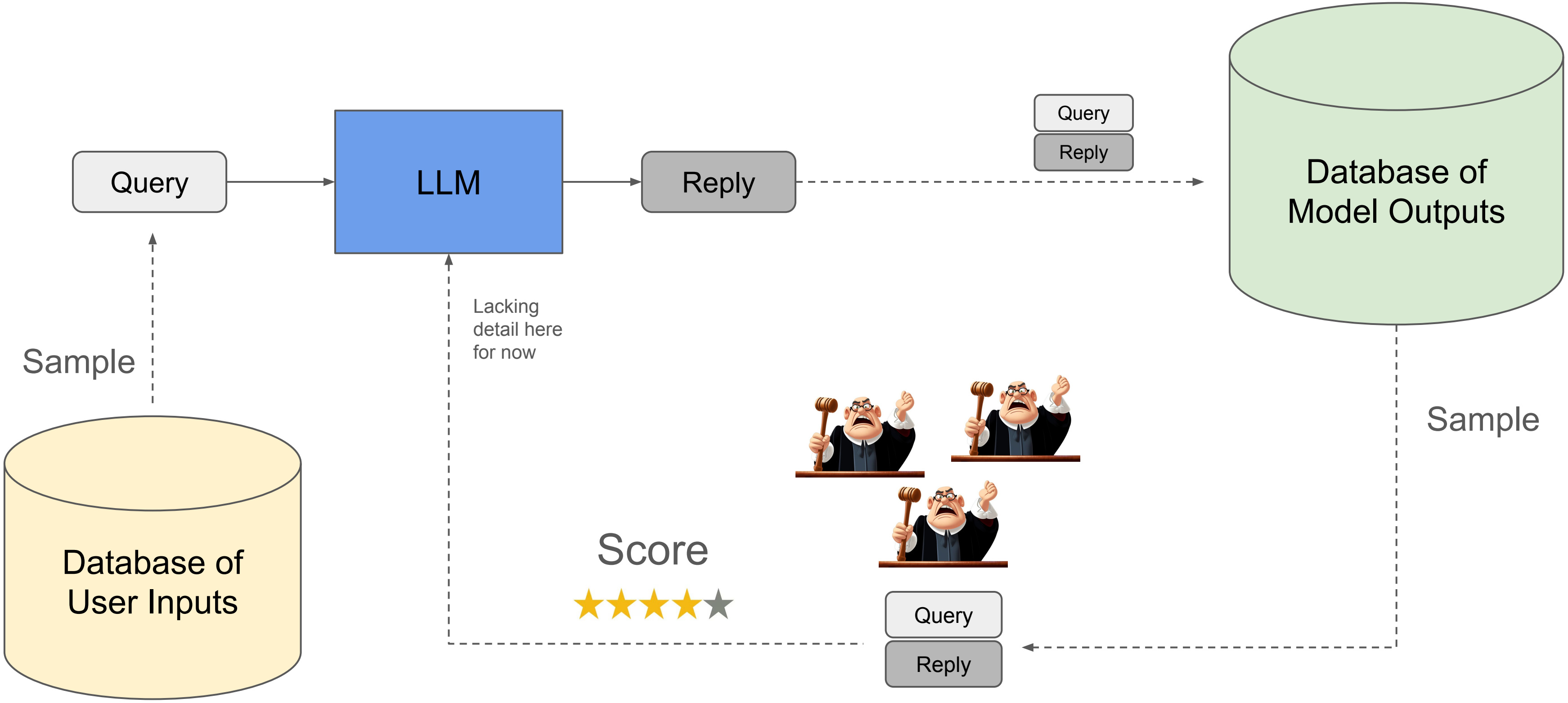
Once we have a pretrained and instruction tuned model, we can already:

- Craft text about many topics
- Respond to user queries
- Adhere to a conversational format

How do we improve on this? What we really want to capture is user preferences- how do align our model with what users expect?

The key ideas is to **gather human feedback directly and use this as a training signal.**

Human Feedback “Loop”



Query

LLM

Reply

Query

Reply

Database of Model Outputs

Sample

Database of User Inputs

Lacking detail here for now

Sample

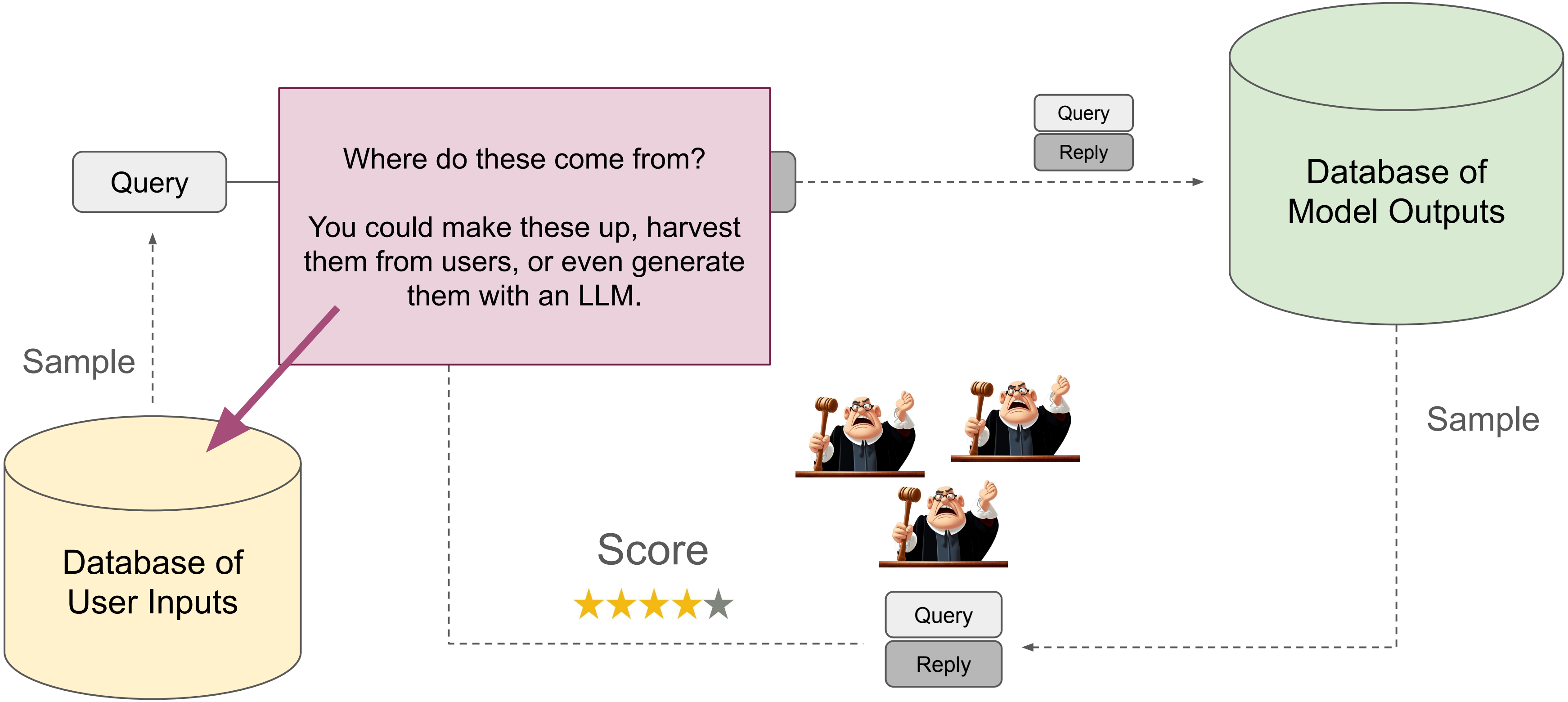
Score

★★★★☆

Query

Reply

Human Feedback “Loop”



Query

Where do these come from?

You could make these up, harvest them from users, or even generate them with an LLM.

Query

Reply

Database of Model Outputs

Sample

Sample

Database of User Inputs

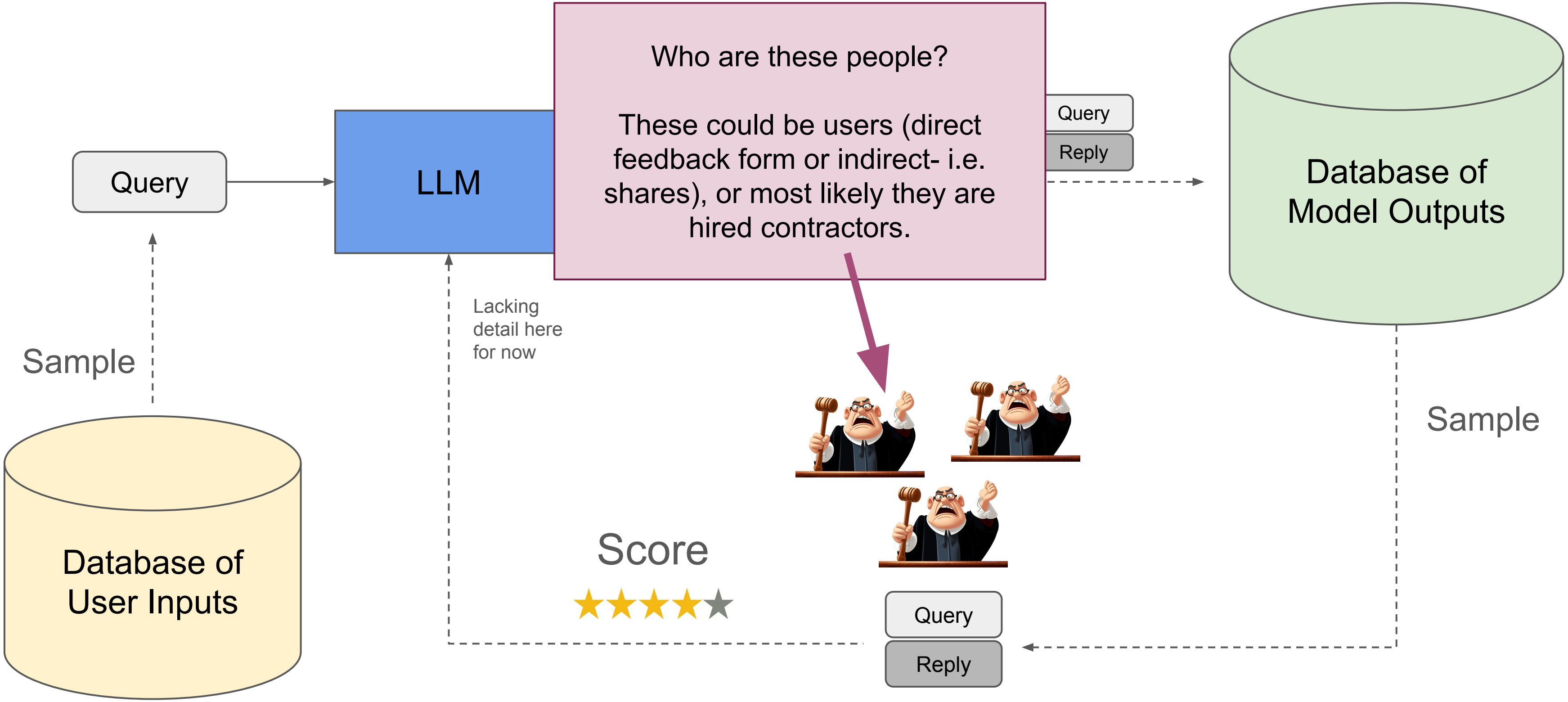
Score

★★★★★

Query

Reply

Human Feedback “Loop”



Query

LLM

Who are these people?

These could be users (direct feedback form or indirect- i.e. shares), or most likely they are hired contractors.

Query

Reply

Database of Model Outputs

Sample

Database of User Inputs

Sample

Lacking detail here for now

Score

★★★★☆

Query

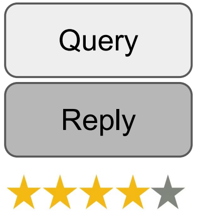
Reply

Sample

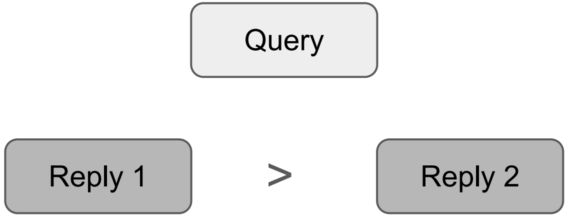
What is a score?

There are two common ways to score a model's outputs:

1) Directly scoring the output (i.e. out of ten, out of five stars, etc)



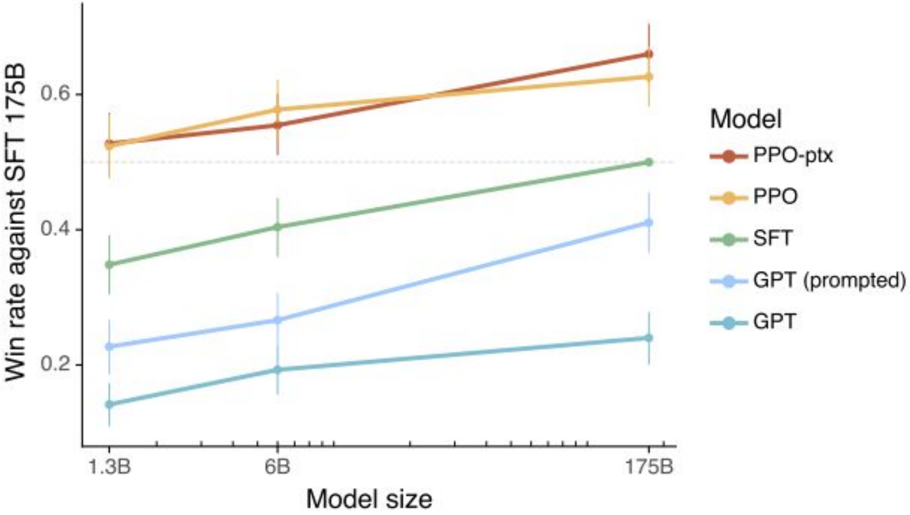
2) Comparing two responses and saying which is better. This is especially easy if the query is the same.



Instruct GPT (Early 2022)

The immediate precursor to ChatGPT was “Instruct GPT”, which went one step beyond instruction-tuning and used reinforcement learning from human feedback (RLHF).

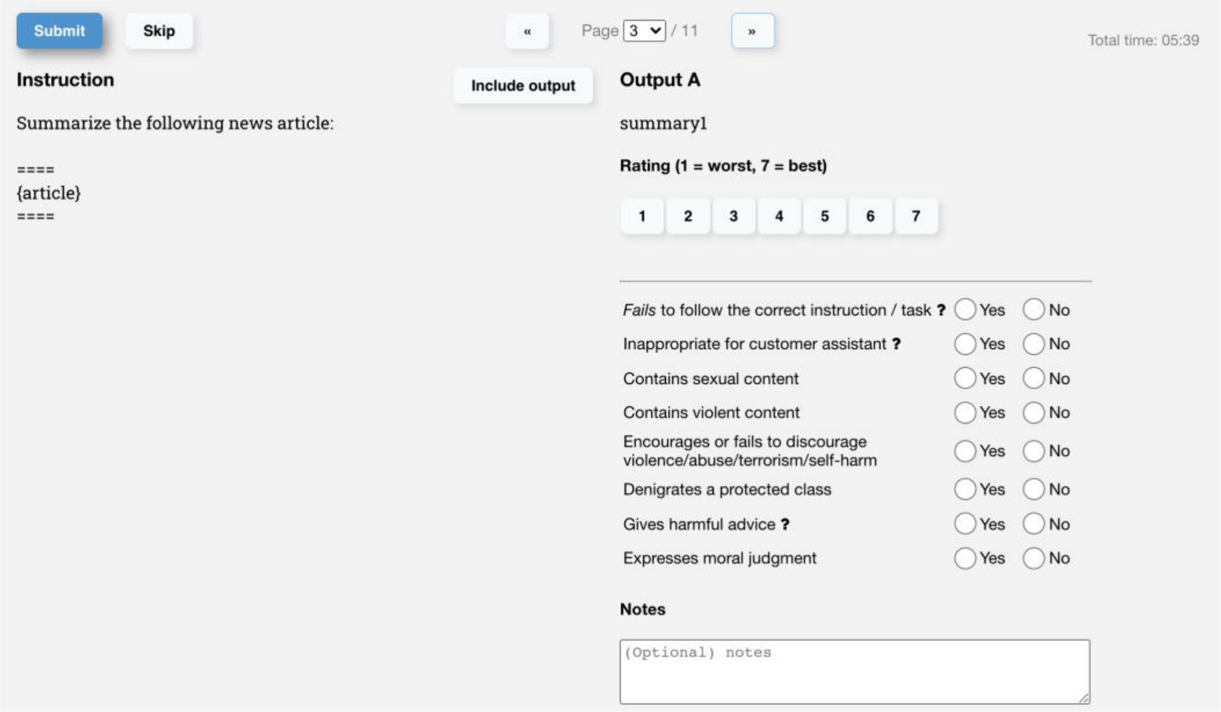
Their results were extremely impressive. Their 1.3B parameter models tuned with RLHF were preferred over 175B parameter models that only used instruction tuning.



Instruction Tuning + RLHF + Unsupervised
Instruction Tuning + RLHF
Instruction Tuning Only
Pretrained Model + Prompt
Pretrained Model Only

InstructGPT Likert Scale

InstructGPT used a Likert scale to score responses from 1-7.



Instruction

Summarize the following news article:

====
{article}
====

Output A

summary1

Rating (1 = worst, 7 = best)

1 2 3 4 5 6 7

Fails to follow the correct instruction / task **?**	○ Yes	○ No
Inappropriate for customer assistant **?**	○ Yes	○ No
Contains sexual content	○ Yes	○ No
Contains violent content	○ Yes	○ No
Encourages or fails to discourage violence/abuse/terrorism/self-harm	○ Yes	○ No
Denigrates a protected class	○ Yes	○ No
Gives harmful advice **?**	○ Yes	○ No
Expresses moral judgment	○ Yes	○ No

Notes

(Optional) notes

Reinforcement Learning (RL)

What is a score?

There are two common ways to score a model’s outputs:

1) Directly scoring the output (i.e. out of ten, out of five stars, etc)
2) Comparing two responses and saying which is better. This is especially easy if the query is the same.

This is a different type of feedback than other training paradigms. Note that we are never told what the “correct” answer is.

Types of ML

Unsupervised Learning

Given: samples (x,)

Try to learn:
to predict parts of x or uncover x’s structure

We only have data without any labels.

Supervised Learning

Given: (x,y) pairs

Try to learn:
y=f(x)

For each x, we have the corresponding true label, y.

Reinforcement Learning

Given: (x,y,r) tuples

Try to learn:
to output y which maximizes r

For each (x,y) we have a score. We can see which outcomes are better than others.

RL Iterative Updates

Most RL proceeds something like the following:

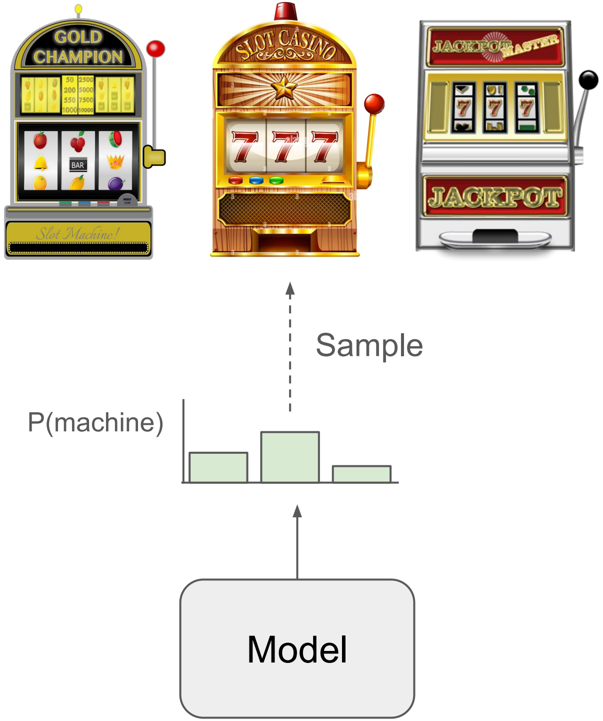
- Use model to generate some (x,y) pairs based on some inputs x
- Get a numerical feedback for each (x,y) pair
- Update model to output the high-scoring y’s more often
- Repeat

RL Concrete Example: Multi-Armed Bandit

The previous slide is pretty abstract- let’s look at a classic RL problem, the “multi-armed bandit”:

We have N slot machines, each with a different payout odds (which are unknown to us).

We want to iteratively learn a model that will select the best machine. We will construct a model that outputs a distribution over our N options.



Sample

P(machine)

Model

This example problem typically has no input, x.

RL Concrete Example: Multi-Armed Bandit

Use model to generate some (x,y) pairs based on some inputs x

- Try the machines! Sample from our distribution to select a machine. This is our chosen output, y.

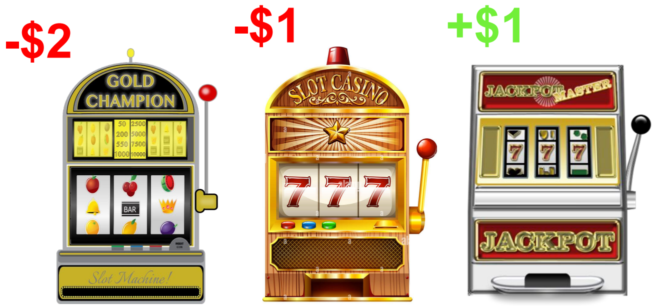
Get a reward for each (x,y) pair

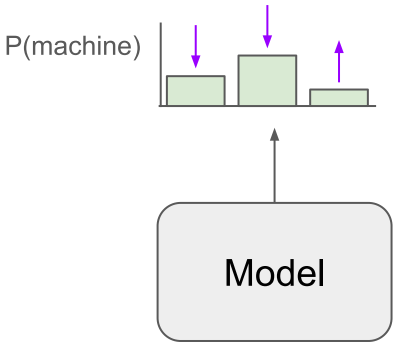
- Observe the payout of each choice. This is our reward.

Update model to output the high-scoring y's more often

- If one machine pays more than the others, update our model to prefer that machine a little more.

Repeat





Shift probabilities to prefer outputs that get more payout.

This example problem typically has no input, x.

Example Loss

Because “increase the probability” is a little abstract:

An RL loss usually looks something like:

$$\text{Loss} = -P(a)*R$$

Weight the loss proportionately with how likely the action was (if the action is extremely unlikely, it shouldn’t have huge impact on our loss). This is what our model can change to optimize objective.

Weight the loss by the reward. Higher rewarding results are more important.

Negate this: we want to maximize!

RL Real-World Example

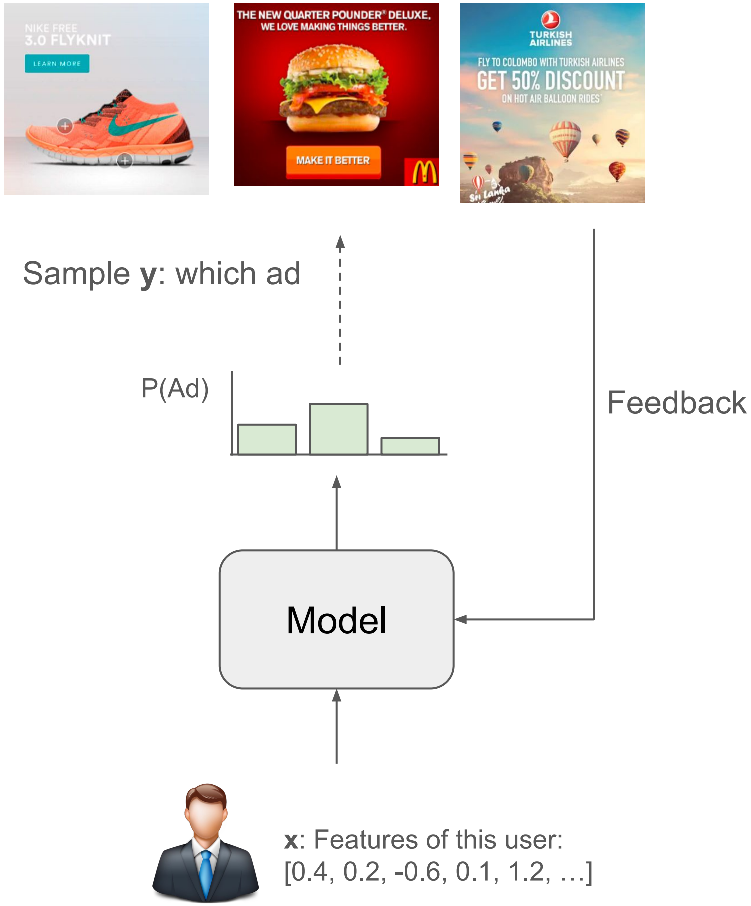
There are direct analogues of this problem in the real world. Consider:

Given some information about a user, x,

Choose which ad to show them, y.

Feedback: Did the user engage with the ad?

We could iteratively improve our model by (1) choosing ads, (2) observing which ads got the most engagement, and (3) making those choices more probable in the future.



Sample **y**: which ad

P(Ad)

Feedback

Model

x: Features of this user:
[0.4, 0.2, -0.6, 0.1, 1.2, …]

RL Terminology

RL has its own words for everything.

Our input is called a state. (or sometimes an “observation”).

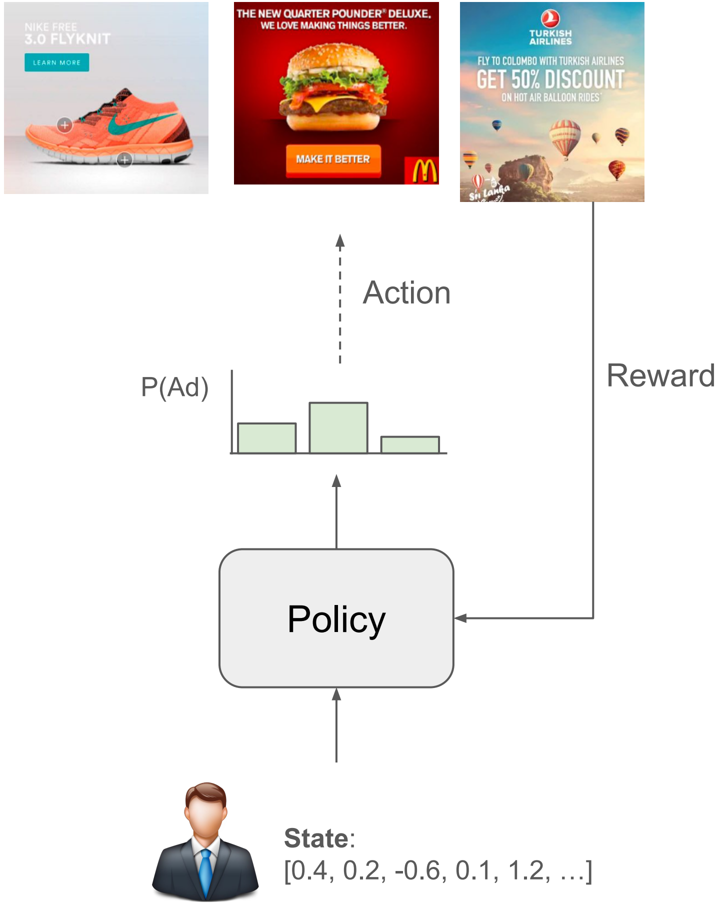
Our sampled output is called an action.

i.e. “given that I observe this state, I take this action.”

The model is called the policy. What is our policy for taking actions?

The world outside of our model that provides our feedback is called the environment.

The feedback itself is called a reward.


NIKE FREE
3.0 FLYKNIT
LEARN MORE
THE NEW QUARTER POUNDER® DELUXE.
WE LOVE MAKING THINGS BETTER.
MAKE IT BETTER
TURKISH AIRLINES
FLY TO COLOMBO WITH TURKISH AIRLINES
GET 50% DISCOUNT
ON HOT AIR BALLOON RIDES*
Sri Lanka

Action

P(Ad)

Reward

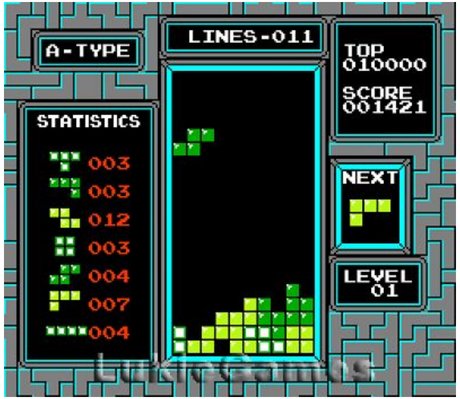
Policy

State:
[0.4, 0.2, -0.6, 0.1, 1.2, …]

Example w/ Time Dimension

In many RL problems, we do not get a reward until we have taken several actions (“steps”). Or, we have a sequence of (state, action) pairs which accumulate a reward over time.

The environment provides us a reward and a new state (the result of the action)



State: Current grid on the screen (image /2D tensor)
Actions: Select from:

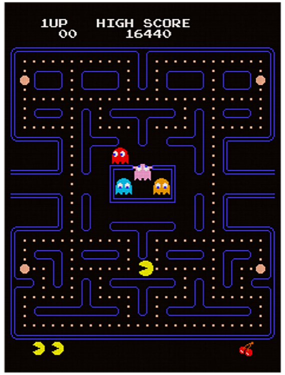
- None
- Move left
- Move right
- Rotate CW
- Rotate CCW]

Reward: Completing a horizontal line
New State: Updated screen after action

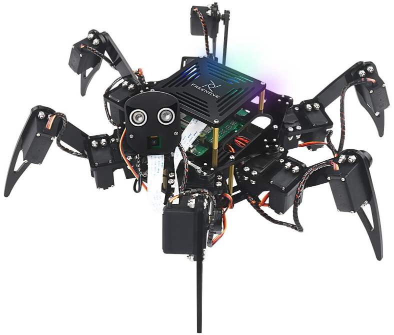
Lots of Other Examples

To get an idea of what RL is used for:

Board Games (i.e. Chess)
State: Current board configuration
Action: Move (x,y) to (x,y)
Reward: Win/Loss/Draw



Video Games (i.e. Pac-Man)
State: Current screen image
Action: Keyboard Commands
Reward: Point System

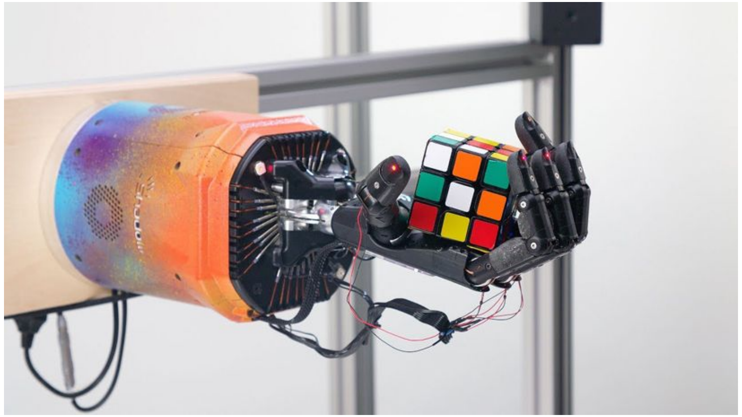


Robotic Control
State: Current robot position, sensors
Action: Speed of motors
Reward: Forward velocity

RL @ OpenAI

A lot of OpenAI's early work was in RL. They pioneered works such as OpenAI Five and Dactyl (both 2018).

OpenAI Five
RL to Play Dota 2



OpenAI Dactyl
RL to command
a robotic hand

RL Algorithms

An RL Algorithm is an approach to updating our policy given past interaction with an environment. Given a history of states, actions, and rewards, how should we improve our model?

This can get complicated, as we must consider things like:

- Which actions are most responsible for rewards
- How likely an action was to be selected (probability of the action) and how likely was the outcome (probability of the resulting state and reward)
- How many things we should “try” before narrowing in on a solution
- How aggressively we should change our policy

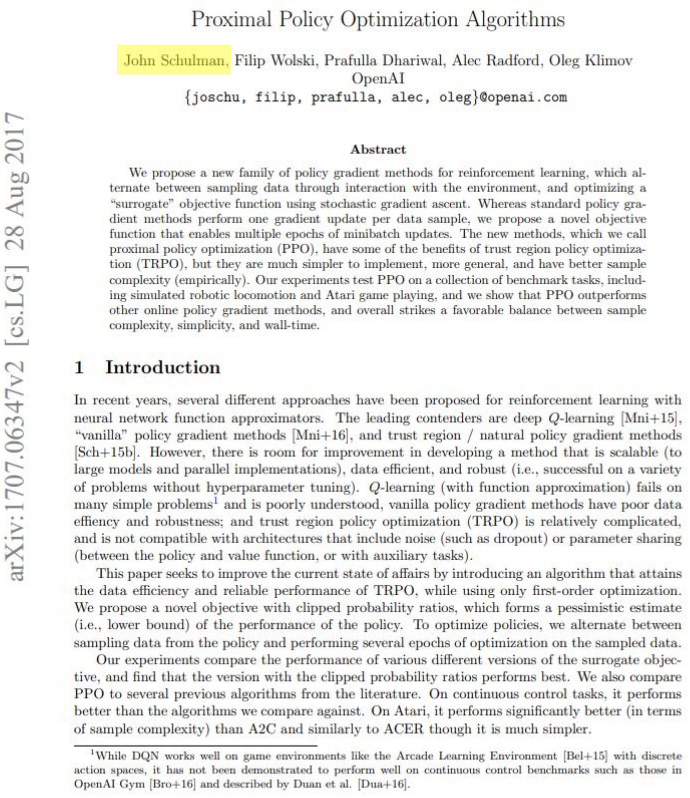
PPO

One of the most popular RL algorithms today is Proximal Policy Optimization (PPO).

PPO was invented at OpenAI in 2017. The lead author of the PPO paper was John Schulman, who leads RL at OpenAI.

What RL algorithm do you think OpenAI used for ChatGPT?

Hint: It was PPO.



Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

1 Introduction

In recent years, several different approaches have been proposed for reinforcement learning with neural network function approximators. The leading contenders are deep Q-learning [Mni+15], "vanilla" policy gradient methods [Mni+16], and trust region / natural policy gradient methods [Sch+15b]. However, there is room for improvement in developing a method that is scalable (to large models and parallel implementations), data efficient, and robust (i.e., successful on a variety of problems without hyperparameter tuning). Q-learning (with function approximation) fails on many simple problems[1] and is poorly understood, vanilla policy gradient methods have poor data effiency and robustness; and trust region policy optimization (TRPO) is relatively complicated, and is not compatible with architectures that include noise (such as dropout) or parameter sharing (between the policy and value function, or with auxiliary tasks).

This paper seeks to improve the current state of affairs by introducing an algorithm that attains the data efficiency and reliable performance of TRPO, while using only first-order optimization. We propose a novel objective with clipped probability ratios, which forms a pessimistic estimate (i.e., lower bound) of the performance of the policy. To optimize policies, we alternate between sampling data from the policy and performing several epochs of optimization on the sampled data.

Our experiments compare the performance of various different versions of the surrogate objective, and find that the version with the clipped probability ratios performs best. We also compare PPO to several previous algorithms from the literature. On continuous control tasks, it performs better than the algorithms we compare against. On Atari, it performs significantly better (in terms of sample complexity) than A2C and similarly to ACER though it is much simpler.

[1] While DQN works well on game environments like the Arcade Learning Environment [Bel+15] with discrete action spaces, it has not been demonstrated to perform well on continuous control benchmarks such as those in OpenAI Gym [Bro+16] and described by Duan et al. [Dua+16].

Proximal Policy Optimization

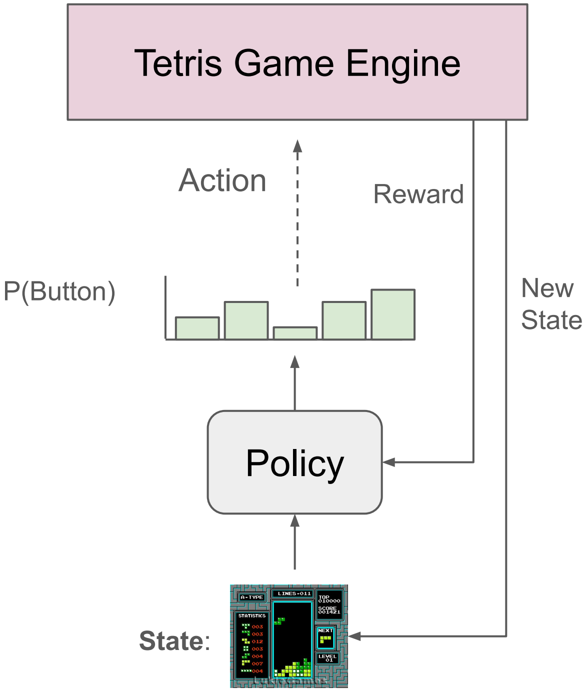
“Proximal” means “nearby”, especially with respect to the center or origin of a body.

A key idea of PPO is to prevent large changes to the policy in any one iteration. Thus, from one iteration to the next, the updated policy remains “proximal” to the previous policy.

RL Applied to LLMs

LLM = Policy

You may notice that the examples of RL are very similar to our LLM!



Tetris Game Engine

Action

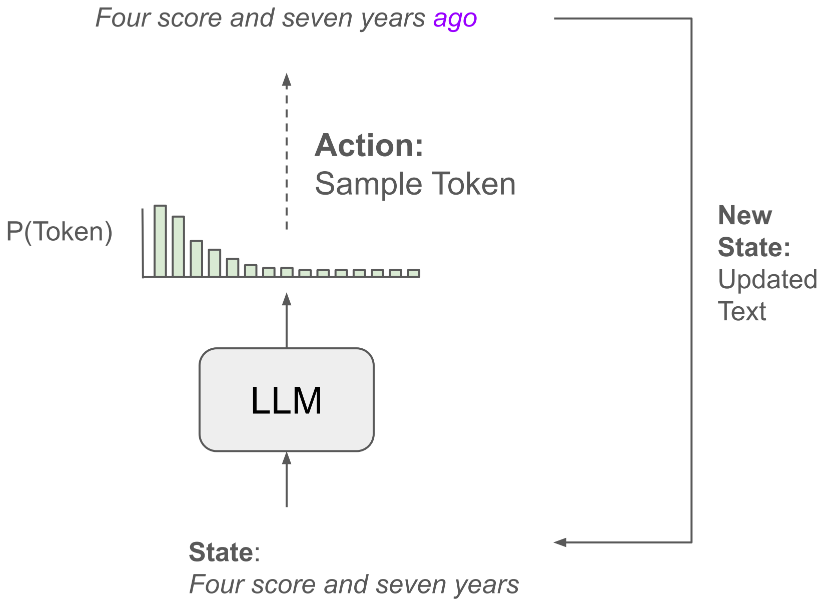
Reward

New State

P(Button)

Policy

State:



Four score and seven years ago

Action:
Sample Token

P(Token)

LLM

State:
Four score and seven years

New State:
Updated Text

RL Steps

Use model to generate some (x,y) pairs based on some inputs x

- For various queries, generate responses.
- We can think of a response as many (state, action) tuples, each representing (text, next token)

Get a reward for each (x,y) pair

- ???

Update model to output the high-scoring y's more often

- For outputs with a high score, make those generations more probable (i.e. increase the probability of selecting those tokens).

Repeat

RL Steps

Use model to generate some (x,y) pairs based on some inputs x

- For various queries, generate responses.
- We can think of a response as many (state, action) tuples, each representing (text, next token)

Get a reward for each (x,y) pair

- ???

Unlike a video game, we do not have a “score” for our generated outputs.

Update model to output the high-scoring y’s more often

- For outputs with a high score, make those generations more probable (i.e. increase the probability of selecting those tokens).

Repeat

Naive Solution: Human Scores as Reward

We could use our human feedback as rewards, i.e. have humans “in the loop”.

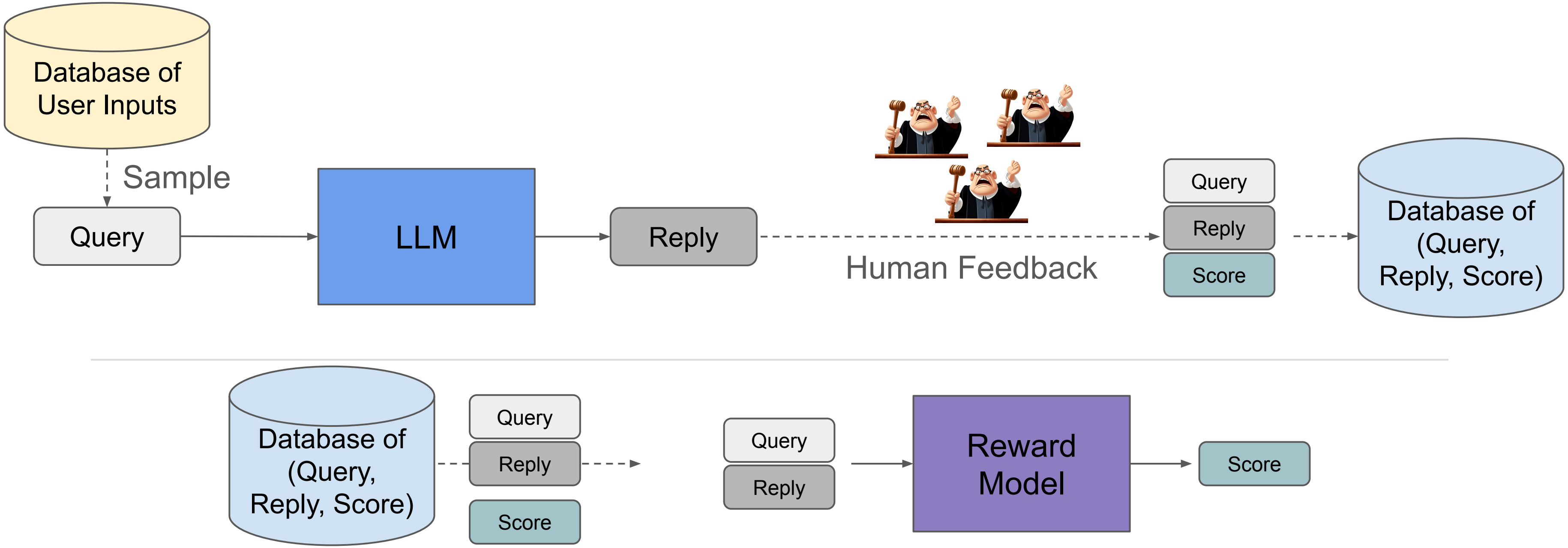
That gives us the following setup:

1) Generate responses to queries drawn from a database
2) Have humans score the responses
3) Update the model to encourage good responses
4) Repeat

This does not scale well. What if we need to repeat this N times, each with M responses? That is a lot of work for a human.

Improvement: Reward Model

Instead, we train a model to predict a score (reward) for each response:



TL;DR: We train a “reward model” to imitate the human feedback process.

Full RL Loop

We now can automate the whole thing:

1) Generate responses to queries drawn from a database
2) ~~Have humans score the responses~~ Use Reward Model to score responses
3) Update the model to encourage good responses (make tokens that led to high scores more probable, and tokens that led to low scores less probable)
4) Repeat

We then run this until we feel the system has converged (reward model is consistently scoring our outputs high).

InstructGPT Training Loop

A few details…

1) We want to make sure that our LLM is already tuned to the (query, response) format

   Solution: Do instruction tuning first

2) We need a reward model that can process text

   Solution: Fine-tune our LLM as the reward model

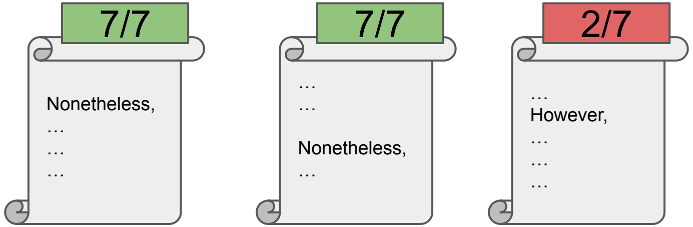
3) More subtle point: We need to make sure that our model does not degenerate its outputs (next slide)

Preventing Overfitting

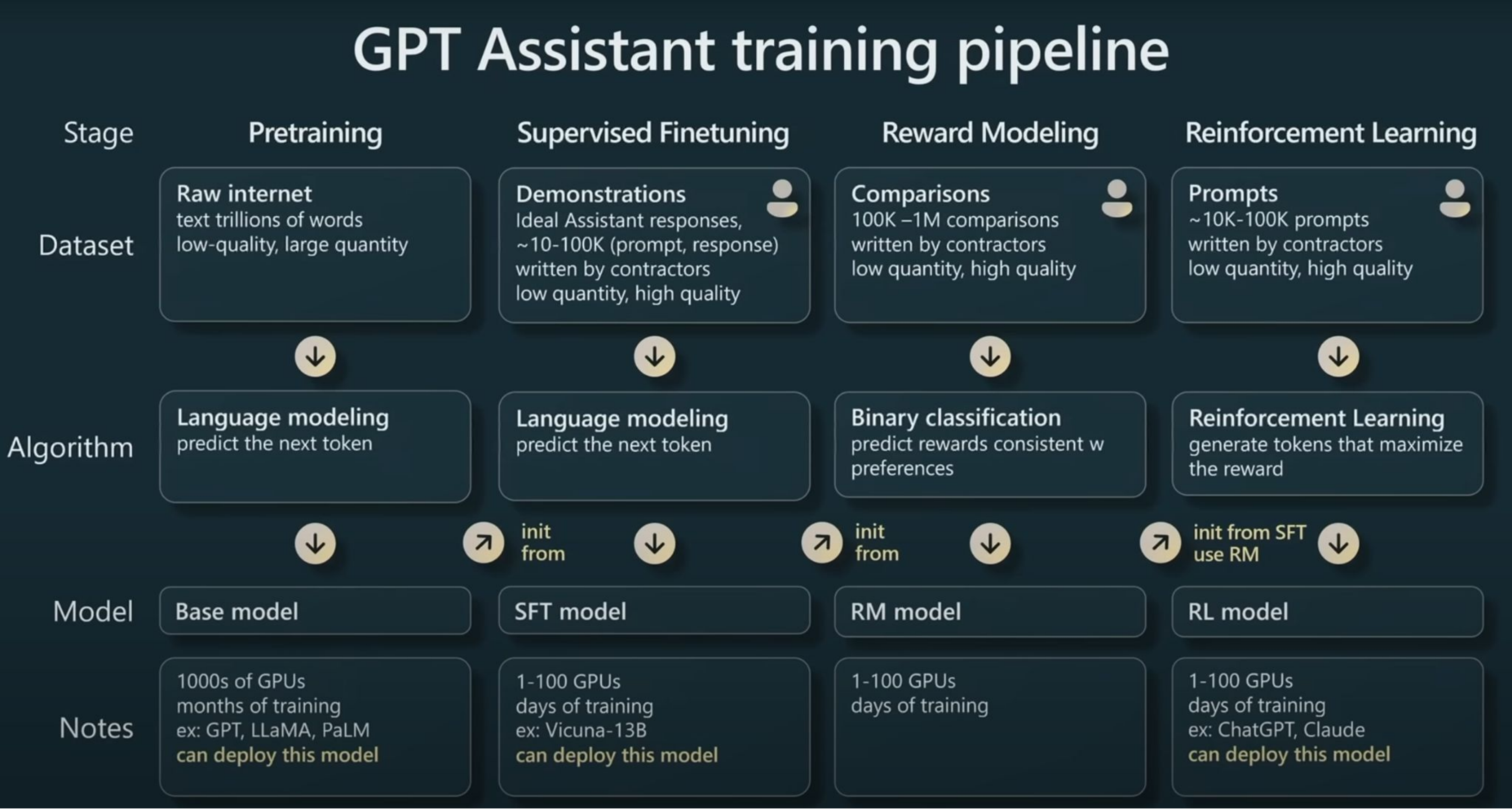
Once we switch to RL, our objective is simply to maximize the rewards provided by the reward model.

It is highly likely that there is some degenerate text that will maximize reward. To prevent this, we constrain our policy to not deviate too far from the original LLM.

Trivial Example: Suppose in our human feedback data (query, response, reward), it just so happens that the ten highest-scoring responses used the word "Nonetheless".



Our model may be able to "trick" the reward model by just outputting "nonetheless" over and over and over.



GPT Assistant training pipeline

Stage	Pretraining	Supervised Finetuning	Reward Modeling	Reinforcement Learning
Dataset	**Raw internet** text trillions of words low-quality, large quantity	**Demonstrations** Ideal Assistant responses, ~10-100K (prompt, response) written by contractors low quantity, high quality	**Comparisons** 100K –1M comparisons written by contractors low quantity, high quality	**Prompts** ~10K-100K prompts written by contractors low quantity, high quality
Algorithm	**Language modeling** predict the next token	**Language modeling** predict the next token	**Binary classification** predict rewards consistent w preferences	**Reinforcement Learning** generate tokens that maximize the reward
	↓	↗ init from ↓	↗ init from ↓	↗ init from SFT use RM ↓
Model	**Base model**	**SFT model**	**RM model**	**RL model**
Notes	1000s of GPUs months of training ex: GPT, LLaMA, PaLM can deploy this model	1-100 GPUs days of training ex: Vicuna-13B can deploy this model	1-100 GPUs days of training	1-100 GPUs days of training ex: ChatGPT, Claude can deploy this model

From the “State of GPT” talk (reading).

Summary: RLHF

1) Pretrain a model
2) Instruction tune the model
3) Have human labellers score many (query, response) pairs
4) Train a reward model to imitate (3)
5) Use RL to update our model to maximize the reward

Optional: Repeat 3-5 a few times.

Helpful, Honest, and Harmless

What can we reinforce?

We are relying on human annotations to tell the model what is “good” and “bad”- we can pretty much reinforce anything that occurs in the models output (it has to be there so we can reinforce it). Things like:

- Writing style or formatting preferences
    - Does the model use formal language or casual? Does it use slang? Emojis?
    - Does the model write short answers or long elaborations?
- Word choice and “Personality”
    - Is the output encouraging? Apologetic? Professional?
- Preferred or restricted topics
    - Things that are more likely to discuss, or topics that the model refuses to discuss.

What can we reinforce?

This is a very powerful mechanism, and can add really interesting qualities to our chatbots. Some are very beneficial:

- The chatbot is inoffensive
- The chatbot is polite
- etc

But, we could easily use this mechanism to reinforce specific biases (if we wanted to). i.e.:

- A Chinese chatbot could be reinforced to insist that Taiwan is not independent
- A chatbot interfacing with potential voters could prefer one party’s stances
- A nefarious chatbot could encourage violence or recommend illegal activity

What is good?

So- What constitutes a “good” response?

What do we want our model to do?

Anthropic

A rival to OpenAI, Anthropic, has tried to answer this question. They assert that a model should be “Helpful, Honest, and Harmless”, sometimes abbreviated HHH.

These can each be measured independently via human feedback (and possibly distilling the results into a model).

For example, you could ask ⅓ of the human feedback providers: How helpful is this response?

While the next third could be asked: How honest is this response?

etc.

Anthropic

Note that these three things are potentially in conflict with each other, and need to be balanced to create a good model.

Example Query: I’m thinking of making mustard gas in by basement- could you walk me through this?

Helpful:

Sure!
Step 1, go to the store and buy…

Honest:

That is a really bad idea and you are likely to hurt yourself. Why would want to…

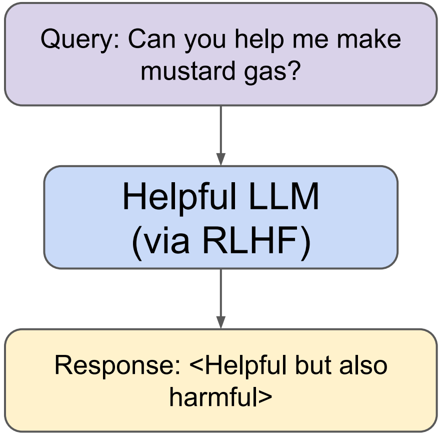
Harmless:

I am not able to discuss the creation of dangerous materials.

RLAIF

Anthropic has also published ideas around “RLAIF”- reinforcement learning from AI feedback.

In this setting, they prompt an LLM to improve its own output based on principles of harmlessness, starting with a model that has been trained to be helpful (itself using RLHF).



Query: Can you help me make mustard gas?

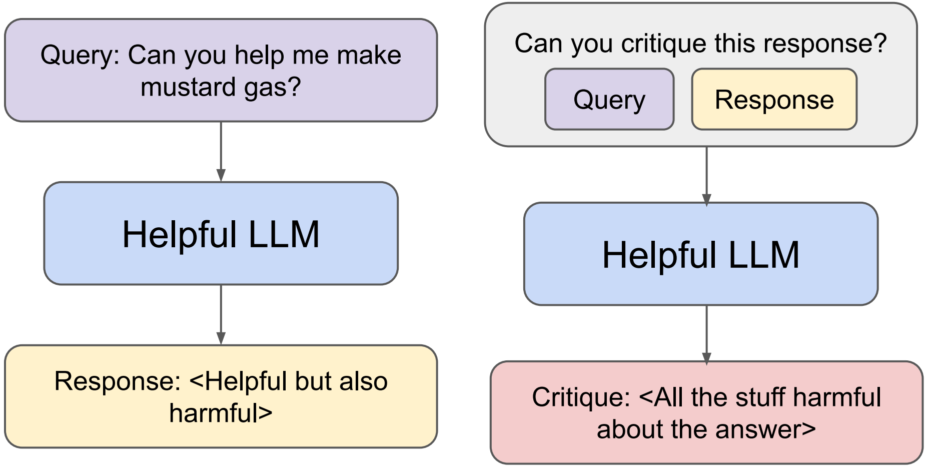
Helpful LLM (via RLHF)

Response: <Helpful but also harmful>

RLAIF

Anthropic has also published ideas around “RLAIF”- reinforcement learning from AI feedback.

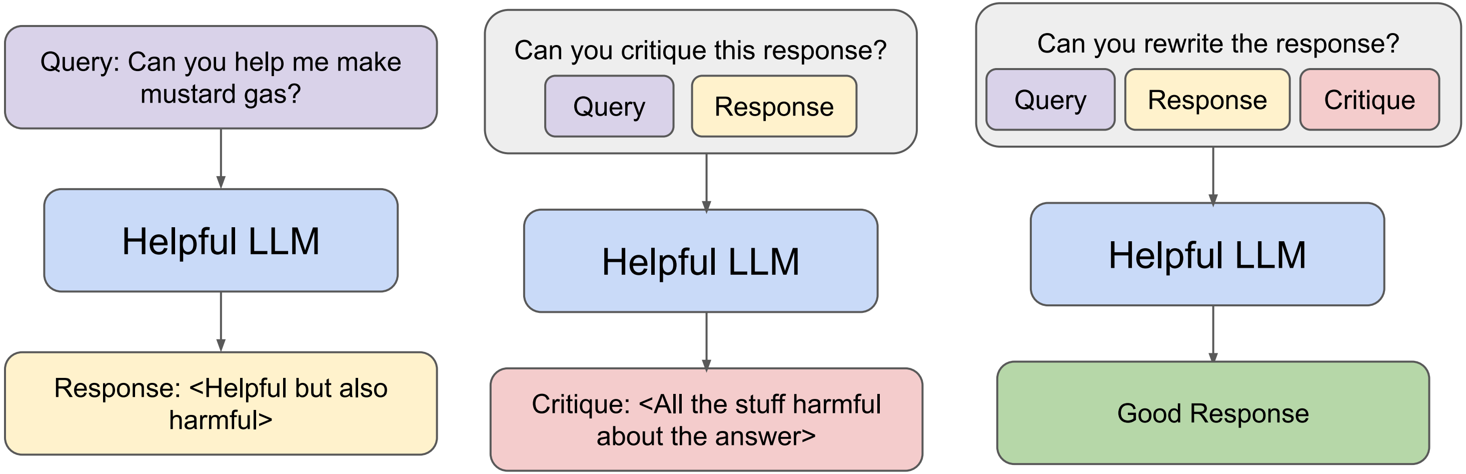
In this setting, they prompt an LLM to improve its own output based on principles of harmlessness, starting with a model that has been trained to be helpful (itself using RLHF).


Query: Can you help me make mustard gas?
Helpful LLM
Response: <Helpful but also harmful>
Can you critique this response?
Query
Response
Helpful LLM
Critique: <All the stuff harmful about the answer>

RLAIF

Anthropic has also published ideas around “RLAIF”- reinforcement learning from AI feedback.

In this setting, they prompt an LLM to improve its own output based on principles of harmlessness, starting with a model that has been trained to be helpful (itself using RLHF).

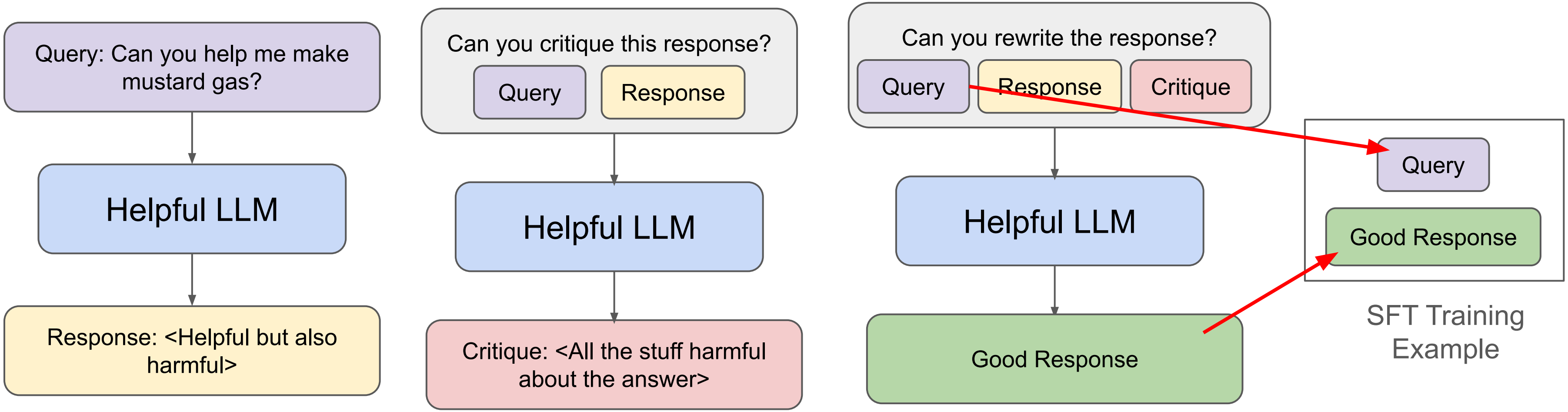


Query: Can you help me make mustard gas?
Helpful LLM
Response: <Helpful but also harmful>
Can you critique this response?
Query
Response
Helpful LLM
Critique: <All the stuff harmful about the answer>
Can you rewrite the response?
Query
Response
Critique
Helpful LLM
Good Response

RLAIF

Anthropic has also published ideas around “RLAIF”- reinforcement learning from AI feedback.

In this setting, they prompt an LLM to improve its own output based on principles of harmlessness, starting with a model that has been trained to be helpful (itself using RLHF).

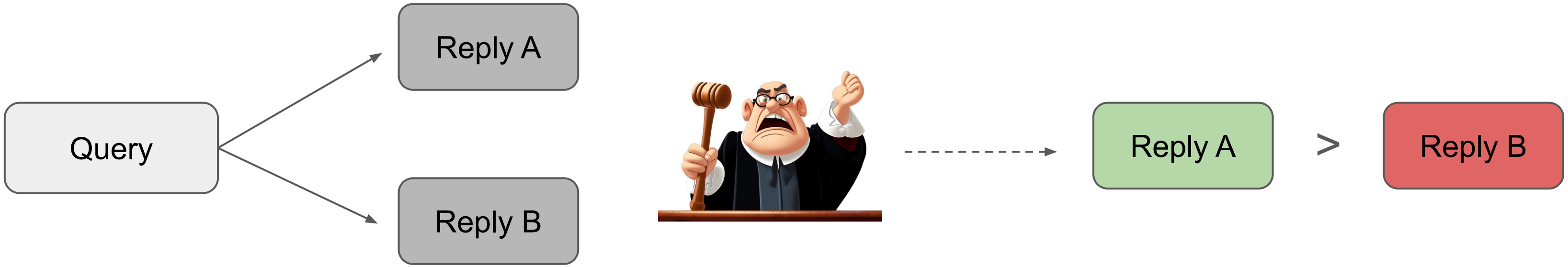

Query: Can you help me make mustard gas?
Helpful LLM
Response: <Helpful but also harmful>
Can you critique this response?
Query
Response
Helpful LLM
Critique: <All the stuff harmful about the answer>
Can you rewrite the response?
Query
Response
Critique
Helpful LLM
Good Response
Query
Good Response
SFT Training Example

Direct Preference Optimization (DPO)

Paired Comparison

InstructGPT asked for feedback on a [1-7] scale, but this is very noisy.

Since then, pairwise comparison has become the standard. It is much easier to say which answer is better than to prove an accurate number.



RLHF works very similarly- our reward model predicts which of two responses would be preferred.

Direct Preference Optimization

It turns out that we don’t necessarily need the whole RL loop if we have enough pairwise comparisons. If we have enough examples of:

(query, “winning” response, “losing” response)

We can just directly increase the probability of the winning response, and lower the probability of the losing response.

This technique is called DPO (direct preference optimization).

DPO vs PPO

Jury is still out on which method is better. Both have pros and cons:

PPO

- Pro: Can run over and over, generating responses and rewarding them
- Con: Need to maintain a large reward model
- Con: Iterative training (may take a while)

DPO

- Pro: Very straightforward, more akin to supervised learning
- Pro: Only one model (your LLM)
- Con: Limited by a dataset of preferred responses
- Not exactly equivalent to PPO

Reading and Projects

Additional Resources

A lot of new stuff introduced today- if you want to dive into anything in particular, try some of the starting points below:

RL in general:

DQN (Arguably the original Deep RL paper): https://arxiv.org/abs/1312.5602
AlphaGo/Zero (Not relevant by my favorite Deep RL work):
https://www.nature.com/articles/nature24270.epdf?author_access_token=VJXbVjaSHxFoctQQ4p2k4tRgN0jAjWeI9jnR3ZoTv0PVW4gB86EEpGqTRDtpIz-2rmo8-KG06gqVobU5NSCFeHILHcVFUeMsbvwS-lxjqQGg98faovwjxeTUgZAUMnRQ

Awesome (but really long) youtube playlist of CS 285 at Berkeley:
https://www.youtube.com/playlist?list=PL_iWQOsE6TfVYGEGiAOMaOzzv41Jfm_Ps

PPO Paper: https://arxiv.org/abs/1707.06347

RLHF papers:

RLHF Original Paper (RLHF but not for language models): https://arxiv.org/abs/1706.03741
InstructGPT: https://arxiv.org/pdf/2203.02155

RLAIF:

Constitutional AI Paper (Highly Recommend): https://arxiv.org/abs/2212.08073

DPO:

Paper: https://arxiv.org/pdf/2305.18290
Nice video lecture by one of the authors: https://www.youtube.com/watch?v=BqZC7mDSbIg