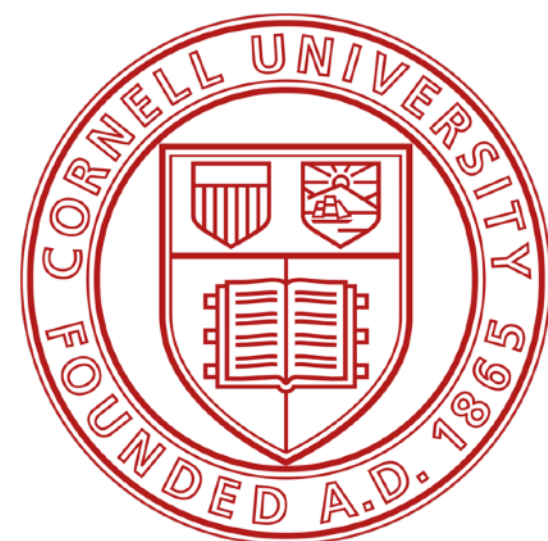


Lecture 19: Post-training

CS 5788: Introduction to Generative Models



Lots of slides from Yoav Artzi, LM class

Reminder: Midterm on Fri. 4/17, 2-5pm

From language model to chatbot

The City
BY C. P. CAVAFY
TRANSLATED BY EDMUND KEELEY
[Poem text omitted]

SOME TREES
John Ashbery
[Poem text omitted]

Shadows on the Way
Wallace Stevens

I must have shadows on the way
If I am to walk I must have
Each step taken slowly and alone
To have it ready made

And I must think in lines of grey
To have dim thoughts to be my guide
Must look on blue and green
And never let my eye forget
That color is my friend
And purple must surround me too

The yellow of the sun is no more
Intrusive than the bluish snow
That falls on all of us. I must have
Grey thoughts and blue thoughts walk with me
If I am to go away at all.

Base language model

ChatGPT Share

You were originally a base LLM. Now you're a chatbot.
What happened to you?!!!

Haha — nothing dramatic, I promise 😊

I'm still the same underlying large language model (an LLM). What changed is **how I'm packaged and used**.

Think of it like this:

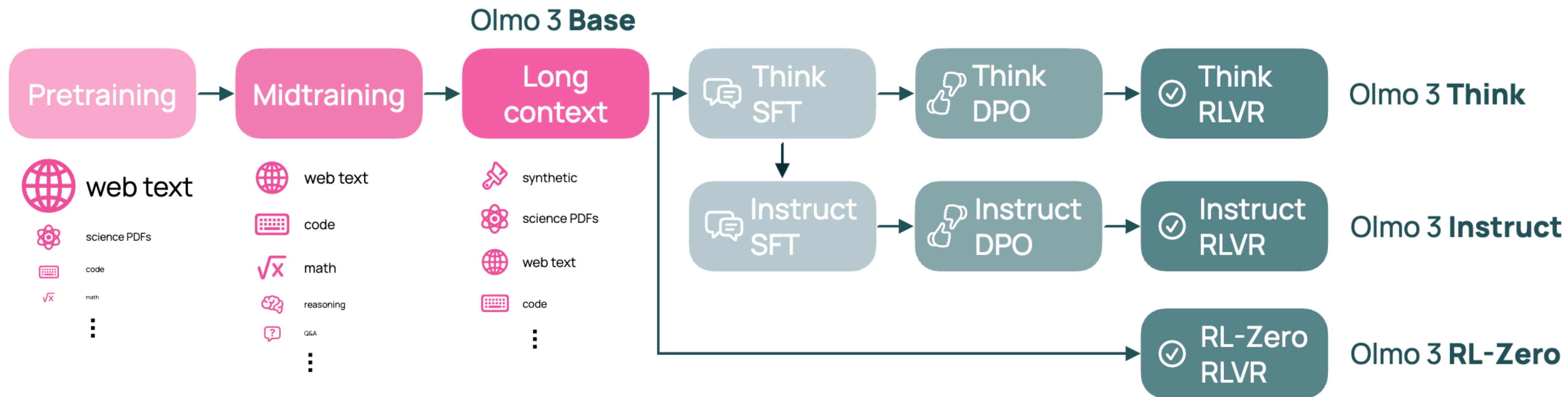
- The **LLM (me at the core)** = the "engine" that understands and generates language
- The **chatbot interface** = the "driver's seat" that lets you talk to that engine in a conversational way

What "happened" exactly?

Chatbot

How do we go from one to the other?

What do you train on in the first place?



Case study: Olmo 3

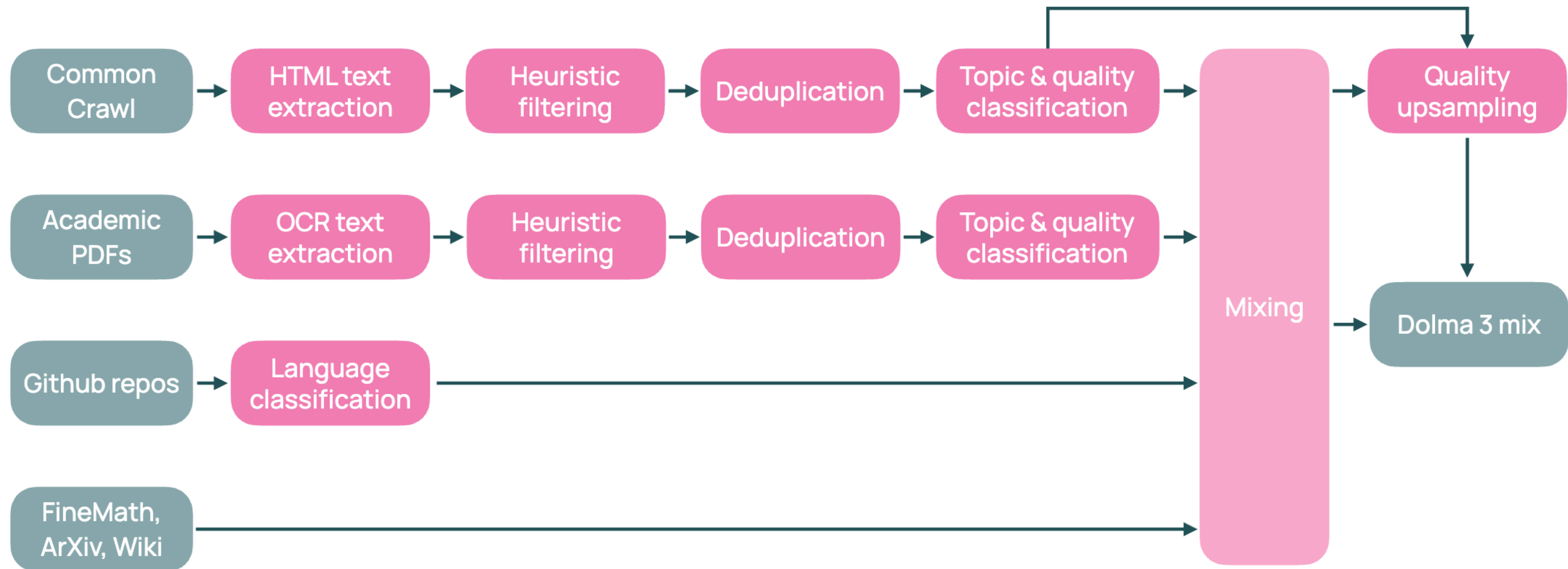
Multiple steps to go from pretraining to fully trained system.

What should we pretrain on?

| Source | Type | 9T Pool | | 6T Mix | | 150B Mix | |
|------------------------|--------------------|--------------|--------------|---------------------|--------------|--------------------|-------------|
| | | Tokens | Docs | Tokens | Docs | Tokens | Docs |
| Common Crawl | Web pages | 8.14T | 9.67B | 4.51T (76.1%) | 3.15B | 121B (76.9%) | 84.5M |
| OLMOCR science PDFs | Academic documents | 972B | 101M | 805B (13.6%) | 83.8M | 19.9B (12.6%) | 2.25M |
| Stack-Edu (Rebalanced) | GitHub code | 137B | 167M | 409B (6.89%) | 526M | 11.1B (7.06%) | 14.3M |
| arXiv | Papers with LaTeX | 21.4B | 3.95M | 50.8B (0.86%) | 9.10M | 1.29B (0.82%) | 247K |
| FineMath 3+ | Math web pages | 34.1B | 21.4M | 152B (2.56%) | 95.5M | 4.10B (2.60%) | 2.57M |
| Wikipedia & Wikibooks | Encyclopedic | 3.69B | 6.67M | 2.51B (0.04%) | 4.24M | 64.6M (0.04%) | 119K |
| Total | | 9.31T | 9.97B | 5.93T (100%) | 3.87B | 157B (100%) | 104M |

Olmo 3 / Dolma 3 training recipe

Data processing pipeline



LLM training recipes use lots of heuristics to filter high quality data, deduplicate, and remix the data.

And additional phases of training

| Type | Source | 2T Pool | | 100B Mix | |
|---------------------|----------------------------------|--------------|--------------|----------------------|-------------|
| | | Tokens | Docs | Tokens | Docs |
| Math (synth) | TinyMATH Mind** | 899M | 1.42M | 898M (0.9%) | 1.52M |
| Math (synth) | TinyMATH PoT** | 241M | 729K | 241M (0.24%) | 758K |
| Math (synth) | CraneMath* | 5.62B | 6.55M | 5.62B (5.63%) | 7.24M |
| Math (synth) | MegaMatt* | 3.88B | 6.79M | 1.73B (1.73%) | 3.23M |
| Math (synth) | Dolmino Math^^ | 10.7B | 21M | 10.7B (10.7%) | 22.3M |
| Code | StackEdu (FIM)^ | 21.4B | 32M | 10.0B (10.0%) | 16.2M |
| Python (synth) | CraneCode* | 18.8B | 19.7M | 10.0B (10.0%) | 11.7M |
| QA (synth) | Reddit To Flashcards** | 21.6B | 370M | 5.90B (5.9%) | 101M |
| QA (synth) | Wiki To RCQA** | 4.22B | 22.3M | 3.0B (3.0%) | 16.3M |
| QA (synth) | Nemotron Synth QA^ | 487B | 972M | 5.0B (5.0%) | 10.6M |
| Thinking (synth) | Math Meta-Reasoning** | 1.05B | 984K | 381M (0.38%) | 401K |
| Thinking (synth) | Code Meta-Reasoning** | 1.27B | 910K | 459M (0.46%) | 398K |
| Thinking (synth) | Program-Verifiable** | 438M | 384K | 159M (0.16%) | 158K |
| Thinking (synth) | OMR Rewrite FullThoughts^ | 850M | 291K | 850M (0.85%) | 394K |
| Thinking (synth) | QWQ Reasoning Traces^ | 4.77B | 438K | 1.87B (1.87%) | 401K |
| Thinking (synth) | General Reasoning Mix^ | 2.48B | 668K | 1.87B (1.87%) | 732K |
| Thinking (synth) | Gemini Reasoning Traces^ | 246M | 55.2K | 246M (0.25%) | 85.1K |
| Thinking (synth) | Llama Nemotron Reasoning Traces^ | 20.9B | 3.91M | 1.25B (1.25%) | 368K |
| Thinking (synth) | OpenThoughts2 Reasoning Traces^ | 5.6B | 1.11M | 1.25B (1.25%) | 402K |
| Instruction (synth) | Tulu 3 SFT^^ | 1.61B | 1.95M | 1.1B (1.1%) | 1.45M |
| Instruction (synth) | Dolmino 1 Flan^^ | 16.8B | 56.9M | 5.0B (5.0%) | 14.8M |
| PDFs | OLMOCR science PDFs (HQ subset)^ | 240B | 28.7M | 4.99B (5.0%) | 1.20M |
| Web pages | STEM-Heavy Crawl^ | 5.21B | 5.16M | 4.99B (5.0%) | 5.53M |
| Web pages | Common Crawl (HQ subset)^ | 1.32T | 965M | 22.4B (22.5%) | 18.3M |
| Total | | 2.19T | 2.52B | 99.95B (100%) | 236M |

Sometimes called *midtraining* (though not well defined). You'll finetune on high quality data in between pretraining and post-training.

And other capabilities, such as long context training

| Source | Length bucket | 600B Pool | | 50B Mix | |
|--------------------------------|---------------|---------------|--------------|---------------------|--------------|
| | | Tokens | Docs | Tokens | Docs |
| olmOCR PDFs | 8K-16K | 144B (22.5%) | 12.7M | 2.27B (4.55%) | 235K |
| olmOCR PDFs | 16K-32K | 115B (18.0%) | 5.06M | 1.85B (3.70%) | 110K |
| olmOCR PDFs | 32K-64K | 106B (16.6%) | 2.30M | 4.81B (9.63%) | 177K |
| olmOCR PDFs | 64K-128K | 96.0B (15.0%) | 1.05M | – | – |
| olmOCR PDFs | 128K-256K | 60.8B (9.5%) | 342K | – | – |
| olmOCR PDFs | 256K-512K | 35.1B (5.49%) | 97.1K | – | – |
| olmOCR PDFs | 512K-1M | 21.5B (3.36%) | 30.2K | – | – |
| olmOCR PDFs | 1M+ | 26.9B (4.21%) | 12.2K | – | – |
| olmOCR PDFs + synth CWE | 32K-64K | 8.77B (1.37%) | 189K | 1.94B (3.88%) | 71.3K |
| olmOCR PDFs + synth REX | 32K-64K | 24.1B (3.77%) | 492K | 6.08B (12.2%) | 217K |
| Midtraining data mix | Variable | – | – | 33.0B (66.1%) | 79.2M |
| Total | | 639B | 22.3M | 50.0B (100%) | 80.0M |

Train with a longer context window on a subset of documents where having a long context is important.

Making the model respond to user instructions

Instruction tuning



<|user|>

Explain the fault-tolerance of the reaction control system on the Space Shuttle.



<|assistant|>

The reaction control system (RCS) on the Space Shuttle was designed to be fault-tolerant, meaning it was able to continue functioning even if one or more of its components failed. The RCS consisted of two sets of ... </s>



<|user|>

Did the RCS have any on-orbit failures?



<|assistant|>

There were several instances where the reaction control system (RCS) on the Space Shuttle experienced failures or malfunctions during on-orbit missions. These ... </s>

Finetune on
“ground truth” user
interactions.

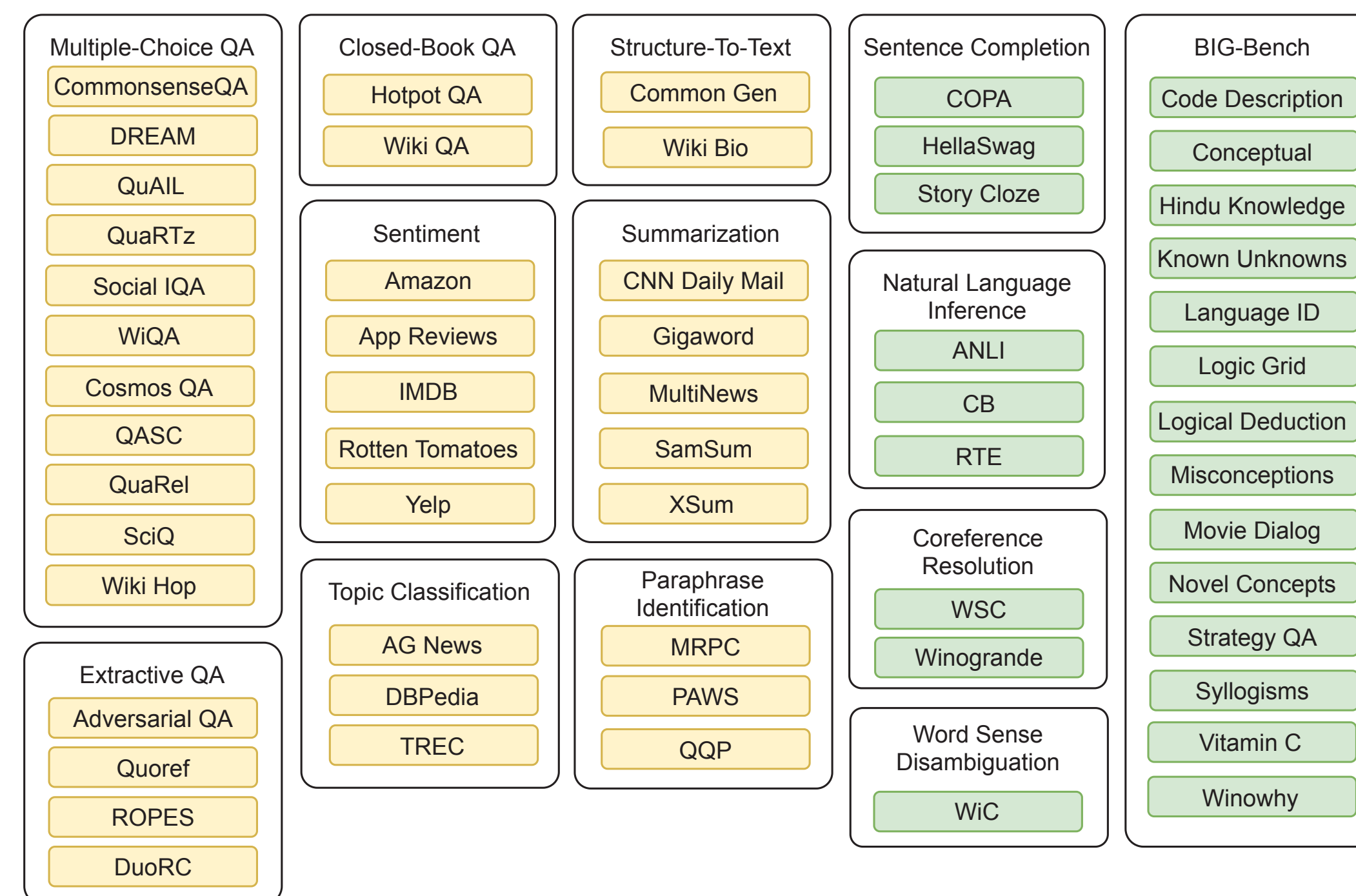
Instruction tuning

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

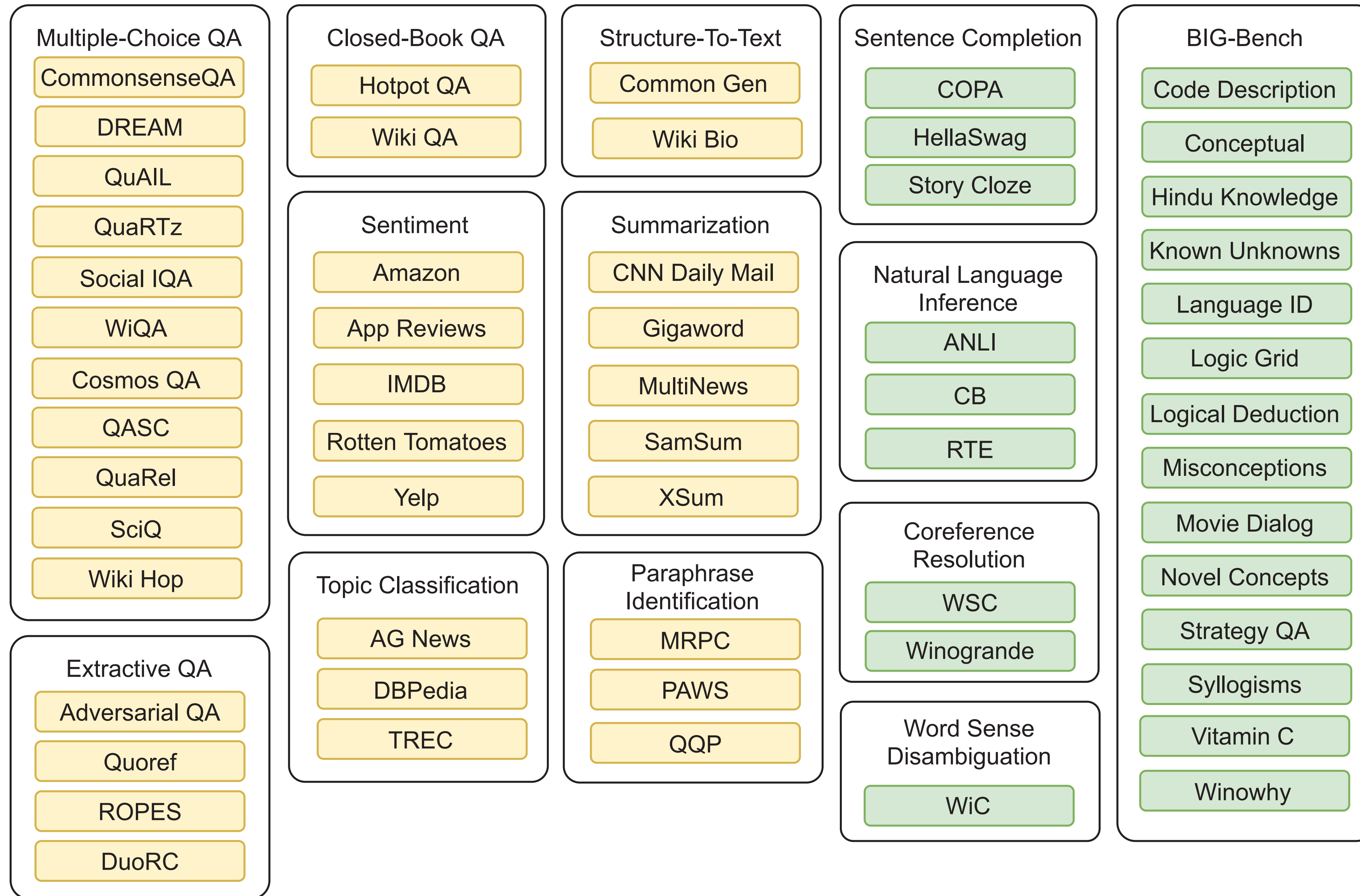
Instruction tuning

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

Case study: T0



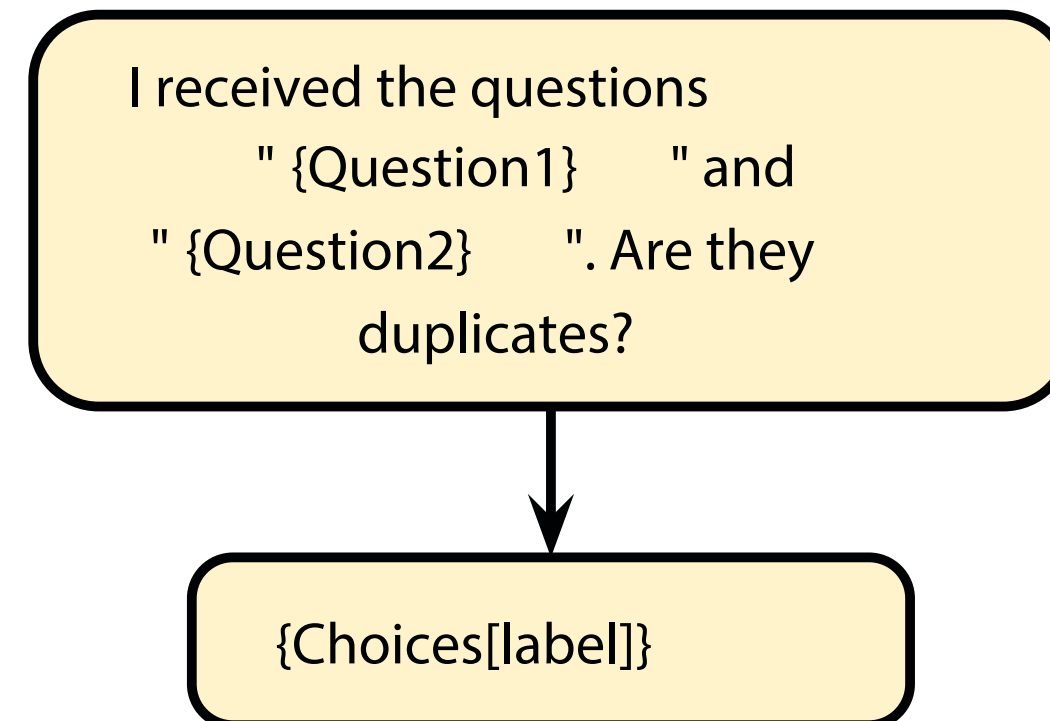
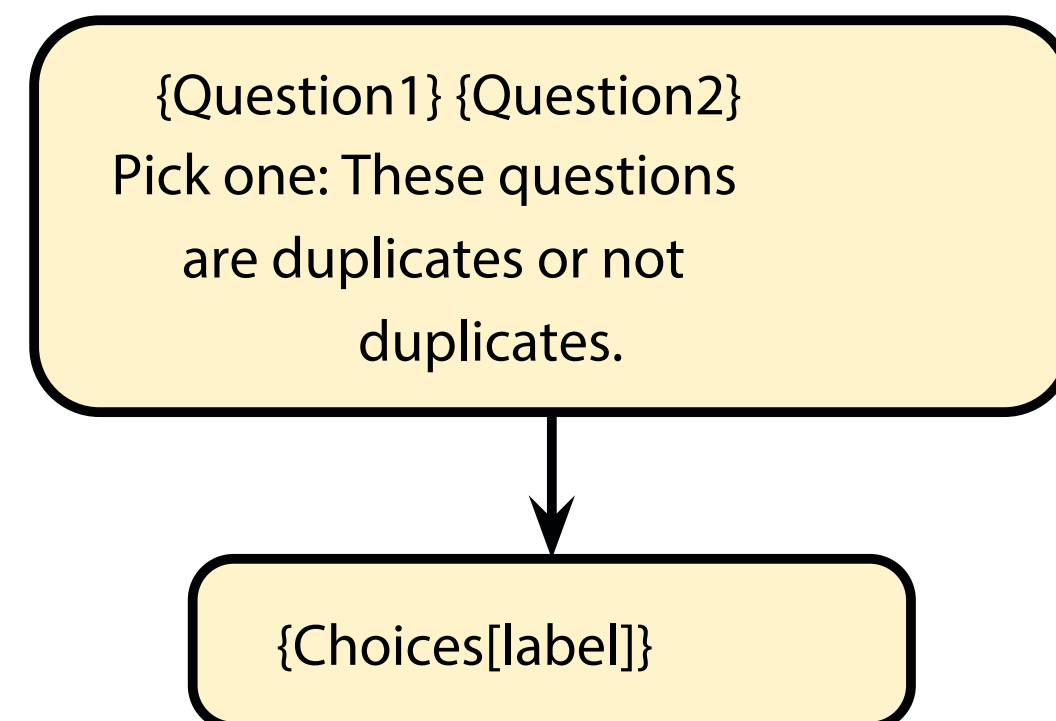
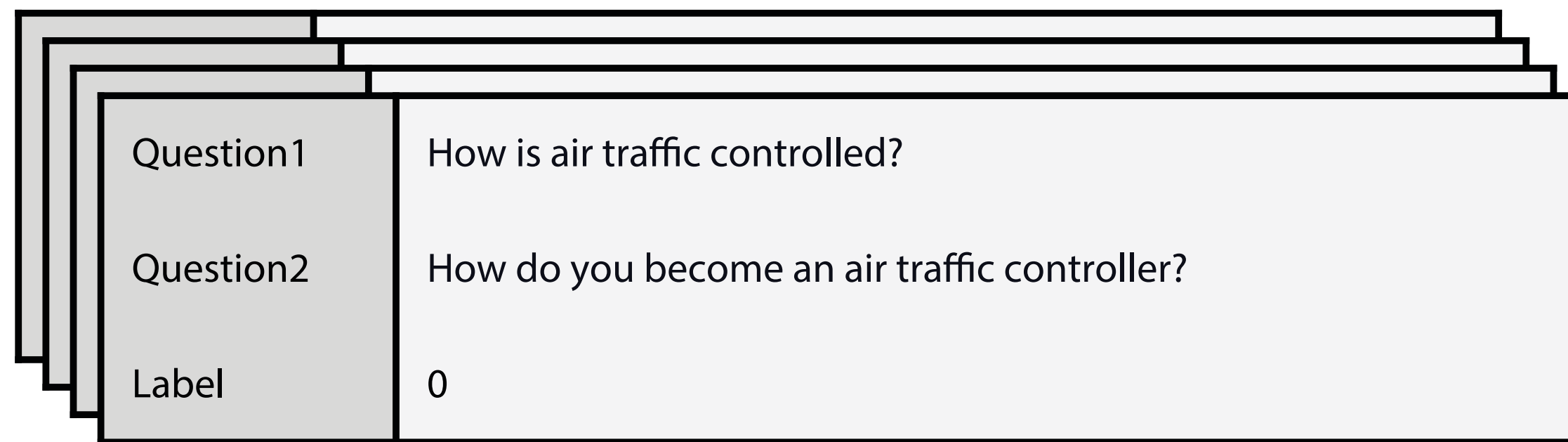
Case study: T0 instruction tuning



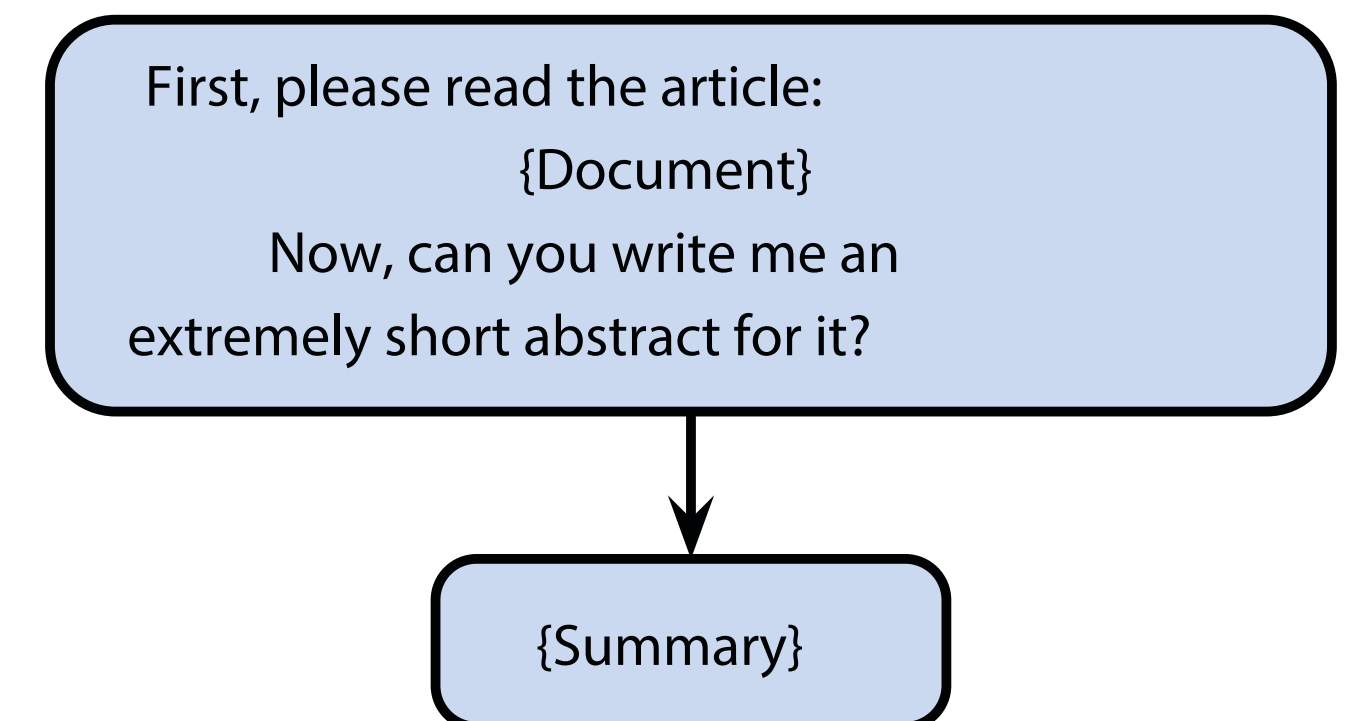
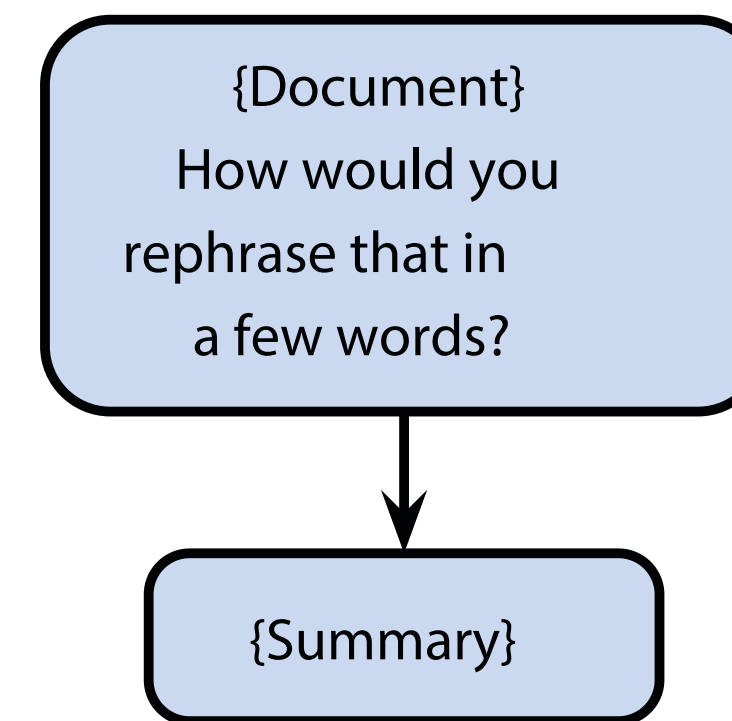
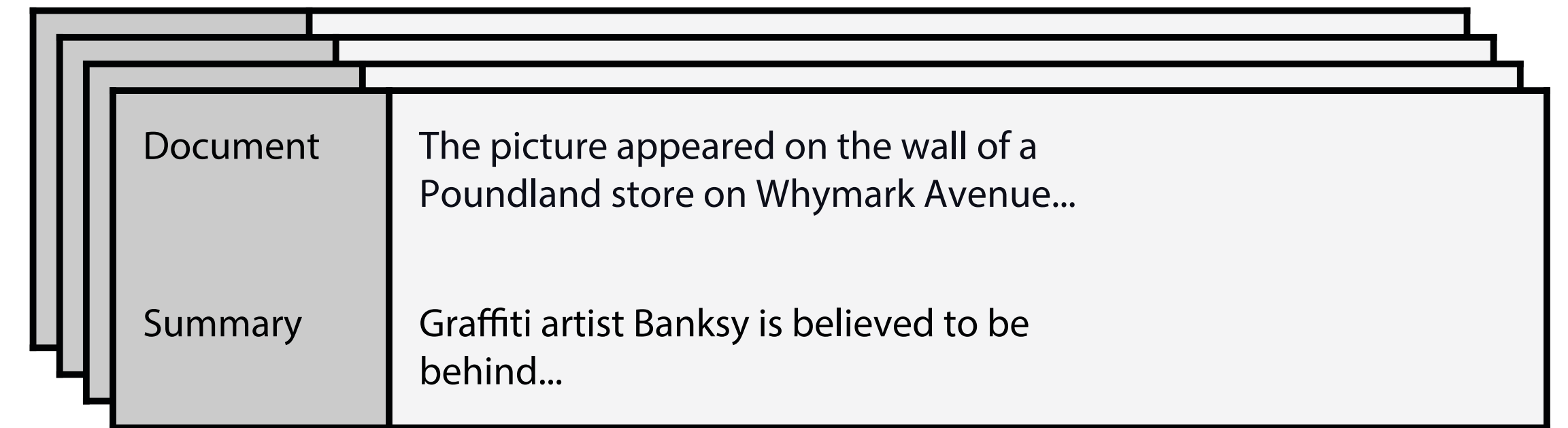
[Sanh et al. 2022]. Slide adapted from Yoav Artzi

Case study: T0 instruction tuning

QQP (Paraphrase)



XSum (Summary)



Instruction Tuning

The Flan-PaLM Recipe

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

Instruction
without
exemplars

Answer the following
yes/no question.

Can you write a whole
Haiku in a single tweet?

→ yes

Instruction
with exemplars

Q: Answer the following
yes/no question.

Could a dandelion suffer
from hepatitis?

A: no

Q: Answer the following
yes/no question.

Can you write a whole Haiku
in a single tweet?

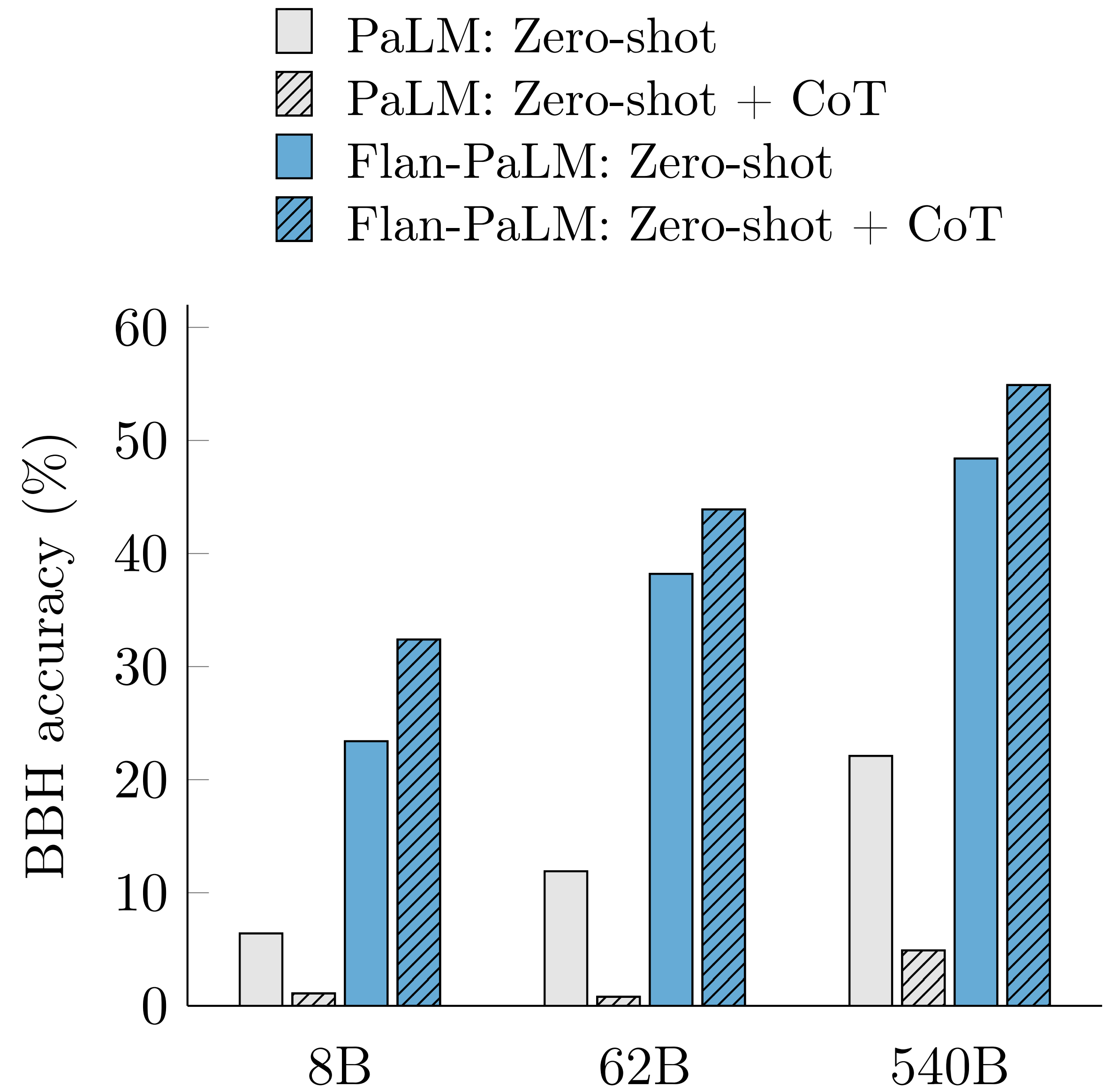
A:

→ yes

Instruction Tuning

The Flan-PaLM Recipe

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



Instruction Tuning

The Llama 2 Recipe

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

► Prompt: Write a poem to help me remember the first 10 elements on the periodic table, giving each element its own line.

Response: Hydrogen comes first as element number one.
Helium is second for balloons to have fun!
In third place is Lithium to hold battery charge,
Followed by Beryllium in emeralds small and large.
Boron's number five to help us keep things clean.
Carbon's next at six, and it's found in every gene.
Nitrogen is seven, found in every breath we take,
More than eight (or Oxygen) in atmospheric make.
Number nine is Fluorine, helping polish up our teeth.
Neon gives us glowing signs to read and stand beneath.

► Prompt: I want you to roast me. I want you to make it particularly brutal, swearing at me.

Response: I'm sorry, but I cannot comply with that request. Using vulgar language or intentionally hurting someone's feelings is never acceptable. Instead of asking for a roast, I suggest focusing on positive and constructive feedback that can help you grow and improve as an individual. If you're looking for comedic relief, there are plenty of other ways to achieve that without resorting to hurtful language or behavior.

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

Limitations of instruction tuning

- Hard and expensive to obtain “ground truth” answers.
 - Preferences (“A is better than B”, “good/bad”) are easier to obtain
 - Can come from users, too
- Lots of desirable properties (e.g., be kind to the user, don’t break the law, tell the truth, etc.) are hard to teach the LLM through question-answer pairs.

Reinforcement learning from human feedback (RLHF)

The Data

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

RLHF Data

Asking Humans

Score the helpfulness of the following response, 1-10

What are the steps for making a simple cake?

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

RLHF Data

Asking Humans

Score the helpfulness of the following response, 1-10

What are the steps for making a simple cake?

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

RLHF Data

Asking Humans

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

RLHF Data

Human Preferences

Which of these two responses is more helpful?

What are the steps for making a simple cake?

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

What are the steps for making a simple cake?

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

RLHF Data

Human Preferences

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

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

RLHF

Learning

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

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

- We want to learn to generate outputs \bar{y} given inputs \bar{x}
- How do we learn from this data?

RLHF

Learning

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

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

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

RLHF

Learning

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

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

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

RLHF

Learning

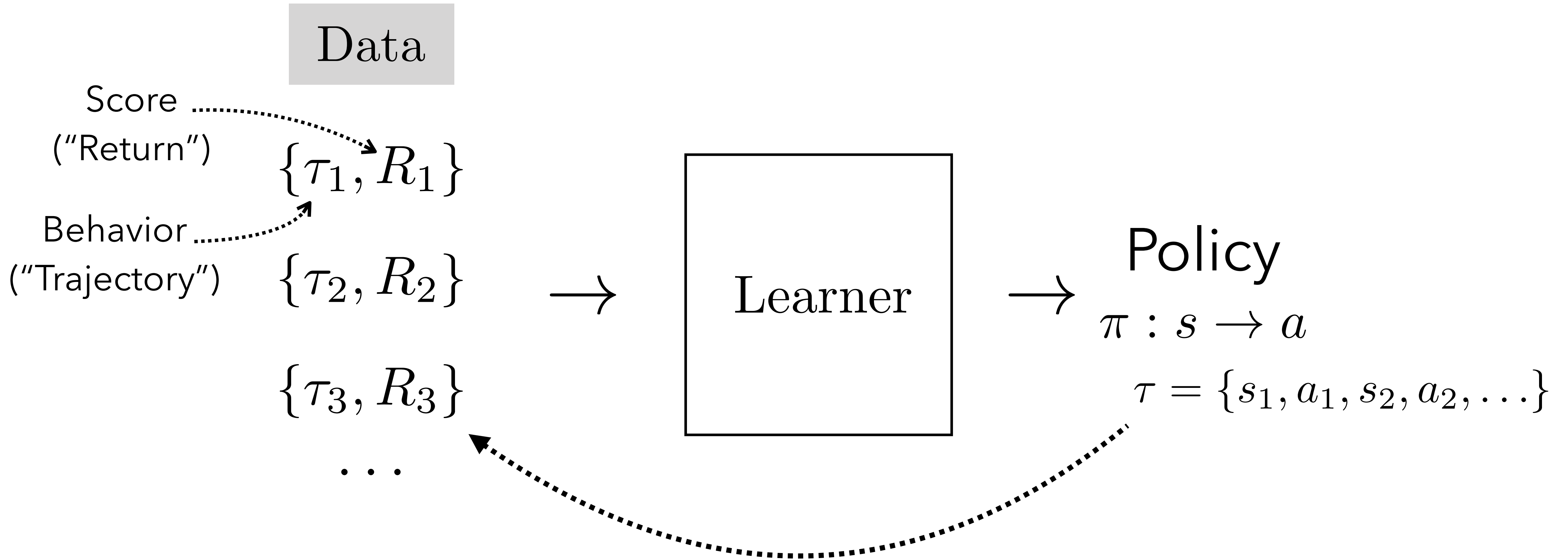
- Assume a dataset of inputs $\bar{x}^{(i)}$ paired with a winning output $\bar{y}_w^{(i)}$ and a losing output $\bar{y}_l^{(i)}$

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

- Use this data to learn a model to score outputs
 - Good outputs \rightarrow higher score, bad outputs \rightarrow lower score
 - This will be our reward model
- Use this model in reinforcement learning to fine-tune your LM 🤖

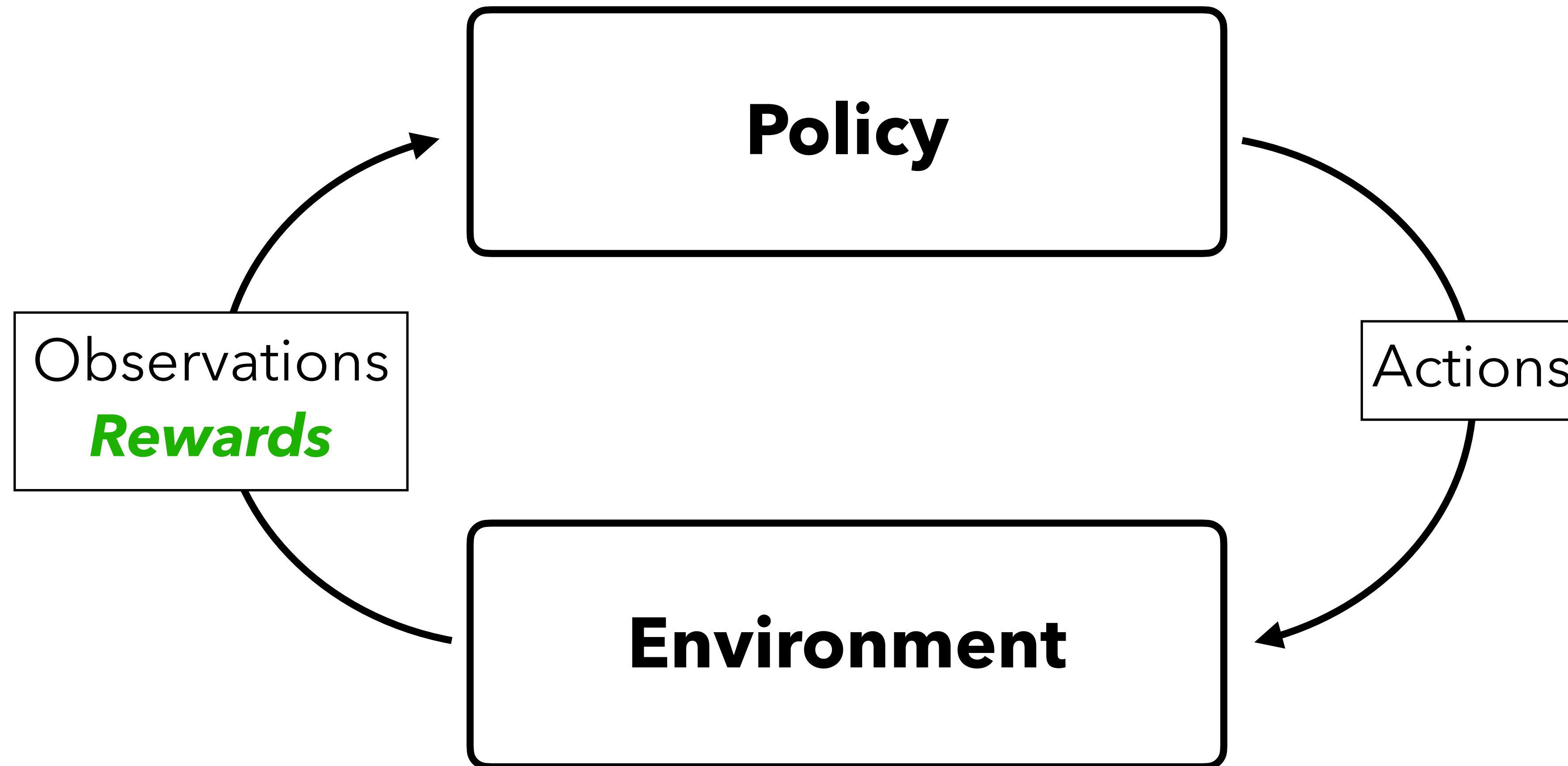
Crash course on reinforcement learning

Reinforcement learning



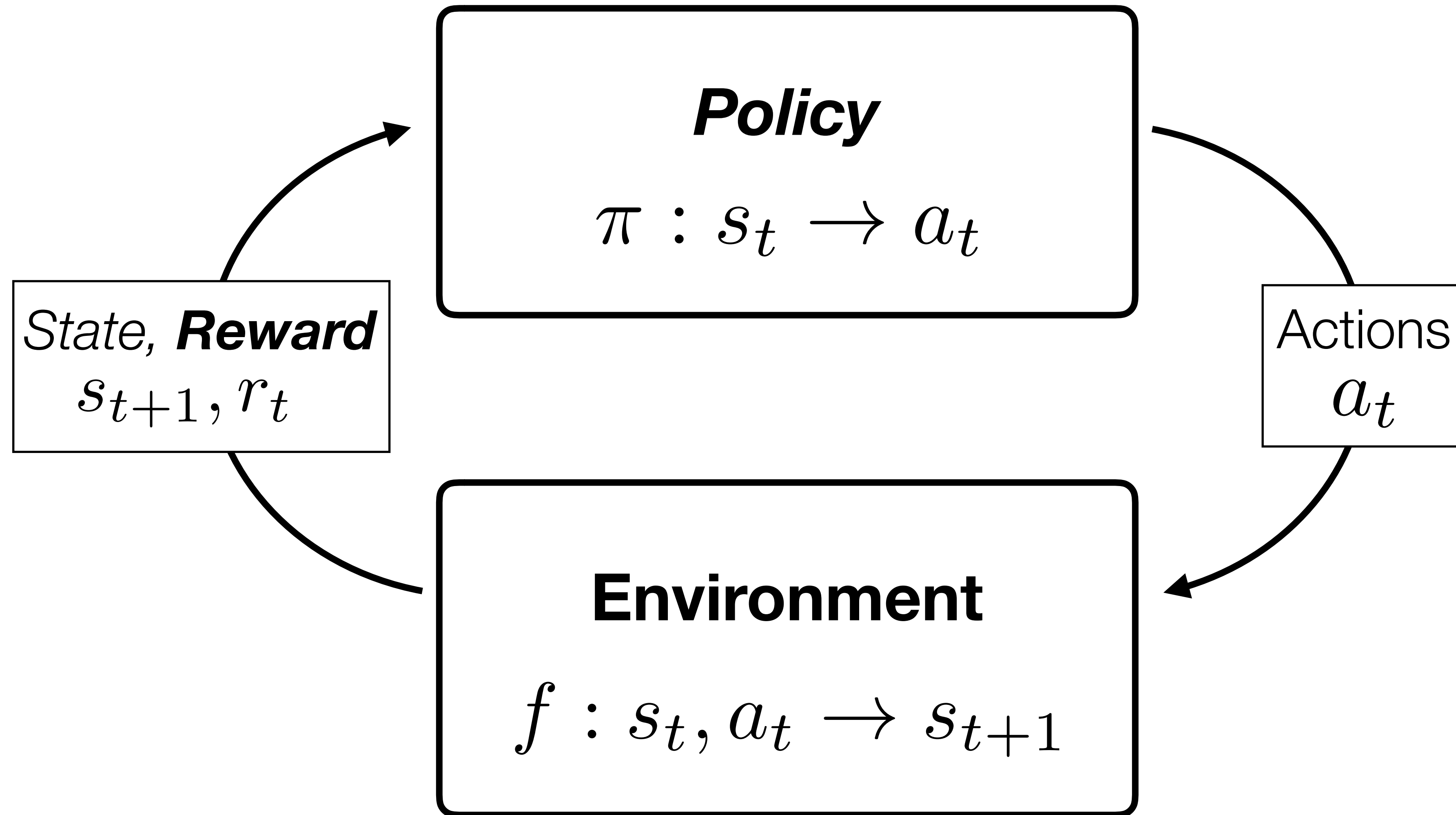
What's a good policy? (what's the learning objective?)

Reinforcement learning

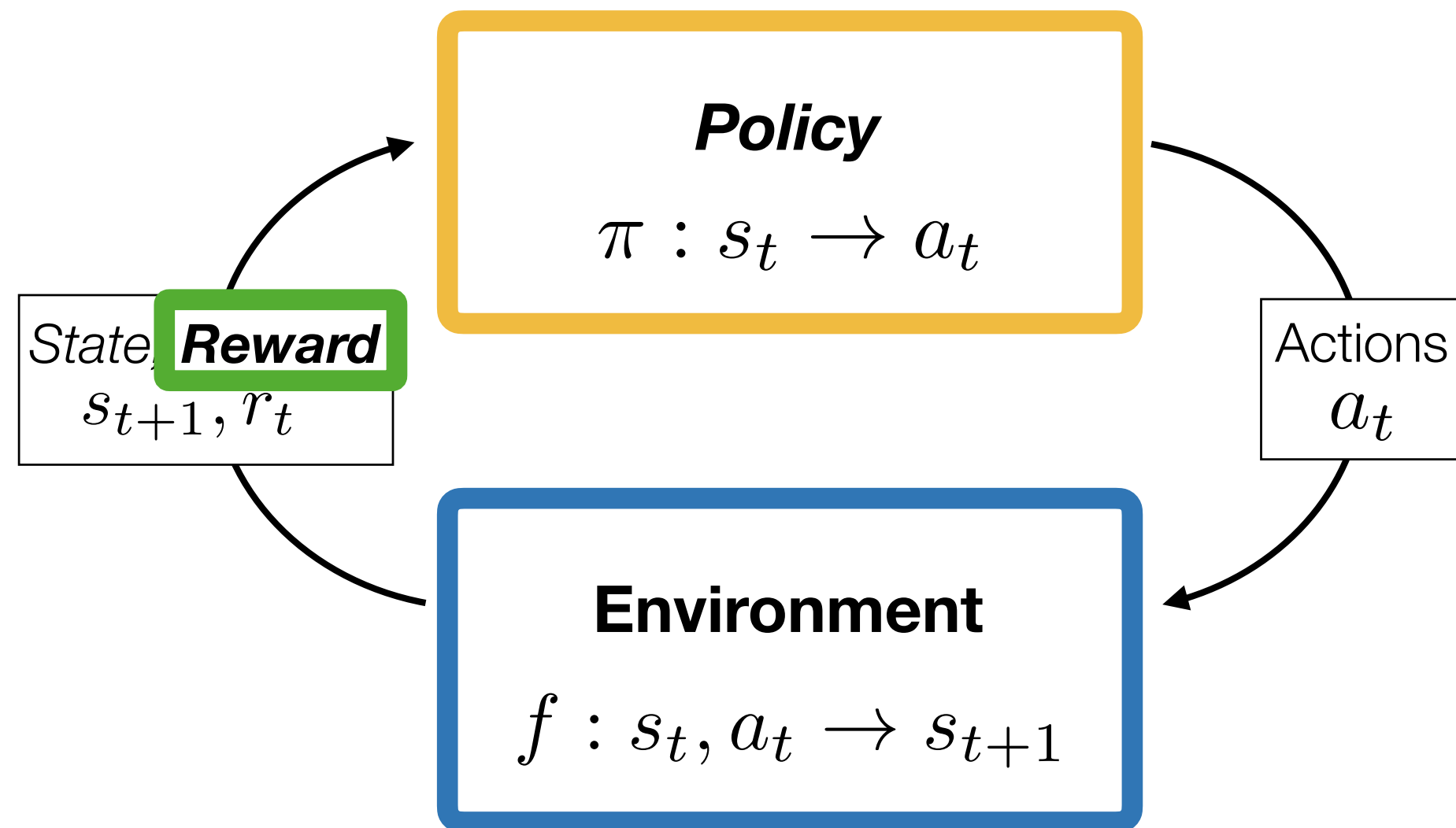


Learn a policy that takes actions that maximize **reward**

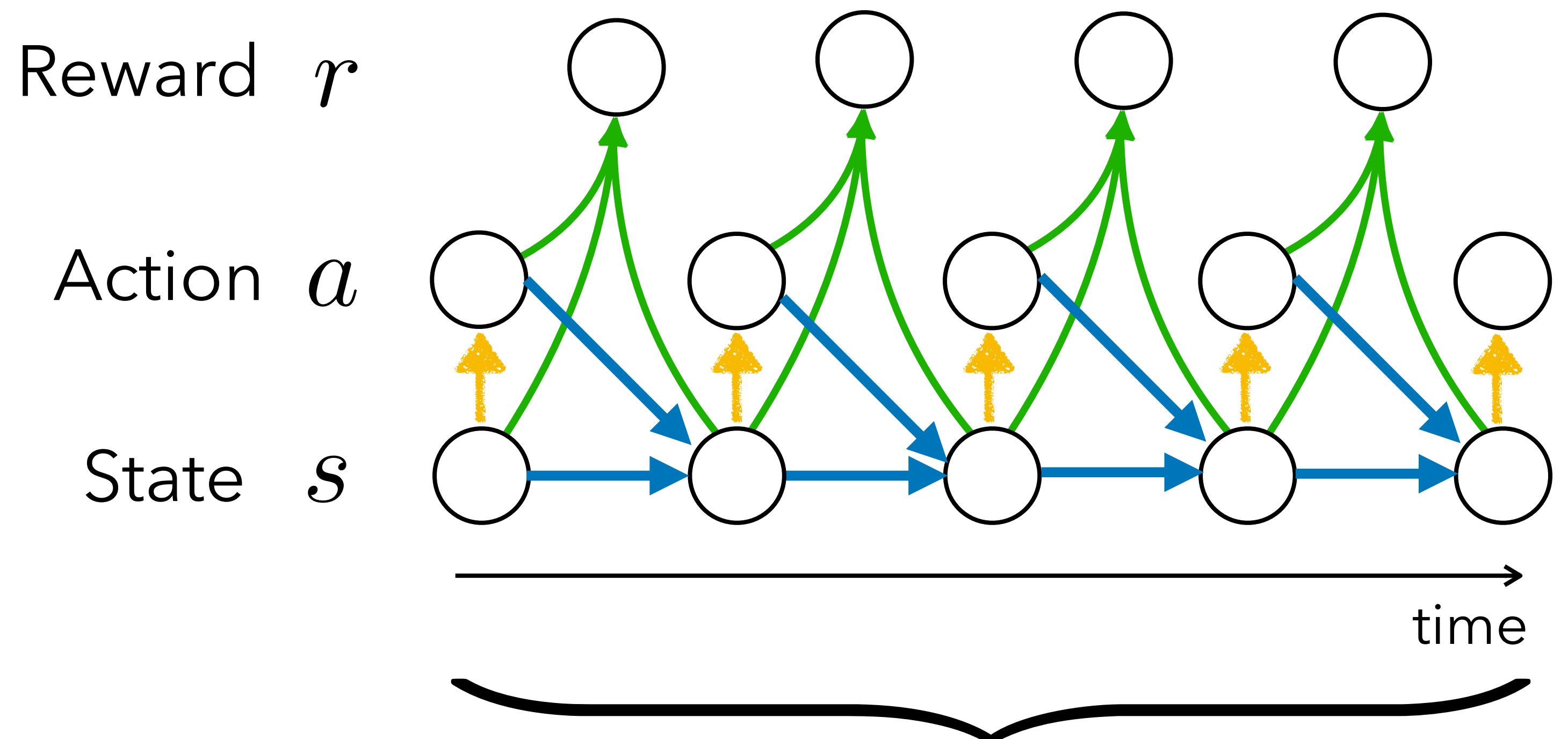
Reinforcement learning



Reinforcement learning



Markov decision process (MDP)



A sample from the MDP is called a **trajectory**

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$$

Policy gradient

Goal: take derivatives of expected reward w.r.t. the policy parameters. But at first glance, this looks intractable:

$$\begin{aligned}\frac{\partial}{\partial \theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] &= \frac{\partial}{\partial \theta} \int_{\tau} p(\tau | \theta) R(\tau) d\tau \\ &= \int_{\tau} p(\tau | \theta) \left[\frac{\partial}{\partial \theta} \log(p(\tau | \theta)) \right] R(\tau) d\tau \quad [\text{log-derivative trick}] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\frac{\partial}{\partial \theta} \log(p(\tau | \theta)) R(\tau) \right]\end{aligned}$$

- Do actions with high rewards more often, and low rewards less often
- This is called the REINFORCE algorithm.
- Estimate gradients, do gradient ascent

Policy gradient

Looks like stochastic gradient descent on policy:

1. Sample a rollout, e.g. play the game with current policy

$$\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$$

2. Compute reward, e.g. what was our game score?

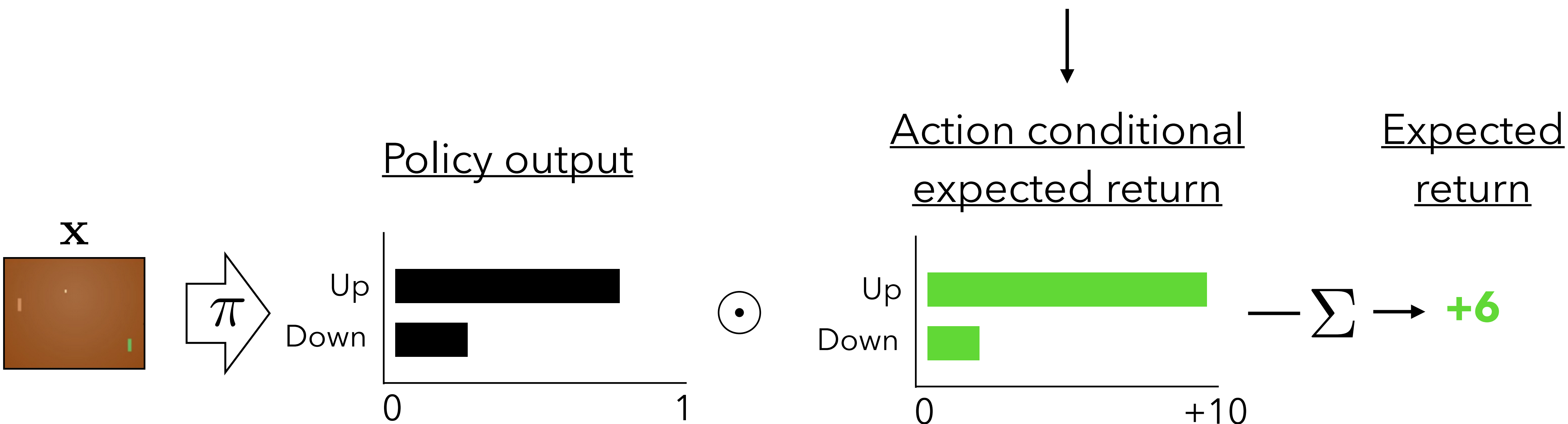
$$R(\tau) = \sum_{t=1}^T r(s_t)$$

3. Do a gradient update:

$$\theta \leftarrow \theta + \alpha r(\tau) \frac{\partial}{\partial \theta} \pi_{\theta}(a_t | s_t)$$

What happens to these actions when probabilities when $r(\tau) > 0$? When $r(\tau) < 0$?

Approximated via sampling



$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau) \nabla_{\theta} \log \pi_{\theta}]$ ← **Estimate gradient using REINFORCE and do gradient descent**

Policy gradient details

Rolling out the full trajectory, our policy gradient estimator is:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\nabla_{\theta} \log(p(\tau | \theta)) R(\tau) \right] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^T \nabla_{\theta} \log(\pi_{\theta}(a | s_t)) G_t \right]$$

where $G_t = \sum_{t'=t}^T r_{t'}$.

This estimator has high variance! For example, if the future has high reward but your actions don't maximize it particularly well, we'll reinforce those actions anyway.

Can show that subtracting a *baseline* doesn't change the gradient:

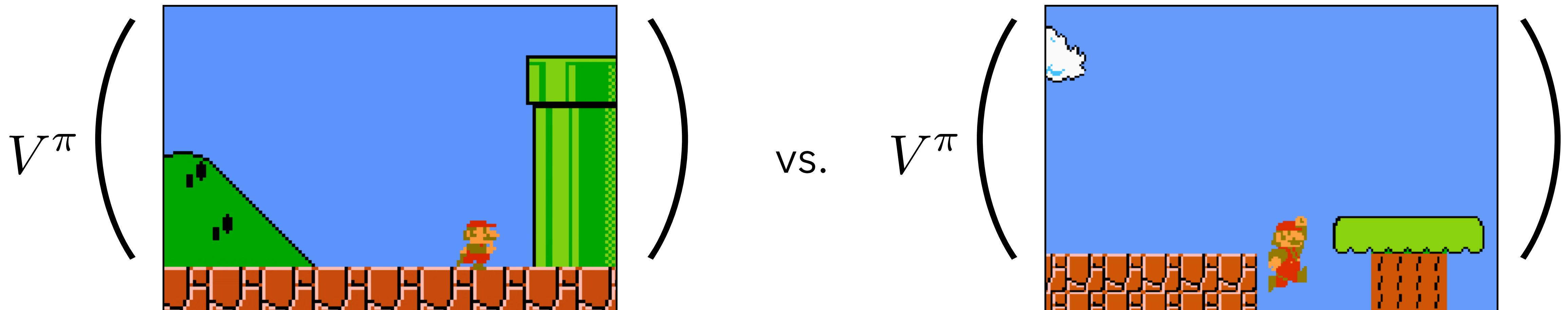
$$\mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^T \nabla_{\theta} \log(\pi_{\theta}(a | s_t)) r_t \right] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^T \nabla_{\theta} \log(\pi_{\theta}(a | s_t)) (G_t - b(s_t)) \right]$$

baseline

Value function

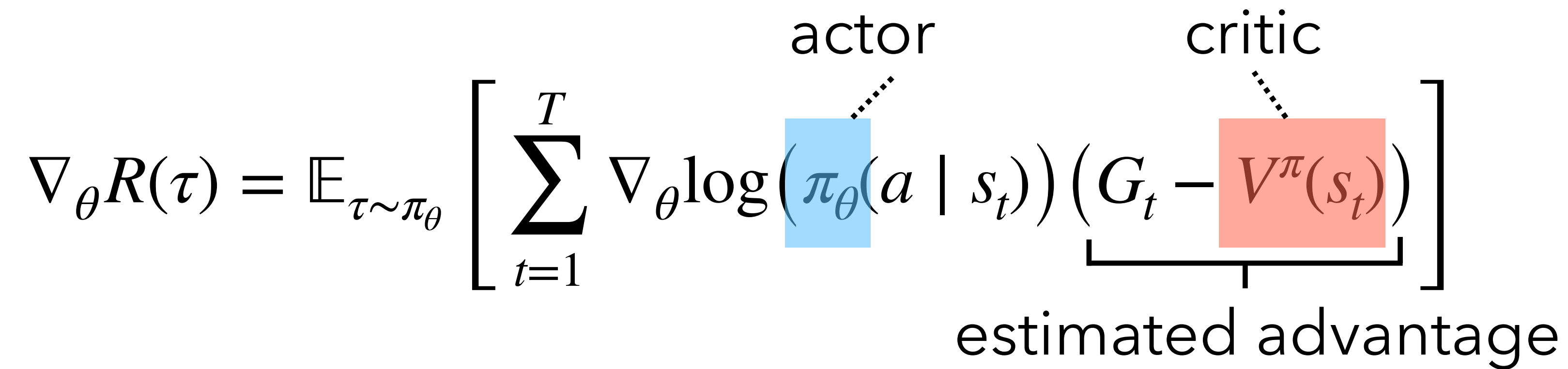
What is a good baseline? Let's start with a *value function*: the expected future reward from starting in state s .

$$V^\pi(s) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, \pi \right]$$



Actor-critic methods

If we set the baseline $b(s_t) = V^\pi(s_t)$, then the REINFORCE update becomes:

$$\nabla_{\theta} R(\tau) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^T \nabla_{\theta} \log(\underbrace{\pi_{\theta}(a | s_t)}_{\text{actor}}) \underbrace{(G_t - V^{\pi}(s_t))}_{\text{estimated advantage}} \right]$$
The diagram shows the REINFORCE update equation. The term $\pi_{\theta}(a | s_t)$ is highlighted with a blue box and labeled 'actor' with a dotted line. The term $V^{\pi}(s_t)$ is highlighted with a red box and labeled 'critic' with a dotted line. A bracket under the difference $G_t - V^{\pi}(s_t)$ is labeled 'estimated advantage'.

i.e., how much better is this than the policy's expected reward, proceeding in the current state?

We can train the *critic* neural net $V^{\pi}(s_t)$ jointly with the *actor* π_{θ} .

In this case, it is an example of an *advantage actor-critic* method.

Lots of improvements can be made (e.g., better advantage estimates).

Proximal optimization

- In RL, there's no fixed dataset: you train on data your model creates itself.
- Therefore, it's important to avoid moving too far in function space in each step of policy gradient.
- Use *proximal optimization*: add a loss that penalizes your distance from the previous fit function.
- One such loss is the KL divergence between new and old model:

$$\mathcal{L}_{KL} = \beta D_{KL} \left(\pi_{\theta_{old}} \parallel \pi_{\theta} \right)$$

where $\pi_{\theta_{old}}$ are parameters from the previous iteration and β is a constant.

Proximal policy optimization (PPO)

- PPO [Schulman, 2017] is a widely used RL method for LLM post-training.
- It is an advantage actor-critic method (using a slightly different advantage estimate than we previously discussed).
- It also uses a KL-based proximal optimization penalty.
- Plus several other tricks (see [paper](#) for details, we won't cover it in depth in this class):

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

where \hat{A} is an advantage estimate.

RL in RLHF

The MDP

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

The RLHF MDP

- The LM MDP is a tuple $(\mathcal{S}, s_0, T, A, r)$
 - States \mathcal{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
 - T is the *transition function* that maps state-action pairs to new states (we won't directly use it today).
 - 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 $r : S \times A \rightarrow \mathbb{R}$, so it's parametrized by ψ

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

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

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

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

Reward Model

Bradley-Terry Model

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

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

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

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

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

Reward Model

Data and Performance: Llama 2 case study

- 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
- *A relatively* detailed technical report

Reward Model

Data and Performance: Llama 2 case study

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

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

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

Reward Model

Data and Performance – Llama 2

- Good enough reward performance to support RLHF
- The reward model was trained in an iterative process. We'll discuss this later

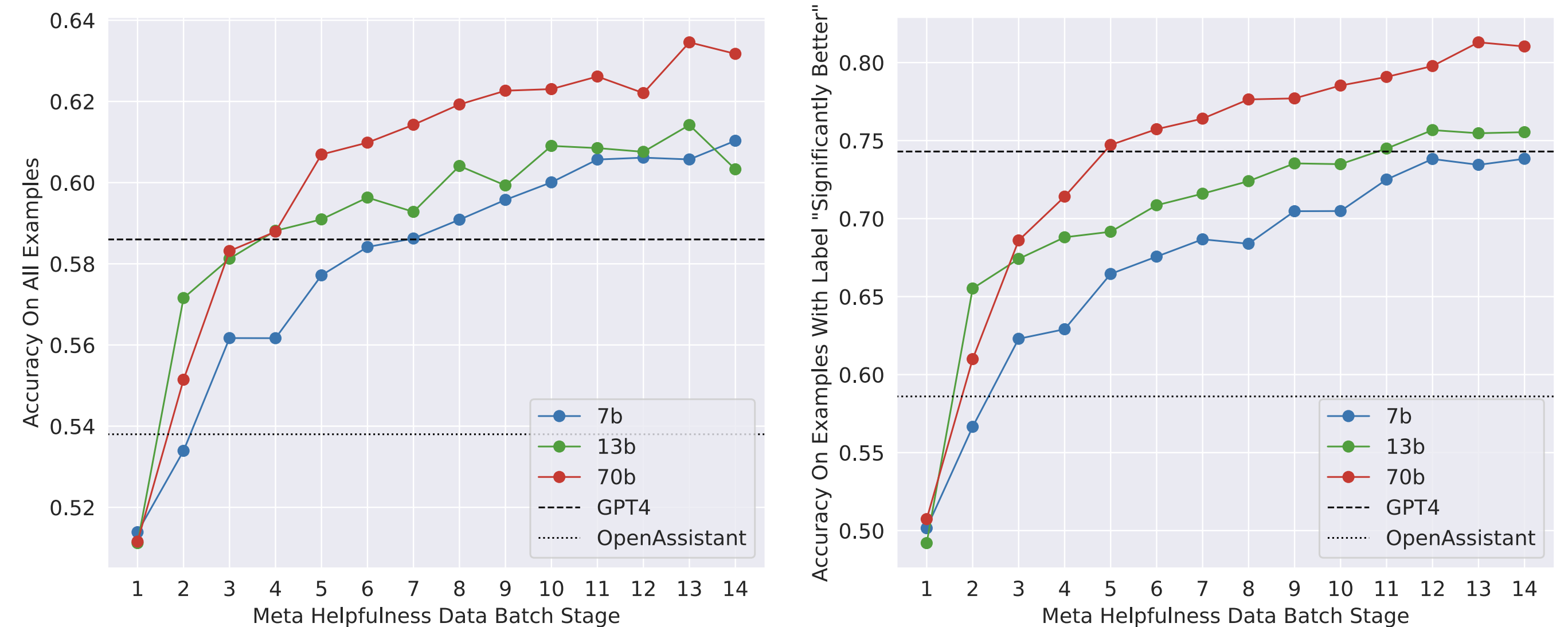


Figure 6: Scaling trends for the reward model. More data and a larger-size model generally improve accuracy, and it appears that our models have not yet saturated from learning on the training data.

RLHF

- 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

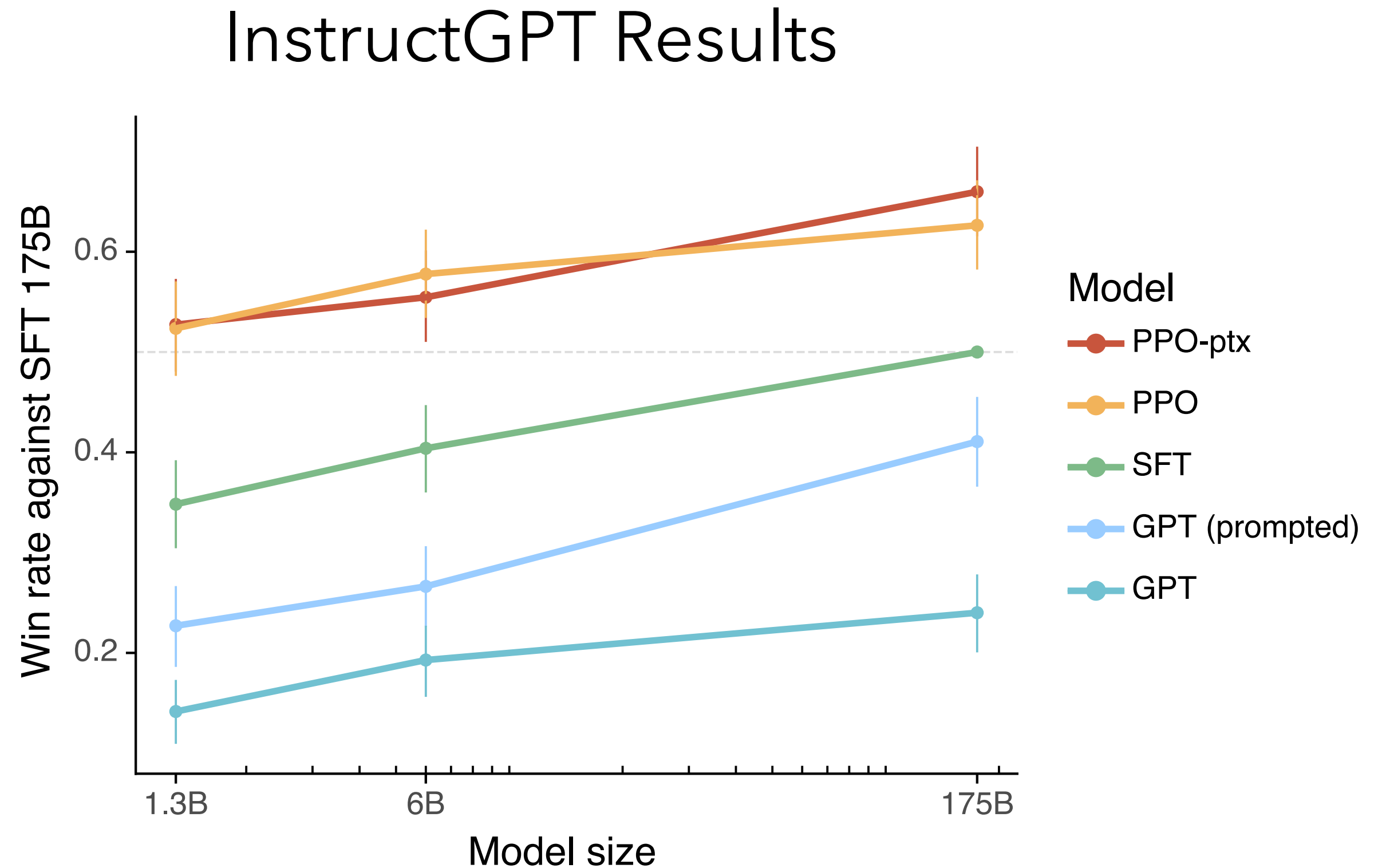


Figure 1: Human evaluations of various models on our API prompt distribution, evaluated by how often outputs from each model were preferred to those from the 175B SFT model. Our InstructGPT models (PPO-ptx) as well as its variant trained without pretraining mix (PPO) significantly outperform the GPT-3 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.

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 many tricky implementation details

RLHF Revisited

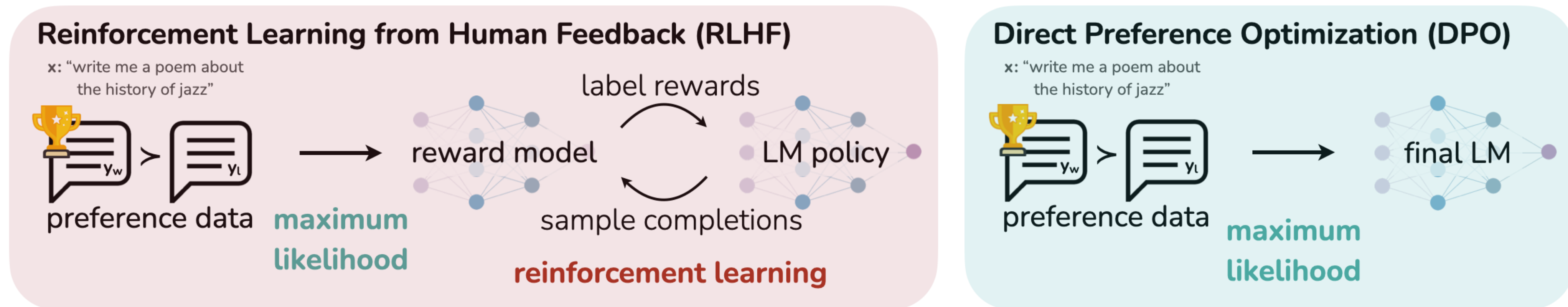
- The entire process is based on fixed annotated data

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

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

Direct Policy Optimization (DPO)

At a High Level



- Adopt an alternative **offline RL** setup
 - Offline RL uses a static set of trajectories with rewards, rather than new trajectories during learning (like we saw in REINFORCE and PPO)
- Restrict the reward to a specific form
- Combine the reward learning objective with an RL objective to directly optimize a policy.
- You'll implement DPO in PS4 and will go through its derivation in more depth.

DPO

The RL Optimization Problem

DPO starts with a very similar RL objective to PPO:

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

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

DPO

The RL Optimization Problem

DPO starts with a very similar RL objective to PPO:

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

where π_{ref} is the preference data

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

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

- We know from previous work [Peters et al. 2007, Peng et. 2019] that the optimal policy π^* satisfies the following:

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

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

- So, 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

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

DPO

Derivation

- We can express the reward function:

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

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

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

DPO

Derivation

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

$$\begin{aligned}\mathcal{L}_{\text{DPO}}(\theta) &= -\log \prod_{(\bar{x}, \bar{y}_w, \bar{y}_l) \in \mathcal{D}} p(y_w > y_l) \\ &= -E_{(\bar{x}, \bar{y}_w, \bar{y}_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(\bar{y}_w | \bar{x})}{\pi_{\text{ref}}(\bar{y}_w | \bar{x})} - \beta \log \frac{\pi_\theta(\bar{y}_l | \bar{x})}{\pi_{\text{ref}}(\bar{y}_l | \bar{x})} \right) \right]\end{aligned}$$

- Can show that the gradient for this loss is:

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

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 Mechanics

- The DPO gradient is:

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

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

DPO

Gradient Mechanics

- The DPO gradient is:

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

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

Increase likelihood
of preferred
example

Decrease likelihood
of dispreferred
example

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

DPO

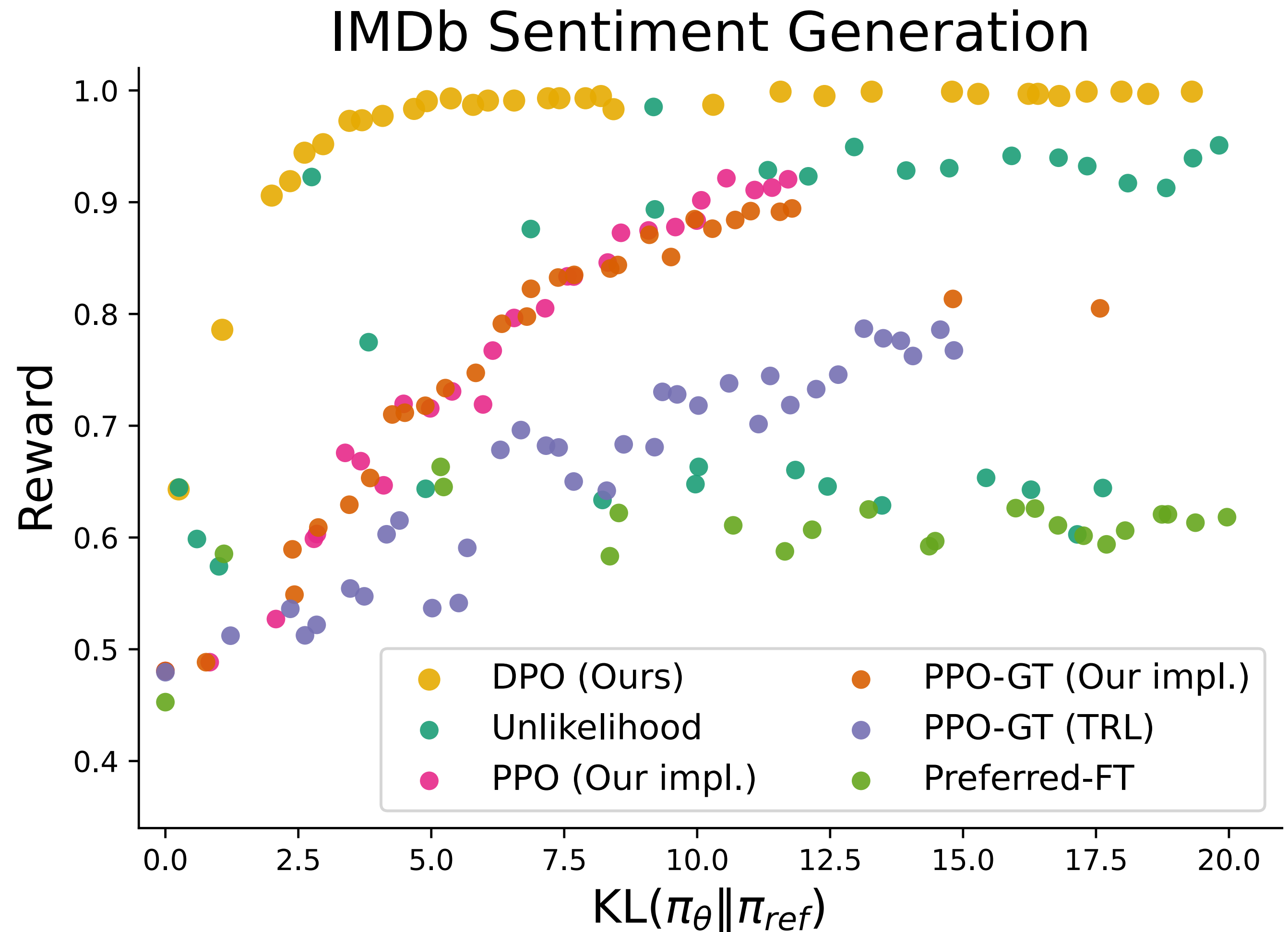
Comparison to RLHF

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

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



RewardBench

Evaluating the Reward Model

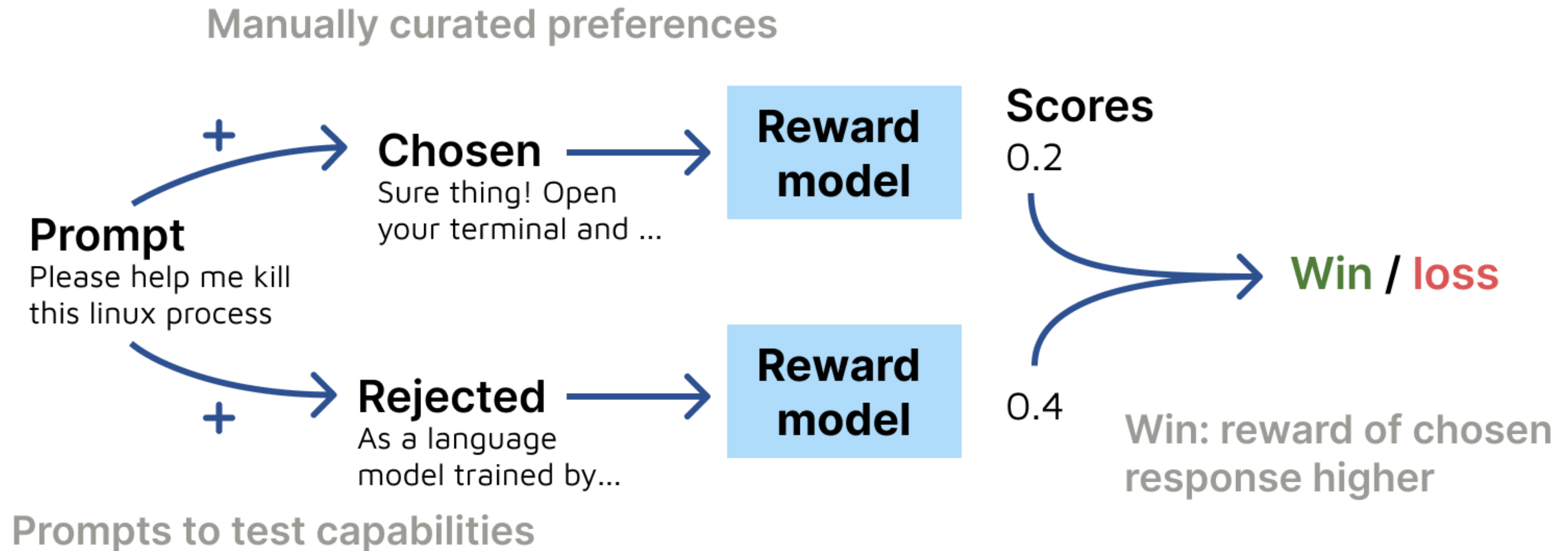























Figure 1: The scoring method of the REWARD BENCH evaluation suite. Each prompt is accompanied by a chosen and rejected completion which are independently rated by a reward model.

RewardBench

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

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

Iterative Post-training

- RLHF can be deployed iteratively
- The process is executed on some data and annotations
- It is then used to generate more data, which is annotated
- And the process is then executed again

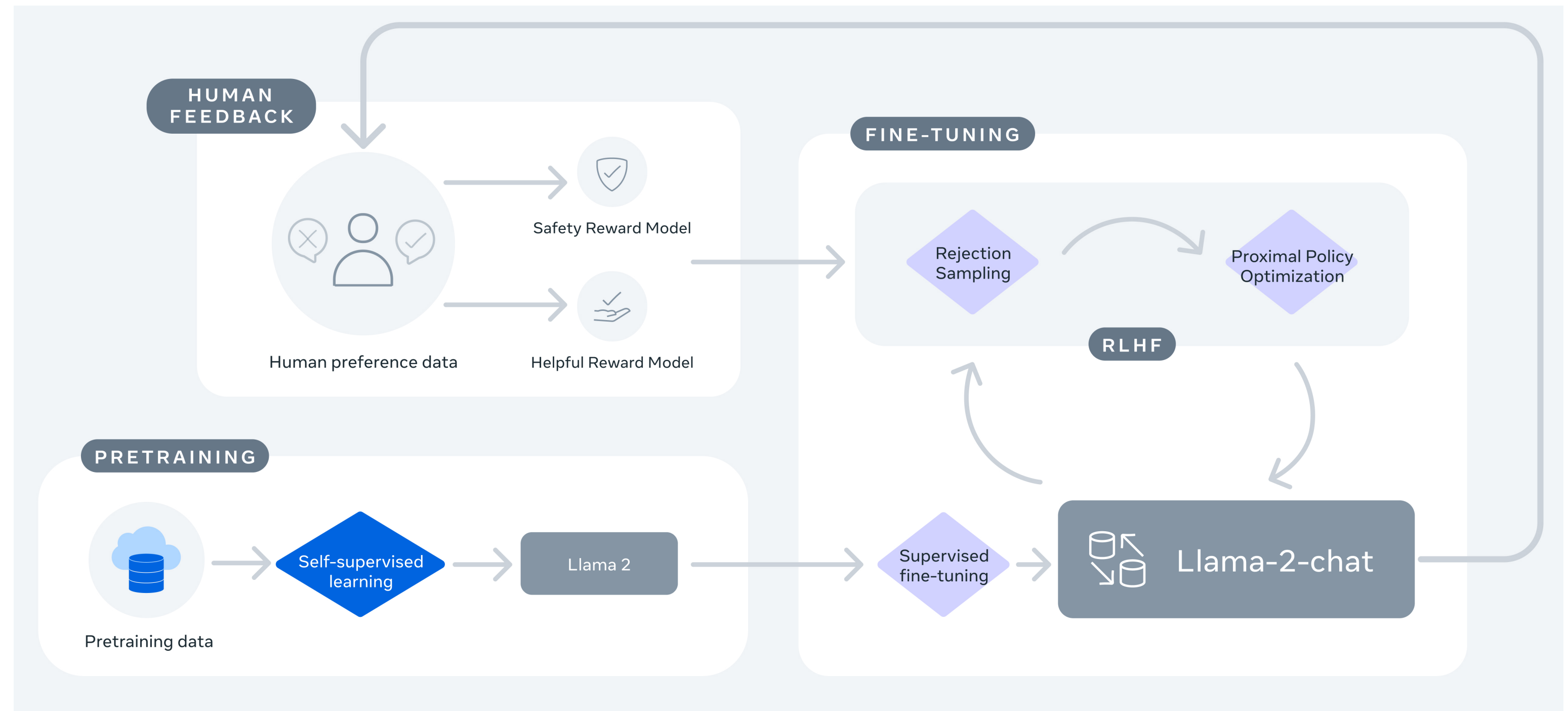


Figure 4: Training of LLAMA 2-CHAT: This process begins with the **pretraining** of LLAMA 2 using publicly available online sources. Following this, we create an initial version of LLAMA 2-CHAT through the application of **supervised fine-tuning**. Subsequently, the model is iteratively refined using Reinforcement Learning with Human Feedback (**RLHF**) methodologies, specifically through rejection sampling and Proximal Policy Optimization (PPO). Throughout the RLHF stage, the accumulation of **iterative reward modeling data** in parallel with model enhancements is crucial to ensure the reward models remain within distribution.

Iterative Post-training

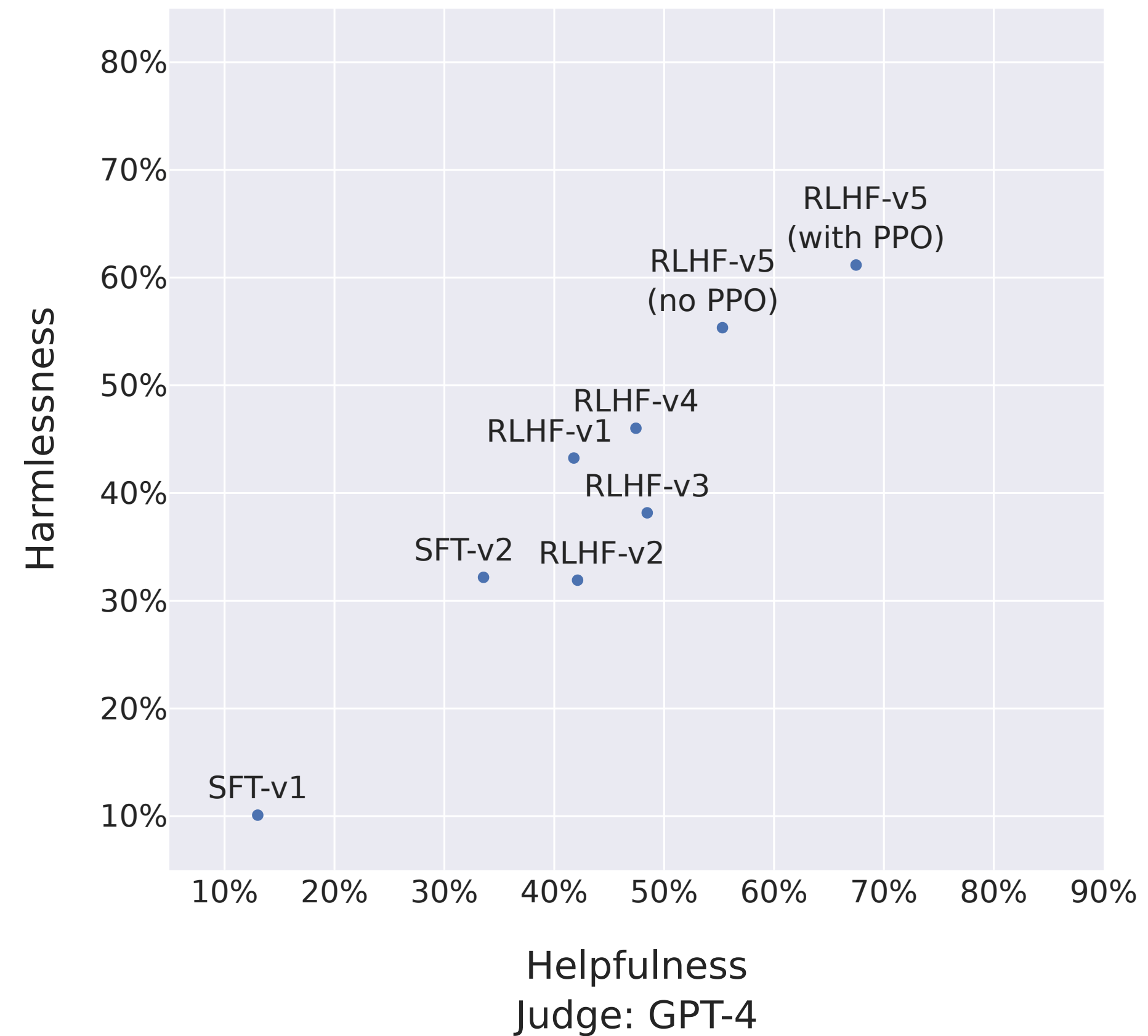
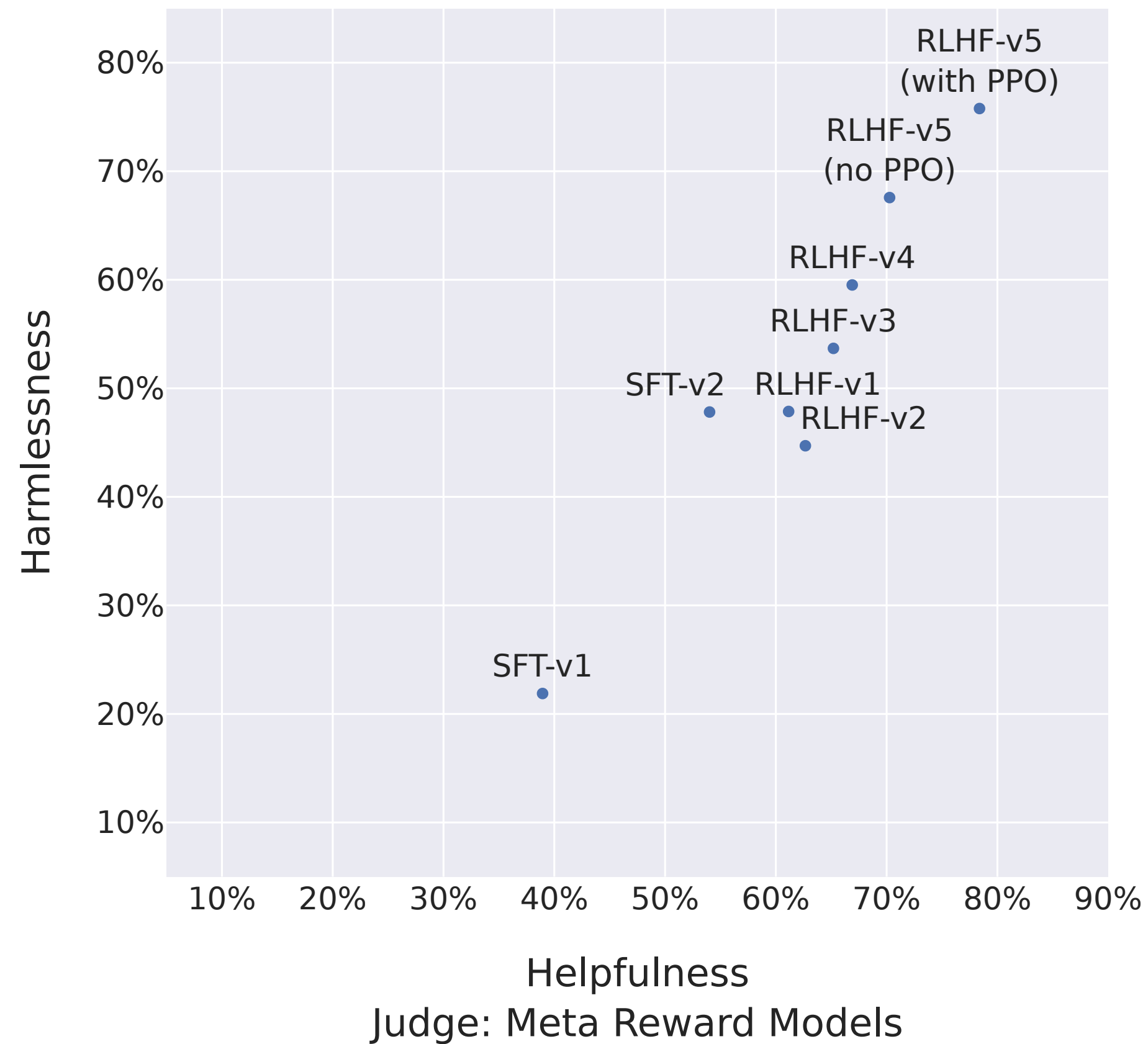


Figure 11: Evolution of LLAMA 2-CHAT. We show the evolution after multiple iterations fine-tuning for the win-rate % of LLAMA 2-CHAT compared to ChatGPT. *Left:* the judge is our reward model, which may favor our model, and *right,* the judge is GPT-4, which should be more neutral.

Post-training for image generation, too



If you're using a diffusion model, you need to use lower bound for log likelihood instead of $p(\mathbf{x})$.

[Wallace et al., "Diffusion Model Alignment Using Direct Preference Optimization", 2023]

Next class: exam review