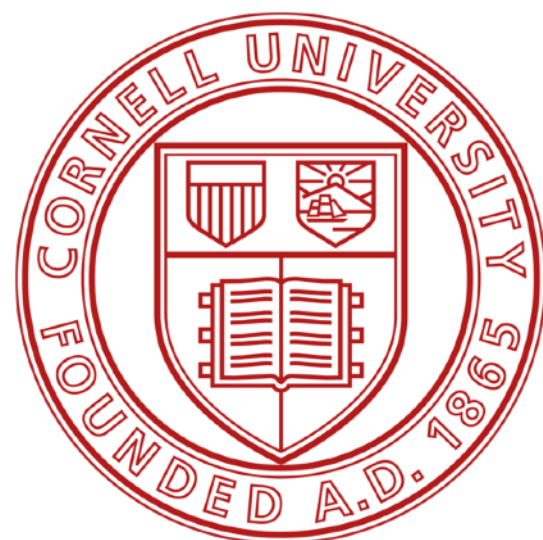


Lecture 21: Post-training (part 2)

CS 5788: Introduction to Generative Models



Lots of slides from Yoav Artzi, LM class

Recall: 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?

Recall: 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 🤖

Recall: 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 $R(\tau) < 0$?

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:

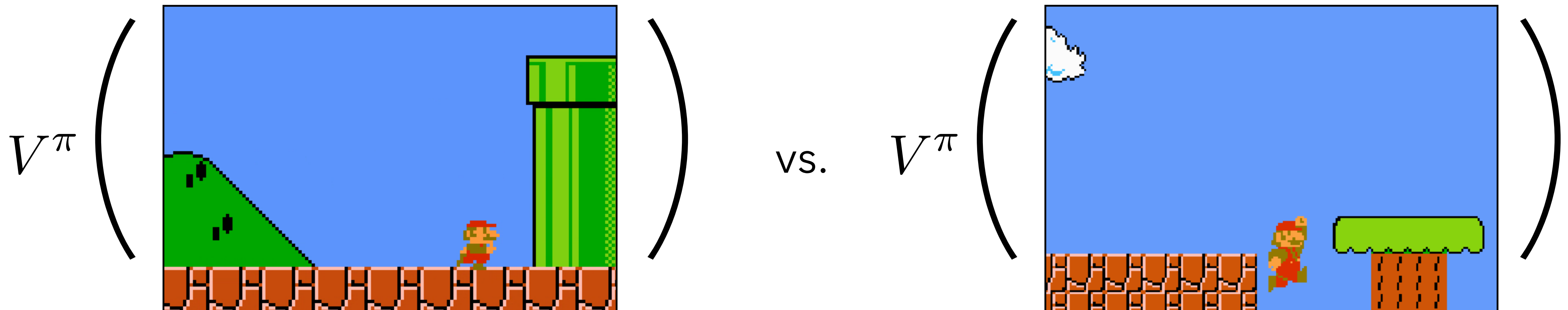
$$\mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^T \nabla_{\theta} \log(\pi_{\theta}(a | s_t)) r_t \right] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^T \nabla_{\theta} \log(\pi_{\theta}(a | s_t)) (G_t - b(s_t)) \right]$$

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:

$$\nabla_{\theta} R(\tau) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=1}^T \nabla_{\theta} \log(\underbrace{\pi_{\theta}(a | s_t)}_{\text{actor}}) \underbrace{(G_t - V^{\pi}(s_t))}_{\text{estimated advantage}} \right]$$

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

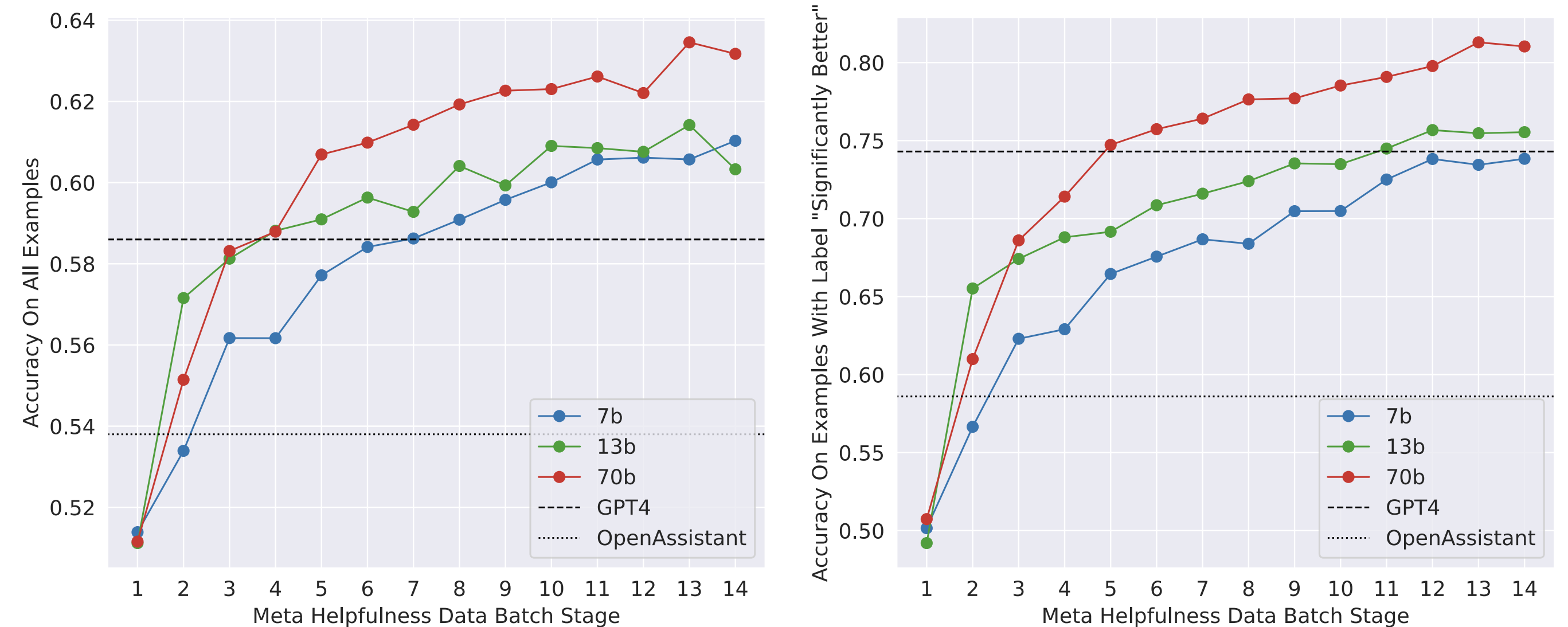


Figure 6: Scaling trends for the reward model. More data and a larger-size model generally improve accuracy, and it appears that our models have not yet saturated from learning on the training data.

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

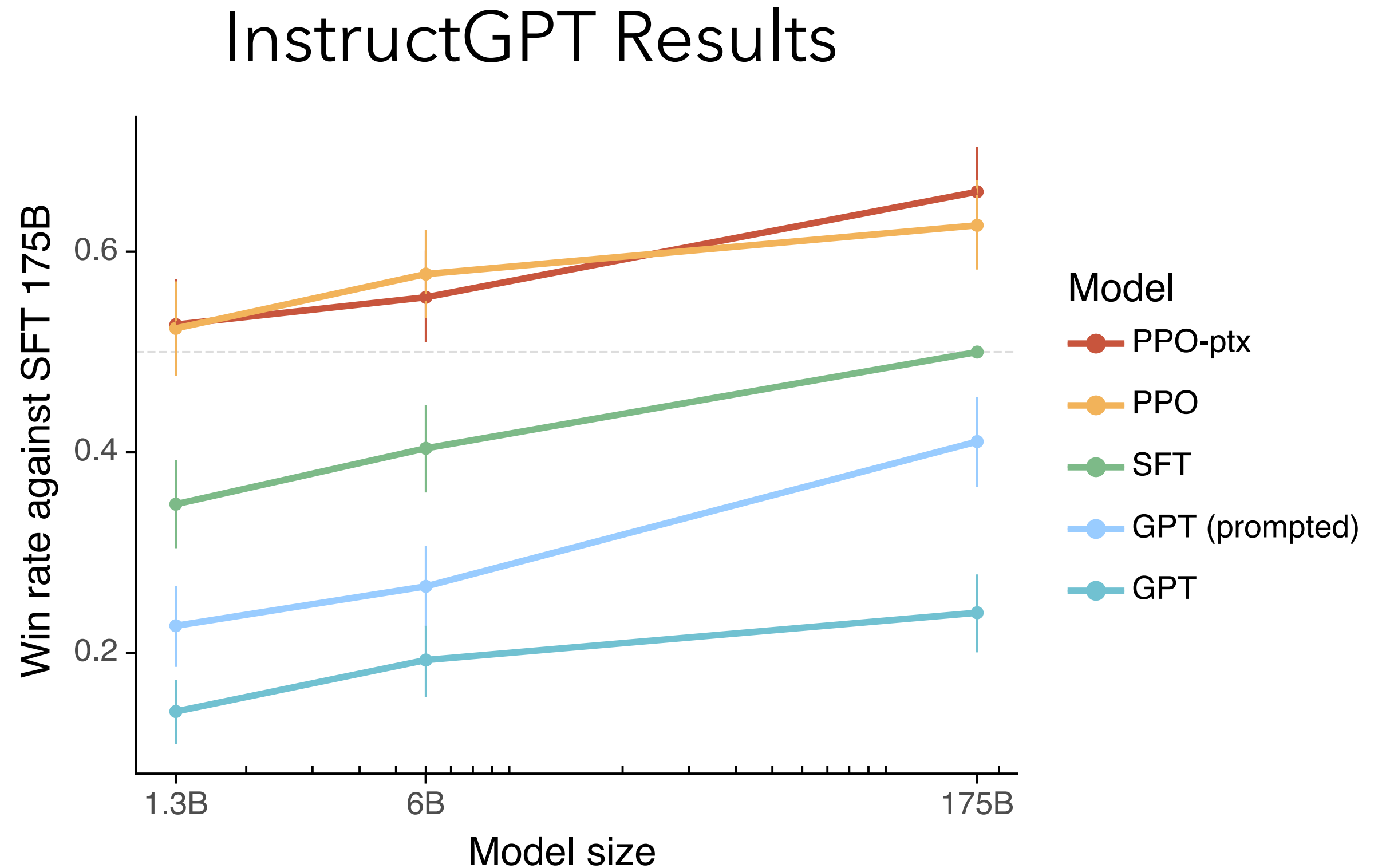


Figure 1: Human evaluations of various models on our API prompt distribution, evaluated by how often outputs from each model were preferred to those from the 175B SFT model. Our InstructGPT models (PPO-ptx) as well as its variant trained without pretraining mix (PPO) significantly outperform the GPT-3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

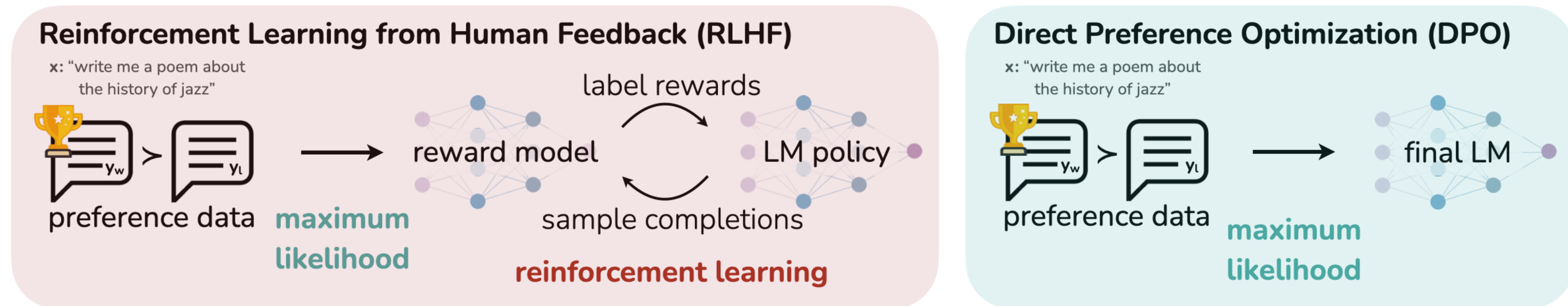
- The entire process is based on fixed annotated data

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

- 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:

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

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:

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

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

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

- 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

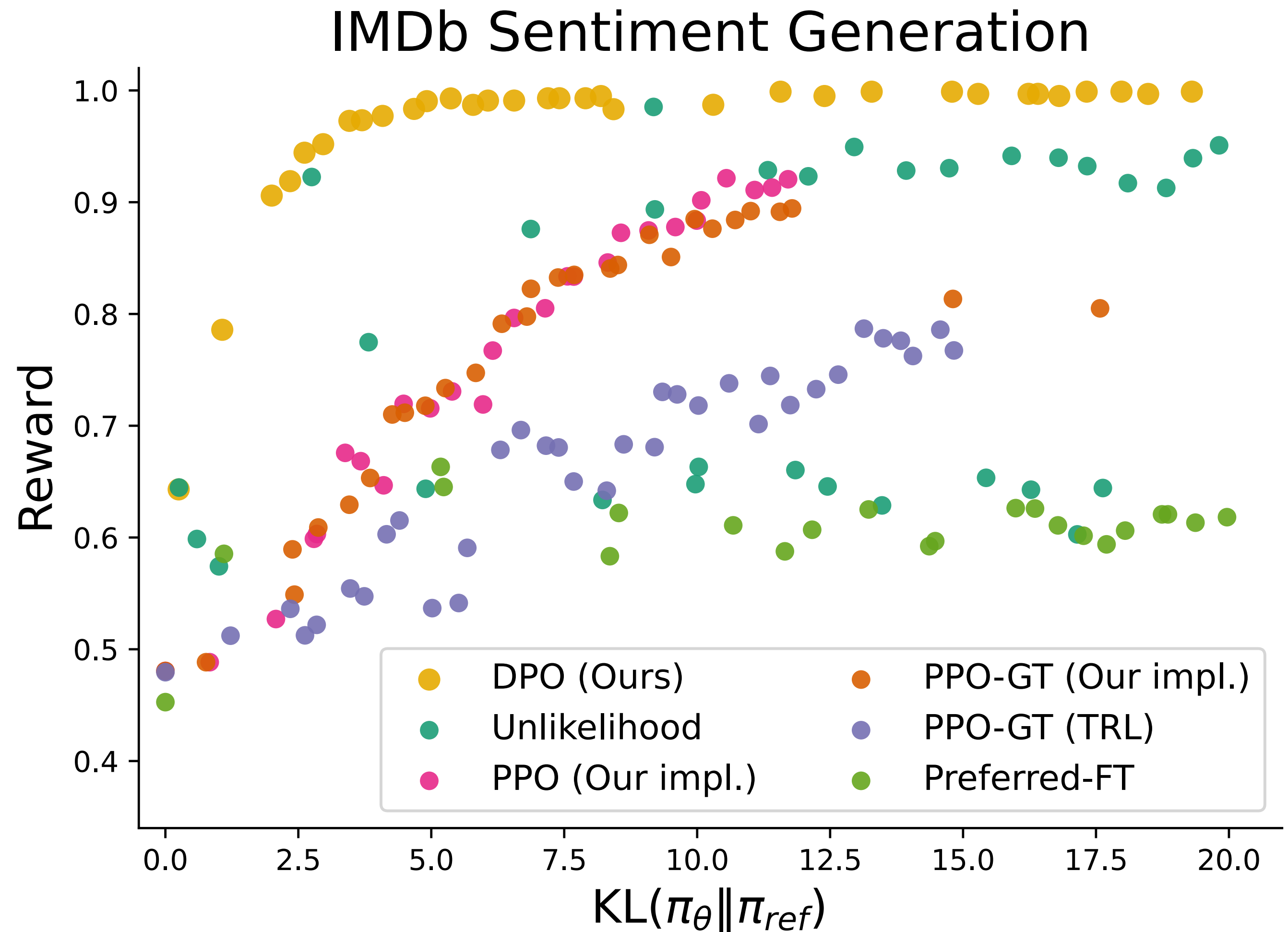
Comparison to RLHF

- 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

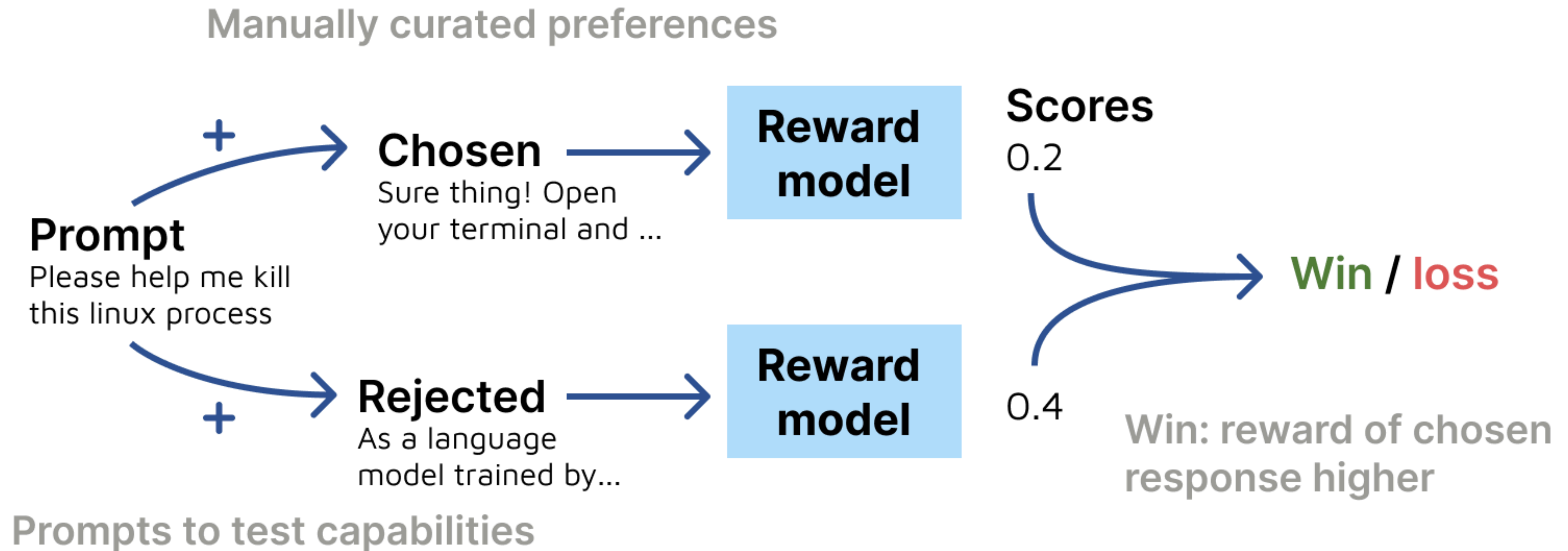























Figure 1: The scoring method of the REWARD BENCH evaluation suite. Each prompt is accompanied by a chosen and rejected completion which are independently rated by a reward model.

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

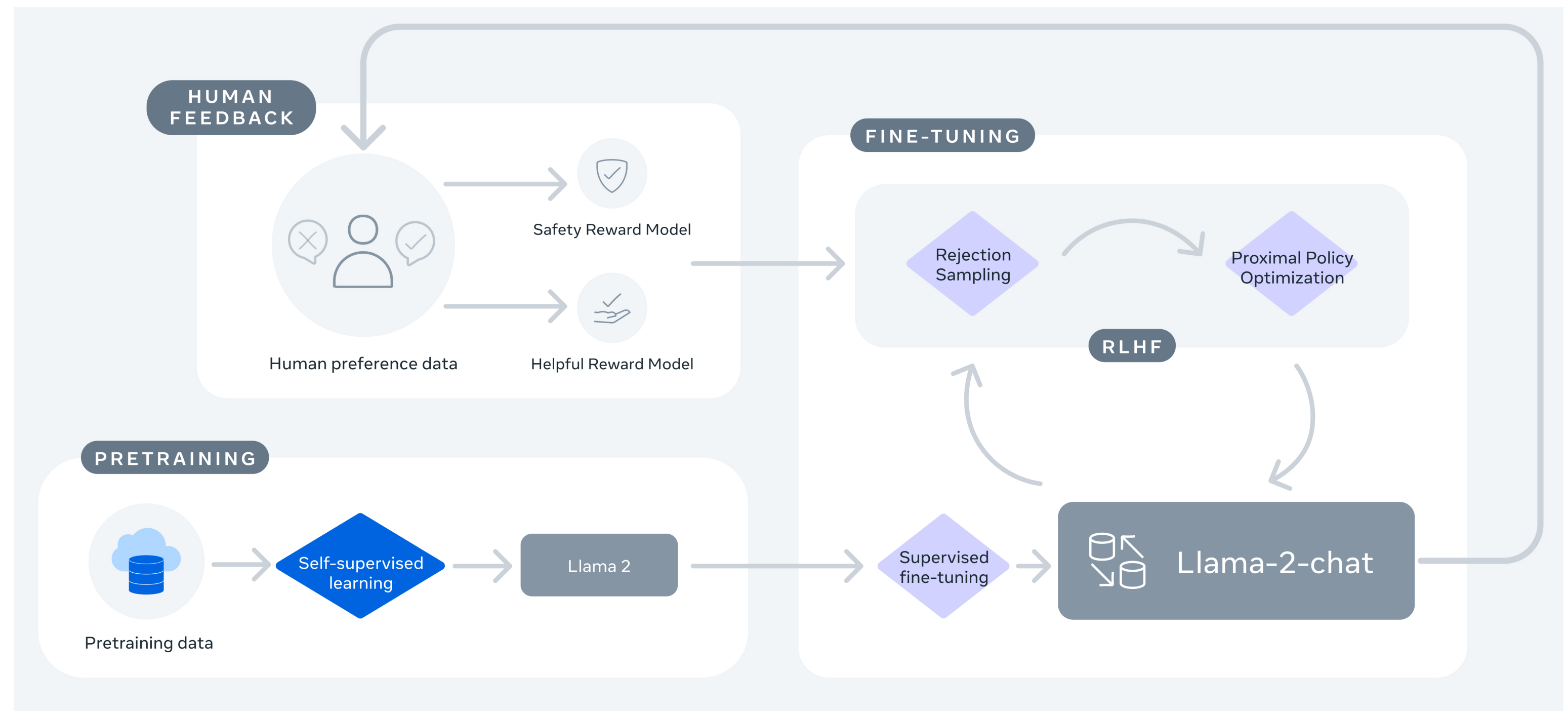


Figure 4: Training of LLAMA 2-CHAT: This process begins with the **pretraining** of LLAMA 2 using publicly available online sources. Following this, we create an initial version of LLAMA 2-CHAT through the application of **supervised fine-tuning**. Subsequently, the model is iteratively refined using 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

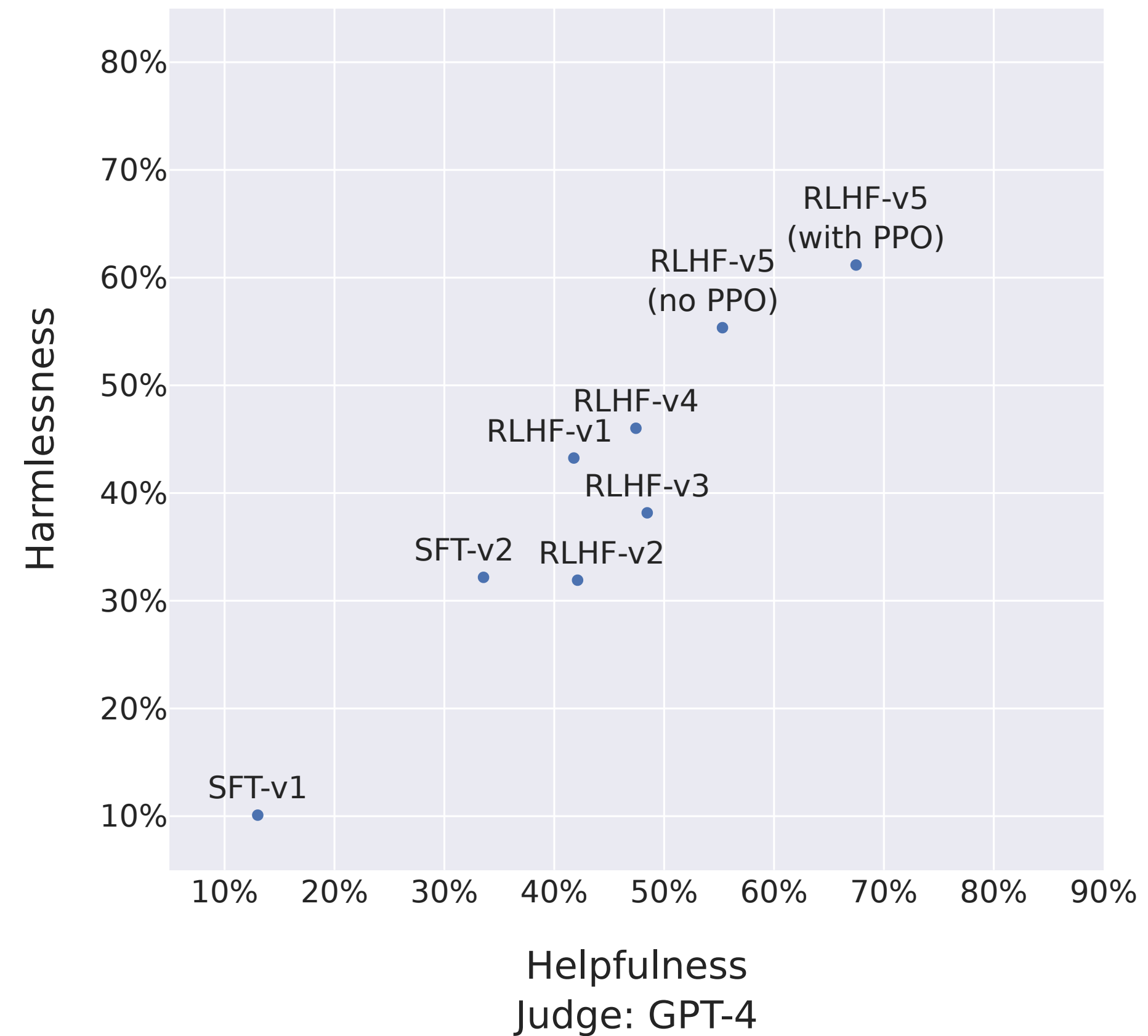
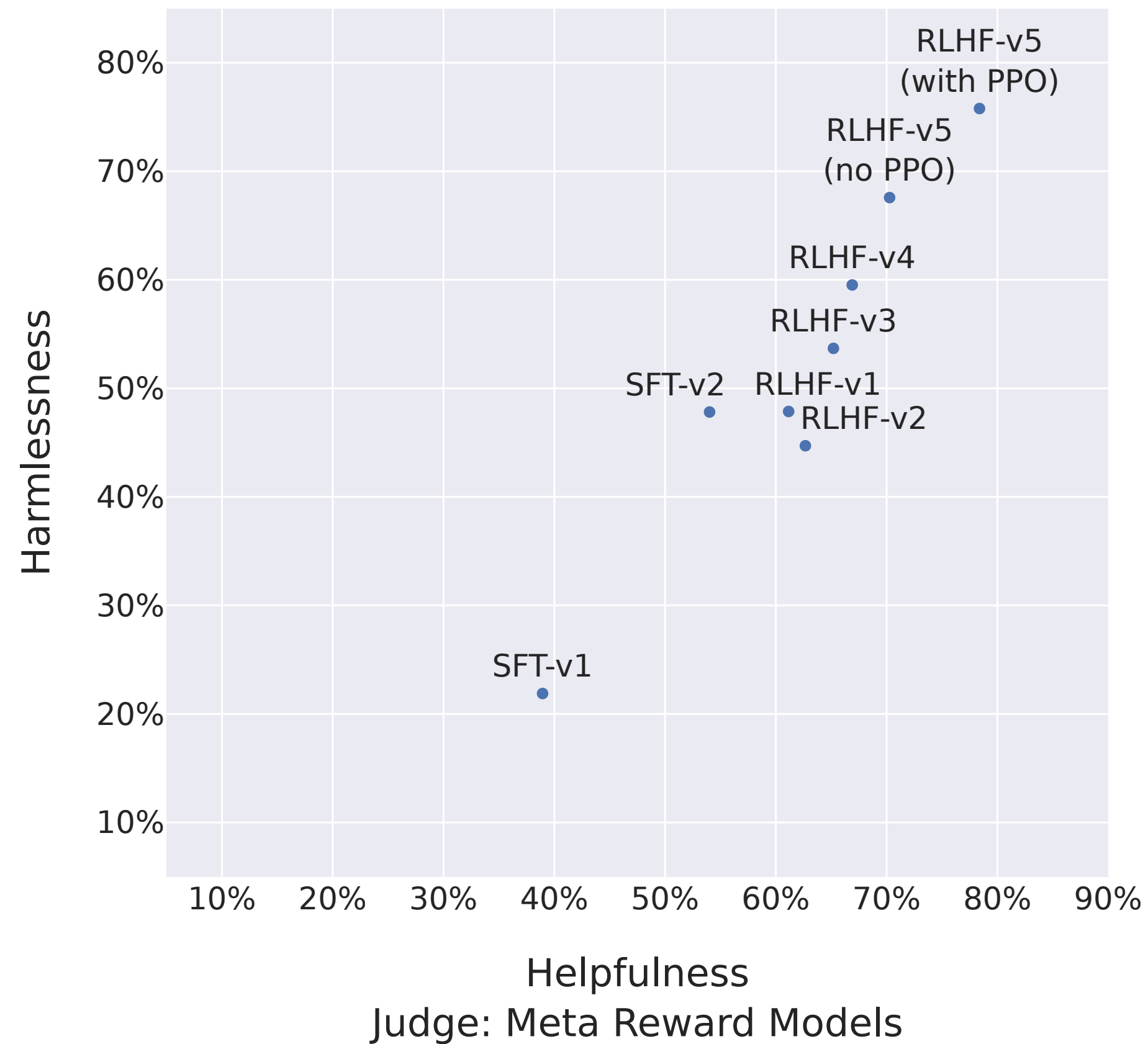


Figure 11: Evolution of LLAMA 2-CHAT. We show the evolution after multiple iterations fine-tuning for the win-rate % of LLAMA 2-CHAT compared to ChatGPT. *Left:* the judge is our reward model, which may favor our model, and *right,* the judge is GPT-4, which should be more neutral.

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]

Reasoning in LLMs

- So far, we've talked about adapting LLMs to be chatbots
- How do we handle challenging questions that need *reasoning*?
 - Hard questions that require multi-step answers
 - Solving complex math problems
- Can we use test-time compute more effectively?
 - Spend more time computing, get a better answer

Recall: chain-of-thought prompting

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ❌

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23 - 20 = 3$. They bought 6 more apples, so they have $3 + 6 = 9$. The answer is 9. ✅

Give examples of reasoning in the prompt, making it more likely for “complete-the-sequence” to yield the right answer.

Recall: chain-of-thought prompting

Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 **X**

Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let's think step by step.**

(Output) *There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓*

Adding the text "Let's think step by step" has a similar effect

Source: [Kojima et al., "Large Language Models are Zero-Shot Reasoners", 2022]

Training on math and technical text

Question: Assume that the variance of the first n natural numbers is 10, and the variance of the first m even natural numbers is 16. Compute $m + n$.

Model output: Let σ^2 be the variance of the first m even natural numbers, and τ^2 be the variance of the first n natural numbers. Then, we have $\tau^2 = \frac{n^2-1}{12}$ and $\sigma^2 = \frac{m^2-1}{3}$. We solve the system of equations:

$$\begin{aligned}\frac{n^2 - 1}{12} &= 10 \\ \frac{m^2 - 1}{3} &= 16.\end{aligned}$$

This translates into the system

$$\begin{aligned}n^2 &= 121 \\ m^2 &= 48 + 1 = 49.\end{aligned}$$

Therefore, $n = \boxed{11}$ and $m = \boxed{7}$, so $n + m = \boxed{18}$.

Question: For every $a, b, b \neq a$ prove that

$$\frac{a^2 + b^2}{2} > \left(\frac{a + b}{2}\right)^2.$$

Model output:

$$\begin{aligned}\frac{a^2+b^2}{2} &> \left(\frac{a+b}{2}\right)^2 \\ \iff \frac{a^2+b^2}{2} &> \frac{a^2+b^2+2ab}{4} \\ \iff a^2 + b^2 &> \frac{a^2+b^2+2ab}{2} \\ \iff 2a^2 + 2b^2 &> a^2 + b^2 + 2ab \\ \iff a^2 + b^2 &> 2ab \\ \iff a^2 + b^2 - 2ab &> 0 \\ \iff (a - b)^2 &> 0\end{aligned}$$

which is true, because the square of a real number is positive.

Can we train a model to reliably do this type of reasoning?

Learning to reason via RL

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within `<think>...</think>` and `<answer>...</answer>` tags, respectively, i.e., `<think> reasoning process here </think>` `<answer> answer here </answer>`. User: **prompt**. Assistant:

- Prompt the model to the answer within `<answer>` `</answer>` tags.
- Also ask it to put reasoning within `<think>``</think>` tags. There's no explicit constraint on what's in there.
- Use reinforcement learning to force it to generate the right results!

Guess and check

- How do we define the reward function? One option: *outcome supervision*
- **Accuracy reward:** did you get the right answer?
 - For math problems with a deterministic result, put answer in a box and perform rule-based verification.
 - For code generation, compile the code and run unit tests.
- **Formatting reward:** encourage the model to put its reasoning within the `<think>` tags.

Recipe from ["DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning", 2025]

Guess and check

- Train using a policy gradient methods
- DeepSeek-R1 trained using a simple PPO-like called Group Relative Policy Optimization (GRPO).
- Instead of training a value function, it measures the improvement by computing the mean and variance of the reward for question x using the previous policy.

Recipe from ["DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning", 2025]

Table 2 | An interesting “aha moment” of an intermediate version of DeepSeek-R1-Zero. The model learns to rethink using an anthropomorphic tone. This is also an aha moment for us, allowing us to witness the power and beauty of reinforcement learning.

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a + x}} = x$, let's start by squaring both ...

$$\left(\sqrt{a - \sqrt{a + x}}\right)^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step-by-step to identify if the correct sum can be ...

We started with the equation:

$$\sqrt{a - \sqrt{a + x}} = x$$

First, let's square both sides:

$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

Next, I could square both sides again, treating the equation: ...

...

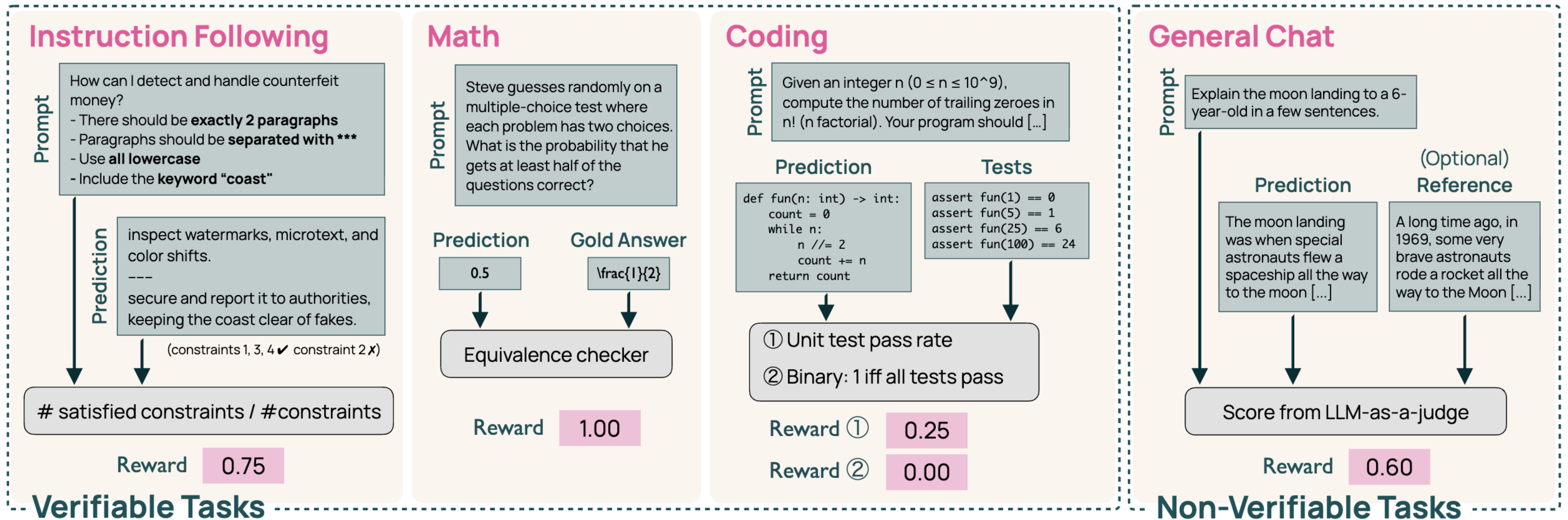
Big improvements on coding, math, and STEM Q&A, in comparison to the base models (and model post-trained without reasoning).

Table 12 | A Comparative Analysis of DeepSeek-V3 and DeepSeek-R1. DeepSeek-V3 is a non-reasoning model developed on top of DeepSeek-V3-Base, which also serves as the foundational base model for DeepSeek-R1. Numbers in bold denote the performance is statistically significant (t-test with $p < 0.01$).

	Benchmark (Metric)	V3-Base	V3	R1-Zero	R1
English	MMLU (EM)	87.1	88.5	88.8	90.8
	MMLU-Redux (EM)	86.2	89.1	85.6	92.9
	MMLU-Pro (EM)	64.4	75.9	68.9	84.0
	DROP (3-shot F1)	89.0	91.6	89.1	92.2
	IF-Eval (Prompt Strict)	58.6	86.1	46.6	83.3
	GPQA Diamond (Pass@1)	-	59.1	75.8	71.5
	SimpleQA (Correct)	20.1	24.9	30.3	30.1
	FRAMES (Acc.)	-	73.3	82.3	82.5
	AlpacaEval2.0 (LC-winrate)	-	70.0	24.7	87.6
	ArenaHard (GPT-4-1106)	-	85.5	53.6	92.3
Code	LiveCodeBench (Pass@1-COT)	-	36.2	50.0	65.9
	Codeforces (Percentile)	-	58.7	80.4	96.3
	Codeforces (Rating)	-	1134	1444	2029
	SWE Verified (Resolved)	-	42.0	43.2	49.2
	Aider-Polyglot (Acc.)	-	49.6	12.2	53.3
Math	AIME 2024 (Pass@1)	-	39.2	77.9	79.8
	MATH-500 (Pass@1)	-	90.2	95.9	97.3
	CNMO 2024 (Pass@1)	-	43.2	88.1	78.8
Chinese	CLUEWSC (EM)	82.7	90.9	93.1	92.8
	C-Eval (EM)	90.1	86.5	92.8	91.8
	C-SimpleQA (Correct)	-	68.0	66.4	63.7

Source: ["DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning", 2025]

Verifiers for different domains



Making the chain-of-thought more readable

Listing 1 | Prompt for producing a human-readable solution.

```
## Question
{question}

## Thought process
{thought_process}

---
Based on the above thought process, provide a clear, easy-to-follow, and well-formatted
solution to the question. Use the same language as the question.

The solution must strictly follow these requirements:
- Stay faithful and consistent with the given thought process. Do not add new reasoning
  steps or conclusions not shown in the original.
- Show key steps leading to final answer(s) in clear, well-formatted LaTeX.
- Use \boxed{} for final answer(s).
- Be clean and concise. Avoid colloquial language. Do not use phrases like "thought
  process" in the solution.

Your response should start with the solution right away, and do not include anything
else. Your task is solely to write the solution based on the provided thought
process. Do not try to solve the question yourself.
```

- Nothing constrains thought traces to be human-readable.
- For example, might mix two spoken languages together.
- Use extra rewards (using LLMs) and prompting to address this.
- Still no guarantee that the reasoning steps are correct or are an honest representation of the LLM's internal process!

Rewarding intermediate steps

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x) \cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

I notice that the given polynomial has even degree and only even powers of x , so I can try to make a substitution to simplify it.

Let $y = x^4$, then the polynomial becomes $y^2 + 3y - 4$, which is a quadratic equation.

I can factor this quadratic equation as $(y + 4)(y - 1)$, so the original polynomial is $(x^4 + 4)(x^4 - 1)$.

Now I need to factor each of these factors further, if possible, over the integers.

For the first factor, $x^4 + 4$, I recall the Sophie Germain identity, which says that $a^4 + 4b^4 = (a^2 + 2b^2 + 2ab)(a^2 + 2b^2 - 2ab)$.

Applying this identity with $a = x$ and $b = 1$, I get $x^4 + 4 = (x^2 + 2x + 2)(x^2 - 2x + 2)$.

For the second factor, $x^4 - 1$, I can use the difference of squares formula, which says that $a^2 - b^2 = (a + b)(a - b)$.

Applying this formula with $a = x^2$ and $b = 1$, I get $x^4 - 1 = (x^2 + 1)(x^2 - 1)$.

Then I can apply the difference of squares formula again with $a = x$ and $b = 1$, to get $x^2 - 1 = (x + 1)(x - 1)$.

So the original polynomial is $(x^4 + 4)(x^4 - 1) = (x^2 + 2x + 2)(x^2 - 2x + 2)(x^2 + 1)(x + 1)(x - 1)$.

These are all the irreducible factors over the integers, so $p_1(x) = x^2 + 2x + 2$, $p_2(x) = x^2 - 2x + 2$, $p_3(x) = x^2 + 1$, $p_4(x) = x + 1$, and $p_5(x) = x - 1$.

To find the sum of their values at $x = 1$, I just need to plug in $x = 1$ into each factor and add them up.

I get $p_1(1) = 1^2 + 2 \cdot 1 + 2 = 5$, $p_2(1) = 1^2 - 2 \cdot 1 + 2 = 1$, $p_3(1) = 1^2 + 1 = 2$, $p_4(1) = 1 + 1 = 2$, and $p_5(1) = 1 - 1 = 0$.

So the final answer is $5 + 1 + 2 + 2 + 0 = 10$.

Answer: 10

Let

$$x^8 + 3x^4 - 4 = p_1(x)p_2(x) \cdots p_k(x),$$

where each non-constant polynomial $p_i(x)$ is monic with integer coefficients, and cannot be factored further over the integers. Compute $p_1(1) + p_2(1) + \cdots + p_k(1)$.

This problem is about factoring a polynomial over the integers, which can be a tricky task.

One possible strategy is to look for rational roots of the polynomial, using the rational root theorem.

However, that would involve checking a lot of possible factors of -4 , and there is no guarantee that the polynomial has any rational roots at all.

Another possible strategy is to use some algebraic tricks to simplify the polynomial or reduce its degree.

I notice that the polynomial has only even powers of x , so I can try to substitute $y = x^2$ and see what happens.

Then the polynomial becomes $y^4 + 3y^2 - 4$, which is a quadratic equation in y^2 .

I can try to factor this quadratic equation by looking for two numbers that multiply to -4 and add to 3 .

Those numbers are 4 and -1 , so I can write $y^4 + 3y^2 - 4 = (y^2 + 4)(y^2 - 1)$.

Now I can substitute back $x^2 = y$ and get $x^8 + 3x^4 - 4 = (x^4 + 4)(x^4 - 1)$.

This looks promising, because I have reduced the degree of the factors from 8 to 4 .

I can try to factor each of these factors further, using the difference of squares formula.

I get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x^2 - 1)$.

I can apply the difference of squares formula again to the last factor and get $x^8 + 3x^4 - 4 = (x^2 + 2)(x^2 - 2)(x^2 + 1)(x + 1)(x - 1)$.

Now I have factored the polynomial completely into monic linear and quadratic factors with integer coefficients.

These are the $p_i(x)$'s that the problem is asking for.

To find the sum of their values at $x = 1$, I just need to plug in $x = 1$ into each factor and add them up.

I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (1^2 + 2)(1^2 - 2)(1^2 + 1)(1 + 1)(1 - 1)$.

Simplifying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = (3)(-1)(2)(2)(0)$.

Multiplying, I get $p_1(1) + p_2(1) + \cdots + p_k(1) = 0$.

Answer: 0

Figure 2: Two solutions to the same problem, graded by the PRM. The solution on the left is correct while the solution on the right is incorrect. A green background indicates a high PRM score, and a red background indicates a low score. The PRM correctly identifies the mistake in the incorrect solution.

- Can also reward intermediate steps of reasoning (called *process supervision*).
- For example, reward the model for producing logically valid reasoning steps.
- More precise feedback, makes credit assignment easier.
- Much harder to define the reward
- Very susceptible to reward hacking.

Training a reasoning model

Category	Prompt Dataset	# Prompts Used in Think RL	# Prompts Used in Instruct RL	Reference
Precise IF	IF-RLVR	30,186	38,000	Pyatkin et al. (2025)
Math	Open-Reasoner-Zero	3,000	14,000	Hu et al. (2025)
	DAPO-Math	2,584	7,000	Yu et al. (2025)
	AceReason-Math	6,602	–	Chen et al. (2025)
	Polaris-Dataset	–	14,000	An et al. (2025)
	KlearReasoner-MathSub	3,000	9,000	Su et al. (2025c)
	OMEGA-train	15,000	20,000	Sun et al. (2025)
Coding	AceCoder	9,767	20,000	Zeng et al. (2025a)
	KlearReasoner-Code	8,040	–	Su et al. (2025c)
	Nemotron Post-training Code	2,303	–	NVIDIA AI (2025)
	SYNTHETIC-2	3,000	–	PrimeIntellect (2025)
General Chat	Tulu 3 SFT	7,129	18,955	Lambert et al. (2024)
	Wildchat-4.8M	7,129	18,761	-
	Multi-Subject RLVR	7,129	12,234	Su et al. (2025b)
Total		104,869	171,950	

Table 20 Breakdown of datasets in Dolci-Think-RL used for RL training. See §4.4.2 for further details on how each dataset is processed.

Source: Olmo 3 paper

Code execution and tool use

```
<|user|>
What is the 50th Fibonacci number? (Use the standard F_0=0, F_1=1 indexing.)</s>
<|assistant|>
<think>
Okay, I will compute the 50-th Fibonacci number with a simple loop, then return the
result.

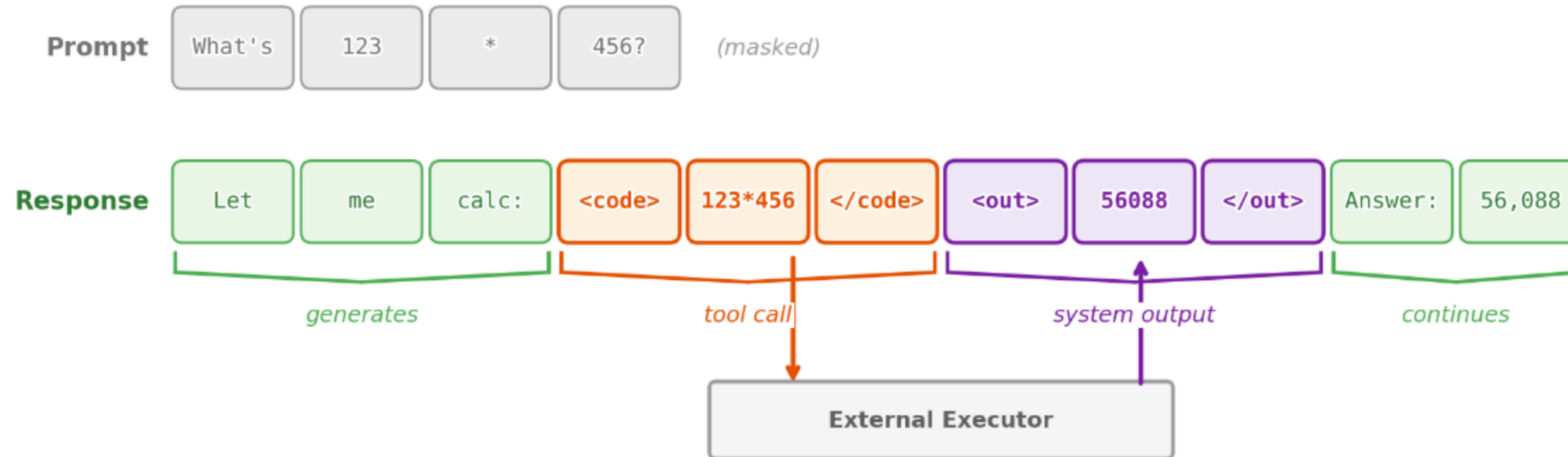
<code>
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

fib(50)
</code>

<output>
12586269025
</output>
</think>
<answer>
The 50-th Fibonacci number is 12 586 269 025.
</answer>
```

- Add special tokens that permit a LM to call external tools or execute code.
- Concatenate the output with the text and continue generation.

Code execution and tool use



- Add special tokens that permit a LM to call external tools or execute code.
- Concatenate the output with the text and continue generation.

Challenges in LLMs with RL

- Hard to deal with tasks with non-verifiable rewards
- Weaker reward signal than supervised learning
- Hard to define rewards
- More computationally expensive than normal autoregressive training and has major engineering challenges.
 - Requires generating text at training time (can't use teacher forcing)
 - Sequences are of variable length

Next class: scaling generative models + more review