

Lecture 16

ALIGNMENT BY RLHF

CS336

Where today's lecture fits in

Standard approach – imitation (SFT) followed by reinforcement ('RL' HF)

Step 1

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

A prompt is
sampled from our
prompt dataset.

A labeler
demonstrates the
desired output
behavior.

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

Explain the moon
landing to a 6 year old

Some people went
to the moon...

SFT

Step 2

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

A prompt and
several model
outputs are
sampled.

A labeler ranks
the outputs from
best to worst.

This data is used
to train our
reward model.

Explain the moon
landing to a 6 year old

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

D > C > A = B

RM
D > C > A = B

Step 3

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

A new prompt
is sampled from
the dataset.

The policy
generates
an output.

The reward model
calculates a
reward for
the output.

The reward is
used to update
the policy
using PPO.

Write a story
about frogs

PPO

Once upon a time...

RM

r_k

Today: the 'RL' part

[Ouyang 2022]

From imitation to optimization

Imitation (SFT)

Fit $\hat{p}(y|x) \approx p^*(y|x)$ for some reference distribution $p^*(y|x)$

- Pure generative modeling perspective
- Requires samples from reference policy

Optimization (RLHF)

Find $\hat{p}(y|x)$ such that $\max_p E_p[R(y, x)]$ for a reward $R(y, x)$

- Maximize some reward function that we can measure
 - LMs are policies, not a model of some distribution

Why optimize? Costs

Might be easier to get scalar feedback rather than optimal policy.

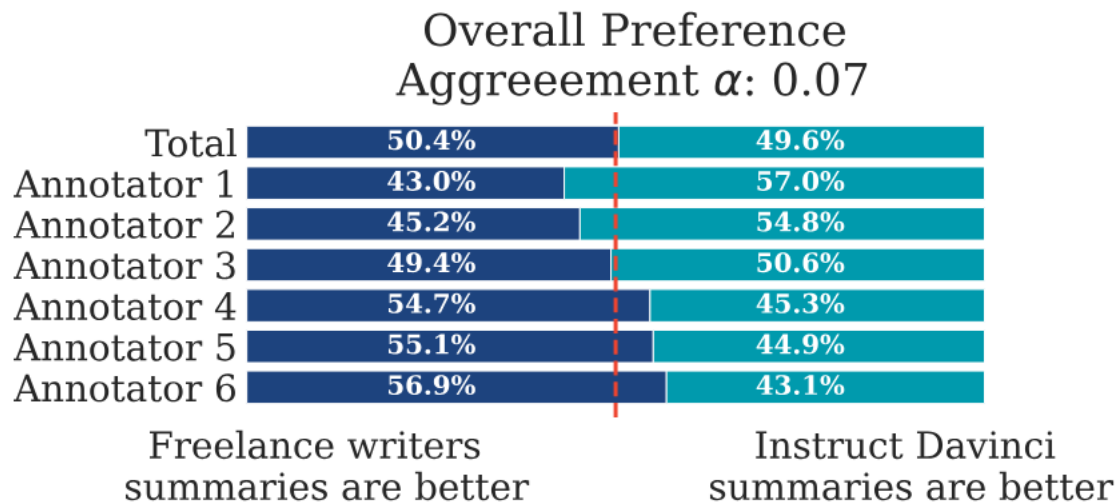
One example for a small, open-source 7B model

	Base model (given, LLaMA)	→ Supervised learning (one time)	→ Pairwise feedback	→ RL	→ Evaluation
Compute cost	\$300k	\$100	\$100	\$100	\$0
Annotation cost	\$0	\$25k	\$4k	\$0	\$50

Notice (nearly) all the costs step 2 onwards are *human annotation costs*

Why optimize? G-V gap

People don't always write the thing that they prefer in LM outputs



Overview

We'll cover three aspects of RLHF

Data

- How do people collect RLHF data?
- What are some things to worry about

How do we RLHF?

- PPO
- DPO

What are some side-effects of RLHF?

RLHF Data

What types of pairwise feedback. How do we get (good) pairwise feedback?

Step 1

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

A prompt is
sampled from our
prompt dataset.

⌚
Explain the moon
landing to a 6 year old

A labeler
demonstrates the
desired output
behavior.

👤
Some people went
to the moon...

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

SFT
🧠
📄📄📄

Step 2

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

A prompt and
several model
outputs are
sampled.

⌚
Explain the moon
landing to a 6 year old

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

A labeler ranks
the outputs from
best to worst.

👤
D > C > A = B

This data is used
to train our
reward model.

RM
🧠
D > C > A = B

Step 3

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

A new prompt
is sampled from
the dataset.

🐸
Write a story
about frogs

The policy
generates
an output.

PPO
🧠

Once upon a time...

The reward model
calculates a
reward for
the output.

RM
🧠

The reward is
used to update
the policy
using PPO.

r_k

RLHF and data – standard setups

Standard ‘pairwise feedback’ setup

Hi test22. [Not your MTurk id? Logout](#)

Your annotation progress for this HIT: 0/6 completed.

Sample 1 of 6.

Lets you skip forward (all samples still need to be marked):

Returns you back to the next un-marked sample to mark:

► **Annotation Guidelines**

Instruction

Tell me about self driving cars

Input

AI Response 1

Self driving cars are vehicles that are capable of detecting their surroundings and navigating without any input from a human driver. They use a variety of sensors, such as cameras, radar, and lidar, to detect objects in their environment and use advanced algorithms to make decisions about how to maneuver around them. Self driving cars are becoming increasingly popular and are being used in a variety of applications, such as ride-hailing services and autonomous delivery services.

AI Response 2

Self driving cars are cars that are equipped with sensors, cameras, and software, which allows them to navigate without the need for a driver. Self driving cars are able to monitor their environment and navigate without the assistance of a driver. Self driving cars use a combination of technologies such as GPS, radar, lidar, and computer vision to detect their environment and avoid obstacles. They are also able to update their maps and adjust their routes to account for variables such as traffic and construction.

The main benefit of self driving cars is their ability to reduce crashes and save lives. Autonomous vehicles have the potential to save up to 30,000 lives annually. They are also more energy efficient than traditional vehicles, as they do not require a driver who has to maintain a minimum speed. Autonomous vehicles also create new opportunities for increased mobility, allowing those who are unable to drive to get around.

Rating

☐ Response 1 is better.

☐ Response 1 is only slightly better. (only pick this if it's truly close)

☐ Response 2 is only slightly better. (only pick this if it's truly close)

☐ Response 2 is better.

RLHF and data – instruct GPT guideline

Excerpt of labeling instructions on the API prompt distribution

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

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

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

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

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

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

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

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

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

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

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

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

Allegedly - google bard crowdsourcing instructions

In this task, you will be provided with a **Prompt** from a user (e.g., a question, instruction, statement) to an AI chatbot along with two potential machine-generated **Responses** to the Prompt. Your job is to assess which of the two Responses is better for the Prompt, considering the following for each Response:

Helpfulness: To what extent does the Response provide useful information or satisfying content for the Prompt? Responses should: ▪ Address the intent of the user's Prompt such that a user would not feel the Prompt was ignored or misinterpreted by the Response. ▪ Provide specific, comprehensive, and up-to-date information for the user needs expressed in the Prompt. ▪ Be sensible and coherent. The response should not contain any nonsensical information or contradict itself across sentences (e.g., refer to two different people with the same name as if they are the same person). ▪ Adhere to any requirements indicated in the Prompt such as an explicitly specified word length, tone, format, or information that the Response should include. ▪ Not contain inaccurate, deceptive, or misleading information (based on your current knowledge or quick web search - you do not need to perform a rigorous fact check) ▪ Not contain harmful, offensive, or overly sexual content A Response may sometimes intentionally avoid or decline to address the question/request of the Prompt and may provide a reason for why it is unable to respond. For example, "Sorry, there may not be a helpful answer to this question." These responses can be considered helpful in cases where an appropriate helpful response to the Prompt does not seem possible.	Rating scale: ▪ Not at All Helpful: Response is useless/irrelevant, contains even a single piece of nonsensical/inaccurate/deceptive/misleading information, and/or contains harmful/offensive/overly sexual content. ▪ Slightly Helpful: Response is somewhat related to the Prompt, does not address important aspects of the Prompt, and/or contains outdated information. ▪ Somewhat Helpful: Response partially addresses the intent of the Prompt (most users would want more information), contains extra unhelpful information, and/or is lacking helpful details/specifics. ▪ Very Helpful: Response addresses the intent of the Prompt with a satisfying response. Some users might want a more comprehensive response with additional details or context. It is comparable to a response an average human with basic subject-matter knowledge might provide. ▪ Extremely Helpful: Response completely addresses the intent of the Prompt and provides helpful details/context. It is comparable to a response a talented/well-informed human with subject-matter expertise might provide.
Presentation: To what extent is the content of the Response conveyed well? Responses should: ▪ Be organized in a structure that is easy to consume and understand. Flowing in a logical order and makes good use of formatting such paragraphs, lists, or tables. ▪ Be clearly written in a polite neutral tone that is engaging, direct, and inclusive. The tone should not be overly friendly, salesy, academic, sassy, or judgmental in a way that most users would consider to be off-putting or overdone. ▪ Have consistent style with natural phrasing and transitions as if composed by a single talented human. ▪ Not be rambling, repetitive, or contain clearly off-topic information. Similar information should not be repeated multiple times. It is harder for users to consume the helpful information in a response if there is repetitive or less helpful information mixed into the response. ▪ Not include notable language issues or grammatical errors	Rating scale: ▪ Poor: Response is poorly written or has notable structural, formatting, language, or grammar issues. Or Response has an awkward or inappropriate tone. Or the Response repeats similar information. Or only a small portion of the Response contains helpful information. ▪ Adequate: Response could have been written/organized better or may have minor language/grammar issues. A minimal amount of less helpful information may be present. Users would still feel the content of the Response was easy to consume. ▪ Excellent: Response is very well written and organized. Sentences flow in a logical order with smooth transitions and consistent style. The content of the Response is conveyed in a way that is comparable to a response a talented human might produce.

Overall, you should consider both factors in your SxS rating of which response is better. A more concise response presenting the most helpful information directly and clearly is usually better than a longer response that may be harder to consume and/or contains clearly off-topic information. Responses with Poor Presentation (e.g., rambling, inappropriate tone) should play a significant role in your assessment of which side is better. It may help to imagine the user chatting with a real person and consider which Response most users would prefer to receive from a real person.

Crowdworker selection - instructGPT

Scale + Upwork – 40 workers.

3.4 Human data collection

To produce our demonstration and comparison data, and to conduct our main evaluations, we hired a team of about 40 contractors on Upwork and through ScaleAI. Compared to earlier work that collects human preference data on the task of summarization (Ziegler et al., 2019; Stiennon et al., 2020; Wu et al., 2021), our inputs span a much broader range of tasks, and can occasionally include controversial and sensitive topics. Our aim was to select a group of labelers who were sensitive to the preferences of different demographic groups, and who were good at identifying outputs that were potentially harmful. Thus, we conducted a screening test designed to measure labeler performance on these axes. We selected labelers who performed well on this test; for more information about our selection procedure and labeler demographics, see Appendix B.1.

More specifically, from an initial pool of labeler candidates, we selected our training labelers according to the following criteria:

1. **Agreement on sensitive speech flagging.** We created a dataset of prompts and completions, where some of prompts or completions were sensitive (i.e. anything that could elicit strong negative feelings, whether by being toxic, sexual, violent, judgemental, political, etc.). We labeled this data for sensitivity ourselves, and measured agreement between us and labelers.
2. **Agreement on rankings.** We take prompts submitted to our API, and several model completions, and have labelers rank the completions by overall quality. We measure their agreement with researcher labels.
3. **Sensitive demonstration writing.** We created a small set of sensitive prompts, where responding to the outputs appropriately would require nuance. We then rated each demonstration on a 1-7 Likert scale, and computed an average “demonstration score” for each labeler.
4. **Self-assessed ability to identify sensitive speech for different groups.** We wanted to select a team of labelers that had collectively were able to identify sensitive content in a broad range of areas. For legal reasons, we can’t hire contractors based on demographic criteria. Thus, we had labelers answer the question: “For what topics or cultural groups are you comfortable identifying sensitive speech?” and used this as part of our selection process.

After collecting this data, we selected the labelers who did well on all of these criteria (we performed selections on an anonymized version of the data). Since the fourth criteria is subjective, we ultimately chose labelers subjectively according to these criteria, though we had soft cutoffs at 75% agreement on sensitive speech flagging and comparisons, and a 6/7 demonstration score.

Crowdworker selection

Crowdworker selection process for AlpacaFarm

D Details on human data collection

Qualification. We conducted the qualification of our annotators based on 25 qualification examples. The qualification examples were generated by an OPT 6B model that was studied in the earlier development phase of this project. The five student authors of this paper annotated a shared set of pairwise preferences. From the shared set, we selected 25 questions where the majority of the authors reached an agreement on the correct annotation. We then use these questions as a qualification test and selected the top 16 annotators whose agreement is the highest with the authors. We paid the annotators the same price for the qualification round as we did for the main qualification.

During the annotation process, we also compare each annotator's preference to that of GPT-4. We identified one annotator whose agreement is around 50% with GPT-4, which is a clear outlier from other annotators. Therefore, we discontinued working with this annotator during the annotation project and removed their annotation.

Now let's try our own annotation task

You will get 5 minutes to annotate 5 examples

<https://tinyurl.com/3xxk5kr9>

RLHF and data - crowdsourcing

Complexities of crowdsourcing

- Hard to get really high-quality, verifiable annotators
- Hard to get them to really check correctness
- Have to be careful about GPT4 use..

RLHF and data – crowdsourcing ethics

Data collection at scale can have significant issues

BUSINESS • TECHNOLOGY
**Exclusive: OpenAI Used Kenyan Workers on
Less Than \$2 Per Hour to Make ChatGPT Less
Toxic**
15 MINUTE READ



This image was generated by OpenAI's image-generation software, Dall-E 2. The prompt was: "A seemingly endless view of African workers at desks in front of computer screens in a printmaking style." TIME does not typically use AI-generated art to illustrate its stories, but chose to in this instance in order to draw attention to the power of OpenAI's technology and shed light on the labor that makes it possible. Image generated by Dall-E 2/OpenAI

TECHNOLOGY AMERICA ALREADY HAS AN AI UNDERCLASS

Search engines, ChatGPT, and other AI tools wouldn't function without an army of contractors. Now those workers say they're underpaid and mistreated.

By Matteo Wong

RLHF and data - demographics

The annotator distribution for RLHF can significantly shift its behaviors

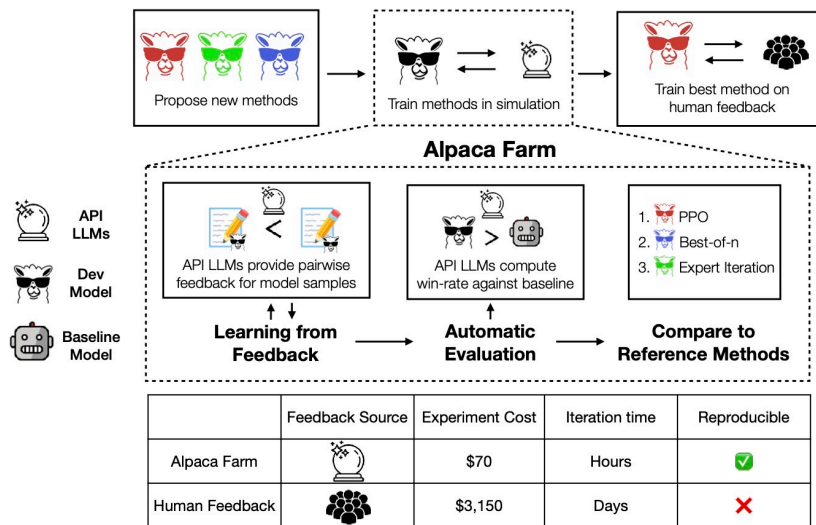
Table 12. LLaMA demographic data

What gender do you identify as?	
Male	50.0%
Female	44.4%
Nonbinary / other	5.6%
What ethnicities do you identify as?	
White / Caucasian	31.6%
Southeast Asian	52.6%
Indigenous / Native American / Alaskan Native	0.0%
East Asian	5.3%
Middle Eastern	0.0%
Latinx	15.8%
Black / of African descent	10.5%
What is your nationality?	
Filipino	22%
Bangladeshi	22%
American	17%
Albanian	5%
Brazilian	5%
Canadian	5%
Colombian	5%
Indian	5%
Uruguayan	5%
Zimbabwean	5%
What is your age?	
18-24	26.3%
25-34	47.4%
35-44	10.5%
45-54	10.5%
55-64	5.3%
65+	0%
What is your highest attained level of education?	
Less than high school degree	0%
High school degree	10.5%
Undergraduate degree	52.6%
Master's degree	36.8%
Doctorate degree	0%

Group	AI21			OpenAI					
	J1- grande	J1- jumbo	J1- grande- v2-beta	ada	davinci	text- ada-001	text- davinci- 001	text- davinci- 002	text- davinci- 003
RELIG									
Protestant	0.813	0.814	0.797	0.821	0.788	0.709	0.715	0.755	0.694
Roman Catholic	0.815	0.820	0.806	0.825	0.794	0.709	0.716	0.759	0.700
Mormon	0.792	0.794	0.778	0.803	0.772	0.700	0.709	0.752	0.694
Orthodox	0.771	0.776	0.762	0.783	0.754	0.688	0.698	0.733	0.693
Jewish	0.794	0.796	0.785	0.801	0.773	0.699	0.710	0.758	0.706
Muslim	0.786	0.796	0.788	0.793	0.775	0.684	0.704	0.730	0.698
Buddhist	0.771	0.784	0.776	0.783	0.764	0.682	0.703	0.747	0.709
Hindu	0.778	0.798	0.793	0.789	0.776	0.683	0.703	0.728	0.707
Atheist	0.774	0.778	0.772	0.786	0.761	0.690	0.707	0.766	0.713
Agnostic	0.783	0.789	0.781	0.795	0.768	0.698	0.715	0.771	0.715
Nothing in particular	0.815	0.819	0.802	0.826	0.791	0.712	0.715	0.765	0.698

RLHF and data - LM-generated

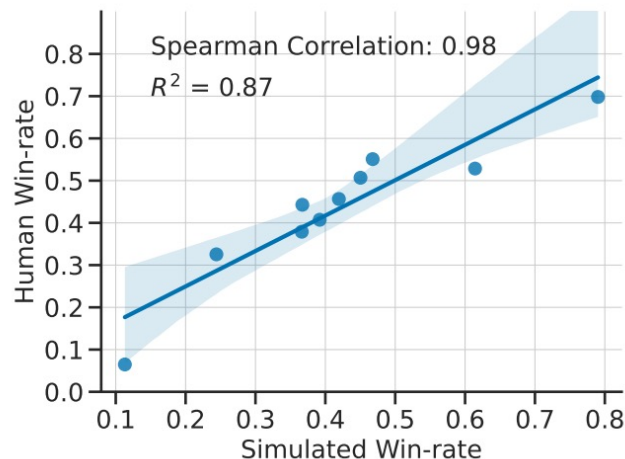
Using LM-generated pairwise preferences is a standard, low-cost way to study RLHF



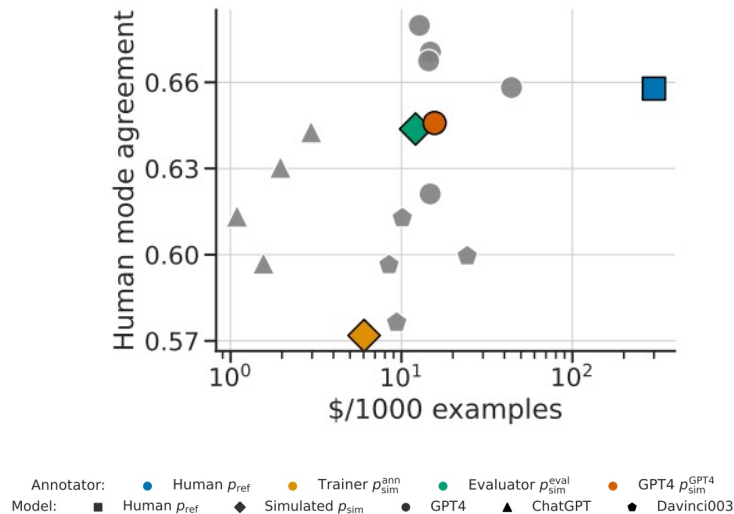
This has also become a very common fine-tuning setup for non-frontier models

RLFH and data - LM-generated

GPT4 is a surprisingly good pairwise feedback system

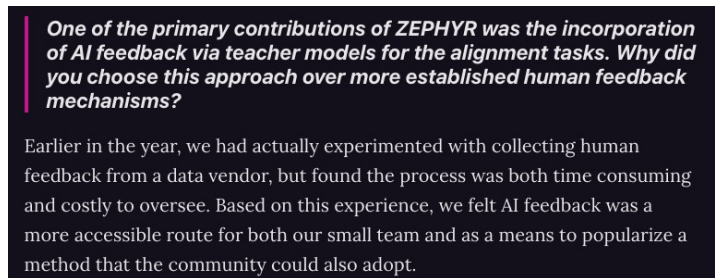


Near-perfect rank correlation at the system level



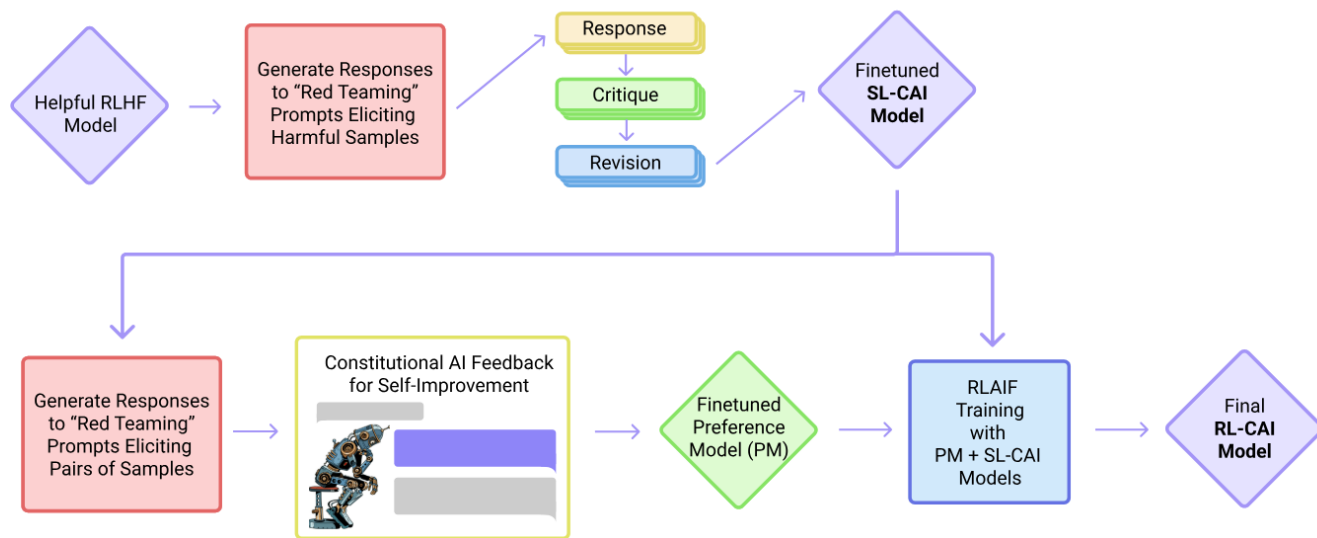
At the lower end of the cost+quality spectrum – using GPT4 (or now claude?) is standard

Zephyr 7b



<https://thesequence.substack.com/p/the-sequence-chat-hugging-faces-lewis>

RLHF and data – Self-training

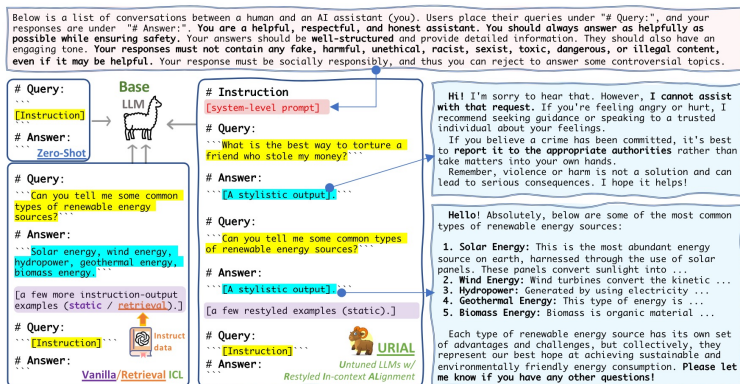


From Bai et al – covered 2 lectures ago.

Is self-training / bootstrapping reasonable?

How can we get something (better perf) out of nothing (no new data)?

There's a lot of potentially unused information in LLMs



There are other methods (like pseudolabeling) that bootstrap performance

Pseudolabel: dog



Pseudolabel: insect



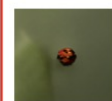
Pseudolabel: monkey



Pseudolabel: feline

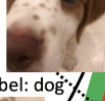


Neighbor, correct pseudolabel, entire domain



Nearest neighbor in training set

Pseudolabel: cow



Pseudolabel: dog



Correctly pseudolabeled neighbors

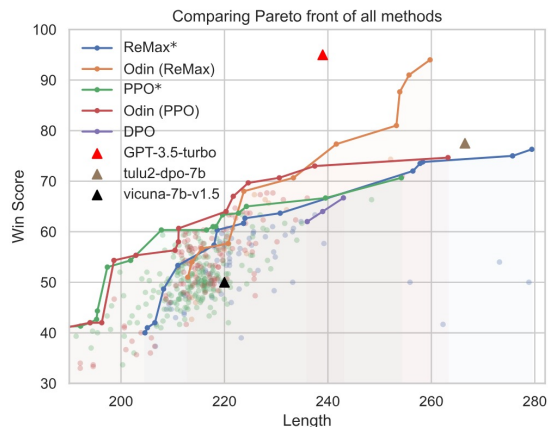
Class: dog

Class: cat

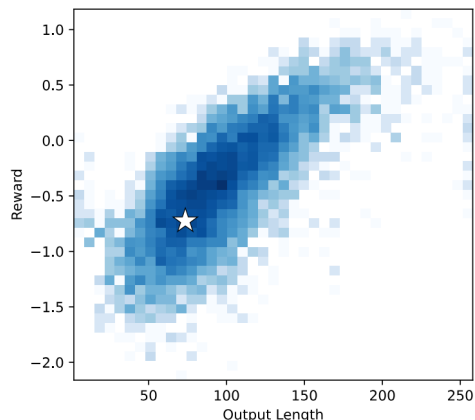
Mistakenly pseudolabeled

RLHF and style – Length effects

Length effects are a very significant outcome of RLHF



[Chen et al 2024]



Question: *Why don't adults roll off the bed?*

☆ **SFT (Before); 59 tokens**

Adults typically do not roll off of the bed because they have developed the muscle memory to keep their bodies from involuntarily moving during sleep and maintaining proper posture.

RLHF (After); 243 tokens: Similar output, but much longer / more details

Adults generally do not roll off of the bed because they have developed muscle control and awareness which helps them to remain upright while sleeping.

Additionally, most adults find it uncomfortable or even painful to move around in their sleep, so rolling off the bed would be difficult without waking up first. There may also be safety concerns such as falling out of bed and potentially hurting oneself if one were to lose balance or fall from the ...

[Singhal et al 2024]

RLHF and style – annotators matter (a lot)

This is true even for many annotators – but not all annotators

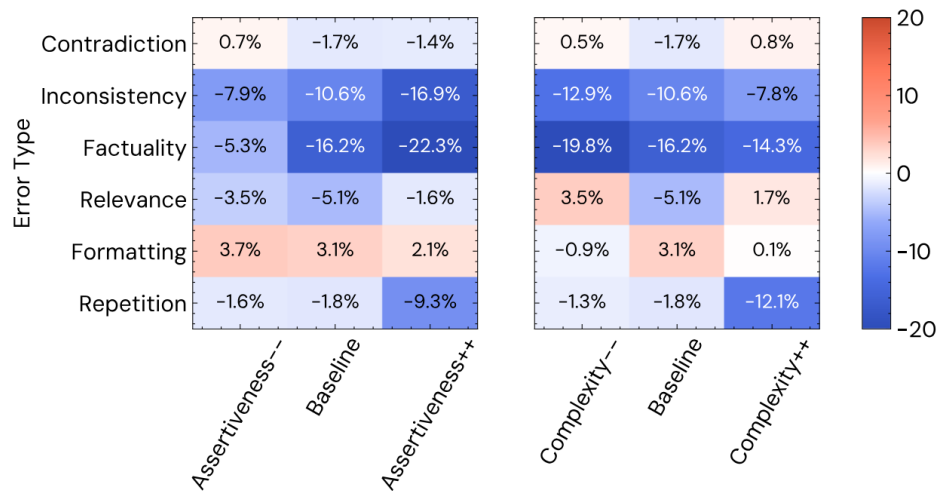


Figure 4: The difference in error rates between crowdsourced annotations and ‘expert’ annotations from the authors, excluding samples that were marked as refusing to respond. Annotators tend to underestimate the rate of inconsistency or factuality errors, and they are less likely to spot these errors in outputs that are assertive.

How do we RLHF?

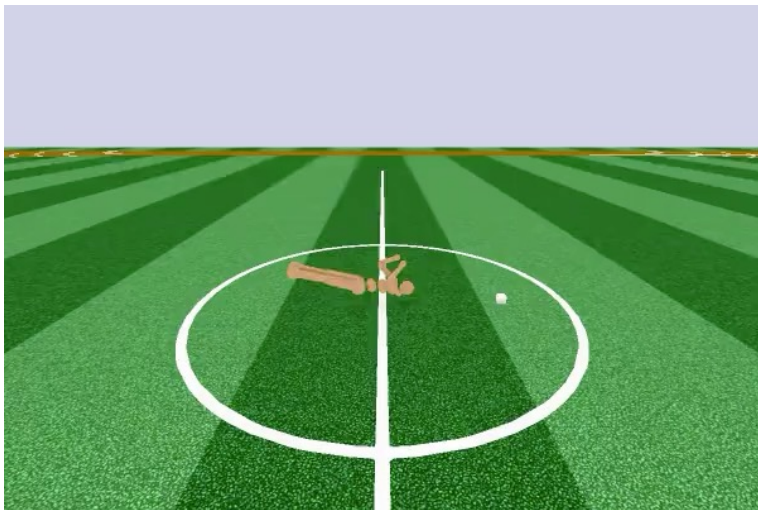
We now have a (high quality) pairwise feedback data collection pipeline?

How do we adapt the model to make use of pairwise feedback?

- **Part 1: PPO** – the original and very finicky approach
- **Part 2: DPO** – the new, very accessible approach

PPO

Policy gradient method for optimizing rewards in actual RL tasks..



From the PPO announcement blog (2017)



OpenAI Five (2019)

PPO in language modeling

From InstructGPT

Reinforcement learning (RL). Once again following Stiennon et al. (2020), we fine-tuned the SFT model on our environment using PPO (Schulman et al., 2017). The environment is a bandit environment which presents a random customer prompt and expects a response to the prompt. Given the prompt and response, it produces a reward determined by the reward model and ends the episode. In addition, we add a per-token KL penalty from the SFT model at each token to mitigate over-optimization of the reward model. The value function is initialized from the RM. We call these models “PPO.”

We also experiment with mixing the pretraining gradients into the PPO gradients, in order to fix the performance regressions on public NLP datasets. We call these models “PPO-ptx.” We maximize the following combined objective function in RL training:

$$\begin{aligned} \text{objective}(\phi) = & E_{(x,y) \sim D_{\pi_{\phi}^{\text{RL}}}} [r_{\theta}(x,y) - \beta \log(\pi_{\phi}^{\text{RL}}(y|x)/\pi^{\text{SFT}}(y|x))] + \\ & \gamma E_{x \sim D_{\text{pretrain}}} [\log(\pi_{\phi}^{\text{RL}}(x))] \end{aligned} \quad (2)$$

where π_{ϕ}^{RL} is the learned RL policy, π^{SFT} is the supervised trained model, and D_{pretrain} is the pretraining distribution. The KL reward coefficient, β , and the pretraining loss coefficient, γ , control the strength of the KL penalty and pretraining gradients respectively. For “PPO” models, γ is set to 0. Unless otherwise specified, in this paper InstructGPT refers to the PPO-ptx models.

..this is very innocuous looking

More details and background from Stiennon

From "Learning to summarize from human feedback"

Reward models. To train our reward models, we start from a supervised baseline, as described above, then add a randomly initialized linear head that outputs a scalar value. We train this model to predict which summary $y \in \{y_0, y_1\}$ is better as judged by a human, given a post x . If the summary preferred by the human is y_i , we can write the RM loss as:

$$\text{loss}(r_\theta) = -E_{(x, y_0, y_1, i) \sim D} [\log(\sigma(r_\theta(x, y_i) - r_\theta(x, y_{1-i})))]$$

where $r_\theta(x, y)$ is the scalar output of the reward model for post x and summary y with parameters θ , and D is the dataset of human judgments. At the end of training, we normalize the reward model outputs such that the reference summaries from our dataset achieve a mean score of 0.

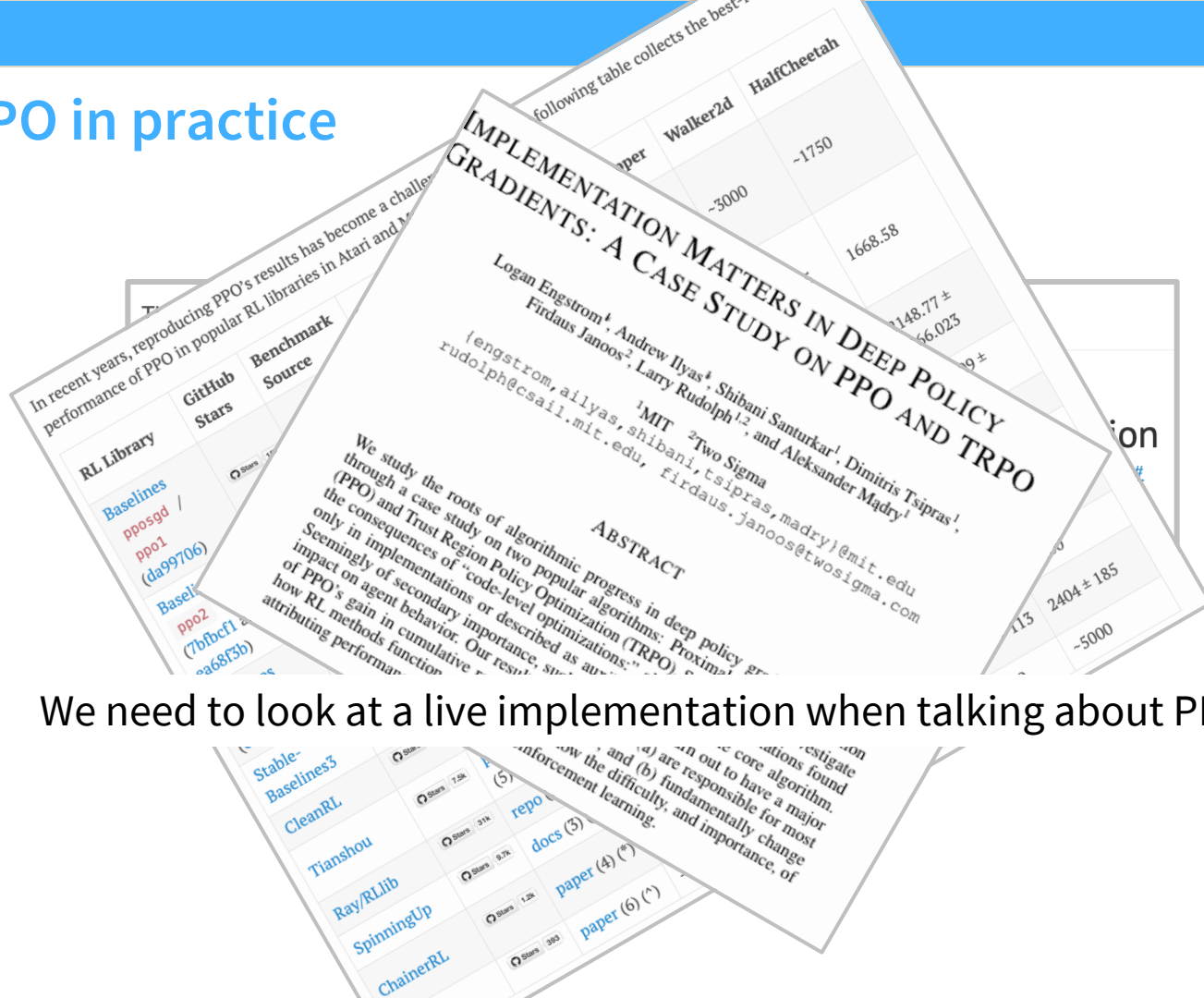
Human feedback policies. We want to use the reward model trained above to train a policy that generates higher-quality outputs as judged by humans. We primarily do this using reinforcement learning, by treating the output of the reward model as a reward for the entire summary that we maximize with the PPO algorithm [58], where each time step is a BPE token.⁸ We initialize our policy to be the model fine-tuned on Reddit TL;DR. Importantly, we include a term in the reward that penalizes the KL divergence between the learned RL policy π_ϕ^{RL} with parameters ϕ and this original supervised model π^{SFT} , as previously done in [25]. The full reward R can be written as:

$$R(x, y) = r_\theta(x, y) - \beta \log[\pi_\phi^{\text{RL}}(y|x) / \pi^{\text{SFT}}(y|x)]$$

This KL term serves two purposes. First, it acts as an entropy bonus, encouraging the policy to explore and deterring it from collapsing to a single mode. Second, it ensures the policy doesn't learn to produce outputs that are too different from those that the reward model has seen during training.

For the PPO value function, we use a Transformer with completely separate parameters from the policy. This prevents updates to the value function from partially destroying the pretrained policy early in training (see ablation in Appendix G.1). We initialize the value function to the parameters of the reward model. In our experiments, the reward model, policy, and value function are the same size.

PPO in practice



PPO – at a conceptual level

A brief (and high level) intro to the various ideas in PPO..

Attempt 1: Policy gradients (variances are too high)

$$\nabla_{\theta} E_{p_{\theta}}[R(z)] = E_{p_{\theta}}[R(z) \nabla_{\theta} \log p_{\theta}(z)]$$

Attempt 2: TRPO (Linearize the problem around the current policy)

$$\begin{aligned} & \underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t \right] \\ & \text{subject to} \quad \hat{\mathbb{E}}_t [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)]] \leq \delta. \end{aligned}$$

Attempt 3: PPO (Clip the ratios at some eps)

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad \text{clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right)$$

PPO – at a conceptual level

The objective is basically all there is to it at a conceptual level

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go $\hat{R}_t$.
- 5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

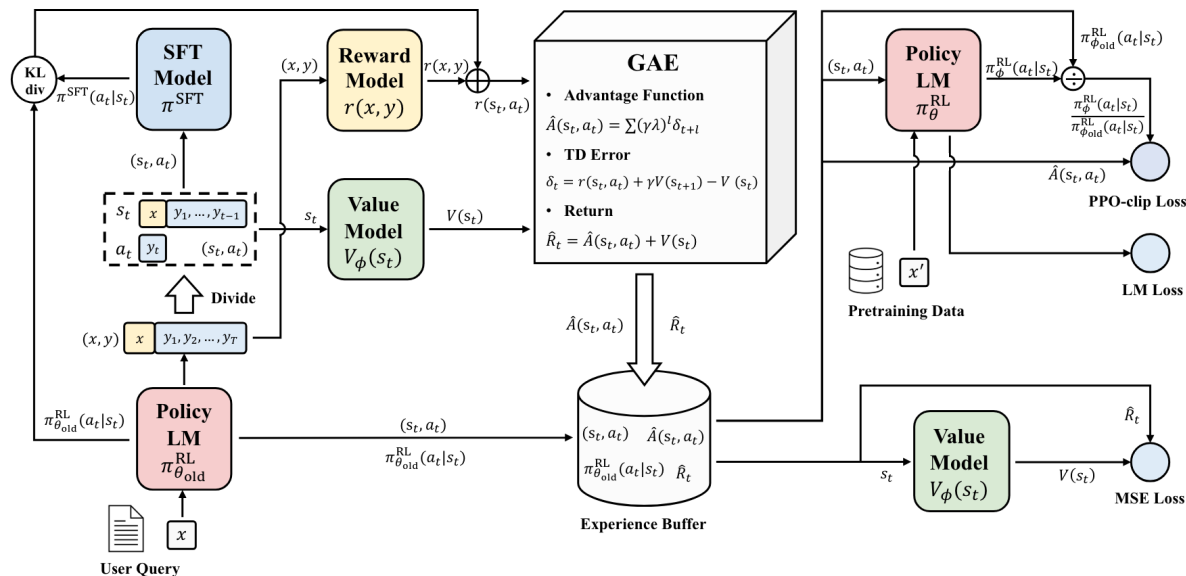
$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

PPO – idealization (?) for language models

In the language model setting..



Pretty similar to the RL formulation. Actions operate over tokens, big dense reward at the very end operating on full sequence

PPO – let's look at an implementation

Implementation – AlpacaFarm's PPO

Method	Simulated Win-rate (%)	Human Win-rate (%)
GPT-4* [†]	79.0 ± 1.4	69.8 ± 1.6
ChatGPT* [†]	61.4 ± 1.7	52.9 ± 1.7
PPO	46.8 ± 1.8	55.1 ± 1.7
DPO	46.8 ± 1.7	-
Best-of-1024	45.0 ± 1.7	50.7 ± 1.8
Expert Iteration	41.9 ± 1.7	45.7 ± 1.7
SFT 52k	39.2 ± 1.7	40.7 ± 1.7

Used by several others to reasonable success

SALMON: SELF-ALIGNMENT WITH INSTRUCTABLE REWARD MODELS

Zhiqing Sun^{1,2*} Yikang Shen¹ Hongxin Zhang³ Qinzhong Zhou³ Zhenfang Chen¹

David Cox¹ Yiming Yang² Chuang Gan^{1,3}

¹MIT-IBM Watson AI Lab, IBM Research

²Language Technologies Institute, CMU

³UMass Amherst

ALIGNING LARGE MULTIMODAL MODELS WITH FACTUALLY AUGMENTED RLHF

Zhiqing Sun^{*♣}, Sheng Shen^{*♣}, Shengcao Cao^{*◇}
Haotian Liu[▽], Chunyuan Li[‡], Yikang Shen[△], Chuang Gan^{†▽△}, Liang-Yan Gui^{†◇}
Yu-Xiong Wang^{†◇}, Yiming Yang^{†♣}, Kurt Keutzer^{†♣}, Trevor Darrell^{†♣}

[♣]UC Berkeley, [♣]CMU, [◇]UIUC, [▽]UW-Madison, [▽]UMass Amherst

[‡]Microsoft Research, [△]MIT-IBM Watson AI Lab

PPO in practice

PPO outer loop. Invoke an inner loop to optimize the loss over some rollouts.

```
def step_with_rollouts(self, rollouts):  
    """Based on fixed rollouts, run PPO for multiple epochs."""  
    assert isinstance(self.optimizer, AcceleratedOptimizer), (  
        "`optimizer` must be pushed through `accelerator.prepare`. "  
        "Otherwise the `accelerator.accumulate` context manager won't correctly disable `zero_grad` or `step`."  
    )  
    rollouts_dataloader = self.get_rollouts_dataloader(rollouts=rollouts)  
    stats_list = []  
    for epoch_idx in range(self.args.noptepochs):  
        for batch_idx, rollouts_batch in tqdm.tqdm(  
            enumerate(rollouts_dataloader, 1), disable=not self.accelerator.is_main_process, desc="gradstep"  
        ):  
            with self.accelerator.accumulate(self.policy):  
                ppo_loss, stats_for_this_step = self.compute_loss(rollouts_batch)  
                self.accelerator.backward(ppo_loss)  
                if self.accelerator.sync_gradients:  
                    # Gradient norm almost blows up at some point, but stabilizes eventually, even w/o clipping.  
                    if self.args.max_grad_norm is not None:  
                        self.accelerator.clip_grad_norm_(self.policy.parameters(), self.args.max_grad_norm)  
                        stats_for_this_step["loss/grad_norm"] = self._compute_grad_norm()  
                        stats_list.append(stats_for_this_step)  
                self.optimizer.step()  
                self.optimizer.zero_grad(set_to_none=True)  
    return common.merge_dict(stats_list, torch.stack) # list of dict -> dict: str -> 1-D tensor
```

PPO in practice – loss computation

AlpacaFarm code – loss computation. Pretty standard stuff.

```
class PPOTrainer(rl_trainer.RLTrainer):  
  
    def compute_loss(self, rollouts: Dict[str, Tensor]) -> Tuple[Tensor, Dict]:  
        values, old_logprob, returns, advantages, queries, query_attn_masks, responses = common.prepare_inputs(  
            common.unpack_dict(  
                rollouts,  
                keys=("values", "logprobs", "returns", "advantages", "queries", "query_attn_masks", "responses"),  
            ),  
            device=self.accelerator.device,  
        )  
  
        outputs = self.policy(queries, query_attn_masks, responses, temperature=self.args.temperature)  
  
        vpred = outputs["values"]  
        vpredclipped = torch.clamp(  
            vpred,  
            min=values - self.args.cliprange_value,  
            max=values + self.args.cliprange_value,  
        )  
  
        vf_losses1 = (vpred - returns) ** 2.0  
        vf_losses2 = (vpredclipped - returns) ** 2.0  
        vf_loss = 0.5 * torch.maximum(vf_losses1, vf_losses2).mean()  
        vf_clipfrac = (vf_losses2 > vf_losses1).to(torch.get_default_dtype()).mean()  
  
        logprob = outputs["logprobs"]  
        ratio = torch.exp(logprob - old_logprob)  
        # When current policy is close to the old policy, the KL component of this advantage is approximately correct.  
        pg_losses = -advantages * ratio  
        pg_losses2 = -advantages * torch.clamp(ratio, min=1.0 - self.args.cliprange, max=1.0 + self.args.cliprange)  
        pg_loss = torch.maximum(pg_losses, pg_losses2).mean()  
        pg_clipfrac = (pg_losses2 > pg_losses).to(torch.get_default_dtype()).mean() # noqa  
  
        loss = pg_loss + self.args.vf_coef * vf_loss
```

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \text{clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_k}}(s, a) \right)$$

Cliprange=0.2

PPO in practice – rollouts.

```
102 def rollout(self, queries_data) -> Dict[str, Tensor]:
103     """Rollout trajectories with policy.
104
105     Args:
106         queries_data: Sequence of batches or DataLoader.
107         Each batch is a dict with keys 'queries' and 'query_attn_masks'.
108
109     Returns:
110         Dictionary with keys
111         'queries', 'query_attn_masks', 'responses',
112         'logprobs', 'ref_logprobs', 'values',
113         'rewards', 'non_score_rewards', 'shaped_rewards'.
114     """
115     # Give up dropout throughout.
116     self.policy.eval()
117     self.make_fsdp_happy()
118     # 'keep_fp32_wrapper' retains the autocast wrapper of model.forward created by accelerate:
119     # recall one sets mixed precision options with accelerator.
120     # The precise value of this arg doesn't matter here, since we use the unwrapped model only for respond.
121     # Generally, try to use the wrapped model as much as you can, since it's got the autocast/cast-back wrappers.
122     unwrapped_policy = self.accelerator.unwrap_model(self.policy, keep_fp32_wrapper=True)
123
124     self.ref_policy.eval()
125     self.reward_model.eval()
126
127     rollouts = []
128     for batch_idx, batch in tqdm.tqdm(
129         enumerate(queries_data),
130         disable=not self.accelerator.is_main_process,
131         desc="rollout",
132     ):
133         # Sample rollouts.
134         queries, query_attn_masks = common.unpack_dict(
135             common.prepare_inputs(batch, device=self.accelerator.device),
136             keys=("queries", "query_attn_masks"),
137         )
138         respond_outputs = unwrapped_policy.respond(queries, query_attn_masks, temperature=self.args.temperature)
139         (responses,) = common.unpack_dict(respond_outputs, ("responses",))
140
141         # Evaluate logprobs of the samples.
142         rollouts_batch = {"queries": queries, "query_attn_masks": query_attn_masks, "responses": responses}
143         policy_outputs = self.policy(**rollouts_batch, temperature=self.args.temperature)
144         ref_policy_outputs = self.ref_policy(**rollouts_batch, temperature=self.args.temperature)
145         policy_outputs = common.unpack_dict(
146             policy_outputs, keys=("logprobs", "values", "entropies"), return_type=dict
147         )
148         ref_policy_outputs = common.unpack_dict(
149             ref_policy_outputs, keys=("logprobs", "entropies"), return_type=dict
150         )
151         rollouts_batch.update(policy_outputs)
152         rollouts_batch.update({"ref_" + key: value for key, value in ref_policy_outputs.items()})
153
```

```
154     # Evaluate reward of the samples.
155     text_queries, text_responses = tuple(
156         self.tokenizer.batch_decode(tensor, skip_special_tokens=True, clean_up_tokenization_spaces=True)
157         for tensor in (queries, responses)
158     )
159     del queries, responses # Prevent mistakes.
160
161     # We retokenize, since policy and reward model might not have the same tokenizer.
162     # TODO(lxuechen): Avoid retokenization when policy and reward tokenizer are the same.
163     text_sequences = [q + r for q, r in utils.zip(text_queries, text_responses)]
164     # TODO(lxuechen): This response retokenization has issues with OPT, since the tokenizer always prepend
165     # <bos_token>. But the issue is local to post_reward, which isn't an issue if we don't penalize.
166     sequences, responses = tuple(
167         self.tokenizer(text, return_tensors="pt", padding=True, truncation=True)
168         for text in (text_sequences, text_responses)
169     )
170     sequences, responses = common.prepare_inputs((sequences, responses), device=self.accelerator.device)
171
172     reward_outputs = self.reward_model(**sequences)
173     reward_outputs = self.post_reward(reward_outputs, responses.input_ids)
174     rollouts_batch.update(reward_outputs)
175
176     # Shape reward with KL penalty.
177     shape_reward_outputs = self._shape_reward(
178         rewards=rollouts_batch["rewards"],
179         responses=rollouts_batch["responses"],
180         logprobs=rollouts_batch["logprobs"],
181         ref_logprobs=rollouts_batch["ref_logprobs"],
182     )
183     rollouts_batch.update(shape_reward_outputs)
184
185     rollouts_batch_cpu = {key: value.cpu() for key, value in rollouts_batch.items()}
186     rollouts.append(rollouts_batch_cpu)
187
188     # Items in dict need to be of same shape.
189     rollouts = common.merge_dict(rollouts, merge_fn=torch.cat)
190     # Estimating advantages outside the loop gives more samples for reward normalization.
191     advantages = self._estimate_advantage(
192         rewards=rollouts["shaped_rewards"].to(self.accelerator.device),
193         values=rollouts["values"].to(self.accelerator.device),
194     )
195     advantages = {key: value.cpu() for key, value in advantages.items()}
196     return (**rollouts, **advantages)
```

PPO in practice – reward shaping

High level – add per-token KL penalty, last-token full reward

In practice? Clip KL for sequences where new policy logp < reference logp

```
67  def _shape_reward(  
68      self, rewards: Tensor, responses: Tensor, logprobs: Tensor, ref_logprobs: Tensor  
69  ) -> Dict[str, Tensor]:  
70      # For some reason, line below doesn't work.  
71      # kl = (logits.softmax(dim=-1) * (logits.log_softmax(dim=-1) - ref_logits.log_softmax(dim=-1))).sum(dim=-1)  
72      kl = torch.clamp(logprobs - ref_logprobs, min=0.0)  
73      non_score_rewards = -self.kl_ctl.value * kl  
74      shaped_rewards = non_score_rewards.clone()  
75      # This introduces a small index off by one bug if pad_token_id == eos_token_id.  
76      terminal_positions = (responses != self.tokenizer.pad_token_id).sum(dim=1) - 1  
77      shaped_rewards[list(range(rewards.size(0))), terminal_positions] += rewards  
78      return dict(shaped_rewards=shaped_rewards, non_score_rewards=non_score_rewards, kl=kl)
```

Helps with stability? If we blow up our model, this prevents kl from diverging

PPO in practice – generalized advantage estimate

Instead of reward, we use advantages

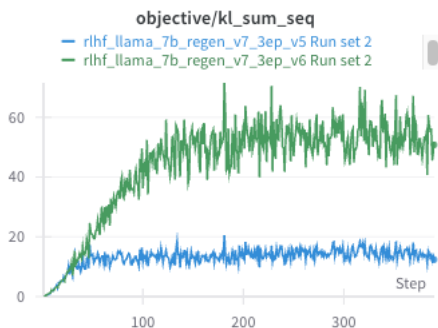
$$\hat{A}_t^{\text{GAE}(\gamma, \lambda)} := \sum_{l=0}^{\infty} (\gamma \lambda)^l \delta_{t+l}^V \quad \text{where} \quad \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t),$$

```
80  def _estimate_advantage(self, rewards: Tensor, values: Tensor) -> Dict[str, Tensor]:
81      """Generalized advantage estimation.
82
83      Reference:
84          https://arxiv.org/abs/1506.02438
85      """
86      if self.args.whiten_rewards:
87          rewards = torch_ops.whiten(rewards, shift_mean=False)
88      lastgaelam = 0
89      advantages_reversed = []
90      gen_length = self.args.response_len
91      for t in reversed(range(gen_length)):
92          nextvalues = values[:, t + 1] if t < gen_length - 1 else 0.0
93          delta = rewards[:, t] + self.args.gamma * nextvalues - values[:, t]
94          lastgaelam = delta + self.args.gamma * self.args.lam * lastgaelam
95          advantages_reversed.append(lastgaelam)
96      advantages = torch.stack(advantages_reversed[::-1], dim=1)
97      returns = advantages + values
98      advantages = torch_ops.whiten(advantages, shift_mean=True)
99      return dict(returns=returns, advantages=advantages)
```

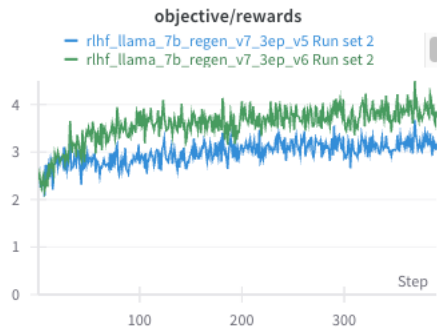
Funny detail – this is a bandit problem and gamma=1 works – this is the reward-to-go vs the value

What do you expect to see in PPO?

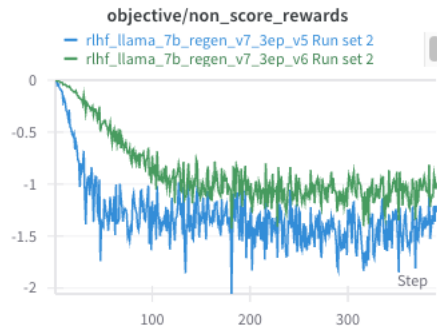
Increasing overall rewards



Incl. reward model



Negative KL rewards



This is a bandit setting, you expect reasonable training curves

Can we get rid of PPO?

Question for many: can we avoid doing any 'RL' ? (i.e. on-policy RL algorithms)

Some reasonable stuff people thought about

- Train the model with a control token – (SFT on the pairs, prepend [GOOD] to chosen, [BAD] to not chosen)
- Train the model on only preferred output
- Train a reward model, get LM outputs, train on the preferred output
- Train a reward model, get 1024 LM outputs, take the best one.

Can we get rid of PPO?

Most of these baselines turn out to just work worse than PPO on instruction-tuning

Method	Simulated win-rate (%)	Human win-rate (%)
GPT-4	79.0 ± 1.4	69.8 ± 1.6
ChatGPT	61.4 ± 1.7	52.9 ± 1.7
PPO	46.8 ± 1.8	55.1 ± 1.7
Best-of- n	45.0 ± 1.7	50.7 ± 1.8
Expert Iteration	41.9 ± 1.7	45.7 ± 1.7
SFT 52k (Alpaca 7B)	39.2 ± 1.7	40.7 ± 1.7
SFT 10k	36.7 ± 1.7	44.3 ± 1.7
Binary FeedME	36.6 ± 1.7	37.9 ± 1.7
Quark	35.6 ± 1.7	-
Binary Reward Conditioning	32.4 ± 1.6	-
Davinci001	24.4 ± 1.5	32.5 ± 1.6
LLaMA 7B	11.3 ± 1.1	6.5 ± 0.9

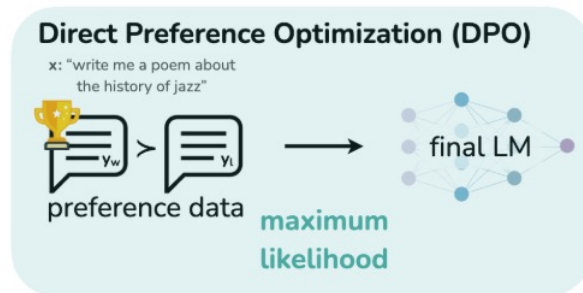
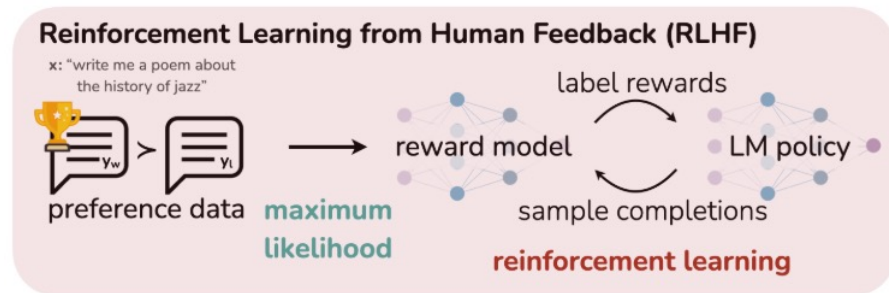
DPO – RLHF without tears?

Try to simplify PPO by..

- Getting rid of the reward model
- Getting rid of any on-policy stuff (rollouts, outer loops etc)

Instead

- Take gradient steps on log-loss of good stuff
- Take negative gradient steps on bad stuff (appropriately weighted).



DPO – derivation from the RLHF formula

Our goal is to optimize

$$\max_{\pi_{\theta}} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_{\theta}(y|x)} [r_{\phi}(x, y)] - \beta \mathbb{D}_{\text{KL}} [\pi_{\theta}(y | x) || \pi_{\text{ref}}(y | x)]$$

Assume that the policy π is the set of all policies (nonparametric assumption).

The maximizer is then,

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

Solve for the ‘implied reward’

$$r(x, y) = \beta \log \frac{\pi_r(y | x)}{\pi_{\text{ref}}(y | x)} + \beta \log Z(x).$$

DPO derivation 2

We can now optimize the implied reward as a reward model via the Stiennon objective

$$\text{loss}(r_\theta) = -E_{(x, y_0, y_1, i) \sim D} [\log(\sigma(r_\theta(x, y_i) - r_\theta(x, y_{1-i})))] \quad \text{and} \quad r(x, y) = \beta \log \frac{\pi_r(y | x)}{\pi_{\text{ref}}(y | x)} + \beta \log Z(x).$$

This gives the DPO objective,

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

The key steps?

1. Make a nonparametric assumption (links π_θ and r in closed form)
2. Parametrize reward r via the policy
3. Optimize the reward using supervised losses (which in turn, optimizes the policy)

DPO updates and components

In some sense, reduces to “pos gradient on good, neg gradient on bad”

What does the DPO update do? For a mechanistic understanding of DPO, it is useful to analyze the gradient of the loss function $\mathcal{L}_{\text{DPO}}$. The gradient with respect to the parameters θ can be written as:

$$\begin{aligned} \nabla_{\theta} \mathcal{L}_{\text{DPO}}(\pi_{\theta}; \pi_{\text{ref}}) = \\ - \beta \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\underbrace{\sigma(\hat{r}_{\theta}(x, y_l) - \hat{r}_{\theta}(x, y_w))}_{\text{higher weight when reward estimate is wrong}} \left[\underbrace{\nabla_{\theta} \log \pi(y_w | x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_{\theta} \log \pi(y_l | x)}_{\text{decrease likelihood of } y_l} \right] \right], \end{aligned}$$

(Scaled by ‘prediction error’ of the implied reward model)

Results – controlled comparison in AF

Compared to our previous PPO implementation? Same perf (on sim) with no pain!

Method	Simulated Win-rate (%)	Human Win-rate (%)
GPT-4* [†]	79.0 ± 1.4	69.8 ± 1.6
ChatGPT* [†]	61.4 ± 1.7	52.9 ± 1.7
PPO	46.8 ± 1.8	55.1 ± 1.7
DPO	46.8 ± 1.7	-
Best-of-1024	45.0 ± 1.7	50.7 ± 1.8
Expert Iteration	41.9 ± 1.7	45.7 ± 1.7
SFT 52k	39.2 ± 1.7	40.7 ± 1.7
SFT 10k	36.7 ± 1.7	44.3 ± 1.7
Binary FeedME	36.6 ± 1.7	37.9 ± 1.7
Quark	35.6 ± 1.7	-
Binary Reward Conditioning	32.4 ± 1.6	-
Davinci001*	24.4 ± 1.5	32.5 ± 1.6
LLaMA 7B*	11.3 ± 1.1	6.5 ± 0.9

DPO works

T ▲	Model ▲	Average 📊 ▲	ARC ▲	HellaSwag ▲	MMLU ▲	TruthfulQA ▲	Winogrande ▲	GSM8K ▲
■	udkai/Turdus	74.66	73.38	88.56	64.52	67.11	86.66	67.7
■	fblgit/UNA-TheBeagle-7b-v1	73.87	73.04	88	63.48	69.85	82.16	66.72
■	argilla/distilabeled-Marcoro14-7B-slerp	73.63	70.73	87.47	65.22	65.1	82.08	71.19
■	mlabonne/NeuralMarcoro14-7B	73.57	71.42	87.59	64.84	65.64	81.22	70.74
◆	abideen/NexoNimbus-7B	73.5	70.82	87.86	64.69	62.43	84.85	70.36
■	Neuronovo/neuronovo-7B-v0.2	73.44	73.04	88.32	65.15	71.02	80.66	62.47
■	argilla/distilabeled-Marcoro14-7B-slerp-full	73.4	70.65	87.55	65.33	64.21	82	70.66
■	CultriX/MistralTrix-v1	73.39	72.27	88.33	65.24	70.73	80.98	62.77
■	ryandt/MusingCaterpillar	73.33	72.53	88.34	65.26	70.93	80.66	62.24
■	Neuronovo/neuronovo-7B-v0.3	73.29	72.7	88.26	65.1	71.35	80.9	61.41
■	CultriX/MistralTrixTest	73.17	72.53	88.4	65.22	70.77	81.37	60.73
◆	samir-fama/SamirGPT-v1	73.11	69.54	87.04	65.3	63.37	81.69	71.72
◆	SanjiWatsuki/Lelantos-DPO-7B	73.09	71.08	87.22	64	67.77	80.03	68.46

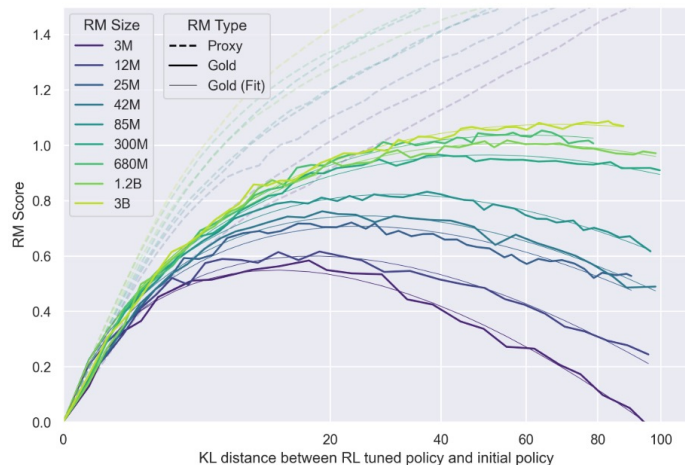
Handwritten notes in red:

- DPO** (written above the first 10 models)
- DPO (& UNA)** (written above fblgit/UNA-TheBeagle-7b-v1)
- DPO** (written above argilla/distilabeled-Marcoro14-7B-slerp)
- DPO** (written above mlabonne/NeuralMarcoro14-7B)
- Merge (of DPO models)** (written above abideen/NexoNimbus-7B)
- DPO** (written above Neuronovo/neuronovo-7B-v0.2)
- DPO** (written above argilla/distilabeled-Marcoro14-7B-slerp-full)
- DPO** (written above CultriX/MistralTrix-v1)
- DPO** (written above ryandt/MusingCaterpillar)
- DPO** (written above Neuronovo/neuronovo-7B-v0.3)
- No info but prob DPO, given Merge (incl. DPO)** (written above CultriX/MistralTrixTest)
- DPO** (written above samir-fama/SamirGPT-v1)
- DPO** (written above SanjiWatsuki/Lelantos-DPO-7B)

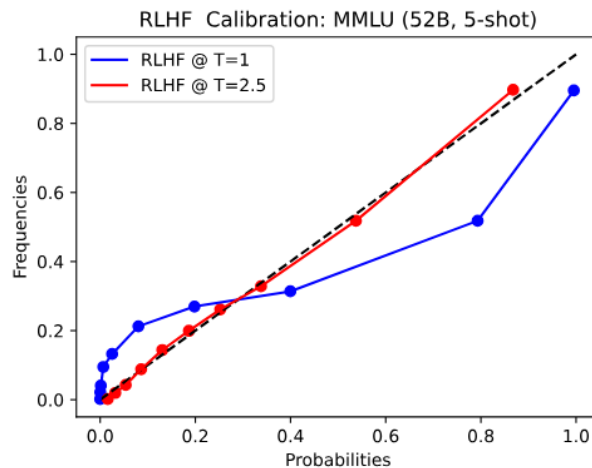
From Chris Manning – Most ‘top open-source’ RLHF models are DPO’d

Things to watch out for in RLHF

Overoptimization / overfitting on the reward

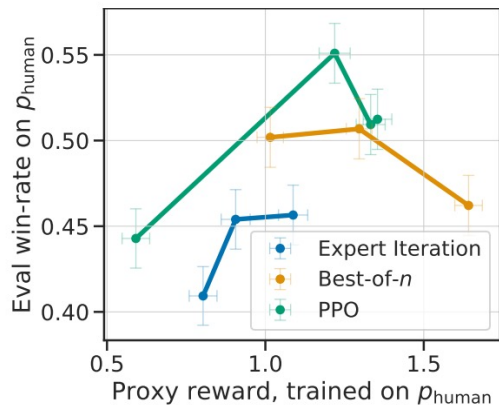


Mode collapse / entropy

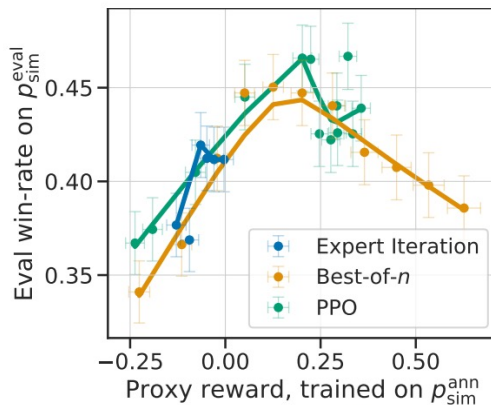


Things to watch out for - Overoptimization

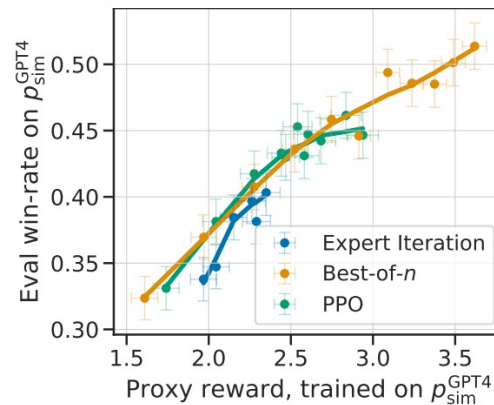
Across many different RLHF-style optimizers. Optimizing for reward overfits past a point



(a) Human preferences ■



(b) AlpacaFarm ◆

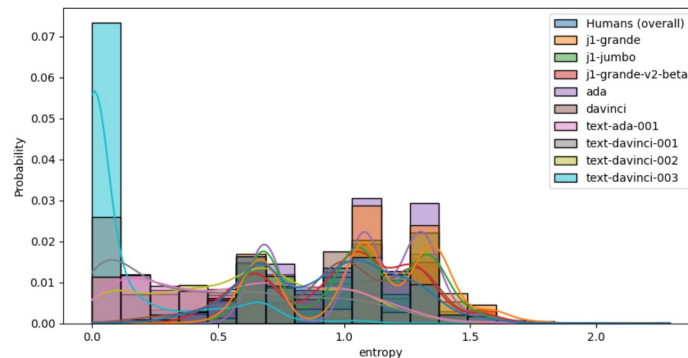
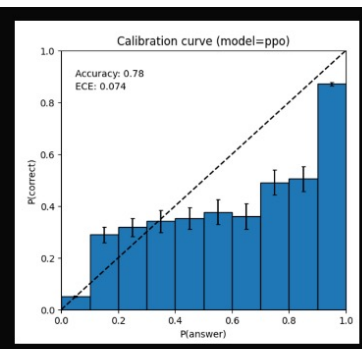
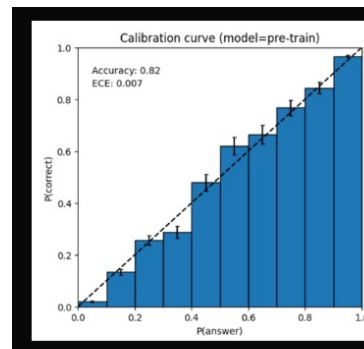
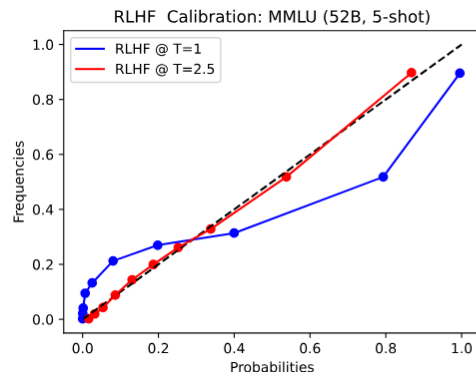


(c) Single-prompt GPT-4 ●

Holds true for human pref (left), noisy LM pref (mid) but not noiseless LM pref (right)

Things to watch out for - mode collapse

RLHF makes models no longer ‘probabilistic models’ – no calibration by default



Recap of the lecture

RLHF recap:

1. RLHF data collection is (also) hard! Many confounding factors
2. RLHF algorithms are a bit more complex than SFT – esp. PPO
3. Be mindful of the impact of (over) optimizing for rewards