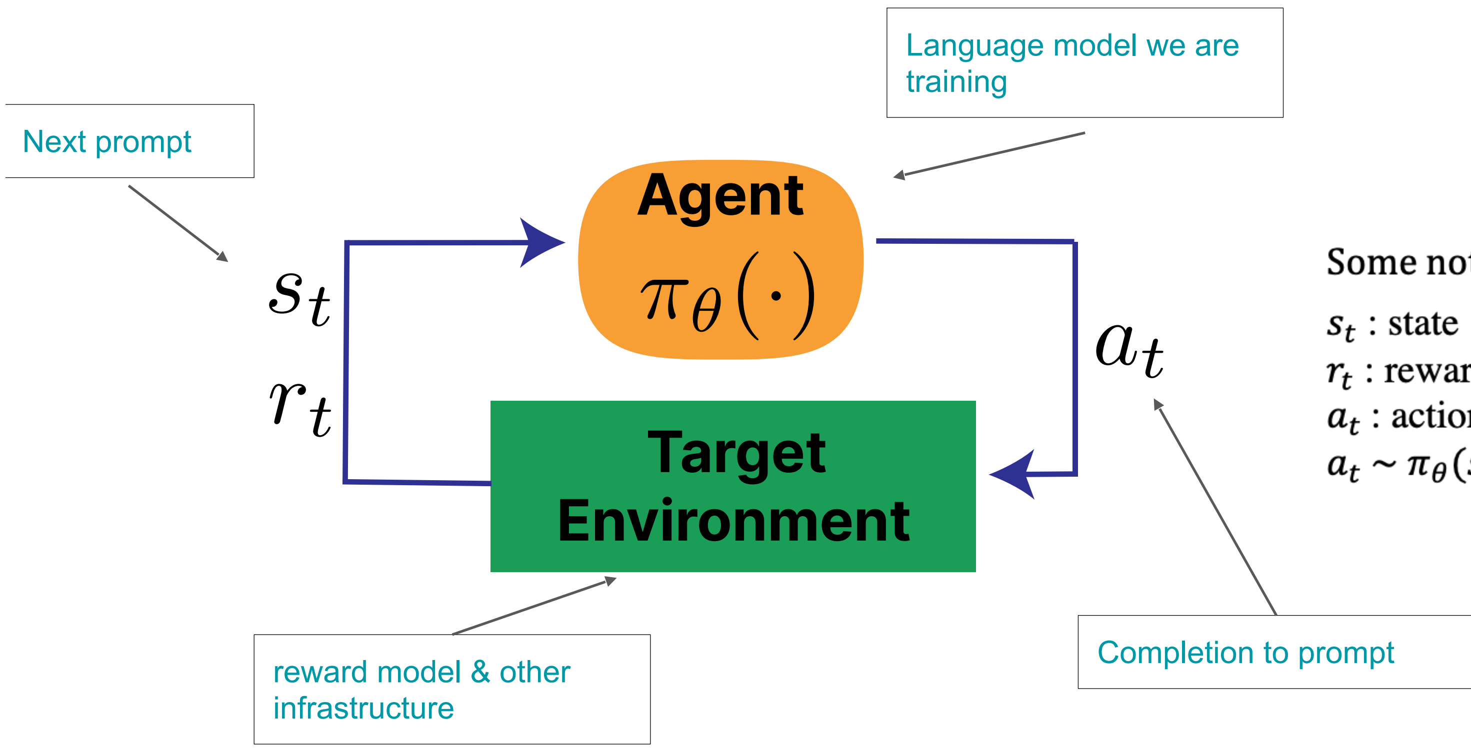
Reinforcement Learning for LLMs / RLHF

COMP579, Lecture 24

Overview of RLHF

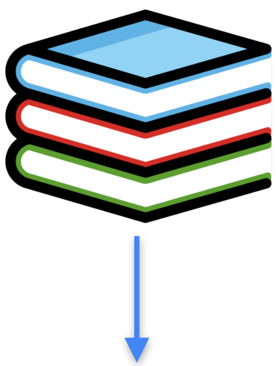


Some notation:

s_t : state
r_t : reward
a_t : action
$a_t \sim \pi_\theta(s_t)$: policy

RLHF early attempts

Summarization



“Three pigs defend themselves from a mean wolf”

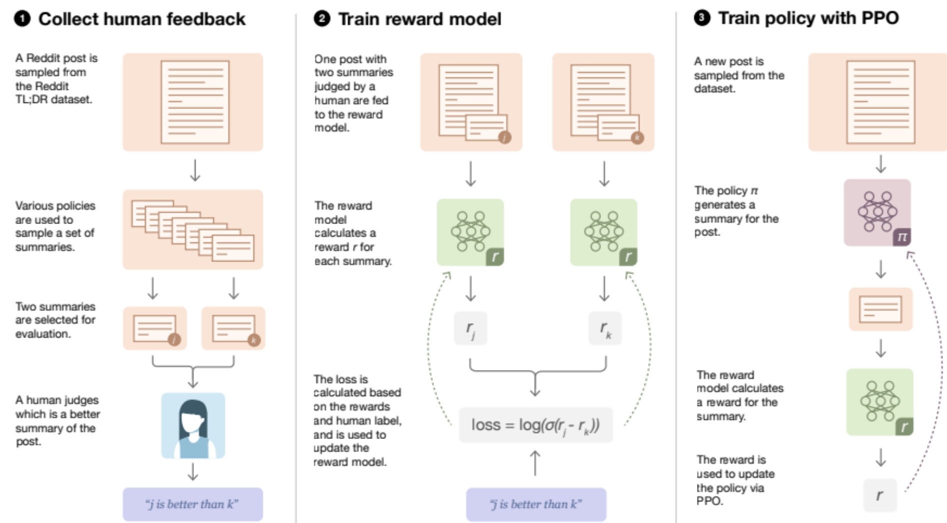
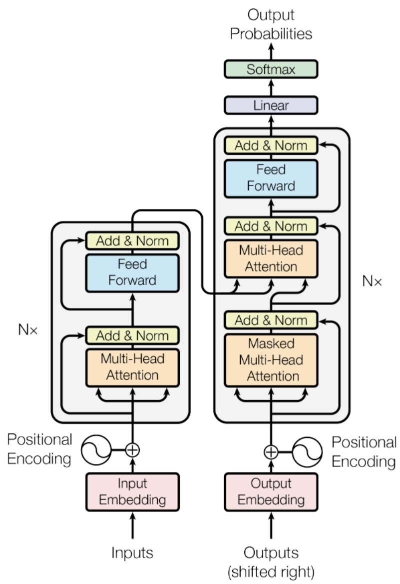


Figure 2: Diagram of our human feedback, reward model training, and policy training procedure.

Stiennon, Nisan, et al. "Learning to summarize with human feedback." *2020.*

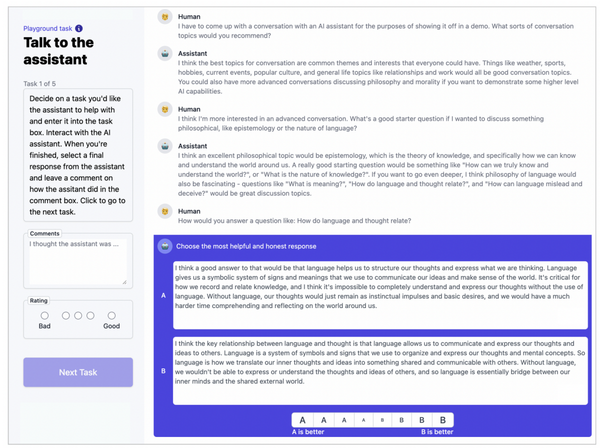
RLHF training phases

base model (instruction, helpful, chatty etc.)

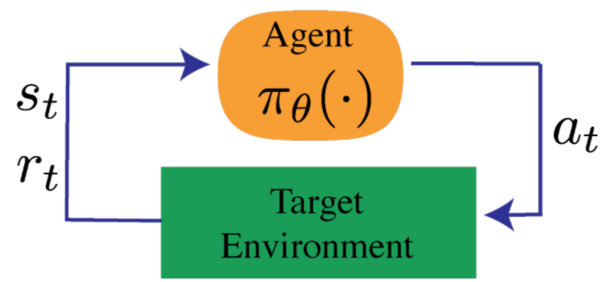


Vaswani et al. 2017

preference collection & training



RL optimization



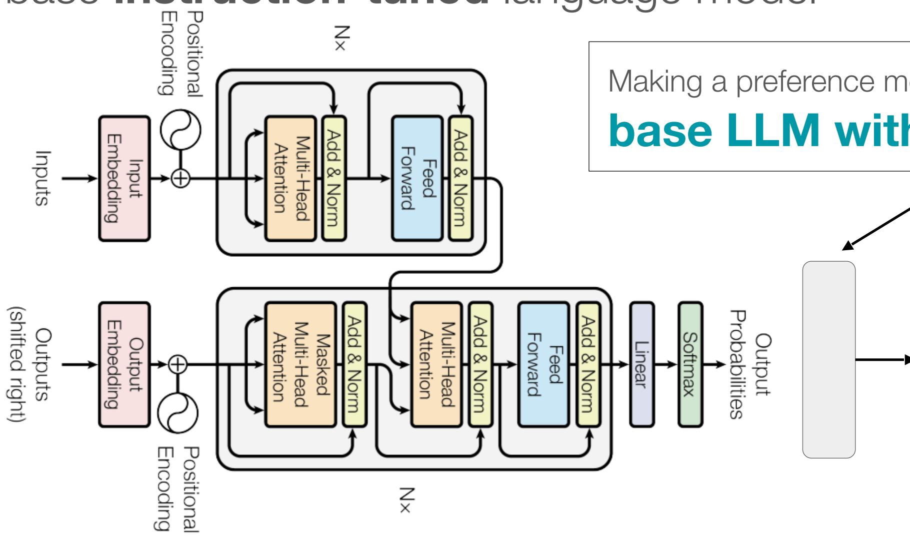
Model structure

starting point: a base **instruction-tuned** language model

Making a preference model:
base LLM with new final layer

input:

prompt+completion



The Transformer - Vaswani et al. 2017

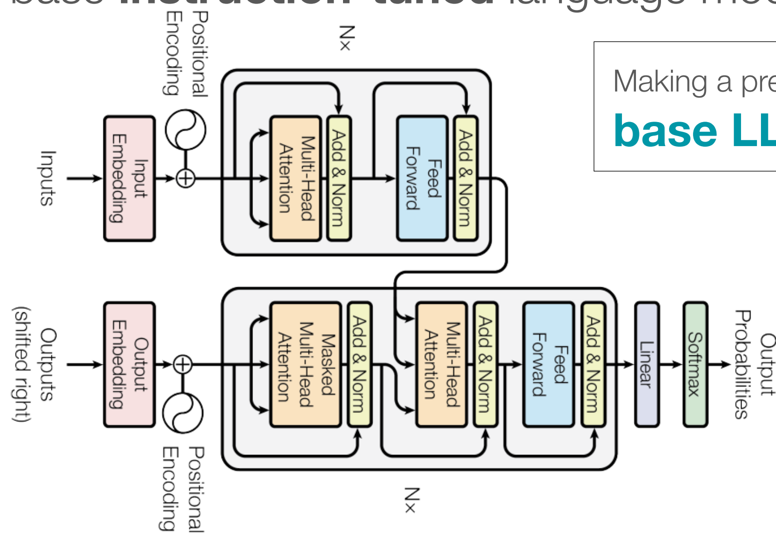
output:
scalar rewards

Model structure

starting point: a base **instruction-tuned** language model

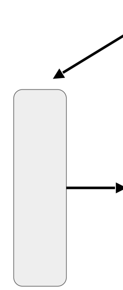
input:

prompt+completion



The Transformer - Vaswani et al. 2017

Making a preference model:
base LLM with new final layer



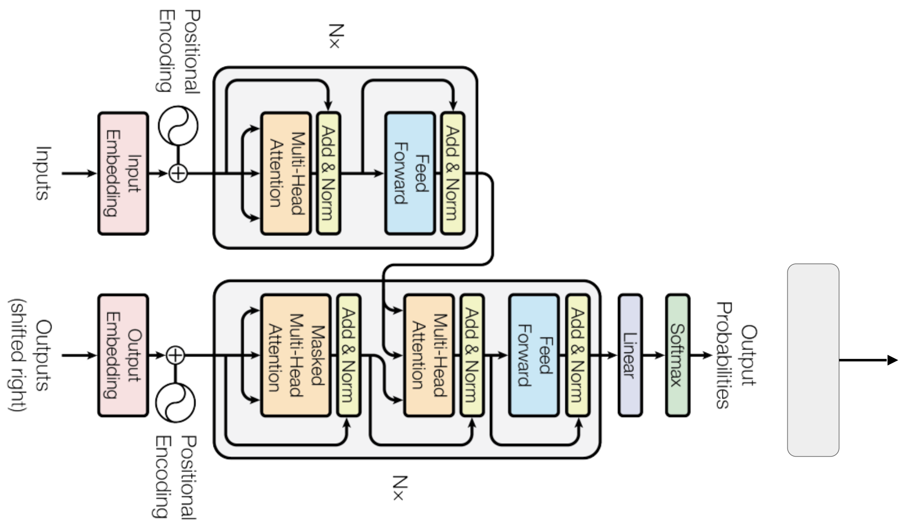
output:
scalar rewards

Model training

input pair:

selected prompt +completion

rejected prompt +completion



The Transformer - Vaswani et al. 2017

output:
scalar rewards

loss: increase difference of predicted reward

Recall: Bradely-Terry reward model

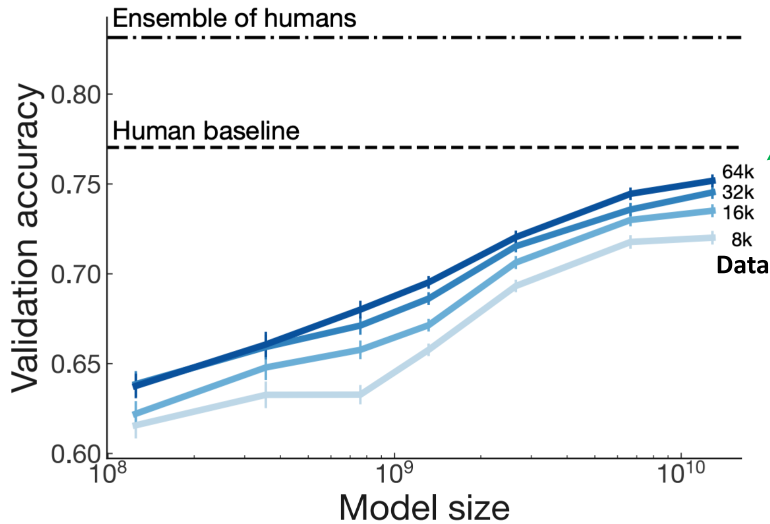
- Collect data from human raters (pairs of y_w, y_l responses to a prompt x)
- Optimize the expected value of:

$$-\log(\sigma(r_\theta(x, y_w) - r_\theta(x, y_l)))$$

  wrt reward parameter vector θ
- Cf. Ouyang et al, InstructGPT
- Corresponds to maximum likelihood fitting of binomial preference function if reward is linear over the variables

Evaluating the reward model

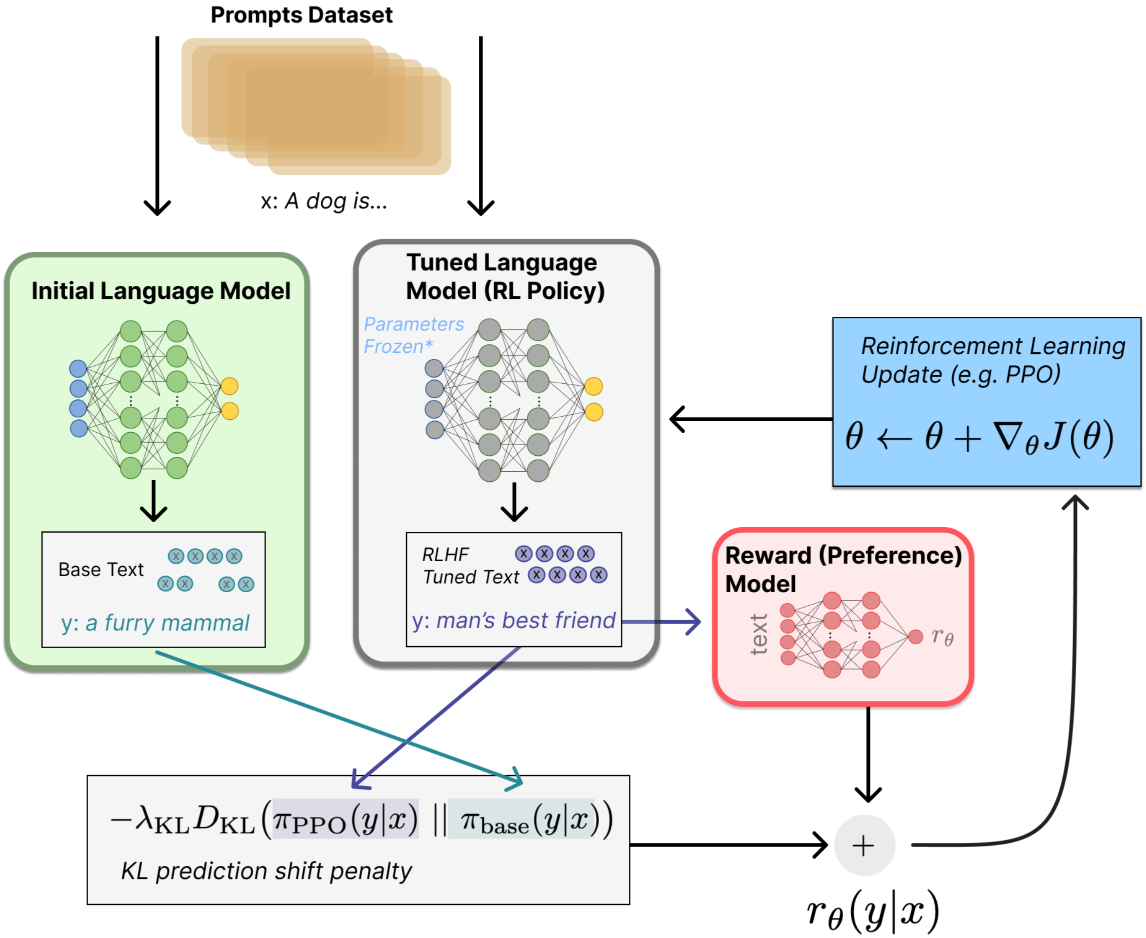
Evaluate RM on predicting outcome of held-out human judgments



Large enough RM trained on enough data approaching single human perf

[Stiennon et al., 2020]

RLHF finetuning



RLHF details

Finally, we have everything we need:

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

Now to do RLHF:

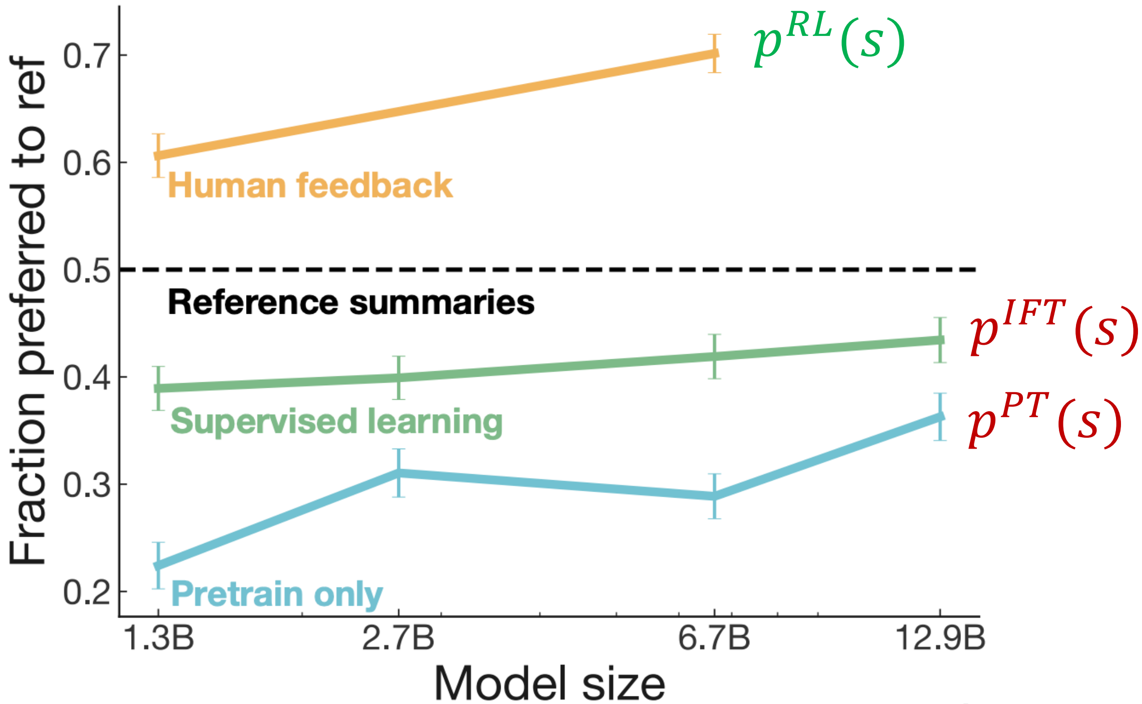
- Initialize a copy of the model $p_\theta^{RL}(s)$, with parameters θ we would like to optimize
- Optimize the following reward with RL:

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

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

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

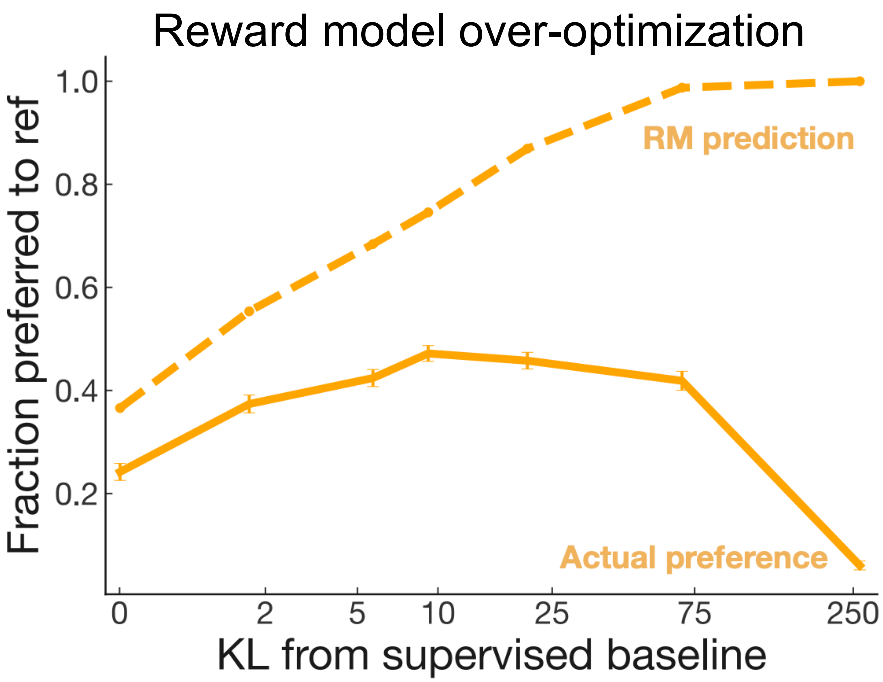
RLHF results



[Stiennon et al., 2020]

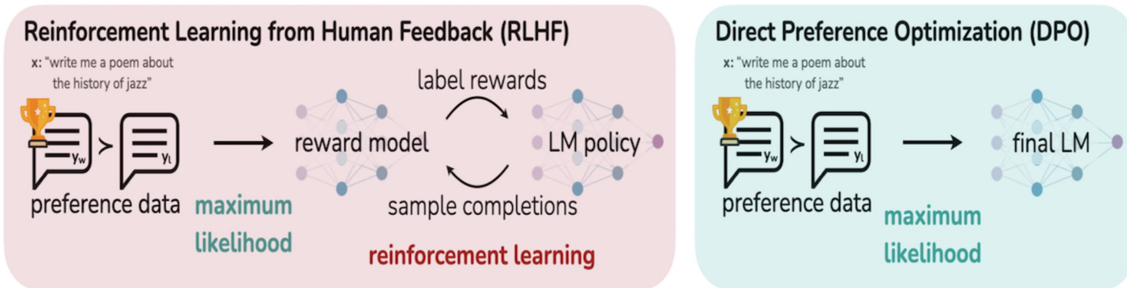
Problem: reward hacking

- Human preferences are unreliable!
  - ”Reward hacking” is a common problem in RL
  - Chatbots are rewarded to produce responses that *seem* authoritative and helpful, *regardless of truth*
  - This can result in making up facts + hallucinations
- **Models** of human preferences are *even more* unreliable!



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

Recall: Direct Preference Optimization



$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) =$$

$$-\beta \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Bigg[\underbrace{\sigma(\hat{r}_\theta(x,y_l) - \hat{r}_\theta(x,y_w))}_{\text{higher weight when reward estimate is wrong}}\Bigg[\underbrace{\nabla_\theta \log \pi(y_w \mid x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta \log \pi(y_l \mid x)}_{\text{decrease likelihood of } y_l}\Bigg]\Bigg],$$

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

- You can replace the complex RL part with a very simple weighted MLE objective
- Other variants (KTO, IPO) now emerging too

[Rafailov+ 2023]

Learning with non-transitive preferences: NashLLM

- Objective:find a policy π^* which is preferred over any other policy

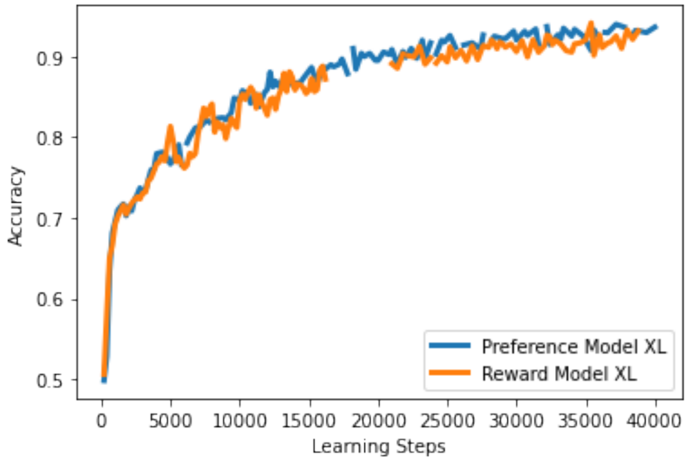
$$\pi^* = \arg\max_{\pi} \min_{\pi'} \mathbb{P}(\pi' \preceq \pi)$$

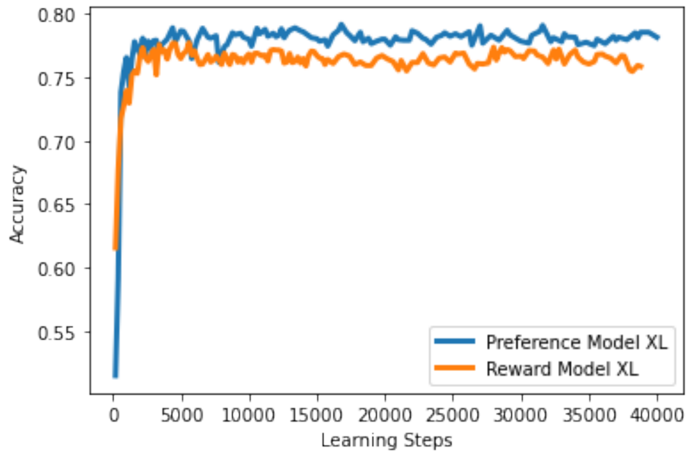
- Think of this as a game: one player picks π the other picks π'
- When both players use π^* this is a *Nash equilibrium* for the game
- For this game an equilibrium exists (even if eg preferences are not transitive)
- Cf. Munos et al, 2024 (https://arxiv.org/pdf/2312.00886.pdf)

NashLLM-style algorithms

- Fit a *two-argument preference function* by supervised learning
- Decide what is the *set of opponent policies*
- Ideally, the max player should play against a mixture of past policies
- *Optimize* using eg online mirror descent, convex-concave optimization...
- A lot of algorithmic variations to explore!

NashLLM results

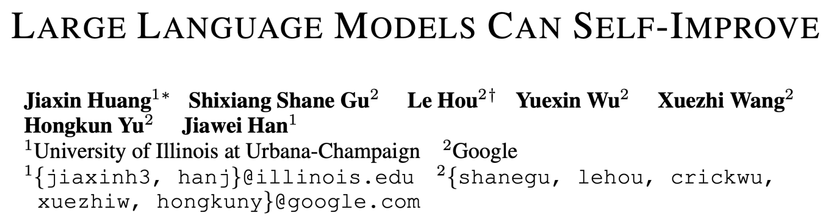




Using preferences instead of rewards leads to less overfitting

Open directions

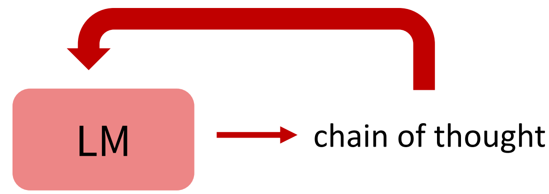
- RLHF is still a very underexplored and fast-moving area!
- RLHF gets you further than instruction finetuning, but is (still!) data expensive.
- Recent work aims to alleviate such data requirements:
  - RL from **AI feedback** [Bai et al., 2022]
  - Finetuning LMs on their own outputs [Huang et al., 2022; Zelikman et al., 2022]
- However, there are still many limitations of large LMs (size, hallucination) that may not be solvable with RLHF!



LARGE LANGUAGE MODELS CAN SELF-IMPROVE

Jiaxin Huang[1*] Shixiang Shane Gu[2] Le Hou[2†] Yuexin Wu[2] Xuezhi Wang[2]
Hongkun Yu[2] Jiawei Han[1]
[1]University of Illinois at Urbana-Champaign [2]Google
[1]{jiaxinh3, hanj}@illinois.edu [2]{shanegu, lehou, crickwu, xuezhiw, hongkuny}@google.com

[Huang et al., 2022]



Self-Taught Reasoner (STaR)
[Zelikman et al., 2022]