# Instruction tuning / RLHF

- Goal: align LLMs with human prefs
  - make their outputs less harmful/toxic
  - increase relevance of outputs
- Two main methods:
  - supervised fine-tuning
  - reinforcement learning

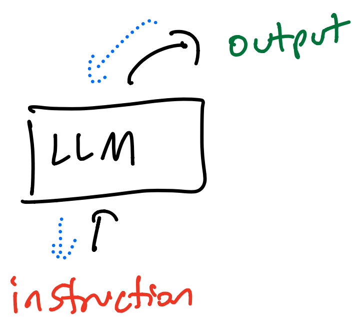
---

## instruction tuning:

1. collect a dataset of instructions on what task to solve, and outputs of that task for one or more examples

Please answer the following question and provide a detailed justification.
What was the average of the CS685 S23 midterm?

I can't answer that question b/c the midterm occurs on April 12 and it is March 27.

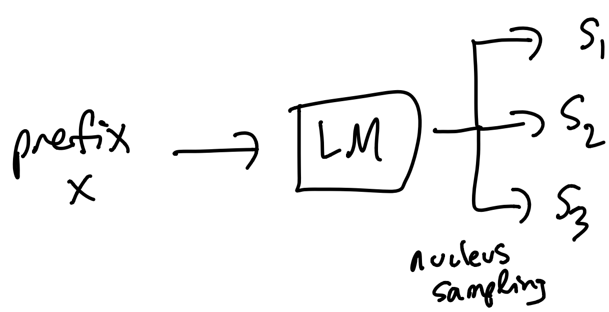



fine-tune LM to produce desired output given instruction

- unlike what we've seen w/ "pretrain ⇒ finetune", instruction tuning finetunes on many diff. tasks at once
- instruction tuning improves generalization on tasks that are not seen during finetuning
- Limitations:
    - getting data is expensive, esp. for very complex tasks
    - Some tasks don't have just one single acceptable output
    - does not directly involve human prefs.

---

RLHF: reinforcement learning from human feedback




humans:
10/10
3/10
6/10

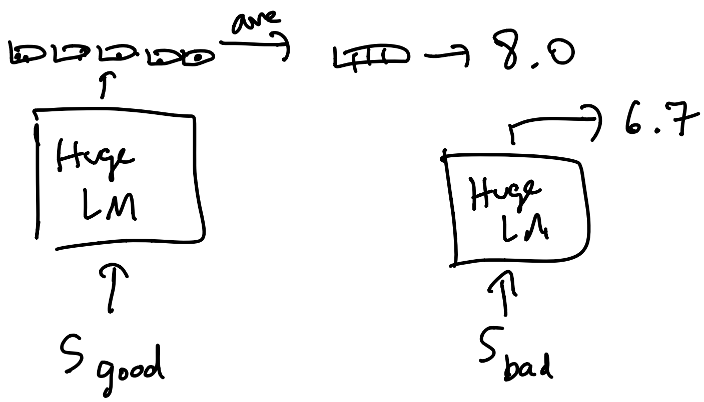
instead,
$s_1 > s_3 > s_2$
rankings/pref. judgments

- extremely expensive to obtain human feedback
- instead, we collect as many human judgments as we can, and then train a reward model to predict the human preferences
    - input: a prefix $x$, a sample $s$
    - output: a scalar score, represents "overall quality" of the sample

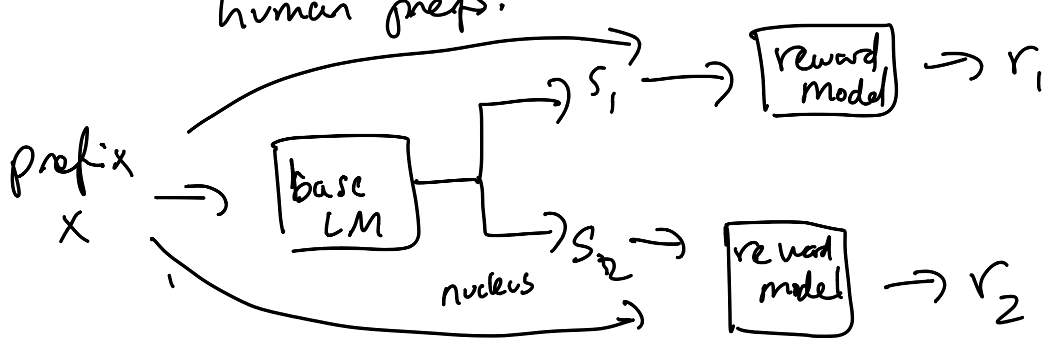
- assume we have two samples $s_{good}$, $s_{bad}$

$$L_{RM} = \log \sigma\left(R(s_{good}) - R(s_{BAD})\right)$$

- intuitively, the good sample's reward should be higher than that of the bad sample




- we can now use our reward model to obtain a score $R(s)$ of any sample $s$ generated from a prefix without needing a human labeler.
   - reward model is trained to mimic human prefs.



- how do we use our reward model to better align LMs to human prefs?

1. Overgeneration + reranking ("best-of-n")
   - ↳ generate $n$ samples, score each w/ reward model, pick the one w/ highest reward
   - ↳ no further training required
2. just fine-tune the LM to maximize $p(s_{good} | x)$
   - ↳ issue: what if $s_{good}$ is not the only acceptable high-reward sample
     - what if $s_{good}$ itself is bad
3. use reinforcement learning to increase $p(s_{good} | x)$ by a small amount, decrease $p(s_{bad} | x)$ by a small amount, where these amounts are functions of the rewards $R(s_{good})$, $R(s_{Bad})$

## RLHF:

- we observe a reward only after generating a full (multi-token) sample via a decoding algo
- goal: maximize $p(s_{good}|x)$, minimize $p(s_{Bad}|x)$ subject to the rewards

RL Loss: $f\left(R(s), p(s|x)\right)$

↳ REINFORCE (Williams 1992)

↳ PPO (Schulman 2016)

↳ used in ChatGPT / GPT4, etc

- important not to deviate too much from the base LM, to prevent reward hacking
- add another loss at the token level that approximates KL divergence between the current model $P_{RLHF}$ and the original $P_{Base}$:

for a given word $w_i$

$$\log \frac{P_{RLHF}(w_i | w_{1 \ldots i-1}, x)}{P_{BASE}(w_i | w_{1 \ldots i-1}, x)}$$

- making it work in practice: