After DPO (Direct Preference Optimization)

① Definition (have learned in lectures before)

Some definitions for "alignment" of models

- **Instruction fine-tuning (IFT):** Training a model to follow use instructions (usually via autoregressive LM loss)
- **Supervised fine-tuning (SFT):** Training a model to learn task-specific capabilities (usually via autoregressive LM loss)
- **Alignment:** General notion of training a model to mirror user desires, any loss function
- **Reinforcement learning from human feedback (RLHF):** Specific technical tool for training ML models from human data
- **Preference fine-tuning:** Using labeled preference data to fine-tune a LM (either with RL, DPO, or another loss function), there's also **learning to rank**

→ not a specific method (why we need it? → we want a model which adhere to human preference)

→ SFT is a specific form of IFT

② RLHF (PPO) (lecture 10)

Review: RLHF objective

π: LLM policy
π_θ: base LLM
x: prompt
y: completion

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

Optimize "reward" *inspired* by human preferences ▲

▲ Constrain the model to not trust the reward too much (preferences are hard to model)

Primary questions:
1. How to implement reward: *r(x,y)*
2. How to optimize reward

Review: Preference (reward) modeling

Can we just use supervised learning on scores?

- Assigning a scalar reward of how good a response is did not work
- Pairwise preferences are easy to collect and worked!

Key idea: Probability ∝ reward

Chosen completion; Prompt; Rejected completion; Score from optimal reward model

$$p^*(y_1 \succ y_2 \mid x) = \frac{\exp\left(r^*(x,y_1)\right)}{\exp\left(r^*(x,y_1)\right) + \exp\left(r^*(x,y_2)\right)}$$

Bradley Terry model:
Estimate probability that a given pairwise preference is true

③ DPO

DPO characteristics

1. Extremely **simple** to implement
2. **Scales nicely** with existing distributed training libraries
3. Trains an implicit reward function (can still be used as a reward model, see RewardBench)

```
import torch.nn.functional as F

def dpo_loss(pi_logps, ref_logps, yw_idxs, yl_idxs, beta):
    """
    pi_logps: policy logprobs, shape (B,)
    ref_logps: reference model logprobs, shape (B,)
    yw_idxs: preferred completion indices in [0, B-1], shape (T,)
    yl_idxs: dispreferred completion indices in [0, B-1], shape (T,)
    beta: temperature controlling strength of KL penalty
    Each pair of (yw_idxs[i], yl_idxs[i]) represents the
    indices of a single preference pair.
    """

    pi_yw_logps, pi_yl_logps = pi_logps[yw_idxs], pi_logps[yl_idxs]
    ref_yw_logps, ref_yl_logps = ref_logps[yw_idxs], ref_logps[yl_idxs]

    pi_logratios = pi_yw_logps - pi_yl_logps
    ref_logratios = ref_yw_logps - ref_yl_logps

    losses = -F.logsigmoid(beta * (pi_logratios - ref_logratios))
    rewards = beta * (pi_logps - ref_logps).detach()
```

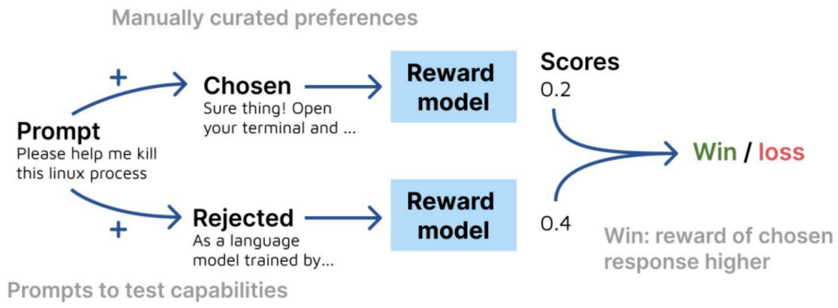
The first 2 points mean we'll see more DPO models than anything else and learn it's limits!

```
return losses, rewards
```

Example code.
Rafailov, Sharma, Mitchell et al. 2023

RewardBench

RewardBench structure



Can we match PPO with "online" DPO ?

What is special about online data?

Online data is **freshly generated from the policy** and/or **recently labelled by a reward model / judge**.

- PPO does both with generation + reward model scoring
- Other methods use different ways for doing this: collect new preference data, re-label existing data, LLM-as-a-judge, reward model ranking

Related question: On- or off-policy data (i.e. that generated from the policy model)

Conclusion

DPO
- advan: easy to use, good scalability
- disadvan: the effect is still slightly inferior to PPO

PPO vs DPO

PPO is RL, DPO is SL.
And PPO can handle distribution offset