OPD

↳ generate a rollout from the student given a prompt x

$$y \sim \pi_S(\cdot \mid x)$$

↳ score the rollout using a teacher model

$$\log \pi_T(y_i \mid x, y_{<i})$$

↳ occurs in parallel, like prefill

↳ compute per-token score

$$r_i = \log \pi_T(y_i \mid x, y_{<i}) - \log \pi_S(y_i \mid x, y_{<i})$$

↳ update the student model via backprop, only if we have access to the full prob. dist from the teacher

OPD update:

↳ at every position of the rollout

compute

$$KL_t = \sum_{w \in V} \pi_s(w \mid x, y_{<t}) \left[r_t[w] \right]$$

↳ backprop thru this

---

x: what is 15% of 80?

y: 15% of 80 = 0.15 × 80 = 12

r_t: ~0 0 0 0 0 0 0 0 <<<0

↳ what if I don't have a teacher model?

but instead I have access to a binary verifier: 1 if correct, 0 if wrong

y: 15% of 80 = 0.15 × 80 = 12

$r(x, y) = 0$ because y is wrong

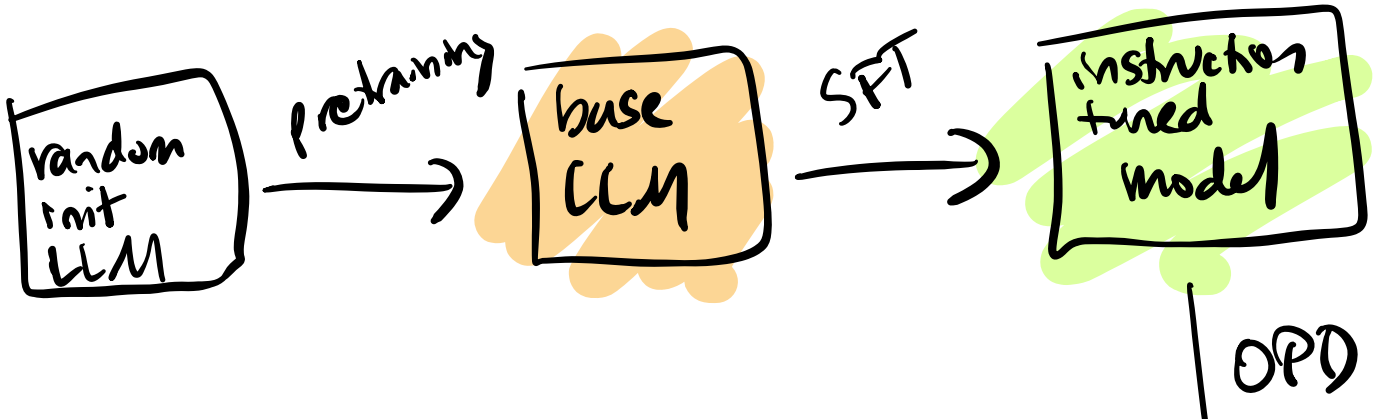
↳ disadvantage: I don't know which token(s) y_t were wrong (credit assignment)

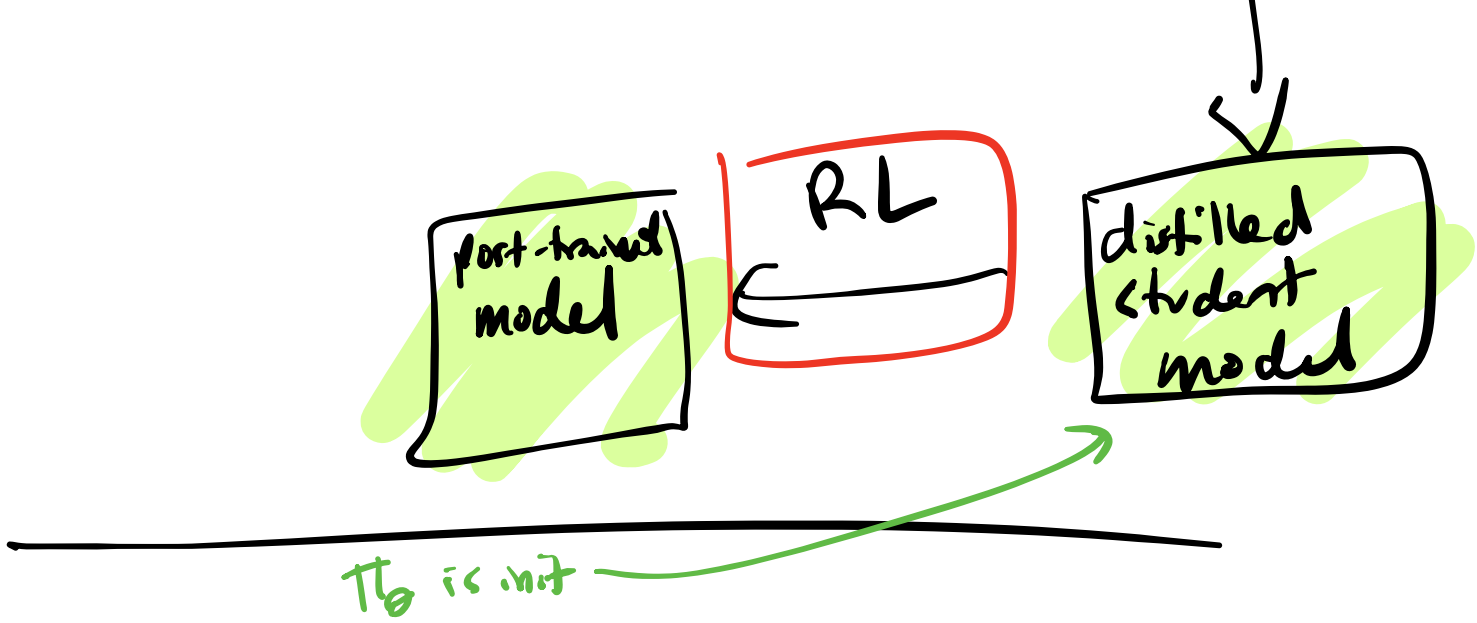
↳ advantage: student perf not capped by a teacher

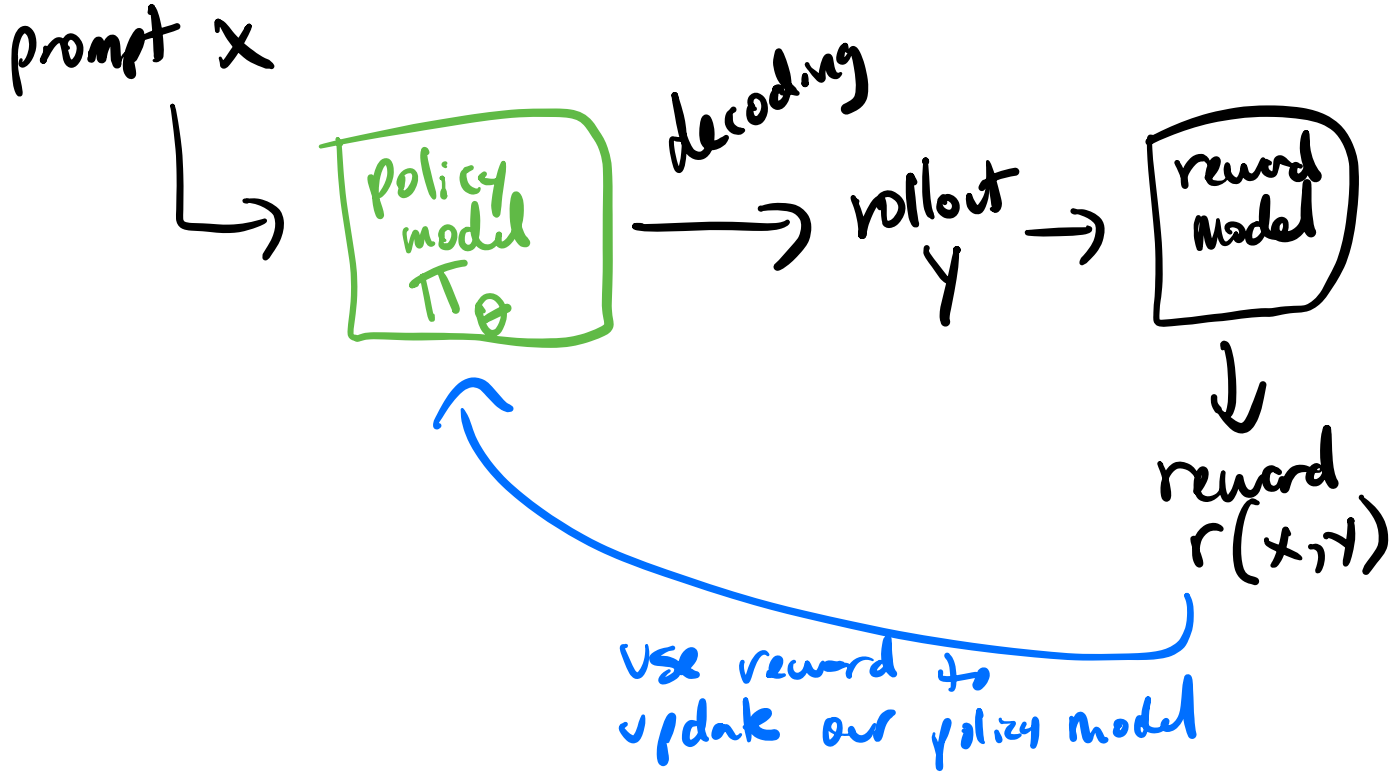
---

goal: improve our model by learning from a single score assoc. w/ a rollout ↳ "reward"

↳ start w/ a pretrained model that was then instruction tuned via SFT, and potentially further improved by OPD







reward (x, y):

↳ "verifiable": is the response correct?

- ↳ pass unit test
- ↳ run w/o errors
- ↳ agree w/ calculator
- ↳ proof verified

↳ for more complex / subjective tasks, we use a reward model

↳ reward model is trained on human preferences

↳ RLHF
↳ human feedback

↳ whether the reward is verifiable or from a reward model, we receive a single score for a rollout, and no gradient of the score

---

intuition: make rollouts with high rewards more probable, make rollouts with low rewards less probable

Concretely

↳ maximize the expected reward of our rollouts

$$J(\theta) = \underset{y \sim \pi_\theta(\cdot|x)}{E}\left[r(x,y)\right]$$

$$= \sum_y \pi_\theta(y|x)\, r(x,y)$$

↳ problem: cannot enumerate all rollouts y

↳ estimate by sampling

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} r(x,y)$$

↳ problem: we need to compute $\frac{dJ}{d\theta}$

↳ unfortunately, we can't backprop thru the sampling process that produces rollouts y

why not?

let's compare to SFT:

policy π_θ → probability dist of next word → compute $L = -\log \pi_\theta(y|x)$

given to us by human/teacher

↳ if we change θ by a little, our loss L correspondingly changes

↳ smooth

RL is different

policy π_θ → probabilities of next token → sample the next token → reward $r(x,y)$

↳ unlike in SFT, y is sampled from the policy

↳ if I change θ a little bit, probabilities change a little, but I might still sample the same output y

↳ changing θ modifies how often I might sample a particular response

---

REINFORCE: Williams, 1992

↳ policy gradient algorithms

goal: $\frac{dJ}{d\theta}$, where $J(\theta) = \sum_y \pi_\theta(y|x) r(x,y)$

↑ assumes a single prompt x

↳ note that only the probs $\pi_\theta(y|x)$ depend on θ, not $r(x,y)$

$$\frac{dJ}{d\theta} = \sum_y r(x,y) \frac{d\pi_\theta(y|x)}{d\theta}$$

↑ no $\pi_\theta(y|x)$ here

↳ just like for $J(\theta)$, we can't enumerate all y_s in the sum

↳ with $J(\theta)$, we just approximated by sampling

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} r(x, y)$$

but we can't easily do the same for $\frac{dJ}{d\theta}$

↳ $\frac{dJ}{d\theta}$ has no $\pi_\theta(y|x)$ term multiplied in the sum

if you sample anyway:

$$\frac{1}{N} \sum_{i=1}^{N} r(x, y_i) \frac{d\pi_\theta(y|x)}{d\theta}$$

$$\approx \sum_{y} \pi_\theta(y|x)\, r(x,y)\, \frac{d\pi_\theta(y|x)}{d\theta}$$

$\pi_\theta(y|x)$ → extra term