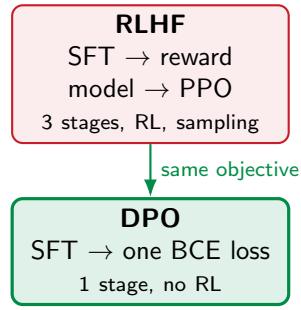
Alignment without the RL

Last time (GRPO) we made RL *cheaper* by deleting the critic. DPO asks a bolder question:

What if we delete the reward model **and** *the RL loop entirely?*

DPO shows the RLHF objective has a **closed-form** solution, and that inverting it turns preference alignment into a **single classification loss** – trained directly on preference pairs, no sampling, no PPO.

The title says it: *"Your language model is secretly a reward model."* Today, why that is literally true.



Outline

Why alignment is hard

Preliminaries: the math of preferences

The DPO derivation

Understanding the update

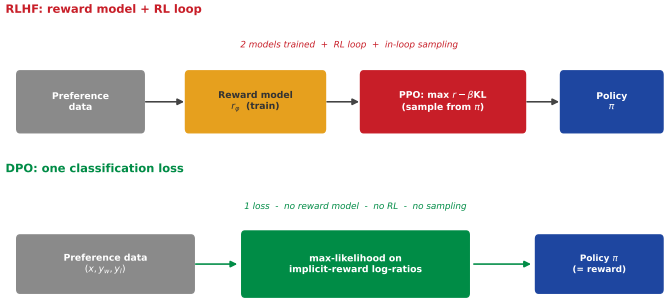
Does it work?

Recap

Why is aligning an LLM hard?

The standard RLHF pipeline has a lot of moving parts.

The three-stage RLHF pipeline



RLHF (top): SFT → fit a reward model on human comparisons → optimize the policy with PPO against that reward, with a KL leash to the SFT model.

The pain: **multiple** large models (reward + policy + critic), **in-loop sampling** from the policy, and **unstable** RL that is sensitive to hyperparameters and needs reward normalization.

Why preferences, and why a KL leash?

Why preferences over demonstrations?

- Humans are better at saying *"A is better than B"* than at writing the gold answer.
- Relative judgments are cheaper and more consistent to collect.

Why keep a KL constraint to π_{ref}?

- Stay where the reward model is *accurate*.
- Preserve diversity; avoid collapsing onto one high-reward answer (reward hacking).

The RLHF objective:

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

Maximize reward, but do not stray far from the reference.

DPO will optimize *exactly* this objective – just without ever running RL.

The math of preferences

Bradley-Terry, and the reward objective we want to solve.

Bradley-Terry: turning preferences into probabilities

A preference dataset is triples (x, y_w, y_l): prompt, **preferred**, **dispreferred**. The Bradley-Terry model says the chance y_1 beats y_2 is a softmax over rewards:

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

A reward model is fit by maximum likelihood – i.e. binary classification on which answer won:

$$\mathcal{L}_R = -\mathbb{E}_{(x, y_w, y_l)}\Big[\log \sigma\big(r_\varphi(x, y_w) - r_\varphi(x, y_l)\big)\Big]$$

Notice: only the **difference** of rewards ever appears. Hold onto this – it is the hinge of the whole derivation.

The derivation

From a KL-constrained objective to a classification loss, in four steps.

The idea, before the algebra

Strip away the machinery and DPO is one sentence: *make the preferred answer more likely than the dispreferred one – relative to where the model started.*

The one move. A model already scores every answer with a probability. Compare that to a **frozen "before" snapshot** π_{ref} (the SFT model):

- If π_θ likes y *more* than π_{ref} did, it has *learned* to prefer y. That gap is already a reward – no separate reward net needed.
- Training just **widens** the gap for winners y_w, **shrinks** it for losers y_l.

Think **grading on a curve against your past self**: not "is this answer good in the absolute," but "did I get *more* inclined toward the human-preferred answer than before training?"

So the loss will simply read: push the winner's $\beta \log \frac{\pi_\theta}{\pi_{\text{ref}}}$ *above* the loser's. The next four steps prove this is **exactly** the RLHF objective – not an approximation.

Step 1: the RLHF objective has a closed-form optimum

For *any* reward r, the policy that maximizes reward $-\ \beta\mathrm{KL}$ is known in closed form:

$$\pi_r(y \mid x) = \frac{1}{Z(x)}\,\pi_{\text{ref}}(y \mid x)\,\exp\Big(\tfrac{1}{\beta} r(x,y)\Big) \qquad Z(x) = \sum_y \pi_{\text{ref}}(y \mid x)\,\exp\Big(\tfrac{1}{\beta} r(x,y)\Big)$$

Why (sketch): rewrite the objective as $\min_\pi \mathbb{E}\big[\mathbb{D}_{\mathrm{KL}}(\pi \,\|\, \pi^*)\big] - \log Z(x)$, where π^* is the boxed distribution. $Z(x)$ does not depend on π, so by Gibbs' inequality the KL is minimized (to 0) exactly when $\pi = \pi^*$. ■

But $Z(x) = \sum_y(\dots)$ sums over **all possible completions** – astronomically many token sequences. Eq. is exact but **intractable**. This is what forces classic RLHF into sampling-based RL.

Step 2: invert – the reward is a log-ratio

Take logs of the boxed optimum and solve for the reward:

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

- The reward that a policy is optimal for equals the β-**scaled log-ratio** of that policy to the reference ...
- ... plus a term $\beta \log Z(x)$ that depends on the *prompt only*, not on y.

This is the "secretly a reward model" identity: *any* language model π_θ **defines** an implicit reward $\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}$.

Step 3: the intractable $Z(x)$ cancels

Plug the reward identity into Bradley-Terry, which only sees reward *differences*:

$$r(x, y_w) - r(x, y_l) = \beta \log \frac{\pi(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} + \underbrace{\beta \log Z(x)}_{\text{prompt only}} - \beta \log \frac{\pi(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} - \underbrace{\beta \log Z(x)}_{\text{prompt only}}$$

The two $\beta \log Z(x)$ terms are **identical** and **cancel**. What was intractable becomes *irrelevant*.

Why it works: (a) both completions share the same prompt x, and (b) the preference model depends only on reward *differences*. Rewards differing by any prompt-only $f(x)$ give the same preferences – DPO just picks the representative with $Z(x) = 1$.

Step 4: the DPO loss

Substitute, and fit π_θ by maximum likelihood on the preference data:

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

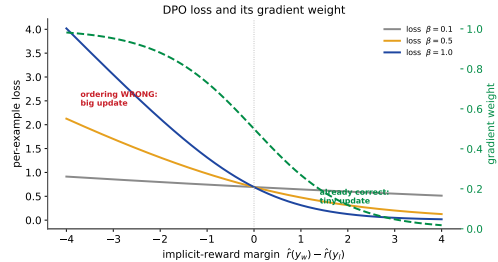
- y_w, y_l – preferred / dispreferred completion.
- π_θ – the policy we train; π_{ref} – frozen SFT model (never updated).
- β – KL strength / sigmoid temperature.

It is just **binary cross-entropy**: "make the preferred completion's implicit reward exceed the dispreferred one's."

The gradient: focus on what you got wrong

$$\nabla_\theta \mathcal{L}_{\text{DPO}} = -\beta\, \mathbb{E}\Big[\underbrace{\sigma\big(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w)\big)}_{\text{weight: large when ranking is WRONG}} \big(\nabla_\theta \log \pi_\theta(y_w \mid x) - \nabla_\theta \log \pi_\theta(y_l \mid x)\big)\Big]$$

- Push **up** y_w, push **down** y_l.
- Scale each pair by how badly the current *implicit* reward orders it.
- Pairs already ranked right barely move; mis-ranked pairs dominate the update.



Drop that weight (naive "raise y_w, lower y_l") and the model **degenerates** into garbage. The dynamic weight is the secret sauce.

The recipe in code

```python
import torch.nn.functional as F

def dpo_loss(pi_logps, ref_logps, yw, yl, beta):
    # log-probs of the whole chosen / rejected sequences
    pi_w,  pi_l  = pi_logps[yw],  pi_logps[yl]
    ref_w, ref_l = ref_logps[yw], ref_logps[yl]

    pi_logratio  = pi_w  - pi_l
    ref_logratio = ref_w - ref_l

    # binary cross-entropy on the implicit-reward margin
    loss = -F.logsigmoid(beta * (pi_logratio -
         ref_logratio))

    # implicit rewards (for logging only)
    rewards = beta * (pi_logps - ref_logps).detach()
    return loss, rewards
```

What you need:

- preference triples (x, y_w, y_l)
- a frozen reference π_{ref} (= SFT model)

Per batch: four log-probs (policy & ref, on y_w & y_l) $\rightarrow$ sigmoid loss.

What you do not need: reward model, critic, policy sampling, RL tuning.

Defaults: $\beta = 0.1$, batch 64, lr 10^{-6}.

DPO vs RLHF, side by side

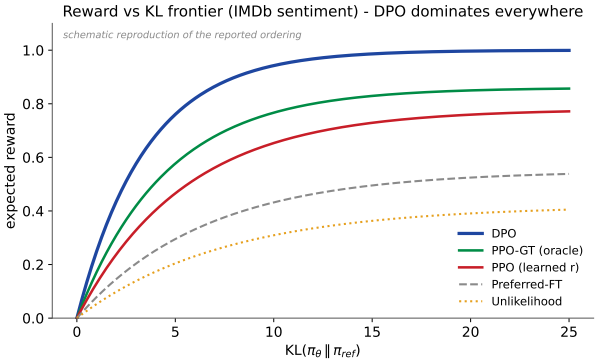
	RLHF (SFT→RM→PPO)	**DPO**
Explicit reward model	yes (separate net)	**no** – implicit $\hat{r} = \beta \log \frac{\pi_\theta}{\pi_{\text{ref}}}$
RL loop / critic	yes	**no**
Sample from policy in training	yes (on-policy)	**no** (offline)
Loss type	policy gradient (unstable)	classification / BCE (stable)
Objective optimized	KL-constrained reward max	**the same**, reparameterized

DPO is not an approximation – it optimizes *exactly* the RLHF objective, just written as a loss over the policy itself.

Does it work?

A computable frontier, win rates, and temperature robustness.

Reward vs KL: DPO dominates the frontier

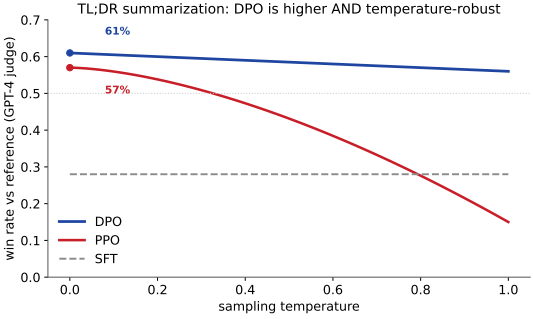


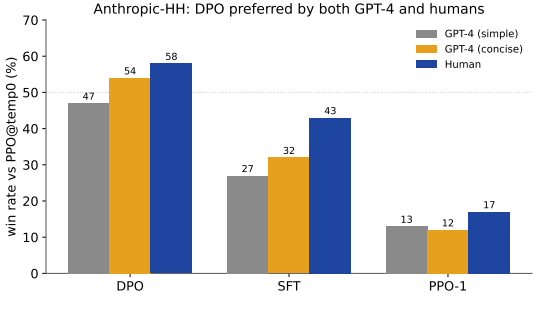
On IMDb the reward is a known sentiment classifier, so the exact reward-vs-KL trade-off is *computable*.

- DPO reaches **higher reward at every KL** budget.
- It even beats **PPO-GT** – PPO given the *ground-truth* reward.

Curves reproduce the reported ordering (schematic shape).

Win rates and temperature robustness





Summarization (left): DPO $\approx 61\%$ win rate vs $\approx 57\%$ for PPO – and DPO stays high as sampling temperature rises, while PPO collapses. **Dialogue (right):** on Anthropic-HH, DPO is preferred over PPO by *both* GPT-4 and human raters. GPT-4 agrees with humans about as often as humans agree with each other.

Recap and caveats

- DPO optimizes the **exact RLHF objective** with a single classification loss.
- Key identity: a language model *is* a reward model, $\hat{r}_\theta = \beta \log(\pi_\theta / \pi_{\text{ref}})$.
- The intractable partition function $Z(x)$ **cancels** because preferences use only reward differences within a prompt.
- No reward model, no critic, no policy sampling, minimal tuning – and it matches or beats PPO.
- The reference π_{ref} is **essential** (it defines the implicit reward); and the dynamic gradient weight is what prevents collapse.
- Caveat: DPO is **offline** – open questions remain on exploration and out-of-distribution behavior vs on-policy RL.

Next: *LoRA* – once we know *how* to fine-tune (SFT, DPO, GRPO), how do we do it **cheaply**, without a full copy of a 175B model per task?