Direct Preference Optimization

v0.4 · June 2026

Matthew Willetts, with assistance from Codex and Claude

A derivation of DPO, the loss, its semantics, and the family of fixes that came after.

1. The problem DPO solves

Classical RLHF needs four moving parts:

1. SFT model (policy initialisation).
2. Reward model trained on preference data.
3. Online rollouts from the policy.
4. PPO-style RL loop using the RM as the reward signal.

DPO collapses (2)–(4) into a single supervised loss on preference data. No reward model, no rollouts, no PPO. Just train the policy directly on the preference pairs you would have used to train the RM.

The cost is real (no online exploration, no separate RM artefact). Even so, a large fraction of modern preference-tuning uses DPO or a DPO derivative.

2. The standard RLHF objective

Classical PPO-RLHF maximises expected reward minus KL to a frozen reference policy:

$$\max_{\pi} \; \mathbb{E}_{x\sim D,\, y\sim\pi(\cdot|x)}[r(x,y)] - \beta\, D_{KL}(\pi(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x))\,.$$

r is the learned reward model; π_{ref} is the SFT model; β controls how far the policy may drift from the reference.

For a fixed r, this problem has a known **closed-form optimal policy**:

$$\pi^*(y \mid x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y \mid x)\exp\!\left(\frac{1}{\beta}\, r(x,y)\right), \qquad Z(x) = \sum_y \pi_{\text{ref}}(y \mid x)\exp(r(x,y)/\beta).$$

The optimal policy is the reference reweighted by an exponential reward, normalised. Standard maximum-entropy result.

3. The Rafailov inversion (DPO's key step)

Rearrange the closed-form expression to write r in terms of π^* and π_{ref}:

$$r(x,y) = \beta \log \frac{\pi^*(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x).$$

The reward is now a function of the optimal policy, the reference policy, and the partition function $Z(x)$ — which depends only on x, not on y.

Plug into Bradley-Terry. The reward model would have been trained on the likelihood:

$$P(y_w \succ y_l \mid x) = \sigma(r(x, y_w) - r(x, y_l)) \,.$$

Compute the reward *difference*:

$$r(x, y_w) - r(x, y_l) = \beta \log \frac{\pi^*(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi^*(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} + \underbrace{\beta \log Z(x) - \beta \log Z(x)}_{=0} .$$

The partition function **cancels**, and that cancellation is what makes DPO possible: reward differences are computable from policy and reference log-probs alone, no $Z(x)$ needed.

4. The DPO loss

Parameterise the policy as π_θ (the LLM you're training). The Bradley-Terry likelihood becomes:

$$P(y_w \succ y_l \mid x; \theta) = \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right) .$$

The DPO loss is the negative log-likelihood of the observed preferences:

$$\boxed{\mathcal{L}_{DPO}(\theta) = -\,\mathbb{E}_{(x, y_w, y_l) \sim D}\left[\log \sigma(\beta\, h_\theta(x, y_w, y_l))\right],}$$

where

$$h_\theta(x, y_w, y_l) = \underbrace{\left[\log \pi_\theta(y_w \mid x) - \log \pi_{\text{ref}}(y_w \mid x)\right]}_{\text{chosen log-ratio}} - \underbrace{\left[\log \pi_\theta(y_l \mid x) - \log \pi_{\text{ref}}(y_l \mid x)\right]}_{\text{rejected log-ratio}} .$$

Operationally, the loss is a log-sigmoid of a β-weighted difference of log-ratios.

5. What the loss is doing intuitively

The argument of $\log \sigma$ is positive when the policy assigns *more* probability than the reference to the chosen response, relative to the rejected one. Minimising the loss pushes the chosen log-ratio up and the rejected log-ratio down — i.e., reshape the policy so it prefers chosen over rejected by more than the reference does.

The reference acts as an anchor: you're not asked to maximise chosen probability absolutely, only to increase the chosen-vs-rejected gap beyond what the reference shows. This anchoring is what corresponds to the KL constraint in the original RL objective.

6. The reference model's role

π_{ref} is a frozen LLM, typically the SFT checkpoint. Its log-probs $\log \pi_{\text{ref}}(y \mid x)$ are computed once per training example and **detached** — gradients should never flow through the reference, since it's defining the anchor not being optimised.

In code:

```
ref_chosen_logp = ref_chosen_logp.detach()
ref_rejected_logp = ref_rejected_logp.detach()
```

Forgetting this detach is one of the easiest ways to make the loss look numerically plausible while training the wrong object.

In memory terms, training DPO requires **two LLMs**: the policy (π_θ) being updated, and the frozen reference (π_{ref}) for the anchor (and since the reference is frozen, its log-probs can be precomputed offline, dropping it to one). Classical PPO-RLHF carries four: policy, reference, reward model, and value model. DPO halves it.

7. The β parameter

β is the same temperature as in the original KL-constrained RL objective — large β corresponds to a strong KL constraint (policy stays close to reference); small β corresponds to a weak constraint (policy free to drift).

Typical values: $\mathbf{0.01 - 0.5}$. Sensitive to dataset and model scale. In practice, the effective β for matching PPO-RLHF behaviour is often smaller than papers' default of 0.1 — model size matters.

8. Implementation

The standard PyTorch shape:

```
import torch.nn.functional as F

def dpo_loss(
    chosen_logp,         # (B,)  Σ_t log π_θ(y_w,t | x, y_w,<t)
    rejected_logp,       # (B,)  Σ_t log π_θ(y_l,t | x, y_l,<t)
    ref_chosen_logp,     # (B,)  same under π_ref - frozen
    ref_rejected_logp,   # (B,)  same - frozen
    beta=0.1,
):
    ref_chosen_logp = ref_chosen_logp.detach()
    ref_rejected_logp = ref_rejected_logp.detach()

    chosen_log_ratio = chosen_logp - ref_chosen_logp
    rejected_log_ratio = rejected_logp - ref_rejected_logp
    logits = beta * (chosen_log_ratio - rejected_log_ratio)
    return -F.logsigmoid(logits).mean()
```

Each `*_logp` is the *summed* log-probability of the full response under the relevant model (summed over response tokens, optionally with a response mask handling padding). Computing those summed log-probs is where most of the practical engineering goes — applying the response mask correctly, handling prompt vs response tokens, batching efficiently — but the loss itself is one `logsigmoid` call.

9. Known issues with vanilla DPO

Vanilla DPO has well-documented failure modes. Each motivated a successor algorithm.

Length bias. Chosen responses tend to be longer than rejected ones in many preference datasets. DPO's implicit reward is a sum over tokens, so longer responses give the optimiser more room to inflate the margin. The policy learns to produce longer outputs as a shortcut. Fix: length normalisation (divide log-probs by response length) or use SimPO.

Reduction to base distribution. As training proceeds, DPO can lower the rejected log-prob much more aggressively than it raises the chosen one. The policy "improves" the loss by pushing rejected mass down, without putting that mass on chosen. End result: distributional drift where overall response quality drops. Fixes include stronger reference regularisation and bounded-margin variants such as IPO.

Sensitivity to reference. DPO is anchored to π_{ref}. A weak SFT reference produces a weak DPO policy. The reference is doing a lot of implicit work.

No online exploration. Pure offline supervised loss. The policy never sees its own rollouts, so it can't learn from outputs not present in the preference set. PPO with online rollouts can.

Distribution shift. Preferences were collected on SFT-model outputs. DPO trains a policy that will produce different outputs than SFT. The implicit assumption — "preferences generalise from SFT-distribution to DPO-distribution outputs" — can fail at the tail.

10. The DPO family

The post-DPO papers fix one or more of the issues above:

- **IPO** (Azar et al. 2023). Identity Preference Optimization. Squared loss instead of log-sigmoid. Prevents the log-ratio from being driven to infinity. Useful when preferences are noisy or systematic biases (e.g., length) are present.
- **KTO** (Ethayarajh et al. 2024). Kahneman-Tversky Optimization. Single-example version — needs only "this output is good/bad" labels, not pairs. Uses prospect-theory utility shape. Practical when only thumbs-up/thumbs-down data is available.
- **ORPO** (Hong et al. 2024). Combines SFT and preference loss in a single training step. Avoids the need for a separate SFT stage. Doesn't need a reference model.
- **SimPO** (Meng et al. 2024). Simple Preference Optimization. Drops the reference model entirely. Length-normalises log-probs. Practical when you don't want the memory cost of a second LLM during training. Can outperform vanilla DPO when length effects and reference cost dominate.

Read order if you want depth: **DPO** → **IPO** → **KTO** → **SimPO**. KTO and SimPO are useful next stops because they change different assumptions: paired-vs-unpaired feedback for KTO, and reference-free length-normalised training for SimPO.

11. DPO vs PPO-RLHF — the trade-off

	PPO-RLHF	DPO
Models in memory	4 (policy, ref, RM, value)	2 (policy, ref; 1 with precomputed ref log-probs)
Training data	preference pairs (for RM) + online rollouts	preference pairs only
Online exploration	yes	no
Reward model artefact	yes (reusable)	no
Complexity	high (RL loop)	low (supervised loss)
Hyperparameter sensitivity	many (β, ϵ, K epochs, etc.)	few (β mostly)
Empirical performance	strong; harder to tune	competitive; sometimes worse on hard tasks

DPO is the "supervised approximation" of PPO-RLHF — much simpler, but less able to improve by sampling its own new behaviours. The trade-off is often right for preference tuning; online RL is still the natural tool when exploration against a verifier or live reward is the point.

12. Summary

DPO derives that the optimal policy under KL-constrained RLHF can be parameterised by log-ratios against a reference, and that Bradley-Terry preference likelihoods over such policies have a partition-function-cancelling form, so the policy can be trained directly on preferences with a log-sigmoid loss, with no reward model and no rollouts.