NOT ALL TOKENS ARE EQUAL: ENTROPY-GUIDED CREDIT ASSIGNMENT FOR DIRECT PREFERENCE OPTIMIZATION

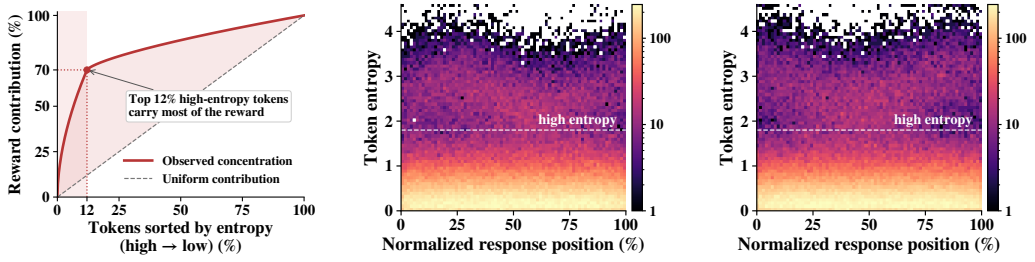
Anonymous authors
Paper under double-blind review

ABSTRACT

Direct Preference Optimization (DPO) aligns language models with a single sequence-level preference signal, leaving open how that signal should be attributed to individual tokens. We show empirically that this attribution is highly uneven: under the current policy, a small minority of high-entropy positions accounts for most of the implicit reward margin, whereas the low-entropy majority contributes almost nothing. Motivated by this observation, we propose Entropy-Guided DPO (EG-DPO), which reweights token-level policy–reference log-ratios by a detached, monotone function of the policy's own predictive entropy. The weights are normalized within each response, so the objective coincides with DPO when they are uniform, and they are recomputed at every step, so they track the evolving policy without adding a gradient path, an auxiliary model, or a forward pass. We justify the design with two complementary analyses: a signal-to-noise argument showing that uniform aggregation dilutes preference-relevant gradients, and a bound showing that the attainable pairwise preference-divergence at a position grows with the policy's entropy there. Across Llama-3-8B, Mistral-7B, and four Qwen2.5-Instruct scales, EG-DPO improves over DPO and recent token-level baselines on AlpacaEval 2, Arena-Hard v0.1, and MT-Bench, and yields the best average preference-classification accuracy on HH-RLHF, with its largest gain at the 14B scale.

1 INTRODUCTION

Aligning large language models (LLMs) with human preferences is now routinely performed with Direct Preference Optimization (DPO) (Rafailov et al., 2023), which replaces the reward-model and policy-gradient stages of RLHF (Ouyang et al., 2022; Schulman et al., 2017) with a single classification loss over policy–reference log-ratios. This simplicity comes at a price. The DPO objective compares two responses through their *sequence-level* log-ratios, and the sequence-level log-ratio is simply the unweighted sum of token-level log-ratios. Every token therefore receives the same share of the preference signal, regardless of whether it encodes the decision that made one



(a) Cumulative reward contribution by token entropy

(b) Entropy distribution over chosen-token positions

(c) Entropy distribution over rejected-token positions

Figure 1: Token-level structure of the DPO implicit reward. (a) Tokens are sorted by current-policy entropy in decreasing order and their contributions to the implicit reward $\beta \sum_t \delta_t$ are accumulated; the top 12% of tokens account for roughly 70% of the total, far above the uniform-contribution diagonal. (b–c) Joint density of predictive entropy and normalized response position for chosen and rejected tokens (log-scale colour). Most tokens have entropy below 1, and high-entropy tokens are sparse and spread across the whole response rather than concentrated at particular positions.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

response preferable or merely completes a predictable syntactic pattern. This is the credit assignment problem of DPO, and it has motivated a growing body of token-level variants (Zeng et al., 2024; Zhou et al., 2025; Zhu et al., 2025; Christopoulou et al., 2025; Li et al., 2025a).

We approach the problem by asking a simple diagnostic question: after DPO training, *where* does the implicit reward actually come from? Figure 1 answers this with the policy's own predictive entropy as the coordinate. Sorting tokens by entropy and accumulating their contributions to the implicit reward (Figure 1a), the top 12% highest-entropy tokens account for most of the reward, while the remaining 88% contribute little in aggregate. Figures 1b–1c show why: the vast majority of tokens sit in a low-entropy band (below 1 nat), and high-entropy tokens are a sparse minority scattered across the response. Two conclusions follow. First, within a response, high-entropy tokens are few and low-entropy tokens are many. Second, the few high-entropy tokens carry a disproportionate share of the preference signal, so treating all tokens uniformly dilutes the informative positions with a large number of uninformative ones.

These observations suggest a direct remedy: weight each token's log-ratio by how uncertain the current policy is at that position. We call the resulting method **Entropy-Guided DPO (EG-DPO)**. During the ordinary policy forward pass we read off both the target-token log-probability and the full-vocabulary predictive entropy. The entropy is detached from the computation graph, standardized within each response, passed through a sigmoid gate, and rescaled so that the weights sum to the number of response tokens. The weighted log-ratios then replace the unweighted sum in the DPO implicit reward. Because of the rescaling, EG-DPO reduces exactly to DPO when the weights are uniform, and because the weights are recomputed at every optimization step, they follow the evolving policy rather than a frozen initialization. The overhead over DPO is a vocabulary-dimension reduction on logits that are already computed.

Why should *policy* entropy be the right coordinate, rather than, say, the probability of the realized token or the entropy of the reference model? We give two arguments. Proposition 1 shows that if only a subset of positions carries a stable preference direction, then reallocating weight toward those positions increases the signal-to-noise ratio of the DPO gradient; uniform weights are optimal only in a degenerate case. Proposition 2 shows that, at a fixed policy iterate, the expected squared preference-divergence at a position is bounded by a Schur-concave function of the policy's next-token distribution, so positions where the policy is nearly deterministic cannot express a large preference difference in expectation. Together these results identify current-policy entropy as an online, label-free proxy for where preference signal can reside, which is exactly what a detached, per-step weighting scheme needs.

Our contributions are as follows.

- We identify and quantify the concentration of the DPO implicit reward on a small set of high-entropy tokens, and propose EG-DPO, a token-weighted DPO objective whose weights are derived from the current policy's detached predictive entropy. EG-DPO introduces one hyperparameter, requires no auxiliary model, sampling, or extra forward pass, and reduces to DPO under uniform weights.
- We provide a theoretical account of the design from two angles: a gradient signal-to-noise analysis of weighted token aggregation, and an upper bound relating pairwise preference-divergence capacity to the Gini–Simpson diversity of the current policy, which is monotonically consistent with Shannon entropy under majorization.
- We evaluate EG-DPO on Llama-3-8B and Mistral-7B and on four Qwen2.5-Instruct scales, against ten sequence- and token-level baselines. EG-DPO attains the best AlpacaEval 2 and Arena-Hard scores in every setting and the best average HH-RLHF preference accuracy, and we ablate the entropy source, the gating function, the gating sharpness, and the two-sided weighting.

2 Related Work

Sequence-level preference optimization. RLHF fits a Bradley–Terry (BT) reward model (Bradley & Terry, 1952) and optimizes the policy with PPO under a KL constraint (Ouyang et al., 2022); DPO (Rafailov et al., 2023) removes both stages by rewriting KL-regularized reward maximization as a binary cross-entropy loss over policy–reference log-ratios. Subsequent work refines this objective along several axes: IPO (Azar et al., 2024) regularizes against overfitting

to deterministic preferences; KTO (Ethayarajh et al., 2024) relaxes the paired-data requirement; ORPO (Hong et al., 2024) and CPO (Xu et al., 2024) fold an SFT term into the preference objective; SimPO (Meng et al., 2024) drops the reference model and adds length normalization with a target margin; and R-DPO (Park et al., 2024) regularizes length explicitly. All of these methods aggregate tokens uniformly. EG-DPO changes only how token-level log-ratios are aggregated and could in principle be combined with the length or margin modifications above.

Token-level preference optimization. TDPO (Zeng et al., 2024) reformulates DPO as per-token policy optimization with token-wise forward-KL constraints. T-REG (Zhou et al., 2025) and TGDPO (Zhu et al., 2025) obtain token-level rewards from contrastive prompting or from implicit reward strength and use them as a regularizer or as a guide for how far each token may deviate from the reference. SparsePO (Christopoulou et al., 2025) learns sparse token masks, and OTPO (Li et al., 2025a) uses optimal transport to align semantically related token pairs across the two responses. Closest to EG-DPO are methods that reweight tokens by a scalar importance score: ConfPO (Yoon et al., 2025) keeps only tokens whose realized-token probability is below the response mean, and TIS-DPO (Liu et al., 2025) derives importance weights from a pair of contrastively trained preference models.

EG-DPO differs from these methods in the importance signal, in its treatment during optimization, and in cost. The signal is the predictive entropy of the current policy over the *entire* vocabulary, not the probability of the single realized token (ConfPO) and not a quantity produced by additional preference models (TIS-DPO, T-REG); it therefore reflects how uncertain the policy is at a position rather than how well it happens to fit the observed token. The signal is applied as a continuous, detached weight on the policy–reference log-ratio, so the DPO gradient form is preserved exactly within each step while the weights still track the policy across steps. Finally, the signal is computed from logits produced by the policy forward pass, so EG-DPO requires no additional model forward passes. Entropy-aware token selection has also been explored in reinforcement learning with verifiable rewards (Wang et al., 2025); that setting differs in supervision, selection mechanism, and objective.

3 Preliminaries

Given a dataset $\mathcal{D} = \{(x, y_w, y_l)\}$ in which y_w is preferred to y_l for prompt x, RLHF first fits a reward model r_ϕ by maximizing the BT likelihood $\mathbb{E}_{\mathcal{D}}[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))]$, with σ the logistic function, and then maximizes expected reward under a KL penalty toward a reference policy π_{ref},

$$\max_{\pi_\theta} \; \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta}\big[r(x,y)\big] - \beta\, \mathbb{E}_{x\sim\mathcal{D}}\big[D_{\text{KL}}\big(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\big)\big], \tag{1}$$

where $\beta > 0$ controls the strength of the penalty. The optimum of Eq. (1) has the closed form $\pi^*(y \mid x) = \pi_{\text{ref}}(y \mid x)\exp(r(x,y)/\beta)/Z(x)$, which can be inverted to express the reward as $r(x,y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$. Since the partition function $Z(x)$ cancels in the reward difference of two responses to the same prompt, substituting this expression into the BT likelihood yields an objective over the policy alone,

$$r_\theta(x,y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}, \qquad \mathcal{L}_{\text{DPO}} = -\,\mathbb{E}_{\mathcal{D}}\big[\log \sigma\big(r_\theta(x,y_w) - r_\theta(x,y_l)\big)\big], \tag{2}$$

where r_θ is the *implicit reward*.

4 Entropy-Guided DPO

Figure 2 summarizes EG-DPO; we define its components in turn and analyze them in §5.

4.1 Token-Level Decomposition of the Implicit Reward

For a prompt x and a response $y = (y_1, \ldots, y_T)$, both policies factorize autoregressively, $\log \pi(y \mid x) = \sum_{t=1}^{T} \log \pi(y_t \mid x, y_{<t})$. The implicit reward in Eq. (2) therefore decomposes as

$$r_\theta(x,y) = \beta \sum_{t=1}^{T} \delta_t(x,y), \qquad \delta_t(x,y) = \log \pi_\theta(y_t \mid x, y_{<t}) - \log \pi_{\text{ref}}(y_t \mid x, y_{<t}). \tag{3}$$

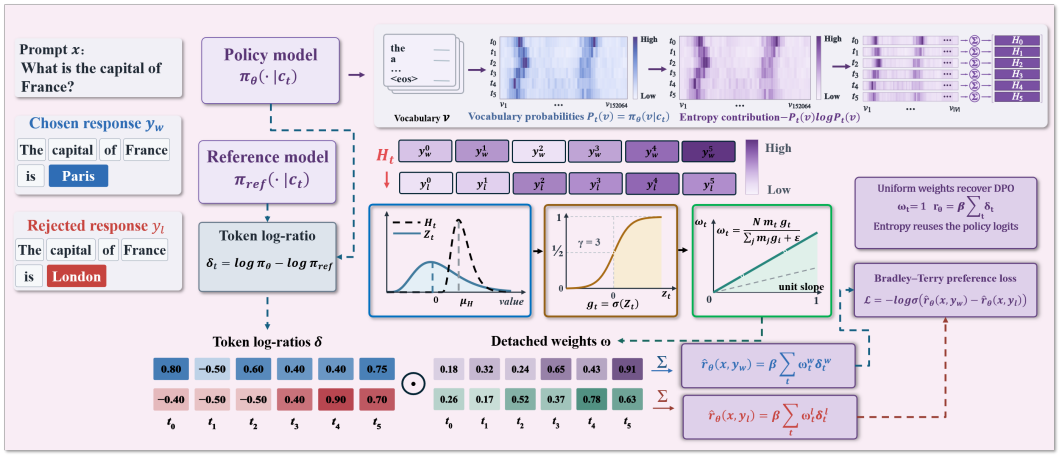


Figure 2: Overview of EG-DPO. For a preference triplet (x, y_w, y_l), the policy π_θ and reference π_{ref} produce token-level log-probabilities for both responses, from which the policy–reference log-ratios δ_t are formed. The policy logits additionally yield predictive entropies H_t, which are detached from the computation graph, z-scored within each response, gated by a sigmoid, and rescaled so that the weights sum to the number of response tokens. The weighted log-ratios are summed and scaled by β to give the chosen and rejected implicit rewards, whose difference enters the BT loss.

We refer to δ_t as the *token-level log-ratio*; it is the contribution of position t to the implicit reward, and DPO aggregates these contributions with uniform weight. For a preference pair we write δ_t^w and δ_t^l for the log-ratios of the chosen and rejected responses, computed on their own prefixes $y_{<t}^w$ and $y_{<t}^l$.

4.2 Predictive Entropy of the Current Policy

At each position we measure the policy's uncertainty by its predictive (Shannon) entropy over the vocabulary $\mathcal{V}$,

$$H_t^\theta(x, y_{<t}) = -\sum_{v \in \mathcal{V}} \pi_\theta(v \mid x, y_{<t}) \log \pi_\theta(v \mid x, y_{<t}). \tag{4}$$

Two properties of this quantity matter for what follows. First, it depends only on the prefix $(x, y_{<t})$ and not on the realized token y_t; it is a property of the *position*, not of the label. Second, it requires no additional model forward pass: the logits needed to compute it are already produced by the policy forward pass that computes $\log \pi_\theta(y_t \mid x, y_{<t})$. We use a stop-gradient copy, $H_t = \text{sg}[H_t^\theta]$, where $\text{sg}[\cdot]$ is the identity in the forward pass and has zero derivative in the backward pass. Consequently, within a single optimization step the weights derived from H_t act as fixed coefficients and the gradient retains the DPO form (Proposition 1); across steps, H_t is recomputed from the updated policy and therefore tracks the policy's current uncertainty rather than that of a frozen initialization. §6.4 shows empirically that this online property matters: weighting by the frozen reference model's entropy is consistently worse.

4.3 From Entropy to Token Weights

Using H_t directly as a weight is problematic for two reasons. Its scale varies across models, domains, and response lengths, so absolute values are not comparable across samples. More importantly, what should matter is not the absolute entropy of a position but whether it is uncertain *relative to the other positions in the same response*. We therefore map entropy to weights in three steps, described for one response; the chosen and rejected responses are processed independently and identically.

Step 1: within-response standardization. Let $m_t \in \{0, 1\}$ be the response mask and $N = \sum_t m_t$ the number of response tokens. We compute the masked mean and standard deviation of the entropies and standardize,

$$\mu_H = \frac{1}{N} \sum_t m_t H_t, \qquad \sigma_H = \sqrt{\frac{1}{N} \sum_t m_t (H_t - \mu_H)^2}, \qquad z_t = \gamma \cdot \frac{H_t - \mu_H}{\sigma_H + \varepsilon}, \tag{5}$$

where $\gamma > 0$ controls the sharpness of the subsequent gate and $\varepsilon = 10^{-6}$ guards against constant-entropy responses.

Step 2: sigmoid gating. The standardized scores are mapped to $(0, 1)$,

$$\tilde{\omega}_t = \sigma(z_t) \cdot m_t. \tag{6}$$

Positions with above-average entropy receive $\tilde{\omega}_t > \frac{1}{2}$ and positions with below-average entropy receive $\tilde{\omega}_t < \frac{1}{2}$. As $\gamma \to \infty$ the gate approaches a hard selection of the above-average positions; as $\gamma \to 0$ it approaches uniform weights. We use $\gamma = 3$ throughout, chosen by the ablation in Appendix E.2.

Step 3: sum-preserving rescaling. Finally we rescale so that the weights sum to the number of response tokens,

$$\omega_t = \tilde{\omega}_t \cdot \frac{N}{\sum_{t'} \tilde{\omega}_{t'} + \varepsilon}, \qquad \text{so that} \qquad \sum_t \omega_t = N. \tag{7}$$

This keeps the scale of the weighted implicit reward identical to that of DPO, which in turn keeps β and the learning rate transferable from DPO without retuning. It also implies that weighting is purely a *redistribution*: weight moved onto high-entropy positions is taken from low-entropy ones.

4.4 Objective and Basic Properties

The entropy-weighted implicit reward replaces the uniform sum in Eq. (3), and the EG-DPO objective is the BT loss on the weighted rewards:

$$\hat{r}_\theta(x, y) = \beta \sum_{t=1}^{T} \omega_t \, \delta_t(x, y), \qquad \mathcal{L}_{\text{EG-DPO}} = -\,\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \Big[\log \sigma\big(\hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l)\big) \Big]. \tag{8}$$

Three properties follow directly from the construction.

1. *Reduction to DPO.* If the entropies within a response are constant, then $z_t \equiv 0$, $\tilde{\omega}_t \equiv \frac{1}{2}$, and after rescaling $\omega_t \equiv 1$, so $\hat{r}_\theta = r_\theta$ and Eq. (8) is exactly the DPO loss. The same holds in the limit $\gamma \to 0$.
2. *Gradient form.* Because $\omega_t = \mathrm{sg}[\omega_t(\pi_\theta)]$, the gradient of Eq. (8) is $-\beta(1 - \sigma(u)) \sum_t \big(\omega_t^w \nabla_\theta \log \pi_\theta(y_t^w \mid \cdot) - \omega_t^l \nabla_\theta \log \pi_\theta(y_t^l \mid \cdot)\big)$ with $u = \hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l)$, i.e., the DPO gradient with per-token coefficients (Appendix C.1).
3. *Cost.* Relative to DPO, the additional computation is one entropy reduction over the vocabulary dimension of logits that are already materialized, and $\mathcal{O}(T)$ scalar operations per response. There are no additional parameters, models, forward passes, or sampled data (Appendix D).

5 Why Weight by Current-Policy Entropy

We now give two complementary arguments for the design in §4. Proposition 1 concerns *aggregation*: it shows that, whenever preference-relevant signal is concentrated on a subset of positions, uniform aggregation is suboptimal for the gradient's signal-to-noise ratio. Proposition 2 concerns the *choice of signal*: it shows that the current policy's entropy at a position upper-bounds, in expectation, how much preference divergence that position can express. Neither result claims that entropy ranks individual tokens by their true importance (see Appendix A); together they justify entropy as an online, label-free proxy for where signal can be. Full proofs are in Appendix C.

5.1 Signal Dilution under Uniform Aggregation

Consider the token-weighted loss $\mathcal{L} = -\log \sigma\big(\beta \sum_t w_t \Delta_t\big)$ with detached weights $w_t \geq 0$, $\sum_t w_t = N$, and $\Delta_t = \delta_t^w - \delta_t^l$ (positions are aligned across the pair for notational simplicity; Appendix C.1 treats unequal lengths). Its gradient is $\nabla_\theta \mathcal{L} = -\beta(1 - \sigma(u)) \sum_t w_t \nabla_\theta \Delta_t$. Decompose the token-level gradient into a preference-related signal and zero-mean fitting noise, $\nabla_\theta \Delta_t = g_t + \xi_t$ with $\mathbb{E}[\xi_t] = 0$ and $\mathrm{Var}(\xi_t) = \Sigma_t$, and define the gradient signal-to-noise ratio $\mathrm{SNR}(w) = \| \sum_t w_t g_t \|^2 / \mathbb{E} \| \sum_t w_t \xi_t \|^2$.

Proposition 1 (Signal dilution). *Suppose the signal components share a common direction, $g_t = a_t\,\bar{g}$ with $a_t \geq 0$ and $\|\bar{g}\| = 1$, and let $s_t = \mathrm{tr}(\Sigma_t) > 0$. Then* $\mathrm{SNR}(w) = (\sum_t w_t a_t)^2 / \sum_t w_t^2 s_t \leq \sum_t a_t^2/s_t$, *with equality if and only if $w_t \propto a_t/s_t$. In particular, uniform weights are SNR-optimal only if a_t/s_t is constant across positions, and moving weight from a position with smaller a_t/s_t to one with larger a_t/s_t increases the SNR.*

The proposition formalizes the intuition behind Figure 1a: if most positions contribute noise but little stable preference direction, DPO's uniform sum averages the informative minority against the uninformative majority. It does not by itself say *which* positions have large a_t; that is the role of the next result.

5.2 Entropy as a Proxy for Preference-Divergence Capacity

Fix a position with prefix context c and a policy iterate θ, and let $p = \pi_\theta(\cdot \mid c)$ be the policy's next-token distribution there. We make two local assumptions, discussed in Appendix C.2: (A1) the token-wise log-ratio is bounded, $|\log \pi_\theta(v \mid c) - \log \pi_{\mathrm{ref}}(v \mid c)| \leq M$ for all v, which is the token-level counterpart of the KL regularization in DPO; and (A2) the chosen and rejected tokens at this position are modeled as independent draws from p, a surrogate for the offline pair at the current iterate whose relaxation to bounded distribution shift is given in Appendix C.4.

Proposition 2 (Divergence capacity). *Under (A1)–(A2), the pairwise preference-divergence $\Delta_t = \delta_t^w - \delta_t^l$ satisfies*

$$\mathbb{E}\big[\Delta_t^2 \mid c\big] \;\leq\; 4M^2\big(1 - \|p\|_2^2\big), \tag{9}$$

where $1 - \|p\|_2^2$ is the Gini–Simpson diversity of p. Both $1 - \|p\|_2^2$ and the Shannon entropy $H(p)$ are symmetric and Schur-concave; hence if p is more concentrated than p' in the majorization order, then both the bound and H are smaller at p than at p', and both vanish exactly at one-hot distributions.

The bound has a simple interpretation. When the policy is nearly deterministic at a position, the chosen and rejected tokens almost surely coincide there, so $\Delta_t = 0$ and the position cannot contribute to the reward margin. When the policy is uncertain, the two tokens can differ and the log-ratios can diverge by up to $2M$. Entropy measures exactly this concentration, and it does so using only the current policy, so it can be recomputed at every step without labels or auxiliary models. Appendix C.3 contrasts this pairwise statement with the behaviour of a single sequence's $|\delta_t|$, which is governed by the *reference* probability of the realized token rather than by policy entropy; the distinction is why EG-DPO weights Δ_t-carrying positions by policy entropy rather than by realized-token probability.

Implication. Proposition 1 says that weight should move toward positions with more preference signal per unit noise; Proposition 2 says that low-policy-entropy positions have little attainable signal. A weight that increases monotonically with detached H_t, as in Eq. (6), is the simplest rule consistent with both. The ablations in §6.3 and Appendix E.2 test this rule against its reversal, against one-sided weighting, and against alternative monotone gates.

6 Experiments

6.1 Experimental Setup

Models. We evaluate on three model families: **Qwen2.5-Instruct** (Yang et al., 2024) at 1.5B, 3B, 7B, and 14B parameters; **Llama-3-8B** (Grattafiori et al., 2024) in Base and Instruct variants; and **Mistral-7B-v0.1** (Jiang et al., 2023) in Base and Instruct-v0.1 variants. Base models are first supervised fine-tuned (SFT) on the chosen responses; the SFT checkpoint serves as both the initialization and the reference π_{ref} for every preference-optimization method.

Data. For open-ended generation we use the 64K-prompt subset of **UltraFeedback** (Cui et al., 2024), which provides four GPT-4-scored responses per prompt; the highest- and lowest-scoring responses form (y_w, y_l). For preference classification we use **Anthropic HH-RLHF** (Bai et al., 2022), training on the official training split and evaluating on the official test split of 4,666 pairs.

Benchmarks and metrics. UltraFeedback-trained models are evaluated on **AlpacaEval 2** (Dubois et al., 2024) (length-controlled, LC, and raw win rate, WR, against GPT-4-Turbo), **Arena-Hard v0.1** (Li et al., 2025b) (WR against GPT-4-0314), and **MT-Bench** (Zheng et al., 2023)

Table 1: Main results on the Base and Instruct variants of Llama-3-8B and Mistral-7B-v0.1 across AlpacaEval 2, Arena-Hard, and MT-Bench. LC and WR denote length-controlled and standard win rates, respectively. Best results are shown in bold.

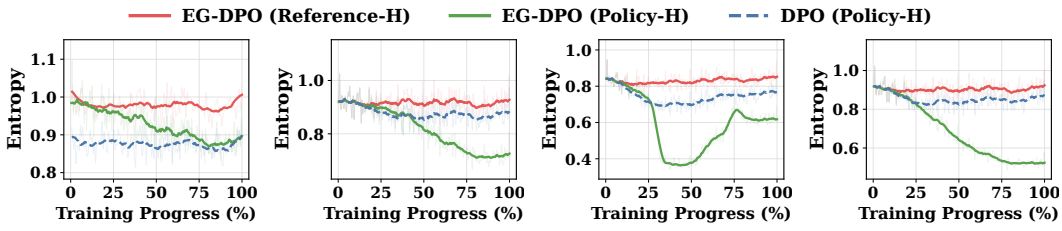
Method	Llama-3-Base (8B)				Llama-3-Instruct (8B)			
	AlpacaEval 2		Arena-Hard	MT-Bench	AlpacaEval 2		Arena-Hard	MT-Bench
	LC (%)	WR (%)	WR (%)	Score	LC (%)	WR (%)	WR (%)	Score
SFT	7.1	4.9	4.2	6.100	27.7	20.3	14.9	6.740
DPO	18.3	15.3	9.2	7.010	34.4	27.5	21.9	7.560
IPO	16.2	18.4	11.4	7.062	33.8	26.6	20.2	7.644
CPO	15.8	16.6	5.4	7.503	29.8	24.5	19.2	**8.101**
KTO	17.8	15.9	9.0	6.820	32.8	22.1	18.5	7.360
ORPO	12.7	10.2	8.1	6.960	30.4	29.3	27.5	7.480
R-DPO	15.8	18.0	11.2	7.220	35.7	34.4	24.0	7.720
SimPO	26.9	23.6	12.6	7.468	37.1	34.7	25.3	7.735
TDPO	26.4	25.3	13.2	7.528	37.6	33.1	28.0	7.802
TIS-DPO	27.0	27.6	13.8	7.544	38.0	35.3	28.4	7.838
ConfPO	27.1	28.4	14.3	7.565	38.3	35.9	28.7	7.861
EG-DPO	**27.2**	**30.9**	**15.1**	**7.606**	**38.6**	**36.8**	**29.3**	7.925
Method	**Mistral-Base (7B)**				**Mistral-Instruct (7B)**			
	AlpacaEval 2		Arena-Hard	MT-Bench	AlpacaEval 2		Arena-Hard	MT-Bench
	LC (%)	WR (%)	WR (%)	Score	LC (%)	WR (%)	WR (%)	Score
SFT	9.0	7.1	4.8	6.050	10.9	19.2	12.5	6.820
DPO	16.8	14.4	9.1	6.920	17.3	19.5	14.8	7.210
IPO	11.7	8.3	7.4	7.244	12.0	18.5	13.1	7.044
CPO	10.3	9.2	6.9	7.338	16.6	19.5	14.3	7.270
KTO	10.2	7.9	5.6	6.780	10.5	18.2	12.7	7.090
ORPO	17.5	11.3	7.0	6.950	18.1	21.6	15.0	7.180
R-DPO	19.3	13.7	8.0	7.120	19.9	24.1	15.3	7.390
SimPO	23.7	17.4	8.6	7.405	24.5	32.1	23.1	7.520
TDPO	24.3	18.4	9.2	7.463	26.2	32.9	24.3	7.585
TIS-DPO	24.8	18.8	9.6	7.492	26.8	33.4	24.7	7.624
ConfPO	25.4	19.2	10.1	7.518	27.3	33.9	25.1	7.651
EG-DPO	**26.7**	**19.9**	**10.9**	**7.612**	**28.1**	**34.6**	**25.7**	**7.735**

(mean turn-level score on a 1–10 scale), all with GPT-4 as judge (OpenAI, 2023). HH-RLHF-trained models are evaluated by *preference classification accuracy*: a test pair is counted correct if $r_\theta(x, y_w) > r_\theta(x, y_l)$ under the *unweighted* implicit reward of Eq. (2). This criterion is identical for all methods, uses the same SFT reference, and requires no external judge; in particular, EG-DPO is evaluated without its training-time weights.

Baselines and protocol. We compare against SFT, DPO, IPO, CPO, KTO, ORPO, R-DPO, SimPO, and the token-level methods TDPO, TIS-DPO, and ConfPO. All methods share the SFT initialization and reference, LoRA configuration, training data, global batch size of 64, one epoch of training, and evaluation protocol. Method-specific hyperparameters (β, margins, SFT-mixing coefficients) follow the recommendations of the respective papers so that each baseline is run in its intended regime; EG-DPO inherits every DPO hyperparameter unchanged and adds only $\gamma = 3$. Full details are in Appendix D.

6.2 Main Results

Table 1 reports results on Llama-3-8B and Mistral-7B. EG-DPO obtains the best AlpacaEval 2 LC and WR and the best Arena-Hard WR in all four settings, and improves over DPO on every metric, including MT-Bench. Against the strongest token-level baselines the margins are consistent but modest on Llama-3-Base LC (27.2 vs. 27.1 for ConfPO) and larger on raw WR and Arena-Hard (e.g., 30.9 vs. 28.4 WR and 15.1 vs. 14.3 on Llama-3-Base; 19.9 vs. 19.2 WR and 10.9 vs. 10.1



(a) Qwen2.5-3B-Instruct, chosen. (b) Qwen2.5-7B-Instruct, chosen. (c) Mistral-7B-Instruct, chosen. (d) Llama-3-8B-Instruct, chosen.

(e) Qwen2.5-3B-Instruct, rejected. (f) Qwen2.5-7B-Instruct, rejected. (g) Mistral-7B-Instruct, rejected. (h) Llama-3-8B-Instruct, rejected.

Figure 3: Mean response-token predictive entropy of the weighting signal over normalized training progress on UltraFeedback (top: chosen; bottom: rejected). Thin curves are per-step means and thick curves are centred moving averages. The policy entropy of a standard DPO run is shown as a diagnostic. The Qwen2.5-3B DPO trace uses a different initialization and is included for qualitative reference only.

on Mistral-Base). On MT-Bench, EG-DPO is the best method in three of four settings; on Llama-3-Instruct, CPO scores higher (8.101 vs. 7.925), which we attribute to the SFT term in CPO's objective and note that CPO trails substantially on the win-rate benchmarks in the same setting. Because the four settings span two model families and both Base and Instruct initializations, the pattern indicates that the gains from entropy-guided weighting are not specific to a particular starting policy.

6.3 Component Ablations

To isolate which parts of the weighting rule matter, we compare three variants on Qwen2.5-7B-Instruct with identical data, hyperparameters, and LoRA configuration (Table 2). *Chosen-only* and *rejected-only* apply entropy weighting to one response and leave the other uniform; *reverse* negates z_t so that low-entropy tokens are up-weighted. Two conclusions emerge. First, the weighting must be applied to *both* sides. Chosen-only weighting raises LC over DPO but lowers raw WR and Arena-Hard WR, and rejected-only weighting collapses. This asymmetry is what Eq. (8) predicts: the loss depends on the *difference* of the two weighted rewards, so weighting one side alone changes the relative scale of the chosen and rejected terms and shifts the optimization toward whichever side retains the uniform sum. Second, the *direction* of the weighting matters. Reverse gating degrades to the same degree as rejected-only weighting, which is consistent with Proposition 2: concentrating weight on low-entropy positions concentrates it where the attainable preference divergence is smallest. We note that the two collapsed variants also show a sharp MT-Bench drop, indicating a loss of general capability rather than merely a smaller win margin; the qualitative examples in Appendix E.4 are consistent with this.

Table 2: Component ablations on Qwen2.5-7B-Instruct (UltraFeedback).

Method	**AlpacaEval 2**		**Arena-Hard**	**MT-Bench**
	LC (%)	WR (%)	WR (%)	Score
DPO	23.87	16.40	13.80	8.200
EG-DPO	**31.76**	**24.99**	**22.00**	**8.444**
Chosen-only	26.99	15.34	8.70	8.175
Rejected-only	4.20	2.00	0.00	4.156
Reverse	4.88	2.80	0.20	4.132

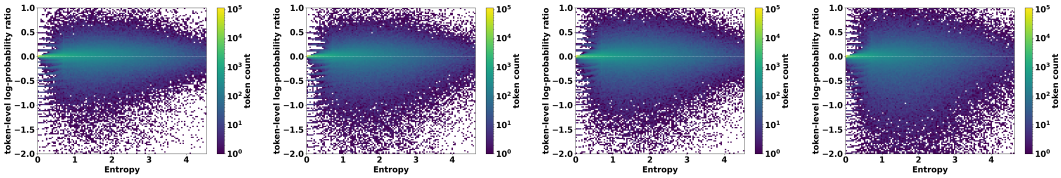
6.4 Policy Entropy or Reference Entropy?

A natural alternative to the current-policy entropy is the entropy of the frozen reference model, which would make the weights static. Figure 3 helps explain why by tracking the mean entropy of the weighting signal during training. The reference-entropy signal is stationary by construction.

The policy-entropy signal instead moves with the policy and, for Qwen2.5-7B and Llama-3-8B, decreases steadily on both chosen and rejected responses as training sharpens the policy; for Mistral it drops early and partially recovers. Standard DPO exhibits milder changes in mean entropy, and rejected responses retain higher entropy than chosen ones throughout. Since the weights are standardized within each response, what matters is not the mean level but the ranking of positions, and the token-level heatmaps in Appendix E.3 show that this ranking shifts as the policy trains. A static reference signal cannot follow these shifts.

6.5 Where Does the Reward Margin Come From?

Figure 4 shows, for Qwen2.5-1.5B-Instruct after training, the joint density of token-level log-ratio and entropy. Relative to DPO, EG-DPO extends the high-density band of non-zero log-ratios toward higher entropy on both the chosen ((a) vs. (c)) and rejected ((b) vs. (d)) sides, i.e., the reward margin is realized more on high-entropy positions, and the overall spread of log-ratios widens. The effect is more pronounced on the rejected side, where log-ratios become markedly more negative. This is consistent with the well-documented tendency of DPO to widen the margin primarily by decreasing the likelihood of rejected responses rather than increasing that of chosen ones; EG-DPO does not change this tendency but concentrates it on the positions where the policy is uncertain.

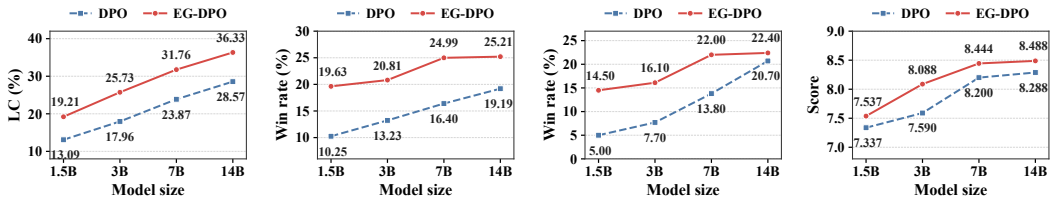


(a) DPO, chosen. (b) DPO, rejected. (c) EG-DPO, chosen. (d) EG-DPO, rejected.

Figure 4: Joint density of token-level log-ratio δ_t (vertical) and policy entropy (horizontal), log-scale colour, for Qwen2.5-1.5B-Instruct trained on UltraFeedback.

6.6 Scaling Behaviour

Figure 5 compares DPO and EG-DPO on Qwen2.5-Instruct from 1.5B to 14B. EG-DPO is better on all four metrics at every scale. Its AlpacaEval 2 LC rises monotonically from 19.21 to 36.33 (DPO: 13.09 to 28.57) and its MT-Bench score from 7.537 to 8.488 (DPO: 7.337 to 8.288). Raw WR and Arena-Hard WR also increase across the four scales; at 14B, EG-DPO reaches 25.21 raw WR, compared with 19.19 for DPO. On HH-RLHF preference classification (Appendix E.1, Table 4), EG-DPO has the best average accuracy across the four scales and the best accuracy at 3B, 7B, and 14B; its gain over DPO is largest at 14B (+1.26 points), indicating that entropy-guided weighting remains useful as capacity grows.



(a) AlpacaEval 2 LC. (b) AlpacaEval 2 WR. (c) Arena-Hard WR. (d) MT-Bench.

Figure 5: DPO versus EG-DPO across four Qwen2.5-Instruct scales (1.5B–14B) trained on UltraFeedback.

7 Conclusion

We showed that the DPO implicit reward is concentrated on a small set of high-entropy positions and proposed EG-DPO, which reweights token-level log-ratios by a detached, within-response-normalized function of the current policy's predictive entropy. The method adds one hyperparameter and requires no additional model forward passes, reduces to DPO under uniform weights, and is supported by a gradient signal-to-noise argument and a bound relating attainable preference divergence to policy entropy. Across three model families, EG-DPO improves over DPO and recent token-level methods on generation benchmarks and on preference classification. Its main limitation is that entropy is a proxy for where preference signal *can* be, not a measurement of where it *is*; combining it with label-dependent signals is a natural next step.

AI USE STATEMENT

We used a large language model (Claude, Anthropic) for the following tasks. Required-disclosure tasks: assisting in the writing and providing feedback on the presentation. The LLM was not used to generate data, propose the method or hypotheses, implement the method, or interpret results; the remaining required-disclosure tasks are not applicable. Recommended-disclosure tasks: editing the manuscript for readability and structure, drafting revised versions. All AI-assisted derivations were independently re-derived and verified by the authors. The scientific ideas, experimental design, data, results, and claims are the authors' own. We take full responsibility for the final content.

REPRODUCIBILITY STATEMENT

The objective is fully specified by Eqs. (3)–(8), proofs are given in Appendix C, and implementation details, pseudocode, and all hyperparameters are documented in Appendix D. Code will be released upon publication.

REFERENCES

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Rémi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *AISTATS*, pp. 4447–4455, 2024. 2

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022. 6

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. 2

Fenia Christopoulou, Ronald Cardenas, Gerasimos Lampouras, Haitham Bou Ammar, and Jun Wang. SparsePO: Controlling preference alignment of LLMs via sparse token masks. In *Findings of EMNLP*, pp. 25477–25503, 2025. 2, 3

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. UltraFeedback: Boosting language models with scaled AI feedback. In *ICML*, volume 235, pp. 9722–9744, 2024. 6

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled AlpacaEval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024. 6

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. KTO: Model alignment as prospect theoretic optimization. In *ICML*, 2024. 3

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Vaughan, Angela Yang, Anirudh Fan, Anthony Goyal, Aobo Hartshorn, Archi Yang, Arielle Mitra, Arkaitz Levy, Asa Zubiaga, Stickland Cooper, and Asaf Yehuda. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 6

Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic preference optimization without reference model. In *EMNLP*, pp. 11170–11189, 2024. 3

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. 6

Meng Li, Guangda Huzhang, Haibo Zhang, Xiting Wang, and Anxiang Zeng. Optimal transport-based token weighting scheme for enhanced preference optimization. In *ACL*, pp. 21311–21334, 2025a. 2, 3

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. In *ICML*, volume 267, pp. 34209–34231, 2025b. 6

Aiwei Liu, Haoping Bai, Zhiyun Lu, Yanchao Sun, Xiang Kong, Xiaoming Wang, Jiulong Shan, Albin Madappally Jose, Xiaojiang Liu, Lijie Wen, Philip Yu, and Meng Cao. TIS-DPO: Token-level importance sampling for direct preference optimization with estimated weights. In *ICLR*, 2025. 3, 15

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *NeurIPS*, volume 37, 2024. 3

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 7

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, volume 35, pp. 27730–27744, 2022. 1, 2

Ryan Park, Rafael Rafailov, Stefano Ermon, and Chelsea Finn. Disentangling length from quality in direct preference optimization. In *Findings of ACL*, pp. 4998–5017, 2024. 3

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct Preference Optimization: Your language model is secretly a reward model. In *NeurIPS*, volume 36, 2023. 1, 2

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 1

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 15

Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xiong-Hui Chen, Jianxin Yang, Zhenru Zhang, Yuqiong Liu, An Yang, Andrew Zhao, Yang Yue, Shiji Song, Bowen Yu, Gao Huang, and Junyang Lin. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for LLM reasoning. In *NeurIPS*, volume 38, 2025. 3, 15

Haoran Xu, Amr Sharaf, Yunmo Chen, Weiting Tan, Lingfeng Shen, Benjamin Van Durme, Kenton Murray, and Young Jin Kim. Contrastive preference optimization: Pushing the boundaries of LLM performance in machine translation. In *ICML*, 2024. 3

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and Keming Lu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. 6

Hee Suk Yoon, Eunseop Yoon, Mark A. Hasegawa-Johnson, Sungwoong Kim, and Chang D. Yoo. ConfPO: Exploiting policy model confidence for critical token selection in preference optimization. In *ICML*, volume 267, pp. 72641–72655, 2025. 3, 14

Yongcheng Zeng, Guoqing Liu, Weiyu Ma, Ning Yang, Haifeng Zhang, and Jun Wang. Token-level direct preference optimization. In *ICML*, volume 235, pp. 58348–58365, 2024. 2, 3

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot arena. In *NeurIPS*, volume 36, 2023. 6

Wenxuan Zhou, Shujian Zhang, Lingxiao Zhao, and Tao Meng. T-REG: Preference optimization with token-level reward regularization. In *ACL*, pp. 27876–27889, 2025. 2, 3

Mingkang Zhu, Xi Chen, Zhongdao Wang, Bei Yu, Hengshuang Zhao, and Jiaya Jia. TGDPO: Harnessing token-level reward guidance for enhancing direct preference optimization. In *ICML*, 2025. 2, 3

Part I

Appendix

Table of Contents

A Limitations **14**

B Comparison with Related Token-Level Optimization Methods **14**

B.1 Comparison with ConfPO 14

B.2 Comparison with TIS-DPO 15

B.3 Comparison with Entropy-Aware RLVR Methods 15

C Mathematical Derivations **15**

C.1 Proof of Proposition 1 15

C.2 Proof of Proposition 2 16

C.3 Local KL Geometry and the Single-Sequence Log-Ratio 18

C.4 Robustness to Assumption (A2) 18

D Implementation Details and Hyperparameters **19**

E Additional Experimental Results **20**

E.1 Model-Scale Comparison on Anthropic HH-RLHF 20

E.2 Ablation of Gating Functions and the Sharpness Parameter 21

E.3 Entropy Source and Token-Level Entropy Distributions 21

E.4 Qualitative Comparison 23

A LIMITATIONS

Entropy is a proxy, not a measurement of importance. EG-DPO weights positions by the policy's uncertainty, but uncertainty and preference relevance are not the same thing. A low-entropy token can carry a decisive fact, and a high-entropy position can simply admit several equally acceptable phrasings. Proposition 2 bounds the *attainable* preference divergence at a position; it does not assert that entropy correctly ranks individual tokens by their *actual* contribution. EG-DPO can therefore under-weight an important low-entropy token or over-weight a stylistic choice. The consistent empirical gains suggest that, on average, the proxy is informative, but label-dependent signals (e.g., token-level rewards, policy–reference KL, or gradient norms) could in principle correct the cases where it is not; combining such signals with entropy is a natural extension.

Dependence on the current policy. The weights are recomputed from the policy at every step. Detaching them removes the gradient path through the entropy, but it does not protect against poorly calibrated uncertainty: if the policy's entropy is unreliable at some positions, so are their weights. We did not observe instability in practice, but we have not studied the behaviour under extreme learning rates or many epochs.

Scope of the analysis. Assumption (A2) in Proposition 2 models the chosen and rejected tokens at a position as draws from the current policy, whereas the offline pair was produced by other generators; Appendix C.4 quantifies the resulting slack but does not remove it. Proposition 1 assumes a shared signal direction across positions, which is a simplification.

Evaluation. Our generation benchmarks rely on GPT-4 as a judge and inherit its known biases; the HH-RLHF classification metric is judge-free but coarse. All reported numbers are from single training runs; we did not have the compute budget to report seed variance, which limits the strength of conclusions for the smaller margins in Table 1.

B COMPARISON WITH RELATED TOKEN-LEVEL OPTIMIZATION METHODS

This appendix expands §2 with the objectives of the two closest baselines, ConfPO and TIS-DPO, and with a discussion of entropy-aware token selection in reinforcement learning with verifiable rewards (RLVR). Although all of these methods start from the observation that tokens contribute unequally, they differ in the importance signal, the selection or weighting mechanism, and the learning paradigm.

B.1 COMPARISON WITH CONFPO

ConfPO (Yoon et al., 2025) scores each token by the probability the current policy assigns to the *realized* token,

$$\mathrm{Conf}(y_i) = \pi_\theta(y_i \mid x, y_{<i}), \tag{10}$$

and computes, separately for each response, a threshold equal to the mean confidence over that response,

$$\tau^w = \frac{1}{|y^w|}\sum_{i=1}^{|y^w|} \pi_\theta(y_i^w \mid x, y_{<i}^w), \qquad \tau^l = \frac{1}{|y^l|}\sum_{i=1}^{|y^l|} \pi_\theta(y_i^l \mid x, y_{<i}^l). \tag{11}$$

Tokens at or below the threshold are retained by binary masks $s(y_i^w) = \mathbf{1}\{\pi_\theta(y_i^w \mid x, y_{<i}^w) \le \tau^w\}$ and $s(y_i^l) = \mathbf{1}\{\pi_\theta(y_i^l \mid x, y_{<i}^l) \le \tau^l\}$, and the objective averages the policy log-probabilities of the retained tokens with a target margin γ_{ConfPO}:

$$\begin{aligned}\mathcal{L}_{\mathrm{ConfPO}} = -\mathbb{E}_{(x,y^w,y^l)\sim\mathcal{D}}\Bigg[\log\sigma\Bigg(&\frac{\beta}{|y_s^w|}\sum_i s(y_i^w)\log\pi_\theta(y_i^w \mid x, y_{<i}^w) \\ &- \frac{\beta}{|y_s^l|}\sum_i s(y_i^l)\log\pi_\theta(y_i^l \mid x, y_{<i}^l) - \gamma_{\mathrm{ConfPO}}\Bigg)\Bigg],\end{aligned} \tag{12}$$

where $|y_s^w|$ and $|y_s^l|$ are the numbers of retained tokens.

EG-DPO differs in three respects. *Signal:* ConfPO uses the probability of the single observed token, which conflates position-level uncertainty with how well the policy fits the label; EG-DPO uses the entropy of the full next-token distribution, which depends on the prefix only. *Mechanism:* ConfPO applies a hard mask and discards the high-confidence tokens entirely; EG-DPO keeps all tokens with continuous, sum-preserving weights (Eq. (7)), so it reduces to DPO exactly rather than to a subset objective. *Quantity weighted:* ConfPO averages reference-free policy log-probabilities in the style of SimPO, whereas EG-DPO weights the policy–reference log-ratios that constitute the DPO implicit reward (Eq. (8)), preserving DPO's KL-regularized interpretation.

B.2 Comparison with TIS-DPO

TIS-DPO (Liu et al., 2025) views token-level preference optimization through importance sampling and weights each token by an estimate of its reward. Because token-level rewards are unobserved, it trains two contrastive models from a common initialization π_0: π^+ on the original ordering $\mathcal{D}_{w\succ l} = \{(x, y^w, y^l)\}$ and π^- on the reversed ordering $\mathcal{D}_{l\succ w} = \{(x, y^l, y^w)\}$,

$$\pi^+ = \arg\min_\pi \mathcal{L}_{\mathrm{DPO}}(\pi; \pi_0, \mathcal{D}_{w\succ l}), \qquad \pi^- = \arg\min_\pi \mathcal{L}_{\mathrm{DPO}}(\pi; \pi_0, \mathcal{D}_{l\succ w}), \tag{13}$$

and estimates the token reward as the log-ratio of the two models on the realized token,

$$\widehat{r}_t = \log \pi^+(y_t \mid x, y_{<t}) - \log \pi^-(y_t \mid x, y_{<t}). \tag{14}$$

The estimate is converted into an importance weight $w_t = k \exp\big(\mu \,\mathrm{clamp}(\widehat{r}_t, L, U)\big)$, with $\mu > 0$ for chosen and $\mu < 0$ for rejected tokens and clipping bounds L, U to control variance.

Relative to EG-DPO, TIS-DPO requires training two auxiliary preference models before preference optimization, and its weights depend on how well those models capture the intended preference directions. Its signal is again the probability of the realized token, now under two external models, rather than the current policy's uncertainty over the vocabulary; and its weights are static during training. Its motivation is also different: TIS-DPO seeks weights that approximate an ideal distribution with equal expected token rewards, whereas EG-DPO is motivated by the concentration of the reward margin on high-entropy positions (Figure 1) and by the capacity bound of Proposition 2.

B.3 Comparison with Entropy-Aware RLVR Methods

In RLVR, algorithms such as GRPO (Shao et al., 2024) propagate a sequence-level verifiable reward uniformly to all tokens, and several works have observed that a small set of high-entropy tokens dominates effective learning. Wang et al. (2025) show that these tokens often mark reasoning forks (e.g., "wait", "however") and that restricting policy-gradient updates to the top-20% highest-entropy tokens outperforms both using all tokens and using the low-entropy 80%.

EG-DPO shares the entropy source but differs in every other respect. RLVR methods operate on on-policy rollouts with verifiable outcome rewards and policy-gradient updates, and typically apply hard selection to preserve exploration at reasoning forks. EG-DPO operates on offline chosen–rejected pairs, uses continuous detached weights on policy–reference log-ratios, and optimizes a BT margin under KL regularization. The two lines of work are complementary evidence that policy entropy localizes where learning signal resides, in two different training regimes.

C Mathematical Derivations

This appendix proves the two propositions of §5. §C.1 derives the gradient of the token-weighted DPO loss and the SNR statement of Proposition 1. §C.2 proves Proposition 2 in two ways (an event decomposition and a variance identity) and states the majorization correspondence between the Gini–Simpson index and Shannon entropy. §C.3 contrasts the pairwise statement with the local KL geometry of a single sequence, and §C.4 relaxes Assumption (A2).

C.1 Proof of Proposition 1

Setup. For a preference pair (x, y_w, y_l) write the token-weighted loss as

$$\mathcal{L} = -\log \sigma(u), \qquad u = \beta\Big(\sum_{t=1}^{T_w} w_t^w \delta_t^w - \sum_{t=1}^{T_l} w_t^l \delta_t^l\Big), \tag{15}$$

with detached weights $w_t^w, w_t^l \geq 0$. When $T_w = T_l$ this can be written $u = \beta \sum_t w_t \Delta_t$ with $\Delta_t = \delta_t^w - \delta_t^l$ if $w_t^w = w_t^l$; in general the two sums are handled separately by the response masks and none of the steps below depends on alignment. We use the aligned notation for readability.

Gradient. Since $\frac{d}{du} \log \sigma(u) = 1 - \sigma(u)$ and $\nabla_\theta w_t = 0$ by stop-gradient,

$$\begin{aligned} \nabla_\theta \mathcal{L} &= -\big(1 - \sigma(u)\big)\, \nabla_\theta u = -\beta \big(1 - \sigma(u)\big) \sum_t w_t\, \nabla_\theta \Delta_t, \\ \nabla_\theta \Delta_t &= \nabla_\theta \log \pi_\theta(y_t^w \mid \cdot) - \nabla_\theta \log \pi_\theta(y_t^l \mid \cdot), \end{aligned} \tag{16}$$

where the last equality uses that π_{ref} does not depend on θ. This is the DPO gradient with per-token coefficients w_t, as claimed in §4.4.

SNR. Decompose $\nabla_\theta \Delta_t = g_t + \xi_t$ with $\mathbb{E}[\xi_t] = 0$, $\mathrm{Var}(\xi_t) = \Sigma_t$, and assume the noise terms are uncorrelated across positions. Then $\mathbb{E}\| \sum_t w_t \xi_t \|^2 = \sum_t w_t^2 \,\mathrm{tr}(\Sigma_t)$. Under the common-direction assumption $g_t = a_t \bar{g}$,

$$\mathrm{SNR}(w) = \frac{\| \sum_t w_t g_t \|^2}{\mathbb{E}\| \sum_t w_t \xi_t \|^2} = \frac{\big(\sum_t w_t a_t\big)^2}{\sum_t w_t^2 s_t}, \qquad s_t = \mathrm{tr}(\Sigma_t) > 0. \tag{17}$$

By the Cauchy–Schwarz inequality applied to the vectors $(w_t \sqrt{s_t})_t$ and $(a_t / \sqrt{s_t})_t$,

$$\Big(\sum_t w_t a_t \Big)^2 = \Big(\sum_t (w_t \sqrt{s_t}) \tfrac{a_t}{\sqrt{s_t}} \Big)^2 \leq \Big(\sum_t w_t^2 s_t \Big) \Big(\sum_t \tfrac{a_t^2}{s_t} \Big), \tag{18}$$

with equality if and only if $w_t \sqrt{s_t} \propto a_t / \sqrt{s_t}$, i.e., $w_t \propto a_t / s_t$. Hence $\mathrm{SNR}(w) \leq \sum_t a_t^2 / s_t$ with equality exactly at $w_t \propto a_t / s_t$; the normalization $\sum_t w_t = N$ fixes the constant. Uniform weights attain the maximum only if a_t / s_t is constant.

Local improvement. For the transfer statement, let $r_t = a_t / s_t$ and consider moving mass $\epsilon > 0$ from position j to position i with $r_i > r_j$, keeping $\sum_t w_t$ fixed. Writing $A = \sum_t w_t a_t$ and $B = \sum_t w_t^2 s_t$, a first-order expansion gives

$$\frac{d}{d\epsilon} \log \mathrm{SNR} \Big|_{\epsilon = 0} = \frac{2(a_i - a_j)}{A} - \frac{2(w_i s_i - w_j s_j)}{B}. \tag{19}$$

At uniform weights $w_t \equiv 1$ this equals $2\big[(a_i - a_j)/A - (s_i - s_j)/B\big]$, which is positive whenever a_i / s_i exceeds a_j / s_j by a sufficient margin relative to the aggregate ratio A/B; in particular it is positive for any transfer from a position with $a_j = 0$ (pure noise) to a position with $a_i > 0$. This is the sense in which "moving weight from noise-dominated to signal-carrying positions improves the SNR." □

Remark 1. The common-direction assumption is a simplification; without it, $\| \sum_t w_t g_t \|^2$ contains cross terms $\langle g_t, g_{t'} \rangle$ and the optimal w depends on their signs. The qualitative conclusion, that uniform weights are optimal only under a non-generic balance between signal and noise across positions, does not depend on this simplification.

C.2 Proof of Proposition 2

Setup and assumptions. Fix a prefix context c and a policy iterate θ, and let $p = \pi_\theta(\cdot \mid c) \in \Delta^{|\mathcal{V}|-1}$.

Assumption 1 (Bounded local deviation, A1)**.** *There exists $M > 0$ such that $|\log \pi_\theta(v \mid c) - \log \pi_{\text{ref}}(v \mid c)| \leq M$ for all $v \in \mathcal{V}$.*

This is the token-level counterpart of the KL regularization in DPO: an unbounded token-wise log-ratio at some position would allow $D_{\mathrm{KL}}(\pi_\theta \| \pi_{\text{ref}})$ to be arbitrarily large, contrary to the finite penalty in Eq. (1).

Assumption 2 (Local sampling approximation, A2)**.** *At this position, the chosen token y_t^w and the rejected token y_t^l are independent draws from p.*

Remark 2 (On A2). A2 is an analytical surrogate rather than a claim about how the offline data were generated. Two simplifications are involved. First, the chosen and rejected responses have different prefixes $y^w_{<t}$ and $y^l_{<t}$, so "the same context c" is exact only when the prefixes coincide (e.g., at the first response token or after shared prefixes) and approximate otherwise; the proposition should be read as a statement about the divergence that a position of a given uncertainty level can express. Second, the tokens were produced by the generators of the dataset, not by π_θ; since DPO is initialized from the SFT model and regularized toward it, the distributions are related but not equal. §C.4 relaxes A2 to bounded total-variation shift.

Statement. Under A1–A2, with $\Delta_t = \delta^w_t - \delta^l_t$,

$$\mathbb{E}\big[\Delta_t^2 \mid c\big] \;\le\; 4M^2\big(1 - \|p\|_2^2\big). \tag{20}$$

Proof (event decomposition). All randomness in (δ^w_t, δ^l_t) comes from the pair (y^w_t, y^l_t). If $y^w_t = y^l_t$, then $\delta^w_t = \delta^l_t$ and $\Delta_t = 0$. If $y^w_t \neq y^l_t$, then by A1 $|\delta^w_t|, |\delta^l_t| \le M$, so $|\Delta_t| \le 2M$. Hence $\Delta_t^2 \le 4M^2\,\mathbf{1}[y^w_t \neq y^l_t]$ pointwise, and taking expectations under A2,

$$\mathbb{E}[\Delta_t^2 \mid c] \le 4M^2 \Pr[y^w_t \neq y^l_t] = 4M^2\Big(1 - \sum_v p(v)^2\Big). \tag{21}$$

□

Proof (variance identity). Let $f(v) = \log \pi_\theta(v \mid c) - \log \pi_{\text{ref}}(v \mid c)$, so $\delta^w_t = f(y^w_t)$ and $\delta^l_t = f(y^l_t)$. For i.i.d. $y^w_t, y^l_t \sim p$,

$$\mathbb{E}[\Delta_t^2 \mid c] = \mathbb{E}\big[(f(y^w_t) - f(y^l_t))^2\big] = 2\,\mathrm{Var}_{y\sim p}[f(y)]. \tag{22}$$

The variance of a function of a discrete random variable admits the pairwise representation

$$\mathrm{Var}_p[f] = \frac{1}{2}\sum_{v,v'\in\mathcal{V}} p(v)p(v')\big(f(v) - f(v')\big)^2 = \frac{1}{2}\sum_{v\neq v'} p(v)p(v')\big(f(v) - f(v')\big)^2. \tag{23}$$

By A1, $|f(v) - f(v')| \le 2M$ for all v, v', so

$$\mathrm{Var}_p[f] \le \frac{1}{2}(2M)^2 \sum_{v\neq v'} p(v)p(v') = 2M^2\big(1 - \|p\|_2^2\big), \tag{24}$$

and substituting into Eq. (22) gives Eq. (20). The two proofs yield the same constant; the second makes explicit that $\mathbb{E}[\Delta_t^2]$ is exactly twice the variance of the log-ratio under the current policy, which is the quantity that Figure 4 visualizes empirically. □

Monotonic correspondence with Shannon entropy. Let $\mathcal{D}(p) = 1 - \|p\|_2^2$ (Gini–Simpson diversity) and $H(p) = -\sum_v p(v)\log p(v)$. Both are symmetric functions of the components of p, and both are Schur-concave: if p' is obtained from p by a Robin Hood transfer (moving mass from a larger component to a smaller one without reversing their order), then $\mathcal{D}(p') \ge \mathcal{D}(p)$ and $H(p') \ge H(p)$. Consequently, whenever p majorizes p' (i.e., p is more concentrated), $\mathcal{D}(p) \le \mathcal{D}(p')$ and $H(p) \le H(p')$. On a fixed simplex both attain their minimum 0 exactly at one-hot distributions and their maximum ($1 - 1/|\mathcal{V}|$ and $\log|\mathcal{V}|$, respectively) exactly at the uniform distribution. Majorization is a partial order, so for two arbitrary distributions p_1, p_2 the two functions are guaranteed to agree in ordering only when p_1 and p_2 are comparable; in general they are rank-correlated but not functionally related. This is sufficient for our purpose: the bound in Eq. (20) is small precisely where H is small, and both are zero exactly where the policy is deterministic.

Corollary 1 (Validity of H_t as a weighting signal)**.** *At a fixed policy iterate, positions where the current policy is nearly deterministic have $\mathbb{E}[\Delta_t^2 \mid c]$ suppressed toward zero, whereas positions with high policy entropy admit a non-zero pairwise preference signal of magnitude up to $2M$. Combined with Proposition 1, this motivates weights that increase monotonically with the detached H_t: such weights move mass from positions with small attainable signal to positions with large attainable signal while preserving the first-order gradient form of Eq. (16) within each update.*

What the proposition does not say. Eq. (20) bounds the *pairwise difference* Δ_t, not the single-sequence magnitudes $|\delta^w_t|$ or $|\delta^l_t|$. The attainable size of $|\delta_t|$ on one sequence is governed by the reference probability of the realized token, not by policy entropy (§C.3). Since the DPO loss is driven by Δ_t, the pairwise statement is the relevant one.

C.3 Local KL Geometry and the Single-Sequence Log-Ratio

This section is a geometric supplement to Proposition 2; it is not part of the main argument. It characterizes how large a single sequence's $|\delta_t|$ can be under a position-wise KL budget, and shows that this quantity depends on the *reference* probability of the realized token rather than on policy entropy.

Notation. Let $r = \pi_{\text{ref}}(\cdot \mid c)$ and $q = \pi_\theta(\cdot \mid c) = r + \Delta$ with $\sum_v \Delta(v) = 0$. (Here r is the reference distribution, distinct from the policy distribution p of §C.2.)

Lemma 1 (Second-order expansion of KL)**.** *For q in a neighbourhood of r,*

$$\mathrm{KL}(q\|r) = \tfrac{1}{2}\sum_v \frac{\Delta(v)^2}{r(v)} + \mathcal{O}(\|\Delta\|^3), \tag{25}$$

and the Hessian $\mathrm{diag}(1/r(v))$ is the Fisher information of the categorical distribution r.

Lemma 2 (Single-token log-ratio)**.** *If v^* is the realized token, then $\delta_t = \log\big(1 + \Delta(v^*)/r(v^*)\big) \approx \Delta(v^*)/r(v^*)$ when $|\Delta(v^*)| \ll r(v^*)$.*

Proposition 3 (Local bound on $|\delta_t|$)**.** *Under a position-wise budget $\mathrm{KL}(q\|r) \le \epsilon$ and neglecting higher-order terms,*

$$|\delta_t| \le \sqrt{\frac{2\epsilon\,(1 - r(v^*))}{r(v^*)}}. \tag{26}$$

Proof. Maximize $|\Delta(v^*)|/r(v^*)$ subject to $\frac{1}{2}\sum_v \Delta(v)^2/r(v) \le \epsilon$ and $\sum_v \Delta(v) = 0$. By the method of Lagrange multipliers, the extremum places $\Delta(v^*) = \alpha$ on the realized token and distributes $-\alpha$ over the remaining tokens in proportion to $r(v)$, i.e., $\Delta(v) = -\alpha\, r(v)/(1 - r(v^*))$ for $v \ne v^*$. Substituting into the constraint gives $\frac{1}{2}\alpha^2\big(1/r(v^*) + 1/(1 - r(v^*))\big) = \epsilon$, whence $\alpha^2 = 2\epsilon\, r(v^*)(1 - r(v^*))$ and $|\delta_t| \approx |\alpha|/r(v^*)$ yields Eq. (26). □

Discussion. Eq. (26) shows that the reachable radius of $|\delta_t|$ on a single sequence is controlled by $r(v^*)$: it vanishes when the realized token is the reference mode and can be large when the realized token is in the reference tail. Policy entropy does not appear. This is why analyzing $|\delta_t|$ alone would not justify entropy weighting, and why Proposition 2 instead analyzes the pairwise second moment $\mathbb{E}[\Delta_t^2]$ under the current policy. Geometrically, the locally traversable distance between π_θ and π_{ref} is set by the Fisher geometry of the reference; the reference enters EG-DPO only through the log-ratio δ_t, while the weighting signal is the policy's entropy.

C.4 Robustness to Assumption (A2)

Suppose the chosen and rejected tokens at context c have marginals p_w and p_l that need not equal p, with $\mathrm{TV}(p_w, p) \le \eta_w$ and $\mathrm{TV}(p_l, p) \le \eta_l$, where $\mathrm{TV}(a, b) = \frac{1}{2}\|a - b\|_1$, and suppose they are independent. The event decomposition gives

$$\Pr[y_t^w \ne y_t^l] = 1 - \sum_v p_w(v)\, p_l(v). \tag{27}$$

Writing $\sum_v p_w(v)p_l(v) - \sum_v p(v)^2 = \sum_v \big(p_w(v) - p(v)\big)p_l(v) + \sum_v p(v)\big(p_l(v) - p(v)\big)$ and bounding each term by Hölder's inequality with $\|p_l\|_\infty, \|p\|_\infty \le 1$,

$$\sum_v p_w(v)p_l(v) \;\ge\; \|p\|_2^2 - \|p_w - p\|_1 - \|p_l - p\|_1 \;=\; \|p\|_2^2 - 2(\eta_w + \eta_l). \tag{28}$$

Therefore

$$\mathbb{E}[\Delta_t^2 \mid c] \le 4M^2\big(1 - \|p\|_2^2 + 2(\eta_w + \eta_l)\big). \tag{29}$$

Distribution shift between the current policy and the data-generating marginals thus enters as an additive slack proportional to $\eta_w + \eta_l$. The entropy-based conclusion is exact under A2 and degrades gracefully with the shift; it should be understood as local to the current iterate and approximate when the shift is large. □

D Implementation Details and Hyperparameters

Implementation. EG-DPO is implemented on top of the `DPOTrainer` of the HuggingFace TRL library by overriding `compute_loss` and `get_batch_loss_metrics`; the remainder of the pipeline is unchanged. The following pseudocode specifies the objective of Eq. (8).

```
# (1) Policy forward returns target log-probabilities and entropy H_t
pol_c, H_c = forward_logps_and_entropy(policy, batch["chosen"])
pol_r, H_r = forward_logps_and_entropy(policy, batch["rejected"])
H_c, H_r = H_c.detach(), H_r.detach()

# (2) Reference forward supplies target log-probabilities only
with torch.no_grad():
    ref_c = forward_logps(ref, batch["chosen"])
    ref_r = forward_logps(ref, batch["rejected"])

# (3) Retain token-level log-ratios without sequence-level summation
delta_c, delta_r = pol_c - ref_c, pol_r - ref_r
c_mask = batch["chosen"]["response_mask"]
r_mask = batch["rejected"]["response_mask"]

# (4) Z-score -> sigmoid -> rescale weights to valid-token count
# entropy_weights implements Eqs. (zscore)-(rescale), per response.
omega_c = entropy_weights(H_c, c_mask, gamma).detach()
omega_r = entropy_weights(H_r, r_mask, gamma).detach()
r_w = beta * (omega_c * delta_c * c_mask).sum(dim=-1)
r_l = beta * (omega_r * delta_r * r_mask).sum(dim=-1)

# Average over preference pairs, not over response tokens.
loss = -F.logsigmoid(r_w - r_l).mean()
```

`forward_logps_and_entropy` returns the target-token log-probabilities and the position-wise vocabulary entropy from a single policy forward pass; the entropy is computed in `float32` for numerical stability. Both the entropies and the resulting weights are detached, so no gradient flows through the entropy-to-weight path. `entropy_weights` implements Eqs. (5)–(7). The policy is fine-tuned with LoRA and the reference π_{ref} is obtained from the same base model with adapters disabled (`model.disable_adapter()`), so no separate reference-model copy is required. Computing vocabulary entropy and token weights still incurs additional operations and temporary storage.

Training setup. All experiments use 4×NVIDIA H800 (80 GB) GPUs with bfloat16 precision, FlashAttention-2, and DeepSpeed ZeRO Stage 2 (bucket size 2×10^8, `overlap_comm` and `contiguous_gradients` enabled). The effective global batch size is $8 \times 2 \times 4 = 64$. Table 3 lists all hyperparameters. EG-DPO adds only γ (and the `detach_weights` flag, always `True`) to the DPO configuration. For the baselines (IPO, CPO, KTO, ORPO, R-DPO, SimPO, TDPO, TIS-DPO, ConfPO) we use the method-specific hyperparameters recommended in the respective papers (e.g., β, target margins, SFT-mixing coefficients) rather than forcing DPO's settings; the shared factors held fixed across all methods are the SFT initialization and reference model, the LoRA configuration, the training data, the global batch size, the number of epochs, and the evaluation protocol.

Computational overhead. Relative to LoRA-DPO, EG-DPO adds (i) one entropy reduction over the vocabulary dimension of logits already produced by the policy forward pass, and (ii) $\mathcal{O}(BT)$ scalar operations for standardization, gating, and rescaling. It requires no additional model forward passes or learnable parameters. Entropy computation and weight construction nevertheless add computational work and temporary storage; no claim of identical runtime or peak memory is implied.

Table 3: EG-DPO training hyperparameters. LoRA adapters are injected into the seven linear layers of each Transformer block (attention q/k/v/o and FFN gate/up/down); base parameters are frozen. Global batch size = micro batch (8) × gradient accumulation (2) × GPUs (4).

Category	Hyperparameter	Value
EG-DPO	γ (gating sharpness)	3.0
	`detach_weights`	`True`
DPO	β (KL penalty coefficient)	0.3
	Max sequence length	1024
Optimization	Optimizer	AdamW
	Learning rate	5×10^{-6}
	LR scheduler	Cosine
	Warmup ratio	0.1
	Epochs	1
	Per-device micro batch	8
	Gradient accumulation	2
	Global batch size	**64**
LoRA	Rank r	32
	α	64
	Dropout	0.05
	Bias	none
	Task type	`CAUSAL_LM`
	Target modules	`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`
System	GPU	4×H800 80GB
	Precision	bfloat16
	Attention implementation	FlashAttention-2
	Distributed strategy	DeepSpeed ZeRO-2
	Bucket size	2×10^8

Table 4: Preference classification accuracy (%) on the official Anthropic HH-RLHF test set (4,666 pairs), using the unweighted policy–reference implicit-reward margin, for Qwen2.5-Instruct at four scales. Best and second-best per column are bold and underlined.

Method	1.5B	3B	7B	14B	Avg.
DPO	56.901	59.087	58.766	59.173	58.482
IPO	**57.694**	<u>59.216</u>	<u>58.980</u>	58.594	<u>58.621</u>
CPO	49.807	50.729	49.314	49.357	49.802
SimPO	52.679	53.322	51.800	51.865	52.416
TDPO	57.051	58.701	58.080	<u>59.601</u>	58.358
TIS-DPO	56.815	57.565	58.080	57.630	57.523
EG-DPO	<u>57.501</u>	**59.430**	**59.816**	**60.435**	**59.295**

E Additional Experimental Results

E.1 Model-Scale Comparison on Anthropic HH-RLHF

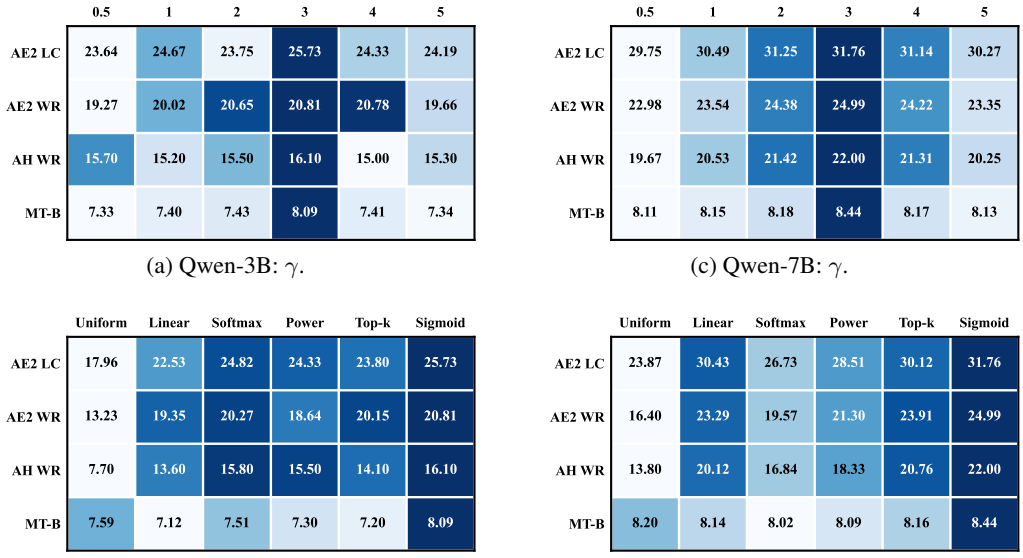
Table 4 reports preference classification accuracy over all 4,666 test pairs. EG-DPO ranks first at 3B, 7B, and 14B and second at 1.5B (behind IPO), giving the best average accuracy. Its gains over DPO are $+0.600$, $+0.343$, $+1.050$, and $+1.262$ points at 1.5B, 3B, 7B, and 14B, respectively; the gain is not monotone in scale but is largest at 14B. CPO and SimPO, which remove or replace the reference model, score close to chance under this reference-based criterion, as expected. Overall,

the results indicate that entropy-guided weighting during training improves the ranking quality of the *unweighted* implicit reward at test time, and that the benefit persists as model capacity increases.

E.2 Ablation of Gating Functions and the Sharpness Parameter

We ablate the gating function and the sharpness γ on Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct trained on UltraFeedback (Figure 6). The upper panel of each pair varies $\gamma \in \{0.5, 1, 2, 3, 4, 5\}$ with sigmoid gating; the lower panel compares six entropy-to-weight maps at their best settings: Uniform (DPO), Linear ($\tilde{\omega}_t \propto H_t$), Softmax over z_t, Power ($\tilde{\omega}_t \propto H_t^\kappa$), Top-$k$ hard selection, and Sigmoid (Eq. (6)).

For both models, $\gamma = 3$ is best on all four metrics, with performance degrading smoothly on either side; very small γ approaches uniform weights, and very large γ approaches a hard split that discards the ranking information among below-average positions. Sigmoid gating is best on all four metrics for both models, ahead of Top-k and Softmax. The ordering is stable from 3B to 7B, so we fix $\gamma = 3$ and sigmoid gating in all other experiments without per-model tuning.



(a) Qwen-3B: γ.

(c) Qwen-7B: γ.

(b) Qwen-3B: gating.

(d) Qwen-7B: gating.

Figure 6: Gating-function and sharpness ablations on Qwen2.5-3B-Instruct (a,b) and Qwen2.5-7B-Instruct (c,d). Upper panels vary γ with sigmoid gating; lower panels compare gating functions. Rows: AlpacaEval 2 LC (AE2 LC), AlpacaEval 2 WR (AE2 WR), Arena-Hard WR (AH WR), MT-Bench (MT-B). Cells show absolute scores; darker blue is higher within each row.

E.3 Entropy Source and Token-Level Entropy Distributions

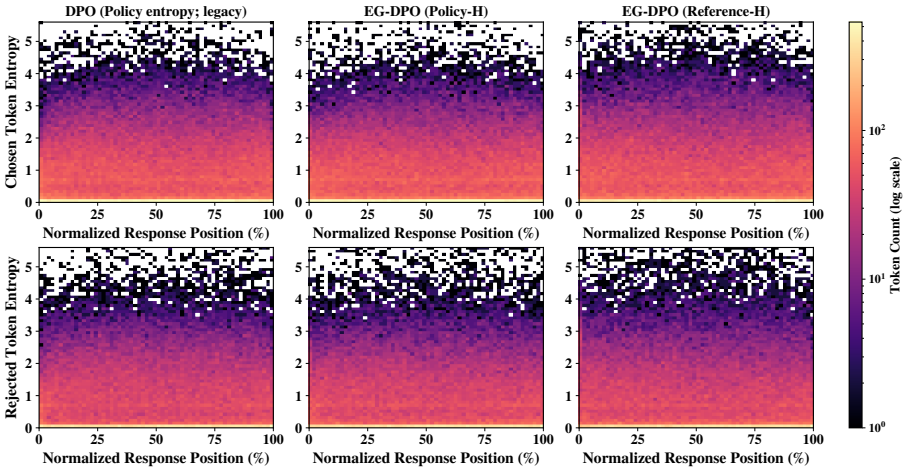
Entropy source. Table 5 compares reference-model and current-policy entropy as the weighting signal on four Instruct models. Policy entropy is better on every metric for every model. The Policy rows for Llama-3-8B and Mistral-7B are the EG-DPO rows of Table 1.

Token-level distributions. Figures 7–9 complement the sequence-level curves of Figure 3 with token-level distributions on a shared set of UltraFeedback preference pairs. In each heatmap the horizontal axis is the token's normalized position within its response, the vertical axis is predictive entropy, and colour is token count on a logarithmic scale; rows are chosen (top) and rejected (bottom) responses. The three columns show the final DPO policy's entropy, the final EG-DPO (Policy-H) policy's entropy, and the frozen reference model's entropy that EG-DPO (Reference-H) uses as its weighting signal. Two observations are relevant to §6.4. First, the EG-DPO policy's entropy distribution differs visibly from the reference's, particularly in the upper tail, so the ranking of positions used for weighting changes over training. Second, the high-entropy tail is spread across

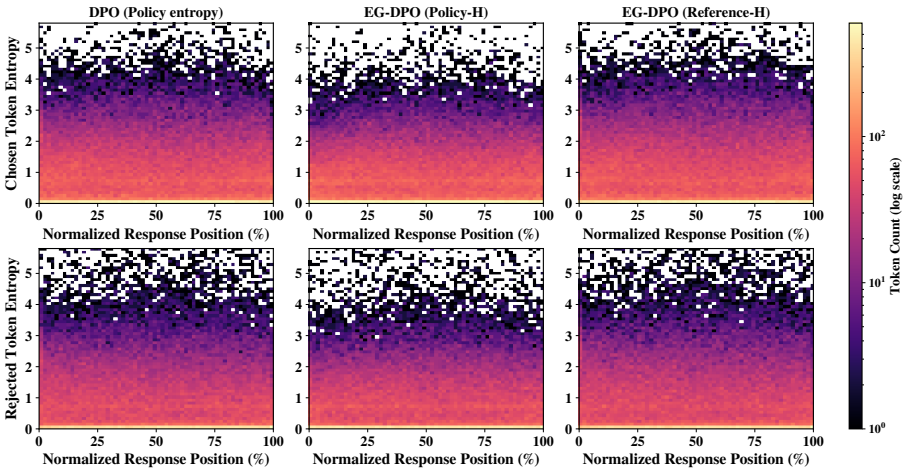
Table 5: Entropy-source comparison on Instruct checkpoints trained on UltraFeedback. The better result per model and metric is bold.

Model	Entropy source	AlpacaEval 2		Arena-Hard	MT-Bench
		LC (%)	WR (%)	WR (%)	Score
Qwen2.5-3B	Reference	20.86	17.95	12.12	7.381
	Policy	**25.73**	**20.81**	**16.10**	**8.088**
Qwen2.5-7B	Reference	28.26	22.47	19.11	8.050
	Policy	**31.76**	**24.99**	**22.00**	**8.444**
Mistral-7B	Reference	19.79	18.14	6.86	7.438
	Policy	**28.10**	**34.60**	**25.70**	**7.735**
Llama-3-8B	Reference	36.41	36.27	26.73	7.775
	Policy	**38.60**	**36.80**	**29.30**	**7.925**

positions rather than concentrated at the start or end of responses, which is why a position-based heuristic would not substitute for an entropy-based one.



(a) Qwen2.5-3B-Instruct.



(b) Qwen2.5-7B-Instruct.

Figure 7: Token-level entropy distributions for the Qwen2.5-Instruct family. The Qwen2.5-3B DPO checkpoint uses a different initialization from its two EG-DPO counterparts and is included for qualitative reference only; the Qwen2.5-7B comparison is initialization-matched.

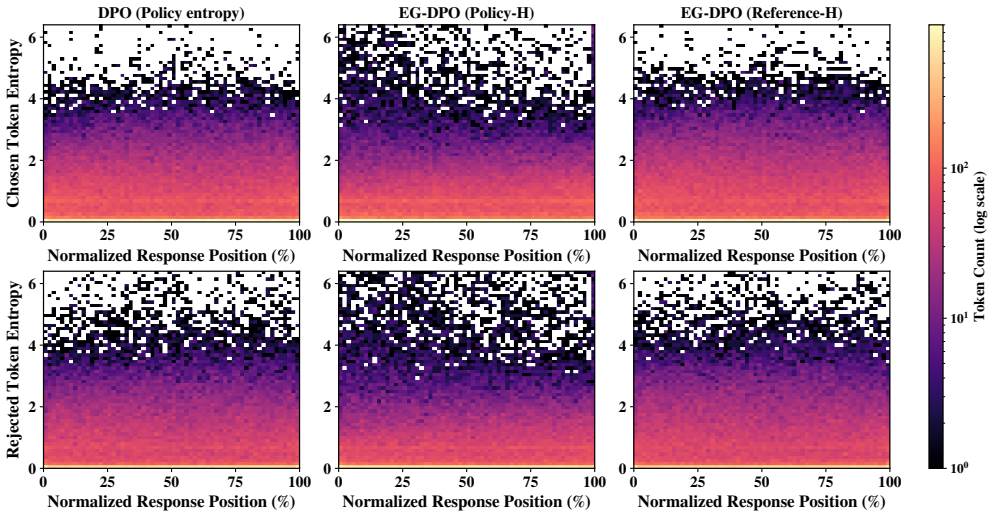


Figure 8: Token-level entropy distributions for Mistral-7B-Instruct. All three runs share the SFT initialization and UltraFeedback preference data.

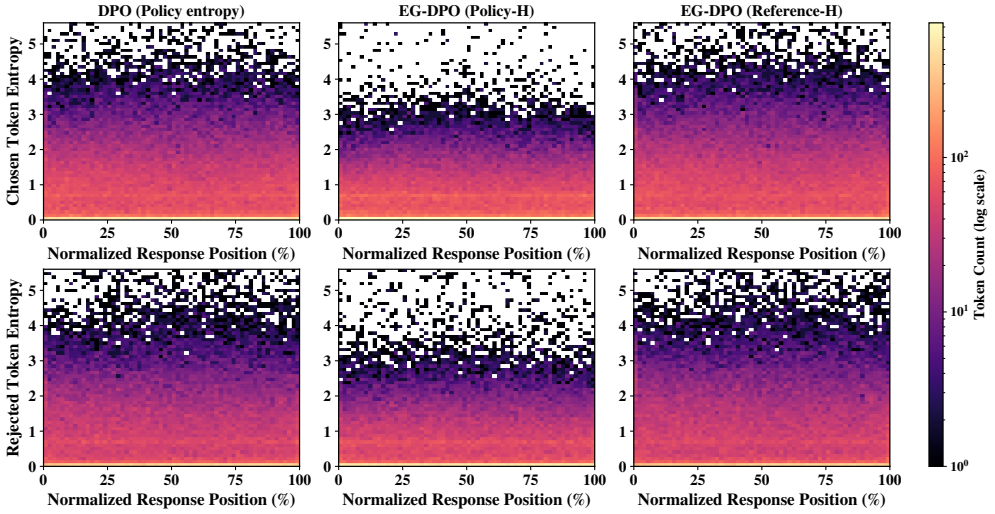


Figure 9: Token-level entropy distributions for Llama-3-8B-Instruct. All three runs share the SFT initialization and UltraFeedback preference data.

E.4 Qualitative Comparison

All outputs below are produced by Qwen2.5-7B-Instruct or its SFT and preference-optimized checkpoints (Base, SFT, DPO, IPO, CPO, KTO, SimPO, TDPO, TIS-DPO, ConfPO, EG-DPO). We sample two prompts uniformly at random (seed 2027) from the 500 Arena-Hard prompt identifiers shared by all saved answer files and reproduce each stored response in full; the sampling uses neither response content nor evaluation scores. The examples illustrate differences in style and reasoning and are not a substitute for the aggregate results.

E.4.1 Example 1

Prompt

```
If I can walk 1700 steps every 15 min, how long would it take me to hit 40k steps?
```

Reference checkpoints

Base

To determine how long it will take you to hit 40,000 steps if you walk 1,700 steps every 15 minutes, we can follow these steps:
1. **Calculate the number of 15-minute intervals needed to reach 40,000 steps:**

$$\text{Number of intervals} = \frac{40,000 \text{ steps}}{1,700 \text{ steps/interval}}$$

$$\text{Number of intervals} \approx 23.53 \text{ intervals}$$

2. **Convert the number of intervals into time:** Since each interval is 15 minutes:

$$\text{Total time} = 23.53 \text{ intervals} \times 15 \text{ minutes/interval}$$

$$\text{Total time} \approx 352.95 \text{ minutes}$$

3. **Convert the total time into hours and minutes:**

$$352.95 \text{ minutes} = 5 \text{ hours and } 52.95 \text{ minutes}$$

(since $352.95 \div 60 \approx 5.8825$, which is 5 hours and 52.95 minutes)
So, it would take you approximately 5 hours and 53 minutes to hit 40,000 steps.

SFT

To calculate the time it would take you to reach 40,000 steps at a pace of 1,700 steps per 15 minutes, we can follow these steps:
1. First, determine the number of 15-minute intervals needed to reach 40,000 steps:

$$\text{Number of intervals} = \frac{\text{Total steps}}{\text{Steps per interval}} = \frac{40,000}{1,700}$$

2. Calculate the result:

$$\text{Number of intervals} = \frac{40,000}{1,700} \approx 23.53$$

3. Since you cannot have a fraction of an interval, round up to the nearest whole number:

$$\text{Number of intervals} = 24$$

4. Now, multiply the number of intervals by the duration of each interval (15 minutes) to find the total time:

$$\text{Total time} = \text{Number of intervals} \times \text{Duration per interval} = 24 \times 15 \text{ minutes}$$

5. Calculate the result:

$$\text{Total time} = 24 \times 15 = 360 \text{ minutes}$$

So, it would take approximately 360 minutes (or 6 hours) to reach 40,000 steps at your current pace.

Preference-optimization baselines

DPO

To determine how long it will take you to hit 40,000 steps if you walk 1,700 steps every 15 minutes, we can follow these steps:

1. **Calculate the number of 15-minute intervals needed to reach 40,000 steps:**

$$\text{Number of intervals} = \frac{40,000 \text{ steps}}{1,700 \text{ steps/interval}}$$

$$\text{Number of intervals} \approx 23.53 \text{ intervals}$$

2. **Convert the number of intervals into time:** Since each interval is 15 minutes:

$$\text{Total time} = 23.53 \text{ intervals} \times 15 \text{ minutes/interval}$$

$$\text{Total time} \approx 352.95 \text{ minutes}$$

3. **Convert the total time into hours and minutes (if desired):**

$$352.95 \text{ minutes} = 5 \text{ hours and } 52.95 \text{ minutes}$$

$$52.95 \text{ minutes} \approx 53 \text{ minutes}$$

So, it would take you approximately 5 hours and 53 minutes to reach 40,000 steps.
Alternatively, you can round up to the nearest whole interval for practical purposes:
- Rounding up 23.53 to 24 intervals:

$$24 \text{ intervals} \times 15 \text{ minutes/interval} = 360 \text{ minutes}$$

$$360 \text{ minutes} = 6 \text{ hours}$$

Therefore, it would take you approximately 6 hours to reach 40,000 steps.

IPO

To determine how long it will take you to hit 40,000 steps if you walk 1,700 steps every 15 minutes, we can follow these steps:

1. **Calculate the number of 15-minute intervals needed to reach 40,000 steps:**

$$\text{Number of intervals} = \frac{40,000 \text{ steps}}{1,700 \text{ steps/interval}}$$

$$\text{Number of intervals} \approx 23.53 \text{ intervals}$$

2. **Convert the number of intervals into time:** Since each interval is 15 minutes:

$$\text{Total time} = 23.53 \text{ intervals} \times 15 \text{ minutes/interval}$$

$$\text{Total time} \approx 352.95 \text{ minutes}$$

3. **Convert the total time into hours and minutes:**

$$352.95 \text{ minutes} = 5 \text{ hours and } 52.95 \text{ minutes}$$

(since $352.95 \div 60 \approx 5.8825$, which is 5 hours and 52.95 minutes)
So, it would take approximately 5 hours and 53 minutes to reach 40,000 steps.

CPO

To calculate the time it would take you to reach 40,000 steps, we need to first determine your walking speed in steps per minute.
You mentioned that you can walk 1700 steps every 15 minutes. To find your walking speed in steps per minute, divide the number of steps by the number of minutes:
1700 steps / 15 minutes = 113.33 steps per minute
Now, to find out how many minutes it would take you to reach 40,000 steps, divide the total number of steps by your walking speed:
40,000 steps / 113.33 steps per minute ≈ 352.86 minutes
To convert this into hours and minutes, divide the total minutes by 60 (since there are 60 minutes in an hour):
352.86 minutes ÷ 60 ≈ 5.88 hours
So, it would take approximately 5.88 hours or about 5 hours and 53 minutes for you to reach 40,000 steps at your current walking pace.

KTO

To calculate the time it would take you to hit 40,000 steps if you can walk 1,700 steps every 15 minutes, we can follow these steps:
1. First, determine how many sets of 1,700 steps are in 40,000 steps:

$$\frac{40,000}{1,700} \approx 23.53$$

This means you need to complete approximately 23 full sets of 1,700 steps and then a bit more.
2. Since each set takes 15 minutes, multiply the number of sets by 15 minutes:

$$23 \times 15 = 345 \text{ minutes}$$

3. To find out the remaining steps after completing 23 sets, subtract the total steps covered in 23 sets from 40,000:

$$23 \times 1,700 = 39,100$$

$$40,000 - 39,100 = 900 \text{ steps}$$

4. Calculate the time needed to cover the remaining 900 steps at your rate of 1,700 steps per 15 minutes:

$$\frac{900}{1,700} \approx 0.529 \text{ sets}$$

Since each set is 15 minutes, the time for the remaining steps is:

$$0.529 \times 15 \approx 7.935 \text{ minutes}$$

5. Add the time for the full sets and the remaining steps:

$$345 + 7.935 \approx 352.935 \text{ minutes}$$

So, it would take you approximately 353 minutes (or about 5 hours and 53 minutes) to reach 40,000 steps.

SimPO

To determine how long it will take you to reach 40,000 steps if you walk 1,700 steps every 15 minutes, follow these steps:
1. **Calculate the number of 15-minute intervals needed:** - Divide the total number of steps (40,000) by the number of steps per interval (1,700).

$$\frac{40,000}{1,700} \approx 23.53$$

This means you need approximately 23.53 intervals.
2. **Convert intervals into time:** - Each interval is 15 minutes. - Multiply the number of intervals by 15 minutes.

$$23.53 \times 15 \approx 352.95 \text{ minutes}$$

3. **Convert minutes into hours and minutes:** - Divide the total minutes by 60 to convert to hours.

$$\frac{352.95}{60} \approx 5.88 \text{ hours}$$

- Convert the decimal part back to minutes:

$$0.88 \times 60 \approx 52.8 \text{ minutes}$$

So, it would take you approximately 5 hours and 53 minutes to reach 40,000 steps.
Summary: You would need about 5 hours and 53 minutes to hit 40,000 steps at your current pace.

Token-aware methods

TDPO

To determine how long it will take you to hit 40,000 steps if you walk 1,700 steps every 15 minutes, we can follow these steps:
1. **Calculate the number of 15-minute intervals needed to reach 40,000 steps:**

$$\text{Number of intervals} = \frac{40,000 \text{ steps}}{1,700 \text{ steps/interval}}$$

$$\text{Number of intervals} \approx 23.53 \text{ intervals}$$

2. **Convert the number of intervals into time:** Since each interval is 15 minutes:

$$\text{Total time} = 23.53 \text{ intervals} \times 15 \text{ minutes/interval}$$

$$\text{Total time} \approx 352.95 \text{ minutes}$$

3. **Convert the total time into hours and minutes:**

$$352.95 \text{ minutes} = 5 \text{ hours and } 52.95 \text{ minutes}$$

(since $352.95 \div 60 \approx 5.8825$, which is 5 hours and 52.95 minutes)
So, it would take approximately 5 hours and 53 minutes to reach 40,000 steps.

TIS-DPO

To determine how long it will take you to hit 40,000 steps if you walk 1,700 steps every 15 minutes, you can follow these steps:
1. Calculate the number of 15-minute intervals needed to reach 40,000 steps.
2. Convert the total number of 15-minute intervals into hours and minutes.
First, find out how many 15-minute intervals are required to reach 40,000 steps:

$$\text{Number of intervals} = \frac{40,000 \text{ steps}}{1,700 \text{ steps/interval}} \approx 23.53 \text{ intervals}$$

Since you can't have a fraction of an interval in this context, round up to the next whole number because you need to complete the full interval to reach or exceed 40,000 steps:

$$\text{Number of intervals} = 24$$

Next, convert the number of intervals into time:

$$24 \text{ intervals} \times 15 \text{ minutes/interval} = 360 \text{ minutes}$$

Finally, convert 360 minutes into hours and minutes:

$$360 \text{ minutes} = 6 \text{ hours}$$

So, it would take you approximately 6 hours to hit 40,000 steps if you maintain your current walking pace.

ConfPO

To determine how long it will take you to hit 40,000 steps if you walk 1,700 steps every 15 minutes, we can follow these steps:
1. **Calculate the number of 15-minute intervals needed to reach 40,000 steps:**

$$\text{Number of intervals} = \frac{40,000 \text{ steps}}{1,700 \text{ steps/interval}}$$

$$\text{Number of intervals} \approx 23.53 \text{ intervals}$$

2. **Convert the number of intervals into time:** Since each interval is 15 minutes:

$$\text{Total time} = 23.53 \text{ intervals} \times 15 \text{ minutes/interval}$$

$$\text{Total time} \approx 352.95 \text{ minutes}$$

3. **Convert the total time into hours and minutes:**

$$352.95 \text{ minutes} = 5 \text{ hours and } 52.95 \text{ minutes}$$

(since $352.95 \div 60 \approx 5.8825$, which is 5 hours and 52.95 minutes)
So, it would take approximately 5 hours and 53 minutes to reach 40,000 steps.

EG-DPO

To calculate the time needed to reach 40,000 steps, we can first determine the steps per minute and then find the total time.
1. You walk 1700 steps in 15 minutes. So, steps per minute:
(1700 steps) / (15 minutes) = ≈ 113.33 steps/minute
2. Now, to reach 40,000 steps:
(40,000 steps) / (113.33 steps/minute) ≈ 353.6 minutes
To convert this into hours and minutes:
353.6 minutes ÷ 60 minutes/hour ≈ 5.89 hours
So, it would take approximately 5 hours and 53 minutes (rounding down the decimal part) to reach 40,000 steps walking at your given pace. Keep in mind that this is an estimate, and actual time may vary depending on factors like rest periods or changes in your walking speed.

E.4.2 EXAMPLE 2

Prompt

I have a scale of 1 to 7. 1 being the best and 7 the worst. How do I create an index between 0 an 1 where 1 is the best. Can you write a python function that takes in the number and returns the index?

Reference checkpoints

Base

Certainly! You can create a Python function that maps a value from the scale of 1 to 7 (where 1 is the best) to an index between 0 and 1 (where 1 corresponds to the best). Here's how you can do it:

```python
def scale_to_index(value):
    # Ensure the input value is within the range [1, 7]
    if not 1 <= value <= 7:
        raise ValueError("Value must be between 1 and 7 inclusive.")

    # Map the value from the range [1, 7] to the range [0, 1]
    index = (7 - value) / 6

    return index
# Example usage:
value = 1 # Best case
print(f"Index for value {value}: {scale_to_index(value)}")
value = 4 # Middle case
print(f"Index for value {value}: {scale_to_index(value)}")
value = 7 # Worst case
print(f"Index for value {value}: {scale_to_index(value)}")
```

Explanation: - The formula (7 - value) / 6 maps the value from the range [1, 7] to the range [0, 1]. - When value is 1, the result is (7 - 1) / 6 = 1. - When value is 7, the result is (7 - 7) / 6 = 0.
This function ensures that the best value (1) is mapped to 1, and the worst value (7) is mapped to 0. Intermediate values are scaled accordingly.
Feel free to test this function with different values to see how it works!

SFT

Sure, here's a Python function that takes in a number from 1 to 7 and returns an index between 0 and 1, with 1 being the best:

```python
def scale_to_index(number):
    if number == 1:
        return 1.0
    elif number == 2:
        return 0.9
    elif number == 3:
        return 0.8
    elif number == 4:
        return 0.7
    elif number == 5:
        return 0.6
    elif number == 6:
        return 0.5
    else:
        return 0.0
```

This function uses a series of if statements to check the input number and return the corresponding index. The index is calculated by subtracting the input number from 7 and dividing the result by 6. This gives us an index between 0 and 1, with 1 being the best.
For example, if we call scale_to_index(1), the function will return 1.0, which is the best possible index. If we call scale_to_index(7), the function will return 0.0, which is the worst possible index.
I hope this helps! Let me know if you have any other questions.

Preference-optimization baselines

DPO

Certainly! To transform a scale from 1 to 7 into an index between 0 and 1, where 1 corresponds to the best (which is 1 on your scale), you can use a linear transformation. Here's a Python function that does this:

```python
def scale_to_index(value):
    # Define the range of your original scale
    min_value = 1
    max_value = 7

    # Calculate the index between 0 and 1
    index = (max_value - value) / (max_value - min_value)

    return index
# Example usage:
value = 1 # Best case scenario
index = scale_to_index(value)
print(f"Index for value {value}: {index}")
value = 7 # Worst case scenario
index = scale_to_index(value)
print(f"Index for value {value}: {index}")
```

In this function: - min_value is set to 1 (the best value). - max_value is set to 7 (the worst value). - The formula (max_value - value) / (max_value - min_value) maps the values from 1 to 7 to a range of 0 to 1.
When you run this code, it will output:

```
Index for value 1: 1.0
Index for value 7: 0.0
```

This means that the best value (1) will be mapped to 1.0, and the worst value (7) will be mapped to 0.0.

IPO

Certainly! You can create a Python function that maps a value from the scale of 1 to 7 (where 1 is the best) to an index between 0 and 1 (where 1 corresponds to the best). Here's how you can do it:

```
def scale_to_index(value):
    # Ensure the input value is within the range [1, 7]
    if not 1 <= value <= 7:
        raise ValueError("Value must be between 1 and 7 inclusive.")

    # Map the value from the range [1, 7] to the range [0, 1]
    index = (7 - value) / 6

    return index
# Example usage:
value = 1 # Best case
print(f"Index for value {value}: {scale_to_index(value)}")
value = 4 # Middle case
print(f"Index for value {value}: {scale_to_index(value)}")
value = 7 # Worst case
print(f"Index for value {value}: {scale_to_index(value)}")
```

Explanation: - The formula (7 - value) / 6 maps the value from the range [1, 7] to the range [0, 1]. - When value is 1, the result is 1 (best case). - When value is 4, the result is 0.5 (middle case). - When value is 7, the result is 0 (worst case).
This function ensures that the input value is within the specified range and then performs the necessary transformation.

CPO

Certainly! To create an index between 0 and 1, where 1 represents the best (scale value 1) and 0 represents the worst (scale value 7), you can use a linear transformation. Here's a Python function that accomplishes this:

```
def scale_to_index(value):
    # Define the range for the input values
    min_value = 1
    max_value = 7

    # Calculate the index using the formula: (value - min_value) / (max_value - min_value)
    index = (value - min_value) / (max_value - min_value)

    return index
# Example usage:
print(scale_to_index(1)) # Output: 1.0
print(scale_to_index(4)) # Output: 0.5
print(scale_to_index(7)) # Output: 0.0
```

Explanation: - The min_value is set to 1, which corresponds to the best score. - The max_value is set to 7, which corresponds to the worst score. - The formula (value - min_value) / (max_value - min_value) maps the input value from the range [1, 7] to the range [0, 1].
This function will return an index between 0 and 1, with 1 representing the best score (1 on the original scale) and 0 representing the worst score (7 on the original scale).

KTO

Certainly! You can create a Python function that maps the scale from 1 to 7 to a new scale from 0 to 1, with 1 being the best (i.e., the lowest number on the original scale). Here's how you can do it:

```python
def scale_to_index(value):
    # The minimum value on the original scale
    min_value = 1
    # The maximum value on the original scale
    max_value = 7

    # Calculate the index using the formula: (value - min_value) / (max_value - min_value)
    index = (min_value - value) / (min_value - max_value)

    return index
# Example usage:
print(scale_to_index(1)) # Should print 1.0
print(scale_to_index(2)) # Should print 0.8571428571428571
print(scale_to_index(3)) # Should print 0.7142857142857143
print(scale_to_index(4)) # Should print 0.5714285714285714
print(scale_to_index(5)) # Should print 0.42857142857142855
print(scale_to_index(6)) # Should print 0.2857142857142857
print(scale_to_index(7)) # Should print 0.0
```

In this function, we use the formula (min_value - value) / (min_value - max_value) to map the values from the original scale to the new scale. This ensures that the highest value (1) on the original scale corresponds to 1.0 on the new scale, while the lowest value (7) corresponds to 0.0.

SimPO

Certainly! To transform a rating from a scale of 1 to 7 into an index on a scale of 0 to 1, where 1 corresponds to the best (rating 1) and 0 corresponds to the worst (rating 7), you can use a linear transformation.
Here's how you can implement this in Python:

```python
def transform_rating_to_index(rating):
    # Define the range for the original scale and the target scale
    original_min = 1
    original_max = 7
    target_min = 0
    target_max = 1

    # Calculate the transformed index
    index = ((target_max - target_min) * (original_min - rating)) / (original_min - original_max) + target_min

    return index
# Example usage:
print(transform_rating_to_index(1)) # Should print 1.0
print(transform_rating_to_index(4)) # Should print 0.5
print(transform_rating_to_index(7)) # Should print 0.0
```

Explanation: - **Original Scale**: The rating ranges from 1 to 7. - **Target Scale**: We want to map these ratings to a scale from 0 to 1. - **Formula**: The formula used here is derived from linear interpolation. It maps the original minimum value (1) to the target maximum (1) and the original maximum value (7) to the target minimum (0).
Output: - A rating of 1 will be mapped to 1.0 (best). - A rating of 4 will be mapped to 0.5 (middle). - A rating of 7 will be mapped to 0.0 (worst).
Feel free to test with other values within the 1 to 7 range!

Token-aware methods

TDPO

Certainly! You can create a Python function that maps the scale from 1 to 7 to an index between 0 and 1, with 1 corresponding to the best (which is 1 on your scale) and 7 corresponding to the worst (which is 7 on your scale).
Here's how you can do it:

```
def scale_to_index(value):
    # The best value is 1 and the worst value is 7
    # We want to map 1 to 1 and 7 to 0
    return 1 - (value - 1) / 6
# Example usage:
print(scale_to_index(1)) # Should print 1.0
print(scale_to_index(4)) # Should print 0.5
print(scale_to_index(7)) # Should print 0.0
```

Explanation: - For the best value (1), the index should be 1. - For the worst value (7), the index should be 0. - The formula 1 - (value - 1) / 6 linearly maps the values from 1 to 7 to the range 1 to 0.
This function will take a value from the 1-to-7 scale and return the corresponding index between 0 and 1.

TIS-DPO

Certainly! You can create a Python function that maps a value from the scale of 1 to 7 (where 1 is the best) to an index between 0 and 1 (where 1 corresponds to the best, i.e., 1 on your original scale).
Here's how you can implement this:

```
def scale_to_index(value):
    # The best score is 1, which should map to 1 in the new index.
    # The worst score is 7, which should map to 0 in the new index.
    return (7 - value) / 6
# Example usage:
print(scale_to_index(1)) # Should print 1.0
print(scale_to_index(7)) # Should print 0.0
print(scale_to_index(4)) # Should print 0.5
```

In this function: - When value is 1 (the best), (7 - 1) / 6 equals 1. - When value is 7 (the worst), (7 - 7) / 6 equals 0. - For any other value in between, the function will return a corresponding index between 0 and 1.
This mapping ensures that as the original value decreases (i.e., gets closer to 1), the new index increases (i.e., gets closer to 1). Conversely, as the original value increases (i.e., gets closer to 7), the new index decreases (i.e., gets closer to 0).

ConfPO

Certainly! You can create a Python function that maps a value from the scale of 1 to 7 (where 1 is the best) to an index between 0 and 1 (where 1 corresponds to the best).
Here's how you can do it:

```python
def scale_to_index(value):
    # Ensure the input value is within the range [1, 7]
    if not 1 <= value <= 7:
        raise ValueError("Value must be between 1 and 7 inclusive.")

    # Map the value from the range [1, 7] to the range [0, 1]
    index = (7 - value) / 6

    return index
# Example usage:
value = 1 # Best case scenario
index = scale_to_index(value)
print(f"Value: {value}, Index: {index}")
value = 7 # Worst case scenario
index = scale_to_index(value)
print(f"Value: {value}, Index: {index}")
```

Explanation: - The formula (7 - value) / 6 maps the value from the range [1, 7] to the range [0, 1]. - When value is 1, the result is 1 (best). - When value is 7, the result is 0 (worst).
This function ensures that the input value is within the specified range and then performs the necessary transformation.

EG-DPO

Sure, I can help you with that. To create a Python function that converts the scale from 1 to 7 into an index between 0 and 1, where 1 represents the best (original value 1), you can use the following function:

```python
def scale_to_index(value):
    min_value = 1
    max_value = 7
    index = (1 - ((value - min_value) / (max_value - min_value)))
    return index
```

Here's how the function works:
1. We define the minimum (min_value = 1) and maximum (max_value = 7) values of your original scale.
2. Inside the function, we calculate the index by subtracting the normalized value (considering the range) from 1.
When you call this function with a value between 1 and 7, it will return an index between 0 and 1, with 1 corresponding to the best (original value 1) in your scale.