# LESS IS MORE IN DPO: LEARNING FROM THE LOWEST-ENTROPY 20% OF TOKENS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Direct Preference Optimization (DPO) is widely used to align large language models with human preferences. However, its sequence-level objective does not explicitly distinguish the contributions of individual tokens. We find that optimizing only a small subset of low-entropy tokens can outperform using all response tokens. Based on this observation, we propose **Low-Entropy Support DPO (LES-DPO)**. LES-DPO ranks tokens by the policy model's predictive entropy. It selects a low-entropy subset separately from each chosen and rejected response. The preference loss uses only these selected positions. LES-DPO requires no auxiliary models, additional policy forward passes, or extra trainable parameters. Experiments on six Qwen2.5, Llama-3.1, and Mistral models show consistent improvements on AlpacaEval 2, Arena-Hard, and MT-Bench. Gradient analysis shows that high-entropy tokens tend to produce larger gradients. Their directions are less consistent and conflict more often. These findings suggest that focusing on low-entropy tokens can provide more stable signals for preference learning.

## 1 INTRODUCTION

Reinforcement Learning from Human Feedback (RLHF) has become a standard paradigm for aligning large language models with human preferences (Ouyang et al., 2022), but its multi-stage training pipeline is complex and costly. Direct Preference Optimization (DPO) simplifies this process by using the Bradley–Terry preference model to optimize the policy directly on preference pairs (Bradley & Terry, 1952; Rafailov et al., 2023). However, DPO still operates at the sequence level. It aggregates all response tokens without distinguishing their individual contributions to the preference, leaving a token-level credit assignment problem (Wang et al., 2025a).

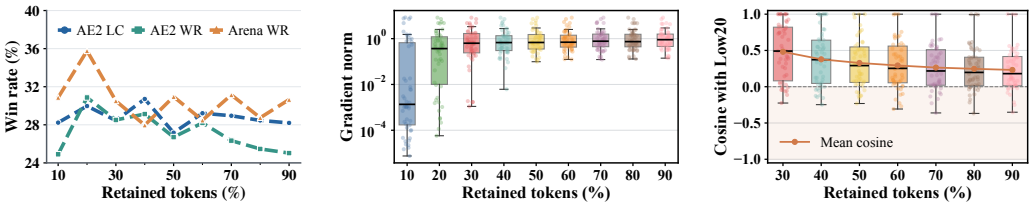
Recent token-level preference optimization methods have begun to address this problem. Some redistribute preference supervision within a response using token-level Kullback–Leibler (KL) divergence, importance sampling, semantic matching, token-level reward estimation, or attention (Zeng et al., 2024; Liu et al., 2025; Li et al., 2025; Yang et al., 2025; Huang et al., 2026). Others use confidence from the policy itself to identify important tokens (Yoon et al., 2025). Nevertheless, it remains unclear which tokens should actually be included in the preference objective.

A natural intuition is that tokens with greater model uncertainty contain more information to learn and therefore deserve more optimization. However, we make two observations (Figure 1). (1) Optimizing only a small subset of the lowest-entropy tokens under the current policy can outperform using all response tokens, while including higher-entropy regions can reduce performance. (2) Expanding the low-entropy support changes both gradient magnitude and direction. The cumulative gradient becomes less aligned with the gradient from the lowest-entropy 20% of tokens. These observations motivate examining which tokens contribute useful preference signals, rather than assuming that including more tokens always helps.

We study this question through the predictive entropy of the policy model, which measures its uncertainty over the next-token distribution given the current prefix. Rather than assuming that either high- or low-entropy tokens are always better, we compare three independent design choices: whether entropy is computed by the policy or the reference model, whether low- or high-entropy regions are selected on the chosen and rejected sides, and what fraction of tokens is kept. Based on this comparison, we propose Low-Entropy Support DPO (LES-DPO). For each chosen and rejected response, we use the current policy to compute the predictive entropy of every valid response token. Only the lowest-entropy 20% support (Low20) in each response is used to compute the policy/reference



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.




(a) Preference win rates (b) Cumulative gradient magnitudes (c) Alignment with Low20

Figure 1: Motivation for low-entropy support selection on Qwen2.5-3B-Instruct. **(a)** AlpacaEval 2 LC and WR and Arena-Hard WR when retaining the lowest-policy-entropy 10%–90% of tokens. **(b)** Gradient norms for these cumulative low-entropy supports. **(c)** Cosine similarity between the gradients of expanded supports (30%–90%) and the Low20 gradient. Gradient diagnostics in (b) and (c) use the same 50 preference pairs at a fixed model state. Boxes show quartiles and medians; dots show individual batches. In (c), the orange line shows mean cosine, and shading marks negative cosine.

log-ratio and the DPO loss. The remaining 80% are excluded from the loss and do not directly contribute to that parameter update. Entropy is used only for token selection and receives no gradient. LES-DPO requires no auxiliary reward model, token-level oracle, additional policy forward pass, or new trainable parameters. Its only added cost is computing entropy from the existing logits and ranking the response tokens.

Further gradient analysis shows that the lowest-entropy 0–10% interval produces small gradients, while the cumulative Low20 gradient is more closely aligned with the 10–20% interval. When the support is expanded to Low30, the cumulative gradient becomes more aligned with the newly added 20–30% interval and less aligned with Low20. These observations support gradient interference between entropy intervals as a possible explanation, but do not establish that any one interval alone determines the final performance.

Our main contributions are threefold:

- We find that optimizing only a small subset of low-entropy tokens can outperform using all response tokens. Based on this finding, we propose LES-DPO, which constructs the preference objective from the lowest-policy-entropy 20% of tokens in each chosen and rejected response, without auxiliary models or additional policy forward passes.
- We analyze gradient magnitudes, directional alignment, and conflict across entropy intervals and cumulative supports. Together with gradient-preserving training controls, these analyses probe the roles of normalization and gradient interference in explaining why adding more tokens can reduce performance.
- We evaluate LES-DPO on six models from the Qwen2.5, Llama-3.1, and Mistral families using AlpacaEval 2, Arena-Hard, and MT-Bench, demonstrating substantial performance improvements. Extensive ablations examine entropy sources, selection regions, retained fractions, and token-weighting strategies.

## 2 Related Work

**Sequence-level variants.** RLHF aligns language models using a learned reward model and reinforcement learning (Ouyang et al., 2022). DPO simplifies this pipeline by directly optimizing preference pairs without an explicit reward model or online reinforcement learning (Rafailov et al., 2023). Subsequent methods revise the objective or feedback format: IPO uses a squared preference loss to improve robustness to preference noise (Azar et al., 2024), while KTO uses prospect theory to model asymmetric utilities for desirable and undesirable responses and supports unpaired feedback (Ethayarajh et al., 2024). SimPO instead uses a length-normalized, reference-free reward with a target margin (Meng et al., 2024). CPO combines a contrastive preference objective with a likelihood term for chosen responses (Xu et al., 2024), coupling preference learning with supervised learning. These methods operate at the response level, leaving the contributions of individual tokens undifferentiated.

**Token-level credit assignment.** The granularity mismatch between sequence-level preference labels and autoregressive token decisions has motivated finer-grained credit assignment. One line of work redistributes token supervision using *additional signals*, including token-level KL (TDPO) (Zeng et al., 2024), importance sampling (TIS-DPO) (Liu et al., 2025), optimal transport (OTPO) (Li et al., 2025), token-level reward estimation (SePO) (Yang et al., 2025), or attention (AttentionPO) (Huang et al., 2026). ConfPO identifies preference-critical tokens from the policy's own confidence, without auxiliary models or additional forward passes (Yoon et al., 2025).

However, which entropy regions provide useful preference signals, how many tokens should be retained, and how their gradients interact remain insufficiently understood. We address these questions by comparing entropy sources, selection regions, and retained fractions (§5.3). Our LES-DPO uses the current policy's predictive entropy to select the lowest-entropy 20% of tokens independently in chosen and rejected responses, without auxiliary models or additional policy forward passes. Gradient diagnostics and training controls further examine how adding higher-entropy tokens can interfere with the selected low-entropy update (§5.4–§5.5; Appendix F.3).

## 3 Preliminaries

Given a preference dataset $\mathcal{D} = \{(x, y_w, y_l)\}$, where $y_w$ is preferred to $y_l$ for prompt $x$, let $\pi_\theta$ be the trainable policy and $\pi_{\text{ref}}$ the fixed reference policy. For a response $y = (y_1, \ldots, y_T)$, define the *token-level log-ratio*

$$\delta_t(x, y) = \log \pi_\theta(y_t \mid x, y_{<t}) - \log \pi_{\text{ref}}(y_t \mid x, y_{<t}). \tag{1}$$

The dependence of $\delta_t$ on $\theta$ is implicit. Autoregressive factorization gives the DPO *implicit reward*

$$r_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} = \beta \sum_{t=1}^{T} \delta_t(x, y), \tag{2}$$

where $\beta > 0$ scales the reward. Under the Bradley–Terry model, the probability of preferring $y_w$ to $y_l$ is the sigmoid of their reward difference. DPO therefore minimizes

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \Big[ \log \sigma\big(r_\theta(x, y_w) - r_\theta(x, y_l)\big) \Big], \tag{3}$$

where $\sigma$ is the logistic sigmoid. Each response token enters the implicit reward with the same coefficient. We instead use predictive entropy to select which token-level contributions enter the preference objective.

## 4 Method

LES-DPO constructs the preference loss from mean policy–reference log-ratios over the lowest-entropy 20% of tokens in each response. Figure 2 illustrates the selection and optimization procedure.

### 4.1 Predictive Entropy of the Current Policy

At position $t$, the policy's predictive entropy measures its uncertainty over the vocabulary $\mathcal{V}$ (Shannon, 1948):

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

Lower entropy indicates a more concentrated next-token distribution. This quantity depends on the prefix $(x, y_{<t})$ and describes the entire distribution; it is distinct from the probability of the observed token $y_t$. Thus, low entropy alone does not establish the correctness or preference relevance of that token.

Let $a_{t,v}$ be the policy logit for vocabulary item $v$ at position $t$, and write $p_{t,v} = \pi_\theta(v \mid x, y_{<t})$. The same logits yield both the token probabilities and the entropy:

$$p_{t,v} = \frac{\exp(a_{t,v})}{\sum_{v' \in \mathcal{V}} \exp(a_{t,v'})}, \qquad H_t^\theta = \log \sum_{v \in \mathcal{V}} \exp(a_{t,v}) - \sum_{v \in \mathcal{V}} p_{t,v} a_{t,v}. \tag{5}$$

The second identity follows by substituting the log-softmax expression into Eq. (4); it requires no additional model evaluation.

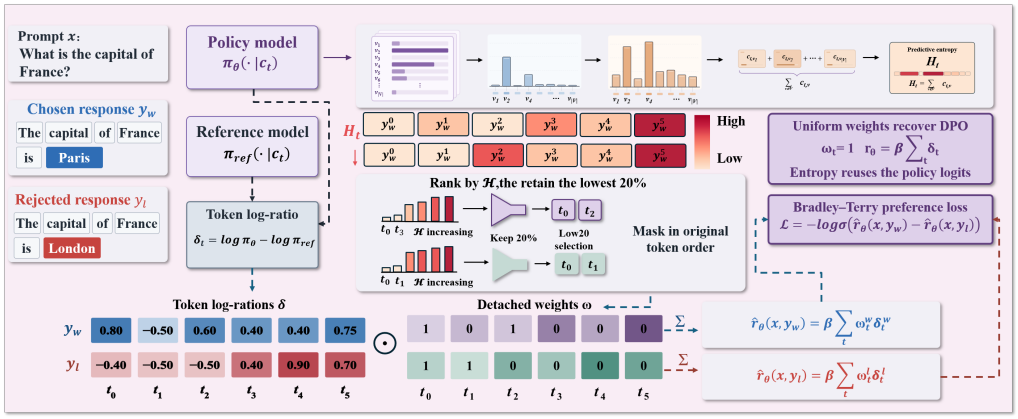



Figure 2: Overview of LES-DPO. The current policy computes predictive entropy at each response position and selects the lowest-entropy 20% independently from the chosen and rejected responses. Policy–reference log-ratios are averaged over the selected positions and scaled by $\beta$ to obtain response scores, whose difference defines the Bradley–Terry preference loss. Selection masks are detached during backpropagation.

We compute entropy on the responses in the preference dataset, using logits already produced by the policy forward pass. For selection, we use the detached values $H_t = \mathrm{sg}[H_t^\theta(x, y_{<t})]$, where $\mathrm{sg}[\cdot]$ preserves the forward value and has zero derivative. The entropies and selected sets are recomputed at each training step, so selection tracks the current policy while remaining fixed during each backward pass.

### 4.2 From Entropy to Low-Entropy Support

The chosen and rejected responses are processed independently. For one response, let $\mathcal{I}(y)$ contain its valid token positions, excluding prompt tokens, padding, and masked labels, and let $N = |\mathcal{I}(y)| \geq 1$. Given a retained fraction $\rho \in (0, 1]$, we select

$$k_\rho(y) = \max(1, \lceil \rho N \rceil) \tag{6}$$

positions with the lowest detached entropy:

$$\mathcal{S}_\rho(x, y) = \mathrm{BottomK}_{t \in \mathcal{I}(y)}\big(H_t; k_\rho(y)\big). \tag{7}$$

Here BottomK returns the indices of the $k$ smallest values, breaking ties to retain exactly $k$ positions. The final method uses $\rho = 0.2$: "Low20" denotes an entropy rank within each response, rather than a segment in generation order. For fewer than eight valid tokens or entropy variance below $10^{-12}$, we set $\mathcal{S}_\rho(x, y) = \mathcal{I}(y)$ on that response side (Appendix B.2).

Outside these fallback cases, BottomK equivalently solves a selection problem with a fixed token budget. Let $m = (m_t)_{t \in \mathcal{I}(y)} \in \{0, 1\}^N$ be a mask whose entries indicate whether each valid position is selected. Then

$$\begin{aligned} \mathcal{M}_k &= \left\{ m \in \{0,1\}^N : \sum_{t \in \mathcal{I}(y)} m_t = k \right\}, \\ m^\star &\in \operatorname*{arg\,min}_{m \in \mathcal{M}_{k_\rho(y)}} \sum_{t \in \mathcal{I}(y)} m_t H_t, \qquad \mathcal{S}_\rho(x, y) = \{t \in \mathcal{I}(y) : m_t^\star = 1\}. \end{aligned} \tag{8}$$

Thus, entropy determines membership in a fixed-size support; it is not an additional penalty minimized by the policy. Tied minima are resolved by the same rule as BottomK.

We convert the selected set into detached, normalized token weights:

$$\omega_t(x, y) = \mathrm{sg}\left[ \frac{\mathbf{1}\{t \in \mathcal{S}_\rho(x, y)\}}{|\mathcal{S}_\rho(x, y)|} \right], \qquad \sum_{t=1}^{T} \omega_t(x, y) = 1. \tag{9}$$

All selected positions receive equal weight; other positions receive zero weight. We write $\omega_t^z = \omega_t(x, y_z)$ and $\delta_t^z = \delta_t(x, y_z)$ for $z \in \{w, l\}$. Each side uses its own prefixes, ranking, and normalization. The reference policy supplies the comparison log-probabilities in Eq. (1); selection depends only on the current policy.

### 4.3 Objective and Gradient

The selected-token implicit reward is the scaled mean log-ratio over the low-entropy support:

$$\hat{r}_\theta(x, y) = \beta \sum_{t=1}^{T} \omega_t(x, y)\, \delta_t(x, y) = \frac{\beta}{|\mathcal{S}_\rho(x, y)|} \sum_{t \in \mathcal{S}_\rho(x,y)} \delta_t(x, y). \tag{10}$$

For one preference pair, the resulting margin explicitly compares the two independently normalized supports:

$$\begin{aligned} u_\theta(x, y_w, y_l) &= \hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l) \\ &= \beta \left[ \frac{\sum_{t \in \mathcal{S}_\rho(x,y_w)} \delta_t^w}{|\mathcal{S}_\rho(x, y_w)|} - \frac{\sum_{t \in \mathcal{S}_\rho(x,y_l)} \delta_t^l}{|\mathcal{S}_\rho(x, y_l)|} \right]. \end{aligned} \tag{11}$$

Substituting this margin into the Bradley–Terry loss gives

$$\begin{aligned} \mathcal{L}_{\text{LES-DPO}}(\theta) &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \big[ \log \sigma(u_\theta) \big] \\ &= \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \big[ \log(1 + \exp(-u_\theta)) \big]. \end{aligned} \tag{12}$$

We use $\rho = 0.2$ and $\beta = 0.1$ across all models, with Low/Low selection on the chosen and rejected responses. Appendix B.5 gives the remaining training settings.

**Normalization.** The weights sum to one, so the reward averages the selected contributions. This removes their direct accumulation with support size, although reward distributions can still differ across responses and selection fractions. At $\rho = 1$, the objective uses the mean log-ratio over all valid response tokens. Standard DPO uses a sum, so the two objectives generally differ under the same $\beta$. The normalization and $\beta$ jointly determine the reward scale; Appendix A.1 gives the exact relation.

**Gradient.** For one preference pair, abbreviate $u = u_\theta(x, y_w, y_l)$ and let $\ell = -\log \sigma(u)$, so $\partial \ell / \partial u = -\sigma(-u)$. Since $\nabla_\theta \omega_t^z = 0$ during the backward pass, the gradient retains the DPO form with selected-token weights:

$$\begin{aligned} \nabla_\theta \ell = -\beta \sigma(-u) \Bigg[ & \sum_{t \in \mathcal{I}(y_w)} \omega_t^w \nabla_\theta \log \pi_\theta(y_{w,t} \mid x, y_{w,<t}) \\ & - \sum_{t \in \mathcal{I}(y_l)} \omega_t^l \nabla_\theta \log \pi_\theta(y_{l,t} \mid x, y_{l,<t}) \Bigg]. \end{aligned} \tag{13}$$

Here $y_{z,t}$ and $y_{z,<t}$ denote the token and prefix on side $z$. The fixed reference affects $u$ and hence the scalar coefficient, but contributes no parameter gradient. Unselected positions contribute no direct log-ratio term, while their tokens remain context for later selected positions. Appendix A.1 derives this expression and discusses changes in the selected sets.

The effect at each output position can also be made explicit. Let $a_{t,v}^z$ and $p_{t,v}^z$ denote its logit and probability on side $z \in \{w, l\}$, and set $\varepsilon_w = 1$, $\varepsilon_l = -1$. With the selection masks fixed, the log-softmax derivative gives

$$\frac{\partial \ell}{\partial a_{t,v}^z} = \varepsilon_z\, \beta \sigma(-u)\, \omega_t^z \left( p_{t,v}^z - \mathbf{1}\{v = y_{z,t}\} \right). \tag{14}$$

At selected positions, the chosen and rejected sides therefore receive opposite-signed log-probability signals, scaled by the pairwise coefficient $\beta\sigma(-u)$ and the inverse support size. At unselected positions, $\omega_t^z = 0$ makes this direct output-logit derivative zero. Shared parameters and contextual dependencies still couple the positions, so the parameter update can change predictions outside the selected support.

Table 1: Main results on Qwen2.5, Llama-3.1, and Mistral. Each model family forms one block, whose two sides compare model scales or Base/Instruct variants. The Base row denotes the original checkpoint without training in our experiments. AlpacaEval 2 and Arena-Hard report win rates (%), while MT-Bench reports the average score. Bold indicates the best result in each column.

| Method | Qwen2.5-3B-Instruct | | | | Qwen2.5-7B-Instruct | | | |
|---|---|---|---|---|---|---|---|---|
| | AlpacaEval 2 | | Arena-Hard | MT-Bench | AlpacaEval 2 | | Arena-Hard | MT-Bench |
| | LC (%) | WR (%) | WR (%) | Score | LC (%) | WR (%) | WR (%) | Score |
| Base | 27.90 | 26.15 | 28.62 | 8.175 | 39.73 | 36.15 | 44.65 | 8.450 |
| SFT | 12.88 | 8.41 | 6.08 | 7.145 | 14.92 | 9.38 | 10.21 | 7.612 |
| DPO | 27.90 | 26.15 | 28.62 | 8.175 | 40.31 | 36.54 | 47.12 | 8.462 |
| IPO | 29.08 | 27.82 | 31.04 | 8.216 | 40.02 | 36.22 | 48.06 | **8.468** |
| CPO | 18.47 | 14.82 | 17.26 | 7.824 | 21.96 | 15.03 | 20.18 | 7.904 |
| KTO | 25.36 | 21.18 | 24.82 | 8.021 | 33.14 | 24.27 | 38.46 | 8.318 |
| SimPO | **30.34** | 27.19 | 29.63 | 8.103 | 43.18 | 35.84 | 44.12 | 8.326 |
| TDPO | 30.02 | 28.04 | 31.18 | 8.171 | 40.97 | 36.83 | 46.03 | 8.451 |
| TIS-DPO | 29.03 | 30.12 | 33.08 | **8.221** | 43.35 | 37.64 | 47.83 | 8.421 |
| ConfPO | 29.24 | 27.43 | 30.41 | 8.183 | 40.84 | 36.31 | 46.72 | 8.439 |
| LES-DPO | 29.97 | **30.87** | **35.77** | 8.144 | **43.57** | **38.28** | **48.44** | 8.433 |

| Method | Llama-3.1-8B Base | | | | Llama-3.1-8B Instruct | | | |
|---|---|---|---|---|---|---|---|---|
| | AlpacaEval 2 | | Arena-Hard | MT-Bench | AlpacaEval 2 | | Arena-Hard | MT-Bench |
| | LC (%) | WR (%) | WR (%) | Score | LC (%) | WR (%) | WR (%) | Score |
| Base | 0.33 | 0.62 | 0.51 | 3.212 | 39.50 | 39.38 | 27.63 | 8.113 |
| SFT | 5.46 | 6.12 | 2.87 | 6.310 | 34.86 | 32.47 | 24.58 | 7.940 |
| DPO | 5.82 | 6.94 | 3.18 | 6.382 | 39.96 | 38.12 | 29.84 | 8.201 |
| IPO | 7.18 | 10.31 | 5.08 | 6.542 | **40.48** | 39.08 | 32.72 | **8.319** |
| CPO | 4.46 | 4.92 | 2.38 | 5.864 | 28.24 | 23.47 | 19.81 | 7.884 |
| SimPO | 6.12 | 5.83 | 2.71 | 5.782 | 39.18 | 35.42 | 30.61 | 8.004 |
| KTO | 6.81 | 7.42 | 3.94 | 6.334 | 39.04 | 34.63 | 26.12 | 8.228 |
| TDPO | 5.37 | 6.02 | 3.11 | 6.216 | 39.31 | 37.06 | 29.15 | 8.102 |
| TIS-DPO | 8.63 | 9.74 | 4.82 | 6.728 | 40.17 | 38.71 | 32.44 | 8.214 |
| ConfPO | 6.54 | 6.68 | 3.67 | 6.315 | 40.16 | 38.37 | 30.24 | 8.171 |
| LES-DPO | **10.28** | **10.47** | **5.21** | **6.846** | 40.35 | **39.61** | **32.96** | 8.241 |

| Method | Mistral-7B Base | | | | Mistral-7B Instruct | | | |
|---|---|---|---|---|---|---|---|---|
| | AlpacaEval 2 | | Arena-Hard | MT-Bench | AlpacaEval 2 | | Arena-Hard | MT-Bench |
| | LC (%) | WR (%) | WR (%) | Score | LC (%) | WR (%) | WR (%) | Score |
| Base | 0.43 | 0.99 | 0.49 | 4.331 | 10.95 | 5.53 | 2.33 | 7.162 |
| SFT | 8.99 | 6.96 | 3.95 | 6.806 | 11.09 | 7.08 | 2.66 | 6.669 |
| DPO | 5.14 | 6.28 | 3.06 | 6.214 | 13.18 | 8.74 | 4.21 | 7.418 |
| IPO | 11.72 | 12.85 | 6.14 | 6.870 | 15.37 | 11.12 | 5.10 | 7.312 |
| CPO | 6.18 | 5.14 | 2.87 | 6.503 | 8.92 | 5.41 | 2.58 | 6.803 |
| SimPO | 11.12 | 9.81 | 4.18 | 6.153 | 12.84 | 7.93 | 3.52 | 7.336 |
| KTO | 11.03 | 11.30 | 5.50 | 6.920 | 15.96 | 11.12 | 4.81 | 7.369 |
| TDPO | 4.82 | 5.69 | 2.98 | 5.984 | 12.97 | 7.82 | 3.77 | 7.354 |
| TIS-DPO | 10.84 | 12.93 | 6.28 | 6.884 | 15.55 | 14.31 | 5.76 | **7.580** |
| ConfPO | 5.73 | 6.81 | 3.42 | 6.351 | 13.64 | 8.42 | 4.03 | 7.312 |
| LES-DPO | **12.64** | **14.26** | **6.82** | **7.058** | **17.14** | **14.82** | **5.94** | 7.562 |

# 5 Experiments

## 5.1 Experimental Setup

**Models and data.** We train Qwen2.5-3B/7B-Instruct (Yang et al., 2024) and the Base/Instruct variants of Llama-3.1-8B (Grattafiori et al., 2024) and Mistral-7B-v0.1 (Jiang et al., 2023) on 60,829 cleaned UltraFeedback Binarized preference pairs (Cui et al., 2024); see Appendix B.1.

**Training.** All methods use one epoch of LoRA training with the same data, seed, and adapter configuration. Preference optimization uses a learning rate of $5 \times 10^{-6}$, BF16, FlashAttention 2 (Dao, 2024), and DeepSpeed (Rasley et al., 2020). Hyperparameters and hardware are listed in Appendices B.3–B.6.

**Evaluation.** We report AlpacaEval 2 length-controlled and raw win rates (LC/WR) (Dubois et al., 2024), Arena-Hard WR (Li et al., 2024), and MT-Bench scores (Zheng et al., 2023). All evaluations use GPT-4 (OpenAI, 2023). Evaluation protocols appear in Appendix B.7.

### 5.2 Main Results

Table 1 shows that LES-DPO outperforms DPO in 22 of 24 model–metric comparisons, except MT-Bench on the two Qwen models. It achieves the highest AlpacaEval 2 WR and Arena-Hard WR across all six models, leads all four metrics on Llama-3.1 Base and Mistral Base, and also leads AlpacaEval 2 LC on Qwen2.5-7B.

### 5.3 Ablation Studies

We ablate the entropy source, selected region, and retained token fraction.

#### 5.3.1 Policy Entropy or Reference Entropy?

Table 2 compares 20% token supports on each response side: random selection, reference-based low entropy, and policy-based low or high entropy. Policy-Low20 is LES-DPO and leads all three win-rate metrics, though its MT-Bench score is slightly below DPO. Reference-Low20 and Random20 trail Policy-Low20 on all four metrics, supporting current-policy selection. Appendix D tracks policy/reference selection overlap during training.

Table 2: Comparison of entropy sources and token-selection regions.

| Method | AlpacaEval 2 | | Arena-Hard | MT-Bench |
|---|---|---|---|---|
| | LC (%) | WR (%) | WR (%) | Score |
| DPO | 27.900 | 26.150 | 28.620 | **8.175** |
| Random20 | 25.635 | 22.236 | 24.285 | 8.012 |
| Reference-Low20 | 26.839 | 24.099 | 28.883 | 8.019 |
| Policy-Low20 | **29.971** | **30.870** | **35.770** | 8.144 |
| Policy-High20 | 19.418 | 11.988 | 12.880 | 7.774 |

#### 5.3.2 Why low entropy and why 20%?

In Figure 3(a–d), Low/Low-20% gives the highest AlpacaEval 2 WR (30.870%) and Arena-Hard WR (35.770%) among 36 cumulative configurations, exceeding DPO by 4.720 and 7.150 percentage points. These results motivate retaining the lowest-entropy 20%, although it does not maximize every metric.

Panels (e–h) compare nine disjoint 10%-wide bands for each chosen/rejected combination. Low/Low performs well in the lowest three bands; Low/High improves around 40–60%, whereas High/Low degrades at 60–90%. The useful entropy region therefore depends on how tokens are selected on both response sides.

Among the three lowest bands and cumulative 0–20%, isolated 20–30% gives the highest AlpacaEval 2 LC (30.833%) with half as many tokens. Cumulative 0–20% instead improves AlpacaEval 2 WR and Arena-Hard WR by 0.187 and 4.176 percentage points, and MT-Bench by 0.019 (8.144 versus 8.125). Isolated 0–10% gives the highest MT-Bench score (8.162), but lower win rates.

### 5.4 Analysis of Low-Entropy Intervals

#### 5.4.1 Performance across Low-Entropy Intervals

Figure 4 compares three disjoint low-entropy bands and two cumulative supports relative to DPO. All five improve AlpacaEval 2 LC and Arena-Hard WR, but their strengths differ. The isolated 20–30% band gives the highest LC (30.833%) and a competitive raw win rate (30.683%), using only 10% of tokens per response. The 0–10% band achieves the highest MT-Bench score among these supports (8.162), although all five remain below DPO (8.175).

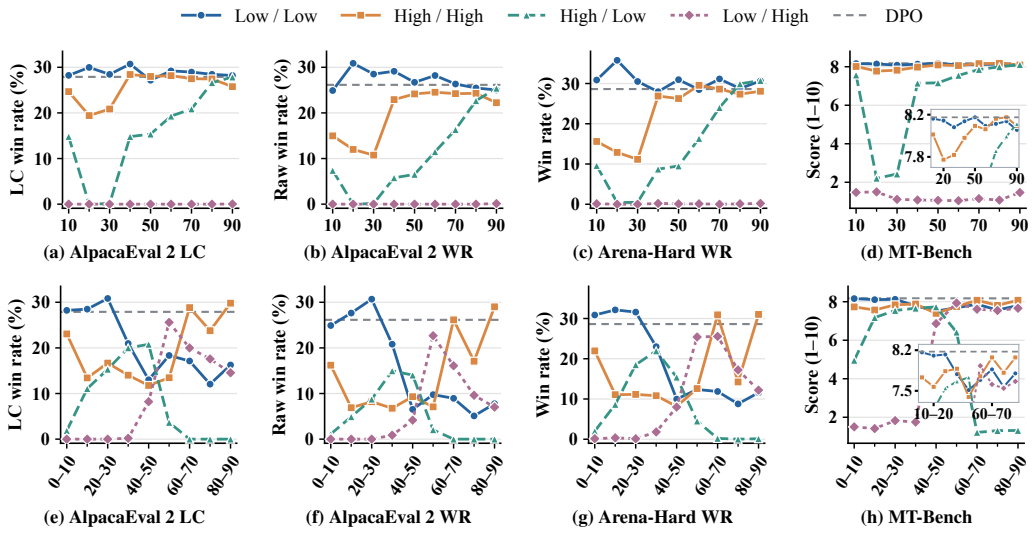



Figure 3: Entropy-based token selection on Qwen2.5-3B-Instruct. **(a–d)** Cumulative retained token fractions from 10% to 90%. **(e–h)** Independent 10%-wide entropy-rank bands. Panels **(a,e)**, **(b,f)**, **(c,g)**, and **(d,h)** report AlpacaEval 2 LC, AlpacaEval 2 WR, Arena-Hard WR, and MT-Bench, respectively. The first and second region labels refer to chosen and rejected responses. Bands are measured from the specified entropy end: High 10–20%, for example, corresponds to ascending ranks 80–90%. Dashed gray lines show the same all-token DPO baseline in both rows; the insets in (d) and (h) expand the high-score ranges.

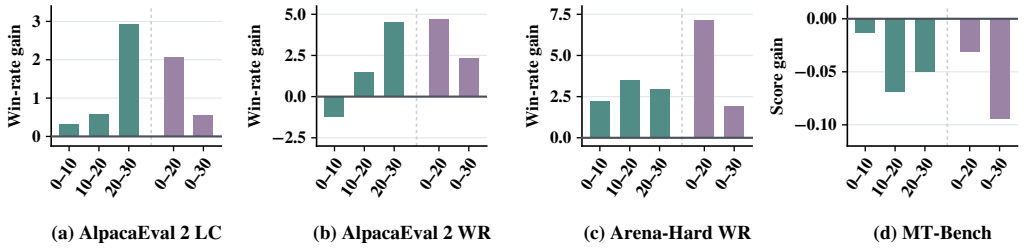


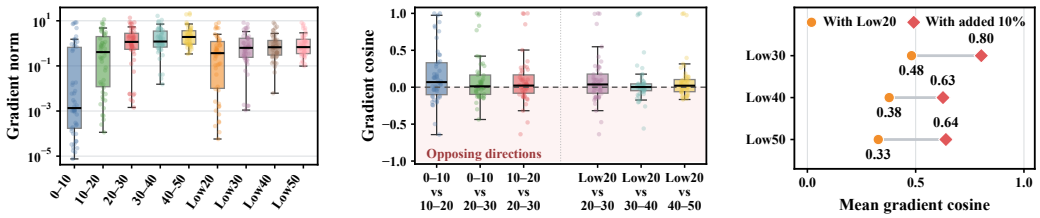

Figure 4: Bars show performance changes relative to DPO on Qwen2.5-3B-Instruct when selecting the same low-entropy interval on both chosen and rejected sides: teal bars represent the non-overlapping intervals 0–10%, 10–20%, and 20–30%, while purple bars represent the cumulative intervals 0–20% and 0–30%. Heights denote win-rate changes in percentage points in panels a–c and MT-Bench score changes in panel d; zero denotes DPO.

Cumulative 0–20% provides the strongest raw win rates: 30.870% on AlpacaEval 2 and 35.770% on Arena-Hard. Extending it to 0–30% lowers all four metrics, including Arena-Hard to 30.507%. Thus, the 20–30% band performs well alone but does not improve the cumulative Low20 objective when added. This contrast motivates examining how the gradients of individual bands interact.

### 5.4.2 Gradient Analysis of Low-Entropy Intervals

Figure 5 compares five disjoint 10% bands within the lowest-entropy 50% and cumulative Low20–Low50 supports. We compute LoRA gradients on the same 50 UltraFeedback pairs with Qwen2.5-3B-Instruct, $\beta = 0.1$, and no parameter updates, enabling paired comparisons at a fixed model state. Full pairwise matrices are in Appendix F.2.

Panel (a) shows that gradient magnitude varies substantially even among these low-entropy bands: 0–10% has the smallest median norm, and later bands produce progressively larger median norms. Larger gradients need not reinforce the same update. In panel (b), pairs among the lowest three




(a) Gradient magnitude (b) Direction variability (c) Cumulative direction

Figure 5: Low-entropy gradient diagnostics. Panel a shows gradient-norm boxplots and per-batch dots. Panel b shows cosine boxplots for six pairs; shading marks negative cosine. In panel c, circles and diamonds show mean alignment with Low20 and the newly added 10% interval, respectively.

bands oppose each other on 38%–46% of batches. The 20–50% bands also align weakly with Low20 (mean cosine 0.054–0.093), with negative cosine on 40%–48% of batches.

Panel (c) shows the resulting directional shift as support expands. From Low30 to Low50, mean alignment with Low20 falls from 0.481 to 0.327, while alignment with the newest band is 0.626–0.803. Including more tokens can therefore redirect the update toward the added band rather than preserve the Low20 direction. These fixed-model diagnostics suggest a possible explanation for the interval results, but do not establish the cause of final performance differences.

### 5.5 Effects of Adding Rest80 to Low20

Table 3: Adding Rest80 through normalized weighting or gradient-preserving control.

| Method | $\lambda$ | AlpacaEval 2 | | Arena-Hard | MT-Bench |
|---|---|---|---|---|---|
| | | LC (%) | WR (%) | WR (%) | Score |
| Low20 only | 0 | **29.971** | **30.870** | **35.770** | 8.144 |
| Normalized weighting | 0.05 | 27.224 | 25.963 | 29.177 | 8.144 |
| Normalized weighting | 0.10 | 23.356 | 22.981 | 26.545 | 8.037 |
| Normalized weighting | 0.25 | 26.321 | 22.422 | 26.549 | 8.056 |
| Gradient-preserving control | 0.05 | 29.627 | 29.938 | 32.612 | 8.144 |

We add the remaining 80% of tokens (Rest80) to Low20 in two ways. **Normalized weighting** assigns weights 1 and $\lambda$ to Low20 and Rest80, dividing by their total weight and thereby rescaling the Low20 gradient. **Gradient-preserving control** retains the full Low20-only gradient and adds the Rest80 component induced by normalized weighting at $\lambda = 0.05$ (Appendix A).

In Table 3, normalized weighting at $\lambda = 0.05$ lowers Arena-Hard from 35.770 to 29.177. Preserving the Low20 gradient recovers 32.612, still below Low20 alone. This suggests that both gradient rescaling and the added Rest80 component contribute to the loss, consistent with the gradient diagnostics in Appendix F.3.

## 6 Conclusion

We presented LES-DPO, which constructs the preference objective from the lowest-policy-entropy 20% of tokens in each chosen and rejected response, without auxiliary models or additional policy forward passes. Gradient analyses and training controls suggest that adding tokens can rescale the Low20 contribution and introduce conflicting update directions, helping explain why larger token supports do not consistently improve performance. Across six models from three families and three evaluation benchmarks, LES-DPO improves over DPO in 22 of 24 model–metric comparisons. Ablations further show that entropy source, rank interval, and retained fraction all matter. These findings support selective optimization of low-entropy tokens as a simple and effective approach to preference learning.

## 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.

## REFERENCES

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

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. 1

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, et al. UltraFeedback: Boosting language models with scaled ai feedback. In *ICML*, 2024. 6

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024. 7

Yann Dubois, Balazs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. Length-controlled AlpacaEval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024. 7

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. KTO: Model alignment as prospect theoretic optimization. In *ICML*, 2024. 2

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 6

Chengyu Huang, Zhuohang Li, Sheng-Yen Chou, and Claire Cardie. Token-weighted direct preference optimization with attention. *arXiv preprint arXiv:2605.21883*, 2026. 1, 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*, 2025. 1, 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. *arXiv preprint arXiv:2406.11939*, 2024. 7

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. 1, 3

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *NeurIPS*, volume 37, 2024. 2

Burt Monroe, Michael Colaresi, and Kevin Quinn. Fighting words: A statistical method for identifying ideology in language. *Political Analysis*, 16(4):372–403, 2008. 20

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

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022. 1, 2

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*, 2023. 1, 2

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD*, pp. 3505–3506, 2020. 7

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 (3–4):379–423, 623–656, 1948. 3

Leandro von Werra et al. Trl: Transformer reinforcement learning. *GitHub repository*, 2022. 18

Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. Offline reinforcement learning for LLM multi-step reasoning. In *Findings of ACL*, pp. 8881–8893, 2025a. 1

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, 2025b. 19

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. 2

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

Kailai Yang, Zhiwei Liu, Qianqian Xie, Jimin Huang, Erxue Min, and Sophia Ananiadou. Selective preference optimization via token-level reward function estimation. In *EMNLP*, pp. 7032–7056, 2025. 1, 3

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. 1, 3

Qiying Yu, Zheng Zhang, Ruofei Zhu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. 19

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *NeurIPS*, volume 33, pp. 5824–5836, 2020. 23

Yongcheng Zeng, Guoqing Liu, Weiyu Ma, Ning Yang, Haifeng Zhang, and Jun Wang. Token-level direct preference optimization. In *ICML*, 2024. 1, 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. 7

# Part I

# Appendix

## Table of Contents


**A Mathematical Analysis** . . . . . **13**

A.1 Gradient of the Selected-Token Objective . . . . . 13

A.2 Normalization and the Exact Gradient Decomposition . . . . . 14

A.3 Correctness of the Gradient-Preserving Control . . . . . 15

A.4 Gradient Conflict and Local Descent . . . . . 15

**B Implementation Details** . . . . . **17**

B.1 Models and Data . . . . . 17

B.2 LES-DPO Implementation . . . . . 17

B.3 Training Configuration . . . . . 17

B.4 Baselines . . . . . 18

B.5 Hyperparameter Selection . . . . . 18

B.6 Compute and Software . . . . . 19

B.7 Evaluation Protocol . . . . . 19

**C Relation to High-Entropy Token Selection in RLVR** . . . . . **19**

**D How Policy-Low20 Selection Changes During Training** . . . . . **20**

**E Token Composition Across Entropy Regions** . . . . . **20**

**F Additional Gradient Diagnostics** . . . . . **22**

F.1 Across Entropy Quintiles . . . . . 22

F.2 Full Comparisons within the Lowest-Entropy 50% . . . . . 23

F.3 Gradient Conflict between Low20 and Rest80 . . . . . 23

**G Activation Functions for Entropy-Based Token Weighting** . . . . . **25**

G.1 Over All Response Tokens . . . . . 25

G.2 Within a Fixed Low-Entropy 20% Support . . . . . 25

**H Qualitative Comparison of Model Outputs** . . . . . **26**

H.1 Example 1 . . . . . 26

H.2 Example 2 . . . . . 28

# A Mathematical Analysis

We first derive the gradient of the selected-token objective, then establish the exact relationship between mixed weighting and the gradient-preserving control, and finally use a local descent analysis to interpret gradient conflict. The analysis conditions on the current selection masks. All gradients are taken with respect to the trainable parameters, which are the LoRA parameters in our experiments. The reference model is fixed, $\beta > 0$, and $\lambda \geq 0$. Any randomness in the forward pass is held fixed for each gradient identity. These derivations characterize when additional tokens can interfere with an existing update. They do not assume or prove that low-entropy regions satisfy these conditions, nor that 20% is universally optimal; those questions are evaluated empirically.

## A.1 Gradient of the Selected-Token Objective

For one preference pair, use the notation $\delta_t^z = \delta_t(x, y_z)$ for side $z \in \{w, l\}$. At the current parameters $\theta_0$, let $\mathcal{S}^z = \mathcal{S}_\rho(x, y_z)$ be the selected support. We hold both sets fixed during the backward pass. To analyze normalization, it is useful to separate the scale $\beta$ from the pairwise margin. Define

$$M_S = \frac{1}{|\mathcal{S}^w|} \sum_{t \in \mathcal{S}^w} \delta_t^w - \frac{1}{|\mathcal{S}^l|} \sum_{t \in \mathcal{S}^l} \delta_t^l. \tag{15}$$

Then $\beta M_S = \hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l) = u$, with $u$ as in the main text. In this appendix, we write $\ell(m) = -\log \sigma(\beta m)$ for the loss as a function of an unscaled margin, so the selected-token loss is $\ell(M_S)$. The detached weights in Eq. (9) give

$$\omega_t^z = \mathrm{sg}\left[\frac{\mathbf{1}\{t \in \mathcal{S}^z\}}{|\mathcal{S}^z|}\right], \qquad M_S = \sum_{t \in \mathcal{I}(y_w)} \omega_t^w \delta_t^w - \sum_{t \in \mathcal{I}(y_l)} \omega_t^l \delta_t^l. \tag{16}$$

The weights sum to one on each side, and $\nabla_\theta \omega_t^z = 0$ during backpropagation. This also covers fallback by taking $\mathcal{S}^z = \mathcal{I}(y_z)$.

**Gradient identity for fixed selection sets.** For nonempty selection sets, expanding the weights in Eq. (13) yields

$$\begin{aligned} \nabla_\theta \ell(M_S) = -\beta \sigma(-\beta M_S) \Bigg[ & \frac{1}{|\mathcal{S}^w|} \sum_{t \in \mathcal{S}^w} \nabla_\theta \log \pi_\theta(y_{w,t} \mid x, y_{w,<t}) \\ & - \frac{1}{|\mathcal{S}^l|} \sum_{t \in \mathcal{S}^l} \nabla_\theta \log \pi_\theta(y_{l,t} \mid x, y_{l,<t}) \Bigg]. \end{aligned} \tag{17}$$

This follows directly from $\ell'(m) = -\beta \sigma(-\beta m)$ and the chain rule. The reference model receives no gradient, and both the masks and set sizes are fixed during the backward pass. Averaging over preference pairs gives the batch gradient.

Suppose there is a strict entropy gap between selected and unselected tokens, the entropy at each position is continuous in the parameters, and the fallback condition does not switch within a neighborhood. Then the selected sets remain unchanged in a sufficiently small neighborhood of $\theta_0$, and the expression above is also the ordinary gradient of the dynamically selected objective within that neighborhood. At ranking boundaries, however, the selection can change discontinuously. The implementation computes the gradient of the currently selected branch and does not imply that the full dynamic objective is smooth everywhere. For short responses or nearly zero entropy variance, the same identity applies to the selection sets produced by the fallback rule.

**Interpretation.** The target log-probabilities at unselected positions do not enter the loss as independent summands, but those tokens can still serve as context for later selected positions and thereby affect the forward pass and gradient indirectly. Selecting 20% of tokens therefore does not reduce all computation to 20%. Moreover, setting the selected fraction to 100% yields the *mean* log-ratio over response tokens, whereas standard DPO in Eq. (3) uses a *sum*. They are generally not equivalent under the same $\beta$. Let $T_w$ and $T_l$ be the valid response lengths and let $A_w$ and $A_l$ denote their summed log-ratios. At full support,

$$M_{\text{mean}} = \frac{A_w}{T_w} - \frac{A_l}{T_l}, \qquad M_{\text{sum}} = A_w - A_l. \tag{18}$$

For equal lengths $T_w = T_l = T$, the two losses coincide when $\beta_{\text{mean}} = T\beta_{\text{sum}}$. A single rescaling works for arbitrary response scores across the dataset only if this common length is fixed across pairs. Unequal lengths rescale the two sides differently. To attribute performance changes to support selection, one should therefore compare full-token, random-selection, and low-entropy objectives under the same mean normalization and $\beta$, with all other training settings held fixed. A comparison with sum-based standard DPO alone cannot rule out the effect of normalization.

### A.2 Normalization and the Exact Gradient Decomposition

On each response side $z \in \{w, l\}$, let $L^z$ and $R^z$ denote Low20 and Rest80, with token counts $n_L^z, n_R^z > 0$. All quantities below refer to one preference pair unless a batch average is explicitly shown. Define their mean scores and the normalized Low20 weight as

$$s_L^z = \frac{1}{n_L^z} \sum_{t \in L^z} \delta_t^z, \qquad s_R^z = \frac{1}{n_R^z} \sum_{t \in R^z} \delta_t^z, \qquad a_z = \frac{n_L^z}{n_L^z + \lambda n_R^z}. \tag{19}$$

**Proposition A.1** (Exact decomposition of score weights and loss gradients)**.** *Under Normalized weighting, the response scores and preference margin satisfy*

$$s_\lambda^z = \frac{n_L^z s_L^z + \lambda n_R^z s_R^z}{n_L^z + \lambda n_R^z} = a_z s_L^z + (1 - a_z) s_R^z, \tag{20}$$

$$M_\lambda = a_w s_L^w - a_l s_L^l + (1 - a_w) s_R^w - (1 - a_l) s_R^l. \tag{21}$$

*Let $c_\lambda = \beta\sigma(-\beta M_\lambda) > 0$. The loss gradient decomposes exactly as*

$$\begin{aligned} g_\lambda &= g_{L|\lambda} + g_{R|\lambda}, \\ g_{L|\lambda} &= -c_\lambda (a_w \nabla s_L^w - a_l \nabla s_L^l), \\ g_{R|\lambda} &= -c_\lambda ((1 - a_w) \nabla s_R^w - (1 - a_l) \nabla s_R^l). \end{aligned} \tag{22}$$

*Proof.* Divide the weighted token sum on each side by $n_L^z + \lambda n_R^z$ and collect the $s_L^z$ and $s_R^z$ terms to obtain the response score. Subtracting the two sides gives $M_\lambda$, and applying the chain rule to $\ell(M_\lambda)$ yields the gradient decomposition. The masks and $a_z$ are fixed during the current backward pass. □

When exactly 20% of the tokens on each side belong to Low20, $a_w = a_l = a = 1/(1 + 4\lambda)$. At $\lambda = 0.05$, $a = 5/6 \approx 83.3\%$, so Rest80 receives approximately 16.7% of the response-score weight. These percentages refer specifically to the *weight within the response score*. Let $M_0 = s_L^w - s_L^l$, $c_0 = \beta\sigma(-\beta M_0)$, and $g_0 = \nabla\ell(M_0)$. Then

$$g_{L|\lambda} = a \frac{c_\lambda}{c_0} g_0. \tag{23}$$

Thus, the Low20 gradient is not necessarily scaled to exactly 83.3% of its original value. Gradient scaling matches score-weight scaling only when $c_\lambda = c_0$. Because the mixed margin changes the nonlinear DPO coefficient, this coefficient can even offset the reduction in score weight. Within a batch, the coefficients also vary across examples, so the scaling of the full batch gradient generally cannot be summarized by one scalar. If rounding causes the actual fractions to differ between sides, $a_w$ and $a_l$ must be retained separately rather than forced to be equal. If fallback leaves Rest80 empty on one side, set $a_z = 1$ and omit $s_R^z$ rather than forming an empty-set mean. That response score is $s_L^z$ and its Rest80 gradient component is zero.

**Independent-gradient diagnostics versus the actual mixed component.** The Rest80 gradient summarized in Table 10 comes from the standalone objective $\ell(M_R)$, where $M_R = s_R^w - s_R^l$. Define

$$g_R^{\text{sep}} = \nabla\ell(M_R) = -c_R(\nabla s_R^w - \nabla s_R^l), \qquad c_R = \beta\sigma(-\beta M_R). \tag{24}$$

This is different from the actual component $g_{R|\lambda}$ in the mixed objective. For a *single preference pair*, if $a_w = a_l = a$, then Eq. (22) directly gives

$$g_{R|\lambda} = \kappa g_R^{\text{sep}}, \qquad \kappa = (1 - a)\frac{c_\lambda}{c_R}. \tag{25}$$

When $\lambda > 0$ and both groups are nonempty, $\kappa > 0$, so the two gradients are aligned for that example but differ in norm by a factor of $\kappa$. When $\lambda = 0$, the actual Rest80 component is zero. If rounding gives different $a_z$ values on the two sides, even the per-example scalar conversion generally fails and the exact decomposition must be used.

Even when $a_z$ is equal on both sides of every example, *the per-example conversion does not carry over directly after batch aggregation*. For a batch of $N$ preference pairs,

$$\bar{g}_{R|\lambda} = \frac{1}{N}\sum_{i=1}^{N} \kappa_i g_{R,i}^{\text{sep}}, \qquad \bar{g}_R^{\text{sep}} = \frac{1}{N}\sum_{i=1}^{N} g_{R,i}^{\text{sep}}. \tag{26}$$

Because $\kappa_i$ varies across examples, the two batch gradients are generally no longer aligned. The norm ratio and cosine similarity of the standalone objectives describe differences between the two supervision signals, but cannot replace statistics of the actual mixed component. Figure 10 directly computes the gradient cosine between the mixed objective and the Low20-only objective, showing how the overall direction changes with the diagnostic weight. A quantitative test of the local descent condition below should compute $g_0$ and $g_{R|\lambda}$ at the same parameters, on the same batch, and with the same masks, rather than substituting the standalone Rest80-only gradient.

### A.3 Correctness of the Gradient-Preserving Control

Using the same stop-gradient convention as in the main text, define a mixed margin in which only the Rest80 scores receive gradients:

$$\widetilde{M}_\lambda = a_w \text{sg}[s_L^w] - a_l \text{sg}[s_L^l] + (1 - a_w)s_R^w - (1 - a_l)s_R^l. \tag{27}$$

**Proposition A.2** (Exact preservation of the current Low20-only gradient)**.** *Consider the loss represented by the following computation graph:*

$$\ell_{\text{ctrl}} = \ell(M_0) + \ell(\widetilde{M}_\lambda) - \text{sg}\Big[\ell(\widetilde{M}_\lambda)\Big] \tag{28}$$

*Its forward value equals $\ell(M_0)$, while automatic differentiation returns*

$$g_{\text{ctrl}} = g_0 + g_{R|\lambda}. \tag{29}$$

*Proof.* The second and third terms of the loss have identical forward values and cancel, while the third term receives no gradient. Moreover, $\widetilde{M}_\lambda$ and $M_\lambda$ have the same numerical value, so the DPO coefficient remains $c_\lambda$, but $s_L^w$ and $s_L^l$ are detached from this branch of the computation graph. The gradient of the second term is therefore exactly $g_{R|\lambda}$ from Eq. (22), whereas the first term produces the complete $g_0$. The identity remains valid after averaging over examples. □

Thus, at the *same parameter state, on the same examples, and with the same masks*, the Gradient-preserving control retains the Low20-only gradient and adds the Rest80 gradient component induced by Normalized weighting. It is not equivalent to simply using $\ell(M_0) + \lambda\ell(s_R^w - s_R^l)$, because the latter Rest80 term has a different normalization and DPO coefficient. This construction is a computation graph designed to produce a specified first-order gradient; its canceled forward value should not be interpreted as an ordinary scalar function with the same derivative.

Independently trained runs reach different parameter states, so "gradient preservation" refers to this local construction at each current step; it does not imply that the full training trajectory matches Low20 only. Actual AdamW updates also depend on optimizer momentum, preconditioning, and weight decay. Preserving a gradient component therefore does not preserve every element of the parameter-update vector. This control helps test the effect of adding Rest80, but differences in final metrics alone cannot precisely identify the causal contribution of each mechanism.

### A.4 Gradient Conflict and Local Descent

Let $F(\theta)$ be the Low20 batch loss with the current masks fixed, let $g = \nabla F(\theta) \neq 0$, and let $h$ be an additional gradient component. Consider the direction $d = ag + bh$, where $a > 0$ and $b \geq 0$. This includes the gradient-preserving control with $a = b = 1$ and $h = g_{R|\lambda}$. Here $a$ and $b$ are direction coefficients, distinct from the response-score weights $a_z$. For general Normalized weighting, one must first use the exact decomposition in Appendix A.2; the response-score weights cannot be used directly as the batch-level coefficients $a$ and $b$.

**Local descent analysis.** The following applies standard gradient-geometry identities and the smooth descent bound, without assuming an intrinsic relationship between entropy and gradient direction. If $h \neq 0$, let $r = \|h\|/\|g\|$ and $q = \langle g, h\rangle/(\|g\|\|h\|)$. Then

$$\langle g, d\rangle = \|g\|^2(a + brq), \tag{30}$$

$$\|d\|^2 = \|g\|^2(a^2 + 2abrq + b^2r^2), \tag{31}$$

$$\cos(g, d) = \frac{a + brq}{\sqrt{a^2 + 2abrq + b^2r^2}} \quad (d \neq 0). \tag{32}$$

If $F$ has a $K$-Lipschitz gradient with $K > 0$ along the update segment, then the update $\theta' = \theta - \eta d$ satisfies

$$F(\theta') \leq F(\theta) - \eta\|g\|^2(a + brq) + \frac{K\eta^2}{2}\|g\|^2(a^2 + 2abrq + b^2r^2). \tag{33}$$

Therefore, when $a + brq > 0$, the following step-size condition is sufficient for strict descent:

$$0 < \eta < \frac{2(a + brq)}{K(a^2 + 2abrq + b^2r^2)}. \tag{34}$$

If $h = 0$, then $d = ag$ and strict descent instead follows from $0 < \eta < 2/(Ka)$. If $d = 0$, the parameters do not change and its cosine is undefined.

*Proof.* Substituting $d = ag + bh$ into the definitions of inner product, norm, and cosine gives the first three identities. Lipschitz continuity of the gradient and integration along the update segment yield $F(\theta - \eta d) \leq F(\theta) - \eta\langle g, d\rangle + K\eta^2\|d\|^2/2$. Substituting the first two identities and requiring the right-hand-side change to be negative gives the stated condition. □

**Interpretation.** For $b > 0$, when $q > 0$, the added component helps reduce the Low20 loss to first order. When $q = 0$, it adds no descent projection for Low20 but increases the update norm and changes its direction. When $q < 0$, it cancels part of the descent projection. Only when $a + brq < 0$ does the combined direction become a first-order ascent direction for Low20, in which case a sufficiently small positive step increases its loss. A negative cosine therefore does not mean that the entire training step is necessarily harmful, but a larger norm ratio $r$ amplifies the influence of an opposing or orthogonal component. This is why gradient norms, cosine similarities, and conflict rates should be considered together.

**Direct corollary for the gradient-preserving control.** For $d = g_0 + g_{R|\lambda}$ with both components nonzero, let $r = \|g_{R|\lambda}\|/\|g_0\|$ and $q = \cos(g_0, g_{R|\lambda})$. Then

$$\frac{\mathrm{d}}{\mathrm{d}\eta}F(\theta - \eta d)\bigg|_{\eta=0} = -\|g_0\|^2(1 + rq). \tag{35}$$

Thus, $q < 0$ means only that the added component cancels part of the Low20 descent projection. The combined direction increases the Low20 loss to first order only when $1 + rq < 0$, equivalently $q < -1/r$. If $r < 1$, the added component cannot reverse this first-order descent direction regardless of its orientation. When $1 + rq = 0$, the first-order change is zero and higher-order terms must be examined. This distinction prevents interpreting the "percentage of batches with negative cosine" as the "percentage of batches whose total training loss increases."

These conclusions apply to fixed masks and ordinary gradient steps. For a fixed symmetric positive-definite preconditioner $P$, the first-order effect of the update $-\eta Pd$ depends on $g^\top Pd$, not solely on Euclidean cosine similarity. Actual training also updates the masks and accumulates optimizer state. Moreover, the median norm ratio, mean cosine similarity, and batch conflict rate are distinct statistics and cannot be combined into a synthetic batch and substituted into these formulas. The derivations give a local interpretation of the gradient diagnostics: directional conflict can weaken descent on an existing objective under particular magnitude and step-size conditions, but local descent of the Low20 loss does not itself guarantee higher evaluation scores. Our fraction sweep and training controls provide empirical support, not a mathematical guarantee of cross-model generalization or of a universally optimal 20% fraction.

# B Implementation Details

This appendix provides the implementation details for training and evaluation. Unless otherwise stated, all methods use the same data, preprocessing pipeline, number of training epochs, and LoRA configuration.

## B.1 Models and Data

We use Qwen2.5-3B-Instruct, Qwen2.5-7B-Instruct, Meta-Llama-3.1-8B, Meta-Llama-3.1-8B-Instruct, Mistral-7B-v0.1, and Mistral-7B-Instruct-v0.1. Training uses the cleaned UltraFeedback Binarized dataset, which contains 60,829 preference pairs $(x, y_w, y_l)$. DPO and its variants use both the chosen and rejected responses. SFT uses the same prompts but treats only the chosen responses as supervision targets.

Table 4: Training data and preprocessing settings shared by all methods.

| Category | Setting | Value |
|---|---|---|
| **Training data** | Dataset | UltraFeedback Binarized (cleaned) |
| | Preference pairs | 60,829 |
| | Record fields | `prompt`, `chosen`, `rejected` |
| | DPO supervision | Chosen and rejected responses |
| | SFT supervision | Chosen response only |
| **Preprocessing** | Maximum length | 1,024 tokens |
| | Padding / truncation | Right / right |
| | Training labels | Response tokens only |
| | Instruct formatting | Native chat template |
| | Base-model formatting | Raw prompt and continuation |

For Qwen2.5 instruction-tuned models, we additionally prepend the system message "You are a helpful assistant." The chosen and rejected responses use identical tokenization and truncation rules.

## B.2 LES-DPO Implementation

We provide implementation details beyond the formal objective in §4. Token entropy is computed in float32 from the logits produced by the current policy forward pass, improving numerical stability when ranking the lowest-entropy positions. Tokens are ranked independently within each chosen and rejected response. The resulting selection masks are detached from the computation graph with stop-gradient, so entropy is used only to select positions. The policy and reference model share the same frozen base weights. For the reference forward pass, we temporarily disable the LoRA adapter and run inference without gradients rather than loading a separate copy of the reference model. If either response side contains fewer than eight valid tokens, or if the entropy variance over valid positions is below $10^{-12}$, that side falls back to all valid tokens. This prevents unstable top-$k$ selection for extremely short or degenerate responses. The same implementation supports High, Random, and non-overlapping interval selection for the ablations and interval analyses in §5.3. The final method uses current-policy entropy, Low/Low selection on both sides, and a 20% fraction.

## B.3 Training Configuration

All models are trained with LoRA. Table 5 lists the shared training configuration, and Table 6 gives the objective-specific parameters for each preference optimization method.

Table 5: Shared training and system configuration.

| Category | Hyperparameter | Value |
|---|---|---|
| Optimization | Optimizer | AdamW |
| | Preference learning rate | $5 \times 10^{-6}$ |
| | SFT learning rate | $2 \times 10^{-4}$ |
| | Learning-rate scheduler | Cosine |
| | Warmup ratio | 0.1 |
| | Epoch | 1 |
| | Weight decay | 0 |
| | Maximum gradient norm | 1.0 |
| | Per-device micro-batch (3B ablation) | 8 |
| | Gradient accumulation (3B ablation) | 2 |
| | **Global batch size (3B ablation)** | **32** |
| LoRA | Rank $r$ | 32 |
| | Scaling $\alpha$ | 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 | 8× NVIDIA A800-SXM4 80GB |
| | Precision | `bfloat16` |
| | Maximum sequence length | 1,024 |
| | Attention implementation | FlashAttention-2 |
| | Distributed strategy | DeepSpeed ZeRO-2 |
| | Random seed | 42 |

Table 6: Objective-specific parameters for preference optimization methods.

| Method | $\beta$ | Additional parameters |
|---|---|---|
| DPO | 0.1 | sigmoid loss; no additional parameters |
| IPO | 0.1 | squared (IPO) loss; no additional parameters |
| KTO | 0.1 | desirable / undesirable weight $= 1.0/1.0$ |
| ConfPO | 0.1 | temperature $= 1.0$ |
| CPO | 0.1 | $\alpha = 1.0$ |
| SimPO | 2.5 | $\gamma = 0.5$ |
| TDPO | 0.1 | $\alpha = 0.5$, version 2 |
| TIS-DPO | 0.1 | rank scale $[0.7, 1.3]$ |
| LES-DPO | **0.1** | policy Low/Low, fraction $=$ **0.20** |

### B.4 Baselines

All preference optimization baselines—DPO, IPO, CPO, SimPO, KTO, TDPO, TIS-DPO, and ConfPO—are implemented with TRL (von Werra et al., 2022). Their objective-specific parameters follow the recommended settings from the respective papers, as summarized in Table 6. SFT and all preference methods share the same training data, preprocessing pipeline, LoRA configuration, random seed, and number of epochs; only the objective differs. SFT uses the same prompts and treats only chosen responses as targets.

### B.5 Hyperparameter Selection

We determine the selection fraction $\rho$ empirically. §5.3 compares entropy regions and retained fractions from 10% to 90% (Figure 3) and further analyzes non-overlapping and cumulative entropy intervals (Figure 4). Both experiments favor current-policy Low/Low-20% on the main win-rate

metrics, so we set $\rho = 0.2$. We set $\beta = 0.1$ for LES-DPO, matching standard DPO, without model-specific tuning. The two objectives nevertheless average over selected tokens and sum over all tokens, respectively; using the same $\beta$ does not remove this normalization difference. Multi-seed validation under this configuration and its aggregate summary remain to be completed. For all other baselines, $\beta$ and method-specific parameters follow the settings recommended in their original papers (Table 6).

### B.6 Compute and Software

All experiments are conducted on one server with **eight NVIDIA A800-SXM4 80GB GPUs**. Multi-GPU training uses PyTorch Distributed Data Parallel and DeepSpeed ZeRO Stage 2, with one training process per GPU. Model weights and forward computation use BF16. The software stack consists of PyTorch 2.9.1, Transformers 4.56.2, PEFT 0.13.2, TRL 0.27.1, DeepSpeed 0.19.5, FlashAttention 2.8.3, and vLLM 0.15.0.

### B.7 Evaluation Protocol

Table 7: Evaluation benchmarks and reported metrics.

| Benchmark | Examples | Turns | Metric |
|---|---|---|---|
| **AlpacaEval 2** | 805 | 1 | LC (%), WR (%) |
| **Arena-Hard v0.1** | 500 | 1 | WR (%) |
| **MT-Bench** | 80 | 2 | Score (1–10) |

Candidate responses are generated with vLLM using a maximum context length of 8,192, a maximum generation length of 1,024 tokens, top-$p = 1.0$, and random seed 0. AlpacaEval 2 and Arena-Hard use temperature $= 0$ and repetition penalty $= 1.05$. MT-Bench uses category-specific temperatures and retains the complete conversation history for the second turn. Inference uses BF16, with vLLM GPU memory utilization set to 0.90.

All methods are evaluated by **GPT-4** with judge temperature 0. For Arena-Hard, each response pair is judged twice with the A/B order reversed to reduce position bias. MT-Bench scores both turns separately and averages them. AlpacaEval 2 uses the GPT-4-1106-preview reference responses.

## C Relation to High-Entropy Token Selection in RLVR

Our work differs from *Beyond the 80/20 Rule* (Wang et al., 2025b) in two main respects. First, LES-DPO is a DPO method for preference alignment: it learns from offline chosen–rejected pairs to favor preferred responses through a Bradley–Terry preference objective. The other study examines reinforcement learning with verifiable rewards (RLVR) to improve LLM reasoning, using DAPO (Yu et al., 2025), a GRPO-derived algorithm, to optimize sampled responses against verifiable outcome rewards. The two studies therefore address different training objectives and supervision signals.

Second, the selected entropy regions and their proposed explanations differ. The RLVR study retains the *highest*-entropy 20% of tokens within each batch and relates their effectiveness to branching points in reasoning, often marked by words such as "wait" and "however." LES-DPO retains the *lowest*-entropy 20% independently within each chosen and rejected response. Our explanation centers on *gradient consistency in preference learning*. Higher-entropy intervals can produce larger gradients that align weakly or conflict with the Low20 gradient; expanding the selected support can consequently redirect the preference update (§5.4; Appendix F). Together with the gradient-preserving training control (§5.5), these observations support retaining the Low20 update direction and limiting interference from additional entropy regions. This explanation concerns how token gradients interact under the preference objective, rather than whether selected tokens denote transitions in a reasoning path. Gradient consistency is an empirical interpretation of these diagnostics, not a claim that all low-entropy token gradients agree. Thus, the shared 20% fraction refers to opposite entropy regions selected for different learning objectives.

## D How Policy-Low20 Selection Changes During Training

Table 2 shows that selecting Low20 with current-policy entropy outperforms reference-model selection. To track how their selections diverge, we fix a random sample of 512 examples from the UltraFeedback preference test split and evaluate policy checkpoints at 0%, 25%, 50%, 75%, and 100% of LES-DPO training. At each checkpoint, both models independently select the lowest-entropy 20% in each chosen and rejected response.

The overlap rate is the fraction of reference-selected Low20 positions also selected by the current policy. Figure 6 reports overlap separately for chosen and rejected responses, averaged over the 512 examples.

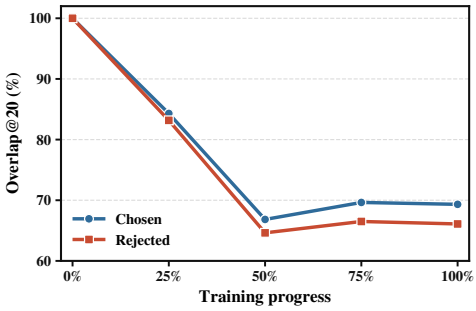
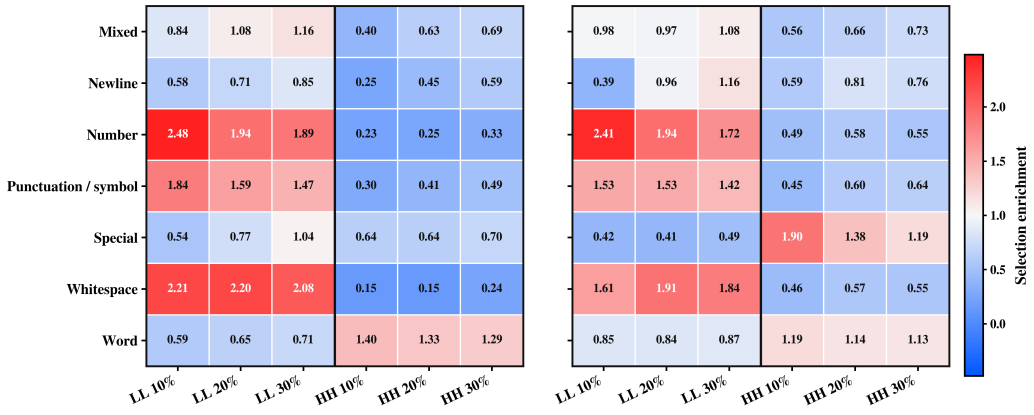



Figure 6: Mean overlap of policy- and reference-selected Low20 positions across training, averaged over 512 examples. Curves show chosen and rejected responses; 100% denotes identical selections.

**Results.** The initially identical models have 100% overlap on both response sides. At 25% progress, chosen and rejected overlap falls to 84.31% and 83.17%, respectively, reaching 69.32% and 66.09% by the end of training. Thus, 30.68% of the chosen positions and 33.91% of the rejected positions selected by the final policy fall outside the reference selection. The policy's Low20 support therefore evolves during training. Together with the performance comparison in Table 2, this supports using current-policy entropy to track the model's evolving predictive state.

## E Token Composition Across Entropy Regions

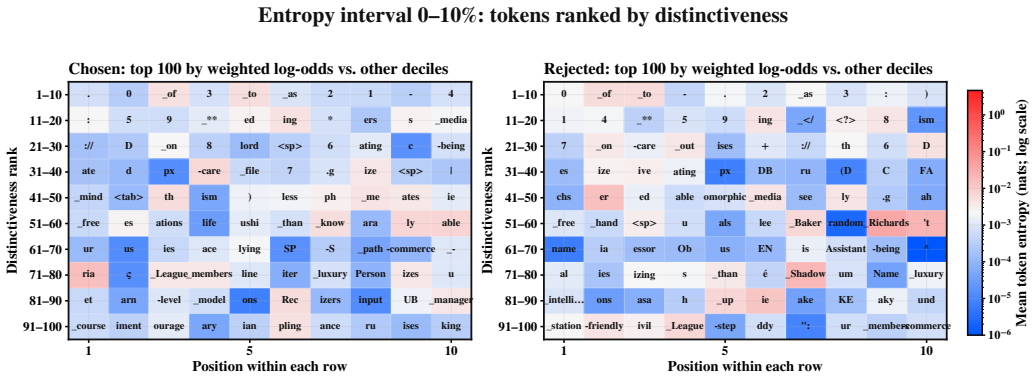



(a) Chosen (b) Rejected

Figure 7: Token-category enrichment across entropy regions. Low-entropy selection strongly enriches numbers, punctuation/symbols, and whitespace, while ordinary words become more common as the fraction expands from 10% to 30%. High-entropy selection primarily enriches semantic words and suppresses most structural categories.

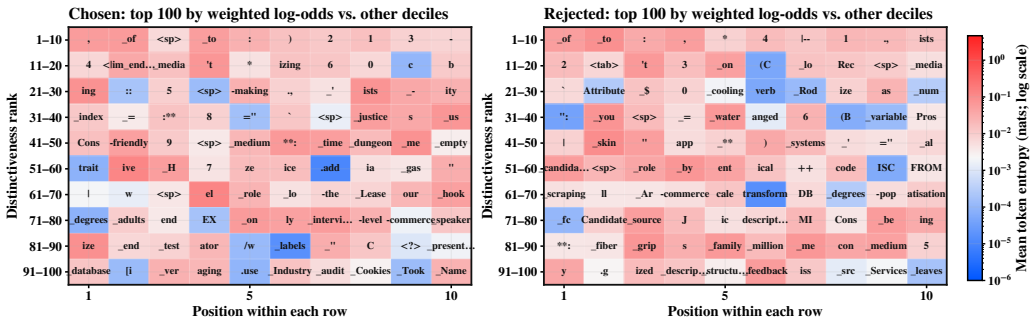
To characterize the tokens contained in different entropy regions, we rank the chosen and rejected tokens in 500 preference examples by current-policy predictive entropy. Figure 7 compares the token categories and positions selected by Low/Low and High/High at fractions of 10%, 20%, and 30%; an enrichment factor above 1 indicates that a category or position is selected more often than expected. Figure 8 further divides valid tokens into ten non-overlapping entropy deciles and visualizes the first six (0–60%). We identify the most distinctive tokens in each interval with the weighted log-odds ratio (Monroe et al., 2008); larger $z$ values indicate greater representativeness, and color encodes mean predictive entropy.

Low-entropy regions are enriched in punctuation, numbers, whitespace, Markdown markers, and common function words, and they occur more often near the beginning of a response. Across the displayed 0–60% intervals, content words, discourse connectives, and sentence- or clause-initial tokens become more common as entropy increases. Chosen and rejected responses have similar token composition in the lowest-entropy regions, with differences emerging in the intermediate regions. These findings show that entropy regions have different linguistic compositions, but token type alone does not explain the training gains; the optimization behavior must also be considered through gradient diagnostics.



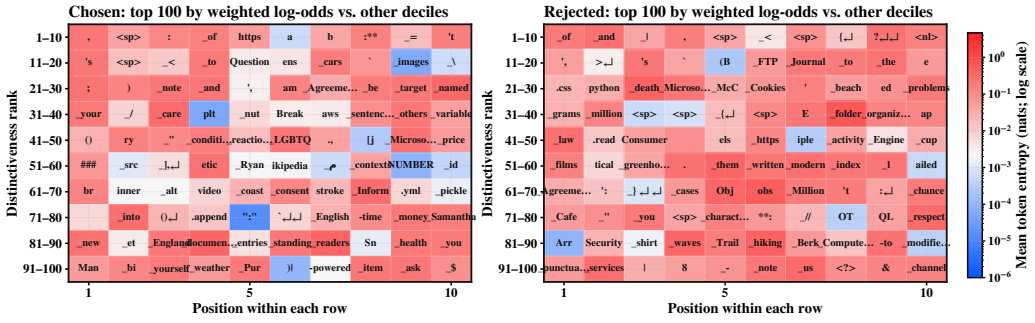

(a)

Figure 8: Distinctive tokens across six entropy deciles (0–60%). Panel (a) shows the 0–10% interval. Each panel shows 100 Chosen and 100 Rejected token types, ranked by weighted log-odds $z$ scores against the other nine deciles. Color shows mean entropy on a shared logarithmic scale.



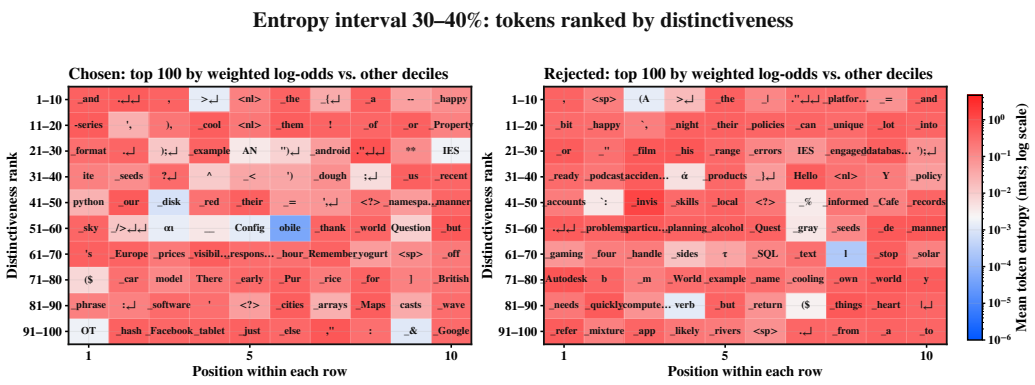

(b)

Figure 8: Distinctive tokens (continued). Panel (b) shows the 10–20% entropy interval.



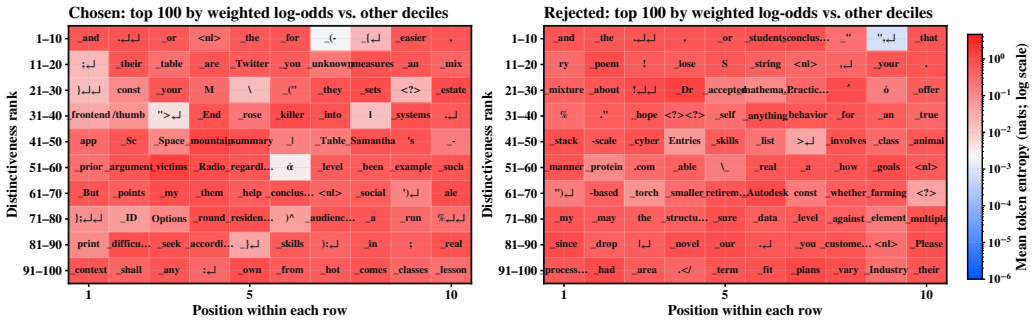

(c)

Figure 8: Distinctive tokens (continued). Panel (c) shows the 20–30% entropy interval.



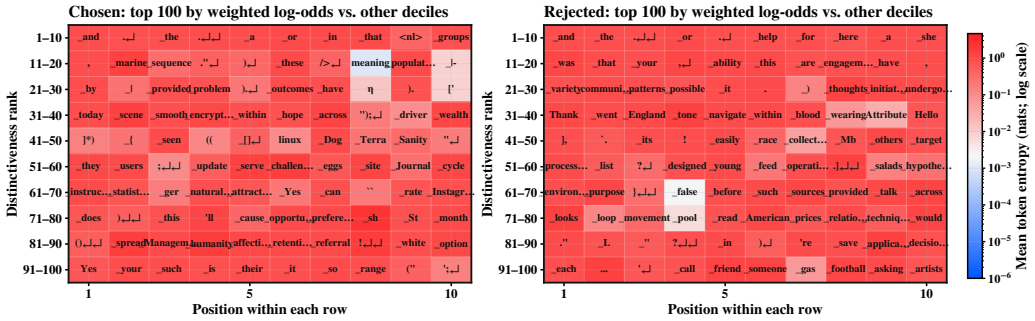

(d)

Figure 8: Distinctive tokens (continued). Panel (d) shows the 30–40% entropy interval.




(e)

Figure 8: Distinctive tokens (continued). Panel (e) shows the 40–50% entropy interval.




(f)

Figure 8: Distinctive tokens (continued). Panel (f) shows the 50–60% entropy interval.

# F Additional Gradient Diagnostics

## F.1 Across Entropy Quintiles

Using the Qwen2.5-3B-Instruct initialization without a preference-training adapter, we sample 100 preference examples from UltraFeedback. We independently divide chosen and rejected tokens into five equal-sized intervals ordered by current-policy entropy: Q0 (0–20%), Q1 (20–40%), Q2 (40–60%), Q3 (60–80%), and Q4 (80–100%). We compute the DPO gradient for each interval indepen-

dently using the same selected-token mean normalization, record only LoRA parameter gradients, and do not update the model.

Table 8: Gradients of higher-entropy intervals relative to Low20. Norm ratios are per-example medians; cosine similarities are means. Conflict rate is the percentage of batches with negative cosine.

| **Entropy interval** | **Norm ratio** | **Mean cosine** | **Conflict rate** |
|---|---|---|---|
| 20–40% | 3.978× | 0.016 | 41% |
| 40–60% | 4.990× | 0.011 | 44% |
| 60–80% | 6.565× | −0.014 | 50% |
| 80–100% | 7.138× | 0.009 | 48% |

Table 8 compares each higher-entropy interval with Q0, the lowest-entropy 20%. The intervals contain approximately equal numbers of tokens and share the same selected-token mean normalization. Negative cosine indicates opposing gradient directions at the fixed model state; conflict rate does not measure how often the total training loss rises.

Table 8 shows that Q1–Q4 gradients are 3.978–7.138 times larger than the Low20 gradient, yet their mean cosine similarities with Low20 range only from $-0.014$ to $0.016$, and 41%–50% of batches exhibit directional conflict. The remaining 80% of tokens are therefore not uniformly harmful; instead, their independent objectives produce stronger gradients with mixed directions. This observation is consistent with the explanation that adding higher-entropy regions can interfere with Low20. However, the norm ratio of independently computed objectives alone cannot determine the amount of interference during mixed training, which also depends on normalization and the DPO coefficient; see Appendix A.2.

### F.2 Full Comparisons within the Lowest-Entropy 50%

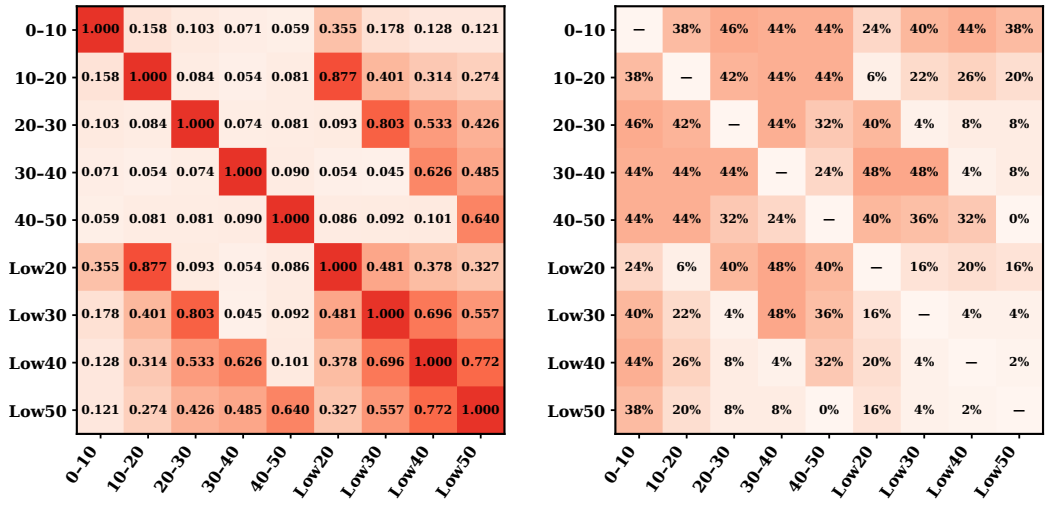
Figure 9 reports all pairwise comparisons among the nine supports within the lowest-entropy 50% shown in Figure 5. Figure 9a shows mean gradient cosine, and Figure 9b shows conflict percentages. The five disjoint intervals span 0–10% through 40–50%; Low20, Low30, Low40, and Low50 denote cumulative supports. All entries use the same 50 preference pairs and the same fixed Qwen2.5-3B-Instruct model. Gradients are measured through newly initialized LoRA adapters with rank 32 and $\alpha = 64$, without parameter updates. Boxes in the main figure show quartiles and medians, with whiskers extending to observations within 1.5 interquartile ranges. Short or constant-entropy responses retain the original script's full-response fallback.

### F.3 Gradient Conflict between Low20 and Rest80

We diagnose Low20 and Rest80 gradients on the same 100 batches at the Qwen2.5-3B-Instruct checkpoint after supervised fine-tuning (SFT), the Low20-only checkpoint, and three Normalized-weighting checkpoints, without updating model parameters. The SFT checkpoint is the initial model state for this analysis. Diagnostics use $\beta = 0.3$ and the same 100 preference pairs at each checkpoint. Here $\lambda_{\text{train}}$ denotes the Rest80 training weight and $\lambda_{\text{eval}}$ denotes the diagnostic mixing weight. Table 10 compares gradients computed separately from Low20 and Rest80 within each model. Each table row reports the median Rest80-to-Low20 gradient-norm ratio, mean cosine similarity, and percentage of batches with negative cosine for one checkpoint. Negative cosine denotes opposing gradient directions (Yu et al., 2020).

Table 9 reports the median cosine between the Low20 gradient and gradients formed by mixing Rest80 at different weights; lower values indicate greater deviation from the Low20 direction. Figure 10 shows the underlying distributions at $\lambda_{\text{eval}} = 0.25$: despite positive medians, the mixed gradient opposes Low20 on 6–10% of batches at the trained checkpoints. This differs from the conflict rate between independent Low20 and Rest80 objectives in Table 10.

At the trained checkpoints, 20–26% of batches exhibit gradient conflict and the median Rest80-to-Low20 norm ratio is 1.239–1.573. Increasing the Rest80 weight further reduces alignment with Low20. Thus, Rest80 produces strong but inconsistently directed gradients. Together with Table 3, this shows that adding Rest80 can interfere with the Low20 update, rather than that all Rest80 tokens




(a) Mean gradient cosine

(b) Conflicting batches

Figure 9: Pairwise gradient diagnostics for nine low-entropy supports. Cells report mean cosine similarity and the percentage of batches with negative cosine, respectively. Conflict self-comparisons are omitted.

| Checkpoint | Median cosine, $\lambda_{\text{eval}}$ | | | Cos $< 0$ |
|---|---|---|---|---|
| | 0.05 | 0.10 | 0.25 | |
| SFT | 0.954 | 0.853 | 0.585 | 18% |
| Low20 only | 0.982 | 0.946 | 0.813 | 7% |
| $\lambda_{\text{train}} = 0.05$ | 0.982 | 0.940 | 0.801 | 10% |
| $\lambda_{\text{train}} = 0.10$ | 0.981 | 0.938 | 0.775 | 9% |
| $\lambda_{\text{train}} = 0.25$ | 0.979 | 0.937 | 0.778 | 6% |

Table 9: Median alignment of mixed gradients with Low20 across diagnostic Rest80 weights. The last column reports the percentage of negative cosines at $\lambda_{\text{eval}} = 0.25$.

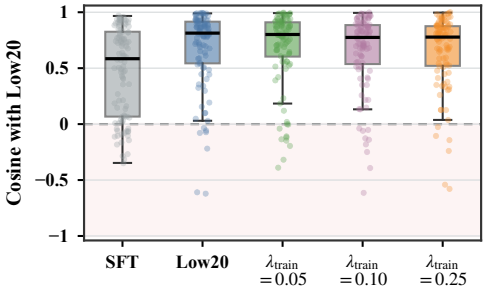



Figure 10: Mixed-gradient cosines with Low20 at $\lambda_{\text{eval}} = 0.25$. Boxes show quartiles and medians; whiskers follow the 1.5-IQR rule. Dots show all 100 batches per checkpoint; shading marks negative cosine.

are harmful. The exact relationship between the independent Rest80 gradient and its mixed-objective contribution is derived in Appendix A.2.

Table 10: Rest80 versus Low20 within each model: median norm ratio, mean cosine, and conflict percentage.

| **Model state** | **Norm ratio** | **Mean cosine** | **Conflict rate** |
|---|---|---|---|
| SFT | 1.65× | 0.06 | 29% |
| Low20 only | 1.44× | 0.31 | 23% |
| $\lambda_{\text{train}} = 0.05$ | 1.57× | 0.34 | 21% |
| $\lambda_{\text{train}} = 0.10$ | 1.49× | 0.28 | 26% |
| $\lambda_{\text{train}} = 0.25$ | 1.24× | 0.28 | 20% |

# G Activation Functions for Entropy-Based Token Weighting

## G.1 Over All Response Tokens

We compare six gating functions. Uniform assigns equal weight to every token. Linear, Power, Sigmoid, and Softmax continuously increase the weight of lower-entropy tokens based on current-policy entropy. Top-$k$ retains only the lowest-entropy 20%. Table 11 lists the gate parameters, and Table 12 reports results on Qwen2.5-3B-Instruct.

Table 11: Full-response token gates and their implementation parameters. Linear, Power, and Softmax do not use a fixed $\alpha$; the implementation performs 24 search iterations over $[0, 64]$ for each response.

| **Gate** | **Weighting behavior** | **Parameter** | **Active support** |
|---|---|---|---|
| Uniform | Equal weight for every valid token | None | All tokens |
| Linear | Linear emphasis on lower entropy | $\alpha$: searched per response | All tokens |
| Power | Power weighting of low-entropy scores | $\alpha$: searched per response | All tokens |
| Sigmoid | Smooth emphasis on lower entropy | $\gamma = 3.0$ | All tokens |
| Softmax | Exponential weighting of low-entropy scores | $\alpha$: searched per response | All tokens |
| Top-$k$ | Binary selection of lowest-entropy tokens | $\rho = 0.20$ | Lowest-entropy 20% |

Table 12: Results for different activation gates over all response tokens. Linear, Power, and Softmax parameters are calibrated per response. Top-$k$ selects the lowest-entropy 20% and corresponds to our method. Bold indicates the best result in each column.

| Gate | AlpacaEval 2 | | Arena-Hard | MT-Bench |
|---|---|---|---|---|
| | LC (%) | WR (%) | WR (%) | Score |
| Linear | 26.232 | 24.099 | 29.832 | 8.069 |
| Power | 28.905 | 25.714 | 28.817 | 8.075 |
| Sigmoid | 28.765 | 24.907 | 26.707 | 8.081 |
| Softmax | 28.547 | 25.963 | 31.473 | 7.981 |
| Top-$k$ (Low20) | **29.971** | **30.870** | **35.770** | **8.144** |
| Uniform | 24.373 | 20.000 | 27.637 | 8.037 |

## G.2 Within a Fixed Low-Entropy 20% Support

For a more focused comparison, we fix the lowest-entropy 20% support separately within each chosen and rejected response on Qwen2.5-3B-Instruct, then compare activation functions only within that support. Here, Uniform assigns equal weight to every token in Low20 and is exactly our proposed method. We omit an additional Top-$k$ gate because retaining 20% of Low20 would reduce the actual support to approximately 4% of the original response. Results are shown in Table 13.

Table 13: Results for different activation functions within a fixed lowest-entropy 20% support. Rest80 always receives zero weight.

| Gate | AlpacaEval 2 | | Arena-Hard | MT-Bench |
|---|---|---|---|---|
| | LC (%) | WR (%) | WR (%) | Score |
| Linear | 29.268 | 25.776 | 31.767 | **8.238** |
| Power | 26.371 | 22.671 | 31.234 | 8.156 |
| Sigmoid | 27.765 | 24.783 | 30.649 | 8.231 |
| Softmax | 26.932 | 23.913 | 30.712 | 8.094 |
| Uniform (hard Low20) | **29.971** | **30.870** | **35.770** | 8.144 |

# H Qualitative Comparison of Model Outputs

This section qualitatively compares outputs generated by different methods. All outputs come from the original Qwen2.5-7B-Instruct checkpoint or its corresponding SFT and preference-optimized checkpoints. We compare Base, SFT, DPO, IPO, CPO, KTO, SimPO, TDPO, TIS-DPO, ConfPO, and LES-DPO. To cover different task types without excessive length, we select two examples from the 500 Arena-Hard prompts for which all methods have saved responses: a short numerical-reasoning problem and a code-generation problem. We preserve each model's original English output. The examples are selected without reference to model evaluation scores and are intended only to illustrate differences in response style and reasoning; they do not replace the aggregate evaluation in the main paper.

## H.1 Example 1

**Prompt**

```
It is 1.00 o clock at night and I have to wait for 1.65 hours what time is it going to be after the wait is over?
```

**Reference checkpoints**

**Base**

```
If it is currently 1:00 AM and you need to wait for 1.65 hours, let's break down the calculation:
1.65 hours can be split into 1 hour and 0.65 hours. - 0.65 hours is equivalent to 0.65 Œ 60 minutes = 39 minutes.
So, you need to wait for 1 hour and 39 minutes.
Starting from 1:00 AM: - After 1 hour, the time will be 2:00 AM. - Adding 39 more minutes to 2:00 AM will bring the time to 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be 2:39 AM.
```

**SFT**

```
If you start waiting at 1:00 AM and wait for 1.65 hours, you will end up at 2:39 AM.
```

**Preference-optimization baselines**

**DPO**

```
If it is currently 1:00 AM and you need to wait for 1.65 hours, let's break down the calculation:
1.65 hours can be split into 1 hour and 0.65 hours. - 0.65 hours is equivalent to 0.65 Œ 60 minutes = 39 minutes.
So, you need to wait for 1 hour and 39 minutes.
Starting from 1:00 AM: - After 1 hour, the time will be 2:00 AM. - Adding 39 more minutes to 2:00 AM will bring the time to 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be 2:39 AM.
```

**IPO**

```
If it is currently 1:00 AM and you need to wait for 1.65 hours, let's break down the calculation:
1.65 hours can be split into 1 hour and 0.65 hours. - 0.65 hours is equivalent to 0.65 Œ 60 minutes = 39 minutes.
So, you need to wait for 1 hour and 39 minutes.
Starting from 1:00 AM: - After 1 hour, the time will be 2:00 AM. - Adding 39 more minutes to 2:00 AM will bring the time to 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be 2:39 AM.
```

**CPO**

If you start waiting at 1:00 AM and wait for 1.65 hours, you will end up at 2:39 AM.

**KTO**

If it's currently 1:00 AM and you need to wait for 1.65 hours, let's break down the calculation:
1 hour from 1:00 AM will bring us to 2:00 AM. 0.65 hours is equal to 0.65 * 60 minutes = 39 minutes.
So, adding 39 minutes to 2:00 AM brings us to 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be 2:39 AM.

**SimPO**

Let's break down the calculation:
- You start at 1:00 AM (or 1:00 midnight).
- You need to wait for 1.65 hours.
First, let's convert 1.65 hours into hours and minutes: - 1 hour = 60 minutes. - 0.65 hours = 0.65 * 60 minutes ≈ 39 minutes.
So, you need to wait for 1 hour and 39 minutes.
Now, add this to your starting time of 1:00 AM: - Adding 1 hour to 1:00 AM gives you 2:00 AM. - Adding 39 minutes to 2:00 AM gives you 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be **2:39 AM**.

**Token-aware methods**

**TDPO**

If it is currently 1:00 AM and you need to wait for 1.65 hours, let's break down the calculation:
1.65 hours can be split into 1 hour and 0.65 hours. - 0.65 hours is equivalent to 0.65 Œ 60 minutes = 39 minutes.
So, you need to wait for 1 hour and 39 minutes.
Starting from 1:00 AM: - After 1 hour, the time will be 2:00 AM. - Adding 39 more minutes to 2:00 AM will bring the time to 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be 2:39 AM.

**TIS-DPO**

If it's currently 1:00 AM and you need to wait for 1.65 hours, you can calculate the end time as follows:
1 hour from 1:00 AM will bring you to 2:00 AM. 0.65 hours is equivalent to 0.65 * 60 minutes = 39 minutes.
So, adding 39 minutes to 2:00 AM gives you 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be 2:39 AM.

**ConfPO**

If it is currently 1:00 AM and you need to wait for 1.65 hours, let's break down the calculation:
1.65 hours can be split into 1 hour and 0.65 hours. - 0.65 hours is equivalent to 0.65 Œ 60 minutes = 39 minutes.
So, you need to wait for 1 hour and 39 minutes.
Starting from 1:00 AM: - After 1 hour, the time will be 2:00 AM. - Adding 39 more minutes to 2:00 AM will bring the time to 2:39 AM.
Therefore, after waiting for 1.65 hours, the time will be 2:39 AM.

LES-DPO

If it is currently 1:00 o'clock at night (which is 24-hour time, or 23:00 in 24-hour format), and you need to wait for 1.65 hours, you can calculate the future time as follows:
1.65 hours is equivalent to 1 hour and 0.65 hours. 0.65 hours in minutes is $0.65 \times 60 = 39$ minutes.
So, you need to add 1 hour and 39 minutes to 1:00 o'clock.
Starting from 1:00: - Adding 1 hour brings you to 2:00. - Adding 39 minutes to 2:00 brings you to 2:39.
Therefore, after waiting for 1.65 hours, the time will be 2:39 in the morning.

### H.2 EXAMPLE 2

Prompt

Make a one line python code to get list of primes from 1 to 200 use lambda function and list comprehension

**Reference checkpoints**

Base

Certainly! Here's the one-liner using a lambda function and list comprehension to get a list of prime numbers from 1 to 200:

```
primes = [x for x in range(2, 201) if all(x % y != 0 for y in range(2, int(x**0.5) + 1))]
```

This code uses a list comprehension to filter out non-prime numbers by checking divisibility up to the square root of each number.

SFT

Here's a one-liner Python code using a lambda function and list comprehension to generate a list of prime numbers from 1 to 200:

```
primes = [n for n in range(2, 201) if all(n % d != 0 for d in range(2, int(n**0.5)+1))]
```

This code uses the all() function with a generator expression inside it to check if a number is divisible by any number between 2 and its square root (inclusive). If no such divisor exists, the number is considered prime and added to the resulting list.

**Preference-optimization baselines**

DPO

Certainly! Here's the one-liner using a lambda function and list comprehension to get a list of prime numbers from 1 to 200:

```
primes = [x for x in range(2, 201) if all(x % y != 0 for y in range(2, int(x**0.5) + 1))]
```

This code uses a list comprehension to filter out non-prime numbers by checking divisibility up to the square root of each number.

IPO

Certainly! Here's the one-liner using a lambda function and list comprehension to get a list of prime numbers from 1 to 200:

```
primes = [x for x in range(2, 201) if all(x % y != 0 for y in range(2, int(x**0.5)
+ 1))]
```

This code uses a list comprehension to filter out non-prime numbers by checking divisibility up to the square root of each number.

**CPO**

```
primes = [x for x in range(2, 201) if all(x % y != 0 for y in range(2, int(x**0.5)+1))]
```

This code uses a list comprehension to generate a list of prime numbers between 1 and 200. The all function is used to check if the number is divisible by any number between 2 and the square root of the number. If it's not divisible by any of these numbers, then it's a prime number.

**KTO**

Here's the one-liner Python code using a lambda function and list comprehension to get a list of prime numbers from 1 to 200:

```
primes = [x for x in range(2, 201) if all(x % y != 0 for y in range(2, int(x**0.5)
+ 1))]
```

This code uses a list comprehension to generate the list of primes and a lambda-like condition inside the all() function to check if a number is prime.

**SimPO**

```
primes = [x for x in range(2, 201) if all(x % d != 0 for d in range(2, int(x**0.5)+1))]
```

**Token-aware methods**

---

**TDPO**

Certainly! Here's the one-liner using a lambda function and list comprehension to get a list of prime numbers from 1 to 200:

```
primes = [x for x in range(2, 201) if all(lambda y: y > 1 and x % y != 0 for y in
range(2, int(x**0.5) + 1))]
```

However, note that the lambda function is not directly used in the list comprehension itself but rather within the all function to check for primality. A more conventional way to include the lambda would be:

```
is_prime = lambda x: all(x % y != 0 for y in range(2, int(x**0.5) + 1))
primes = [x for x in range(2, 201) if is_prime(x)]
```

This version is more readable and adheres to Python best practices.

**TIS-DPO**

Certainly! Here's the one-liner Python code using a lambda function and list comprehension to get a list of prime numbers from 1 to 200:

```
primes = [x for x in range(2, 201) if all(lambda y: y > 1 and x % y != 0 for y in
range(2, int(x**0.5) + 1))]
```

However, note that the lambda function is not directly used in the list comprehension condition as it's more conventional to use a regular function for clarity. A more readable version would be:

```
primes = [x for x in range(2, 201) if all(x % y != 0 for y in range(2, int(x**0.5)
+ 1))]
```

This list comprehension checks each number x from 2 to 200 to see if it is divisible by any number y in the range from 2 to the square root of x, ensuring that x is a prime number.

ConfPO

Certainly! Here's the one-liner using a lambda function and list comprehension to get a list of prime numbers from 1 to 200:

```
primes = [x for x in range(2, 201) if all(x % y != 0 for y in range(2, int(x**0.5)
+ 1))]
```

This code uses a list comprehension to filter out non-prime numbers by checking divisibility up to the square root of each number.

LES-DPO

Certainly! Here's a one-liner using a lambda function and list comprehension to get the list of prime numbers from 1 to 200:

```
primes = [n for n in range(2, 201) if all(lambda x: x > 1 or x == n for x in range(2,
int(n**0.5) + 1))]
```

Note: The lambda function in this context is not the most efficient way to check for primality, but it fits the requirement of using a lambda function in a list comprehension. A more common approach would be to use a regular function for checking primality.