

Critique DPO

Zihe Liu, Peter Hayes

March 15, 2026

Abstract

Direct Preference Optimization (DPO) enables fine-tuning language models from pairwise preferences without reinforcement learning. We propose Critique DPO (CDPO), which augments the DPO loss with an additional negative log-likelihood term on an external verifier’s chain-of-thought critique. We evaluate CDPO against DPO and SFT baselines on Qwen2.5-0.5B-Instruct using 120,000 math preference triples from the GenRM dataset with math problems from GSM8K. Our results are a negative finding: both DPO and CDPO suffer reward-hacking and performance collapse on GSM8K and MATH-500, with CDPO collapsing faster than vanilla DPO.

1 Introduction

As large language models (LLMs) become more capable, fine-tuning techniques to align with human intent are increasingly important. A key consideration for aligning these models is how to most effectively use human resources, or model resources in the case where LLMs themselves are used as oracles. Reinforcement learning from human or AI preferences (RLHF/RLAIF) is the most prominent example of such a technique but is complex and often unstable. Direct Preference Optimization (DPO) [Rafailov et al., 2024] has recently been proposed as a simpler and more stable alternative, bypassing the reward modelling and RL steps entirely. However, DPO reduces the oracle’s judgement to a single binary label — preferred or rejected — discarding any reasoning the oracle provides for its preference.

Asking these oracles to provide reasoning for their binary preferences is common practice. In particular, for LLM-as-a-judge style workflows, asking the model to provide chain-of-thought reasoning has been shown to improve the quality of the judgements themselves. This raises a natural question: can we leverage the oracle’s reasoning — not just its binary label — to improve training sample efficiency when fine-tuning the base model?

In this work, we propose Critique DPO (CDPO), which augments the DPO objective with a negative log-likelihood term on the verifier’s chain-of-thought critique, training the base policy to reproduce step-by-step verification reasoning alongside learning from pairwise preferences. Our experiments on Qwen2.5-0.5B-Instruct yield a negative finding: the critique term exacerbates rather than mitigates DPO’s reward-hacking failure mode. We analyse this collapse and its implications for critique-augmented training.

2 Background: DPO

For a given prompt x , two completions are sampled from the policy $(y_w, y_l) \sim \pi_\theta(y|x)$ and an oracle judges which it prefers. We denote y_w as the preferred completion and y_l as the other. This process is repeated over N prompts resulting in the pairwise preference dataset $\mathcal{D} = \{x^{(i)}, y_w^{(i)}, y_l^{(i)}\}_{i=1}^N$.

2.1 Bradley-Terry Preference Model

We can model the judge preferences using the Bradley-Terry (BT) model, which expresses the probability that completion y_1 is preferred over y_2 under a latent reward function r^* :

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

A key property is that BT depends only on the *difference* of rewards, not their absolute values.

2.2 Standard RLHF

The standard RLHF pipeline comprises 2 stages. First, we learn a reward model r_ϕ where we find the maximum likelihood estimate (MLE) of ϕ under our preference dataset $\mathcal{D}$ and BT model - termed reward modelling. Then, we use RL (PPO) to learn an optimal policy π_θ for this learned reward model.

2.2.1 Reward Modelling

Since r^* is unknown, we learn a parametric approximation r_ϕ . Under the BT model, the likelihood of the observed preference dataset is $\ell(\mathcal{D} \mid \phi) = \prod_{i=1}^N \sigma(r_\phi(x^{(i)}, y_w^{(i)}) - r_\phi(x^{(i)}, y_l^{(i)}))$. Taking the log and writing as an expectation, the MLE is obtained by minimising

$$\mathcal{L}_{\text{RM}}(\phi) = -\mathbb{E}_{\mathcal{D}} [\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))] \quad (1)$$

2.2.2 RL Fine-Tuning

The learned reward model is then used to fine-tune the policy. The aim is to maximise the KL-constrained objective

$$\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(y|x)} [r_\phi(x, y)] - \beta D_{\text{KL}}(\pi_\theta(y|x) \parallel \pi_{\text{ref}}(y|x)) \quad (2)$$

where π_{ref} is the reference policy (typically the SFT model), kept frozen. The KL term prevents the policy from deviating too far from the distribution on which the reward model is accurate, and $\beta > 0$ controls this trade-off. In practice this is optimised with PPO, requiring four models in memory: the policy, reference, reward model, and value function.

2.3 DPO

DPO [Rafailov et al., 2024] bypasses the reward modelling and RL steps entirely through a change of variables. The key insight is that there exists a bijection between reward functions and their corresponding optimal policies under the KL-constrained objective (2), and this bijection can be exploited to reparameterise the preference likelihood directly in terms of the policy. We follow the expositions by [Wolfe, 2025] and [Romero, 2024] in our presentation of the derivation below.

2.3.1 Optimal Policy

For *any* reward function r (not just the learned r_ϕ or the latent r^*), the KL-constrained objective (2) can be rewritten as a minimisation of a KL divergence between π and a target distribution defined by r , plus a constant. Specifically, letting $Z_r(x) = \sum_y \pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x, y)\right)$ be a partition function that depends on x and r but is constant with respect to π , the objective becomes

$$\min_{\pi} \mathbb{E}_x \left[D_{\text{KL}} \left(\pi(y|x) \parallel \frac{1}{Z_r(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x, y)\right) \right) - \log Z_r(x) \right]$$

By Gibbs' inequality ($D_{\text{KL}} \geq 0$, with equality iff the distributions are identical), the unique optimum is

$$\pi_r(y|x) = \frac{1}{Z_r(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x, y)\right) \quad (3)$$

This defines a bijection: every reward r maps to a unique optimal policy π_r , and conversely every policy π_r defines a unique reward¹ r for which it is optimal. Using the latent reward r^* gives the optimal policy $\pi^* \equiv \pi_{r^*}$ for our latent rewards.

¹We really mean unique equivalence class of rewards and its optimal policy; see Lemma 1 and Lemma 2 in [Rafailov et al., 2024].

2.3.2 Reparameterisation

Rearranging (3) for the reward expresses any reward purely in terms of its corresponding optimal policy and a prompt-dependent partition function:

$$r(x, y) = \beta \log \frac{\pi_r(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z_r(x)$$

Substituting the latent reward $r^*(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z_{r^*}(x)$ into the BT model, the $\beta \log Z_{r^*}(x)$ terms cancel because BT depends only on reward *differences* and $Z_{r^*}(x)$ is independent of y :

$$p^*(y_w \succ y_l | x) = \sigma \left(\beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right)$$

The preference model now has no dependence on a reward model — it is expressed entirely in terms of policies.

2.3.3 DPO Loss

We replace the fixed π^* with a learnable π_θ and maximise the likelihood of the observed preferences, conditioned on our policy parameters θ . Defining the *implicit reward*

$$\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} \quad (4)$$

the DPO loss is

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{\mathcal{D}} [\log \sigma(\hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l))] \quad (5)$$

This is identical in form to the reward model loss (1), but conditioned on the policy parameters θ instead of the reward model parameters ϕ . Traditionally, RLHF performs MLE on $\mathcal{D}$ conditioned on ϕ to learn a reward model, then uses RL to find the optimal policy for that reward. DPO instead performs MLE on $\mathcal{D}$ conditioned directly on θ . Because of the reward–policy bijection, improving π_θ to better fit the preference data simultaneously yields an implicit reward $\hat{r}_\theta$ for which π_θ is guaranteed to be optimal. This eliminates both the reward modelling step and the RL step, requiring only two models in memory (the policy π_θ and the frozen reference π_{ref}).

3 Literature Review

We review research incorporating reasoning and critique data into fine-tuning, building toward the critique-augmented preference optimisation approach of this work. We adopt a unified notation: π_θ denotes the learnable policy, π_{ref} the frozen reference (usually the base SFT model), x the prompt, y_w and y_l the preferred and rejected responses, $r_\phi(x, y)$ a learned reward (where ϕ parameterises any judge, verifier, or reward model), and $\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ the implicit reward. For chain-of-thought traces, c_w, c_l denote the model’s own reasoning, while a v subscript $c_{v,w}, c_{v,l}$ denotes a verifier’s or external oracle’s critique for the winning or losing response accordingly.

The review is organised around a central question: *what kind of signal is being used, and for which model?* The narrative moves from binary AI feedback (Section 3.1), through enriching DPO with supervised reasoning targets (Section 3.2), to methods that improve verifiers by having them reason (Section 3.3), and finally to the closest work — methods that bring verifier critique directly into the base model’s training objective (Section 3.4).

3.1 Beyond Binary Feedback

3.1.1 RLAI

RL from AI Feedback (RLAI) replaces human annotators with LLM oracles. [Bai et al., 2022] first use LLM-generated labels for harmlessness finetuning, doing RL with a preference model trained with

LLM preferences. [Lee et al., 2024] use LLM to directly give rewards, showing that RLAIIF matches or exceeds RLHF on summarisation and dialogue, even when the labeller is the same model being aligned. [Li et al., 2024] instruct the labeller to reason explicitly and report modest improvements. A shared limitation is that the oracle’s reasoning is discarded after labelling: the sole training signal is a binary or scalar preference label, creating an information bottleneck that the remainder of this review seeks to address.

3.1.2 Impact of Chain-of-Thought Data for DPO

[H. Liu et al., 2025] provide a systematic empirical study of CoT’s impact on DPO. In text-to-SQL — where preference datasets traditionally contain only final SQL queries *without* reasoning steps — DPO degrades performance across the majority of ten tested LLMs. Augmenting the same datasets with synthetically generated chain-of-thought solutions achieves consistent DPO improvements for the first time. The analysis identifies three mechanisms. First, *reward hacking mitigation*: without CoT, the implicit reward (4) can be inflated by superficial correlates (longer SQL, non-executable patterns) that do not reflect reasoning quality. Model performance initially peaks then decreases, while the implicit reward continues to rise. With CoT, the implicit reward tracks actual performance. Second, *discriminative calibration*: CoT maintains stable accuracy in distinguishing correct from incorrect solutions, whereas vanilla DPO becomes unstable. Third, *sample efficiency*: CoT-augmented DPO continues to improve with additional data, while vanilla DPO saturates.

3.1.3 Self-Taught Reasoning: STaR and V-STaR

STaR. [Zelikman et al., 2022] establish that CoT data, even when self-generated, is a richer training signal than outcome labels alone. Given a pretrained model M and a dataset $\mathcal{D} = \{(x_i, y_i)\}$ of problems with ground-truth answers, STaR begins with a small few-shot prompt set $\mathcal{P}$ containing rationale examples. At each iteration, the model generates a reasoning trace $\hat{c}_i$ and answer $\hat{y}_i$ for every problem, using $\mathcal{P}$ in the context. Only correct generations where $\hat{y}_i = y_i$ are retained; the model is then fine-tuned on this filtered set with the standard NLL objective on rationale-answer pairs:

$$\mathcal{L}_{\text{STaR}}(M) = -\mathbb{E}_{(x,y) \sim \mathcal{D}} \left[\mathbf{1}[\hat{y} = y] \cdot \left(\sum_{t=1}^{|\hat{c}|} \log M(\hat{c}_t \mid x, \hat{c}_{<t}) + \log M(\hat{y} \mid x, \hat{c}) \right) \right] \quad (6)$$

V-STaR. [Hosseini et al., 2024] extend STaR by repurposing incorrect solutions (discarded by STaR) to co-train a verifier V alongside the generator G . At each iteration, the generator is fine-tuned via SFT on correct solutions using Eq. (6). The verifier is trained with DPO on preference pairs (x, y_w, y_l) , formed by pairing correct and incorrect solutions for each problem, using G_{SFT} (the base model fine-tuned once) as the fixed reference policy:

$$\mathcal{L}_{\text{DPO}}(V^t; G_{\text{SFT}}) = -\mathbb{E}_{(x,y_w,y_l) \sim \mathcal{D}_{\text{VER}}} [\log \sigma(\hat{r}(x, y_w) - \hat{r}(x, y_l))], \quad \hat{r}(x, y) = \beta \log \frac{V^t(y \mid x)}{G_{\text{SFT}}(y \mid x)} \quad (7)$$

Crucially, the verifier provides no training-time gradient to the generator — it is used only for Best-of- k reranking at inference. This distinguishes V-STaR from later methods that feed verifier reasoning directly into the generator’s training.

3.2 DPO with NLL

A recurring structural pattern in the literature combines the DPO preference loss with an auxiliary negative log-likelihood (NLL) term on a supervised target. We present the generic template first, then show how each paper instantiates it with a **different NLL target t** :

$$\mathcal{L}_{\text{DPO+NLL}} = \underbrace{-\log \sigma(\hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l))}_{\text{preference term}} + \alpha \underbrace{\mathcal{L}_{\text{NLL}}(t \mid x)}_{\text{supervised anchor}} \quad (8)$$

The DPO term provides a contrastive signal (push apart y_w and y_l), while the NLL term anchors the policy to a specific supervised target. As we show below, the choice of t is what distinguishes these methods from one another.

3.2.1 Iterative Reasoning Preference Optimization (IRPO)

[Pang et al., 2024] set the NLL target to the **model’s own winning chain-of-thought**: $t = (c_w, y_w)$. At each iteration, the model samples N reasoning traces per prompt, partitions them into correct (c_w, y_w) and incorrect (c_l, y_l) by checking the final answer against ground truth, and optimises:

$$\begin{aligned}\mathcal{L}_{\text{IRPO}} &= \mathcal{L}_{\text{DPO}}(c_w, y_w, c_l, y_l \mid x) + \alpha \mathcal{L}_{\text{NLL}}(c_w, y_w \mid x) \\ &= -\log \sigma \left(\beta \log \frac{\pi_\theta(c_w, y_w \mid x)}{\pi_{\text{ref}}(c_w, y_w \mid x)} - \beta \log \frac{\pi_\theta(c_l, y_l \mid x)}{\pi_{\text{ref}}(c_l, y_l \mid x)} \right) - \alpha \frac{\log \pi_\theta(c_w, y_w \mid x)}{|c_w| + |y_w|}\end{aligned}\quad (9)$$

The NLL term is length-normalised by $|c_w| + |y_w|$ to avoid a bias toward shorter sequences. Crucially, the reference model π_{ref} is updated to π_θ at each iteration before re-sampling. Under standard DPO (i.e. $\alpha = 0$), the log-probabilities of *chosen* sequences may actually decrease during training — the model achieves the required preference margin by pushing rejected probabilities down faster than chosen ones fall. The NLL term prevents this collapse: with $\alpha > 0$, chosen log-probabilities increase while rejected ones decrease.

3.2.2 NLL on Chosen Responses: AIPO and RPO

Both AIPO [Shen et al., 2024] and RPO [Z. Liu et al., 2024] set the NLL target to the **winning response**: $t = y_w$. AIPO adds a length-normalised NLL anchor alongside α -DPO, a DPO variant that scales the reference log-ratio by $(1 + \alpha)$ to introduce a dynamic preference margin weighted by the reference model’s agreement with the ranking:

$$\mathcal{L}_{\text{AIPO}} = \mathcal{L}_{\alpha\text{-DPO}} + \lambda \cdot \mathcal{L}_{\text{NLL}}, \quad \mathcal{L}_{\text{NLL}} = -\frac{1}{|y_w|} \log \pi_\theta(y_w \mid x) \quad (10)$$

RPO derives the same NLL term from first principles: framing DPO overoptimisation as a minimax problem over adversarially chosen reward models, the objective collapses exactly to DPO plus NLL on chosen responses:

$$\mathcal{L}_{\text{RPO}}(\theta) = \mathcal{L}_{\text{DPO}}(\theta) - \eta \beta \mathbb{E}_{(x, y_w) \sim \mathcal{D}} [\log \pi_\theta(y_w \mid x)] \quad (11)$$

Both methods improve over vanilla DPO, confirming that the NLL anchor mitigates reward hacking regardless of whether the motivation is empirical or theoretical.

3.2.3 Direct Judgement Preference Optimization (DJPO)

[P. Wang et al., 2024] set the NLL target to the **verifier’s CoT critique and judgment**: $t = (c_v, j)$, where c_v is the verifier’s chain-of-thought critique and j is the final judgment. Crucially, the model being trained is a judge, not the base policy. They construct DPO preference pairs over judge evaluations — pairing correct and incorrect judgements (with and without CoT reasoning) — and train with:

$$\mathcal{L}_{\text{DJPO}} = -\frac{\log \pi_\phi(c_{v,w} \mid x)}{|c_{v,w}| + |x|} - \log \sigma \left(\beta \log \frac{\pi_\phi(c_{v,w} \mid x)}{\pi_{\text{ref}}(c_{v,w} \mid x)} - \beta \log \frac{\pi_\phi(c_{v,l} \mid x)}{\pi_{\text{ref}}(c_{v,l} \mid x)} \right) \quad (12)$$

where $c_{v,w}$ and $c_{v,l}$ are correct and incorrect judge evaluations (critique plus judgment) respectively. Using 680K preference pairs, they train SFR-Judge models at 8B, 12B, and 70B scales. SFR-Judge-8B outperforms GPT-4o on aggregate pairwise evaluation across 7 benchmarks; SFR-Judge-70B achieves the best overall score of 84.25%.

3.3 Generative Verifiers and Critique-Augmented Reward Models

The methods above train the policy model. A parallel line of work improves the verifiers and reward models that judge responses, by having them produce reasoning before scoring. These methods establish that critique data is a powerful training signal for judging. Generative verifiers aim to provide a scalar verifier score that is used during inference for Best-of-N sampling, whereas critique-augmented reward models produce better reward models for downstream preference training.

3.3.1 Generative Verifiers (GenRM)

[Zhang et al., 2024] recast verification as next-token prediction rather than scalar classification. Instead of a separate reward head, the verifier is an LLM fine-tuned via SFT to answer “Is the answer correct?” given a problem x and candidate solution y .

In **GenRM-Direct**, the verifier is trained on pairs of $(x, y_w, I) \rightarrow \text{'Yes'}$ and $(x, y_l, I) \rightarrow \text{'No'}$, where $I = \text{"Is the answer correct (Yes/No)?"}$. At inference, the verifier score is the next-token probability of ‘Yes’:

$$r_{\text{Direct}}(x, y) = \pi_{\phi}(\text{'Yes'} \mid x, y, I) \quad (13)$$

GenRM-CoT inserts a chain-of-thought verification rationale c_v before the final Yes/No token. The rationales are generated offline by a capable LLM and filtered by majority agreement. At inference, the verifier first samples its own rationale, then scores:

$$r_{\text{CoT}}(x, y) = \pi_{\phi}(\text{'Yes'} \mid x, y, I_v, c_v, I), \quad c_v \sim \pi_{\phi}(\cdot \mid x, y, I_v) \quad (14)$$

where $I_v = \text{"Let's verify step by step."}$. GenRM-CoT achieves 94.2% Best-of-32 on GSM8K (vs. $\sim 90\%$ for a discriminative RM) with $6.4\times$ better sample efficiency. This dataset — including the per-solution verification rationales — is the data source for our experiments.

3.3.2 GenRM-STaR-DPO: Training Judges with DPO on Rationales

[Mahan et al., 2024] (Mahan et al., 2024) propose a unified RLHF–RLAIF framework for training a generative judge as a reward model. They create 3 training variants. Given a dataset $\mathcal{D} = \{(x, y_1, y_2, I)\}$ of unranked response pairs where I is the indicator encoding which response won:

GenRM (no CoT) — standard SFT on the indicator token:

$$\mathcal{L}_{\text{GenRM}}(\pi_{\phi}) = \mathbb{E}_{(x, y_1, y_2, I) \sim \mathcal{D}} [-\log \pi_{\phi}(I \mid x, y_1, y_2)] \quad (15)$$

CoT-GenRM with Rationalization — a post-rationalization model q (e.g. GPT-4 or a stronger LLM) generates a critique rationale c_v conditioned on the correct answer, and the model is trained to both generate the rationale and predict the indicator:

$$\mathcal{L}_{\text{Rationalizer}}(\pi_{\phi}) = \mathbb{E}_{(x, y_1, y_2, c_v, I) \sim \mathcal{D}} [-\log \pi_{\phi}(I \mid x, y_1, y_2, c_v) - \log \pi_{\phi}(c_v \mid x, y_1, y_2)] \quad (16)$$

CoT-GenRM-STaR-DPO — instead of discarding incorrect self-generated critique rationales (as STaR-SFT does), STaR-DPO constructs preference pairs $(c_{v,w}, I_w)$ vs $(c_{v,l}, I_l)$ from correct and incorrect reasoning traces and applies DPO over the full rationale-plus-judgment sequence:

$$\mathcal{L}_{\text{STaR-DPO}}(\pi_{\phi}) = -\mathbb{E}_{\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_{\phi}(I_w, c_{v,w} \mid x, y_1, y_2)}{\pi_{\text{ref}}(I_w, c_{v,w} \mid x, y_1, y_2)} - \beta \log \frac{\pi_{\phi}(I_l, c_{v,l} \mid x, y_1, y_2)}{\pi_{\text{ref}}(I_l, c_{v,l} \mid x, y_1, y_2)} \right) \right] \quad (17)$$

On-policy rationales are essential: using off-policy rationales from a stronger model (GPT-4 or Llama-3.1-70B) matches STaR-DPO in-distribution but collapses OOD (RewardBench drops to 63–70%). The authors attribute this to distributional mismatch: the model imitates the rationaliser’s style during training but generates different reasoning at test time.

3.3.3 Critique-out-Loud Reward Models (CLOUD)

[Ankner et al., 2024] train reward models with a shared base θ_B , an LM head θ_{LM} that generates critiques, and a scalar reward head θ_R . Training proceeds in three stages: (1) SFT θ_{LM} on oracle critiques; (2) self-generate critiques to form $\mathcal{D}_{\text{self}}$; (3) jointly optimise all parameters on $\mathcal{D}_{\text{self}}$:

$$\mathcal{L}_{\text{CLOUD}} = \underbrace{-\mathbb{E}[\log \sigma(r_{\theta_R}(x, y_w, \hat{c}_w) - r_{\theta_R}(x, y_l, \hat{c}_l))]}_{\mathcal{L}_{\text{RM}}} + \lambda \underbrace{\mathcal{L}_{\text{SFT}}(\theta_B, \theta_{LM}, \mathcal{D}_{\text{self}})}_{\text{critique quality}} \quad (18)$$

The self-generation step is critical: training the reward head on oracle (off-policy) critiques causes a 5.6% drop in RewardBench accuracy, because oracle critiques are systematically different from self-generated ones, creating a distributional shift at inference. [Yu et al., 2025] extend this with a variational formulation (Critic-RM) that treats self-generated critiques as latent variables.

3.4 Training the Base Model with Critique

The methods in Section 3.3 improve reward models and judges by having them generate critique. A closely related cluster instead trains the base *policy* model directly using critique signals.

3.4.1 Critique Fine-Tuning

[Y. Wang et al., 2025] ask: what happens if we use *only* the NLL term on critique data, without any preference optimisation? In Critique Fine-Tuning (CFT), the base model is trained on the teacher’s critique c_v of a given query-response pair by maximising the log-likelihood:

$$\mathcal{L}_{\text{CFT}}(\theta) = - \sum_{t=1}^{|c_v|} \log \pi_{\theta}(c_{v,t} \mid c_{v,<t}, [x; y]) \quad (19)$$

where $[x; y]$ concatenates the query and a (potentially flawed) response, and c_v is a critique generated by a teacher model (GPT-4o). At inference, the CFT-trained model generates direct answers without producing critiques.

3.4.2 SuperCorrect: Cross-Model Collaborative DPO

[Yang et al., 2024] introduce a two-stage framework. Stage 1 distils hierarchical thought templates from a large teacher model (o1-preview) via SFT. Stage 2 applies **cross-model collaborative DPO** on teacher-guided error-correction traces. Given an erroneous student solution with the first error at step k , the teacher generates a correction trace (a_k^+, c_k^+) (error analysis and correction guidance) while the student independently generates (a_k^-, c_k^-) . The DPO loss is:

$$\mathcal{L}_{\text{Cross-DPO}} = -\mathbb{E} \left[\log \sigma \left(\beta \log \frac{\pi_{\theta}(a_k^+, c_k^+ \mid x, s_{<k})}{\pi_{\text{ref}}(a_k^+, c_k^+ \mid x, s_{<k})} - \beta \log \frac{\pi_{\theta}(a_k^-, c_k^- \mid x, s_{<k})}{\pi_{\text{ref}}(a_k^-, c_k^- \mid x, s_{<k})} \right) \right] \quad (20)$$

where $s_{<k}$ are the correct preceding steps. The teacher’s correction trace plays the role of c_v in CDPO. SuperCorrect-7B achieves 70.2% on MATH (surpassing DeepSeekMath-7B by 7.8%) and demonstrates that cross-model DPO at the step level substantially improves self-correction capabilities. CriticGPT [McAleese et al., 2024] and CTRL [Xie et al., 2025] provide complementary evidence that LLMs can be explicitly taught to generate high-quality critiques.

3.5 Positioning of CDPO

The three defining features of CDPO are: **(A)** a DPO loss on answer preferences (y_w, y_l) ; **(B)** an NLL term on the verifier’s CoT critique c_v ; and **(C)** applied to the base policy model.

Paper	DPO on answers	NLL on verifier CoT	Base model
CDPO (this work)	✓	✓	✓
IRPO [Pang et al., 2024]	✓	NLL on own CoT	✓
AIPO [Shen et al., 2024] / RPO [Z. Liu et al., 2024]	✓	NLL on y_w	✓
DJPO [P. Wang et al., 2024]	✓	✓	× (judge)
GenRM-STaR-DPO [Mahan et al., 2024]	✓	✓	× (judge)
CFT [Y. Wang et al., 2025]	×	✓	✓
SuperCorrect [Yang et al., 2024]	On traces	Teacher correction	✓

CDPO sits at the intersection of three lines of work (1) IRPO establishes (DPO + NLL) but anchors to the model’s own CoT rather than an external verifier’s critique; (2) DJPO and GenRM-STaR-DPO use verifier reasoning as the NLL target but apply it to train a standalone judge rather than the answer-generating base policy; and (3) CFT applies verifier critique as a direct NLL target for the base policy but omits the DPO preference term entirely.

4 Methodology

We now describe the CDPO objective, dataset construction, and baseline methods.

4.1 Data Construction

We use the GenRM dataset [Zhang et al., 2024], which provides math problems from the GSM8K [Cobbe et al., 2021] training set. For each problem x , multiple candidate solutions are generated by Gemini 1.0 Pro. Each candidate solution is labelled correct or incorrect by checking the extracted final answer against the ground-truth answer. An LLM oracle generates a step-by-step verification rationale *for each individual solution*, verifying each reasoning step against the expected answer and concluding with a “Yes” (correct) or “No” (incorrect) judgement. We denote the verification rationale for a correct solution as $c_{v,w}$ and for an incorrect solution as $c_{v,l}$.

From this raw data, we construct preference triples (x, y_w, y_l, c_v) by pairing, for each problem, a correct solution y_w with an incorrect solution y_l . The critique sequence c_v is then assembled by concatenating *both* solutions together with their respective verification rationales into a single contrastive narrative:

$c_v = \text{“This is a correct solution and preferred:”} + y_w + \text{“Here’s why this solution is correct and preferred:”} + c_{v,w} + \text{“This is an incorrect solution and not preferred:”} + y_l + \text{“Here’s why this solution is incorrect and not preferred:”} + c_{v,l}$

From this data we derive four training methods, described below.

4.2 CDPO: Critique Direct Preference Optimization

Each CDPO sample is (x, y_w, y_l, c_v) . CDPO augments the standard DPO loss with an auxiliary negative log-likelihood (NLL) term that trains the base policy to reproduce the verifier’s chain-of-thought critique:

$$\mathcal{L}_{\text{CDPO}}(\theta) = \mathbb{E}_{\mathcal{D}}[(1 - \alpha) \mathcal{L}_{\text{DPO}}(\theta) + \alpha \mathcal{L}_{\text{critique}}(\theta)] \quad (21)$$

where $\alpha \in [0, 1]$ controls the trade-off between the preference signal and the critique signal. The DPO term $\mathcal{L}_{\text{DPO}}$ is the standard loss from Eq. (5). The critique term is the length-normalised NLL over the composite critique c_v :

$$\mathcal{L}_{\text{critique}}(\theta) = -\frac{1}{|c_v|} \sum_{t=1}^{|c_v|} \log \pi_{\theta}(c_{v,t} \mid x, c_{v,<t}) \quad (22)$$

Concretely, the model receives the input sequence $[x; c_v]$ — the problem followed by the critique — and the NLL loss is computed only over the c_v tokens. Because c_v embeds both solutions y_w

and y_l within its narrative (as shown above), it is self-contained: the model learns to generate the full contrastive analysis given only the problem statement. We hypothesise that reproducing this analysis teaches the model to distinguish correct from incorrect reasoning steps, thereby improving its own solution quality.

CDPO follows the DPO+NLL pattern of Eq. (8) with the NLL target set to the external verifier’s critique c_v , but uses a convex combination rather than an additive weight so that α directly controls the relative contribution of each term (see Section 3.5 for positioning against related methods). Note that at inference time, the CDPO-trained model generates solutions directly — it does not produce critiques.

4.3 Baseline Methods

We compare CDPO against three baselines, all trained on the same underlying data:

DPO. Each sample is (x, y_w, y_l) . The standard DPO loss Eq. (5), equivalent to setting $\alpha = 0$ in Eq. (21). This isolates the contribution of the critique term.

SFT (no verification). Each sample is (x, y_w) . Standard supervised fine-tuning on correct solutions only:

$$\mathcal{L}_{\text{SFT}}(\theta) = - \sum_{t=1}^{|y_w|} \log \pi_{\theta}(y_{w,t} \mid y_{w,<t}, x) \quad (23)$$

SFT (with verification). Each sample is $(x, y_w, c_{v,w})$. The same SFT objective, but the training target is $[y_w; c_{v,w}]$ — the correct solution followed by its individual verification rationale. This tests whether the verification signal is useful as a pure SFT target, without the DPO preference term.

5 Experimental Setup

5.1 Training Configuration

We use Qwen2.5-0.5B-Instruct [Qwen-Team, 2024] as the base model. All four methods share the same base configuration: LoRA with rank 8 applied to all linear modules, learning rate 1×10^{-6} with cosine decay and 10% warmup, 2 training epochs, bf16 mixed precision, FlashAttention-2, and DeepSpeed ZeRO Stage 3 across 8 NVIDIA H100 80GB GPUs. For DPO and CDPO we use $\beta = 0.1$; CDPO additionally uses $\alpha = 0.5$ with length-normalised critique. Batch sizes differ across methods because CDPO processes three sequences per example (chosen, rejected, critique), requiring significantly more memory. Full hyperparameters are listed in Table 2 (Appendix B).

5.2 Evaluation

We evaluate on two mathematical reasoning benchmarks: GSM8K [Cobbe et al., 2021] (500 test samples) and MATH-500 [Hendrycks et al., 2021]. All evaluations use greedy decoding (temperature 0, top- $p = 1$) via vLLM for efficient inference. LoRA adapters are merged with the base model before evaluation. We evaluate checkpoints saved at every 10% of training. Note that our training data is drawn from GSM8K training problems, so GSM8K evaluation is in-distribution while MATH-500 is out-of-distribution; this partly explains the more stable GSM8K results for SFT methods.

6 Results

6.1 Main Results

Table 1 summarises the best and final checkpoint accuracy for each method. The baseline row shows the pre-training accuracy of the unmodified Qwen2.5-0.5B-Instruct model (checkpoint 0, shared across all methods).

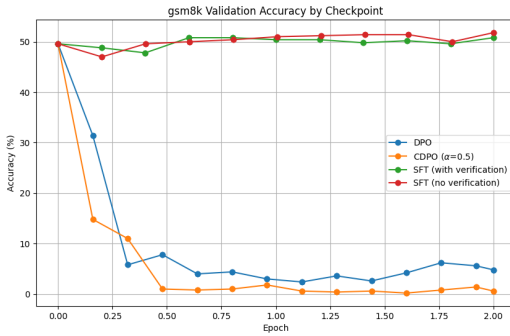
Table 1: Test accuracy (%) on GSM8K and MATH-500. *Best* reports the highest accuracy across all checkpoints; *Final* reports the last checkpoint. The baseline is the unmodified Qwen2.5-0.5B-Instruct model.

Method	GSM8K		MATH-500	
	Best	Final	Best	Final
Baseline (ckpt-0)	49.6	—	34.4	—
SFT (no verification)	51.8	51.4	35.4	32.6
SFT (with verification)	50.8	50.8	35.2	33.0
DPO	49.6	4.8	34.4	3.2
CDPO ($\alpha=0.5$)	49.6	0.6	34.4	0.8

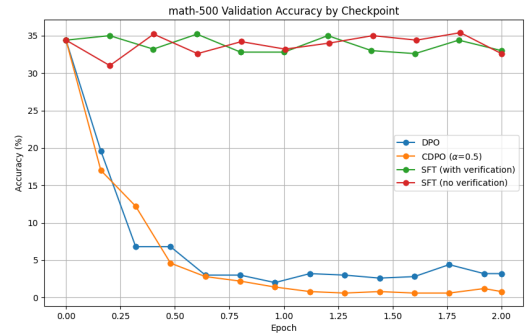
SFT methods provide modest, stable improvements over the baseline, while both preference-based methods (DPO and CDPO) suffer catastrophic performance collapse. Critically, CDPO — the method designed to leverage richer feedback — performs *worse* than vanilla DPO.

6.2 Training Dynamics

Figure 1 shows evaluation accuracy across training checkpoints. The SFT variants maintain stable accuracy throughout training, with SFT (no verification) reaching 51.8% on GSM8K — a modest 2.2 percentage point improvement over the 49.6% baseline. Both DPO and CDPO begin at the baseline accuracy (checkpoint 0 is the unmodified model) and collapse rapidly: DPO drops to $\sim 6\%$ within the first 0.3 epochs, while CDPO collapses even more to $\sim 1\%$ by 0.5 epochs.



(a) GSM8K



(b) MATH-500

Figure 1: Evaluation accuracy over training checkpoints for all four methods. DPO and CDPO collapse rapidly from the baseline, while both SFT variants remain stable. CDPO collapses faster and to a lower floor than DPO.

Figure 2 reveals the mechanism behind this collapse. Despite the catastrophic drop in downstream accuracy, both DPO and CDPO show smoothly decreasing training loss and reward accuracy approaching 1.0 throughout training. This suggests reward hacking: the model learns to separate chosen from rejected examples in the implicit reward space (by pushing log-probability ratios apart), but this optimisation does not correspond to improved generation quality — it instead degrades the model’s ability to produce coherent solutions.

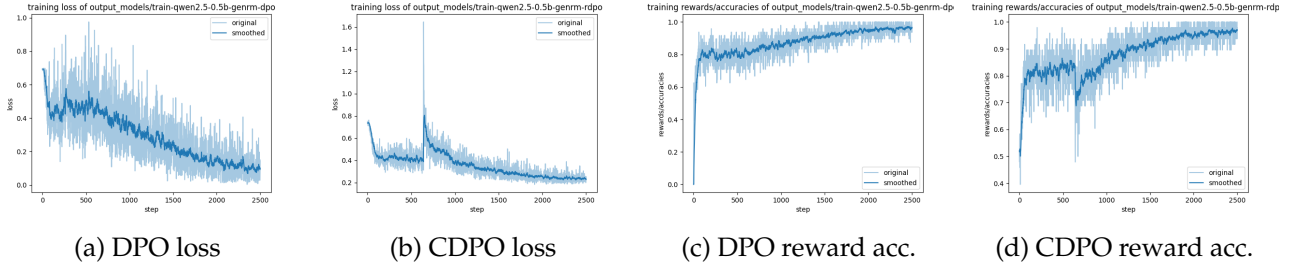


Figure 2: Training metrics for DPO and CDPO. Both methods show smoothly decreasing loss (a, b) and reward accuracy approaching 1.0 (c, d), despite the catastrophic collapse in downstream evaluation accuracy shown in Figure 1.

7 Discussion

The dramatic collapse of both DPO and CDPO, combined with smoothly improving training metrics, is a well-known failure mode of DPO. [H. Liu et al., 2025] identify exactly this pattern in text-to-SQL tasks: the implicit reward $\hat{r}_\theta$ can be inflated by superficial correlates that do not reflect reasoning quality — model performance initially peaks then decreases, while the implicit reward continues to rise. At the 0.5B parameter scale, the model appears to lack sufficient capacity to simultaneously maintain generation quality and satisfy the DPO preference margin. We identify two potential mechanisms behind this collapse:

Distributional mismatch. The verification rationale c_v is generated by an external oracle whose writing style, reasoning depth, and vocabulary differ substantially from the base model’s natural distribution. This echoes findings from GenRM-STaR-DPO [Mahan et al., 2024], where off-policy rationales from a stronger model matched STaR-DPO in-distribution but collapsed out-of-distribution. CCloud [Ankner et al., 2024] similarly reports a 5.6% drop when training on oracle critiques versus self-generated ones, attributing it to distributional shift between training and inference.

Task conflict. The verification rationale is fundamentally different from the solution-generation task the model performs at inference. Verification text contains meta-analytical language (“Step 1: check...”, “Expected Answer: Correct”, “Verification: Is the answer correct? Yes/No”) that is absent from problem-solving. Training the model to produce both types of text may create a multi-task conflict that degrades performance on the primary task.

8 Conclusion

We proposed Critique DPO (CDPO), which extends the DPO objective with an auxiliary NLL term on a verifier’s chain-of-thought critique. We evaluated CDPO against DPO and SFT baselines on Qwen2.5-0.5B-Instruct using 120,000 math preference triples from the GenRM dataset, with evaluation on GSM8K and MATH-500. Our results are a negative finding: both DPO and CDPO suffer catastrophic model collapse at this scale, with CDPO collapsing faster and more severely than vanilla DPO. The critique NLL term, intended to transfer the verifier’s analytical capabilities to the base model, instead exacerbates the well-documented DPO reward-hacking failure mode. Meanwhile, simple SFT on correct solutions provides modest but stable improvements, and adding verification rationales to SFT yields no additional benefit. Future work should explore: (1) on-policy critique generation (STaR-style self-generated rationales instead of off-policy oracle critiques), (2) larger model scales where the multi-task learning implicit in CDPO may become tractable.

A CDPO Training Algorithm

Algorithm 1 CDPO Training (per batch)

Require: Policy π_θ , reference π_{ref} , batch $\{(x_i, y_{w,i}, y_{l,i}, c_{v,i})\}_{i=1}^B$, weights α, β

- 1: **Concatenate** chosen, rejected, and critique into single batch of size $3B$
- 2: $\text{logits} \leftarrow \pi_\theta(\text{batch})$
- 3: $\text{all_logps}, \text{lengths} \leftarrow \text{GETBATCHLOGPS}(\text{logits}, \text{labels})$ ▷ Shape $3B$
- 4: **Delete** logits; free GPU memory
- 5: $\log \pi_\theta(y_w), \log \pi_\theta(y_l), \log \pi_\theta(c_v) \leftarrow \text{SPLIT}(\text{all_logps}, B)$
- 6: $L_w, L_l, L_c \leftarrow \text{SPLIT}(\text{lengths}, B)$ ▷ Valid token counts
- 7: **// Reference model log-probs (LoRA: disable adapter)**
- 8: $\log \pi_{\text{ref}}(y_w), \log \pi_{\text{ref}}(y_l) \leftarrow$ same procedure with π_{ref}
- 9: **// Implicit rewards**
- 10: $\hat{r}_w \leftarrow \beta (\log \pi_\theta(y_w) - \log \pi_{\text{ref}}(y_w))$
- 11: $\hat{r}_l \leftarrow \beta (\log \pi_\theta(y_l) - \log \pi_{\text{ref}}(y_l))$
- 12: $\mathcal{L}_{\text{DPO}} \leftarrow -\log \sigma(\hat{r}_w - \hat{r}_l)$
- 13: $\mathcal{L}_{\text{critique}} \leftarrow -\log \pi_\theta(c_v) / L_c$ ▷ Length-normalised NLL
- 14: $\mathcal{L}_{\text{CDPO}} \leftarrow \text{mean}[(1 - \alpha) \mathcal{L}_{\text{DPO}} + \alpha \mathcal{L}_{\text{critique}}]$
- 15: **return** $\mathcal{L}_{\text{CDPO}}$

B Hyperparameter Details

Table 2: Complete training hyperparameters for all methods.

Parameter	CDPO	DPO	SFT-veri	SFT-no.veri
Base model	Qwen2.5-0.5B-Instruct			
Fine-tuning method	LoRA (rank 8, all linear modules)			
Learning rate	1×10^{-6}			
LR scheduler	Cosine with 10% warmup			
Training epochs	2			
Precision	bf16			
Attention	FlashAttention-2			
Parallelism	DeepSpeed ZeRO Stage 3, 8× H100			
Batch size / GPU	6	16	32	48
Cutoff length (tokens)	1664	640	1024	768
β (DPO temperature)	0.1	0.1	—	—
α (critique weight)	0.5	—	—	—
Normalise critique	Yes	—	—	—
Preference loss	sigmoid	sigmoid	—	—
Gradient checkpointing	Yes	No	Yes	Yes
Dataset size	120K	120K	120K	120K

References

- Rafailov, Rafael et al. (2024). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv: 2305.18290 [cs.LG]. URL: <https://arxiv.org/abs/2305.18290>.
- Wolfe, Cameron R. (2025). *Direct Preference Optimization (DPO)*. Blog post. URL: <https://cameronrwolfe.substack.com/p/direct-preference-optimization>.
- Romero, Tyler (2024). *Direct Preference Optimization Explained In-depth*. Blog post. URL: <https://www.tylerromero.com/posts/2024-04-dpo/>.
- Bai, Yuntao et al. (2022). *Constitutional AI: Harmlessness from AI Feedback*. arXiv: 2212.08073 [cs.CL]. URL: <https://arxiv.org/abs/2212.08073>.
- Lee, Harrison et al. (2024). *RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback*. arXiv: 2309.00267 [cs.CL]. URL: <https://arxiv.org/abs/2309.00267>.
- Li, Ang et al. (2024). *HRLAIF: Improvements in Helpfulness and Harmlessness in Open-domain Reinforcement Learning From AI Feedback*. arXiv: 2403.08309 [cs.LG]. URL: <https://arxiv.org/abs/2403.08309>.
- Liu, Hanbing et al. (2025). *Uncovering the Impact of Chain-of-Thought Reasoning for Direct Preference Optimization*. arXiv: 2502.11656 [cs.CL]. URL: <https://arxiv.org/abs/2502.11656>.
- Zelikman, Eric et al. (2022). *STaR: Bootstrapping Reasoning With Reasoning*. arXiv: 2203.14465 [cs.LG]. URL: <https://arxiv.org/abs/2203.14465>.
- Hosseini, Arian et al. (2024). *V-STaR: Training Verifiers for Self-Taught Reasoners*. arXiv: 2402.06457 [cs.LG]. URL: <https://arxiv.org/abs/2402.06457>.
- Pang, Richard Yuanzhe et al. (2024). *Iterative Reasoning Preference Optimization*. arXiv: 2404.19733 [cs.CL]. URL: <https://arxiv.org/abs/2404.19733>.
- Shen, Yaojie et al. (2024). *AIPO: Improving Training Objective for Iterative Preference Optimization*. arXiv: 2409.08845 [cs.CL]. URL: <https://arxiv.org/abs/2409.08845>.
- Liu, Zhihan et al. (2024). *Provably Mitigating Overoptimization in RLHF: Your SFT Loss is Implicitly an Adversarial Regularizer*. arXiv: 2405.16436 [cs.LG]. URL: <https://arxiv.org/abs/2405.16436>.
- Wang, Peifeng et al. (2024). *Direct Judgement Preference Optimization*. arXiv: 2409.14664 [cs.CL]. URL: <https://arxiv.org/abs/2409.14664>.
- Zhang, Lunjun et al. (2024). *Generative Verifiers: Reward Modeling as Next-Token Prediction*. arXiv: 2408.15240 [cs.LG]. URL: <https://arxiv.org/abs/2408.15240>.
- Mahan, Dakota et al. (2024). *Generative Reward Models*. arXiv: 2410.12832 [cs.LG]. URL: <https://arxiv.org/abs/2410.12832>.
- Ankner, Zachary et al. (2024). *Critique-out-Loud Reward Models*. arXiv: 2408.11791 [cs.LG]. URL: <https://arxiv.org/abs/2408.11791>.
- Yu, Yue et al. (2025). *Self-Generated Critiques Boost Reward Modeling for Language Models*. arXiv: 2411.16646 [cs.CL]. URL: <https://arxiv.org/abs/2411.16646>.
- Wang, Yubo, Xiang Yue, and Wenhui Chen (2025). *Critique Fine-Tuning: Learning to Critique is More Effective than Learning to Imitate*. arXiv: 2501.17703 [cs.CL]. URL: <https://arxiv.org/abs/2501.17703>.
- Yang, Ling et al. (2024). *SuperCorrect: Advancing Small LLM Reasoning with Thought Template Distillation and Self-Correction*. arXiv: 2410.09008 [cs.CL]. URL: <https://arxiv.org/abs/2410.09008>.
- McAleese, Nat et al. (2024). *LLM Critics Help Catch LLM Bugs*. arXiv: 2407.00215 [cs.CL]. URL: <https://arxiv.org/abs/2407.00215>.
- Xie, Zhihui et al. (2025). *Teaching Language Models to Critique via Reinforcement Learning*. arXiv: 2502.03492 [cs.CL]. URL: <https://arxiv.org/abs/2502.03492>.
- Cobbe, Karl et al. (2021). *Training Verifiers to Solve Math Word Problems*. arXiv: 2110.14168 [cs.LG].
- Qwen-Team (2024). *Qwen2.5 Technical Report*. arXiv: 2412.15115 [cs.CL].
- Hendrycks, Dan et al. (2021). *Measuring Mathematical Problem Solving With the MATH Dataset*. arXiv: 2103.03874 [cs.LG].