LEARNING FROM NEGATIVE FEEDBACK, OR POSITIVE FEEDBACK OR BOTH: A Review and Analysis of the DSL501-PROJECT Implementation

Team Name: He He!! We cook real stuff
IIT Bhilai
India

Abstract

This report reviews and analyzes the open-source repository "DSL501-PROJECT," which implements works upon learning from negative feedback ,positive feedback or both rather than only positive feedback which has been the conventional method till date. The project explores a hybrid objective combining positive preference alignment, negative rejection signals, and KL-regularization. This report summarizes the theoretical foundations, evaluates the design and implementations as well as coding the part described the research paper becuase the code has been not provided several of the wrok has been done on developing the code and testing it on a lightweight environment given our constraints(for ex:- rgb stacking requires almost 300-400 gb of memory to train) thus in some of the experiments such as bandit rl we have not exactly achieved the same but implemented the algo according to the paper. Where as in PMPO we have achieved better than the benchmark. Our method technically enables broader applicability in settings such as reinforcement learning, offline RL, and LLM alignment, where structured paired data is not always feasible.

ACM Reference Format:
Team Name: He He!! We cook real stuff. 2025. LEARNING FROM NEGATIVE FEEDBACK, OR POSITIVE
FEEDBACK OR BOTH:
A Review and Analysis of the DSL501-PROJECT Implementation. In . ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The use of preference annotated data for training machine learning models has a long history going back to early algorithms for recommender systems.Nowadays preference optimization algorithms are being heavely used in Deep learning models such as Large Language Models.Thye are also used in reinforcement learning as majority of reinforcement learning is about make decisions to maximise results thus preference optimisation also plays a crucial role in reinforcement learning as well .Preference based optimization is beneficial when feedback from human is not available but but one instead aims to optimize a machine learning model based on feedback from a hand-coded or learned critic function.Preference methods are more of use since they let us optimize models to use acheive desirable results on the basis of relative ranking between the probable outcomes availaible.

Several Preference optimization analogyms are available amongst which Direct Preference Optimisation is quite popular as it is an offline opitmization algorithm which works on the basis of paired dataset .however the problem being that the requirement of a dataset that is tailored according to the task's domain.Now, this becomes a bottleneck in case of LLMs, rl based enivronments where the domain and choices are not finite thus pair based dataset becomes redundant in these specific scenarios. Due to which the need of unpaired responses arises which has been addressed by the research paper :-LEARNING FROM NEGATIVE FEEDBACK, OR POSITIVE FEEDBACK OR BOTH where they have tried to optimise perference based algorithms using unpiared preferred or dis-preferred outcomes.The method described in the paper can learn both positive and negative values. What we have done is that we have implemented the methods given in the paper using the code written by us and then tested on the benchmarks give and we have also implemented some former methods such as EXPECTANCY MAXIMISATION so as to compare different methods side by side and to show the improvements we get as described by the paper although we might not have been able to reach some of the benchmarks due to some constrains but we have implemented and tested those experiments according to the methodology described by the paper and even exceeded the benchmark expectaction in LLM accept/reject response .During training we used different methods of divergence such as rene divergence along with the standard kl divergence so as to keep the function sane during the training and not deviate much from the former distribution. The main experiments conducted by us, which were also given in the paper, are as follows:

- LLM response acceptance using PMPO.
- RL bandit experiment.
- Control Suite experiment.
- RGB stacking using synthetic accept–reject data.
- Expectancy maximisation using Gaussian mixture models.

. We used LMSYS 1-M Dataset for most of our work with some data preprocessing tailored according to our needs .We used that dataset as it consists of chat beween humans and several llms with responses and the chat being rated on several scenarios thus it gives us a mixed response dataset thus making it easy for use to perform our experiments especially PMPO AND EM.

Contributions

[leftmargin=*]

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- **Member 1:** Ajay Chikate, 12340580 – Implementation of PMPO and divergence-based training experiments, LLM as a judge.
- **Member 2:** Farhan Alam, 12340740 – Dataset preprocessing, implementation of EM and Bandit RL, Control Suite
- **Member 3:** Pobitro Bhattacharya, 12341580 – Evaluation metrics, benchmarking against DPO and analysis.
- **Member 4:** Sidhesh Kumar Patra, 12342060 – LLM fine-tuning, mathematical formulation, KTO, exploring different regularizers and divergences.

2 Related Work

2.1 Maximum a Posteriori Policy Optimization

2.1.1 Overview and notation. We summarise MPO (Maximum a Posteriori Policy Optimization) in a compact, self-contained derivation. MPO is a principled EM / constrained-optimization method for policy improvement that (i) constructs an improved non-parametric policy q in a trust-region around a current policy π_{old} (E-step), and (ii) fits a parametric policy π_θ to q (M-step). MPO naturally handles continuous or discrete actions and admits a sampled implementation suitable for large action spaces and sequence models.

Notation:

- x — state / context (in language modelling: the prompt).
- y — action / candidate output (in sequence models: the completion).
- $\pi_\theta(y|x)$ — parametric policy to be learned (parameters θ).
- $\pi_{\text{old}}(y|x)$ — current (behavior) policy used for sampling.
- $Q(x, y)$ — action-value function (reward or score) for candidate y in context x.
- $\eta > 0$ — E-step temperature (dual variable / Lagrange multiplier).
- ϵ — KL budget (trust-region size) used to constrain the E-step.

2.1.2 MPO as a constrained optimization. For a fixed state x one may want to find a new (non-parametric) policy $q(\cdot \mid x)$ that improves expected value while staying close to π_{old}. Formulate:

$$\begin{aligned} \max_{q(\cdot|x)} \quad & \mathbb{E}_{y\sim q(\cdot|x)}\big[Q(x,y)\big] \\ \text{subject to} \quad & D_{\text{KL}}\big(q(\cdot \mid x) \,\|\, \pi_{\text{old}}(\cdot \mid x)\big) \le \epsilon, \\ & \sum_y q(y \mid x) = 1, \qquad q(y \mid x) \ge 0 \ \forall y. \end{aligned}$$

This is a convex optimization over the distribution q for fixed Q and π_{old}. Introduce Lagrange multipliers: $\eta \ge 0$ for the KL constraint and λ for normalization. The Lagrangian is

$$\mathcal{L}(q,\eta,\lambda) = \sum_y q(y)Q(x,y) - \eta\Big(\sum_y q(y)\log\frac{q(y)}{\pi_{\text{old}}(y)} - \epsilon\Big) + \lambda\Big(\sum_y q(y) - 1\Big).$$

2.1.3 E-step solution (closed form). Take derivative w.r.t. $q(y)$ and set to zero:

$$\frac{\partial \mathcal{L}}{\partial q(y)} = Q(x,y) - \eta\Big(\log\frac{q(y)}{\pi_{\text{old}}(y)} + 1\Big) + \lambda = 0.$$

Rearrange to obtain:

$$\log q(y) = \log \pi_{\text{old}}(y) + \frac{1}{\eta}Q(x,y) + C,$$

where C is a scalar (absorbing λ and constants). Exponentiating and normalizing yields the closed-form optimal $q^\star$:

$$\boxed{q^\star(y \mid x) = \frac{\pi_{\text{old}}(y \mid x)\ \exp\big(\frac{1}{\eta}Q(x,y)\big)}{\sum_{y'} \pi_{\text{old}}(y' \mid x)\ \exp\big(\frac{1}{\eta}Q(x,y')\big)}}$$

This is the familiar MPO E-step: reweight the old policy by exponentiated (scaled) Q-values.

The dual parameter η is chosen to satisfy the KL budget:

$$D_{\text{KL}}(q^\star(\cdot \mid x) \,\|\, \pi_{\text{old}}(\cdot \mid x)) = \epsilon,$$

if an explicit budget ϵ is enforced. Equivalently, η can be treated as a temperature hyper-parameter.

2.1.4 Practical sampled form. When the action space is large or continuous, sample K candidates $\{y_k\}k = 1^K$ from the behavior policy πold$(\cdot \mid x)$. Denote $Q_k := Q(x, y_k)$. The unnormalized weights from the E-step for sampled candidates are:

$$\tilde{w}_k = \pi_{\text{old}}(y_k \mid x)\exp\big(\tfrac{1}{\eta}Q_k\big).$$

Because y_k are drawn from π_{old}, the $\pi_{\text{old}}(y_k)$ factor cancels in an importance-sampling estimator and we may use the simpler normalized softmax over Q_k/η:

$$\boxed{w_k = \frac{\exp(Q_k/\eta)}{\sum_{j=1}^K \exp(Q_j/\eta)} \qquad \text{(sampled approximation)}}$$

These weights approximate the E-step's $q^\star$ mass on the sampled candidates.

2.1.5 M-step: projection to parametric policy (weighted ML / KL projection). The M-step fits the parametric policy π_θ to the improved non-parametric policy $q^\star$. A natural objective is the (per-state) maximum likelihood / KL projection:

$$\theta^\star = \arg\min_\theta \sum_x D_{\text{KL}}\big(q^\star(\cdot \mid x) \,\|\, \pi_\theta(\cdot \mid x)\big),$$

which is equivalent to maximising the expected log-likelihood under $q^\star$:

$$\theta^\star = \arg\max_\theta \sum_x \mathbb{E}y \sim q^\star(\cdot \mid x)\big[\log \pi\theta(y \mid x)\big].$$

In the sampled approximation this becomes minimizing the negative weighted log-likelihood:

$$\boxed{\mathcal{L}\text{M}(\theta) = -\sum i \sum_{k=1}^K w_{i,k}\ \log \pi_\theta(y_{i,k} \mid x_i),}$$

where i indexes states/prompts in the batch and $w_{i,k}$ are E-step weights for that state.

2.1.6 Connection to EM / dual viewpoint. MPO can be seen as a coordinate-ascent procedure on a variational lower bound:

$$\begin{aligned} &\log\Big(\sum_y \pi_\theta(y \mid x)\exp\big(\tfrac{1}{\eta}Q(x,y)\big)\Big) \\ &\quad \ge \sum_y q(y \mid x)\Big(\log \pi_\theta(y \mid x) + \tfrac{1}{\eta}Q(x,y) - \log q(y \mid x)\Big). \end{aligned}$$

The E-step chooses q to maximize RHS (giving the closed form above), and the M-step maximizes the RHS w.r.t. θ (weighted ML).

Thus MPO is an instance of EM/variational optimization with a particular form for the latent posterior q.

2.1.7 Solving for η. If a KL constraint $D_{\mathrm{KL}}(q^\star \| \pi_{\mathrm{old}}) = \epsilon$ is required, then η is the dual that ensures this constraint. In the sampled approximation, define

$$\mathrm{KLest}(\eta) \;=\; \sum k = 1^K w_k(\eta) \log \frac{w_k(\eta)}{b_k}, \qquad b_k = \frac{\pi_{\mathrm{old}}(y_k \mid x)}{\sum_j \pi_{\mathrm{old}}(y_j \mid x)},$$

and solve $\mathrm{KLest}(\eta) = \epsilon$ for η (monotone root-finding). When sampling from πold, $b_k \approx 1/K$ often and $\mathrm{KL}_{\mathrm{est}}$ reduces to negative entropy of w up to constants. In practice one usually:

- solve η per-state (per x) to satisfy the KL budget, or
- choose a global η as a hyperparameter, or
- use a single η per minibatch (compromise).

2.1.8 Gradient form (for the M-step). The gradient of the M-step loss (sampled weighted NLL) is:

$$\nabla_\theta \mathcal{L}_{\mathrm{M}}(\theta) = -\sum_i \sum_k w_{i,k} \nabla_\theta \log \pi_\theta(y_{i,k} \mid x_i).$$

For autoregressive sequence models this is implemented by multiplying token-level cross-entropy losses by the scalar weight $w_{i,k}$ for the entire example and backpropagating.

2.1.9 Practical considerations and numerical stability.

- **Softmax stability:** subtract $\max_k Q_k$ before exponentiation to avoid overflow: $\exp((Q_k - \max_j Q_j)/\eta)$.
- **Weight clipping / smoothing:** to avoid domination by a single candidate, clip $w_k \in [w_{\min}, 1]$ or apply temperature smoothing.
- **Response-only log-probs:** compute $\log \pi_\theta(y \mid x)$ only on the response token span (exclude prompt tokens) to focus updates.
- **Sampling policy:** sample candidates from π_{old} (or a slightly more exploratory policy) to ensure diversity.
- **LoRA / adapters:** for large models use parameter-efficient fine-tuning (LoRA) and small learning rates for M-step updates.
- **Batching and per-state normalization:** compute softmax normalization and weight computation per state x in the batch.

Algorithm 1 Sampled MPO (minibatch)

behavior policy π_{old}, parametric policy π_θ, score Q, samples per state K, KL budget ϵ or temperature η. each training step sample a minibatch of states $\{x_i\}_{i=1}^B$ each $i = 1, \ldots, B$ draw $y_{i,1}, \ldots, y_{i,K} \sim \pi_{\mathrm{old}}(\cdot \mid x_i)$ compute scores $Q_{i,k} := Q(x_i, y_{i,k})$ solve for η_i (or use global η) compute weights $w_{i,k} \leftarrow \mathrm{softmax}k(Qi, k/\eta_i)$ compute loss $\mathcal{L} \leftarrow -\sum_{i=1}^B \sum_{k=1}^K w_{i,k} \log \pi_\theta(y_{i,k} \mid x_i)$ take gradient step on θ: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$ optionally update $\pi_{\mathrm{old}} \leftarrow \pi_\theta$ or use EMA

2.1.10 Sampled MPO algorithm (pseudocode).

2.1.11 Relation to other algorithms.

- **MPO vs PPO:** MPO performs a per-state trust-region update via an explicit KL-constrained re-weighting and a supervised M-step; PPO enforces a clipped objective / approximate trust-region via policy ratios and advantage estimates.
- **MPO and EM:** MPO corresponds to coordinate ascent of a variational ELBO where the "latent" distribution is the improved non-parametric policy q.
- **MPO and MPO-family for sequence models:** in language-model alignment, MPO is adapted by using sampled completions, reward signals (e.g. human preference scores), and per-example weightings; it is closely related to PMPO used in LLM literature.

2.2 Direct Preference Optimization

Direct Preference Optimization (DPO) is an offline, RL–free method for aligning a policy model π_θ using human preference pairs. Each training sample is a triple (x, y^+, y^-) where:

- x is the prompt,
- y^+ is the preferred (chosen) response,
- y^- is the dispreferred (rejected) response.

Policy and Reference Models. Let:

$$\pi_\theta(y|x) \quad \text{be the trainable policy model}$$

and

$$\pi_{\mathrm{ref}}(y|x) \quad \text{be a frozen reference model.}$$

The objective of DPO is to

$$\text{increase} \quad \pi_\theta(y^+|x) \qquad \text{and decrease} \quad \pi_\theta(y^-|x),$$

while keeping the updated policy close to the reference model.

Reward Model Definition. DPO implicitly defines a reward model using:

$$r_\theta(x, y) = \beta \left[\log \pi_\theta(y|x) - \log \pi_{\mathrm{ref}}(y|x)\right],$$

where $\beta > 0$ is a temperature hyperparameter that controls the KL penalty and stability of learning.

Pairwise Preference Objective. Human preference labels imply:

$$y^+ \succ y^- \quad \text{(i.e. } y^+ \text{ is preferred over } y^-\text{)}.$$

DPO maximizes the probability that the reward model ranks y^+ higher than y^-. This leads to the pairwise logistic objective:

$$\mathcal{L}_{\mathrm{DPO}} = -\mathbb{E}_{(x,y^+,y^-)} \left[\log \sigma\left(r_\theta(x, y^+) - r_\theta(x, y^-)\right)\right],$$

where $\sigma(\cdot)$ is the logistic sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

Substituting the Reward Expression. Plugging in the definition of $r_\theta(x, y)$:

$$r_\theta(x, y^+) - r_\theta(x, y^-) = \beta\Big[\log \pi_\theta(y^+|x) - \log \pi_\theta(y^-|x) - \left(\log \pi_{\mathrm{ref}}(y^+|x) - \log \pi_{\mathrm{ref}}(y^-|x)\right)\Big].$$

Thus the final DPO loss becomes:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y^+,y^-)}\Big[\log\sigma\Big(\beta\big[\log\pi_\theta(y^+|x) - \log\pi_\theta(y^-|x) - (\log\pi_{\text{ref}}(y^+|x) - \log\pi_{\text{ref}}(y^-|x))\big]\Big)\Big]$$

This is the exact equation implemented in modern DPO training libraries.

Interpretation.

- The term
$$\log\pi_\theta(y^+|x) - \log\pi_\theta(y^-|x)$$
increases when the model prefers y^+ over y^-.
- The reference KL constraint is enforced implicitly:
$$\log\pi_{\text{ref}}(y^+|x) - \log\pi_{\text{ref}}(y^-|x)$$
- β controls how strongly the policy is allowed to deviate from the reference model.
- No explicit KL-divergence term is required.

Key Properties.

- Purely offline — no environment, no reinforcement learning.
- Uses human preference pairs instead of scalar rewards.
- Avoids training an explicit reward model.
- Stable and efficient due to implicit KL regularization.

2.3 KTO: Binary-label optimization

We denote a dataset of examples (x, y, z) where x is a prompt, y is a candidate response and $z \in \{0, 1\}$ is a binary feedback label (1 = desirable, 0 = undesirable). Let $\pi_\theta(y \mid x)$ be the parametric policy and $\pi_{\text{ref}}(y \mid x)$ a frozen reference. Define

$$\Delta_\theta(x, y) := \log\pi_\theta(y \mid x), \qquad \Delta_{\text{ref}}(x, y) := \log\pi_{\text{ref}}(y \mid x).$$

Using a temperature $\beta > 0$ we form the scalar score

$$s_\theta(x, y) = \beta\big(\Delta_\theta(x, y) - \Delta_{\text{ref}}(x, y)\big), \qquad p_\theta(z = 1 \mid x, y) = \sigma\big(s_\theta(x, y)\big),$$

where $\sigma(u) = 1/(1+e^{-u})$ is the logistic function. KTO minimizes the weighted negative log-likelihood (weighted binary cross-entropy)

$$\mathcal{L}(\theta) = -\sum_i \Big[\, w_1 z_i \log\sigma(s_\theta(x_i, y_i)) + w_0(1 - z_i)\log\big(1 - \sigma(s_\theta(x_i, y_i))\big)\Big],$$

with positive weight w_1 and negative weight w_0 (these correspond to `desirable_weight` and `undesirable_weight`). The per-example gradient can be written compactly as

$$\frac{\partial\ell_i}{\partial\theta} = \big(-w_1 z_i(1 - p_i) + w_0(1 - z_i)p_i\big)\beta\,\nabla_\theta\Delta_\theta(x_i, y_i), \quad p_i = \sigma(s_\theta(x_i, y_i)).$$

This shows positive examples push $\log\pi_\theta(y \mid x)$ upward whereas negative examples push it downward; the multiplier $(1 - p_i)$ or p_i implements diminishing returns.

3 Background

PMPO is an EM-style algorithm that constructs a non-parametric improved policy q (E-step) by exponentiating a score Q and reweighting the current policy, and then fits the parametric policy π_θ to q (M-step). In sampled, practical form the E-step reduces to a softmax over scores, producing weights used in a weighted supervised update. The algorithm unifies trust-region MPO ideas with preference learning for sequence models and naturally handles paired, positive-only, and negative-only data through appropriate choices of Q and the signed likelihood regularized objective.

3.1 Expectation–Maximization (EM) Algorithm

3.1.1 Problem Setup. Let X denote the observed data and Z denote the latent (unobserved) variables. We assume a parametric model with joint distribution

$$p_\theta(X, Z),$$

where θ are the parameters to be estimated. Our goal is to maximize the marginal log-likelihood

$$\mathcal{L}(\theta) = \log p_\theta(X) = \log\sum_Z p_\theta(X, Z).$$

Direct maximization is typically difficult due to the sum/integral over latent variables.

3.1.2 Variational Decomposition. For any distribution $q(Z)$ over the latent space, we can write:

$$\begin{aligned}
\mathcal{L}(\theta) &= \log\sum_Z q(Z)\frac{p_\theta(X, Z)}{q(Z)} && (1)\\
&= \log\mathbb{E}q(Z)\left[\frac{p\theta(X, Z)}{q(Z)}\right] && (2)\\
&\geq \mathbb{E}q(Z)[\log p\theta(X, Z)] - \mathbb{E}_{q(Z)}[\log q(Z)] && (3)\\
&=: \mathcal{F}(q, \theta), && (4)
\end{aligned}$$

where the inequality comes from Jensen's inequality. Thus,

$$\mathcal{L}(\theta) \geq \mathcal{F}(q, \theta).$$

$\mathcal{F}(q, \theta)$ is the Evidence Lower Bound (ELBO).

We also obtain the exact decomposition:

$$\mathcal{L}(\theta) = \mathcal{F}(q, \theta) + D_{\text{KL}}\big(q(Z) \,\|\, p_\theta(Z \mid X)\big),$$

where the KL term is always non-negative.

3.1.3 EM Algorithm as Coordinate Ascent on the ELBO. The EM algorithm alternates between two steps:

E-step (Optimize ELBO w.r.t. q):

$$q^{(t+1)} = \arg\max_q \mathcal{F}(q, \theta^{(t)}).$$

Taking the functional derivative of $\mathcal{F}(q, \theta)$ w.r.t. $q(Z)$ gives:

$$q^{(t+1)}(Z) = p_{\theta^{(t)}}(Z \mid X),$$

i.e., the posterior under current parameters.

Thus, the E-step sets

$$q^{(t+1)}(Z) = p_{\theta^{(t)}}(Z \mid X).$$

M-step (Optimize ELBO w.r.t. θ):

$$\theta^{(t+1)} = \arg\max_\theta \mathcal{F}(q^{(t+1)}, \theta).$$

Substitute the E-step posterior into the ELBO:

$$\mathcal{F}(q^{(t+1)}, \theta) = \mathbb{E}p\theta^{(t)}(Z \mid X)\big[\log p_\theta(X, Z)\big] + \text{const}.$$

Thus, the M-step maximizes:

$$\theta^{(t+1)} = \arg\max_\theta \mathbb{E}p\theta^{(t)}(Z \mid X)\,[\log p_\theta(X, Z)]\,.$$

3.1.4 Full EM Update Equations.

E-step: Posterior of latent variables.

$$q^{(t+1)}(Z) = p_{\theta^{(t)}}(Z \mid X) = \frac{p_{\theta^{(t)}}(X, Z)}{\sum_{Z'} p_{\theta^{(t)}}(X, Z')}.$$

M-step: Maximize expected complete log-likelihood.

$$\theta^{(t+1)} = \arg\max_{\theta} \mathbb{E}Z \sim p\theta^{(t)}(Z \mid X) \left[\log p_\theta(X, Z)\right].$$

3.1.5 Monotonic Improvement Guarantee.

From the decomposition

$$\mathcal{L}(\theta) = \mathcal{F}(q, \theta) + D_{\mathrm{KL}}(q||p_\theta(Z \mid X)),$$

steps of EM ensure:

$$\mathcal{L}(\theta^{(t+1)}) \geq \mathcal{L}(\theta^{(t)}).$$

Thus EM is guaranteed to not decrease the marginal likelihood.

3.1.6 Summary of EM Algorithm.

E-step: $q^{(t+1)}(Z) = p_{\theta^{(t)}}(Z \mid X)$

M-step: $\theta^{(t+1)} = \arg\max_{\theta} \mathbb{E}_{Z \sim q^{(t+1)}} \left[\log p_\theta(X, Z)\right].$

3.2 Preference Maximum a Posteriori Optimization (PMPO)

3.2.1 Notation and problem statement.

Let x denote a context (prompt) and y denote a candidate completion (response). We denote by $\pi_\theta(y \mid x)$ the parametric policy (the model we train, parameters θ) and by $\pi_{\mathrm{ref}}(y|x)$ a frozen reference policy (pretrained base model). Let $Q(x, y)$ denote a scalar score ("value" or "reward") for output y in context x. In different setups Q is a learned reward model, a human-derived score, or a label-derived constant (e.g. +1 for chosen, −1 for rejected). We write expectations over discrete or continuous y as sums or integrals as appropriate.

The high-level objective is to *maximize the probability of positive outcomes* under the policy:

$$\max_{\theta} \sum_{x \in \mathcal{D}} \log P_\theta(\mathrm{pos} \mid x), \qquad P_\theta(\mathrm{pos} \mid x) := \mathbb{E}y \sim \pi\theta(\cdot \mid x)\big[p_{\mathrm{pos}}(y \mid x)\big],$$

where $p_{\mathrm{pos}}(y \mid x)$ is the (possibly soft) probability that y is judged positive. Our derivation below treats p_{pos} via a score Q and gives a practical EM / constrained-optimization algorithm.

3.2.2 EM lower bound (variational formulation).

For a fixed x define the evidence

$$\mathcal{L}x(\theta) := \log\Big(\sum_y \pi\theta(y \mid x)\, p_{\mathrm{pos}}(y \mid x)\Big).$$

Introduce an arbitrary distribution $q(y \mid x)$ (a variational posterior) with $\sum_y q(y \mid x) = 1$. By Jensen's inequality / the standard variational decomposition we obtain the ELBO:

$$\begin{aligned}\mathcal{L}_x(\theta) &= \log\left(\sum_y q(y \mid x) \frac{\pi_\theta(y \mid x)\, p_{\mathrm{pos}}(y \mid x)}{q(y \mid x)}\right) \\ &\geq \sum_y q(y \mid x)\left(\log \pi_\theta(y \mid x) + \log p_{\mathrm{pos}}(y \mid x) - \log q(y \mid x)\right) \\ &=: \mathcal{F}_x(q, \theta).\end{aligned}$$

Summing over prompts gives the global ELBO $\mathcal{F}(q, \theta) = \sum_x \mathcal{F}_x(q, \theta)$. The EM algorithm alternates:

- **E-step:** maximize $\mathcal{F}$ w.r.t. q (pointwise in x);
- **M-step:** maximize $\mathcal{F}$ w.r.t. θ.

3.2.3 E-step: closed-form solution.

Fix θ. For a single x we maximize

$$\sum_y q(y \mid x)\left(\log \pi_\theta(y \mid x) + \log p_{\mathrm{pos}}(y \mid x) - \log q(y \mid x)\right)$$

subject to $\sum_y q(y \mid x) = 1$. This is a standard constrained optimization whose solution is

$$q^\star(y \mid x) = \frac{\pi_\theta(y \mid x)\, p_{\mathrm{pos}}(y \mid x)}{Z_\theta(x)},$$
$$Z_\theta(x) := \sum_y \pi_\theta(y \mid x)\, p_{\mathrm{pos}}(y \mid x).$$

Thus the E-step reweights the current policy π_θ by the positive probability p_{pos}.

3.2.4 M-step: weighted maximum likelihood.

With q fixed the ELBO reduces to

$$\mathcal{F}(q, \theta) = \sum_x \mathbb{E}y \sim q(\cdot \mid x)\left[\log \pi\theta(y \mid x)\right] + \mathrm{const}(q),$$

so the M-step is the weighted maximum likelihood (supervised) update:

$$\theta \leftarrow \arg\max_{\theta} \sum_x \sum_y q(y \mid x) \log \pi_\theta(y \mid x).$$

This can be implemented by sampling $y \sim q$ or by directly applying per-sample weights $q(y \mid x)$ in a batched log-likelihood gradient.

3.2.5 Expressing p_{pos} via exponentiated Q (MPO connection).

A common and useful modeling choice is to express $p_{\mathrm{pos}}(y \mid x)$ in exponential form using a score $Q(x, y)$ and a temperature $\eta > 0$:

$$p_{\mathrm{pos}}(y \mid x) \propto \exp\Big(\frac{1}{\eta} Q(x, y)\Big).$$

If Q is unknown, it may be provided by a reward model or defined heuristically; if only positive examples are available one can set Q large for positives and small elsewhere.) Substituting into the E-step yields the classical MPO improvement:

$$q^\star(y \mid x) \;\propto\; \pi_\theta(y \mid x)\, \exp\Big(\frac{1}{\eta} Q(x, y)\Big)$$

with normalization $Z_\theta(x) = \sum_y \pi_\theta(y \mid x) \exp(Q(x, y)/\eta)$. This form can also be obtained by solving

$$\max_q \mathbb{E}q[Q] \quad \text{s.t.} \quad D\mathrm{KL}(q \parallel \pi_\theta) \leq \epsilon$$

via a Lagrangian with multiplier η (see below).

3.2.6 Dual/KL interpretation and derivation (trust-region form).

Consider the constrained optimization problem (fixed x):

$$\max_{q(\cdot)} \mathbb{E}q[Q] \quad \text{s.t.} \quad D\mathrm{KL}(q\|\pi_\theta) \leq \epsilon.$$

Forming the Lagrangian with multiplier $\eta > 0$:

$$\mathcal{L}(q, \eta) = \sum_y q(y)Q(y) - \eta\Big(\sum_y q(y) \log \frac{q(y)}{\pi_\theta(y)} - \epsilon\Big) + \lambda\Big(\sum_y q(y) - 1\Big),$$

differentiating w.r.t. $q(y)$ and solving the stationarity condition yields

$$q(y) \propto \pi_\theta(y) \exp\Big(\frac{1}{\eta} Q(y)\Big),$$

consistent with (3.2.5). The dual variable η trades off expected Q and divergence; for a specified KL budget ϵ there is a unique η (monotone relationship), which can be solved by scalar root-finding (e.g. binary search) in the sampled approximation.

3.2.7 Practical sampled approximation (LLMs). Enumerating all y is impossible for large-output spaces. Practically, for each x we sample a finite set of K candidates $\{y_k\}k = 1^K$ from a proposal distribution (typically πold$(\cdot \mid x)$ or π_{ref}). For brevity we denote $Q_k := Q(x, y_k)$ and $\pi_k := \pi_{\text{old}}(y_k \mid x)$. The discrete approximation to the E-step is:

$$\tilde{w}_k \;=\; \pi_k \exp\Big(\frac{1}{\eta} Q_k\Big), \qquad w_k \;=\; \frac{\tilde{w}_k}{\sum_{j=1}^K \tilde{w}_j}.$$

If the candidates y_k were sampled from π_{old}, the π_k factor cancels in importance sampling and one may use the softmax over Q_k directly:

$$\boxed{w_k \;=\; \frac{\exp(Q_k/\eta)}{\sum_{j=1}^K \exp(Q_j/\eta)}}$$

These w_k serve as training weights in the M-step.

3.2.8 M-step in sampled form (weighted supervised loss). Given candidate sets and weights $\{(y_{i,k}, w_{i,k})\}$ for each prompt x_i, the M-step maximizes

$$\mathcal{J}(\theta) \;=\; \sum_i \sum_{k=1}^K w_{i,k} \log \pi_\theta(y_{i,k} \mid x_i).$$

Equivalently, minimize the negative weighted log-likelihood:

$$\mathcal{L}\text{M}(\theta) \;=\; -\sum i \sum_k w_{i,k} \log \pi_\theta(y_{i,k} \mid x_i).$$

Gradients are straightforward:

$$\nabla_\theta \mathcal{L}_{\text{M}}(\theta) = -\sum_i \sum_k w_{i,k}\, \nabla_\theta \log \pi_\theta(y_{i,k} \mid x_i).$$

3.2.9 Choosing / solving for η. When η is not fixed, it is determined by the KL budget ϵ through

$$\text{KL}(q^\star \| \pi_\theta) = \epsilon.$$

In the sampled case, one can define the (approximate) KL using the finite set:

$$\text{KL}_{\text{est}}(\eta) = \sum_k w_k(\eta) \log \frac{w_k(\eta)}{\pi_k / \sum_j \pi_j},$$

or more simply use the entropy of w as a proxy. Because KL_{est} is monotone in η, solve for η via binary search until the KL constraint is met (or until a maximum/minimum η is reached). In many implementations η is treated as a hyperparameter and tuned.

3.2.10 Handling positive-only and negative-only data.

Positive-only. If the dataset contains only examples labeled positive $\{(x, y)\}$, set $Q(x, y)$ large for labeled positives (e.g. $Q = +1$ or a learned reward). Sample additional candidates y' for each x from π_{old} with $Q(y')$ small (e.g. 0), compute softmax weights (3.2.7) and perform the M-step. This concentrates weight on known positives while retaining a KL trust-region.

Negative-only. If only negatives are available, one wants to *decrease* the probability of bad responses. One principled option is to set $Q(y^-)$ negative (e.g. -1) for observed bad y^- and sample other candidates with neutral Q. The E-step will assign smaller weights to negatives and larger weights to other candidates; in practice many implementations instead perform a complementary objective that directly minimizes $\log \pi_\theta(y^-|x)$ with a KL regularizer. Concretely the unified signed-likelihood objective used in many LLM alignment works is:

$$\mathcal{L}_{\text{signed}}(\theta) = -\alpha\, r\, \log \pi_\theta(y \mid x) + \beta\, D_{\text{KL}}\big(\pi_\theta(\cdot \mid x) \,\big\|\, \pi_{\text{ref}}(\cdot \mid x)\big),$$

where $r \in \{+1, -1\}$ denotes positive/negative label and α scales the learning rate for sign; choosing α smaller for negatives improves stability (see experiments and discussion).

3.2.11 Per-token / response-only variants. In practice one often computes $\log \pi_\theta(y \mid x)$ only over the response token span (excluding the prompt) to avoid spurious gradients coming from prompt encoding. If $y = (y_1, \ldots, y_L)$ is tokenized and tokens indices for the response are $t_1, \ldots, t_m$, then

$$\log \pi_\theta(y \mid x) \;=\; \sum_{t \in \text{response indices}} \log \pi_\theta(y_t \mid x, y_{<t}),$$

and the M-step uses these per-response log-likelihoods. The gradient of a weighted loss is simply the weighted sum of token-level gradients.

3.2.12 KL approximations. Computing full sequence-wise KL between π_θ and π_{ref} is expensive; a common approximation is to use the dataset sample y as a Monte Carlo sample:

$$D_{\text{KL}}(\pi_{\text{ref}} \| \pi_\theta; x) \approx \mathbb{E} y \sim p\text{data}(\cdot \mid x)\big[\log \pi_{\text{ref}}(y \mid x) - \log \pi_\theta(y \mid x)\big].$$

When applied as a per-sample regularizer this yields a tractable term proportional to $\log \pi_\theta(y \mid x)$ (constant terms drop out of gradients), and this approximation is widely used in DPO/KTO/PMPO-style objectives.

Algorithm 2 Sampled PMPO (batched)

dataset of prompts $\{x_i\}$, initial policy π_{old}, score $Q(x, y)$ (or reward model), sample count K, dual η (or KL budget ϵ), learning rate γ each training iteration each mini-batch of prompts $\{x_i\}_{i=1}^B$ each $i = 1, \ldots, B$ sample $y_{i,1}, \ldots, y_{i,K} \sim \pi_{\text{old}}(\cdot \mid x_i)$ compute scores $Q_{i,k} = Q(x_i, y_{i,k})$ compute weights $w_{i,k} \leftarrow$ softmax$k(Qi, k/\eta)$ Eq. (3.2.7) compute weighted loss $\mathcal{L} \leftarrow -\sum_{i,k} w_{i,k} \log \pi_\theta(y_{i,k} \mid x_i)$ take gradient step $\theta \leftarrow \theta - \gamma \nabla_\theta \mathcal{L}$ optionally set $\pi_{\text{old}} \leftarrow \pi_\theta$ (or use EMA)

3.2.13 Final algorithm (practical, batched).

3.2.14 Gradients and implementation details. The gradient of the M-step objective is

$$\nabla_\theta \mathcal{L}\mathrm{M}(\theta) = -\sum i \sum_k w_{i,k} \nabla_\theta \log \pi_\theta(y_{i,k} \mid x_i).$$

When π_θ is an autoregressive Transformer, $\nabla_\theta \log \pi_\theta(y \mid x)$ is computed via the usual token-level cross-entropy; multiply per-example loss by $w_{i,k}$ or equivalently multiply token-level losses by $w_{i,k}$ and sum.

For numerical stability:

(1) subtract $\max_k Q_{i,k}$ from $Q_{i,k}$ before exponentiation: $\exp((Q_{i,k} - \max_j Q_{i,j})/\eta)$.
(2) clip weights $w_{i,k}$ to a minimum $\epsilon_w > 0$ or apply temperature smoothing to avoid single-sample domination.
(3) optionally apply weight normalization per-batch to keep gradient norms stable.

3.2.15 Relation to DPO and KTO.

- **DPO:** Direct Preference Optimization optimizes a pairwise logistic objective on pairs (y^+, y^-) using the implicit reward $r_\theta(x, y) = \beta(\log \pi_\theta(y \mid x) - \log \pi_{\text{ref}}(y \mid x))$. DPO can be recovered as a special case of the PMPO family when Q is derived from pairwise labels and when the E-step is collapsed to the two-sample case.
- **KTO:** KTO treats individual (x, y) as binary-labeled examples and minimizes a weighted binary cross-entropy on the sigmoid of $\beta(\log \pi_\theta(y \mid x) - \log \pi_{\text{ref}}(y \mid x))$. KTO corresponds to maximizing the log-likelihood of a Bernoulli model for the positive label with score given by a relative log-likelihood; this is a discriminative (single-example) alternative to PMPO's generative EM view.

3.2.16 Practical recommendations and hyperparameters.

- η (E-step temperature) controls the sharpness of q. Small $\eta \Rightarrow$ very peaked q (aggressive improvement), large $\eta \Rightarrow$ conservative update.
- If working with *only negatives*, use a smaller α in the signed-loss variant or add stronger KL regularization to stabilize.
- Use LoRA / adapters for memory efficiency and small learning rates for the M-step.
- Compute response-only log-probs to focus learning on the generated tokens.
- Run multiple seeds and monitor PPL, KL vs reference, and preference accuracy (win-rate) to choose operating point.

3.2.17 Summary.

4 Codebase Overview

The repository is organized into modules including dataset processing, PMPO training utilities, configuration files, and example notebooks. The project primarily uses the Hugging Face `transformers` library.

4.1 Directory Structure

The repository includes:

- **Dataset/** – scripts and samples for preference datasets.
- **pmpo/** – implementation of PMPO loss, trainer class, and model wrappers.
- **configs/** – training configuration templates.
- **scripts/** – executable utilities for training and evaluation.
- Example Jupyter notebooks.

5 PMPO Training Framework

The central component of this project is the `PMPOTrainer` class. This trainer integrates with pretrained causal language models (e.g., GPT-2) and computes the PMPO loss across token-level log-probabilities.

The PMPO objective is formulated as:

$$\mathcal{L}_{PMPO} = \mathbb{E}[\log \sigma(\Delta - m)] + \beta\, KL(\pi_\theta \| \pi_0), \tag{5}$$

where Δ represents the log-probability difference between chosen and rejected responses, m is a margin parameter, and β controls KL regularization.

The implementation computes sequence log-probabilities, normalizes them, and applies logistic preference modeling.

5.1 Strengths of Implementation

- Clean modular structure for trainer and dataset pipeline.
- Integration with standard Hugging Face utilities.
- Support for margin-based preference tuning.
- KL-regularization compatible with standard LLM fine-tuning.

5.2 Limitations

- Documentation for hyperparameter choices remains incomplete.
- Lack of automated evaluation scripts.
- Missing experiments comparing PMPO against DPO, PPO, or ORPO baselines.

6 Dataset Considerations

The repository references the "lmsys-chat-1m" dataset, a large-scale collection of conversational pairs. The project includes only a sample subset. Full-scale training requires external dataset retrieval.

Preference data is expected in the form of:

```
prompt, chosen response, rejected response
```

The Dataset/ module provides tokenization and batching utilities, though further documentation would strengthen usability.

7 Methodology

7.1 Problem Formulation

We address challenge of aligning large language models (LLMs) with human preferences through a novel approach called as PMPO. Given a pre-trained language model π_{ref} and a dataset of preference pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, where x_i represents prompt, y_i denotes response.

Our objective is to fine-tune a policy model π_θ that maximizes the likelihood of preferred /positive responses while minimizing the likelihood of dispreferred/negative responses,with respect to a KL-divergence constraint that prevents the model from deviating too far from the reference distribution.

7.2 Dataset Preparation

7.2.1 Data Collection. Our training dataset consists of $N = 17,428$ unpaired responses pairs spanning diverse domains including educational content, conversational responses, and safety-critical scenarios. Each sample is structured as:

$$\mathcal{D}_i = \{p_i, r_i, \ell_i\} \tag{6}$$

where:

- $p_i \in \mathcal{P}$: Input prompt or instruction
- $r_i \in \mathcal{R}$: Generated response
- $\ell_i \in \{0, 1\}$: Binary preference label (1 = positive, 0 = negative)

7.2.2 Data Preprocessing. We format each training instance using an instruction-response template:

$$s_i = \text{“### Instruction:}\backslash n\text{”} \oplus p_i \oplus \text{“}\backslash n\backslash n\text{### Response:}\backslash n\text{”} \oplus r_i \tag{7}$$

where $\oplus$ denotes string concatenation. Subsequently, we tokenize each sequence using the GPT-Neo tokenizer with a maximum sequence length of 256 tokens:

$$\mathbf{t}_i = \text{Tokenize}(s_i) \in \mathbb{Z}_+^{L_i}, \quad L_i \leq 256 \tag{8}$$

The dataset is partitioned into training (95%, $n_{\text{train}} = 16,556$) and validation (5%, $n_{\text{val}} = 872$) splits using stratified random sampling to maintain label distribution.

7.3 Model Architecture

7.3.1 Base Model Selection. We have **GPT-Neo-1.3B** as our foundation model, comprising:

- Parameters: $|\theta| = 1.32 \times 10^9$
- Layers: $L = 24$ transformer blocks
- Hidden dimension: $d_{\text{model}} = 2048$
- Attention heads: $n_{\text{heads}} = 16$
- Vocabulary size: $|V| = 50,257$

7.3.2 Low-Rank Adaptation (LoRA). To enable parameter-efficient fine-tuning, we apply Low-Rank Adaptation (LoRA) . For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA decomposes the change as:

$$W = W_0 + \Delta W = W_0 + BA \tag{9}$$

where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ are low-rank matrices with rank $r \ll \min(d, k)$.

LoRA Hyperparameters:

- Rank: $r = 8$
- Scaling factor: $\alpha = 32$
- Target modules: {q_proj, k_proj, v_proj, out_proj}
- Dropout: $p_{\text{dropout}} = 0.05$

This configuration yields:

$$|\theta_{\text{trainable}}| = 3,145,728 \quad (\approx 0.24\% \text{ of total parameters}) \tag{10}$$

7.3.3 Quantization. We used 8-bit quantization using `bitsandbytes` to reduce memory consumption(especially VRAM in colab) during training. Weight matrices are quantized as:

$$\tilde{W} = Q_8(W) = \text{round}\left(\frac{W - \min(W)}{\max(W) - \min(W)} \times 255\right) \tag{11}$$

This reduces model memory requirements from ~5.3GB to ~1.5GB while maintaining numerical stability.

7.4 PMPO Training Algorithm

7.4.1 Objective Function. Our training objective combines three components: positive example loss, negative example loss, and KL-divergence regularization. Thus, we optimize:

$$\mathcal{L}_{\text{PMPO}}(\theta) = \alpha \mathcal{L}_{\text{pos}}(\theta) + (1 - \alpha)\mathcal{L}_{\text{neg}}(\theta) + \beta \mathcal{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \tag{12}$$

where:

- $\alpha \in [0, 1]$: Preference weighting coefficient
- $\beta > 0$: KL regularization coefficient
- π_θ: Policy model (trainable)
- π_{ref}: Reference model (frozen)

7.4.2 Cross-Entropy Loss Components. For a batch $\mathcal{B} = \{(\mathbf{x}_i, \mathbf{y}_i, \ell_i)\}_{i=1}^{|\mathcal{B}|}$, we compute per-token cross-entropy loss:

$$\mathcal{L}_{\text{CE}}(\mathbf{y}_i; \theta) = -\frac{1}{T_i} \sum_{t=1}^{T_i} \log \pi_\theta(y_{i,t} | \mathbf{y}_{i,<t}, \mathbf{x}_i) \tag{13}$$

where T_i is the sequence length and $\mathbf{y}_{i,<t}$ denotes all tokens before position t.

The positive and negative losses are computed as:

$$\mathcal{L}_{\text{pos}}(\theta) = \frac{1}{|\mathcal{B}^+|} \sum_{i:\ell_i=1} \mathcal{L}_{\text{CE}}(\mathbf{y}_i; \theta) \tag{14}$$

$$\mathcal{L}_{\text{neg}}(\theta) = \frac{1}{|\mathcal{B}^-|} \sum_{i:\ell_i=0} \mathcal{L}_{\text{CE}}(\mathbf{y}_i; \theta) \tag{15}$$

where $\mathcal{B}^+ = \{i : \ell_i = 1\}$ and $\mathcal{B}^- = \{i : \ell_i = 0\}$ are the positive and negative subsets.

7.4.3 KL Divergence Regularization. To prevent distribution collapse and maintain linguistic coherence, we constrain the policy model to remain close to the reference distribution so that it wont forget what it learn during its initial training and start generating anything irrevalent so we added KL divergence:

$$\mathcal{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) = \mathbb{E}_{\mathbf{y} \sim \pi_\theta}\left[\sum_{t=1}^{T} \sum_{v \in \mathcal{V}} \pi_\theta(v | \mathbf{y}_{<t}, \mathbf{x}) \log \frac{\pi_\theta(v | \mathbf{y}_{<t}, \mathbf{x})}{\pi_{\text{ref}}(v | \mathbf{y}_{<t}, \mathbf{x})}\right] \tag{16}$$

In practice, we approximate this expectation using the current batch:

$$\hat{\mathcal{D}}_{\text{KL}} = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \frac{1}{T_i} \sum_{t=1}^{T_i} \text{KL}\left(\pi_\theta(\cdot | \mathbf{y}_{i,<t}, \mathbf{x}_i) \| \pi_{\text{ref}}(\cdot | \mathbf{y}_{i,<t}, \mathbf{x}_i)\right) \tag{17}$$

7.4.4 Gradient Computation. The gradient of the combined objective with respect to the LoRA parameters is:

$$\nabla_\theta \mathcal{L}_{\text{PMPO}} = \alpha \nabla_\theta \mathcal{L}_{\text{pos}} + (1-\alpha)\nabla_\theta \mathcal{L}_{\text{neg}} + \beta \nabla_\theta \hat{\mathcal{D}}_{\text{KL}} \tag{18}$$

Note that $\nabla_\theta \hat{\mathcal{D}}_{\text{KL}}$ does not backpropagate through π_{ref} as it is frozen.

7.5 Training Configuration

Table 1: Training Hyperparameters

Parameter	Value
Preference weight (α)	0.7
KL coefficient (β)	0.05
Learning rate (η)	2×10^{-4}
Batch size (per device)	2
Gradient accumulation steps	4
Effective batch size	8
Number of epochs	1
Optimizer	Paged AdamW (8-bit)
Weight decay	0.0
Learning rate schedule	Constant
Warmup steps	0
Max sequence length	256 tokens
Precision	FP16 (mixed precision)

7.5.1 Hyperparameters.

7.5.2 Optimization Details. We also use AdamW optimizer with 8-bit quantized states to reduce memory overhead and faster convergence:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{19}$$

where $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected first and second moment estimates. The effective batch size is:

$$B_{\text{eff}} = B_{\text{device}} \times N_{\text{accum}} \times N_{\text{devices}} = 2 \times 4 \times 1 = 8 \tag{20}$$

Total training steps:

$$N_{\text{steps}} = \left\lceil \frac{n_{\text{train}} \times N_{\text{epochs}}}{B_{\text{eff}}} \right\rceil = \left\lceil \frac{16,556 \times 1}{8} \right\rceil = 2,070 \tag{21}$$

7.6 Implementation Details

7.6.1 Software Stack.

- **Framework:** PyTorch 2.8.0, HuggingFace Transformers 4.33.0
- **Efficient Training:** PEFT 0.7.0, bitsandbytes 0.41.0
- **Accelerator:** CUDA 12.6 on NVIDIA T4 GPU (16GB VRAM)
- **Precision:** Mixed precision (FP16) with automatic loss scaling

7.6.2 Memory Optimization.

(1) **8-bit Quantization:** Reduces model memory from 5.3GB to 1.5GB
(2) **LoRA Adapters:** Only backpropagate through 3.1M parameters
(3) **Gradient Checkpointing:** Trade computation for memory (disabled for speed)
(4) **Paged Optimizers:** Offload optimizer states to CPU when needed

Peak memory usage: ~12GB (model + optimizer states + activations)

7.7 Training Procedure

Algorithm 3 PMPO Training Algorithm

Require: Dataset $\mathcal{D}$, reference model π_{ref}, base model π_{θ_0}
Require: Hyperparameters: $\alpha, \beta, \eta, B, N_{\text{epochs}}$
Initialize LoRA adapters on π_{θ_0}
Freeze all parameters in π_{ref}
for epoch = 1 to N_{epochs} **do**
  for each batch $\mathcal{B}$ in $\mathcal{D}$ **do**
    // **Forward pass through policy model**
    $\{\mathcal{L}_{\text{CE}}(\mathbf{y}_i;\theta)\}_{i\in\mathcal{B}} \leftarrow \text{PolicyModel}(\mathcal{B})$
    // **Forward pass through reference model (no gradients)**
    $\{\log \pi_{\text{ref}}(\mathbf{y}_i|\mathbf{x}_i)\}_{i\in\mathcal{B}} \leftarrow \text{ReferenceModel}(\mathcal{B})$
    // **Compute KL divergence**
    $\mathcal{D}_{\text{KL}} \leftarrow \text{KL}(\pi_\theta \| \pi_{\text{ref}})$ {Equation 13}
    // **Separate positive and negative losses**
    $\mathcal{L}_{\text{pos}} \leftarrow \frac{1}{|\mathcal{B}^+|}\sum_{i\in\mathcal{B}^+} \mathcal{L}_{\text{CE}}(\mathbf{y}_i;\theta)$
    $\mathcal{L}_{\text{neg}} \leftarrow \frac{1}{|\mathcal{B}^-|}\sum_{i\in\mathcal{B}^-} \mathcal{L}_{\text{CE}}(\mathbf{y}_i;\theta)$
    // **Compute combined loss**
    $\mathcal{L} \leftarrow \alpha\mathcal{L}_{\text{pos}} + (1-\alpha)\mathcal{L}_{\text{neg}} + \beta\mathcal{D}_{\text{KL}}$
    // **Backward pass (only through LoRA parameters)**
    $\nabla_\theta \mathcal{L} \leftarrow \text{Backward}(\mathcal{L})$
    // **Update LoRA parameters**
    $\theta \leftarrow \theta - \eta \cdot \text{AdamW}(\nabla_\theta \mathcal{L})$
  end for
end for
return Fine-tuned model π_θ with LoRA adapters

8 Experiments and Results

The repository provides notebooks demonstrating PMPO training on GPT-2. While qualitative improvements are shown, the repository does not yet include:

8.1 PMPO with $\alpha = 0.5$ (Alpha_half)

Approach. We trained PMPO using the α-divergence with $\alpha = 0.5$, replacing the standard KL term while keeping all other hyperparameters fixed for comparability. This choice corresponds to a midpoint in the α-divergence family, balancing mass-covering and mode-seeking behaviour. The aim was to test whether this middle-ground divergence stabilizes learning while allowing effective adaptation.

Observations. This setting produced the best results among all variants, achieving a perplexity of 6.4769 and a loss of 1.8682. The strong performance indicates that $\alpha = 0.5$ provides an effective regularization strength: it constrains the policy enough to prevent drift while enabling flexible updates. In divergence terms, $\alpha = 0.5$

offers a balanced penalty that yields improved predictive fit and stability.

8.2 PMPO with KL Divergence (PMPO_KL)

Approach. We evaluated PMPO with the standard forward KL divergence, corresponding to the $\alpha \to 1$ limit of the α-divergence family. This configuration serves as a classical baseline in preference optimization, allowing us to compare the pure KL formulation with other α values under identical training conditions.

Observations. The KL-based PMPO run produced results nearly identical to the $\alpha = 0.5$ case, with a perplexity of 6.4774 and loss of 1.8683. This similarity suggests that both divergences impose comparable regularization strength for this dataset and hyperparameter setting. KL regularization kept the model close to the reference distribution while still allowing meaningful fine-tuning, leading to strong performance.

8.3 PMPO with $\alpha = 0$ (Alpha_zero)

Approach. We performed a PMPO run where the α-divergence was set to $\alpha = 0$. This choice shifts the divergence toward a more mass-covering regime within the α-divergence spectrum. The objective was to investigate how a more conservative, support-covering divergence influences adaptation under PMPO.

Observations. This configuration yielded a slightly higher perplexity of 6.5144 and a loss of 1.8740. The modest degradation aligns with the expected behaviour of mass-covering divergences, which tend to spread probability more broadly across the token space. This broader distribution appears to slightly reduce predictive sharpness on the evaluation set, explaining the increase in perplexity.

8.4 PMPO with $\alpha = 1$ (Alpha_one)

Approach. We explicitly trained a PMPO model with $\alpha = 1$, the formal forward-KL limit of the α-divergence family. Although close to the KL configuration, this run was included to examine whether the exact $\alpha = 1$ implementation exhibits different optimization dynamics or stability characteristics.

Observations. This model produced the weakest PMPO results, with a perplexity of 6.8736 and loss of 1.9277. The degradation suggests that the $\alpha = 1$ formulation encouraged sharper, more mode-seeking distributions, potentially leading to over-concentration and reduced generalization. Numerical sensitivity at the exact $\alpha = 1$ boundary may also have contributed to the performance drop.

8.5 Base Model (No PMPO Fine-Tuning)

Approach. We evaluated the original base model without any PMPO or LoRA-based fine-tuning. This benchmark serves as the fundamental reference for assessing the effectiveness of divergence-based preference optimization.

Observations. The base model performed significantly worse than all PMPO variants, with a perplexity of 10.984 and a loss of 2.396. This large gap demonstrates that PMPO fine-tuning, regardless of the chosen α value, substantially improves model fit and predictive quality. The results confirm the benefit of preference-driven training with divergence regularization.

8.6 Experiments on Different values of beta/KL coefficient

In our PMPO experiments, we focused on analyzing the effect of the regularization coefficient β, which controls the strength of the divergence penalty during policy updates. By adjusting β, we observed how strongly the policy was constrained to remain close to its previous distribution. Larger values of β produced more conservative updates with smoother learning curves, while smaller values allowed more aggressive policy shifts but introduced higher variance. Our results show that tuning β is crucial for achieving a stable balance between policy improvement and regularization within the PMPO framework.

Table 2: Perplexity Comparison Summary

Model	Perplexity	Loss
PMPO Alpha=0.5	6.476904	1.868243
PMPO KL	6.477354	1.868312
PMPO Alpha=0	6.514394	1.874014
PMPO Alpha=1	6.873585	1.927686
Base Model	10.983902	2.396431

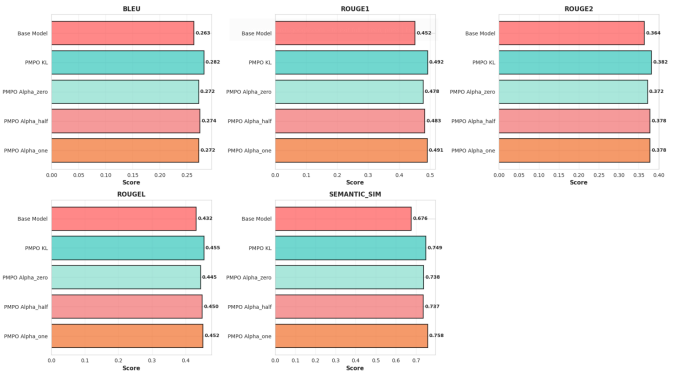

Figure 1: Comparisons

8.7 Deepmind Control Suite

Google's deepmind control suite offers many physics based simulations for reinforcement learning. Each environment provides observations about the environment, actions or physical values (force, torque, etc.) and rewards (feedback).

8.7.1 *Cheetah-Run.*

Environment. In this task, the agent has to control a cheetah-like robot with multiple joints and make it run as fast as possible without falling. The agent is also given joint positions, torque, velocity, torso height and other physical attributes, and the agent gets higher reward when velocity and stability is high.

Observations. The PMPO agent gets an Evaluation reward of 78.8 for this task, which shows that the agent has not yet learnt completely how to run but is still performing good. Low Critic, Actor loss and kl also show that the agent is not drifting from its base policy very rapidly and also fits the targets well

8.7.2 *Cartpole-Swingup.*

Environment. In this task, the agent has to control a cartpole moving on a track with a pole attatched via a hinge on it. The goal of the agent is to swing the pole such that it becomes straight/upright on the cart. Similar to the previous task, the observations are physical quantities like velocity and position, the actions are the forces applied on the cartpole, and the reward is based on how upright the pole is on the cart.

Observations. The PMPO agent scores an evaluation reward of 276.8 by the end of training, which is fairly decent for this task. The critic losses are a bit high (10^6, which show that there is a bit of instability in the agent's learning. The actor loss and KL show that the agent is learning resonably but the learning is unstabe (as shown by critic losses).

Table 3: Perplexity Comparison Summary

Task Actor Loss	Critic Loss 1 KL	Critic Loss 2 Eval Reward
Cheetah-run -0.02	0.004 0.08	0.004 78.8
Cartpole-Swingup -0.48	10^6 0.3	10^6 276.8

8.8 DPO

We evaluated the effect of DPO-LoRA on a GPT-2 model by measuring perplexity (PPL) on a test corpus. The base GPT-2 model achieved a PPL of 21.589. After fine-tuning with DPO-LoRA, the model's PPL increased to 23.182. This modest increase in perplexity is an expected trade-off. It suggests that while DPO is steering the model's outputs towards a preferred distribution (alignment), this comes at the minor cost of its raw predictive fluency on the general text data. The primary goal of DPO is to enhance alignment, not necessarily to optimize for the lowest possible perplexity.

8.9 KTO

We evaluated the model's generative fluency after applying Kahneman-Tversky Optimization (KTO) by measuring perplexity (PPL) on a held-out test set. The aligned model achieved a PPL of 17.386. This strong result indicates that the KTO alignment, which primarily optimizes for desired behavioral outcomes based on preference data, was achieved without degrading the model's core language modeling capabilities. In fact, this low perplexity score suggests a high degree of linguistic coherence, demonstrating that KTO can successfully align a model while maintaining or even enhancing its generative fluency.

8.10 LLM as a Judge

To automate the assessment of response quality, we employed an "LLM as a Judge" methodology. For this, a powerful, pre-trained model was configured as a pointwise evaluator. We designed a specialized prompt instructing the judge model to assess each generated response against a set of predefined criteria (e.g., helpfulness, accuracy, and safety). The judge's task was to provide a binary score: 1 for a "positive" response that met the criteria, and 0 for a "negative" response that failed to do so. This approach allows for scalable, automated evaluation of model alignment and performance on qualitative tasks.

9 Discussion

PMPO offers a promising alternative to classical RLHF pipelines. The repository provides a good starting point for researchers interested in preference optimization. However, for production-level research, improvements in documentation, evaluation, scalability, and experiment reproducibility will be required.

The structure of the trainer allows extensibility—for example, integrating richer datasets, scaling to larger models, or experimenting with alternative normalization strategies.

10 Conclusion

The DSL501-PROJECT provides a clear and modular implementation of Preference-Based Maximum a Posteriori Optimization. Its focus on positive and negative preference signals combined with KL regularization offers a valuable baseline for preference-based alignment research. With further enhancements in documentation, evaluation, and experimentation, this repository can evolve into a strong academic or industrial reference.

11 Future Work

Potential extensions include:

- Scaling experiments to larger LLMs.
- Integrating automatic win-rate evaluators.
- Adding ablation studies on margin and KL parameters.
- Benchmarking against DPO, PPO, KTO, and other methods.

12 References

- Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114. https://arxiv.org/pdf/1312.6114
- Son, K., Lee, J., Park, S. and Lee, S., 2007, August. Reinforcement Learning Using Negative Relevance Feedback. In Sixth International Conference on Advanced Language Processing

and Web Information Technology (ALPIT 2007) (pp. 559-563). IEEE. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4460701

- Rafailov, R., Sharma, A., Mitchell, E., Manning, C.D., Ermon, S. and Finn, C., 2023. Direct preference optimization: Your language model is secretly a reward model. Advances in neural information processing systems, 36, pp.53728-53741. https://arxiv.org/pdf/2305.18290
- Abdolmaleki, A., Springenberg, J.T., Tassa, Y., Munos, R., Heess, N. and Riedmiller, M., 2018. Maximum a posteriori policy optimisation. arXiv preprint arXiv:1806.06920. https://arxiv.org/abs/1806.06920
- Ethayarajh, K., Xu, W., Muennighoff, N., Jurafsky, D. and Kiela, D., 2024. KTO: Model Alignment as Prospect Theoretic Optimization. arXiv preprint arXiv:2402.01306. https://arxiv.org/abs/2402.01306