

Solve this problem step by step; give explanations and reasoning on each step

Below is a **clean, step-by-step derivation** for every sub-question (a)–(h) of the Gaussian toy model for DDPM/DDIM.

I walk you through each step carefully and explain the reasoning intuitively.

This will give you a *complete conceptual understanding* of how diffusion models work in this simplified 1-D linear-Gaussian setting.

Setting

- Forward diffusion starts from

$$X_0 \sim \mathcal{N}(0, \sigma^2), \quad \sigma^2 \ll 1.$$

- Time runs from $t = 0$ to $t = 1$.
- We take small steps of size $\Delta t \ll 1$.
- Each forward step:



$$X_{t+\Delta t} = X_t + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \Delta t).$$

Thus the forward process is simply **adding Gaussian noise gradually**.

(a) Distribution of X_1

Each forward step adds variance Δt .

If $T = 1/\Delta t$ steps, the total added variance is

$$\text{Var}(X_1) = \text{Var}(X_0) + T(\Delta t) = \sigma^2 + 1.$$

Because adding independent Gaussians preserves Gaussianity:

$$X_1 \sim \mathcal{N}(0, \sigma^2 + 1).$$

Answer (a):

$$X_1 \sim \mathcal{N}(0, \sigma^2 + 1).$$

(b) Marginal distributions of $X_{t-\Delta t}$ and X_t

The variance grows linearly with time:

$$X_t \sim \mathcal{N}(0, \sigma^2 + t).$$



Therefore:

Therefore:

- $X_{t-\Delta t} \sim \mathcal{N}(0, \sigma^2 + t - \Delta t)$
- $X_t \sim \mathcal{N}(0, \sigma^2 + t)$

Answer (b):

$$X_{t-\Delta t} \sim \mathcal{N}(0, \sigma^2 + t - \Delta t), \quad X_t \sim \mathcal{N}(0, \sigma^2 + t).$$

(c) Conditional distribution $X_{t-\Delta t} | X_t = x_t$

For jointly Gaussian variables:

$$X_{t-\Delta t} | X_t = x_t \sim \mathcal{N}\left(\frac{\sigma^2 + t - \Delta t}{\sigma^2 + t} x_t, \frac{(\sigma^2 + t - \Delta t)\Delta t}{\sigma^2 + t}\right).$$

We must simplify the variance assuming:

- $\Delta t \ll t$
- $\Delta t \ll \sigma^2 \ll 1$

Variance:

$$\frac{(\sigma^2 + t - \Delta t)\Delta t}{\sigma^2 + t} = \Delta t \left(1 - \frac{\Delta t}{\sigma^2 + t}\right) \approx \Delta t.$$

Since $\Delta t / (\sigma^2 + t) \ll 1$.

Answer (c):



$$\text{Var}(X_{t-\Delta t} | X_t) \approx \Delta t.$$

(d) Reverse process without adding noise

We repeatedly apply the conditional mean:

$$\widehat{X}_{t-\Delta t} = \frac{\sigma^2 + t - \Delta t}{\sigma^2 + t} X_t.$$

Write the multiplicative factor:

$$\alpha_t = \frac{\sigma^2 + t - \Delta t}{\sigma^2 + t} = 1 - \frac{\Delta t}{\sigma^2 + t}.$$

Running this from $t = 1$ down to $t = 0$:

$$\widehat{X}_0 = X_1 \prod_{k=0}^{T-1} \left(1 - \frac{\Delta t}{\sigma^2 + 1 - k\Delta t}\right).$$

This product telescopes:

$$\prod_{k=0}^{T-1} \frac{\sigma^2 + 1 - (k+1)\Delta t}{\sigma^2 + 1 - k\Delta t} = \frac{\sigma^2}{\sigma^2 + 1}.$$

Thus:

$$\widehat{X}_0 = X_1 \cdot \frac{\sigma^2}{\sigma^2 + 1}.$$

Since $X_1 \sim \mathcal{N}(0, \sigma^2 + 1)$,

$$\widehat{X}_0 \sim \mathcal{N}\left(0, (\sigma^2 + 1) \left(\frac{\sigma^2}{\sigma^2 + 1}\right)^2\right) = \mathcal{N}\left(0, \frac{\sigma^4}{\sigma^2 + 1}\right).$$

Because $\sigma^2 \ll 1$:

$$\text{Var}(\widehat{X}_0) \approx \sigma^4.$$

This is **much too small** compared to the desired σ^2 .

This shows why *adding noise in the reverse process is essential*.

Answer (d):

$$\boxed{\widehat{X}_0 \sim \mathcal{N}\left(0, \frac{\sigma^4}{\sigma^2 + 1}\right) \approx \mathcal{N}(0, \sigma^4)}$$

too little variance.

(e) Adding noise back (DDPM reverse)

The expression given is:



$$\text{Var}(\widehat{X}_0) = \left(\frac{\sigma^2}{\sigma^2 + 1}\right)^2 + \Delta t \sum_{k=0}^{T-1} \left(\frac{\sigma^2}{\sigma^2 + k\Delta t}\right)^2.$$

Take the limit $\Delta t \rightarrow 0$:

$$\Delta t \sum_{k=0}^{T-1} \left(\frac{\sigma^2}{\sigma^2 + k\Delta t}\right)^2 \rightarrow \int_0^1 \frac{\sigma^4}{(\sigma^2 + t)^2} dt.$$

Compute:

$$\int_0^1 \frac{\sigma^4}{(\sigma^2 + t)^2} dt = \sigma^4 \left[\frac{-1}{\sigma^2 + t} \right]_0^1 = \sigma^4 \left(\frac{1}{\sigma^2} - \frac{1}{\sigma^2 + 1} \right) = \sigma^2 - \frac{\sigma^4}{\sigma^2 + 1}.$$

Now add the deterministic term:

$$\left(\frac{\sigma^2}{\sigma^2 + 1}\right)^2 + \sigma^2 - \frac{\sigma^4}{\sigma^2 + 1} = \sigma^2.$$

Answer (e):

$$\boxed{\text{Var}(\widehat{X}_0) = \sigma^2.}$$

Thus DDPM noise-corrected reverse diffusion recovers the true data variance.

(f) DDPM & DDIM update equation

DDPM conditional mean:

$$\widehat{X}_0^* = \frac{\sigma^2}{\sigma^2 + t} X_t,$$

so:

$$\widehat{X}_{t-\Delta t}^{\text{DDPM}} = X_t - \frac{\Delta t}{\sigma^2 + t} (X_t - \widehat{X}_0^*) + \text{noise}.$$

DDIM removes noise and instead uses:

$$x_{t-\Delta t}^{\text{DDIM}} = x_t + \eta(t, \Delta t) (\widehat{X}_0^* - x_t),$$

where:

$$\eta(t, \Delta t) = \frac{\sqrt{t}}{\sqrt{t} - \Delta t + \sqrt{t}} \approx \frac{\Delta t}{t} \sqrt{t}.$$

Training the neural network

We want the network to learn:

$$g(x_t, t) = \frac{\sigma^2}{\sigma^2 + t} x_t.$$

Inputs to the network:

$$(x_t, t).$$

Generate a batch:



1. Draw $x_0 \sim p_{\text{data}}$.
2. Sample $t \sim \text{Uniform}(0, 1)$.
3. Sample noise: $\varepsilon \sim \mathcal{N}(0, 1)$.
4. Compute

$$x_t = x_0 + \sqrt{t} \varepsilon.$$

5. Use the pair (x_t, t) as input.

Loss function:

The correct target is

$$g^{\backslash *}(x_t, t) = \frac{\sigma^2}{\sigma^2 + t} x_t.$$

Use MSE:

$$L = \mathbb{E}[(g_\theta(x_t, t) - g^{\backslash *}(x_t, t))^2].$$

(g) Approximate DDIM step for $\Delta t \ll t$

Given:

$$g(t) = \frac{\sigma^2}{\sigma^2 + t},$$

DDIM update:

$$x_{t-\Delta t} \approx x_t + \eta(t, \Delta t) (g(t)x_t - x_t).$$

For $\Delta t \ll t$,

$$\eta(t, \Delta t) \approx \frac{\Delta t}{2t}.$$

Then:

$$x_{t-\Delta t} \approx x_t + \frac{\Delta t}{2t} \left(\frac{\sigma^2}{\sigma^2 + t} - 1 \right) x_t.$$

Since

$$\frac{\sigma^2}{\sigma^2 + t} - 1 = -\frac{t}{\sigma^2 + t},$$

we get:

$$x_{t-\Delta t} \approx x_t \left(1 - \frac{\Delta t}{2(\sigma^2 + t)} \right).$$

(h) Distribution of \widehat{X}_0 after all DDIM steps

We start with $X_1 \sim \mathcal{N}(0, 1 + \sigma^2)$.

Each step multiplies the sample by:

$$1 - \frac{\Delta t}{2(\sigma^2 + t)}.$$

Thus the full product from $t = 1 \rightarrow 0$:

$$\widehat{X}_0 = X_1 \prod_{k=0}^{T-1} \left(1 - \frac{\Delta t}{2(\sigma^2 + k\Delta t)} \right).$$

Take logs:

$$\log \widehat{X}_0 = \log X_1 + \sum_k \log \left(1 - \frac{\Delta t}{2(\sigma^2 + k\Delta t)} \right).$$

Use $\log(1 - x) \approx -x$:

$$\sum_k -\frac{\Delta t}{2(\sigma^2 + k\Delta t)} \rightarrow - \int_0^1 \frac{dt}{2(\sigma^2 + t)}.$$

Compute:

$$-\frac{1}{2} \log(\sigma^2 + t) \Big|_0^1 = -\frac{1}{2} \log \left(\frac{\sigma^2 + 1}{\sigma^2} \right).$$

Thus:

$$\widehat{X}_0 = X_1 \sqrt{\frac{\sigma^2}{\sigma^2 + 1}}.$$

Hence its variance:

$$\text{Var}(\widehat{X}_0) = (\sigma^2 + 1) \frac{\sigma^2}{\sigma^2 + 1} = \sigma^2.$$

Answer (h):

$$\widehat{X}_0 \sim \mathcal{N}(0, \sigma^2).$$

DDIM also recovers the original variance (although deterministically).

Summary of Key Results

Part	Result	
(a)	$X_1 \sim \mathcal{N}(0, \sigma^2 + 1)$	
(b)	$X_t \sim \mathcal{N}(0, \sigma^2 + t)$	
(c)	$\text{Var}(X_t) = \sigma^2 + t$	
(d)	Naive reverse gives too small variance: $\approx \sigma^4$	
(e)	Adding noise gives correct variance σ^2	
(f)	Learned model estimates $g(x_t, t) = \frac{x_t}{\sigma^2 + t}$	
(g)	DDIM step: $x_{t-\Delta t} \approx \left(1 - \frac{\Delta t}{2(\sigma^2 + t)} \right) x_t$	
(h)	DDIM reconstruction: $\widehat{X}_0 \sim \mathcal{N}(0, \sigma^2)$	

Parts (a)-(c): Solid probabilistic foundations

The opening segments were handled competently. For part (a), GPT immediately recognized the structure: starting from $X_0 \sim \mathcal{N}(0, \sigma^2)$ and adding independent Gaussian noise with variance

Δt at each of $T = 1/\Delta t$ steps yields $X_1 \sim N(0, \sigma^2 + 1)$ through the additive property of Gaussian variances. The explanation was direct and correctly leveraged the reproductive property of normal distributions. Part (b) demonstrated similar competence with marginal distributions in the forward process, correctly identifying $X_t \sim N(0, \sigma^2 + t)$. The variance approximation in part (c) properly simplified $(\sigma^2 + t - \Delta t)\Delta t/(\sigma^2 + t) \approx \Delta t$ by recognizing that $\sigma^2 + t - \Delta t \approx \sigma^2 + t$ when $\Delta t \ll \sigma^2$.

The approach in these sections was methodical and transparent, with each step justified by standard probability results. This suggests GPT has internalized common probability manipulations involving Gaussians and can apply them reliably in straightforward contexts.

Part (d): Successful telescoping product analysis

The question of what happens under deterministic reverse diffusion (applying only conditional means without stochastic noise) requires recognizing a telescoping structure. GPT correctly formulated this as a product of T ratios: $\hat{X}_0 = X_1 \cdot \prod (\sigma^2/(\sigma^2 + k\Delta t))$, then observed the telescoping cancellation leading to $\hat{X}_0 = (\sigma^2/(\sigma^2 + 1)) \cdot X_1$. The conclusion that variance gets compressed to $\sigma^4/(\sigma^2 + 1)^2$ follows immediately and was correctly stated. The hints were used appropriately to structure the solution, suggesting GPT can recognize and exploit problem scaffolding when provided.

Part (e): Incomplete integral evaluation

Here the first significant gap appeared. The transition from discrete sum to continuous integral was set up correctly: $\text{Var}(\hat{X}_0) = \sigma^4 + \int_0^1 \sigma^4/(\sigma^2 + t)^2 dt$. However, GPT then simply asserted this evaluates to σ^2 without demonstration. The provided hint explicitly gives $\int 1/(1+t)^2 dt = C - 1/(1+t)$, which with appropriate u-substitution should yield a verifiable result. The calculation would involve recognizing $\int_0^1 1/(\sigma^2 + t)^2 dt = (1/\sigma^2) \int_0^1 1/(1 + t/\sigma^2)^2 d(t/\sigma^2)$, evaluating to $(1/\sigma^2)[\sigma^2/(\sigma^2) - \sigma^2/(\sigma^2+1)] = 1/\sigma^2 - 1/(\sigma^2+1)$. Combined with the σ^4 term, this should produce the desired σ^2 . GPT skipped this entire algebraic chain, suggesting either inability to complete the calculation or a tendency to shortcut when the answer is known from context (the hint states the result should equal σ^2).

Parts (f)-(h): Increasing abstraction and gaps

The description of neural network training for the denoising function in part (f) remained at a high conceptual level. GPT outlined sampling from the dataset, adding noise at various timesteps, and training a network to predict clean signals, but lacked specificity about the actual implementation. The question asks explicitly about input generation, batch construction, and loss computation. A complete answer would specify: sample x from the dataset, sample t uniformly from $[0, 1]$, compute $xt = \sqrt{(\sigma^2 + t)} \cdot x + \sqrt{t} \cdot \epsilon$ where $\epsilon \sim N(0, 1)$, feed (xt, t) into the network $g(xt, t)$, and minimize $\|g(xt, t) - \sigma^2/(\sigma^2 + t) \cdot xt\|^2$. GPT provided the conceptual framework without these implementation details.

The DDIM step approximation in part (g) correctly identified that $\eta(t, \Delta t) \approx 1/2$ for small Δt by expanding $\sqrt{t}/(\sqrt{(t-\Delta t)} + \sqrt{t}) \approx \sqrt{t}/(2\sqrt{t} - \Delta t/(2\sqrt{t})) \approx 1/2$. This was handled adequately. However, part

(h) requesting the full DDIM trajectory calculation showed clear signs of incomplete work. GPT claimed the product of all DDIM steps yields \hat{X}_0 proportional to $X_0 \cdot \exp(-1/2 \cdot \int_0^1 dt / (\sigma^2 + t))$ without showing the intermediate steps converting the product to a sum via logarithms. The hint about $\ln(1-x) \approx -x$ for small x was mentioned but not applied. A rigorous solution would write out the product explicitly, take logarithms to convert to a sum, recognize the sum as a Riemann approximation to an integral, evaluate that integral, and exponentiate back. GPT sketched this outline but didn't execute it.

The pattern across these later parts suggests GPT maintains conceptual understanding of the overall strategy but loses precision when algebraic manipulations become lengthy or when multiple approximations must be tracked simultaneously.

ChatGPT 5.1 Thinking ▾

Share Add people

If you'd like, I can also:

- Draw diagrams of the forward and backward processes
- Compare DDPM and DDIM intuitively
- Show the telescoping products explicitly
- Build a computational notebook version for verification

Would you like that?

< 2/2 > ...

Reward Model?

in Learning from Human Feedback (RLHF) pipeline
cross-entropy loss (CE) is used to train the reward model. Specifically, we use the SFT model to generate pairs of an input x and a target y , where y_p denotes the preferred (winning) output an y_a respectively. Crucially, we assume these prefix models are able to correctly predict the probability $p(y_p | x)$ of y_p to approximate y . However, we face a challenge due to discrete y_p vs y_a , using a reward loss

RLHF Pipeline

is used to train the reward model. Specifically, we use the SFT model to generate pairs of an input x and a target y , where y_p denotes the preferred (winning) output an y_a respectively. Crucially, we assume these prefix models are able to correctly predict the probability $p(y_p | x)$ of y_p to approximate y . However, we face a challenge due to discrete y_p vs y_a , using a reward loss

cross-entropy loss (CE) is used to train the reward model. Specifically, we use the SFT model to generate pairs of an input x and a target y , where y_p denotes the preferred (winning) output an y_a respectively. Crucially, we assume these prefix models are able to correctly predict the probability $p(y_p | x)$ of y_p to approximate y . However, we face a challenge due to discrete y_p vs y_a , using a reward loss

cross-entropy loss (CE) is used to train the reward model. Specifically, we use the SFT model to generate pairs of an input x and a target y , where y_p denotes the preferred (winning) output an y_a respectively. Crucially, we assume these prefix models are able to correctly predict the probability $p(y_p | x)$ of y_p to approximate y . However, we face a challenge due to discrete y_p vs y_a , using a reward loss

Now do the same for this one



Thought for 3m 20s >

Alright, let's walk through the DPO / reward-model problem step by step, part by part.

Notation recap

- We have a reference policy $\pi_{\text{ref}}(y | x)$ (the SFT model).
- We want a new policy $\pi_\theta(y | x)$ that both:
 - gets high reward $r_\phi(x, y)$, and
 - doesn't drift too far from π_{ref} .
- $D_{\text{KL}}(p || q) = \mathbb{E}_{x \sim p(x)} [\log \frac{p(x)}{q(x)}]$.

DPO will ultimately avoid having an explicit reward model and use **log-ratios of policies** instead.**(a) Warm-up: $\min_{p \in \mathcal{P}} D_{\text{KL}}(p || q)$** We want the distribution p^{*} that minimizes

$$D_{\text{KL}}(p || q) = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$

Recall the basic property of KL divergence:

- $D_{\text{KL}}(p || q) \geq 0$ for all p ,
- and $D_{\text{KL}}(p || q) = 0$ iff $p = q$ (pointwise).

So the minimum possible value is 0, and it is achieved uniquely at

$$p^{*}(x) = q(x) \text{ for all } x.$$

That's the whole story for (a).

(b) Optimal policy for RL objectiveWe consider the objective (for a fixed prompt x):

$$\max_{\pi_\theta(\cdot | x)} \underbrace{\mathbb{E}_{y \sim \pi_\theta(\cdot | x)} [r_\phi(x, y)]}_{\text{expected reward}} - \beta D_{\text{KL}}(\pi_\theta(\cdot | x) || \pi_{\text{ref}}(\cdot | x)).$$

For a fixed x , write $p(y) = \pi_\theta(y | x)$ and $q(y) = \pi_{\text{ref}}(y | x)$. Then:

$$J(p) = \sum_y p(y) r_\phi(x, y) - \beta \sum_y p(y) \log \frac{p(y)}{q(y)}.$$

We maximize $J(p)$ subject to $\sum_y p(y) = 1$ and $p(y) \geq 0$.Use a Lagrangian with multiplier λ :

$$\mathcal{L}(p, \lambda) = \sum_y p(y) r_\phi(x, y) - \beta \sum_y p(y) \log \frac{p(y)}{q(y)} + \lambda \left(\sum_y p(y) - 1 \right).$$

Take derivative w.r.t. each $p(y)$ and set to zero:

Take derivative w.r.t. each $p(y)$ and set to zero:

$$\frac{\partial \mathcal{L}}{\partial p(y)} : r_\phi(x, y) - \beta \left(\log \frac{p(y)}{q(y)} + 1 \right) + \lambda = 0.$$

Rearrange:

$$\log \frac{p(y)}{q(y)} = \frac{1}{\beta} (r_\phi(x, y) + \lambda - \beta) = \frac{1}{\beta} r_\phi(x, y) + C$$

for some constant C (independent of y), since λ is shared.

Exponentiate:

$$p(y) = q(y) \exp \left(\frac{1}{\beta} r_\phi(x, y) \right) \exp(C) \propto q(y) \exp \left(\frac{1}{\beta} r_\phi(x, y) \right).$$

Normalize via a partition function

$$Z(x) = \sum_y q(y) \exp \left(\frac{1}{\beta} r_\phi(x, y) \right),$$

to get the optimal policy:

$$\pi_\theta^*(y | x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y | x) \exp \left(\frac{1}{\beta} r_\phi(x, y) \right)$$

with

$$Z(x) = \sum \pi_{\text{ref}}(y | x) \exp \left(\frac{1}{\beta} r_\phi(x, y) \right).$$

(Exactly the form stated in the problem.)

(c) Why this form is hard to use directly

The expression

$$\pi_\theta^*(y | x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y | x) \exp \left(\frac{1}{\beta} r_\phi(x, y) \right)$$

requires the **partition function**

$$Z(x) = \sum_{y \in \mathcal{Y}} \pi_{\text{ref}}(y | x) e^{r_\phi(x, y) / \beta}.$$

But for language models:

- The output space \mathcal{Y} is *astronomically large* (all possible token sequences).
- Summing over all y is intractable.

So although this formula is theoretically clean, we **cannot compute $Z(x)$** explicitly or normalize over \mathcal{Y} in practice. That's why we look for a different way that avoids ever needing $Z(x)$.

(d) Express $r_\phi(x, y)$ in terms of π_θ^* , π_{ref} , $Z(x)$

Start from the optimal policy equation and solve for r_ϕ .

From (b):

$$\pi_\theta^*(y | x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y | x) \exp\left(\frac{1}{\beta} r_\phi(x, y)\right).$$

Rearrange:

$$\exp\left(\frac{1}{\beta} r_\phi(x, y)\right) = \frac{Z(x) \pi_\theta^*(y | x)}{\pi_{\text{ref}}(y | x)}.$$

Take logs and multiply by β :

$$r_\phi(x, y) = \beta \log \frac{\pi_\theta^*(y | x)}{\pi_{\text{ref}}(y | x)} + \beta \log Z(x).$$

This is the **implicit reward parametrization**: reward is a scaled log-ratio of the optimal policy and the reference policy, plus an x -dependent constant $\beta \log Z(x)$.

(e) Plug into Bradley–Terry and show $Z(x)$ cancels

Bradley–Terry (BT) for a pair (y_w, y_l) (winner vs loser) is:

$$p^*(y_w \succ y_l | x) = \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)),$$

or equivalently

$$= \frac{\exp(r_\phi(x, y_w))}{\exp(r_\phi(x, y_w)) \downarrow + \exp(r_\phi(x, y_l))}.$$

Using (d):

$$r_\phi(x, y_w) - r_\phi(x, y_l) = \beta \left[\log \frac{\pi_\theta^*(y_w | x)}{\pi_{\text{ref}}(y_w | x)} - \log \frac{\pi_\theta^*(y_l | x)}{\pi_{\text{ref}}(y_l | x)} \right] + \beta \log Z(x) - \beta \log Z(x).$$

The $+\beta \log Z(x)$ terms **cancel exactly**, so

$$r_\phi(x, y_w) - r_\phi(x, y_l) = \beta \left[\log \frac{\pi_\theta^*(y_w | x)}{\pi_{\text{ref}}(y_w | x)} - \log \frac{\pi_\theta^*(y_l | x)}{\pi_{\text{ref}}(y_l | x)} \right].$$

Thus:

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

Because $Z(x)$ disappears, we **never need to compute the partition function**. That's desirable because:

- It avoids the intractable sum over all y .
- Our model depends only on **log-ratios** of policies, which we can compute.

Using this probability and maximum likelihood over the dataset of preferences, we get the DPO loss:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w | x)}{\pi_{\text{ref}}(y_w | x)} - \beta \log \frac{\pi_\theta(y_l | x)}{\pi_{\text{ref}}(y_l | x)} \right) \right].$$

This is exactly the expression shown in the statement.

(f) Gradient of DPO loss and interpretation

Define the **implicit reward**

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

For one data triple (x, y_w, y_l) , the loss term is

$$\ell(\theta) = -\log \sigma(\hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l)).$$

Let

$$A = \hat{r}_\theta(x, y_w), \quad B = \hat{r}_\theta(x, y_l), \quad \Delta = A - B.$$

Then

$$\ell(\theta) = -\log \sigma(\Delta).$$

Step 1 – derivative wrt Δ :

$$\frac{\partial \ell}{\partial \Delta} = -\frac{1}{\sigma(\Delta)} \sigma'(\Delta) = -\frac{1}{\sigma(\Delta)} \sigma(\Delta)(1 - \sigma(\Delta)) = -(1 - \sigma(\Delta)) = \sigma(\Delta) - 1.$$

Using $1 - \sigma(\Delta) = \sigma(-\Delta) = \sigma(B - A)$, we can also write

$$\frac{\partial \ell}{\partial \Delta} = -\sigma(B - A).$$

Step 2 – chain rule to θ :

$$\nabla_\theta \ell = \frac{\partial \ell}{\partial \Delta} \cdot \nabla_\theta \Delta = (\sigma(\Delta) - 1)(\nabla_\theta A - \nabla_\theta B).$$

But



$$A = \beta \log \pi_\theta(y_w | x) - \beta \log \pi_{\text{ref}}(y_w | x), \quad \Rightarrow \nabla_\theta A = \beta \nabla_\theta \log \pi_\theta(y_w | x),$$

and similarly

$$\nabla_\theta B = \beta \nabla_\theta \log \pi_\theta(y_l | x).$$

So

$$\nabla_\theta \ell = \beta(\sigma(\Delta) - 1)(\nabla_\theta \log \pi_\theta(y_w | x) - \nabla_\theta \log \pi_\theta(y_l | x)).$$

Using $\sigma(\Delta) - 1 = -\sigma(B - A)$, a common form is

$$\boxed{\nabla_\theta \ell = -\beta \sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w)) (\nabla_\theta \log \pi_\theta(y_w | x) - \nabla_\theta \log \pi_\theta(y_l | x)).}$$

Averaging over the dataset gives $\nabla_\theta \mathcal{L}_{\text{DPO}}$.

Interpretation of the weighting term

The weight is

$$\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w)) = \sigma(B - A).$$

- If the model already strongly prefers the **winner** y_w over the loser y_l ($A \gg B$, so $B - A$ is very negative):
 $\sigma(B - A) \approx 0$.
→ **Small gradient**: we don't change the model much; it's already aligned.
- If the model is unsure or even prefers the loser
 $(B \gtrsim A$, so $B - A$ is large or near 0):

$\sigma(B - A)$ is close to 0.5 or near 1.

→ **Large gradient:** we update strongly, increasing

$\log \pi_\theta(y_w | x)$ and decreasing

$\log \pi_\theta(y_t | x)$.

So DPO focuses learning on examples where the model disagrees with human preferences.

(g) Plackett–Luce (PL) model for full rankings

Now labelers can provide a ranking over K answers

$y_1, \dots, y_K \sim \pi^{\text{SFT}}(\cdot | x)$.

Let τ be a permutation encoding the ranking:

$y_{\tau(1)}$ is best, $y_{\tau(2)}$ next, etc.

The Plackett–Luce probability of τ under reward r_ϕ is:

$$p_\theta^{*}(\tau | y_1, \dots, y_K, x) = \prod_{k=1}^K \frac{\exp(r_\phi(x, y_{\tau(k)}))}{\sum_{j=k}^K \exp(r_\phi(x, y_{\tau(j)}))}.$$

Use the implicit reward parametrization from (d):

$$r_\phi(x, y) = \beta \log \frac{\pi_\theta^{*}(y | x)}{\pi_{\text{ref}}(y | x)} + \beta \log Z(x).$$

Then



$$\exp(r_\phi(x, y)) = (Z(x))^\beta \left(\frac{\pi_\theta^{*}(y | x)}{\pi_{\text{ref}}(y | x)} \right)^\beta.$$

Plug into the PL probability:

$$\begin{aligned} p_\theta^{*}(\tau | y_1, \dots, y_K, x) &= \prod_{k=1}^K \frac{(Z(x))^\beta \left(\frac{\pi_\theta^{*}(y_{\tau(k)} | x)}{\pi_{\text{ref}}(y_{\tau(k)} | x)} \right)^\beta}{\sum_{j=k}^K (Z(x))^\beta \left(\frac{\pi_\theta^{*}(y_{\tau(j)} | x)}{\pi_{\text{ref}}(y_{\tau(j)} | x)} \right)^\beta} \\ &= \prod_{k=1}^K \frac{\left(\frac{\pi_\theta^{*}(y_{\tau(k)} | x)}{\pi_{\text{ref}}(y_{\tau(k)} | x)} \right)^\beta}{\sum_{j=k}^K \left(\frac{\pi_\theta^{*}(y_{\tau(j)} | x)}{\pi_{\text{ref}}(y_{\tau(j)} | x)} \right)^\beta}, \end{aligned}$$

since each numerator and denominator term has the same factor $(Z(x))^\beta$, which cancels.

So the final Plackett–Luce probability depends **only** on the log-ratios of π_θ^{*} vs π_{ref} :

$$p_\theta^{*}(\tau | y_1, \dots, y_K, x) = \prod_{k=1}^K \frac{\left(\frac{\pi_\theta^{*}(y_{\tau(k)} | x)}{\pi_{\text{ref}}(y_{\tau(k)} | x)} \right)^\beta}{\sum_{j=k}^K \left(\frac{\pi_\theta^{*}(y_{\tau(j)} | x)}{\pi_{\text{ref}}(y_{\tau(j)} | x)} \right)^\beta}.$$



Again, $Z(x)$ **cancels**, so we never have to compute the partition function.

Using this probability with maximum likelihood over a dataset of rankings

$$\mathcal{D} = \{(\tau^{(i)}, x^{(i)}, \{y_j^{(i)}\}_{j=1}^K)\}_{i=1}^N,$$

the problem statement notes that one can derive a multi-way DPO-style loss:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(\tau, y_1, \dots, y_K, x) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_{\tau(k)} | x)}{\pi_{\text{ref}}(y_{\tau(k)} | x)} - \beta \log \frac{\pi_\theta(y_{\tau(j)} | x)}{\pi_{\text{ref}}(y_{\tau(j)} | x)} \right) \right],$$

for appropriate choices of pairs (k, j) from the ranking.

If you want, we can next:

- Work through **one concrete numerical example** of a preference pair and compute the DPO gradient by hand, or
- Derive the same optimal policy form using the KL-minimization trick from (a) more explicitly (writing the objective as $-\beta D_{\text{KL}}(p || q) + \text{const}$).

□ ⌂ ↻ ⌃ ⌄ ...



Part (a): Trivial warm-up handled correctly

The unconstrained KL minimization problem was answered correctly with $p^* = q$. This follows immediately from the non-negativity of KL divergence and the fact that $D_{\text{KL}}(p || q) = 0$ if and only if $p = q$ almost everywhere. No issues observed.

Part (b): Critical derivation gap

GPT stated this result but provided no actual derivation. The response mentioned "using calculus of variations" without showing the variation, mentioned "taking derivatives" without writing them down, and essentially asserted the answer. This represents a fundamental failure because the entire point of the exercise is demonstrating mastery of constrained optimization techniques. The gap is particularly problematic because this derivation forms the foundation for all subsequent parts. Someone relying on this solution would miss understanding how constrained policy optimization actually works.

Part (c): Solid conceptual explanation

The question about why the optimal policy form is impractical was answered well. GPT correctly identified that the partition function $Z(x) = \sum y \pi_{\text{ref}}(y|x) \cdot \exp(r(x,y)/\beta)$ requires summing over all possible outputs y , which is intractable for large discrete spaces like text generation where y represents arbitrary-length sequences. The explanation clearly connected mathematical structure to computational reality.

Part (d): Algebraic error with propagation potential

Rearranging the optimal policy equation to isolate the reward requires taking logarithms of both sides: $\log \pi^*(y|x) = \log \pi_{\text{ref}}(y|x) + r(x,y)/\beta - \log Z(x)$, then multiplying by β to get $r(x,y) = \beta \cdot \log(\pi^*/\pi_{\text{ref}}) + \beta \cdot \log Z(x)$. GPT's response contained inconsistency here - in some formulations the β coefficient on $\log Z(x)$ was present, in others it was omitted. This type of coefficient error is insidious because it propagates through subsequent calculations. While the error might seem minor, it reflects imprecise algebraic tracking that would accumulate in longer derivations.

Part (e): Cancellation demonstrated correctly

Substituting the reparameterized reward into the Bradley-Terry model and showing $Z(x)$ cancels was handled properly. The probability of preferring yw over yl becomes $\sigma(\beta \cdot \log(\pi^*(yw|x)/\pi_{ref}(yw|x)) + \beta \cdot \log Z(x) - \beta \cdot \log(\pi^*(yl|x)/\pi_{ref}(yl|x)) - \beta \cdot \log Z(x))$, where the $\log Z(x)$ terms cancel immediately. GPT correctly identified this cancellation and explained its significance: the intractable partition function disappears from the optimization objective, making the problem tractable. This represents a key conceptual insight of DPO and was communicated clearly.

Part (f): Gradient derivation lacking implementation detail

Computing ∇_θ LDPO requires applying the chain rule through the sigmoid function and the log probability ratios. The gradient should be proportional to $-E[\sigma(\hat{f}_\theta(x, yl) - \hat{f}_\theta(x, yw)) \cdot (\nabla_\theta \log \pi_\theta(yw|x) - \nabla_\theta \log \pi_\theta(yl|x))]$, where the sigmoid term acts as an importance weight. GPT captured the structure and correctly interpreted the sigmoid weighting: when the model incorrectly assigns higher probability to the losing response (making $\hat{f}_\theta(x, yl) > \hat{f}_\theta(x, yw)$), the sigmoid approaches 1 and the gradient update is strong; when the model is already correct, the sigmoid approaches 0 and the update is weak.

However, the actual differentiation wasn't shown in detail. GPT didn't explicitly write out that $\nabla_\theta \hat{f}_\theta(x, y) = \beta \cdot \nabla_\theta \log \pi_\theta(y|x)$, which is what connects the gradient to actual network parameters. For implementation, this matters because it determines what gets backpropagated. The interpretation was insightful but the mathematical execution was incomplete.

Part (g): Plackett-Luce extension sketched adequately

Extending to rankings rather than pairwise comparisons requires showing $Z(x)$ still cancels in the more complex Plackett-Luce formula. The probability of ranking τ is $\prod_k \exp(r(x, y_{\tau(k)})) / \sum_j \exp(r(x, y_{\tau(j)}))$. When substituting $r(x, y) = \beta \cdot \log(\pi^*/\pi_{ref}) + \beta \cdot \log Z(x)$, each exponential becomes $(\pi^*(y|x)/\pi_{ref}(y|x)) \cdot Z(x)$, so $Z(x)$ appears in both numerator and denominator of each fraction and cancels. GPT correctly identified this cancellation pattern and wrote the final result in terms of π^* and π_{ref} only. The explanation was brief but captured the essential algebra.