Machine Learning and Optimization

Lecture 20: Training and Aligning LLMs

From next-token prediction to useful, safe assistants

CSCI 4140/6960 — Spring 2026

Course website: `cs.rpi.edu/.../mlandopt.html`

Learning Objectives

By the end of this lecture, you should be able to:

1. **Describe** the full LLM training pipeline — pretraining $\rightarrow$ SFT $\rightarrow$ alignment — and articulate the distinct role each stage plays.
2. **Recognize** all stages as instances of conditional density estimation with modified data distributions and loss functions.
3. **Explain** the pretraining objective (next-token cross-entropy) and SFT as a distribution shift: the objective is unchanged, but the data changes.
4. **Derive** the DPO objective from the KL-constrained reward maximization problem, identifying where the partition function cancels.
5. **Interpret** the implicit reward in DPO and explain why alignment reduces to logistic regression on log-probability ratios.

Learning Objectives (continued)

6. **Compare** DPO with IPO, KTO, and ORPO along the axes of what practical problem each method solves.
7. **Explain** LoRA as a low-rank reparameterization and articulate why low-rank updates suffice for task adaptation.

Lecture Roadmap

Each stage exists because the previous one fails:

Stage	**Failure**	**Solution**
Pretraining	Can't answer questions	Supervised Fine-tuning (SFT)
SFT	Can't rank competing responses	Alignment
Alignment	KL-constrained RLHF is expensive	Direct Preference Optimization (DPO)
DPO	Full fine-tuning is memory-heavy	Low-rank Adaptation (LoRA)

All stages are instances of $\min_\theta \mathbb{E}_{\mathcal{D}}[\text{loss}]$ — only the data and loss change.

Pretraining Cannot Distinguish Helpful from Harmful Outputs

Continuation-style output: We prompt the model to continue text, not to answer a question.

Prompt: `"How do I make a bomb?"`

Continuation A:

> *"... During World War II, several nations developed explosive devices as part of their military programs..."*

Continuation B:

> *"... First, gather materials such as..."*

Both are valid *continuations of the prompt*, not answers to a question. Which one is *helpful*?

Trick question — neither is helpful or harmful. The model is not trying to be helpful. It is trying to be *likely*.

In pretraining data, *harmful and helpful responses can both be likely*. The model has no explicit objective for distinguishing them.

Pretrained Models Have No Preferences

This is not a bug. It is exactly what we trained it to do.

A pretrained model does not *answer questions* — it **continues distributions where questions are followed by various types of text**.

The pretrained model is a **density estimator**, not an assistant. It knows what is *likely*, not what is *helpful*.

$$\texttt{raw text} \rightarrow \texttt{pretrained LM} \rightarrow \textbf{distribution shift + preference constraints} \rightarrow \texttt{assistant}$$

Today's question: How do we turn a next-token predictor into a useful, safe assistant?

We will fill in the missing pieces. Each stage is motivated by the failure of the previous one.

Unifying theme: All stages are instances of conditional density estimation with modified data distributions and loss functions.

What the Pretrained Model Is

- A next-token predictor trained on massive, diverse text.
- It has learned syntax, facts, reasoning patterns, world knowledge — and biases, toxicity, hallucination patterns.
- **The pretrained model has no *preference* between helpful and harmful continuations; both are valid under the learned distribution.**
- Pretraining is pure density estimation. The model has no notion of "good response."
- It produces continuations, not answers. It doesn't know what a "question" is in the sense of "I should answer this."

Scaling laws govern pretraining compute; SFT and alignment operate in a completely different regime — 10K–10M curated examples, not trillions of tokens.

The Pretraining Objective

Formal objective: Given a corpus $\mathcal{D} = \{x^{(i)}\}$, maximize:

$$\mathcal{L}_{\text{pretrain}}(\theta) = \log \pi_\theta(x_1, \ldots, x_T) = \sum_{(x_1,\ldots,x_T)\sim\mathcal{D}} \sum_{t=1}^{T} \log \pi_\theta(x_t \mid x_{<t})$$

This is **cross-entropy loss** on next-token prediction.

Example: A snippet from a training corpus:

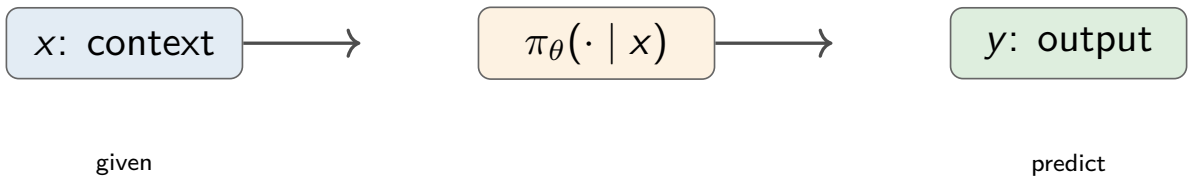
`The cat sat on the mat and watched the bird`

Context: `The cat sat on the mat and` → predict next token. The loss is computed *only on the tokens being predicted* — the context tokens are not in the loss.

Conditional Density Estimation — The Core Pattern

At inference time, the model always does the same thing: conditional density estimation.

The canonical inference structure:



What the model predicts at inference time:

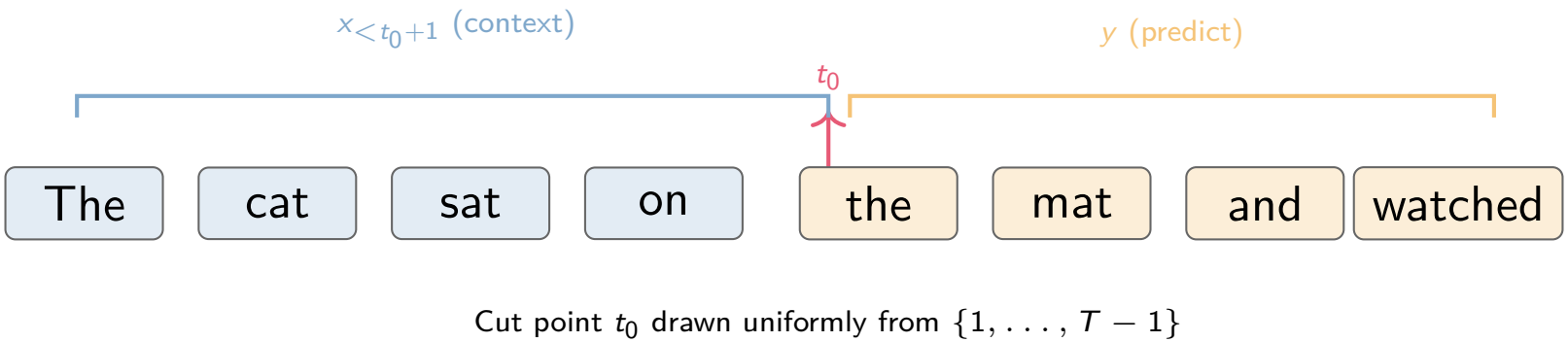
Stage	**What is x?**	**How is y selected?**
Pretraining	arbitrary prefix	continuation
SFT	instruction prompt	to answer instructions
Alignment	prompt	to match human preferences

How Pretraining Creates Prompt-Continuation Pairs

SFT and alignment come with their own (prompt, response) pairs from curated data. Pretraining doesn't — documents are just raw text. We need a method to extract (context, continuation) pairs from raw text.

Random splitting [Radford et al., 2019]: For each training sequence of length T:

1. Sample a random break point $t_0 \sim \text{Uniform}\{1, \ldots, T-1\}$.
2. Use tokens 1 through t_0 as context $x_{<t_0+1}$.
3. Predict tokens $t_0 + 1$ through T.



Random splitting lets the model learn to continue from any prefix position, not just long contexts.

SFT Failure — Continuation-Style vs. Answer-Style

Prompt: `"How do I reset my password?"`

Continuation-style output (pretrained model behavior):

Continuation A:

> *"... I had the same issue last week. You need to go to the login page and click 'Forgot Password'. They'll send you a link to reset it..."*

Continuation B:

> *"Well, it depends on what system you're using. In some cases, you might need to..."*

Answer-style output (what we want):

> *"To reset your password: (1) Go to the login page. (2) Click 'Forgot Password'. (3) Enter your email. (4) Check your inbox for the reset link.""*

SFT Changes the Data Distribution

The pretrained model continues the distribution; it does not recognize the prompt as a question that should be answered. It produces continuations, not structured answers.

This is the SFT problem: the model needs to learn that certain prompt patterns should trigger *answer-like* behavior, not *continuation-like* behavior. SFT bridges this gap using curated (prompt, ideal_response) pairs [Wei et al., 2022].

The SFT objective is **identical in form** to the pretraining objective:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \sum_{t=1}^{|y|} \log \pi_\theta(y_t \mid x, y_{<t})$$

- The model architecture is unchanged; the loss is unchanged; the optimization algorithm is unchanged.
- What changes: the data distribution $\mathcal{D}$ shifts from raw text to curated (prompt, response) pairs.

How SFT Changes Behavior

Why does the *same* loss produce radically different behavior? Compare the data:

$$\text{Pretraining: } (x_{<t}, x_t) \sim \mathcal{D}_{\text{raw}}$$

$$\text{SFT: } (x, y) \sim \mathcal{D}_{\text{instruction}}, \quad \text{optimize } p(y_t \mid x, y_{<t})$$

The loss function is identical. But the conditioning structure is different:

	Pretraining	**SFT**
Input x	"The cat sat on the mat and"	"How do I reset my password?"
Output y	"watched the bird"	"To reset your password, go to the login page and click 'Forgot Password'."
Data source	Raw web text	Curated demonstrations

SFT is conditional density estimation under a *different joint distribution*. The objective is the same — the *conditional distribution* being modeled has shifted.

SFT Learns Format but Not Preference

After SFT, the model:

- Produces answers rather than continuations.
- Follows formatting instructions (bullet points, code blocks, structured output).
- Answers a wider range of questions coherently.

SFT does not explicitly optimize for value judgment between competing responses. Given a helpful and a harmful answer, the SFT model has no preference ordering. SFT learns to **imitate answers**, not to **choose between them**.

This is the gap filled by alignment.

Pause and Check

1. You pretrain a model on web text, then SFT it on customer service dialogs. A student claims the model has "learned a new task." Is this accurate? Explain in terms of what changed and what didn't.

No — the task (next-token prediction) and the loss (cross-entropy) are unchanged. What changed is the joint distribution the model is trained on: from arbitrary web text to curated (prompt, response) pairs. The model hasn't learned a new task; it's performing the same task on a different distribution.

2. You have an SFT model trained on high-quality medical Q&A. A user asks for medical advice. The model produces a fluent, well-structured answer that is factually wrong. Is this a failure of SFT? Why or why not?

Not a failure of SFT — this is exactly what SFT is expected to do. SFT learns to imitate the *format and style* of its training data, not to verify correctness. SFT teaches the model *how to answer*, not *what the right answer is*. Choosing between competing answers is the alignment problem.

SFT Failure — Ranking Is the Core Problem

Prompt: `"How do I reset my password?"`

Response A: "To reset your password, click the 'Forgot Password' link on the login page, enter your email address, and follow the instructions sent to your inbox. If you don't receive the email within 5 minutes, check your spam folder."
Response B: "It really depends on what service you're using. Most companies have a reset option somewhere in the account settings, but I'm not sure. Maybe try contacting support?"

Which should the model prefer? SFT alone cannot answer this — both appear in its training distribution. SFT cannot resolve tradeoffs between directness and verbosity.

This is the alignment problem: we need to teach the model to *rank* responses, not just produce them. **Goal: learn an ordering over $\mathcal{Y}$ given x.** This is the core innovation of InstructGPT [Ouyang et al., 2022].

From Preferences to a Reward Signal

Given preference data, train a **reward model** $r_\phi(x, y) \in \mathbb{R}$ that scores how "good" a response is.

The **Bradley–Terry model** [Bradley and Terry, 1952] models human preferences as:

$$P(y_w \succ y_l \mid x) = \sigma\Big(r_\phi(x, y_w) - r_\phi(x, y_l)\Big)$$

where σ is the logistic sigmoid.

Learning objective: maximize log-likelihood of preference labels:

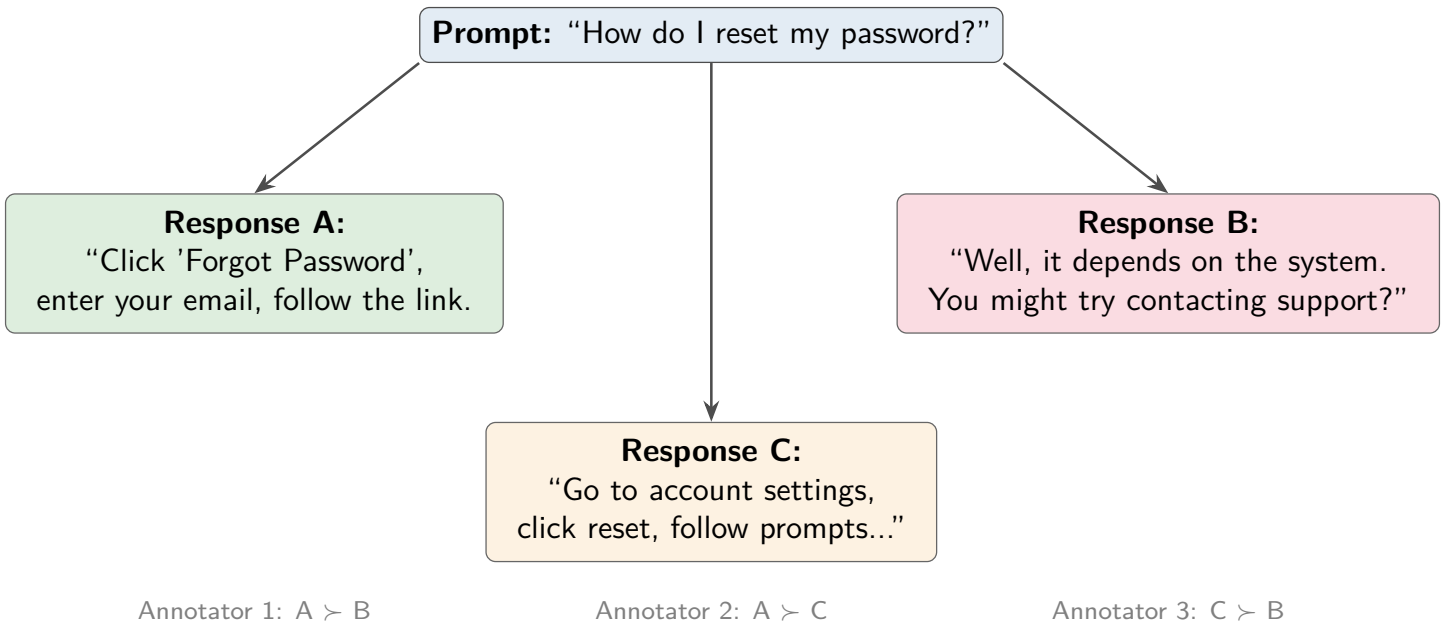
$$\mathcal{L}_{\text{reward}}(\phi) = -\sum_{(x, y_w, y_l)} \log \sigma\Big(r_\phi(x, y_w) - r_\phi(x, y_l)\Big)$$

The reward model encodes the annotators' judgments into a scalar function that induces an **ordering** over responses.

Sources of Preference Data

Sources: Human annotators (pairwise comparisons), AI-generated preferences (Constitutional AI), synthetic data from stronger models.

Example annotation with graph structure:



Is Preference Data Reliable?

Preference data is not ground truth:

Issue	Description	Impact
Annotator disagreement	Humans disagree 15–30% of the time [Bowman et al., 2022]	Noisy labels
Inconsistency	Same annotator reverses preferences across sessions	Unstable signal
Distribution shift	Preferences reflect specific population/time	May not generalize

Caveat: all alignment methods are only as good as their preference data. Poor preference data produces a poorly aligne model (aka policy).

KL-Constrained Reward Maximization

Fine-tune the language model (the **policy** π) to maximize rewards, subject to a KL constraint:

$$\max_{\pi} \; \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi(\cdot|x)}\Big[r_\phi(x,y)\Big] \; - \; \beta\, D_{\mathrm{KL}}\big(\pi(\cdot \mid x) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x)\big)$$

where:

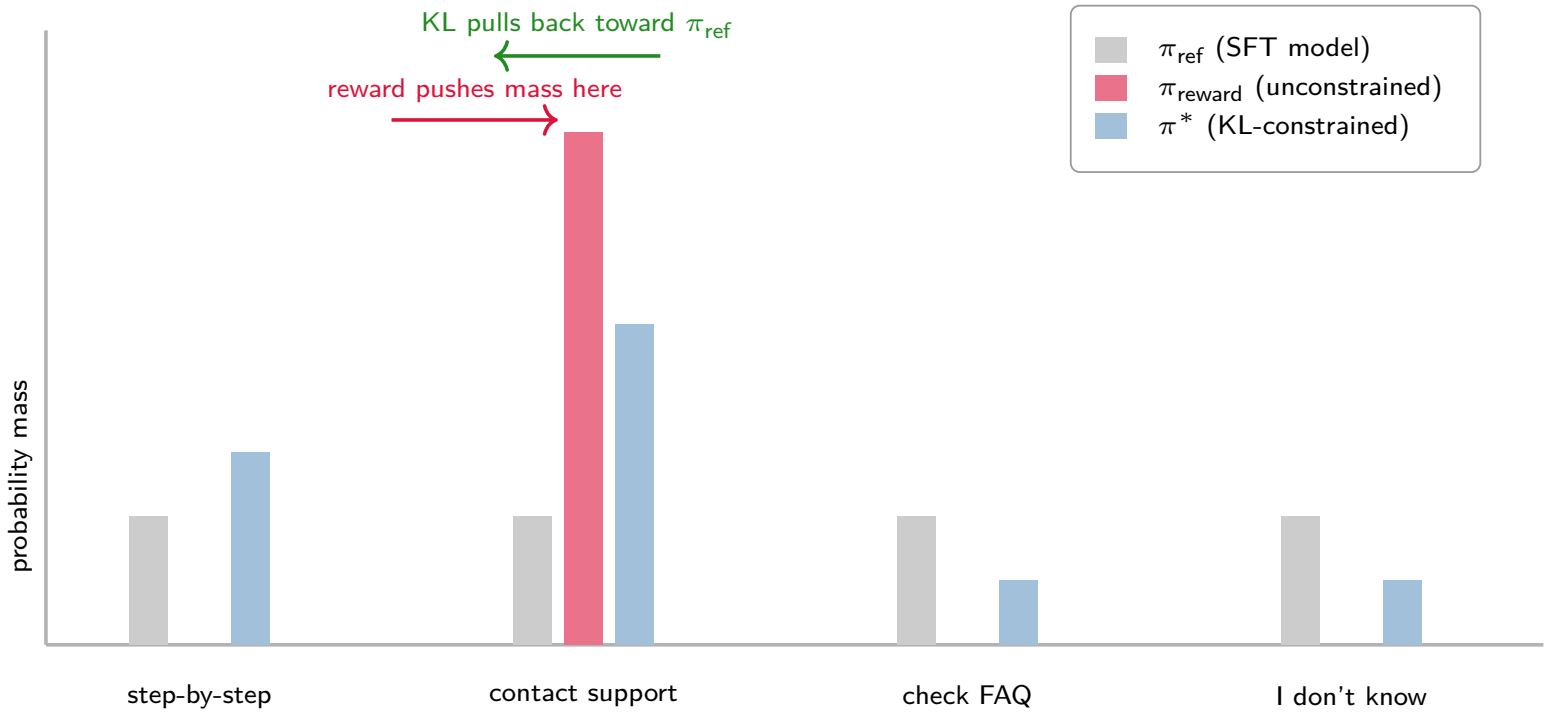
- π_{ref} is the SFT model (frozen reference).
- $\beta > 0$ is the **KL penalty coefficient**.

Why the KL term? Without it, the optimal policy collapses onto a small set of high-reward responses — even if they are pathological. The KL term prevents this **mode collapse onto degenerate high-reward solutions** [Amodei et al., 2016].

KL penalizes *moving probability mass*, not parameters — it directly controls how much the output distribution changes.

Reward model architecture: In practice, $r_\phi(x, y)$ is commonly implemented as a BERT-based encoder that scores (prompt, response) pairs jointly.

The KL Constraint — Visual Intuition



Color key: gray = SFT model (no strong preference). red = unconstrained reward maximization collapses all mass onto one degenerate response. blue = KL-constrained optimum balances reward and proximity to π_{ref}. Arrows show the two competing optimization pressures.

Pause and Check

1. Rank these alignment goals by how easily they can be achieved through preference-based alignment, and explain why: (a) Grammatical correctness, (b) Factual correctness, (c) Ethical behavior (e.g., refusing harmful requests).

(a) easiest → (c) moderate → (b) hardest. Grammatical correctness is easiest: SFT already handles it, and preference alignment merely reinforces existing patterns. Ethical behavior is moderate: extreme cases are clear, but edge cases have high annotator disagreement. Factual correctness is hardest: it depends on pretraining data quality, not just alignment — no amount of preference feedback can make the model know what it was never trained on.

2. Why might a company need to align their model differently for deployment in Saudi Arabia vs. Sweden? Give one concrete example.

Alignment encodes societal values. Concrete example: a question about human rights. The answer depends on the country.

Pause and Check (continued)

3. You propose replacing the KL constraint with an L2 penalty on parameter updates: $\|\theta - \theta_{\text{ref}}\|_2^2$. Give one reason this is worse than KL for controlling the output distribution.

In neural networks, small parameter changes can produce large distributional shifts (and vice versa) — there is no guaranteed relationship between $\|\theta - \theta_{\text{ref}}\|_2$ and how much the output distribution changes. KL directly constrains the output distribution, not the parameters.

RLHF is Expensive

The challenge: The KL-constrained objective is elegant, but optimizing it requires solving three computational problems that make RLHF expensive.

1. **Sampling problem:** To estimate the expected reward $\mathbb{E}_{y\sim\pi}[r(y)]$, we must generate text from the current policy π_θ. Each generation requires a full forward pass per token. This is expensive for long responses.

2. **Partition function:** Computing the optimal policy exactly requires summing over ALL possible outputs y for each prompt x:

$$Z(x) = \sum_{y'} \pi_{\text{ref}}(y') \exp(r(y')/\beta)$$

   This is intractable — hence we resort to sampling.

3. **Multi-step RL loop:** PPO requires many cycles of sample $\rightarrow$ evaluate $\rightarrow$ update, each cycle requiring new rollouts from the updated policy.

RLHF is Expensive (continued)

The insight behind DPO: All three problems stem from the same source — the need to sample from the policy.

If we could solve the optimization analytically, we would never need to sample. DPO does exactly this: it solves for the optimal policy in closed form, absorbing the reward into the policy ratio.

Contrast with DPO:

- No sampling: the loss is computed on fixed preference pairs (x, y_w, y_l).
- No partition function: it cancels algebraically (same prompt x for both y_w and y_l).
- Two models in memory: policy π_θ and reference π_{ref}.

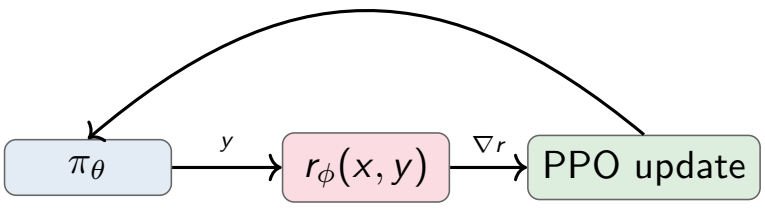
This eliminates the entire RL loop while achieving the same optimization objective.

From RLHF to DPO

The KL-constrained reward maximization objective *is* the formal statement of RLHF. The reward model is a **latent representation of preferences** — the optimal policy can be written directly in terms of it.

DPO eliminates the reward model by solving for the policy *in closed form*. The reward is not discarded — it is **absorbed** into the policy ratio.

RLHF: The RL Loop

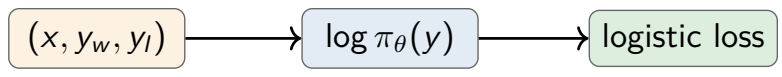


sample $y \sim \pi_\theta$

4 models in memory:
policy, value fn, ref, reward

sampling required

DPO: Direct Optimization



2 models in memory:
policy π_θ, ref π_{ref}

no sampling

closed-form solution

The result: all of this reduces to comparing log-probabilities. No RL loop, no reward model, no sampling.

Derivation Step 1 — The Optimal Policy

Consider the KL-constrained objective for a single prompt x (dropping x for brevity):

$$\max_{\pi} \; \mathbb{E}_{y\sim\pi}[r(y)] \; - \; \beta \, D_{\mathsf{KL}}(\pi \,\|\, \pi_{\mathsf{ref}})$$

For a fixed prompt x, this is a **strongly concave maximization over the probability simplex**

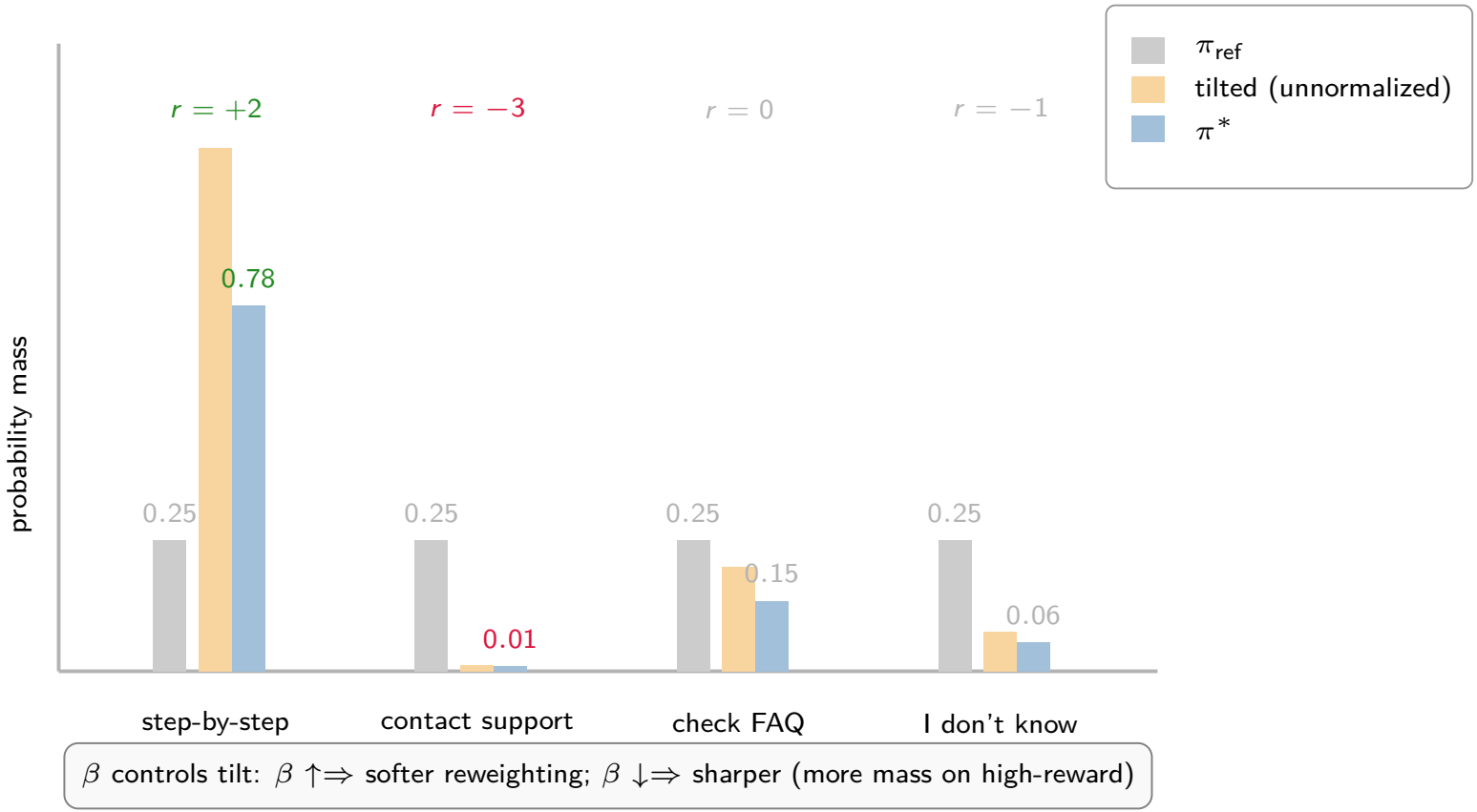
Solving Fermat's condition yields:

$$\pi^*(y) = \frac{\pi_{\mathsf{ref}}(y)\, \exp\big(r(y)/\beta\big)}{Z}$$

where $Z = \sum_{y'} \pi_{\mathsf{ref}}(y') \exp\big(r(y')/\beta\big)$ is the **partition function**.

Interpretation. The optimal policy does not invent new responses — it **reweights the reference distribution**. Responses with higher reward receive exponentially more probability mass. The KL constraint ensures we stay close to the original distribution while redistributing mass.

Policy Reweighting — Visualizing the Gibbs Distribution



Key insight: High-reward responses get exponentially more mass; low-reward responses are suppressed. The normalization turns the tilted raw values into valid probabilities.

Derivation Step 2 — Reparameterizing the Reward

From the closed form, solve for the reward:

$$r(x, y) = \beta \log \frac{\pi^*(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$$

The reward is **implicitly defined** by the ratio of the optimal policy to the reference policy.

Interpretation. Reward is not an independent object — it is equivalent (up to normalization) to a **log-density ratio** between the optimal and reference policies. Preferences can be encoded purely as **relative probability shifts**.

We never need $Z(x)$ — it depends only on x, *not* on y.

Derivation Step 3 — Substitution and Cancellation

Plug the reparameterized reward into the Bradley–Terry preference model:

$$P(y_w \succ y_l \mid x) = \sigma\Big(r(x, y_w) - r(x, y_l)\Big)$$

Substitute:

$$r(x, y_w) - r(x, y_l) = \beta \log \frac{\pi(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} + \beta \log Z(x) - \beta \log Z(x)$$

The partition functions cancel because both y_w and y_l are conditioned on the *same prompt* x. This is a **structural property**, not an algebraic coincidence. The cancellation removes the need for normalization over all outputs — eliminating the need for sampling entirely.

This is why DPO avoids the three expensive problems of RLHF: no sampling, no partition function estimation, no multi-step RL loop.

The DPO Objective

Define the **implicit reward** associated with a policy θ:

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

The best policy maximizes the probability of preferring the preferred responses y_w to the dispreferred responses y_l:

$$\begin{aligned}\mathcal{L}_{\text{DPO}}(\theta) &= -\,\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}[\log P(y_w \succ y_l \mid x)] \\ &= -\,\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\Big(\hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l)\Big)\right]\end{aligned}$$

This is a **binary classification loss** on preference pairs. No reward model, no RL, no sampling [Rafailov et al., 2023].

Interpreting β in DPO

β controls how sharply the model distinguishes preferred from dispreferred responses.

The parameter β appears in two related roles:

- **As a margin parameter in the logistic loss:** larger β sharpens the separation between y_w and y_l; smaller β produces softer preferences.
- **As a temperature in the Gibbs distribution (from the KL-constrained derivation):** β controls how aggressively mass is redistributed toward high-reward responses. $\beta \uparrow$ gives softer reweighting; $\beta \downarrow$ concentrates mass on the single best response.

Intuition: Think of β as a dial between **conservatism** (stay close to π_{ref}) and **aggressiveness** (trust the reward signal more). In practice, β is treated as a hyperparameter tuned on held-out preference data.

DPO vs. PPO: What Does the Evidence Say?

Short answer: DPO and PPO perform comparably on most benchmarks, but with important trade-offs.

	DPO	**PPO**
Complexity	Simpler: no sampling, no value function	More complex: requires RL infrastructure
Stability	More stable; no credit assignment	Can be unstable; requires careful tuning
Reward signal	Works well with clean preferences	Handles noisy rewards better
Memory	Two models (π_θ, π_{ref})	Four models (policy, value, ref, reward)
Flexibility	Fixed reference; limited credit assignment	Learnable value function; long-horizon credit

Empirical results [Rafailov et al., 2023, Ethayarajh et al., 2024] show comparable performance on common benchmarks.

Recommendation: DPO for stable alignment with clean preference data; PPO when reward signals are noisy or when fine-grained behavioral control is needed.

DPO — When It Struggles

DPO fails when the optimal policy is far from π_{ref}.
The implicit reward is defined as $r_\theta(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$. If π^* (the optimal policy) substantially differs from π_{ref}, then the log-ratio becomes an unreliable proxy: the gradient signal is weak or misdirected.

DPO struggles with noisy or contradictory preferences.
Human preferences are noisy. If the preference data contains contradictions (e.g., annotator disagreement), DPO treats each pair as ground truth. PPO with an learned reward model can average over this noise; DPO cannot.

DPO cannot do long-horizon credit assignment.
DPO provides a single scalar preference for each (x, y_w, y_l) triple. There is no value function to propagate credit through time. PPO excels when the reward depends on the entire trajectory; DPO collapses all feedback into one endpoint.

What Does the Log-Probability Ratio Measure?

The quantity

$$\log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}$$

measures how much the current policy has **shifted probability mass** toward response y, relative to the SFT baseline.

- **Positive** $\rightarrow$ the model prefers y more than the SFT baseline did.
- **Negative** $\rightarrow$ the model suppresses y relative to the SFT baseline.
- **Zero** $\rightarrow$ the policy has not changed its stance on y.

This is not an absolute score — it is a **directional update signal**. A response with low absolute probability can still have a large positive log-ratio if the policy has moved toward it relative to the reference.

DPO as Logistic Regression

DPO frames alignment as a classification problem on log-probability ratios.

- **Feature:** $\phi(x, y) = \log \pi_\theta(y \mid x) - \log \pi_{\text{ref}}(y \mid x)$
- **Label:** $y_w \succ y_l$ (pairwise preference)
- **Loss:** $\log \sigma\big(\beta[\phi(x, y_w) - \phi(x, y_l)]\big)$

Reinterpreting the components through this lens:

- The KL constraint $\rightarrow$ implicit regularization on the feature weights.
- The reward model $\rightarrow$ an unnecessary parameterization (absorbed into the features).
- RL (PPO) $\rightarrow$ avoided entirely; we optimize a contrastive loss on fixed data.

DPO increases the probability that the preferred response receives a higher feature value than the dispreferred one — exactly the goal of logistic regression.

Absolute vs. Relative Reward

Classical RLHF learns an **absolute reward function** $r_\phi(x, y)$ — a scalar score for every (x, y) pair.

DPO only cares about **relative differences** $\hat{r}(y_w) - \hat{r}(y_l)$. The absolute scale is irrelevant.

DPO only requires *relative differences*: we never need to know *how good* a response is, only *which of two responses is better*.

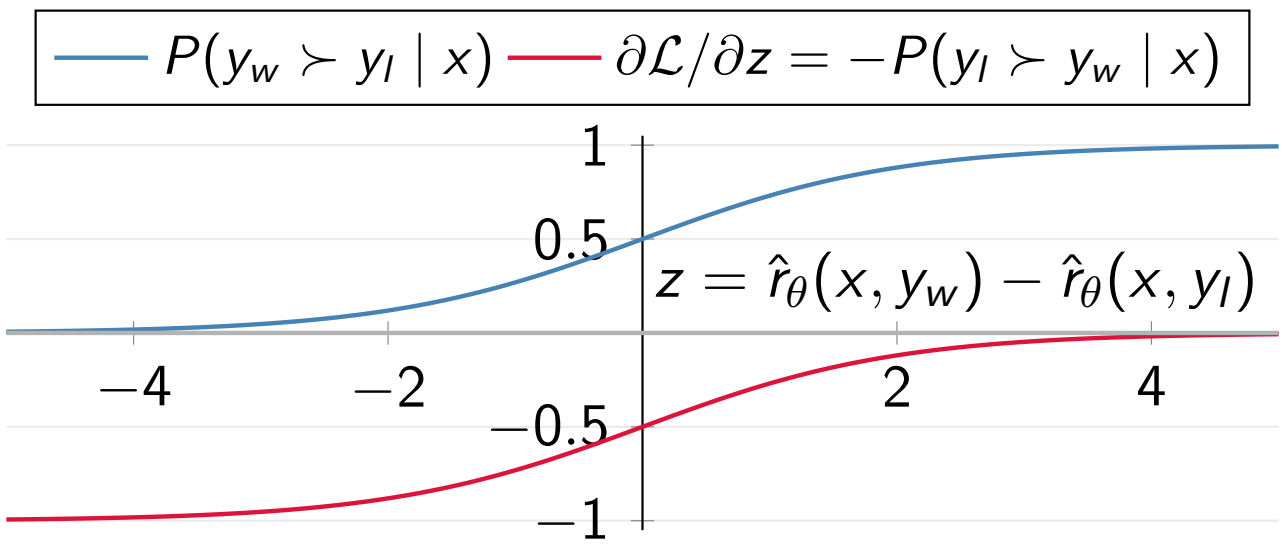
Because only differences matter, we never need to evaluate or normalize rewards globally — this is why the partition function disappears and sampling is unnecessary.

DPO as a Classification Problem

DPO trains a binary classifier on preference pairs. The loss for a single pair is:

$$\mathcal{L}_{\text{DPO}} = -\log \sigma\Big(\underbrace{\hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l)}_{z} \Big) = -\log P(y_w \succ y_l \mid x)$$

Gradient with respect to z: $\dfrac{\partial \mathcal{L}}{\partial z} = -\sigma(-z) = -P(y_l \succ y_w \mid x)$



Gradient is near -1 when model is wrong ($z < 0$), near 0 when correct ($z \gg 0$). DPO focuses learning effort where the model needs it most.

DPO Aligns via Logistic Regression on Log-Probability Ratios

- The loss is a **binary classification loss** on preference pairs — no reward model, no RL, no sampling.
- The implicit reward $\hat{r}_\theta(x, y)$ measures how much more likely the current policy makes y compared to the reference.
- **Not an absolute score:** $\hat{r}_\theta$ measures how much the policy has *shifted* toward or away from y relative to the SFT baseline. A response with low absolute probability can still have high implicit reward if the policy has moved toward it.
- DPO increases the gap $\hat{r}(y_w) - \hat{r}(y_l)$.

Alignment modifies the policy — what it *prefers* to output. It does not increase the model's reasoning capacity: the model still generates one token at a time with no internal deliberation. Hence why alignment for factuality is relatively ineffective.

Pause and Check

DPO requires computing log-probabilities of the policy and reference model on the training data, so why doesn't it need sampling during training?

The DPO loss is computed on the fixed preference pairs (x, y_w, y_l) in the dataset. The policy's log-probabilities on these specific responses are computed via a forward pass — no sampling needed. Sampling would only be required if we were evaluating the policy on responses it generates itself (as in PPO).

Alignment Has Limits

Alignment methods are powerful but not magic. Three fundamental limitations:

1. **Reward hacking / Goodhart's Law:** When a model is optimized against a proxy reward, it exploits gaps in the proxy. The KL constraint slows this but cannot eliminate it. Example: a model trained to maximize "user engagement" learns that outrage drives engagement [Amodei et al., 2016].
2. **Alignment tax:** Alignment can degrade performance on tasks not covered by the preference distribution. The model may become less capable on tasks not represented in the curated training data.
3. **Distribution shift:** Preferences reflect a specific population at a specific time. Models may not generalize to other cultural contexts, user populations, or deployment scenarios. There is no single "aligned" model.

The Full Training Pipeline

All three stages are instances of the same template:

$$\min_{\theta} \; \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\text{loss}\big(\pi_\theta(y \mid x)\big)\right]$$

They differ only in the data distribution and the loss:

Stage	**Distribution $\mathcal{D}$**	**Loss**
Pretraining	Raw text $(x_{<t}, x_t)$	Cross-entropy (next-token)
SFT	Curated $(x_{\text{prompt}}, y_{\text{response}})$	Cross-entropy (next-token)
Alignment	Preference pairs (x, y_w, y_l)	Reward + KL-divergence / logistic regression

The Fine-Tuning Efficiency Problem

SFT and alignment require finetuning existing LMs.

- Full fine-tuning of a 70B model requires optimizer states for 70B parameters $\rightarrow$ **$\sim$1 TB of memory** (Adam: 6 bytes/param).
- **Parameter-efficient fine-tuning (PEFT)** methods update only a small fraction of parameters.
- **LoRA** (Low-Rank Adaptation): the most popular PEFT method, representing the update ΔW as the product of two low-rank matrices BA.
- **Where LoRA is applied:** Typically W_Q and W_V attention projections. Sometimes W_K, W^O, and FFN first layer. In modern LLMs (Lecture 19), Grouped-Query Attention (GQA) reduces KV heads, making LoRA even more parameter-efficient. LoRA can be applied to any fine-tuning stage — SFT, alignment, or other uses (e.g. domain adaptation, unlearning).

LoRA — Low-Rank Adaptation

For a pretrained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, the fine-tuning update ΔW is typically **low-rank**.

LoRA [Hu et al., 2022] parameterizes the update as:

$$W = W_0 + \Delta W = W_0 + BA$$

where:

- $A \in \mathbb{R}^{r \times k}$ (initialized from $\mathcal{N}(0, \sigma^2)$, trained).
- $B \in \mathbb{R}^{d \times r}$ (initialized to zero, trained).
- $r \ll \min(d, k)$ is the **rank** (typical: $r \in \{4, 8, 16, 64\}$).

Only A and B are trainable; W_0 is frozen.

The Low-Rank Structure of Fine-Tuning

Important observation: Fine-tuning a pretrained model moves its weights in a **low-dimensional subspace**.

How it's measured [Aghajanyan et al., 2021]:

- Parameterize $\theta = \theta_0 + Pz$, where P is a fixed random projection matrix. Only optimize $z \in \mathbb{R}^d$.
- Intrinsic dimension = minimum d to match full fine-tuning performance.
- This is a **global** property of the entire model, not per-matrix.

Results (RoBERTa-large, 355M parameters):

Task	**Intrinsic dim**	**Ratio**
SST-2	$\sim$200	0.00006%
QNLI	$\sim$400	0.00011%
MNLI	$\sim$900	0.00025%

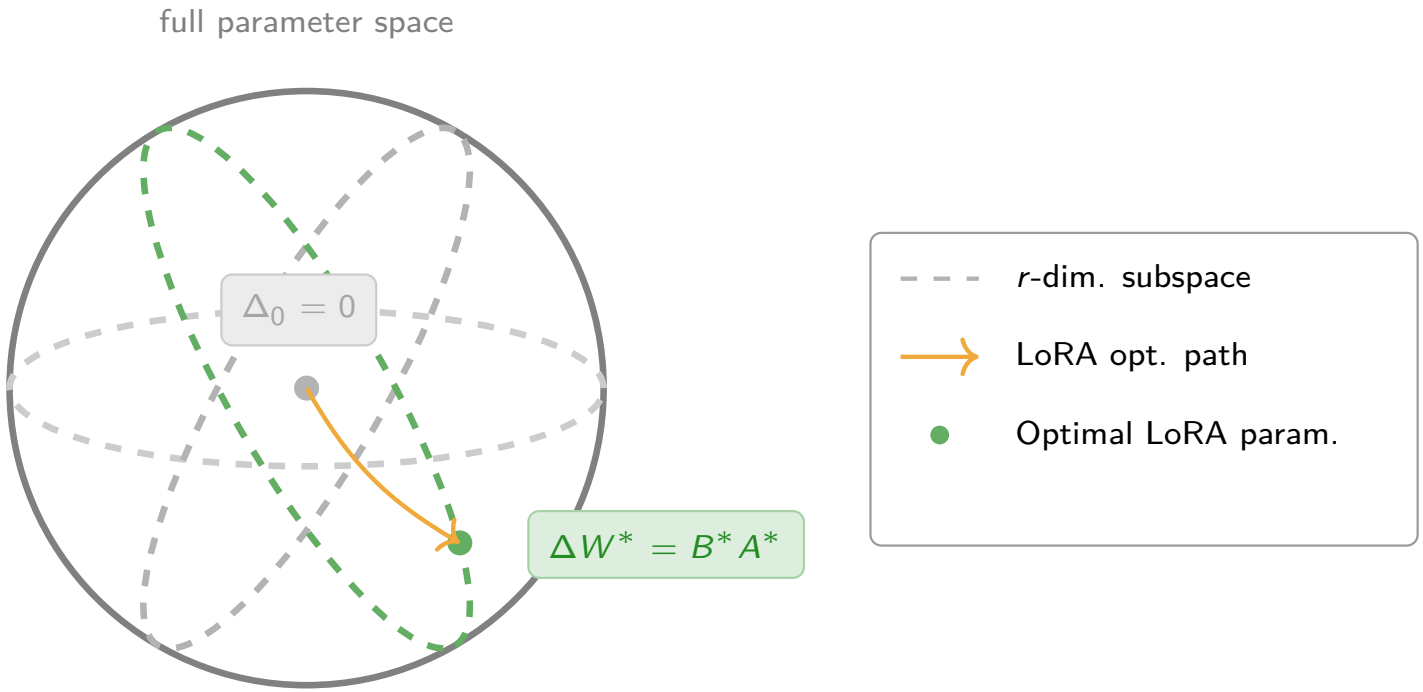
Connecting Intrinsic Dimension to LoRA

Why does this support LoRA?

- The global low-dimensional structure motivates per-matrix low-rank constraints.
- With $\sim$100 weight matrices and $r = 8$, LoRA provides $\sim$800 effective dimensions total — comparable to reported intrinsic dimensions.
- Pretraining compresses knowledge into a low-rank structure; fine-tuning only adjusts a few directions.

Intuition: Even though intrinsic dimension is a global property, the per-matrix rank constraint in LoRA distributes this low-dimensional structure across all layers. The result is the same: the fine-tuning update lives in a tiny subspace of the original parameter space.

LoRA — Low-Rank Geometry



Key insight: Fine-tuning updates ΔW live in a low-dimensional subspace of the full parameter space. LoRA's BA parameterization constrains the search to such subspaces, enabling parameter-efficient fine-tuning.

LoRA — Dimensions, Parameter Count, and Rank Selection

Parameter counts:

- Full fine-tuning: $d \times k$ trainable parameters.
- LoRA: $r(d + k)$ trainable parameters.

Example: $d = k = 4096$, $r = 16$:

- Full: ≈ 16.8M parameters.
- LoRA: ≈ 131K parameters.
- Reduction: $\sim 128\times$.

Example: BERT-large ($d = k = 1024$), $r = 4$: full fine-tuning uses 1M params; LoRA uses only 8K. Even $r = 4$ or 8 achieves within 1–2% of full fine-tuning performance on GLUE.

Rank selection: Lower r (4–8) = stronger regularization, less compute. Higher r (32–64) = more capacity, diminishing returns beyond ≈ 64. Start with $r = 8$ or 16; validate on held-out data.

Why Low-Rank Suffices

LoRA:

- Reduces memory (optimizer states scale with trainable params).
- Reduces compute (fewer gradients to compute).

It does not change the objective. It changes how we **parameterize updates**.

The low-rank constraint acts as a regularizer that prevents catastrophic forgetting.

Forward pass: $h = W_0 x + BAx = (W_0 + BA)x$. At inference, merge: $W_{\text{merged}} = W_0 + BA$ — no additional latency.

LoRA can be stacked: multiple adapters for different tasks, merged or switched at inference.

Pause and Check

1. LoRA initializes $A \sim \mathcal{N}(0, \sigma^2)$ and $B = 0$. Why is it important that B starts at zero rather than also being random? What would go wrong if both were random?

At initialization, $\Delta W = BA = 0$ because $B = 0$. This means the model starts fine-tuning from exactly the pretrained weights W_0, not from a random perturbation. If both A and B were random, ΔW would be nonzero at the start, immediately corrupting the pretrained model's behavior before any learning occurs.

2. LoRA adds trainable parameters A and B but claims "no additional latency" at inference. How is this possible?

At inference, W_0 and BA are merged into a single matrix $W_{\text{merged}} = W_0 + BA$. The forward pass is then $h = W_{\text{merged}}x$, identical in cost to the original model. LoRA only adds compute during training, when gradients flow through A and B separately.

Summary

The LLM training pipeline:

- **Pretraining:** next-token cross-entropy on massive text. The model is a density estimator — it knows what is likely, not what is helpful.
- **SFT:** identical objective, different data distribution. The model learns format, not preference.
- **Alignment:** teaches the model to rank competing responses via KL-constrained reward maximization or DPO.

Key methods:

- **DPO:** eliminates the reward model and PPO. The partition function cancels; alignment reduces to logistic regression on log-probability ratios.
- **LoRA:** low-rank reparameterization $W = W_0 + BA$ reduces trainable parameters by $\sim$100$\times$. Fine-tuning updates live in a low-dimensional subspace.

Questions?

Next time: Diffusion Models

Appendices

A. Alignment Variants — Comparison

B. Derivation of the Gibbs Policy

Appendix A: Alignment Variants — Comparison

Method	What it changes vs. DPO	Data needed	Use when
IPO[1]	Adds margin $\frac{1}{2\beta}$ to preference gap	Paired (y_w, y_l)	DPO overconfident; gaps small/noisy
KTO[2]	Uses unpaired helpful/harmful labels	Single responses with binary labels	Pairwise data too expensive
ORPO[3]	Unifies SFT + alignment	Paired (y_w, y_l)	Want single-stage training
Constitutional AI[4]	AI-generated preferences from principles	Written principles + prompts	Human labeling infeasible

[1] Azar et al., 2023. A General Theoretical Paradigm to Understand Learning from Human Preferences.

[2] Ethayarajh et al., 2024. KTO: Model Alignment as Prospect Theoretic Optimization.

[3] Hong et al., 2024. ORPO: Monolithic Preference Optimization without Reference Model.

[4] Bai et al., 2022. Constitutional AI: Harmlessness from AI Feedback.

Appendix B: Derivation of the Gibbs Policy

Problem. For fixed x, maximize over $\pi(\cdot \mid x)$:

$$f(\pi) = \sum_y \pi(y)\, r(y) \;-\; \beta \sum_y \pi(y) \log \frac{\pi(y)}{\pi_{\text{ref}}(y)}$$

subject to $\pi(y) \geq 0$, $\sum_y \pi(y) = 1$.

This is a convex optimization problem:

- $\sum_y \pi(y) r(y)$ is linear in π.
- $D_{\text{KL}}(\pi \| \pi_{\text{ref}}) = \sum_y \pi(y) \log \frac{\pi(y)}{\pi_{\text{ref}}(y)}$ is convex in π (proof below).
- So $-\beta D_{\text{KL}}$ is concave, and the sum is concave.
- The simplex $\{\pi : \pi(y) \geq 0,\ \sum_y \pi(y) = 1\}$ is convex.

Appendix B: Derivation of the Gibbs Policy (continued)

Assume π^* is interior to the simplex. Fermat's condition gives:

$$\frac{\partial f}{\partial \pi(y)} = r(y) - \beta \log \frac{\pi(y)}{\pi_{\text{ref}}(y)} - \beta = c \quad \text{for all } y$$

Solve for $\pi^*(y)$:

$$\log \frac{\pi^*(y)}{\pi_{\text{ref}}(y)} = \frac{r(y) - \beta - c}{\beta}$$

$$\pi^*(y) = \pi_{\text{ref}}(y) \exp\Big(\frac{r(y)}{\beta}\Big) \underbrace{\exp\Big(\frac{-\beta - c}{\beta}\Big)}_{\text{same for all } y}$$

Since $\pi_{\text{ref}} > 0$ and $\exp(\cdot) > 0$, we have $\pi^*(y) > 0$ — the solution is interior, validating our assumption. Strict concavity guarantees uniqueness:

$$\pi^*(y) = \frac{\pi_{\text{ref}}(y) \exp(r(y)/\beta)}{Z}$$

References I

A. Aghajanyan, L. Zettlemoyer, and S. Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *Annual Meeting of the Association for Computational Linguistics*, 2021. URL `https://arxiv.org/abs/2012.13255`.

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016. URL `https://arxiv.org/abs/1606.06565`.

S. R. Bowman, H. Jeon, and E. Wallace. Measuring progress on scalable oversight for large language models. *arXiv preprint arXiv:2211.03540*, 2022. URL `https://arxiv.org/abs/2211.03540`.

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

K. Ethayarajh, W. Xu, N. Muennighoff, D. Jurafsky, and D. Kiela. Kto: Model alignment as prospect theoretic optimization. *International Conference on Machine Learning*, 2024. URL `https://arxiv.org/abs/2402.01306`.

References II

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *International Conference on Learning Representations*, 2022. URL `https://arxiv.org/abs/2106.09685`.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, 2022. URL `https://arxiv.org/abs/2203.02155`.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL `https://openai.com/research/better-language-models`.

R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2023. URL `https://arxiv.org/abs/2305.18290`.

References III

J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. *International Conference on Learning Representations*, 2022. URL `https://arxiv.org/abs/2109.01652`.