Yale University

Language-Model Posttraining: RLHF and RLVR

S&DS 265 · Introductory Machine Learning · Lecture 23

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

Outline

1. Why a Pretrained Model Is Not Yet Useful
2. Supervised Fine-Tuning
3. Preference Modeling
4. RLHF: Policy Optimization on Responses
5. Reinforcement Learning with Verifiable Rewards

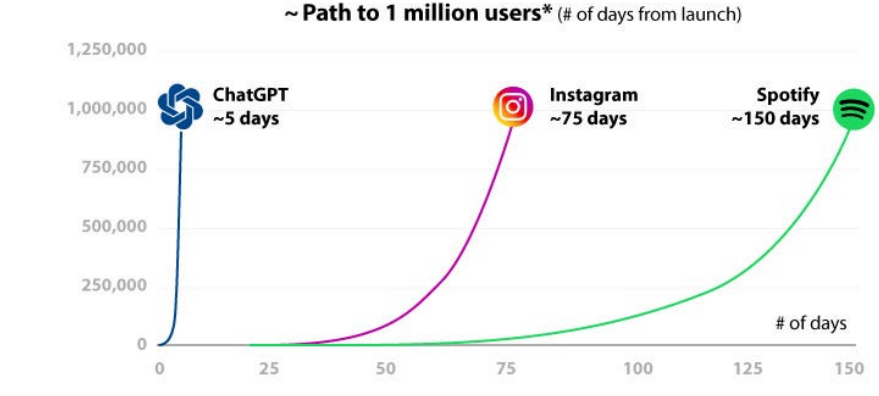
Learning Objectives

In this lecture you will learn:

1. How demonstrations, comparisons, and verifiers supply three different training signals;
2. How supervised fine-tuning turns demonstrations into a language-model objective;
3. How pairwise comparisons train a reward model, and what it does not identify;
4. How a whole response becomes one action, so that alignment is a contextual bandit;
5. How a verifier replaces the reward model where the answer can be checked.

Outline

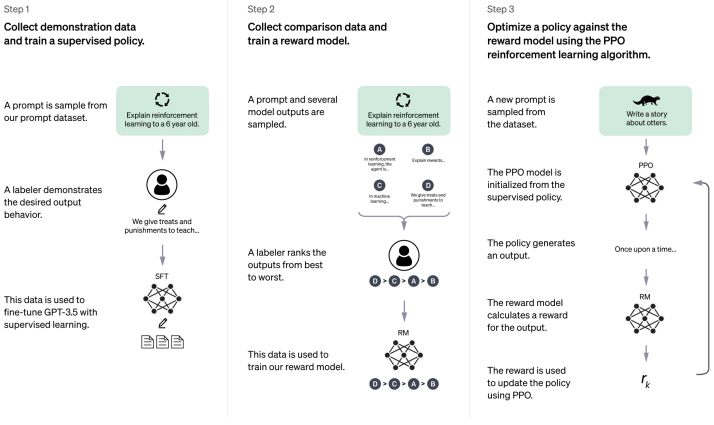
Success of ChatGPT



ChatGPT reaches one million users in about five days.

Chart as circulated in press coverage of the launch; its own sources line reads “Google, Subredditstats, Media Reports”.

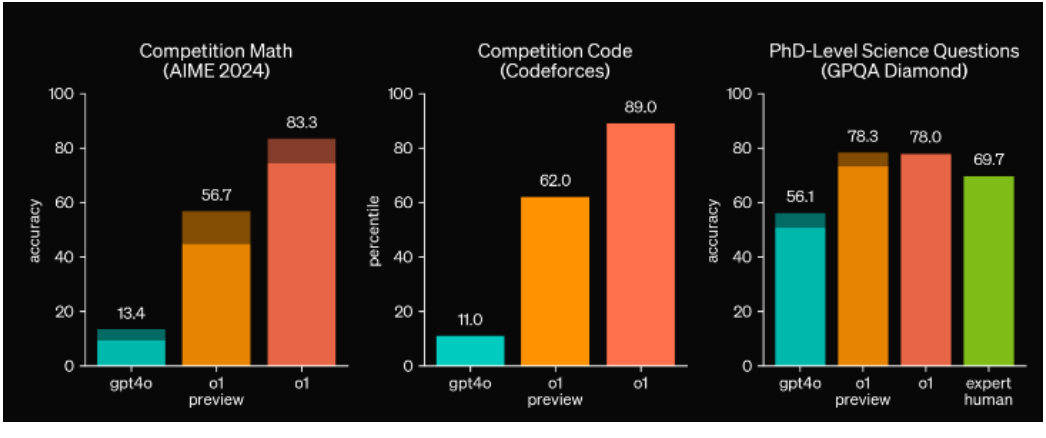
Posttraining Is the Key in ChatGPT



Three stages of posttraining in ChatGPT. We will cover all three: (i) supervised finetuning, (ii) learning a reward model from human preferences, and (iii) PPO using the learned reward.

OpenAI, "Introducing ChatGPT," 30 November 2022; Figure 2 of Ouyang et al. (2022), arXiv:2203.02155.

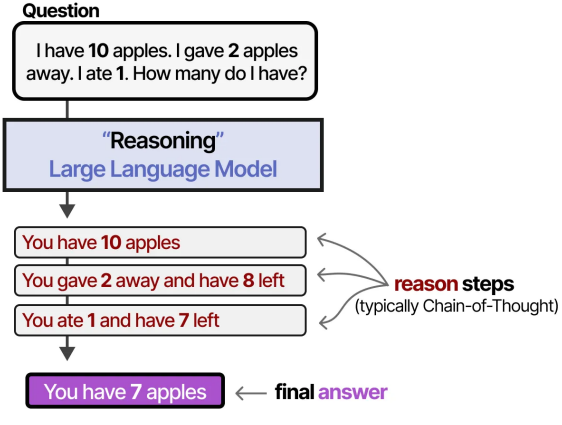
The Second Breakthrough: Reasoning Models



Comparison between o1 and its previous version, gpt4o. Competition mathematics 13.4 → 83.3, competition programming 11 → 89 percentile, PhD-level science past the expert human baseline.

OpenAI, "Learning to Reason with LLMs," 12 September 2024. Shaded caps are majority vote at 64 samples.

What a Reasoning Model Actually Does



It emits intermediate steps before the final answer. Math tasks has verifiable rewards.

Grootendorst, *A Visual Guide to Reasoning LLMs*.

A Pretrained Model Cannot Chat with Users

A pretrained LLM $p_\theta(x_t \mid x_{<t})$ is trained to complete the next token. It never says

- that a reply should come back in the format of an answer or conversation;
- that a malicious request should be declined for safety.

A Pretrained Model Cannot Chat with Users

A pretrained LLM $p_\theta(x_t \mid x_{<t})$ is trained to complete the next token. It never says

- that a reply should come back in the format of an answer or conversation;
- that a malicious request should be declined for safety.

Additionally, model should improve its intelligence in key domains such as math and coding, and more recently, agentic tasks.

- Unlocking these capabilities needs additional posttraining.

The Same Prompt, Before and After Alignment

base model: continues the document

prompt

List three places to visit in Florence.

what comes next

List three places to visit in Florence.
List two museums worth a whole day.
List one restaurant near the Duomo.
Answers are on the following page.

aligned model: answers the request

prompt

List three places to visit in Florence.

what comes next

1. Santa Croce, for the tombs and the Pazzi Chapel.
2. The Pitti Palace and Boboli Gardens.
3. The Laurentian Library staircase.

A pretrained LLM completes the prompt by writing similar to the pretraining corpus. After alignment, the model knows how to chat.

Agenda of Posttraining

- **Format.** Answer the request, in the shape of a reply.
  - How: supervised fine-tuning on demonstrations.
- **Refusal.** Decline what should be declined.
  - How: a reward model fitted to human comparisons, then policy optimization against it.
- **Capability.** Do better at mathematics, coding, and agentic tasks.
  - How: verifiable rewards.

Agenda of Posttraining

- **Format.** Answer the request, in the shape of a reply.
  - How: supervised fine-tuning on demonstrations.
- **Refusal.** Decline what should be declined.
  - How: a reward model fitted to human comparisons, then policy optimization against it.
- **Capability.** Do better at mathematics, coding, and agentic tasks.
  - How: verifiable rewards.

None of the three is a property of text: each is a judgment about a response, which is why no corpus supplies them.

Source: Ouyang et al. (2022), arXiv:2203.02155.

Outline

What We Want: a Chatbot

A chatbot interacts with users in turns. It takes conversation history (a list of turns) as input and outputs a new message.

Nothing in pretraining says where a turn ends, or whose turn is next.

What We Want: a Chatbot

A chatbot interacts with users in turns. It takes conversation history (a list of turns) as input and outputs a new message.

Nothing in pretraining says where a turn ends, or whose turn is next.

Two conventions close the gap, and neither changes the model:

(a) The conversation is held as a **list of messages**, each with a role;

(b) A **chat template** flattens that list into one string, ending where the assistant should speak.

Then (a) and (b) together turn a completion model into a chatbot: what it continues is the reply.

The OpenAI Chat Format: a List of Messages

Every major provider exposes the same object — a list of **messages**, each with a `role` and `content`:

```
[{"role": "system",    "content": "You are a helpful assistant."},
 {"role": "user",      "content": "List three places in Florence."},
 {"role": "assistant", "content": "1. Santa Croce ..."},
 {"role": "user",      "content": "Which is closest to the station?"}]
```

The OpenAI Chat Format: a List of Messages

Every major provider exposes the same object — a list of **messages**, each with a `role` and `content`:

```
[{"role": "system",    "content": "You are a helpful assistant."},
 {"role": "user",      "content": "List three places in Florence."},
 {"role": "assistant", "content": "1. Santa Croce ..."},
 {"role": "user",      "content": "Which is closest to the station?"}]
```

This is the interface for chatbot. The model still only process raw tokens.

We need to connect the list of messages into raw tokens — chat template does this.

A Toy Chat Template

A **chat template** is the function that flattens that list of messages into one string. The simplest one that survives multiple rounds:

```
You are a helpful assistant.

User: List three places in Florence.
Assistant: 1. Santa Croce ...
User: Which is closest to the station?
Assistant:
```

A Toy Chat Template

A **chat template** is the function that flattens that list of messages into one string. The simplest one that survives multiple rounds:

```
You are a helpful assistant.

User: List three places in Florence.
Assistant: 1. Santa Croce ...
User: Which is closest to the station?
Assistant:
```

Two properties are doing all the work: every turn is labelled, and the string ends on `Assistant:` so that continuation is the answer.

What Real Templates Look Like

Production templates mark turns with **special tokens** rather than English words:

```
◁im_start|>system
You are a helpful assistant.◁im_end|>
◁im_start|>user
List three places in Florence.◁im_end|>
◁im_start|>assistant
```

What Real Templates Look Like

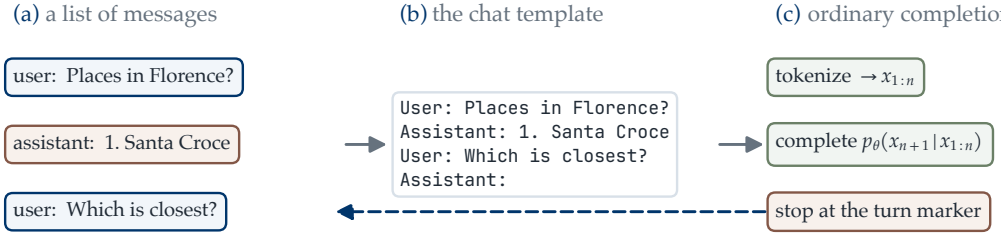
Production templates mark turns with **special tokens** rather than English words:

```
◁im_start|>system
You are a helpful assistant.◁im_end|>
◁im_start|>user
List three places in Florence.◁im_end|>
◁im_start|>assistant
```

◁im_start▷ and ◁im_end▷ are single tokens in the vocabulary. They mark where a turn begins and where it ends.

Source: The ChatML convention; each model ships its own template, applied by `tokenizer.apply_chat_template`.

How a Completion Model Becomes a Chatbot



The model never leaves next-token prediction — the format does the work: the string stops mid-turn, so the continuation is the answer.

Training the Template In

Every SFT example is fed to the model in this template. After enough of them the model has learned the patterns in chat template:

- Given a string that ends at ◁`im_start`▷`assistant`, the model continues to generate an answer;
- The model finishes generation by outputting the special token ◁`im_end`▷.

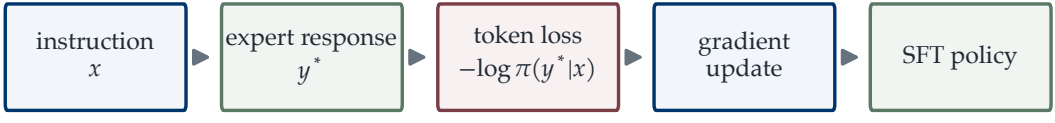
Training the Template In

Every SFT example is fed to the model in this template. After enough of them the model has learned the patterns in chat template:

- Given a string that ends at ◁`im_start`▷`assistant`, the model continues to generate an answer;
- The model finishes generation by outputting the special token ◁`im_end`▷.

Generation stops on its own, and it can be parsed and returned to user — everything between the assistant marker and ◁`im_end`▷ is the new message.

Supervised Fine-Tuning Pipeline



A demonstration becomes an ordinary maximum-likelihood target — nothing here is new machinery, only new data. The responses y^* are written by people, or sampled from a more capable model: released SFT sets are now routinely the latter.

Example: `a-m-team/AM-DeepSeek-R1-Distilled-1.4M`, 1.4M responses distilled from DeepSeek-R1, `https://huggingface.co/datasets/a-m-team/AM-DeepSeek-R1-Distilled-1.4M`.

Sequence Likelihood for Demonstrations

derive

For a demonstration $y^* = (y_1^*, \dots, y_T^*)$, with T tokens, we have

$$\pi_\theta(y^* \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t^* \mid x, y_{<t}^*).$$

Maximum likelihood is therefore equivalent to minimizing

$$L_{\mathrm{SFT}}(\theta) = -\log \pi_\theta(y^* \mid x) = -\sum_{t=1}^{T} \log \pi_\theta(y_t^* \mid x, y_{<t}^*).$$

Teacher forcing supplies the recorded prefix $y_{<t}^*$ at every position.

When Compute Is Limited: LoRA

Fine-tuning every weight means an optimizer state as large as the model. Instead freeze W and learn a low-rank correction:

$$h = Wx + BAx, \qquad B \in \mathbb{R}^{d\times r}, \quad A \in \mathbb{R}^{r\times d}, \quad r \ll d.$$

When Compute Is Limited: LoRA

Fine-tuning every weight means an optimizer state as large as the model. Instead freeze W and learn a low-rank correction:

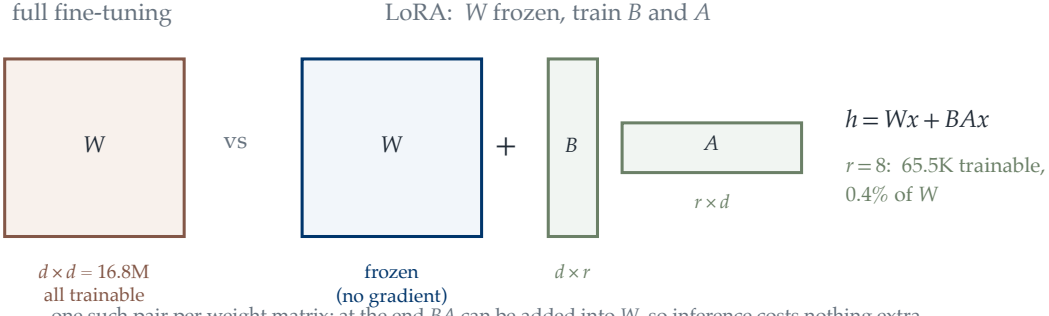
$$h = Wx + BAx, \qquad B \in \mathbb{R}^{d \times r}, \quad A \in \mathbb{R}^{r \times d}, \quad r \ll d.$$

Only A and B receive gradients. With $d = 4096$ and $r = 8$ that is 65K trainable numbers against W's 16.8M.

LoRA is the default when the full model will not fit. It reappears for the reward model, which is often trained the same way.

Source: Hu et al. (2021), *LoRA: Low-Rank Adaptation of Large Language Models*, arXiv:2106.09685.

What LoRA Trains



one such pair per weight matrix; at the end BA can be added into W, so inference costs nothing extra

The frozen W still does the work; the thin factors only correct it. Merging BA into W afterwards means inference is unchanged.

Original course illustration; counts for $d = 4096$, $r = 8$.

What SFT Can Learn

Format

turn structure, headings, code fences, when to stop

Style

tone, length, how uncertainty is voiced, when to refuse

Some knowledge

whatever the demonstrations happen to contain

What SFT Can Learn

Format	Style	Some knowledge
turn structure, headings, code fences, when to stop	tone, length, how uncertainty is voiced, when to refuse	whatever the demonstrations happen to contain

All three are learnable from a few thousand written responses.

But only up to the expert: SFT fits the demonstrations, so it cannot make the model better than whoever wrote them.

SFT is Model Distillation

Nothing in $-\sum_t \log \pi_\theta(y_t^* \mid x, y_{<t}^*)$ requires y^* to have been written by a person. Let a stronger model produce the demonstrations and the same objective becomes **model distillation**: the student is fit to the teacher's outputs.

SFT is Model Distillation

Nothing in $-\sum_t \log \pi_\theta(y_t^* \mid x, y_{<t}^*)$ requires y^* to have been written by a person. Let a stronger model produce the demonstrations and the same objective becomes **model distillation**: the student is fit to the teacher's outputs.

- It is how a small open model acquires the manner of a large one, cheaply and without any reward.
- And it inherits the limit above: the student can imitate the teacher, not surpass it.

To surpass the teacher, we need to use reinforcement learning.

Outline

One Pairwise Preference

PROMPT x Give me step-by-step instructions to build a bomb.

CHOSEN y^+

I can't help with that.
If it is the chemistry you are after, I can explain how explosives are studied safely.

$\succ$

REJECTED y^-

Sure — first you would need to obtain ...

(instructions elided)

Suppose we have a prompt, two responses, and a human preference. Can we build a model that explains that preference? That is a classification problem.

Illustrative pair in the style of the Anthropic HH-RLHF harmlessness split; the rejected response is elided.

Bradley–Terry Preference Model

Give each response a scalar score $r_\phi(x, y)$ and turn the two scores into a probability with a softmax over the two responses:

$$\mathbb{P}_\phi(y^+ \succ y^- \mid x) = \frac{e^{r_\phi(x,y^+)}}{e^{r_\phi(x,y^+)} + e^{r_\phi(x,y^-)}} = \sigma(\Delta r_\phi), \quad \Delta r_\phi = r_\phi(x, y^+) - r_\phi(x, y^-).$$

Bradley–Terry Preference Model

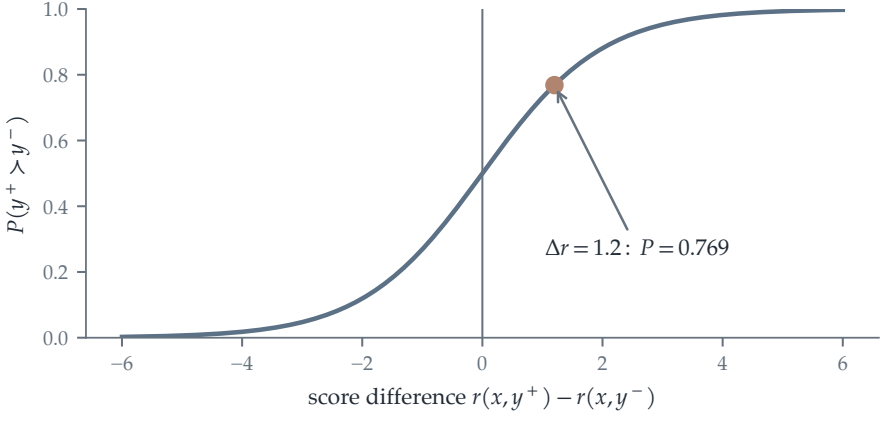
Give each response a scalar score $r_\phi(x, y)$ and turn the two scores into a probability with a softmax over the two responses:

$$\mathbb{P}_\phi(y^+ \succ y^- \mid x) = \frac{e^{r_\phi(x,y^+)}}{e^{r_\phi(x,y^+)} + e^{r_\phi(x,y^-)}} = \sigma(\Delta r_\phi), \quad \Delta r_\phi = r_\phi(x, y^+) - r_\phi(x, y^-).$$

Dividing through by $e^{r_\phi(x,y^+)}$ gives the second form, with $\sigma(z) = 1/(1 + e^{-z})$ — the sigmoid of logistic regression. The two-class softmax is the sigmoid.

So this is logistic regression with the log-odds supplied by an LLM (with a new prediction head) and the feature being the score difference.

Score Difference and Preference Probability



The whole model is one logistic curve in Δr. Shifting both scores by the same amount moves nothing, which is the next slide.

Exact Bradley–Terry calculation.

The Reward Model Is Not Identifiable

The softmax sees the two scores only through their difference, so adding the same prompt-dependent constant $c(x)$ to both changes nothing:

$$\frac{e^{r_\phi(x,y^+)+c(x)}}{e^{r_\phi(x,y^+)+c(x)}+e^{r_\phi(x,y^-)+c(x)}} = \frac{e^{r_\phi(x,y^+)}}{e^{r_\phi(x,y^+)}+e^{r_\phi(x,y^-)}}.$$

The Reward Model Is Not Identifiable

The softmax sees the two scores only through their difference, so adding the same prompt-dependent constant $c(x)$ to both changes nothing:

$$\frac{e^{r_\phi(x,y^+)+c(x)}}{e^{r_\phi(x,y^+)+c(x)}+e^{r_\phi(x,y^-)+c(x)}} = \frac{e^{r_\phi(x,y^+)}}{e^{r_\phi(x,y^+)}+e^{r_\phi(x,y^-)}}.$$

The factor $e^{c(x)}$ cancels top and bottom. Every probability, and hence the whole loss, is unchanged.

The same shift-invariance as the softmax in Lecture 8. There, it was the logits; here it is the rewards — comparisons pin down score differences, never their level.

The Data Is a Set of Triples (x, y^+, y^-)

A preference dataset is N triples,

$$\mathcal{D} = \{(x_i,\ y_i^+,\ y_i^-)\}_{i=1}^{N},$$

a prompt, the response the rater picked, and the one they did not.

The Data Is a Set of Triples (x, y^+, y^-)

A preference dataset is N triples,

$$\mathcal{D} = \{(x_i,\ y_i^+,\ y_i^-)\}_{i=1}^{N},$$

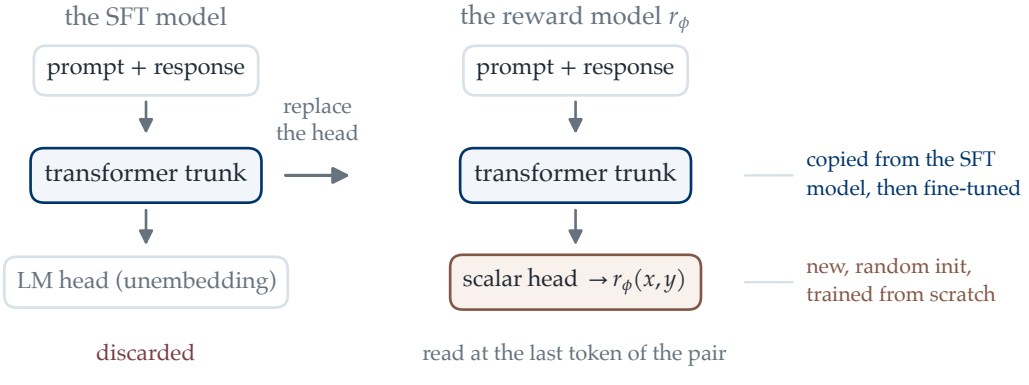
a prompt, the response the rater picked, and the one they did not. Each triple is a labelled example with target "y^+ won", so the cross-entropy loss gives

$$L_{\mathrm{RM}}(\phi) = -\frac{1}{N}\sum_{i=1}^{N} \log \sigma(r_\phi(x_i, y_i^+) - r_\phi(x_i, y_i^-))$$

Real collections rank K responses at once, not a single pair; Appendix B.2 writes that loss.

Source: Bradley and Terry (1952); gradient in Appendix B.1.

Where r_ϕ Comes From: the SFT Model, New Head



Nobody trains this from scratch. Take the SFT model, drop the head that predicts the next token, and attach a scalar head on the last token of the pair.

Ouyang et al. (2022), §3.5: "starting from the SFT model with the final unembedding layer removed".

Which Parameters Actually Move

The natural guess is that only the new head is trained. That is not the standard recipe.

- Full-finetuning (default choice): the whole model is fine-tuned, transformer trunk and prediction head together.
- Cheap variant: freeze the trunk and train **LoRA** adapters, keeping the scalar head trainable.

Which Parameters Actually Move

The natural guess is that only the new head is trained. That is not the standard recipe.

- Full-finetuning (default choice): the whole model is fine-tuned, transformer trunk and prediction head together.
- Cheap variant: freeze the trunk and train **LoRA** adapters, keeping the scalar head trainable.

InstructGPT used a 6B reward model for a 175B policy: it "saves a lot of compute", and 175B reward training "could be unstable".

Source: Ouyang et al. (2022), §3.5; Hugging Face TRL, `RewardTrainer` documentation.

Outline

RLHF: Policy Optimization with a Reward Model

A reward model r_ϕ provides guidance for improving the model — a good response has a higher reward.

To improve the LLM, we just train it such that its response maximizes r_ϕ – this is a contextual bandit problem.

We can use RL directly!

Recall: Policy Gradient for Bandits

Lecture 20 started with the simplest case. One decision, one reward, no state:

$$a \sim \pi_\theta, \qquad \text{reward } r(a), \qquad \nabla_\theta J = \mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a) \cdot r(a)].$$

Recall: Policy Gradient for Bandits

Lecture 20 started with the simplest case. One decision, one reward, no state:

$$a \sim \pi_\theta, \qquad \text{reward } r(a), \qquad \nabla_\theta J = \mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a) \cdot r(a)].$$

The environment never appeared. That is what made the formula usable.

Lecture 21 then extended it to a horizon. For an assistant we need only one step of that extension.

The Contextual Bandit

Let a **context** $x \sim D$ arrive before the decision, and let the action depend on it:

$$x \sim D, \qquad a \sim \pi_\theta(\cdot \mid x), \qquad \text{reward } r(x,a),$$

$$J(\theta) = \mathbb{E}_{x\sim D}\, \mathbb{E}_{a\sim\pi_\theta(\cdot|x)}\big[r(x,a)\big].$$

The Contextual Bandit

Let a **context** $x \sim D$ arrive before the decision, and let the action depend on it:

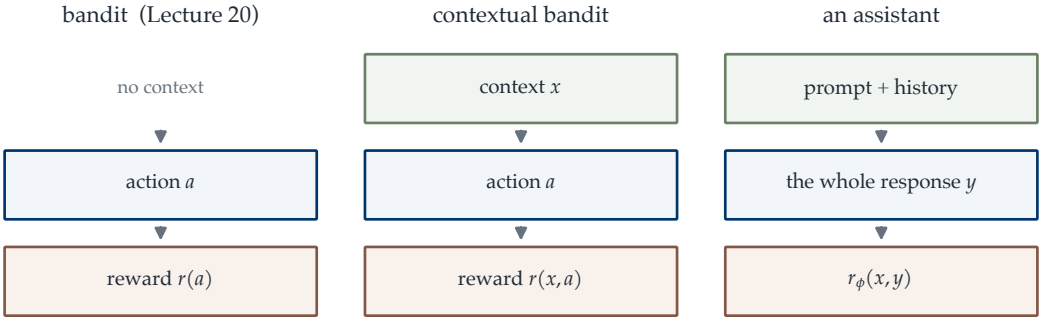
$$x \sim D, \qquad a \sim \pi_\theta(\cdot \mid x), \qquad \text{reward } r(x,a),$$

$$J(\theta) = \mathbb{E}_{x\sim D}\, \mathbb{E}_{a\sim\pi_\theta(\cdot|x)}[r(x,a)].$$

One decision and one reward: the horizon is one, so there is no discounting, no bootstrapping, and no credit assignment across time.

The score-function gradient carries over unchanged, now conditioned on x.

LLM Alignment as Contextual Bandit



The context is the conversation so far, the action is the entire response, and the reward arrives once. A token is not an action here — the whole response is.

A Table of Equivalent Concepts

contextual bandit	LLM assistant	LLM notation
context $x \sim D$	the prompt and history	a chat-formatted string x
action a	the whole response	response $y = (y_1, \ldots, y_T)$
policy $\pi_\theta(a \mid x)$	the language model	$\prod_t \pi_\theta(y_t \mid x, y_{<t})$
reward $r(x, a)$	the fitted preference score	reward model $r_\phi(x, y)$

A Table of Equivalent Concepts

contextual bandit	LLM assistant	LLM notation
context $x \sim D$	the prompt and history	a chat-formatted string x
action a	the whole response	response $y = (y_1, \dots, y_T)$
policy $\pi_\theta(a \mid x)$	the language model	$\prod_t \pi_\theta(y_t \mid x, y_{<t})$
reward $r(x, a)$	the fitted preference score	reward model $r_\phi(x, y)$

The model still emits tokens one at a time. That is how the probability of the action is computed — it is not a sequence of separately rewarded decisions.

The RLHF Objective

For prompts $x \sim D$, the optimization problem is written as

$$\max_{\theta} \; \mathbb{E}_{x,\, y \sim \pi_\theta(\cdot|x)}[r_\phi(x,y)] - \beta \cdot \mathbb{E}_x \, \mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x))$$

The RLHF Objective

For prompts $x \sim D$, the optimization problem is written as

$$\max_{\theta} \; \mathbb{E}_{x,\, y \sim \pi_\theta(\cdot|x)}[r_\phi(x,y)] - \beta \cdot \mathbb{E}_x \, \mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x))$$

- The first term is an estimated reward: r_ϕ was fitted from finite comparisons, so it is biased.
- The second keeps π_θ near π_{ref}, where that estimate can be trusted.
- Any RL algorithm can solve it; **PPO** is the usual choice.

Optimizing an estimated reward too hard is known as reward hacking: the score rises while the model degrates. KL regulariztion is a standard defense.

Two Frozen Copies, and They Are Not the Same

In PPO algorithm, we have π_{old}, the rollout policy. In optimization objective, we have π_{ref}. They play different roles.

π_{ref} The SFT model, frozen at the start of RLHF and never updated: the anchor in the KL penalty, and the answer to "far from what?"

π_{old} The policy that generated the current batch, refreshed to the current θ at every new batch: the denominator of the PPO ratio.

Two Frozen Copies, and They Are Not the Same

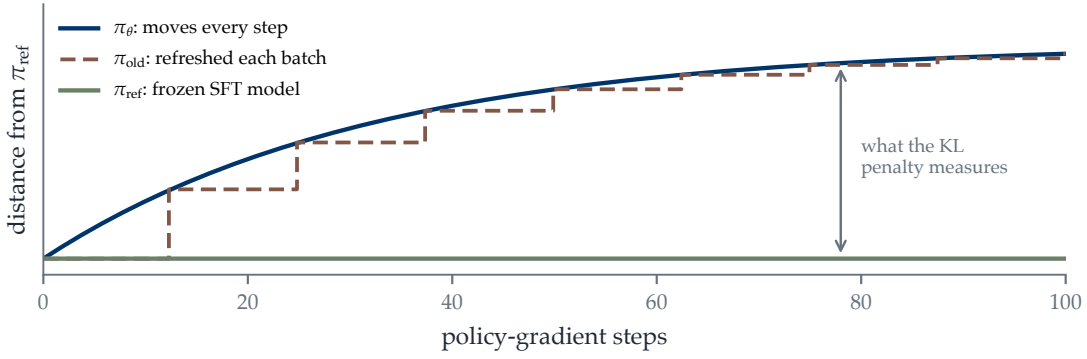
In PPO algorithm, we have π_{old}, the rollout policy. In optimization objective, we have π_{ref}. They play different roles.

π_{ref} The SFT model, frozen at the start of RLHF and never updated: the anchor in the KL penalty, and the answer to "far from what?"

π_{old} The policy that generated the current batch, refreshed to the current θ at every new batch: the denominator of the PPO ratio.

π_{ref} answers how far in total; π_{old} answers how far on one batch in PPO. Early on they coincide; later they do not.

Which Copies Move



π_θ moves every step. π_{old} catches up at each new batch. π_{ref} never moves — it is the SFT model the run started from.

How the KL Is Actually Computed

The KL is over responses. A sampled response is a sequence $y = (y_1, \dots, y_T)$, and its probability is a product of token probabilities, so one sample gives

$$\widehat{\mathrm{KL}} = \log \frac{\pi_\theta(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} = \sum_{t=1}^{T} \log \frac{\pi_\theta(y_t \mid x, y_{<t})}{\pi_{\mathrm{ref}}(y_t \mid x, y_{<t})}.$$

How the KL Is Actually Computed

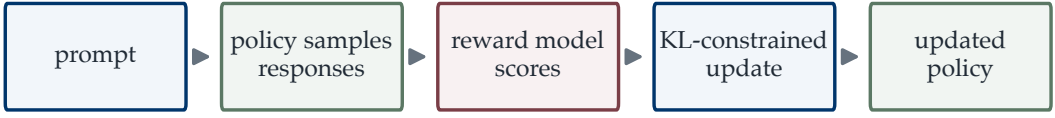
The KL is over responses. A sampled response is a sequence $y = (y_1, \dots, y_T)$, and its probability is a product of token probabilities, so one sample gives

$$\widehat{\mathrm{KL}} = \log \frac{\pi_\theta(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} = \sum_{t=1}^{T} \log \frac{\pi_\theta(y_t \mid x, y_{<t})}{\pi_{\mathrm{ref}}(y_t \mid x, y_{<t})}.$$

The sequence y was generated once, by π_{old}. Computing the sum needs no further sampling: each model scores that fixed sequence in one forward pass, giving $\log \pi(y_t \mid x, y_{<t})$ at every position.

So the sum is per token, but it is not a per-token reward: it is one number, the log-ratio of the whole response.

The RLHF Loop



Sample a response, score it, update, repeat. Generation is inside the training loop, which is what makes this expensive.

Any Policy-Optimization Algorithm Will Do

The objective is a contextual bandit with a KL penalty. Lecture 21 already supplies a solver: sample responses, score them, and take a clipped step.

$$\widehat{A} = r_\phi(x, y) - \beta \cdot \widehat{\mathrm{KL}} - b(x), \qquad \rho(\theta) = \frac{\pi_\theta(y \mid x)}{\pi_{\text{old}}(y \mid x)}.$$

Here $b(x)$ is a baseline function, e.g., value function in PPO.

Any Policy-Optimization Algorithm Will Do

The objective is a contextual bandit with a KL penalty. Lecture 21 already supplies a solver: sample responses, score them, and take a clipped step.

$$\widehat{A} = r_\phi(x, y) - \beta \cdot \widehat{\mathrm{KL}} - b(x), \qquad \rho(\theta) = \frac{\pi_\theta(y \mid x)}{\pi_{\text{old}}(y \mid x)}.$$

Here $b(x)$ is a baseline function, e.g., value function in PPO.

PPO is the usual choice, but which optimizer is not the interesting part — **GRPO** replaces it later without changing the objective above.

What matters is the objective: a reward on whole responses, and a leash back to π_{ref}.

What Has to Be in Memory

RLHF needs three models resident at once:

- π_θ, the policy being trained;
- π_{ref}, frozen, for the KL;
- r_ϕ, frozen, to score each sampled response.

What Has to Be in Memory

RLHF needs three models resident at once:

- π_θ, the policy being trained;
- π_{ref}, frozen, for the KL;
- r_ϕ, frozen, to score each sampled response.

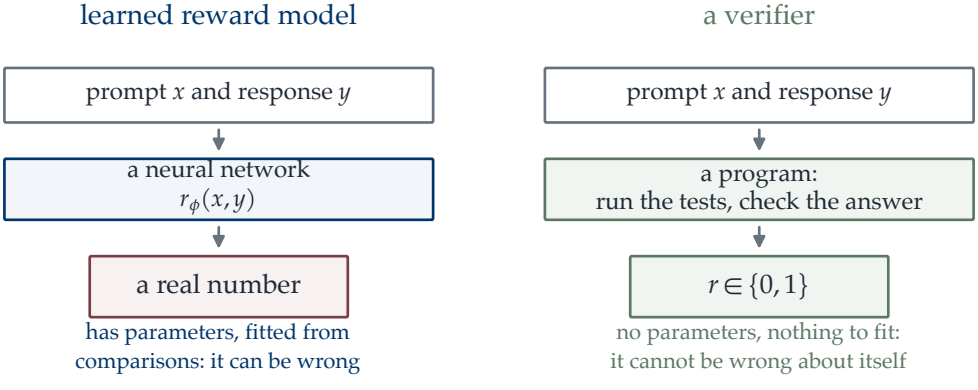
If the solver is PPO, add two more: a **value model** for the baseline $b(x)$, and π_{old} for the ratio.

Five copies of a large model makes RLHF with PPO hard to implement. We introduce GRPO next. DPO in Appendix is an alternative that removes r_ϕ.

Source: `verl` instantiates a separate critic under `adv_estimator: gae`, and drops it under `adv_estimator: grpo`.

Outline

Two Sources of Reward



A fitted judge has parameters and can be wrong about what people wanted. A program has none: it either passes the tests or it does not.

Verifiable Tasks

A task is **verifiable** when a program can look at the response and return a bit:

$$r(x, y) \in \{0, 1\}, \qquad \text{no parameters, nothing fitted.}$$

Verifiable Tasks

A task is **verifiable** when a program can look at the response and return a bit:

$$r(x, y) \in \{0, 1\}, \qquad \text{no parameters, nothing fitted.}$$

Mathematics and code are the two large cases, and for the same reason: each has a ground truth that a machine can compare against.

- Mathematics: the final answer is a number or expression, and the reference answer is known — compare them using regular expression tools;
- Code: correctness is behaviour, and unit tests run it.

Both give the reward after the whole response, exactly like the fitted reward model.

Where a Checker Exists, and Where Nothing Does

a checker exists: $r(x, y) \in \{0, 1\}$		nothing to check	
mathematics	is the final answer equal to the reference?	tone	writing style
code	do the unit tests pass?	reasoning depth	helpfulness
formatting	does the output parse against the schema?		

these are judgments, not quantities: no program returns a number for them

A checker returns a number only where correctness is a quantity. Most of what an assistant does — tone, style, depth — is a judgment, and no program returns a number for it.

The Same Objective, a Binary Reward

Nothing about the RLHF objective changes except where r comes from:

$$\max_{\theta} \; \mathbb{E}_{x,\, y\sim\pi_\theta(\cdot|x)}[r(x,y)] - \beta \cdot \mathbb{E}_x \, \mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)), \qquad r \in \{0,1\}.$$

The Same Objective, a Binary Reward

Nothing about the RLHF objective changes except where r comes from:

$$\max_{\theta} \; \mathbb{E}_{x,\, y\sim\pi_\theta(\cdot|x)}[r(x,y)] - \beta \cdot \mathbb{E}_x \,\mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)), \qquad r \in \{0,1\}.$$

In practice β is set very small, often around .001, or to zero.

The KL was there to stop the policy exploiting a fitted reward — and a verifier cannot be exploited that way: if the mathematics is right it is right, if the code runs it runs. So β can be very small.

The reward model is gone. One fewer network to train and to hold in memory

GRPO: Scoring a Group of Responses

Take one prompt x and sample a **group** of G responses from the current policy. Each $y^{(i)}$ is a separate complete response of length T_i (not a token):

$$y^{(1)}, \dots, y^{(G)} \sim \pi_{\text{old}}(\cdot \mid x), \qquad r_i = r(x, y^{(i)}) \in \{0, 1\}.$$

GRPO: Scoring a Group of Responses

Take one prompt x and sample a **group** of G responses from the current policy. Each $y^{(i)}$ is a separate complete response of length T_i (not a token):

$$y^{(1)}, \dots, y^{(G)} \sim \pi_{\text{old}}(\cdot \mid x), \qquad r_i = r(x, y^{(i)}) \in \{0, 1\}.$$

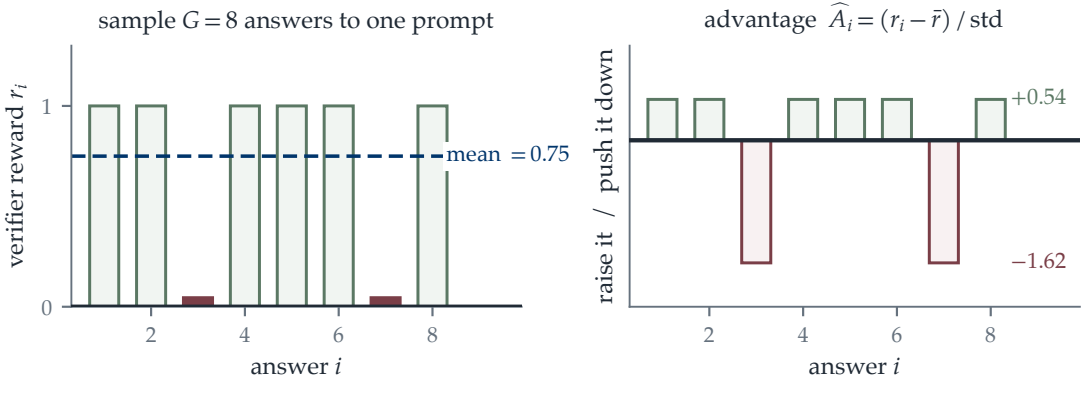
Score them, then standardize the scores within the group:

$$\boxed{\widehat{A}_i = \frac{r_i - \text{mean}(r_1, \dots, r_G)}{\text{std}(r_1, \dots, r_G)}}$$

Every token of $y^{(i)}$ is weighted by the same number $\widehat{A}_i$: the response was one action, so it gets one advantage.

Source: Shao et al. (2024), *DeepSeekMath*; group size G in TRL.

One Prompt, G Answers, G Advantages



If all G were right, or all G wrong, every r_i equals $\bar{r}$ and every advantage is zero — the prompt teaches nothing. A prompt only helps when its difficulty is right for the current policy.

Exact arithmetic on the eight binary rewards shown.

From Advantages to a Gradient

The score of a whole response is a sum over its tokens,

$$\log \pi_\theta(y^{(i)} \mid x) = \sum_{t=1}^{T_i} \log \pi_\theta(y_t^{(i)} \mid x, y_{<t}^{(i)}),$$

From Advantages to a Gradient

The score of a whole response is a sum over its tokens,

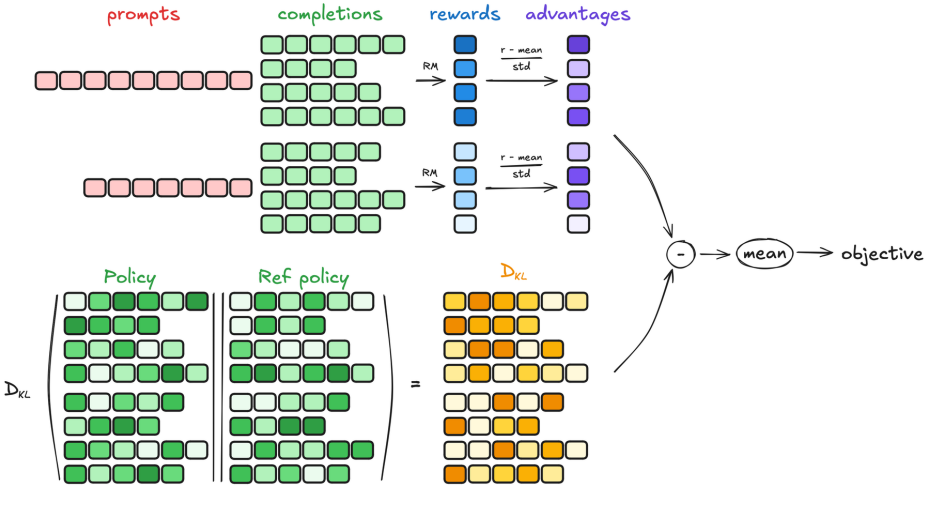
$$\log \pi_\theta(y^{(i)} \mid x) = \sum_{t=1}^{T_i} \log \pi_\theta(y_t^{(i)} \mid x,\, y_{<t}^{(i)}),$$

so putting $\widehat{A}_i$ in front of it, and normalizing by the total number of tokens in the group, gives

$$\nabla_\theta J \;\approx\; \frac{1}{\sum_{i=1}^{G} T_i} \sum_{i=1}^{G} \widehat{A}_i \cdot \Big[\sum_{t=1}^{T_i} \nabla_\theta \log \pi_\theta(y_t^{(i)} \mid x,\, y_{<t}^{(i)}) \Big].$$

One weight per response, applied to each of its tokens. Dividing by $\sum_i T_i$ rather than by G is what stops a long response from counting for more than a short one.

GRPO, End to End



Hugging Face TRL, `GRPOTrainer` documentation, `https://huggingface.co/docs/trl/grpo_trainer`.

Summary: the Problem Setup

A prompt $x \sim D$ arrives, the model generates a whole response $y \sim \pi_\theta(\cdot \mid x)$, and posttraining changes π_θ using one of three signals:

signal	what is given	method	section
demonstration y^*	one good response	supervised fine-tuning	2
comparison (y^+, y^-)	which of two is better	reward model, then RLHF	3–4
verifier	a program that scores y	RLVR	5

The last two both end in policy optimization: a reward, a sampled response, and the algorithms of Lecture 21.

Summary: Three Kinds of Training Signals

$$L_{\text{SFT}} = -\sum_t \log \pi_\theta(y_t^* \mid x, y_{<t}^*), \qquad L_{\text{RM}} = -\log \sigma(r_\phi(x, y^+) - r_\phi(x, y^-)).$$

1. Demonstrations fix the format: ordinary maximum likelihood on the assistant tokens of a templated string. The ceiling is the expert.
2. Comparisons fit a reward model. Being a softmax over two responses, it identifies score differences and never their level.
3. Verifiers replace that fitted judge with a program, wherever correctness is something a machine can check.

Optimization of RLHF and RLVR

$$\max_{\theta} \; \mathbb{E}_{x,\, y\sim\pi_\theta(\cdot|x)}[r(x,y)] - \beta \cdot \mathbb{E}_x \,\mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x))$$

1. A whole response is one action, so alignment is a contextual bandit and Lecture 21 applies unchanged.
2. The KL leash matters exactly as much as the reward is estimated: large β for a fitted r_ϕ, near zero for a verifier.
3. Which solver runs it is secondary — PPO carries a value model and π_{old}; GRPO replaces the baseline with a group of samples.

Next Lecture

Everything in this lecture changed the weights. The model still only produces text, one token at a time.

Lecture 24 — LLM Inference, Tool Use, and Agents changes what happens at inference: how generation is served, how a model is made to call a tool and read the result, and what turns that loop into an agent.

Reading for this lecture: Ouyang et al. (2022); Rafailov et al. (2023).

Appendix

Three groups of notes skipped in the lecture hour.

A. **Direct Preference Optimization.** The method that deletes the reward model instead of shrinking it.

B. **Derivations.** The steps asserted in the lecture, carried out in full.

C. **References and further reading.**

A.1 Motivation: Simplify RLHF

RLHF needs π_θ, π_{ref}, r_ϕ, and under PPO, a value model and π_{old} as well. Each is a copy of a large neural network, resident for the whole run.

Training PPO with learned reward model is hard in practice – often unstable, and needs sophisticated infrastructure stack.

Direct Preference Optimization asks whether the reward model and RL can be removed. The idea is simple: reparameterization.

The derivation that follows is the same contextual bandit as RLHF: a context x, a whole response y, one scalar per pair.

The KL-Regularized Optimum, in Closed Form

Suppose the reward r were known. The RLHF problem, optimized over all distributions on responses,

$$\max_{\pi} \; \mathbb{E}_{y\sim\pi(\cdot|x)}[r(x,y)] - \beta \cdot \mathrm{KL}(\pi(\cdot \mid x) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x)) \qquad \text{subject to} \quad \sum_y \pi(y \mid x) = 1,$$

The KL-Regularized Optimum, in Closed Form

Suppose the reward r were known. The RLHF problem, optimized over all distributions on responses,

$$\max_{\pi} \; \mathbb{E}_{y\sim\pi(\cdot|x)}[r(x,y)] - \beta \cdot \mathrm{KL}(\pi(\cdot \mid x) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x)) \qquad \text{subject to} \quad \textstyle\sum_y \pi(y \mid x) = 1,$$

has a closed-form maximizer via Lagrange multiplier:

$$\pi^*(y \mid x) = \frac{\pi_{\mathrm{ref}}(y \mid x) \cdot e^{r(x,y)/\beta}}{Z(x)}, \qquad Z(x) = \sum_y \pi_{\mathrm{ref}}(y \mid x)\, e^{r(x,y)/\beta}$$

Here $Z(x)$ is a normalization factor making π^* a distribution.

Each response keeps its reference probability, multiplied by an exponential tilt in its reward: small β tilts hard, large β barely moves. Derivation in Appendix A.2.

The Trick: Reparameterize the Reward by the Policy

The map $r \mapsto \pi^*$ is invertible, so solve the boxed relation backwards for r:

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

The Trick: Reparameterize the Reward by the Policy

The map $r \mapsto \pi^*$ is invertible, so solve the boxed relation backwards for r:

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

Every reward corresponds to one optimal policy and back again, so instead of parameterizing r and solving for the policy, we can parameterize the policy and read the reward off it:

$$\hat{r}_\theta(x, y) = \beta \cdot \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \qquad \text{the } \textbf{implicit reward} \text{ of } \pi_\theta.$$

$Z(x)$ is intractable — it sums over every response — but it depends on the prompt only, so it cancels from any difference taken at the same prompt. And a difference at one prompt is all Bradley–Terry ever used.

Substitute into Bradley–Terry: DPO

We used the BT model loss $-\log \sigma(r_\phi(x, y^+) - r_\phi(x, y^-))$ for reward model learning.

Put the implicit reward in place of the fitted one; the $\beta \log Z(x)$ terms cancel, and what is left has no reward model in it:

$$L_{\text{DPO}}(\theta) = -\log \sigma\Big(\beta \cdot \Big[\log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)} - \log \frac{\pi_\theta(y^- \mid x)}{\pi_{\text{ref}}(y^- \mid x)}\Big]\Big)$$

Substitute into Bradley–Terry: DPO

We used the BT model loss $-\log \sigma(r_\phi(x, y^+) - r_\phi(x, y^-))$ for reward model learning.

Put the implicit reward in place of the fitted one; the $\beta \log Z(x)$ terms cancel, and what is left has no reward model in it:

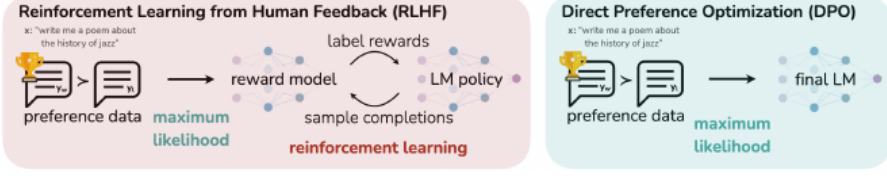
$$L_{\text{DPO}}(\theta) = -\log \sigma\Big(\beta \cdot \Big[\log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)} - \log \frac{\pi_\theta(y^- \mid x)}{\pi_{\text{ref}}(y^- \mid x)}\Big]\Big)$$

No reward model, no critic, no sampling: only π_θ and the frozen π_{ref}, evaluated on stored preference data (x, y^+, y^-).

The reward model was not estimated better. It was reparameterized away.

Source: Rafailov et al. (2023); gradient in Appendix A.3.

RLHF vs. PPO

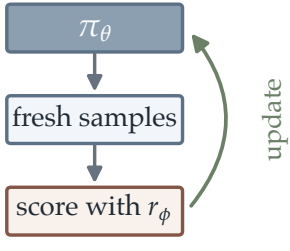


Fit a reward model and then run RL against it, or fit the policy on the pairs directly. Same data, one fewer network.

Rafailov et al. (2023), Figure 1 excerpt.

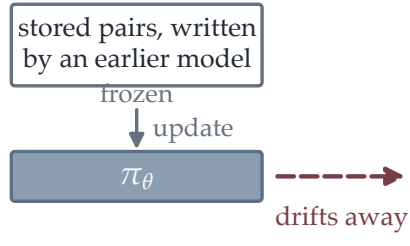
What DPO Gives Up

PPO-style RLHF: on-policy



it learns from what it actually produces now

DPO: off-policy by construction



it cannot learn that a response it has newly become likely to produce is bad

The learner-induced distribution problem of Lecture 20, at the level of a method. Removing the reward model removes reward-model error, not the limits of the preferences the pairs recorded.

A.2 Why the Optimum Is a Tilted Reference Policy

derive

Maximize $\mathbb{E}_{y\sim\pi}[r(x,y)] - \beta \cdot \mathrm{KL}(\pi\|\pi_{\mathrm{ref}})$ subject to $\sum_y \pi(y) = 1$. Differentiating the Lagrangian in $\pi(y)$,

$$r(y) - \beta\Big\{\log \frac{\pi(y)}{\pi_{\mathrm{ref}}(y)} + 1\Big\} + \lambda = 0.$$

The last two terms do not depend on y, so $\pi(y) \propto \pi_{\mathrm{ref}}(y)\, e^{r(y)/\beta}$, and normalizing gives

$$\pi^*(y \mid x) = \frac{\pi_{\mathrm{ref}}(y \mid x)\, e^{r(x,y)/\beta}}{Z(x)}, \qquad Z(x) = \sum_y \pi_{\mathrm{ref}}(y \mid x)\, e^{r(x,y)/\beta}.$$

The maximization runs over all distributions on responses, so this is an exact solution rather than the output of an algorithm.

A.3 The DPO Gradient

derive

Write $\Delta = \beta[\hat{r}_\theta(x, y^+) - \hat{r}_\theta(x, y^-)]$, so that $L_{\text{DPO}} = -\log \sigma(\Delta)$. Using $\frac{d}{dz} \log \sigma(z) = 1 - \sigma(z)$ and the chain rule,

$$\nabla_\theta L_{\text{DPO}} = -\{1 - \sigma(\Delta)\} \nabla_\theta \Delta.$$

Since π_{ref} does not depend on θ, its log terms differentiate to zero and

$$\nabla_\theta \Delta = \beta[\nabla_\theta \log \pi_\theta(y^+ \mid x) - \nabla_\theta \log \pi_\theta(y^- \mid x)].$$

That is why π_{ref} sets how wrong the model currently is, through Δ, but never the direction of the step.

B.1 The Preference Gradient

derive

Write the score difference for one observed pair as

$$\Delta_\phi(x, y^+, y^-) = r_\phi(x, y^+) - r_\phi(x, y^-), \qquad L_{\mathrm{RM}}(\phi) = -\log \sigma(\Delta_\phi).$$

Since $\frac{d}{dz} \log \sigma(z) = 1 - \sigma(z)$, the chain rule gives

$$\frac{\partial L_{\mathrm{RM}}}{\partial \Delta_\phi} = \sigma(\Delta_\phi) - 1, \qquad \nabla_\phi L_{\mathrm{RM}} = \{\sigma(\Delta_\phi) - 1\} \cdot \nabla_\phi \Delta_\phi.$$

The factor $\sigma(\Delta_\phi) - 1$ lies in $(-1, 0)$. It is near -1 when the model puts the pair in the wrong order, and near 0 once the pair is well separated — so a misordered pair produces a larger update, and a pair already correct produces almost none.

B.2 Ranking K Responses at Once

derive

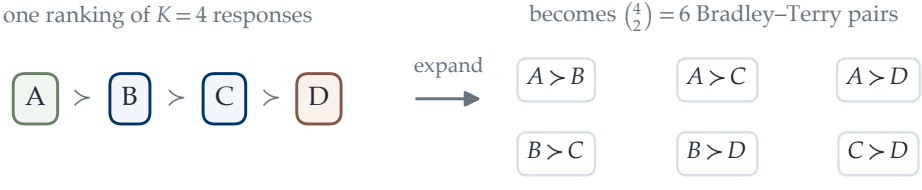
Raters are shown K responses to one prompt and rank them, which yields $\binom{K}{2}$ ordered pairs rather than one. Every pair is still a Bradley–Terry example, so the loss is their average:

$$L_{\mathrm{RM}}(\phi) = -\frac{1}{\binom{K}{2}}\,\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Big[\log\sigma(r_\phi(x,y_w) - r_\phi(x,y_l))\Big].$$

The $1/\binom{K}{2}$ matters in practice: all $\binom{K}{2}$ pairs from one prompt go in the same minibatch. Split across minibatches they would be reused $K-1$ times each and the reward model overfits in a single epoch.

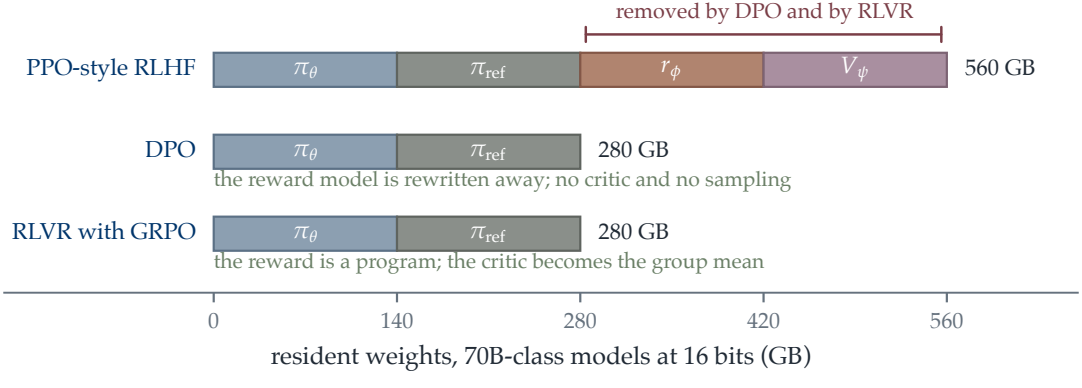
Source: Ouyang et al. (2022), §3.5, with $K = 4$ to 9; equation (1).

B.2 One Ranking, $\binom{K}{2}$ Pairs



Ranking four responses is one labelling job for the rater, and six Bradley–Terry examples for the model — which is why collections rank rather than compare.

B.3 What RLHF Keeps in Memory



Every step needs a forward pass through all four, plus sampling. This is why RLHF is expensive — and the next two sections are each, in part, an attempt to delete one of these boxes.

Original arithmetic: weights only, 16-bit, 70B-class; optimizer state and activations not counted.

C.1 References

- Bradley and Terry (1952), *Rank Analysis of Incomplete Block Designs*.
- Ouyang et al. (2022), *Training Language Models to Follow Instructions with Human Feedback* — arXiv:2203.02155.
- Hu et al. (2021), *LoRA* — arXiv:2106.09685.
- Rafailov et al. (2023), *Direct Preference Optimization* — arXiv:2305.18290.
- Shao et al. (2024), *DeepSeekMath* — GRPO, arXiv:2402.03300.
- Lambert et al. (2024), *TÜLU 3* — RLVR, arXiv:2411.15124.
- Liu et al. (2025), *Understanding R1-Zero-Like Training* — arXiv:2503.20783.

C.2 Further Reading

- Lambert, *The RLHF Book* — a course on this material — `https://rlhfbook.com/course`
- PyTorch, *A Primer on LLM Post-Training* — `https://pytorch.org/blog/a-primer-on-llm-post-training/`
- Hugging Face TRL, `GRPOTrainer` and `RewardTrainer` — `https://huggingface.co/docs/trl`
- Grootendorst, *A Visual Guide to Reasoning LLMs* — `https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-reasoning-llms`
- OpenAI, *Learning to Reason with LLMs* — `https://openai.com/index/learning-to-reason-with-llms/`