20

Post-training: Making LLMs useful

A language model pre-trained on web-scale text is a powerful prompt completion engine. It learns a complex distribution of human language. However, this makes it closer to a mimic. We want AI to be useful, especially in new scenarios that it did not see during training, such as solving new useful tasks. This is called *instruction-following*.

We describe simple methods for inducing instruction-following, including some that involve RL. Furthermore, once AI is good at instruction-following for a broad set of tasks, we want its power to be *aligned* with users' interests. A rough way to define *alignment* is that the model's answers should be *helpful, honest, and harmless*. This is the end-goal for the techniques described in this chapter.

Of course, there is a core challenge that arises in steering of a model's behavior to align with human goals: concepts such as "helpfulness" are complex, and do not boil down to a simple, differentiable loss function. We cannot train a model on "helpfulness" directly. In classical machine learning and AI going back 50 plus years, such problems were considered even impossible since the goal is hard to fully describe.

Surprisingly, alignment becomes reasonably achievable via a multi-stage process, whose success traces to the strong generalization capabilities of LLMs, especially the very large ones. First, we teach the model the *format* of a helpful assistant through **Supervised Fine-Tuning (SFT)**. We show it examples of good instructions and the desired responses. This step teaches the model to be a compliant instruction-follower. Surprisingly, trained with a smallish number of diverse examples, the model is able to then leverage its substantial knowledge store from pre-training to generalize to new situations. To make it aligned we refine its *behavior* by teaching it what humans *prefer*. This is done via preference optimization, using methods like Reinforcement Learning from Human Feedback (RLHF) or its more modern and direct successors, Direct Preference Optimization (DPO) and Group-Relative Policy Optimization (GRPO).

This chapter covers this post-training pipeline, from the initial supervised tuning to the advanced preference-based methods that enable state-of-the-art reasoning.

[Base Model] → [SFT Model] → [Aligned Model]

Figure 20.1: The post-training pipeline, starting with a base pre-trained model and progressively refining it with supervised fine-tuning and then preference optimization.

20.1 *Supervised Fine-Tuning (SFT)*

The first step toward instruction-following is conceptually straightforward. We take a pre-trained base model and fine-tune it on a high-quality, curated dataset of `(prompt, response)` pairs. This dataset is distinct from the pre-training corpus; it contains examples of instructions and the ideal way to follow them. The training objective is maximum likelihood estimation over the response tokens:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{(x,y)\in\mathcal{D}_{\text{SFT}}} \log P_\theta(y|x) \tag{20.1}$$

20.1.1 *A Probabilistic View: SFT as Forward KL-Minimization*

As established in the pre-training chapter, the maximum likelihood objective is equivalent to minimizing the Kullback-Leibler (KL) divergence from the true data-generating distribution to the model's distribution. In SFT, we are doing exactly this, but the data-generating distribution is now the empirical distribution of our curated dataset, let's call it $P_{\text{SFT}}(y|x)$. The model is a policy, $P_\theta(y|x)$. Thus, SFT minimizes the **forward KL-divergence**:

$$\arg\min_\theta D_{KL}(P_{\text{SFT}}||P_\theta) = \arg\min_\theta \mathbb{E}_{y\sim P_{\text{SFT}}}\left[\log \frac{P_{\text{SFT}}(y|x)}{P_\theta(y|x)}\right] \tag{20.2}$$

This objective has a distinct behavior. To avoid an infinite KL divergence, the model P_θ must assign non-zero probability to any response y that has non-zero probability under P_{SFT}. This property is known as **mode-covering**. The model is incentivized to "cover" all the modes (i.e., types of responses) present in the training data. If the SFT dataset contains both verbose and concise answers for similar questions, the model learns that both are plausible. This makes SFT effective at teaching the general format of instruction-following, as it learns the broad distribution of acceptable responses.

However, this mode-covering behavior is also a limitation. For many prompts, an "average" response is not a good response. We often prefer a single, high-quality mode. This motivates considering the alternative, the **reverse KL-divergence**:

$$D_{KL}(P_\theta||P_{\text{SFT}}) = \mathbb{E}_{y\sim P_\theta}\left[\log \frac{P_\theta(y|x)}{P_{\text{SFT}}(y|x)}\right] \tag{20.3}$$

This objective exhibits **mode-seeking** behavior. Here, the model is heavily penalized if it assigns probability to a response y that has zero probability under P_{SFT}. To minimize this loss, the model will seek out a single, high-probability mode in the target distribution and focus all its mass there.

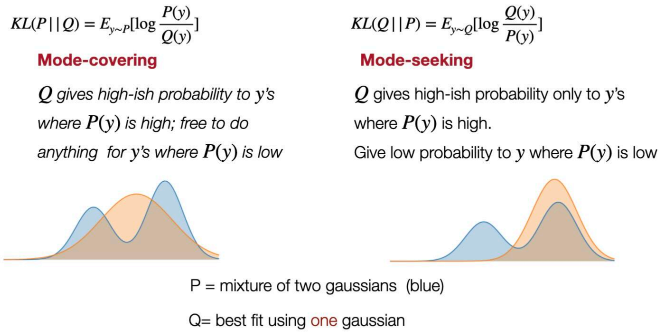

Figure 20.2: Difference between forward and reverse KL. In the latter, P tends not to assign high probability to points that have low probability in Q. (Figure idea from RL Probabilist Blog)

SFT generalizes well thanks to its mode-covering behavior. But this can be sub-optimal because, as illustrated in Figure 21.1, it may end up assigning high probability to many responses that the teacher might assign low probability to. We often desire mode-seeking behavior to improve instruction-following: to output *best* response (or one of the near-best responses). For instance, for a safety-critical query, we want the model to find the single mode of safe and helpful responses, not to cover a broader distribution that might include plausible but undesirable answers. The methods that follow SFT, particularly those based on reinforcement learning, are designed to instill this preference for high-quality modes. They effectively shift the optimization objective from the mode-covering forward KL of imitation learning toward the mode-seeking behavior characteristic of learning from feedback.

Problem 20.1.1. *Consider the problem of* model distillation. *Given a strong model we wish to use its outputs –specifically, token probabilities*[1]*— to train a smaller model Q.*

Explore which KL objective might be best, and whether you can think of some limitations of that objective.

20.2 *Learning from Preferences*

SFT teaches the model the general format of a good response, but it treats all examples in the curated dataset as equally valid. It learns the distribution of good answers, but not what makes one good answer better than another. To refine the model's judgment, we need a finer-grained signal.

[1] Note: Many commercial models don't allow observing token probabilities, to make distillation harder.

The core insight is that for complex tasks, it is far easier for humans to compare two outputs than to produce a high-quality one from scratch. We leverage this by collecting a dataset of preferences. For a given prompt x, we generate two responses from the SFT model, say y_1 and y_2, and ask a human labeler which one they prefer. We denote the preferred response y_w (for "winner") and the rejected one y_l (for "loser"). The goal is to use this data to learn a function that aligns with this latent human preference structure.

20.2.1 The Reward Model: An ELO System for Responses

We begin by positing the existence of a latent reward function, $r^*(y|x)$, that reflects the true quality of a response y to a prompt x in the mind of a human evaluator. [2] We can't access this function directly, but we can model how it influences choices.

The famous Bradley-Terry model provides the link. It posits that the probability of preferring one option over another is a function of the difference in their underlying scores. In our case:

$$P(y_w \succ y_l|x) = \sigma(r(y_w|x) - r(y_l|x)) \tag{20.4}$$

where $\sigma(z) = 1/(1 + e^{-z})$ is the sigmoid function. This model states that the larger the gap in quality between two responses (i.e., higher z), the more deterministic the preference. This is precisely how ELO ratings are inferred in chess: a player's latent "skill" (the reward) is estimated from a history of wins and losses (the preference data).

Our goal is to learn a parametric reward model, $r_\phi(x, y)$, that approximates r^*. This model is typically the SFT model with its language head replaced by a linear layer that outputs a single scalar. We train it by maximizing the likelihood of the observed human preferences in our dataset $\mathcal{D} = (x, y_w, y_l)$.

Problem 20.2.1 (Learning the Reward Model)**.** *Show that the maximum-likelihood objective for a reward function r_ϕ given a preference dataset $\mathcal{D}$ is equivalent to minimizing the following negative log-likelihood loss:*

$$\mathcal{L}_{RM}(\phi) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right] \tag{20.5}$$

20.3 Policy Optimization from Preferences

The previous section has brought us to an old idea in machine learning: *Reinforcement Learning (RL)*. It is often helpful to Viewing the language model —which outputs y when given a query x—as a distribution on actions (i.e., the choice of tokens to produce), which in RL is viewed as a *policy*. Switching to this RL view, we see that we

[2] This conceptual framework for ranking is widely used in creating rankings in sports, e.g. the ELO ratings in Chess. It is assumed that each player has a ground-truth scalar rating, and the outcome of a match between two players is a probabilistic outcome defined by Bradley-Terry formula.

have arrived at the classic two-stage RLHF pipeline: first, train a reward model r_ϕ to capture human preferences, and second, use that model as a reward function to fine-tune the SFT policy π_{SFT} via a KL-regularized RL objective.

This final optimization step is formalized as follows:

$$\max_{\theta} \quad \mathbb{E}_{y \sim \pi_\theta(y|x)}[r_\phi(y|x)] - \beta \cdot D_{KL}(\pi_\theta(y|x)||\pi_{SFT}(y|x)) \tag{20.6}$$

Let's dissect this objective.

The first term, $\mathbb{E}_{y \sim \pi_\theta(y|x)}[r_\phi(y|x)]$, is the pure reinforcement learning objective. It encourages the policy π_θ to generate responses y that achieve a high score from the reward model r_ϕ. The second term, $\beta \cdot D_{KL}(\pi_\theta(y|x)||\pi_{SFT}(y|x))$, is a crucial regularizer. It penalizes the policy π_θ for deviating too far from the initial SFT policy, π_{SFT}. This acts as a "leash," preventing the model from collapsing to a few high-reward but nonsensical sequences (a phenomenon known as "reward hacking") and ensuring it retains its general language competence. The coefficient β controls the strength of this penalty.

This viewpoint rooted in reinforcement learning, while groundbreaking, introduces significant practical challenges.

20.3.1 *The Challenge of On-Policy Reinforcement Learning*

The RL algorithm most commonly used to optimize the RLHF objective (Equation 20.6) is Proximal Policy Optimization (PPO). PPO is an on-policy algorithm. This means it learns by collecting experience (in our case, generating responses y from the policy π_θ) and using that experience to compute a single policy update. To compute the next update, it must discard the old experience and generate a new batch using the newly updated policy.

This on-policy nature creates major hurdles in the LLM context: *(a) Sample Inefficiency*: Training requires generating samples from the very policy being trained at every single step. This involves running expensive forward passes of a large language model to generate responses, which are then scored by the reward model, before a gradient step can be taken. *(b) Instability* The process is sensitive to hyperparameters and can be notoriously unstable, requiring careful tuning to converge. In contrast, an off-policy algorithm can learn from a fixed, static dataset of experience that was collected beforehand (e.g., by a different policy). This is far more efficient, as it decouples the expensive data generation step from the policy optimization step. The entire training run can use a single dataset collected once at the beginning.

The complexity of on-policy RLHF motivates a search for a more direct, off-policy method. Can we achieve the same KL-regularized reward maximization, but with a simpler, supervised-like objective that works on a static dataset?

20.3.2 *The Direct Path: Direct Preference Optimization (DPO)*

The answer lies in the optimal policy equation we derived (Equation 20.6). That equation provides a direct mapping from rewards to the optimal policy. The key insight of Direct Preference Optimization (DPO) is to reverse this mapping: we can use the equation to define the reward implicitly in terms of the policy, and then optimize the policy directly on the static preference data.

Before we can derive the DPO loss, we must first state the solution to the RLHF optimization problem. The optimal policy, π^*, that maximizes the KL-regularized objective in Equation 20.6 can be shown to have the following form:

$$\pi^*(y|x) = \frac{1}{Z(x)}\pi_{SFT}(y|x)\exp\left(\frac{1}{\beta}r_\phi(y|x)\right) \tag{20.7}$$

where $Z(x)$ is a partition function that ensures the probabilities sum to 1. This equation provides a link between the reward function r_ϕ and the optimal policy π^*. The key insight of Direct Preference Optimization (DPO) is to use this equation not to find the policy from the reward, but to define the reward implicitly in terms of the policy.

Now, we can proceed by rearranging Equation 20.7 to solve for the reward function $r_\phi(y|x)$:

$$r_\phi(y|x) = \beta \log \frac{\pi^*(y|x)}{\pi_{SFT}(y|x)} + \beta \log Z(x) \tag{20.8}$$

This equation reveals that the reward function is simply the scaled log-probability ratio between the optimal policy and the reference SFT policy, plus a term that depends only on the prompt x.

Now, let's consider the difference in rewards for a preferred and rejected pair, (y_w, y_l). The term $\beta \log Z(x)$ is constant for a given x and cancels out:

$$r_\phi(y_w|x) - r_\phi(y_l|x) = \beta\left(\log\frac{\pi^{(}y_w|x)}{\pi_{SFT}(y_w|x)} - \log\frac{\pi^{(}y_l|x)}{\pi_{SFT}(y_l|x)}\right) \tag{20.9}$$

We have just expressed the reward difference—the core component of the Bradley-Terry model—entirely in terms of policies. We can now substitute this expression back into the reward model loss from Equation 20.5. Instead of training a separate reward model r_ϕ, we now directly train our language model policy π_θ (with the SFT model as a fixed reference, which we'll call π_{ref}). This yields the DPO loss function:

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}(x, y_w, y_l) \sim \mathcal{D}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right] \tag{20.10}$$

This is a remarkable result. We have collapsed the two stages of RLHF into a single, stable, off-policy supervised learning objective. The term $\beta\log(\pi_\theta(y|x)/\pi_{ref}(y|x))$ can be interpreted as the implicit reward the policy π_θ is assigning to a response. The DPO objective simply increases the implicit reward for winner responses and decreases it for loser responses. It is a simple cross-entropy loss on preference pairs, where the "logits" are the differences in the implicit rewards.

Problem 20.3.1 (DPO Derivation)**.** *Starting with the optimal policy solution in Equation 20.7 and the reward model loss in Equation 20.5, formally derive the DPO loss function shown in Equation 20.10.*

DPO's elegance is its simplicity. It achieves the same objective as the more complex reward-modeling and on-policy RL pipeline but with a single, direct optimization step, making it more efficient and stable in practice.

20.4 Advanced RL: Learning from groups of answers (and Binary Rewards)

DPO provides an effective off-policy method for learning from pairs of responses. It works well, but it is also clear that learning is superior on-policy. Studies show that learning becomes less efficient if the policy is more than a few iterations old.

Now we describe a new method from DeepSeek's R1 paper[3] which involves binary rewards[4] generating a whole group of K responses for a single prompt, and then doing a single update for all of them.

This scenario gives us a different kind of preference signal: for a group of K responses, some are "winners" (pass the rules) and some are "losers" (fail the rules). A naive approach would be to form all possible (winner, loser) pairs and apply the DPO loss, but this is inefficient. A more powerful approach is to learn from the entire group holistically.

20.4.1 Group-Relative Policy Optimization (GRPO)

The most natural way to extend the DPO framework to a group setting is to treat the group as a collection of all possible winner-loser pairs. Given a group of responses for a prompt x partitioned into a

[3] Deeseek AI et al. 2025. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*

[4] These are useful in domains such as code generation or mathematical reasoning, where it is possible to create automated, rule-based systems to evaluate these responses —-e.g., reward 1 if the generated code pass a set of unit tests and 0 otherwise. The advantage of such data is that it can be generated at scale along with ground-truth rewards using automated methods. Furthermore, gains in reasoning from training in the rule-based setting generalizes automatically to standard settings.

set of winners $\mathcal{D}_w$ (of size N_w) and losers $\mathcal{D}_l$ (of size N_l), we can simply sum the standard DPO loss over all $N_w \times N_l$ pairs. This gives the following objective:

$$\mathcal{L}_{GRPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{\mathcal{D}} \left[\sum y_i \in \mathcal{D}_w \sum y_j \in \mathcal{D}_l \log \sigma(r\theta(y_i|x) - r_\theta(y_j|x)) \right] \tag{20.11}$$

where $r_\theta(y|x)$ is the implicit reward $\beta \log(\pi_\theta(y|x)/\pi_{ref}(y|x))$.

At first glance, this seems like a straightforward, even trivial, extension. However, the key insight of the DeepSeek paper lies not in the form of this loss function, but in the analysis of its gradients.

Let's consider the gradient of this loss with respect to the implicit reward of a single response. For a winning response $y_i \in \mathcal{D}_w$, its reward $r_\theta(y_i|x)$ appears in N_l different terms in the sum (once for each loser). Conversely, for a losing response $y_j \in \mathcal{D}l$, its reward $r_\theta(y_j|x)$ appears in N_w terms. This creates an imbalance. The magnitude of the gradient signal for a single winner is proportional to the number of losers, N_l, while the magnitude for a single loser is proportional to the number of winners, N_w.

If we have a group with 1 winner and 10 losers, the winner will receive a gradient update that is roughly 10 times larger than any individual loser. This can lead to unstable training, where the model focuses intensely on pushing the few winners up, while the signal to push the many losers down is comparatively weak and diffuse.

The paper's proposed solution is to rescale the gradients to balance the updates. They introduce a scaling factor α into the advantage term:

$$\mathcal{L}_{GRPO-scaled}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{\mathcal{D}} \left[\sum_{y_i \in \mathcal{D}_w} \sum_{y_j \in \mathcal{D}_l} \log \sigma(\alpha(r\theta(y_i|x) - r_\theta(y_j|x))) \right] \tag{20.12}$$

By setting $\alpha = 1/\sqrt{N_w N_l}$ (or other heuristics), the total magnitude of the updates applied to the winning set and the losing set can be balanced, leading to a more stable and effective optimization process. The core contribution is thus not a new loss function structure, but a deeper understanding and correction of the gradient dynamics of the most natural group-based extension.

Problem 20.4.1. *Consider the unscaled GRPO loss in Equation 20.11. Show that the gradient with respect to a winner's reward, $\frac{\partial \mathcal{L}}{\partial r_\theta(y_i|x)}$, involves a sum over all $j \in \mathcal{D}_l$. Similarly, show that the gradient for a loser, $\frac{\partial \mathcal{L}}{\partial r\theta(y_j|x)}$, involves a sum over all $i \in \mathcal{D}_w$. Explain why this leads to the gradient imbalance.*

20.4.2 A More Direct Approach: Advantage of the Mean

While the formulation in Equation 20.11 is a direct extension of pairwise preference, it can be viewed as summing the log-probabilities of individual advantages. A subsequent version of GRPO, used in DeepSeek-V2, proposes a more direct and arguably more robust objective. Instead of ensuring every individual winner beats every individual loser, it aims to ensure that the average *winner* is better than the average *loser*.

This is accomplished by first calculating the average implicit reward for the winning set and the losing set:

$$\bar{r}_\theta(\mathcal{D}_w|x) = \frac{1}{N_w}\sum_{y_i\in\mathcal{D}_w} r_\theta(y_i|x) \quad \text{and} \quad \bar{r}_\theta(\mathcal{D}_l|x) = \frac{1}{N_l}\sum_{y_j\in\mathcal{D}_l} r_\theta(y_j|x)$$

The loss is then a direct application of the DPO sigmoid loss to the advantage of these two mean rewards:

$$\mathcal{L}_{GRPO-V2}(\pi_\theta;\pi_{ref}) = -\mathbb{E}_{\mathcal{D}}\left[\log\sigma\left(\bar{r}_\theta(\mathcal{D}_w|x) - \bar{r}_\theta(\mathcal{D}_l|x)\right)\right] \qquad (20.13)$$

This "log-probability of the average advantage" approach has several benefits. It focuses on separating the two populations of responses (winners and losers) as a whole, which can be a more stable signal. It also naturally handles cases where some "winners" might be only marginally better than some "losers," by allowing the stronger winners to pull the average up. This formulation avoids the gradient scaling issues discussed for the previous version, providing a more direct and stable objective.

Problem 20.4.2. *Consider a group with two winners, y_1, y_2, and two losers, y_3, y_4. Write out the full loss expressions for this group using both the original GRPO formulation (Eq. 20.12) and the V2 formulation (Eq. 20.13). Compare the loss gradients for a single winner y_i under the original GRPO formulation (Eq. 20.11) and the V2 formulation (Eq. 20.13). How does the V2 formulation avoid the gradient imbalance issue discussed previously? (Hint: Is one always lower than the other? Consider Jensen's inequality.)*

20.4.3 A Margin-based variant of GRPO (and a Curriculum)

The GRPO-V2 objective (Eq. 20.13) is a powerful tool for aligning a policy with group preferences. Its loss function, based on $-\log\sigma(\cdot)$, is a standard choice in machine learning for binary classification, rooted in a probabilistic interpretation. This loss is always positive, and it continuously pushes the model to achieve an ever-larger advantage, even for prompts where the model is already performing well.

But this is not the only valid choice. Instead of the "soft" probabilistic sigmoid loss, we can consider the "hard" margin-based **hinge loss** from the Support Vector Machine literature. If we replace the $-\log \sigma(z)$ term with a hinge loss on the group advantage, we still get a reasonable method that encourages winners to score higher than losers.

Let's define our loss using the same group advantage metric, $A_{group}(x) = \bar{r}_\theta(\mathcal{D}_w|x) - \bar{r}_\theta(\mathcal{D}_l|x)$. The new objective becomes:

$$\mathcal{L}_{prompt-hinge}(\pi_\theta; \pi_{ref}) = \mathbb{E}_{\mathcal{D}} \left[\max(0, m - A_{group}(x))\right] \tag{20.14}$$

where m is a pre-defined margin. This objective is reasonable because, like the sigmoid-based loss, it pushes the model to make the group advantage $A_{group}(x)$ large and positive.

However, this simple change has an interesting implication for the training dynamics, **because it naturally implies a curriculum.** As training progresses, prompts for which the model has achieved a sufficient performance margin automatically drop out of the optimization process. This can be seen by realizing that key difference is the loss's behavior for "easy" prompts.

- The sigmoid-based loss, $-\log \sigma(A_{group}(x))$, approaches zero as the advantage grows, but it is never *exactly* zero. There is always a small, non-zero gradient, encouraging the model to achieve an even larger advantage.
- The hinge loss, $\max(0, m - A_{group}(x))$, becomes **exactly zero** as soon as the group advantage surpasses the margin ($A_{group}(x) > m$).

When the loss for a given prompt is zero, its contribution to the gradient is also zero. Consequently, the model's parameters are no longer updated based on this prompt. This mechanism creates an *implicit curriculum*: as training progresses, prompts that the model has "mastered" (i.e., separated by more than the margin m) automatically drop out of the optimization process. The model then naturally focuses its capacity on the remaining "hard prompts" for which the loss is still positive, leading to a more efficient and targeted training process.