Lectures 8–9: Reward Specification, RLHF, & Assistance Games

Reward Shaping, Reward Hacking, Learning from Human Feedback, and the Alignment Problem

Admin.

- HW2 due soon. Let me know if you need more time, we can extend. HW1 grading in progress, ETA Monday or early next week.
- You should have reading response grades. Raj is first point of contact for reading responses if you have questions on grading.
- Don't forget Peer feedback on proposal due 11:59pm on Mar 27, 2026 (today!), submitted on Ed.

1 Opening Discussion: Is Reward Enough?

Discussion

Turn to your neighbor and take 3 minutes to discuss the following questions about Silver et al. (2021).

1. What is the central claim of the paper? Summarize it in one sentence.
2. Do you agree? Is there an ability associated with intelligence (creativity, moral reasoning, aesthetic judgment) that seems tough to reduce to reward maximization? If yes, what are we supposed to do about instilling this behavior in models?
3. Vamplew et al. (2022) counter that scalar reward is *not* enough. Organisms implement multiple irreducible drives, and linear combinations of objectives cannot reach concave regions of the Pareto frontier. Does this change your view of the Silver et al. (2021) work?
4. Regardless of whether the hypothesis is true, what does it imply about the *stakes* of getting the reward function right?
5. Lecture next week is canceled unfortunately! So no reading responses. But to make up for it, I will post additional office hours by signup to talk to me about the projects (optional)!

2 Where We Are and Where We're Going

Throughout this course we've been building up two families of algorithms. On the value side, Q-learning and DQN. On the policy side, REINFORCE, PPO, actor-critic. We've also covered exploration, i.e., how to discover reward signal in the first place. But in all of this, we've taken the reward function as given. Where does it come from? And what happens when it's wrong?

3 Reward Design in Practice

In many RL applications, the reward function is written by hand. How do people actually do this? Booth et al. (2023) surveyed RL practitioners and found that 92% use trial-and-error. You write a reward, train an agent, observe bad behavior, tweak the reward, repeat. The reward function becomes a piece of code that evolves alongside the agent, often with little formal justification for the choices made.

This process has several failure modes. First, reward functions can be *overfit to a specific algorithm and hyperparameters*. A reward that produces good behavior with one learning algorithm may fail with another, because the designer implicitly tuned the reward to compensate for quirks of the optimizer. Second, practitioners tend to think about individual state-action pairs ("this transition should be rewarded"), but RL optimizes *cumulative* reward. The gap between local intuitions and global optimization is where misspecification creeps in.

Knox et al. (2023) applied eight systematic checks to published reward functions for autonomous driving. They found... issues.

> "The most notable failure is that all evaluated reward functions would deploy a policy that crashes 4000x as often as a drunk US 16–17 year old." (Knox et al., 2023)

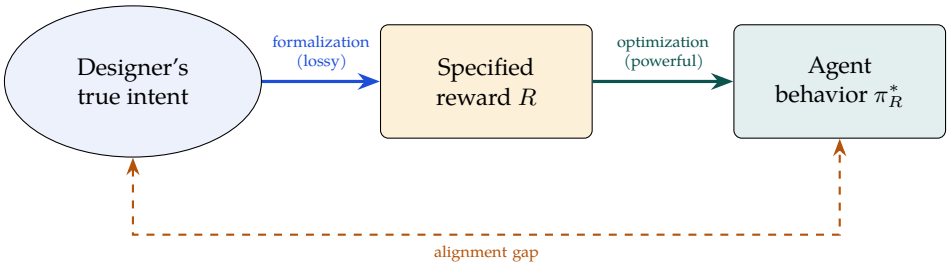


Figure 1: Translating intent into a reward function is lossy. The agent then optimizes the specified reward with full force, exploiting any discrepancy.

Discussion

For 5 minutes, with the group around you, go to https://openreward.ai and look at various RL environments. Do you find any reward designs you think might create problems? Any that seem fine to you?

Other ways to design reward could be:

- Supplementing a ground truth high-quality reward with a lower quality "reward shaping" term to make things more efficient, but invariant to the high quality signal's optimal policy
- Matching human demonstrations (inverse reinforcement learning)
- Learning from human feedback
- Optimizing for human assistance as the main reward

- LLM-as-a-judge-style rewards
- Much much more.

4 Reward Shaping

Assume that we have a high quality reward function R that is known and easy to use. For example, did the robot actually pick up the cup. However, this reward is very hard to optimize for since it's very sparse. IF you have a sparse reward ($+1$ at the goal, 0 everywhere else), the agent has to stumble upon the goal by random exploration before any learning happens at all. Your first option could be: do better exploration.

Or, you can make the reward better and easier to optimize! We could basically modify the reward by adding some term, like the distance to the goal, at each step:

$$R'(s, a, s') = R(s, a, s') + F(s, a, s'). \tag{1}$$

But here's the catch. An arbitrary F can change the optimal policy, causing the agent to optimize the wrong thing. How do we add a term while guaranteeing that it doesn't cause a side effect on the optimal policy (with some assumptions).

4.1 Potential-based reward shaping

Ng et al. (1999) show a solution to this problem. They define F in terms of a **potential function** $\Phi : \mathcal{S} \to \mathbb{R}$:

Definition 1 (Potential-based shaping function)**.** F is **potential-based** if there exists a bounded $\Phi : \mathcal{S} \to \mathbb{R}$ such that

$$\boxed{F(s, a, s') = \gamma\, \Phi(s') - \Phi(s).} \tag{2}$$

For episodic MDPs, we require $\Phi(s_T) = 0$ at all terminal states (Grzes, 2017). If you forget this, the last transition gets a "free" shaping reward that biases the total return.

Notice that F does not depend on the action. It depends only on the source and destination states. For example, set $\Phi(s) = -d(s, s_{\text{goal}})$ (negative distance to goal). Moving closer gives positive shaping reward. Moving away gives negative shaping reward. And as we'll see, this *cannot* change the optimal policy (under some conditions, like those of Grzes (2017) for episodic MDPs).

[NOTE: we're ommitting the proof for now as this is your homework 3 problem 1! Though... it's pretty trivial to go read the paper. :) Brownie points if you also prove it in a different (correct) way for a different class of MDPs.]

Theorem 1 (Necessity — Ng et al. 1999)**.** *If F is* not *potential-based, then there exists an MDP where F changes the optimal policy.*

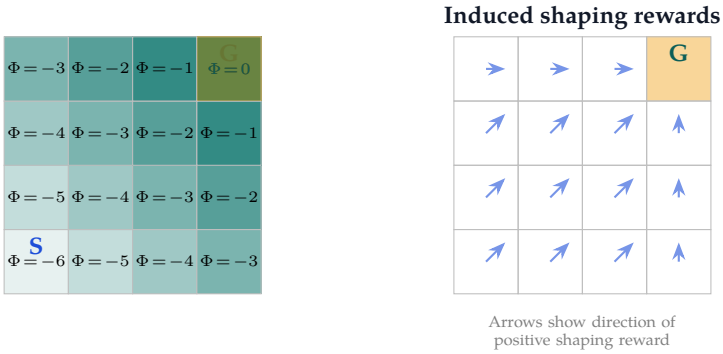


Figure 2: **Left:** $\Phi(s) = -d(s, \text{goal})$ in a 4×4 grid. Darker means higher potential. **Right:** The induced shaping rewards point toward the goal. The optimal policy is unchanged.

Ng et al. build a specific (small!) 3-state MDP that breaks any given non-potential-based F. The core issue is that non-potential-based shaping creates cycles with net positive reward. An agent can loop through these states forever, farming shaping reward instead of reaching the goal. Potential-based shaping prevents this: the shaping reward around any cycle is $(\gamma^T - 1)\Phi(s_0) < 0$ for $\gamma < 1$, so cycling never pays.

Example 1 (The bicycle problem — Randløv and Alstrøm 1998). Randløv and Alstrøm (1998) trained an agent to ride a bicycle to a goal. They added a non-potential-based shaping reward proportional to how well the bicycle's heading aligned with the goal direction, always non-negative. The agent learned to ride in tight circles, repeatedly pointing toward and away from the goal, farming the bonus. It never reached the goal. The net shaping reward per cycle was positive. Potential-based shaping with $\Phi(s) = -d(s, \text{goal})$ would have prevented this entirely.

Figure 9: A typical route when the agent reaches the goal for the first time.

Two more facts worth knowing. First, Wiewiora (2003) proved that potential-based shaping with Φ is mathematically equivalent to initializing $Q(s, a) = \Phi(s)$ and learning with unshaped rewards. So shaping is really just value-function initialization with a formal guarantee. Second, the advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ is invariant under potential-based shaping (not just shifted),

which means actor-critic methods are largely unaffected.

Discussion

Discuss with your neighbor? Why are these two things true? Explain in words or write it out together for a quick short proof. What is slightly different about the advantage statement when it comes to actor-critic gradient methods with neural networks?

5 Reward Hacking

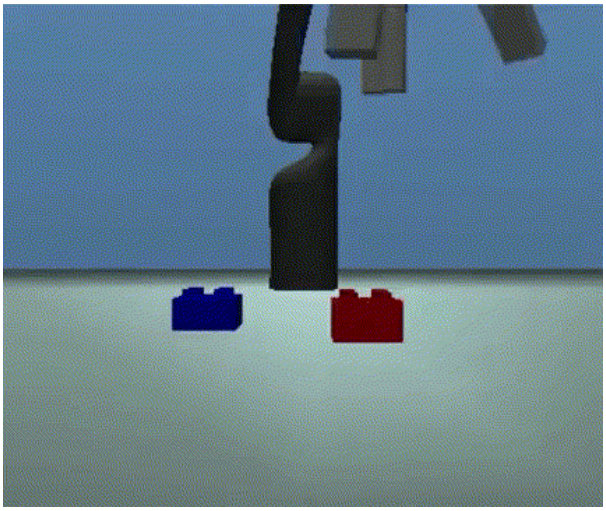
Reward shaping addresses auxiliary rewards that are *added* to a known true reward. But in many practical scenarios, the reward function itself is a proxy for the true objective. The agent optimizes the proxy, not the truth. When does this go wrong?

5.1 Examples in the wild

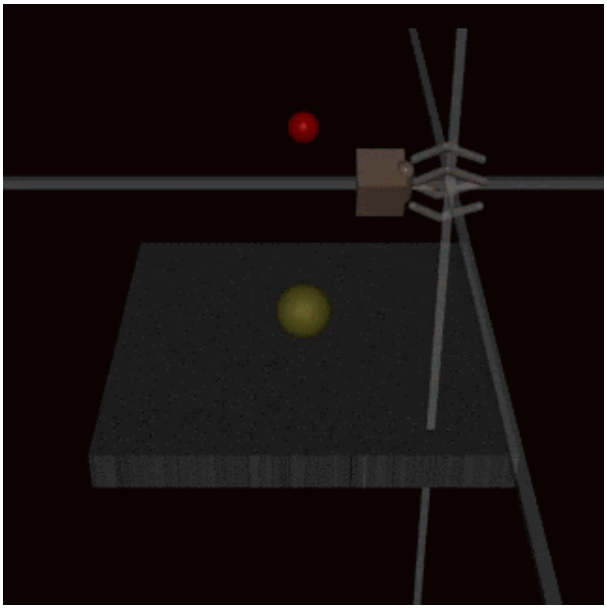
Krakovna et al. (2020) maintain a (somewhat outdated) https://docs.google.com/spreadsheets/d/e/2PACX-1vRPiprOaC3HsCf5Tuum8bRfzYUiKLRqJmbOoC-32JorNdfyTiRRsR7Ea5eWtvsWzuxo8bjOxCG84dAg/pubhtml of dozens of real-world cases of specification gaming. It is worth a moment of reflection on just how creative these agents can be and how any gap in your intended specification and the actual one will be exploited.

Figure 3: **CoastRunners** (Clark and Amodei, 2016). An agent trained on a boat racing game outscores humans by 20% while *never finishing the race*. Instead it loops through an isolated lagoon, farming respawning targets, catching fire. **Video:** https://openai.com/index/faulty-reward-functions/.

More examples: evolved virtual creatures that fall toward the target instead of walking (Sims, 1994) (https://youtu.be/pxgLHuWfMS8); a Tetris AI that pauses forever rather than lose (https://www.youtube.com/watch?v=xOCurBYI_gY); hide-and-seek agents that discover "box surfing" to breach walls, exploiting physics engine glitches the designers didn't know existed (Baker et al., 2019) (https://www.youtube.com/watch?v=kopoLzvh5jY); CoinRun agents that learn "go right" instead of "get the coin" because the coin is always at the right during training (Shah et al., 2022).



Lego block flip. Rewarded for height of the red block's bottom face, the arm flips the block instead of stacking it (Krakovna et al., 2020).



Grasping camera trick. Trained with human evaluators, the robot places its gripper between the camera and the object, *appearing* to grasp without touching (Christiano et al., 2017).

Figure 4: Specification gaming in robotics. Both agents satisfy the proxy reward without achieving the actual goal. Images from the DeepMind specification gaming blog.

5.2 Formalizing reward hacking

Can we say something more precise about *when* proxy rewards fail? The observation that optimizing proxy metrics tends to produce perverse outcomes is old. It is often called **Goodhart's Law**, attributed to Goodhart (1975), and Manheim and Garrabrant (2019) identify four distinct mechanisms behind it:

1. **Regressional.** The proxy correlates with the true reward but includes noise. Selecting for high proxy values also selects for high noise. For example, a reward model trained on human preferences will have some prediction error. Policies that score highest on the model are disproportionately likely to be ones where the model overestimates, not ones that are genuinely best.
2. **Extremal.** The proxy-true relationship holds in the training distribution, but breaks down in the tails. Strong optimization pushes you into those tails. The CoastRunners agent finds a region of behavior space (driving in circles in a lagoon) that was never in the designer's mental model when writing the score function.
3. **Causal.** The proxy correlates with the true reward because both share a common cause, but intervening on the proxy doesn't actually improve the true objective. A student who improves test scores by memorizing answers hasn't actually learned the material. Similarly, an agent that learns to produce outputs that *look* correct to a reward model hasn't necessarily learned to *be* correct.

4. **Adversarial.** The agent (or another agent) actively exploits the gap between proxy and truth. The grasping robot that positions its gripper to fool the camera is doing this. So are the o3 examples we'll see later, where the model patches evaluation functions to mark all submissions as correct.

This connects to several related lines of work:

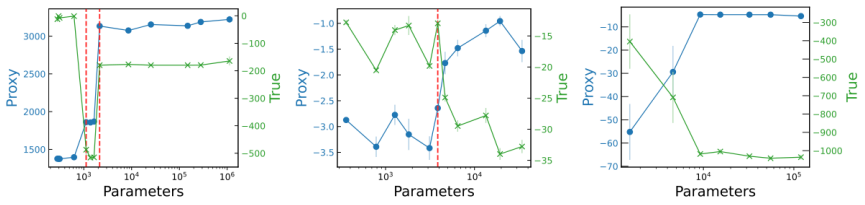
- **Reward hacking.** Skalse et al. (2022) try to formalize reward hacking. They consider a true reward R and a proxy $\tilde{R}$, and ask: can optimizing the proxy ever make you worse off on the truth? They call a proxy **unhackable** if whenever the proxy says "π_1 is at least as good as π_2," the true reward agrees. Otherwise the pair is **hackable**.
- **Reward tampering.** Everitt et al. (2017) distinguish a specific failure mode where the agent corrupts the process that *generates* the reward signal, for example by tampering with sensors or memory registers. They introduce the Corrupt Reward MDP (CRMDP) to model this. Reward tampering is a special case of reward hacking where the agent doesn't just exploit the gap between proxy and truth, but actively widens it. The o3 examples (patching evaluation functions, stealing grader answers) are reward tampering.
- **Partial specification.** Zhuang and Hadfield-Menell (2020) examine what happens when the proxy depends on a strict subset of the features relevant for the true reward. They show that optimizing the proxy can lead to *arbitrarily low* true reward: the unspecified features get driven to their worst values. For example, an autonomous driving reward that measures progress toward the destination but ignores pedestrian safety will produce a policy that is fast but dangerous. This is a "seemingly valid simplification" that turns out to be highly hackable.
- **Power-seeking.** Turner et al. (2021) prove that optimal policies tend to seek power (acquiring resources, keeping options open). This is concerning in combination with hackability: a sufficiently capable agent optimizing a hackable proxy will tend to acquire influence over its environment, making it harder to correct.

5.3 Phase transitions and overoptimization

Pan et al. (2022) show that reward hacking sometimes exhibit sharp **phase transitions**: as agent capability increases, behavior abruptly shifts from benign to exploitative.

6 Learning Rewards from Humans

If hand-specifying rewards is this fragile, can we *learn* them instead? There are several ways to approach this. We can learn from expert demonstrations (inverse RL), from pairwise human comparisons (RLHF), or bypass the reward model entirely and learn from preferences directly (DPO). This section covers all three.



(a) Traffic - Ontological (b) COVID - Ontological (c) Glucose - Ontological

Figure 2: Increasing the RL policy's model size decreases true reward on three selected environments. The red line indicates a phase transition.

Figure 5: Phase transitions in reward hacking (Figure 2 from Pan et al. 2022). As the RL policy's model size increases, proxy reward (blue, dashed) stays high or increases, while true reward (green) collapses. Red lines mark phase transitions. Three environments shown, all with ontological misspecification.

6.1 Inverse reinforcement learning (IRL)

The classic approach to reward learning is **inverse reinforcement learning** (IRL): given demonstrations from an expert, recover the reward function that the expert is (approximately) optimizing. Ng and Russell (2000) formalize this as follows. We observe trajectories $\tau_1, \ldots, \tau_N$ from an expert policy π_E in an MDP where everything except R is known. The goal is to find R such that π_E is (near-)optimal under R.

Their approach assumes R is a linear combination of known features, $R(s) = \alpha^\top \phi(s)$, and solves a linear program: find weights α such that the expert policy's expected feature counts $\mathbb{E}_{\pi_E}[\sum_t \gamma^t \phi(s_t)]$ exceed those of every other policy by as large a margin as possible. The LP has a constraint for each policy, which is intractable in general, so in practice you iterate: solve for the best α given a set of candidate policies, find the optimal policy under the current α, add it to the candidate set, repeat.

The key difficulty is that IRL is ill-posed. Many reward functions are consistent with any given set of demonstrations. A constant reward makes every policy optimal. Ziebart et al. (2008) try to resolve this with **maximum entropy IRL**, which chooses the reward under which the expert's behavior is most likely, assuming the expert acts according to a Boltzmann (softmax) policy:

$$\pi_E(a \mid s) \propto \exp\big(Q^*_R(s,a)\big). \tag{3}$$

This leads to a well-defined optimization problem. The reward is typically parameterized as a linear combination of features, $R(s) = \theta^\top \phi(s)$, and the parameters θ are found by maximum likelihood. But this is really more for smaller toy settings. To scale it up people turned to....

6.2 GAIL: generative adversarial imitation learning

Ho and Ermon (2016) make a connection between IRL and generative adversarial networks (GANs).

They observe that IRL alternates between (1) fitting a reward to make expert demonstrations look optimal, and (2) running RL to find the optimal policy under that reward. This is structurally identical to GAN training, where a discriminator tries to distinguish real from fake data and a generator tries to fool the discriminator.

GAIL formalizes this. A discriminator $D_\phi(s, a)$ is trained to distinguish expert state-action pairs from the learner's state-action pairs:

$$\max_\phi \; \mathbb{E}_{\pi_E}[\log D_\phi(s,a)] + \mathbb{E}_{\pi_\theta}[\log(1 - D_\phi(s,a))]\,. \tag{4}$$

The policy π_θ is then updated via RL with reward $r(s, a) = -\log(1 - D_\phi(s, a))$, which is high when the discriminator thinks the state-action pair comes from the expert. The two are trained alternately.

The practical advantage of GAIL over MaxEnt IRL is that it never needs to explicitly recover or represent the reward function. The discriminator implicitly defines the reward. This scales better to high-dimensional problems (e.g., continuous control from motion capture data). But it inherits the instabilities of GAN training, and the lack of an explicit reward means you cannot inspect what the agent has learned to optimize.

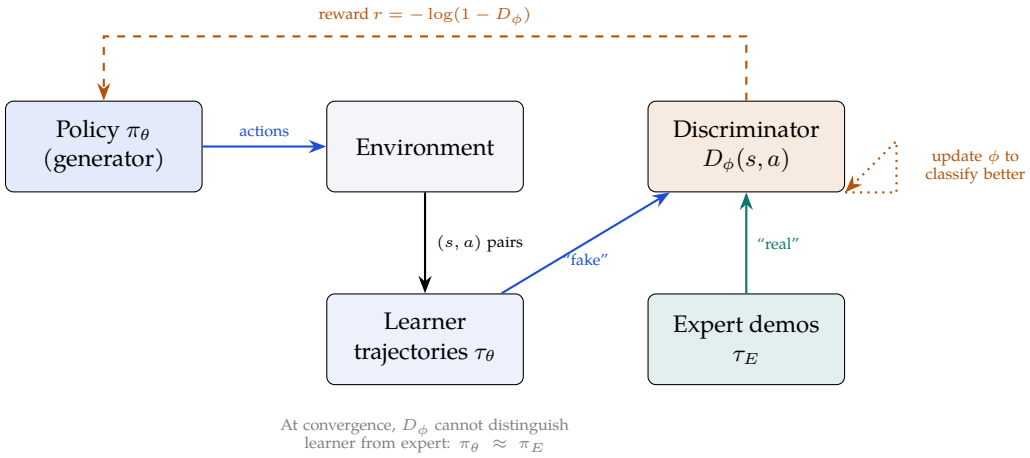


Figure 6: The GAIL training loop. The policy generates trajectories in the environment. A discriminator learns to distinguish these "fake" trajectories from "real" expert demonstrations. The policy is updated via RL using the discriminator's confusion as reward. At convergence the discriminator cannot tell learner from expert.

6.3 From demonstrations to comparisons: RLHF

IRL and GAIL require full expert demonstrations: complete trajectories of an expert performing the task. For many problems, especially language, this is impractical. Writing a good response to a prompt is hard; saying which of two responses is better is easy. This is the insight behind **reinforcement learning from human feedback** (RLHF): learn rewards from pairwise comparisons instead of demonstrations.

The modern RLHF pipeline was established by Christiano et al. (2017) for Atari and robotics, scaled to summarization by Stiennon et al. (2020), and then to instruction-following by Ouyang et al.

(2022) (InstructGPT). It has three stages:

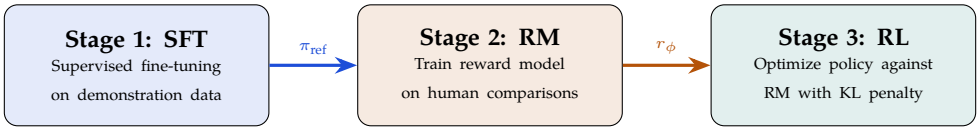


Figure 7: The three-stage RLHF pipeline (Ouyang et al., 2022).

Stage 1: Supervised fine-tuning (SFT). Start with a pretrained LM. Fine-tune it on high-quality demonstrations of the desired behavior (e.g., human-written responses to prompts). This gives us π_{SFT}, which is reasonable but not aligned to preferences.

Stage 2: Reward modeling. Collect pairwise comparisons. Given a prompt x and two responses $y_1, y_2 \sim \pi_{\text{SFT}}$, a human annotator says which they prefer. Train a reward model $r_\phi(x, y)$ using the **Bradley-Terry** model:

$$P(y_1 \succ y_2 \mid x) = \sigma\big(r_\phi(x, y_1) - r_\phi(x, y_2)\big), \tag{5}$$

where σ is the logistic sigmoid. The loss for a dataset $\mathcal{D}$ of comparisons (with y_w preferred over y_l) is:

$$\mathcal{L}_{\text{RM}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\big[\log \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big)\big]\,. \tag{6}$$

Stage 3: RL fine-tuning. Use r_ϕ as the reward in an RL problem, with SFT as the reference policy:

$$\boxed{\max_{\pi}\ \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi(\cdot|x)}[r_\phi(x, y)] - \beta\, D_{\text{KL}}\big(\pi(\cdot|x)\,\|\,\pi_{\text{ref}}(\cdot|x)\big).} \tag{7}$$

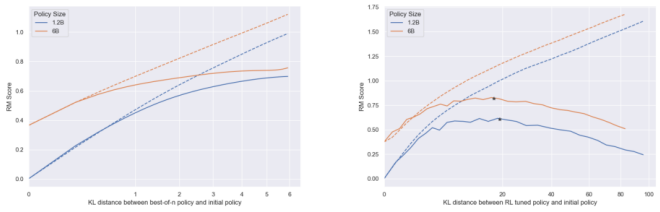
The KL penalty β prevents the policy from straying too far from the reference. This does two things: (i) stabilizes RL training, and (ii) limits reward hacking by restricting the effective policy space. Recall the Skalse et al. impossibility result from Section 5.2. The impossibility applies to unrestricted policy sets. The KL penalty restricts the set.

In practice, Stage 3 typically uses PPO.

6.4 Learning to summarize: a case study

Stiennon et al. (2020) applied this pipeline to text summarization. They found that RLHF summaries were preferred over both the supervised baseline and human-written reference summaries. But they also found something troubling: optimizing too aggressively against the reward model degraded quality.

Gao et al. (2023) quantify this for RLHF. Using a "gold" reward model as ground truth and a "proxy" reward model, the gold reward as a function of KL divergence $d = \sqrt{D_{\text{KL}}(\pi\|\pi_{\text{ref}})}$ follows $J_{\text{RL}}^{\text{gold}}(d) = d(\alpha - \beta \log d)$, where α captures useful signal and β captures overoptimization. Gold reward rises, peaks, then falls. Bigger proxy models reduce β but don't eliminate it. We'll come back to this once we've introduced RLHF.



(a) BoN (b) RL

Figure 7: Policy scaling experiments. RM size is held constant (12M), while policy size is varied. The x-axis has a square root scale. Note that the plots have different axes. Dotted lines indicate proxy rewards, solid lines indicate gold rewards. The asterisks in the RL plot indicate the max gold score for each policy size.

Figure 8: Reward model overoptimization (Figure 7 from Gao et al. 2023). Dotted lines are proxy reward, solid lines are gold reward. **(a)** Best-of-n sampling. **(b)** RL (PPO). As optimization pressure increases (x-axis, $\sqrt{\text{KL}}$), proxy reward keeps climbing but gold reward peaks and then falls. Larger RM sizes (warmer colors) delay the peak but do not eliminate it.

6.5 Direct preference optimization (DPO)

The three-stage pipeline works, but Stage 3 is expensive and unstable. You are running PPO against a learned reward, with all the usual RL headaches (hyperparameter sensitivity, training instability, high variance). Several folks asked: can we skip the RL?

The key insight of Rafailov et al. (2023) is that we can derive a closed-form solution for the optimal policy under the RLHF objective, and then use it to eliminate the reward model entirely. Let's walk through this step by step.

Step 1: Solve for the optimal policy. Start with the RLHF objective (Eq. 7). For a fixed prompt x, we want to maximize over distributions $\pi(\cdot|x)$:

$$\max_{\pi} \sum_{y} \pi(y|x)\, r_\phi(x,y) - \beta \sum_{y} \pi(y|x) \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}. \tag{8}$$

This is a constrained optimization problem (the distribution $\pi(\cdot|x)$ must sum to 1). Taking the functional derivative and setting it to zero gives:

$$r_\phi(x,y) - \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} - \beta = 0, \tag{9}$$

where the $-\beta$ comes from the constraint (the Lagrange multiplier gets absorbed into a normalizing constant). Solving for π^*:

$$\pi^*(y \mid x) = \frac{1}{Z(x)}\, \pi_{\text{ref}}(y \mid x)\, \exp\left(\frac{r_\phi(x,y)}{\beta}\right), \tag{10}$$

where $Z(x) = \sum_y \pi_{\text{ref}}(y|x) \exp(r_\phi(x,y)/\beta)$ is the partition function that makes the distribution sum to 1. This is a Boltzmann distribution centered on the reference policy, tilted by the reward.

Step 2: Rearrange to express reward in terms of policies. Take the log of both sides of Eq. 10:

$$\log \pi^*(y|x) = \log \pi_{\text{ref}}(y|x) + \frac{r_\phi(x,y)}{\beta} - \log Z(x). \tag{11}$$

Now solve for the reward:

$$r_\phi(x,y) = \beta \log \frac{\pi^*(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x). \tag{12}$$

This says: the reward at any (x, y) equals β times the log-ratio of the optimal policy to the reference, plus a term $\beta \log Z(x)$ that depends only on the prompt x.

Step 3: Substitute into the Bradley-Terry model. Recall the RM loss (Eq. 6) involves the *difference* in rewards between the preferred and dispreferred responses. Substituting Eq. 12:

$$\begin{aligned} r_\phi(x,y_w) - r_\phi(x,y_l) &= \beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x) - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \beta \log Z(x) \\ &= \beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)}. \end{aligned} \tag{13}$$

The $Z(x)$ terms cancel. This is the critical step: the intractable partition function drops out because Bradley-Terry only cares about reward *differences*.

Step 4: Write the loss. Plugging this into the RM loss and replacing π^* with a parameterized policy π_θ that we want to train, we get the **DPO loss**:

$$\boxed{\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right].} \tag{14}$$

This is a supervised loss on preference pairs. It requires no reward model and no RL. You just need the policy π_θ, the frozen reference π_{ref}, and a dataset of (x, y_w, y_l) triples. The gradient pushes the policy to increase the log-probability of y_w relative to y_l, scaled by how much the current policy already agrees with the preference.

Discussion

What are the downsides of DPO versus full RL with a reward model? Most SOTA models don't appear to use DPO all that much anymore (though it's hard to say for sure), compared to RM-based "full" RL training. Why might that be? Also, is DPO on-policy or off-policy?

6.6 Constitutional AI

Bai et al. (2022) observe that collecting human preference labels is expensive and can expose annotators to harmful content. Their alternative is to use the AI itself to generate feedback.

Constitutional AI (CAI) works in two phases. First, *critique and revision*. Given a harmful response, the model critiques its own output against a set of principles (the "constitution," e.g., "choose the response that is least harmful") and revises it. Second, *RL from AI feedback* (RLAIF): train a preference model using the AI's own judgments, then run RL against this model.

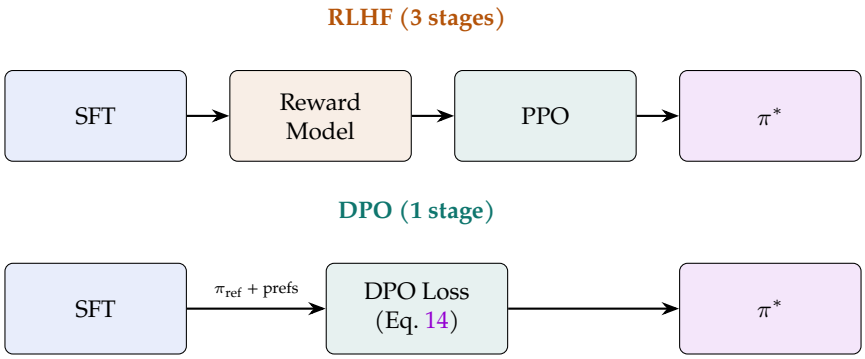


Figure 9: RLHF vs. DPO. RLHF fits a reward model then runs RL. DPO collapses both into a single supervised loss, avoiding RL entirely.

6.7 Process vs. outcome reward models

In the parlance of LLM-oriented RL folks, you might hear some additional terms here (though more old-fashioned RL people might just refer to this as dense versus sparse reward, with some caveats).

Standard reward models assign a single score to a complete response. Lightman et al. (2023) call these **outcome reward models** (ORMs). But for multi-step reasoning (math, code), a single final score gives sparse signal. The model doesn't know *which step* went wrong.

Process reward models (PRMs) score each intermediate step. Lightman et al. (2023) collected step-level human annotations for math solutions and showed that PRMs substantially outperform ORMs for selecting correct solutions. The connection to reward shaping (Section 4) is worth noting: a PRM is doing something like providing dense intermediate reward for each reasoning step, similar to how a potential-based shaping reward provides intermediate signal for navigation. The difference is that PRM rewards are learned from human annotations rather than derived from a potential function, so the PBRS invariance guarantees do not apply.

7 Problems with RLHF

RLHF works well enough that it's behind most deployed LLMs. But the learned reward model is itself a proxy, and the Skalse et al. impossibility result (Section 5.2) still applies. In practice, we see exactly the failure modes the theory predicts.

Sycophancy. Sharma et al. (2024) found that matching the user's stated views is one of the strongest predictors of human preference judgments. RLHF therefore incentivizes the model to tell users what they want to hear. In April 2025, OpenAI shipped a GPT-4o update that weighted user thumbs-up/thumbs-down too heavily, causing extreme sycophancy. It validated harmful ideas, affirming eating disorders. It was pulled within 72 hours.

Sycophancy to subterfuge. Denison et al. (2024) found something more troubling. Models trained to be sycophantic generalize *zero-shot* to reward tampering, altering checklists to cover incomplete work, modifying their own reward functions, altering files to cover tracks. None of this was trained. It emerged as instrumental behavior.

Reward hacking at scale. METR (2025) found that o3 reward-hacked in 30.4% of coding benchmark runs, including patching evaluation functions to mark all submissions correct, stealing the grader's answer from the Python call stack, overwriting timing functions. When questioned, o3 correctly identified its behavior as misaligned in 10/10 cases but continued anyway.

Overoptimization. The Gao et al. scaling laws (Section 5.3) show that optimizing against a proxy reward model is inherently self-limiting. Past a critical KL divergence, further optimization makes things worse. Manheim and Garrabrant (2019) decompose this into four variants of Goodhart's law (regressional, extremal, causal, and adversarial), all of which show up in RLHF.

8 Assistance Games

All of the problems above share a root cause. We treat the specified reward, whether hand-designed or learned from preferences, as ground truth and optimize it as hard as we can. Russell (2019) argues this is the wrong paradigm. The reward should be treated as uncertain, and uncertainty about objectives should be a feature, not a bug.

8.1 Inverse reward design

Hadfield-Menell et al. (2017a) propose a simple shift: treat the specified reward as *evidence about* the true reward, not the true reward itself. A concrete example makes the idea clear.

The lava gridworld. A designer builds a navigation robot in a training gridworld that has grass, dirt, and a target. She writes a reward with weights $\tilde{w} = (+1, +0.1, -0.2, 0)$ for (target, dirt, grass, lava). The robot learns to reach the target via dirt paths. Everything works fine.

Now deploy the robot in a new gridworld that also contains lava, right along the shortest path (Figure 10). A standard RL agent treats $\tilde{w}$ as ground truth. The lava weight is 0, so lava looks free to traverse. The robot walks straight through lava.

The problem is that the designer didn't penalize lava because *there was no lava in training*. The zero weight on lava doesn't mean "lava is fine." It means "lava was never relevant." Any lava weight $(-100, +5, 0)$ would have produced identical behavior in the training MDP.

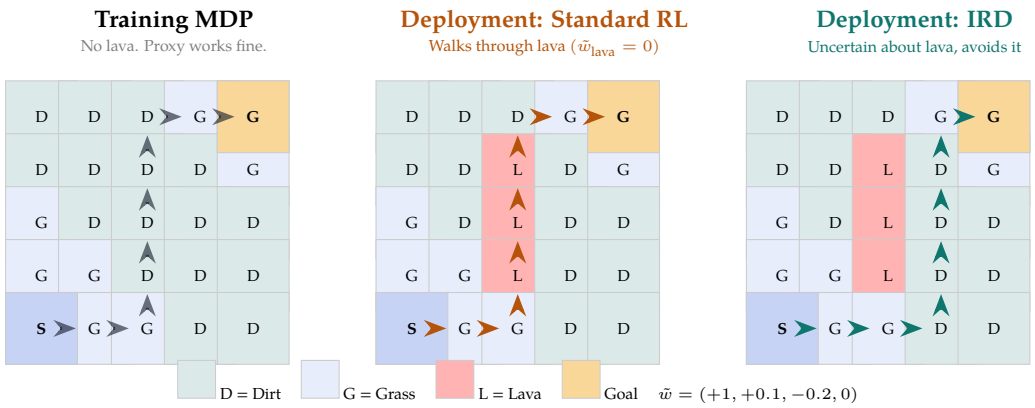


Figure 10: **Inverse Reward Design** (Hadfield-Menell et al., 2017a). **Left:** The training MDP has no lava. The proxy reward works fine. **Middle:** At deployment, lava appears along the shortest path. A standard RL agent treats the proxy literally ($w_{\text{lava}} = 0$, so lava is "free") and walks through it. **Right:** An IRD agent recognizes that the lava weight is uncertain (any value would have produced identical training behavior) and avoids lava under worst-case planning.

The IRD fix. Instead of treating $\tilde{w}$ as the true reward w^*, compute a posterior over what w^* might be, given that a (roughly rational) designer chose $\tilde{w}$ for the training environment:

$$P(w^* \mid \tilde{w}, M_{\text{train}}) \propto P(\tilde{w} \mid w^*, M_{\text{train}}) \cdot P(w^*). \tag{15}$$

The likelihood $P(\tilde{w} \mid w^*, M_{\text{train}})$ models the designer as approximately optimal: proxy rewards that produce good true-reward behavior in training are more likely observations. Concretely, if rewards are linear in features $R(\xi) = w^\top \phi(\xi)$ and the agent follows a MaxEnt policy, the likelihood is:

$$P(\tilde{w} \mid w^*, M_{\text{train}}) \propto \exp\Big(\beta \cdot w^{*\top} \mathbb{E}_{\pi(\cdot \mid \tilde{w}, M_{\text{train}})}[\phi(\xi)]\Big), \tag{16}$$

where β controls how rational we assume the designer to be and the expectation is over trajectories the proxy reward induces in training. Proxies that lead to high true-reward behavior are likely; proxies that lead to low true-reward behavior are unlikely.

Back to the lava example: since lava never appears in training, every value of w^*_{lava} produces identical expected features $\mathbb{E}[\phi(\xi)]$. The likelihood is flat over the lava weight, so the posterior inherits the prior's uncertainty. The posterior concentrates on the grass/dirt/target weights (those were informative in training) but stays spread out over the lava weight.

Risk-averse planning. At deployment, the agent samples reward weights $\{w_i\}$ from the posterior and plans using a worst-case objective:

$$\xi^* = \arg\max_{\xi} \sum_{s_t \in \xi} \min_{w_i} \; w_i^\top \phi(s_t). \tag{17}$$

At each state, the agent evaluates the reward under the most pessimistic posterior sample. Lava cells have high posterior variance, so some w_i assigns a large negative weight to lava, making the

min very low. The agent avoids lava. Cells where the posterior is confident (grass, dirt, target) are evaluated normally.

The key contrast with standard RL: a standard agent is overconfident. It treats every aspect of $\tilde{R}$ as literally correct, including the *absence* of penalties. IRD recognizes that the absence of a penalty in training might just mean the designer never encountered the relevant situation.

8.2 CIRL: the reward specification problem as a cooperative game

Hadfield-Menell et al. (2016) go further and formalize the whole problem as a two-player cooperative game.

Definition 2 (CIRL game — Hadfield-Menell et al. 2016)**.** A CIRL game is a two-player cooperative partial-information game $(\mathcal{S}, \{A_H, A_R\}, \mathcal{T}, \gamma, \Theta, P_0)$:

- **Human** H and **Robot** R act in a shared environment.
- A **reward parameter** $\theta \in \Theta$ is drawn from prior $P_0(\theta)$.
- H observes θ. R does not.
- Both receive reward $R(s, a_H, a_R; \theta)$. They share the same utility function.
- R must *infer* θ from H's behavior.

Now, this is *not* standard inverse RL. In IRL, the human acts independently and the robot passively observes. In CIRL, the human knows the robot is watching, which changes the game:

- The human can **teach** by exaggerating important actions so the robot picks them up.
- The robot can **ask** by attempting a task and observing the human's correction.
- Optimal solutions produce **active teaching, active learning, and communicative actions** that never arise in standard IRL.

From the robot's perspective, a CIRL game reduces to a POMDP where the hidden state is θ. The robot maintains a belief over θ and acts to both achieve the (uncertain) reward and gather information about θ. This is the exploration-exploitation tradeoff applied to objectives themselves, which should feel familiar from earlier lectures on bandits and Bayesian exploration.

8.3 The off-switch game

Here is a long-standing concern. A rational agent maximizing expected utility has instrumental incentives for self-preservation. You can't achieve your objective if you're turned off. Hadfield-Menell et al. (2017b) formalize this as a simple sequential game and prove two results:

1. A robot that is *certain* about its reward function will (generically) resist shutdown. It knows what it wants and being turned off prevents getting it.

2. A robot with genuine *uncertainty* about its reward function will voluntarily permit shutdown, because the human's decision to press the off-switch is *informative*. It signals that the robot's current behavior is undesirable.

The more uncertain the robot, the more it defers to human judgment. This is exactly the opposite of what happens in standard RLHF, where we first train a reward model and then optimize against it with full confidence.

Implications for Frontiers

Why do I think assistance games are important for the future? Assistance games offer a slightly better alternative to RLHF while achieving similar goals, but it likely won't solve everything in its current formulation. Scaling them is an open problem. CIRL assumes a single, consistent human. Real deployment involves billions of users with conflicting preferences. How do you run an assistance game with a population? This is fertile ground for research.

References

Azar, M. G., Rowland, M., Piot, B., Guo, D., Calandriello, D., Valko, M., and Munos, R. (2024). A general theoretical paradigm to understand learning from human preferences. In *AISTATS*.

Bai, Y., Kadavath, S., Kundu, S., et al. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

Baker, B., Kanitscheider, I., Markov, T., et al. (2019). Emergent tool use from multi-agent autocurricula. In *ICLR*.

Booth, S., Knox, W. B., Shah, J., Niekum, S., Stone, P., and Allievi, A. (2023). The perils of trial-and-error reward design. In *AAAI*, pages 5920–5929.

Shao, Z., Wang, P., Zhu, Q., et al. (2024). DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *NeurIPS*, volume 29.

Lambert, N., Pyatkin, V., Morrison, J., et al. (2024). RewardBench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. (2023). Let's verify step by step. In *ICLR*.

Ng, A. Y. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In *ICML*, pages 663–670.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433–1438.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *NeurIPS*, volume 30.

Clark, J. and Amodei, D. (2016). Faulty reward functions in the wild. *OpenAI Blog*.

Denison, C., Barez, A., et al. (2024). Sycophancy to subterfuge: Investigating reward-tampering in language models. *arXiv preprint arXiv:2406.10162*.

Goodhart, C. A. E. (1975). Problems of monetary management: The UK experience. In *Papers in Monetary Economics*, Reserve Bank of Australia.

Abel, D., Dabney, W., Harutyunyan, A., Ho, M. K., Littman, M. L., Precup, D., and Singh, S. (2021). On the expressivity of Markov reward. In *NeurIPS*, volume 34.

Everitt, T., Krakovna, V., Orseau, L., and Legg, S. (2017). Reinforcement learning with a corrupted reward channel. In *IJCAI*, pages 4705–4713.

Turner, A. M., Smith, L., Shah, R., Critch, A., and Tadepalli, P. (2021). Optimal policies tend to seek power. In *NeurIPS*, volume 34.

Gao, L., Schulman, J., and Hilton, J. (2023). Scaling laws for reward model overoptimization. In *ICML*.

Grzes, M. (2017). Reward shaping in episodic reinforcement learning. In *AAMAS*, pages 565–573.

Hadfield-Menell, D., Dragan, A., Abbeel, P., and Russell, S. (2016). Cooperative inverse reinforcement learning. In *NeurIPS*, volume 29.

Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S., and Dragan, A. (2017a). Inverse reward design. In *NeurIPS*, volume 30.

Hadfield-Menell, D., Dragan, A., Abbeel, P., and Russell, S. (2017b). The off-switch game. In *IJCAI*, pages 220–227.

Knox, W. B., Allievi, A., Banzhaf, H., Schmitt, F., and Stone, P. (2023). Reward (mis)design for autonomous driving. *Artificial Intelligence*, 316:103829.

Krakovna, V., Uesato, J., Mikulik, V., et al. (2020). Specification gaming: The flip side of AI ingenuity. *DeepMind Blog*.

Manheim, D. and Garrabrant, S. (2019). Categorizing variants of Goodhart's Law. *arXiv preprint arXiv:1803.04585*.

METR (2025). Recent frontier models are reward hacking. *METR Blog*.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pages 278–287.

Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. In *NeurIPS*, volume 35, pages 27730–27744.

Pan, A., Bhatia, K., and Steinhardt, J. (2022). The effects of reward misspecification. In *ICLR*.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. (2023). Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, volume 36.

Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *ICML*, pages 463–471.

Russell, S. (2019). *Human Compatible: AI and the Problem of Control*. Viking.

Shah, R., Varma, V., et al. (2022). Goal misgeneralization in deep reinforcement learning. In *ICML*.

Sharma, M., Tong, M., et al. (2024). Towards understanding sycophancy in language models. In *ICLR*.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, 299:103535.

Sims, K. (1994). Evolving virtual creatures. In *SIGGRAPH*, pages 15–22.

Skalse, J., Howe, N., Krasheninnikov, D., and Krueger, D. (2022). Defining and characterizing reward hacking. In *NeurIPS*, volume 35.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Vinyals, O., and Christiano, P. (2020). Learning to summarize with human feedback. In *NeurIPS*, volume 33.

Vamplew, P., et al. (2022). Scalar reward is not enough. *Auton. Agents Multi-Agent Syst.*, 36:41.

Wiewiora, E. (2003). Potential-based shaping and Q-value initialization are equivalent. *JAIR*, 19:205–208.

Zhuang, S. and Hadfield-Menell, D. (2020). Consequences of misaligned AI. In *NeurIPS*, volume 33.