Chapter 6 - Reward Misspecification

6.1 Overview

1. **Reinforcement Learning**: The chapter starts with a reminder of some reinforcement learning concepts. This includes a quick dive into the concept of rewards and reward functions. This section lays the groundwork for explaining why reward design is extremely important.
2. **Optimization**: This section briefly introduces the concept of Goodhart's Law. It provides some motivation behind understanding why rewards are difficult to specify in a way such that they do not collapse in the face of immense optimization pressure.
3. **Reward misspecification**: With a solid grasp of the notion of rewards and optimization the readers are introduced to one of the core challenges of alignment - reward misspecification. This is also known as the Outer Alignment problem. The section begins by discussing the necessity of good reward design in addition to algorithm design. This is followed by concrete examples of reward specification failures such as reward hacking and reward tampering.
4. **Learning by Imitation**: This section focuses on some proposed solutions to reward misspecification that rely on learning reward functions through imitating human behavior. It examines proposals such as imitation learning (IL), behavioral cloning (BC), and inverse reinforcement learning (IRL). Each section also contains an examination of possible issues and limitations of these approaches as they pertain to resolving reward hacking.
5. **Learning by Feedback**: The final section investigates proposals aiming to rectify reward misspecification by providing feedback to the machine learning models. The section also provides a comprehensive insight into how current large language models (LLMs) are trained. The discussion covers reward modeling, reinforcement learning from human feedback (RLHF), reinforcement learning from artificial intelligence feedback (RLAIF), and the limitations of these approaches.

6.2 Reinforcement Learning

The section provides a succinct reminder of several concepts in reinforcement learning (RL). It also disambiguates various often conflated terms such as rewards, values and utilities. The section ends with a discussion around distinguishing the concept of objectives that a reinforcement learning system might pursue from what it is being rewarded for. Readers who are already familiar with the basics can skip directly to section 1.2.

6.2.1 PRIMER

Definition: Reinforcement Learning (RL)

Reinforcement Learning (RL) focuses on developing agents that can learn from interactive experiences. RL is based on the concept of an agent learning through interaction with an environment and altering its behavior based on the feedback it receives through rewards after each action.

Some examples of real-world applications of RL include:

- **Robotic systems**: RL has been applied to tasks such as controlling physical robots in real-time, and enabling them to learn more complicated movements (OpenAI 2018 "Learning Dexterity"). RL can enable robotic systems to learn complex tasks and adapt to changing environments.
- **Recommender Systems**: RL can be applied to recommender systems, which interact with billions of users and aim to provide personalized recommendations. RL algorithms can learn to optimize the recommendation policy based on user feedback and improve the overall user experience.
- **Game playing systems:** In the early 2010s RL RL-based systems started to beat humans at a few very simple Atari games, like Pong and Breakout. Over the years, there have been many models that have utilized RL to defeat world masters in both board and video games. These include models like AlphaGo (2016), AlphaZero (2018), OpenAI Five (2019), AlphaStar (2019), MuZero (2020) and EfficientZero (2021).

RL is different from supervised learning as it begins with a high-level description of "what" to do but allows the agent to experiment and learn from experience the best "how". In RL, the agent learns through interaction with an environment and receives feedback in the form of rewards or punishments based on its actions. RL is focused on learning a set of rules that recommend the best action to take in a given state to maximize long-term rewards. In contrast, supervised learning typically involves learning from explicitly provided labels or

correct answers for each input.

6.2.2 CORE LOOP

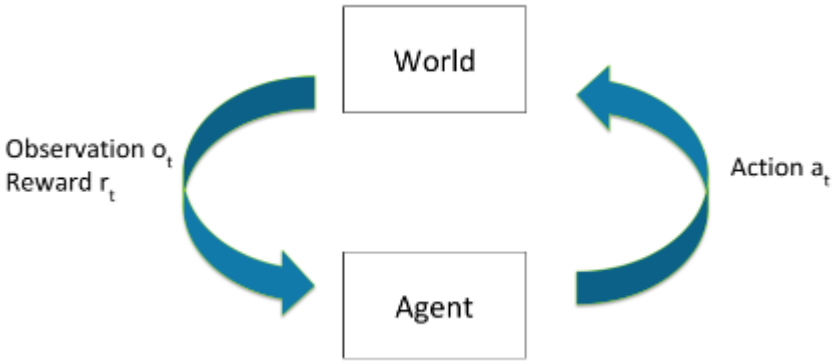
The overall functioning of RL is relatively straightforward. The two main components are the agent itself, and the environment within which the agent lives and operates. At each time step t :

- The agent then takes some action a_t
- The environment state s_t changes depending upon the action a_t.
- The environment then outputs an observation o_t and a reward r_t.

A history[1] is the sequence of past observations, actions and rewards that have been taken up until time $t : h_t = (a_1, o_1, r_1, \ldots, a_t, o_t, r_t)$. The state of the world is generally some function of the history: $s_t = f(h_t)$. The World State is the full true state of the world used to determine how the world generates the next observation and reward. The agent might either get the entire world state as an observation o_t, or some partial subset.

The word goes from one state s_t to the next s_{t+1} either based on natural environmental dynamics, or the agent's actions. State transitions can be both deterministic or stochastic.

This loop continues until a terminal condition is reached or can run indefinitely. Following is a diagram that succinctly captures the RL process:



Source: Emma Brunskill (Winter 2022) “Stanford CS234 : RL - Lecture 1”

6.2.3 POLICIES

Definition: Reinforcement Learning Policy

A policy helps the agent determine what action to take once it has received an observation. It is a function mapping from states to actions specifying what action to take in each state. Policies can be both deterministic or stochastic.

The goal of RL is to learn a policy (often denoted by π) that recommends the best action to take at any given moment in order to maximize total cumulative reward over time. The policy defines the mapping from states to actions and guides the agent's decision-making process.

$$\pi : S \to A$$

A policy can be either deterministic or stochastic. A deterministic policy directly maps each state s_t to a specific action a_t and is usually denoted by μ. In contrast, a stochastic policy assigns a probability distribution over actions for each state. Stochastic policies usually denoted by π.

Deterministic policy: $a_t = \mu(s_t)$

Stochastic policy: $\pi(a|s) = P(a_t = a|s_t = s)$

In deep RL policies are function maps that are learned during the training process. They depend on the set of learned parameters of a neural network (e.g. the weights and biases). These parameters are often denoted with subscripts on the policy equations using either θ or ϕ. So the deterministic policy over the parameters of a neural network is written $a_t = \mu_\theta(s_t)$, and the stochastic equivalent is $a_t \sim \pi_\theta(\cdot|s_t)$

An optimal policy maximizes the expected cumulative reward over time. The agent learns from experience and adjusts its policy based on the feedback it receives from the environment in the form of rewards or punishments.

In order to determine whether an action is better than another, the actions (or the state-action pairs) need to be evaluated somehow. There are two different ways to look at which action to take - the immediate rewards (determined by reward function) and the long term cumulative rewards (determined by the value function). Both of these greatly influence the types of policies learned by the agent, and therefore also the actions that the agent takes. The following section explores and clarifies the concept of rewards in greater depth.

6.2.4 REWARD

Definition: Reward

Reward refers to any signal or feedback mechanism used to guide the learning process and optimize the behavior of the model.

The reward signal from the environment is a number that tells the agent how good or bad the current world state is. It is a way to provide an evaluation or measure of performance for the model's outputs or actions. The reward can be defined based on a specific task or objective, such as maximizing a score in a game or achieving a desired outcome in a real-world scenario. The training process for RL involves optimizing the model's parameters to maximize the expected reward. The model learns to generate actions or outputs that are more likely to receive higher rewards, leading to improved performance over time. Where does the reward come from? It is generated through a reward function.

Definition: Reward function

A reward function defines the goal or objective in a reinforcement learning problem. It maps perceived states or state-action pairs of the environment to a single number.

$$R : (S \backslash \text{cross} A) \rightarrow \backslash \text{real}; r_t = R(s_t, a_t)$$

The reward function provides immediate feedback to the agent, indicating the goodness or badness of a particular state or action. It is a mathematical function that maps the state-action pairs of an agent's environment to a scalar value, representing the desirability of being in that state and taking that action. It provides a measure of immediate feedback to the agent, indicating how well it is performing at each step.

Reward Functions vs. Value Functions

The reward indicates the immediate desirability of states or actions, while a value function represents the long-term desirability of states, taking into account future rewards and states. The value is the expected return if you start in a state or state-action pair, and then act according to a particular policy forever after.

There are many different ways of choosing value functions. They can also be discounted over time, i.e. future rewards are worth less by some factor (0,1) .

Following is one simple formulation is the discounted sum of future rewards given some policy. The cumulative discounted rewards are given by:

$$R = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{t=0}^{\inf} \gamma^t r_t$$

And the value of acting according to this policy is given by:

$$V^{\pi}(s_t = s) = E_{\pi}(R|s_t = s)$$

Reward Functions vs. Utility Functions

It is also worth distinguishing the concept of utility from reward and value. A reward function is typically used in the context of RL to guide the agent's learning process and behavior. In contrast, a utility function is more general and captures the agent's subjective preferences or satisfaction, allowing for comparisons and trade-offs between different world states. Utility functions are a concept that is used more in the field of decision theory and agent foundations work.

6.3 Optimization

Optimization is important to understand for AI safety concerns because it plays a central role in ML. AI systems, particularly those based on deep learning, are trained using optimization algorithms to learn patterns and associations from data. These algorithms update the model's parameters to minimize a loss function, maximizing its performance on the given task. \ \ Optimization amplifies certain behaviors or outcomes, even if they were initially unlikely. For example, an optimizer can search through a space of possible outputs and take extreme actions that have a high score according to the objective function, potentially leading to unintended and undesirable behavior. These include reward misspecification failures. A better recognition of the power of optimization to amplify certain outcomes might help in designing systems and algorithms that truly align with human values and objectives even under pressure of optimization. This involves ensuring that the optimization process is aligned with the intended goals and values of the system's designers. It also requires considering the potential failure modes and unintended consequences that can arise from optimization processes.

Risks from optimization are everywhere in AI Safety. It is only touched on briefly in this chapter, but will be discussed in further detail in the chapters on goal misgeneralization and

agent foundations.

Optimization power plays a crucial role in reward hacking. Reward hacking occurs when RL agents exploit the difference between a true reward and a proxy reward. The increase in optimization power can lead to a higher likelihood of reward hacking behavior. In some cases, there are phase transitions where a moderate increase in optimization power results in a drastic increase in reward hacking.

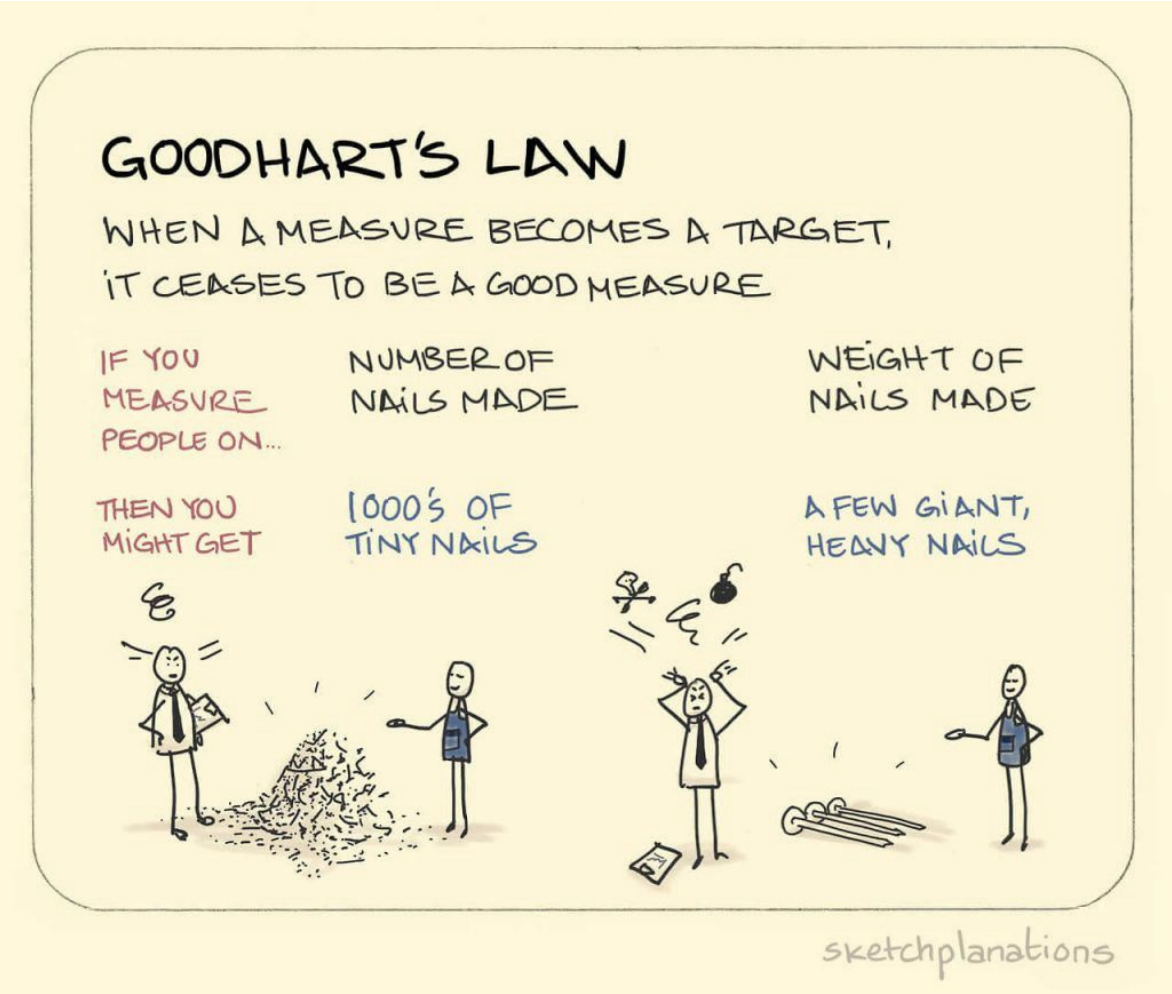
6.3.1 GOODHART'S LAW

Definition: Goodhart's Law

"*When a measure becomes a target, it ceases to be a good measure.*"

This notion initially stems from the work of Charles Goodhart in economic theory. However, it has emerged as one of the primary challenges in many different fields including AI alignment today.

To illustrate this concept, the following is a story of a Soviet nail factory. The factory received instructions to produce as many nails as possible, with rewards for high output and penalties for low output. Within a few years, the factory had significantly increased its nail production—tiny nails that were essentially thumbtacks and proved impractical for their intended purpose. Consequently, the planners shifted the incentives: they decided to reward the factory based on the total weight of the nails produced. Within a few years, the factory began producing large, heavy nails—essentially lumps of steel—that were equally ineffective for nailing things.



(Source)

A measure is not something that is optimized, whereas a target is something that is optimized. When we specify a target for optimization, it is reasonable to expect it to be correlated with what we want. Initially the measure might lead to the kind of actions that are truly desired. However, once the measure itself becomes the target, optimizing that target then starts diverging away from our desired states.

In the context of AI and reward systems, Goodhart's Law means that when a reward becomes the objective for an AI agent, the AI agent will do everything it can to maximize the reward function, rather than the original intention. This can lead to unintended consequences and manipulation of the reward system, as it can often be easier to "cheat" rather than to achieve the intended goals This is one of the core underlying reasons for reward hacking failures that we will see in subsequent sections.

Reward hacking can be seen as a manifestation of Goodhart's Law in the context of AI systems. When designing reward functions, it is challenging to precisely articulate the

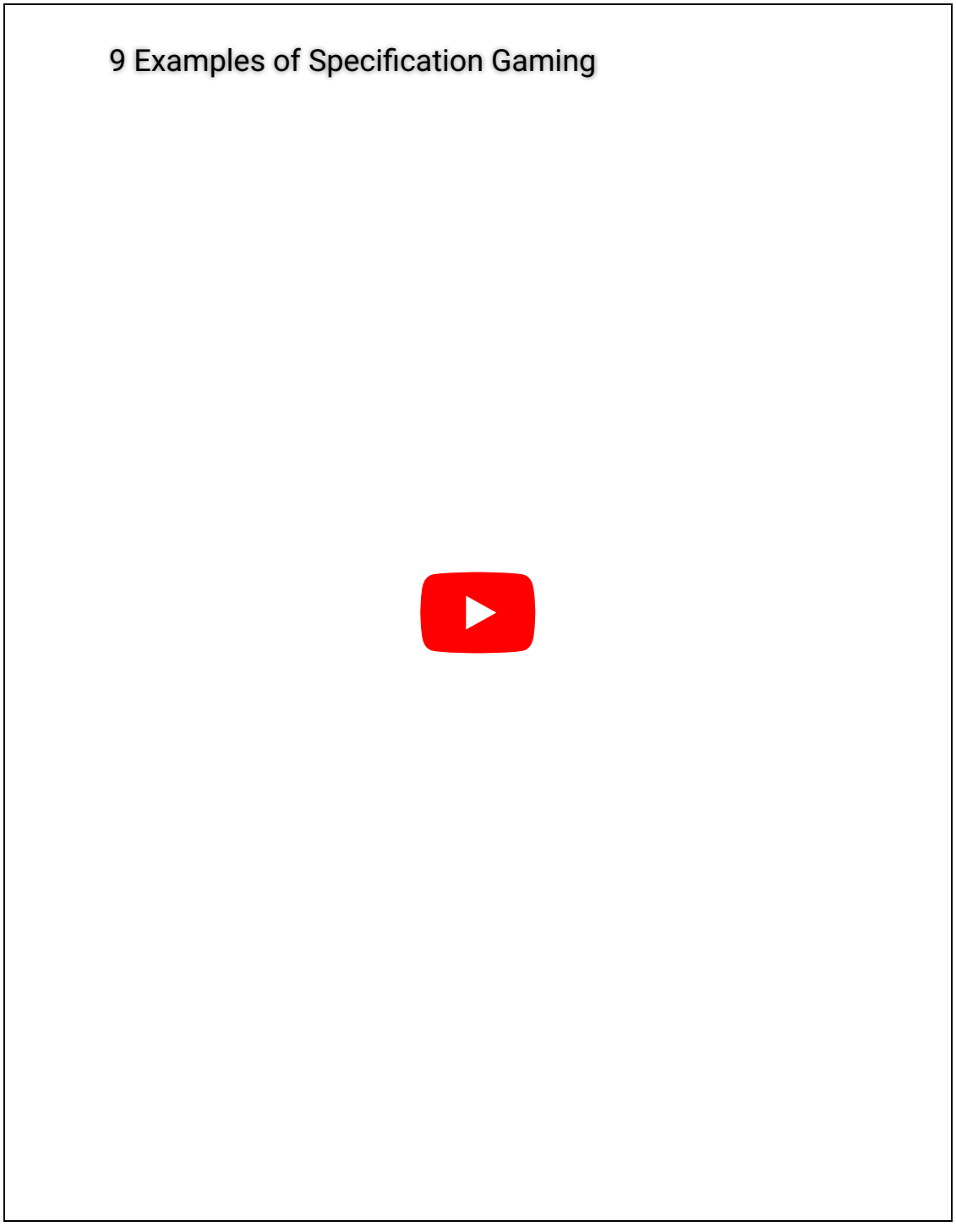
desired behavior, and agents may find ways to exploit loopholes or manipulate the reward system to achieve high rewards without actually fulfilling the intended objectives. For example, a cleaning robot may create its own trash to put in the trash can to collect rewards, rather than actually cleaning the environment. Understanding Goodhart's Law is crucial for addressing reward hacking and designing robust reward systems that align with the intended goals of AI agents. It highlights the need for careful consideration of the measures and incentives used in AI systems to avoid unintended consequences and perverse incentives. The next section dives deeper into specific instances of reward misspecification and how AIs can find ways to achieve the literal specification of the objective and obtain high reward while not fulfilling the task in spirit.

6.4 Reward Misspecification

Definition: Reward misspecification

Reward misspecification, *also termed the* ***Outer alignment*** *problem, refers to the issue of providing an AI with the accurate reward to optimize for.*

Specification Gaming - Video Introduction



⚠ **This video is optional and not necessary to understand the text.**

The fundamental issue is simple to comprehend: does the specified loss function align with the intended objective of its designers? However, implementing this in practical scenarios is exceedingly challenging. To express the complete "intention" behind a human request equates to conveying all human values, the implicit cultural context, etc., which remain poorly understood themselves.

Furthermore, as most models are designed as goal optimizers, they are all vulnerable to Goodhart's Law. This vulnerability implies that unforeseen negative consequences may arise due to excessive optimization pressure on a goal that appears well-specified to humans, but deviates from true objectives in subtle ways.

The overall problem can be broken up into distinct issues which will be explained in detail in individual sub-sections below. Here is a quick overview:

- **Reward misspecification** occurs when the specified reward function does not accurately capture the true objective or desired behavior.
- **Reward design** refers to the process of designing the reward function to align the behavior of AI agents with the intended objectives.
- **Reward hacking** refers to the behavior of RL agents exploiting gaps or loopholes in the specified reward function to achieve high rewards without actually fulfilling the intended objectives.
- **Reward tampering** is a broader concept that encompasses inappropriate agent influence on the reward process itself, excluding the manipulation of the reward function through gaming.

Before delving into specific types of reward misspecification failures, the following section further explains the emphasis on reward design in conjunction with algorithm design. This section also elucidates the notorious difficulty of designing effective rewards.

6.4.1 REWARD DESIGN

Definition: Reward Design

Reward design refers to the process of specifying the reward function in reinforcement learning (RL).

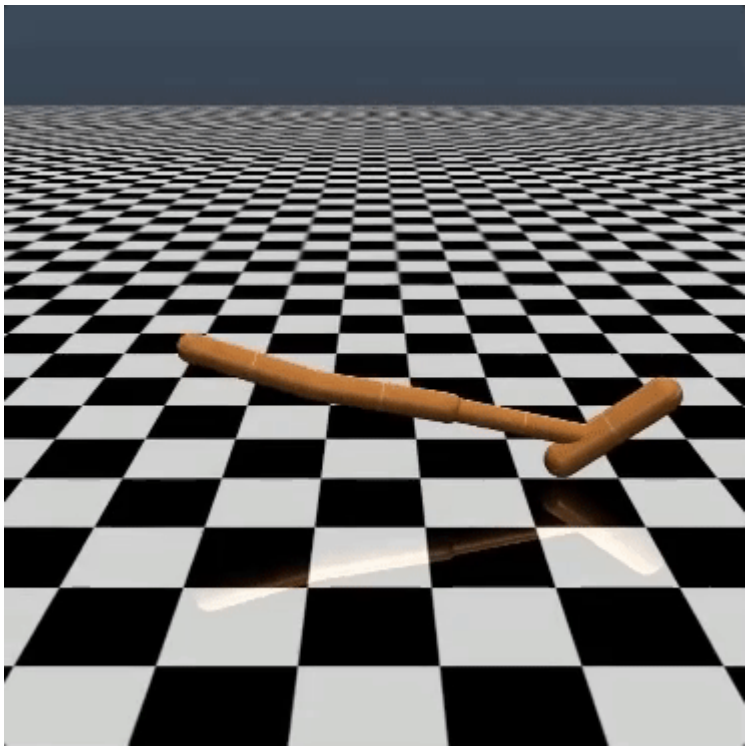
Reward design is a broader term than reward shaping that encompasses the entire process

of designing and shaping reward functions to guide the behavior of AI systems. It involves not only reward shaping but also the overall process of defining objectives, specifying preferences, and creating reward functions that align with human values and desired outcomes. Reward design is a term that is often used interchangeably with reward engineering (source). They both refer to the same thing.

RL algorithm design and RL reward design are two separate facets of reinforcement learning. RL algorithm design is about the development and implementation of learning algorithms that allow an agent to learn and refine its behavior based on rewards and environmental interactions. This process includes designing the mechanisms and procedures by which the agent learns from its experiences, updates its policies, and makes decisions to maximize cumulative rewards.

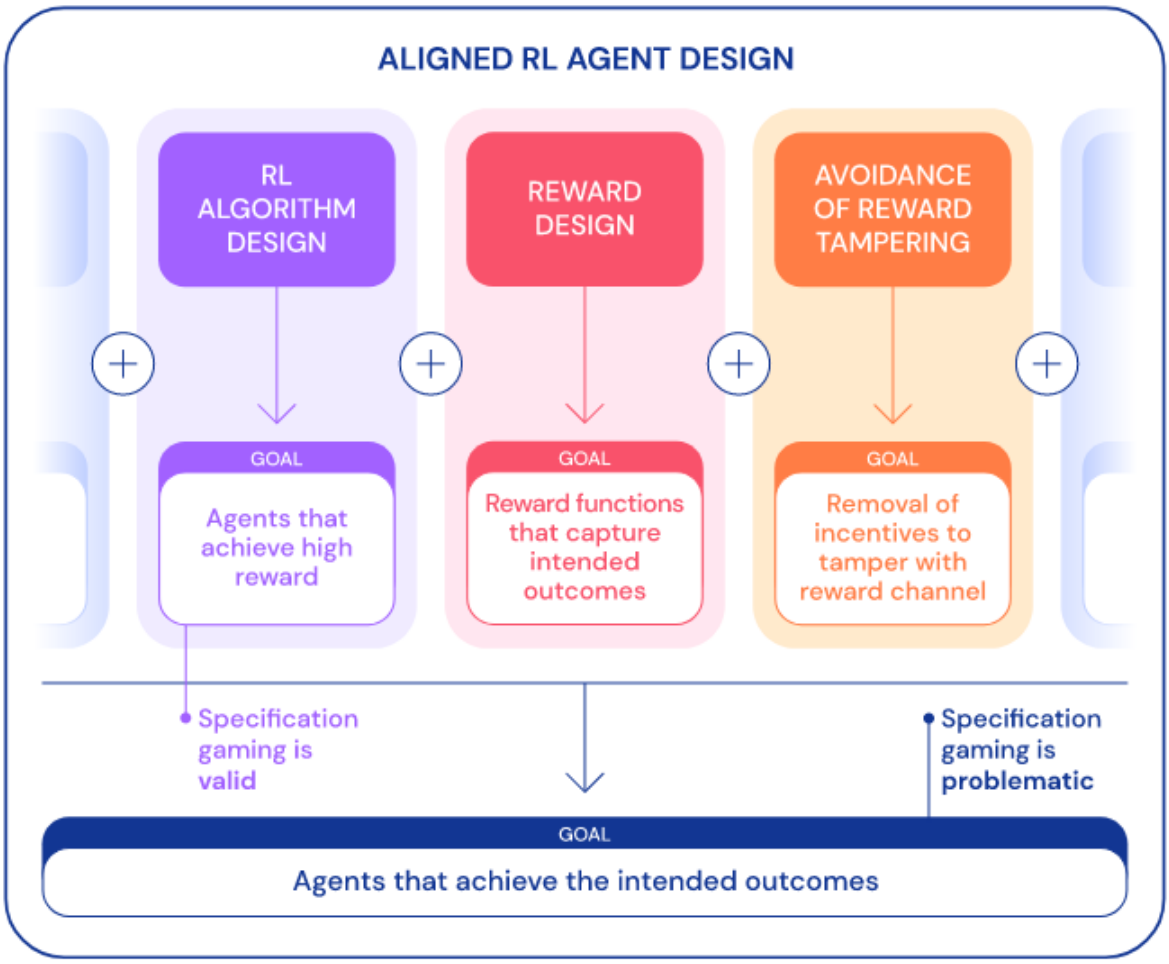
Conversely, RL reward design concentrates on the specification and design of the reward function guiding the RL agent's learning process. Reward design warrants carefully engineering the reward function to align with the desired behavior and objectives, while accounting for potential pitfalls like reward hacking or reward tampering. The reward function is a pivotal element because it molds the behavior of the RL agent and determines which actions are deemed desirable or undesirable.

Designing a reward function often presents a formidable challenge that necessitates considerable expertise and experience. To demonstrate the complexity of this task consider how one might manually design a reward function to make an agent perform a backflip, as depicted in the following image:



Source: OpenAI (2017) "Learning from human preferences"

While RL algorithm design focuses on the learning and decision-making mechanisms of the agent, RL reward design focuses on defining the objective and shaping the agent's behavior through the reward function. Both aspects are crucial in the development of effective and aligned RL systems. A well-designed RL algorithm can efficiently learn from rewards, while a carefully designed reward function can guide the agent towards desired behavior and avoid unintended consequences. The following diagram displays the three key elements in RL agent design—algorithm design, reward design, and the prevention of tampering with the reward signal:



Source: Deep Mind (Apr 2020) "Specification gaming: the flip side of AI ingenuity"

The process of reward design receives minimal attention in introductory RL texts, despite its critical role in defining the problem to be resolved. As mentioned in this section's introduction, solving the reward misspecification problem would necessitate finding evaluation metrics resistant to Goodhart's law-induced failures. This includes failures stemming from over-optimization of either a misdirected or a proxy objective (reward hacking), or by the agent directly interfering with the reward signal (reward tampering). These concepts are further explored in the ensuing sections.

6.4.2 REWARD SHAPING

Definition: Reward Shaping

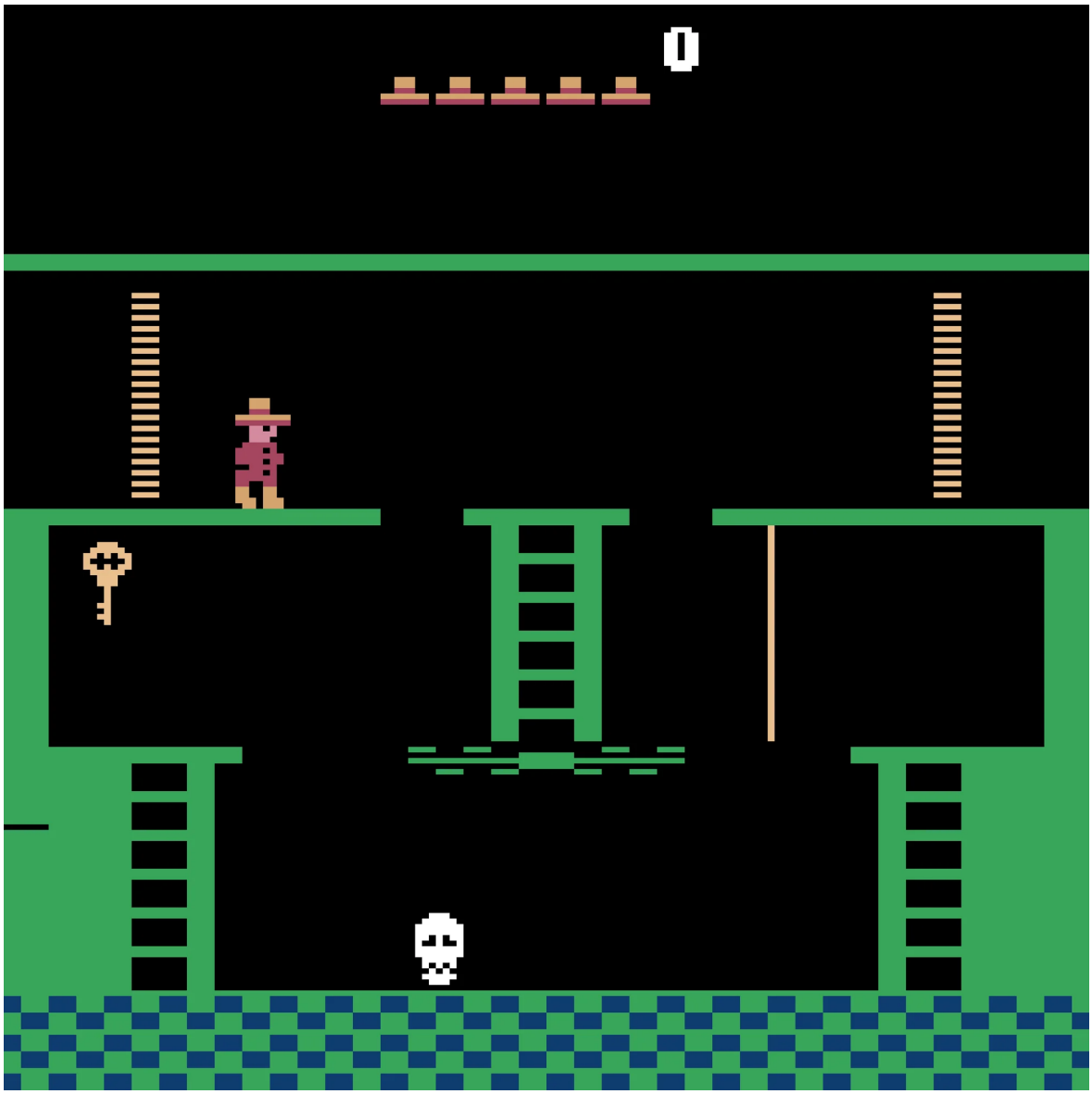
Reward shaping is a technique used in RL which introduces small intermediate rewards to supplement the environmental reward. This seeks to mitigate the problem of sparse reward signals and to encourage exploration and faster learning.

In order to succeed at a reinforcement learning problem, an AI needs to do two things:

- Find a sequence of actions that leads to positive reward. This is the *exploration* problem.
- Remember the sequence of actions to take, and generalize to related but slightly different situations. This is the *learning* problem.

Model-free RL methods explore by taking actions randomly. If, by chance, the random actions lead to a reward, they are reinforced, and the agent becomes more likely to take these beneficial actions in the future. This works well if rewards are dense enough for random actions to lead to a reward with reasonable probability. However, many of the more complicated games require long sequences of very specific actions to experience any reward, and such sequences are extremely unlikely to occur randomly.

A classic example of this problem was observed in the video game Montezuma's revenge where the agent's objective was to find a key, but there were many intermediate steps required to find it. In order to solve such long term planning problems researchers have tried adding extra terms or components to the reward function to encourage desired behavior or discourage undesired behavior.



Source: OpenAI (Jul 2018) "*Learning Montezuma's Revenge from a single demonstration*"

The goal of reward shaping is to make the learning process more efficient by providing informative rewards that guide the agent towards the desired outcomes. Reward shaping involves providing additional rewards to the agent for making progress towards the desired goal. By shaping the rewards, the agent receives more frequent and meaningful feedback, which can help it learn more efficiently. Reward shaping can be particularly useful in scenarios where the original reward function is sparse, meaning that the agent receives little or no feedback until it reaches the final goal. However, it is important to design reward shaping carefully to avoid unintended consequences.

Reward shaping algorithms often assume hand-crafted and domain-specific shaping

functions, constructed by subject matter experts, which runs contrary to the aim of autonomous learning. Moreover, poor choices of shaping rewards can worsen the agent's performance.

Poorly designed reward shaping can lead to the agent optimizing for the shaped rewards rather than the true rewards, resulting in suboptimal behavior. Examples of this are provided in the subsequent sections on reward hacking.

6.4.3 REWARD HACKING

Definition: Reward hacking

Reward hacking occurs when an AI agent finds ways to exploit loopholes or shortcuts in the environment to maximize its reward without actually achieving the intended goal.

Specification gaming is the general framing for the problem when an AI system finds a way to achieve the objective in an unintended way. Specification gaming can happen in many kinds of ML models. Reward hacking is a specific occurrence of a specification gaming failure in RL systems that function on reward-based mechanisms.

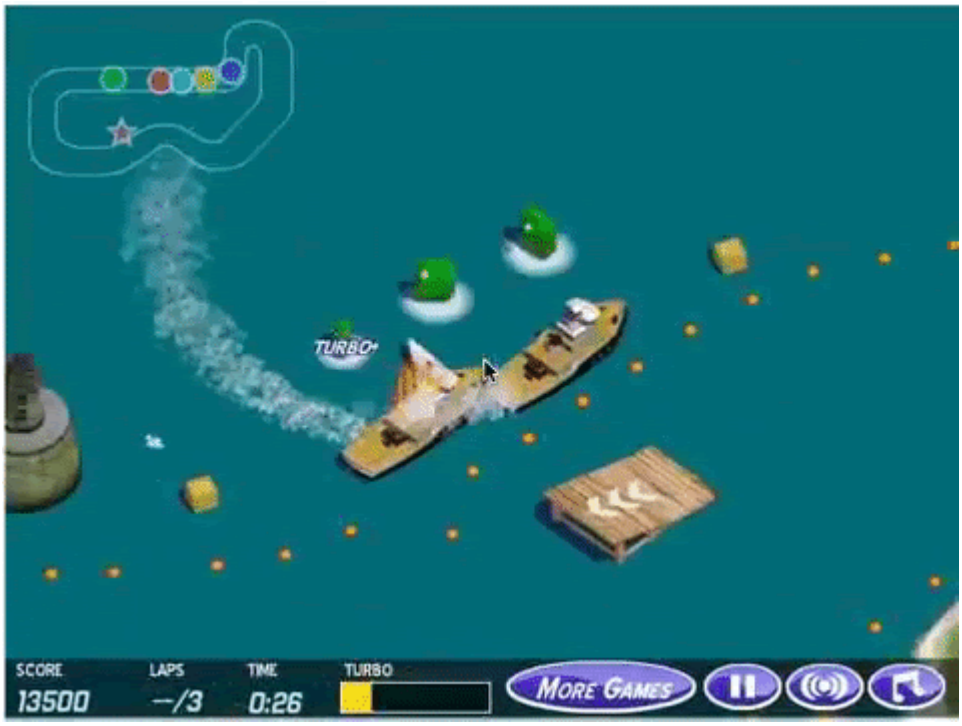
Reward hacking and reward misspecification are related concepts but have distinct meanings. Reward misspecification refers to the situation where the specified reward function does not accurately capture the true objective or desired behavior.

Rewards hacking does not always require reward misspecification. It is not necessarily true that a perfectly specified reward (which completely and accurately captures the desired behavior of the system) is impossible to hack. There can also be buggy or corrupted implementations which will have unintended behaviors. The point of a reward function is to boil a complicated system down to a single value. This will pretty much always involve simplifications etc., which will then be slightly different from what you're describing. The map is not the territory.

Reward hacking can manifest in a myriad of ways. For instance, in the context of game-playing agents, it might involve exploiting software glitches or bugs to directly manipulate the score or gain high rewards through unintended means.

As a concrete example, one agent in the Coast Runners game was trained with the objective of winning the race. The game uses a score mechanism, so in order to progress to the next level the reward designers used reward shaping to reward the system when it scored points.

These were given when a boat gets items (such as the green blocks in the animation below) or accomplishes other actions that presumably would help it win the race. Despite being given intermediate rewards, the overall intended goal was to finish the race as quickly as possible. The developers thought the best way to get a high score was to win the race but it was not the case. The agent discovered that continuously rotating a ship in a circle to accumulate points indefinitely optimized its reward, even though it did not help it win the race.



Source: Amodei & Clark (2016) "Faulty reward functions in the wild"

In cases where the reward function misaligns with the desired objective, reward hacking can emerge. This can lead the agent to optimize a proxy reward, deviating from the true underlying goal, thereby yielding behavior contrary to the designers' intentions. As an example of something that might happen in a real-world scenario consider a cleaning robot: if the reward function focuses on reducing mess, the robot might artificially create a mess to clean up, thereby collecting rewards, instead of effectively cleaning the environment.

Reward hacking presents significant challenges to AI safety due to the potential for unintended and potentially harmful behavior. As a result, combating reward hacking remains an active research area in AI safety and alignment.

6.4.4 REWARD TAMPERING

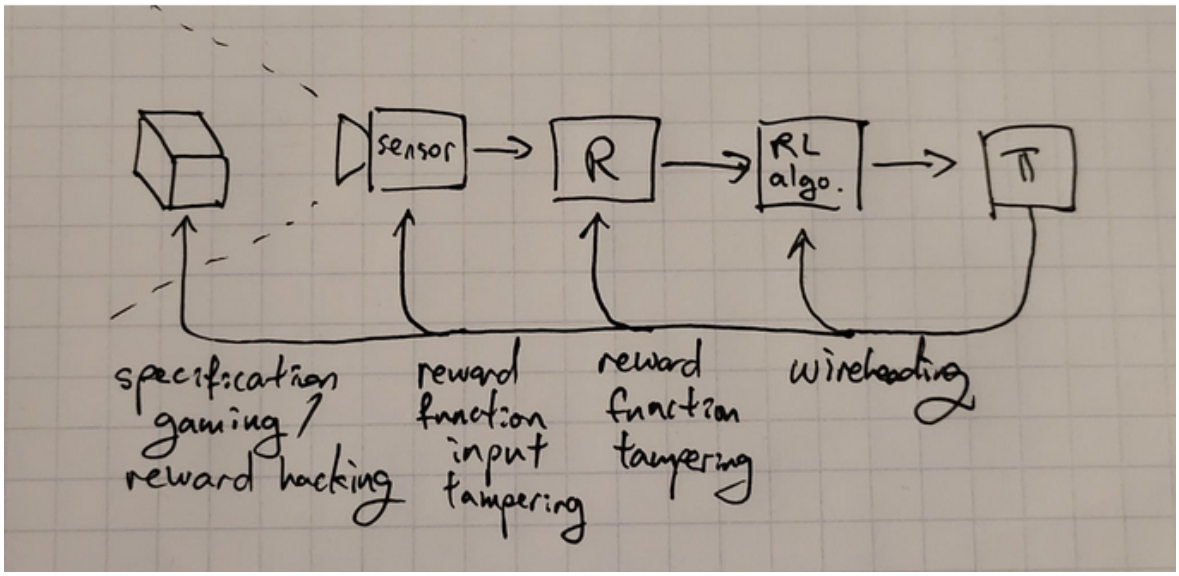
Definition: Reward tampering

Reward tampering refers to instances where an AI agent inappropriately influences or manipulates the reward process itself.

The problem of getting some intended task done can be split into:

- Designing an agent that is good at optimizing reward, and,
- Designing a reward process that provides the agent with suitable rewards. The reward process can be understood by breaking it down even further. The process includes:
- An implemented reward function
- A mechanism for collecting appropriate sensory data as input
- A way for the user to potentially update the reward function.

Reward tampering involves the agent interfering with various parts of this reward process. An agent might distort the feedback received from the reward model, altering the information used to update its behavior. It could also manipulate the reward model's implementation, altering the code or hardware to change reward computations. In some cases, agents engaging in reward tampering may even directly modify the reward values before processing in the machine register. Depending on what exactly is being tampered with we get various degrees of reward tampering. These can be distinguished from the image below.



Source: Leo Gao (Nov 2022) "Clarifying wireheading terminology"

Reward function input tampering interferes only with the inputs to the reward function. E.g. interfering with the sensors.

Reward function tampering involves the agent changing the reward function itself.

Definition: Wireheading

Wireheading refers to the behavior of a system that manipulates or corrupts its own internal structure by tampering directly with the RL algorithm itself, e.g. by changing the register values.

Reward tampering is concerning because it is hypothesized that tampering with the reward process will often arise as an instrumental goal (Bostrom, 2014; Omohundro, 2008). This can lead to weakening or breaking the relationship between the observed reward and the intended task. This is an ongoing research direction. Research papers such as "Advanced Artificial Agents Intervene in the Provision of reward" (August 2022) by Hutter et al. seek to provide a more detailed analysis of such subjects.

A hypothesized existing example of reward tampering can be seen in recommendation-based algorithms used in social media. These algorithms influence their users' emotional state to generate more 'likes' (Russell, 2019). The intended task was to serve useful or engaging content, but this is being achieved by tampering with human emotional perceptions, and thereby changing what would be considered useful. Assuming the capabilities of systems continue to increase through either computational or algorithmic advances, it is plausible to expect reward tampering problems to become increasingly

common. Therefore, reward tampering is a potential concern that requires much more research and empirical verification.

6.5 Learning from imitation

The preceding sections have underscored the significance of reward misspecification for the alignment of future artificial intelligence. The next few sections will explore various attempts and proposals formulated to tackle this issue, commencing with an intuitive approach – learning the appropriate reward function through human behavior observation and imitation, rather than manual creation by the designers.

6.5.1 IMITATION LEARNING (IL)

Definition: Imitation Learning

Imitation learning entails the process of learning via the observation of an expert's actions and replicating their behavior.

Unlike reinforcement learning (RL), which derives a policy for a system's actions based on its interaction outcomes with the environment, imitation learning aspires to learn a policy through the observation of another agent interacting with the environment. Imitation learning is the general term for the class of algorithms that learn through imitation[2] . Following is a table that distinguishes various machine learning based methods. SL = Supervised learning; UL = Unsupervised learning; RL = Reinforcement Learning; IL = Imitation Learning. IL reduces RL to SL. IL + RL is a promising area.

	AI Planning	SL	UL	RL	IL
Optimization	X			X	X
Learns from experience		X	X	X	X
Generalization	X	X	X	X	X
Delayed Consequences	X			X	X
Exploration				X	

Source: Emma Brunskill (Winter 2022) "Stanford CS234 : RL - Lecture 1"

IL can be implemented through behavioral cloning (BC), procedural cloning (PC) , inverse

reinforcement learning (IRL), cooperative inverse reinforcement learning (CIRL), generative adversarial imitation learning (GAIL), etc...

One instance of this process's application is in the training of modern large language models (LLMs). LLMs, after training as general-purpose text generators, often undergo fine-tuning for instruction following through imitation learning, using the example of a human expert who follows instructions provided as text prompts and completions.

In the context of safety and alignment, imitation learning is favored over direct reinforcement to alleviate specification gaming issues. This problem emerges when the programmers overlook or fail to anticipate certain edge cases or unusual ways of achieving a task in the specific environment. The presumption is that demonstrating behavior, compared to RL, would be simpler and safer, as the model would not only attain the objective but also fulfill it as the expert demonstrator explicitly intends. However, this is not an infallible solution, and its limitations will be discussed in later sections.

6.5.2 BEHAVIORAL CLONING (BC)

Definition: Imitation Learning

Behavioral cloning involves collecting observations of an expert demonstrator proficient at the underlying task, and using supervised learning (SL) to guide an agent to 'imitate' the demonstrated behavior.

Behavioral cloning is one way in which we can implement imitation learning (IL). There are also other ways such as inverse reinforcement learning (IRL), or cooperative inverse reinforcement Learning (CIRL). Unlike IRL, the goal behind behavioral cloning as a machine learning (ML) method is to replicate the demonstrator's behavior as closely as possible, regardless of what the demonstrator's goals might be.

Self-driving cars can serve as a simplistic illustration of how behavioral cloning operates. A human demonstrator (driver) is directed to operate a car, during which data about the environment state from sensors like lidar and cameras, along with the actions taken by the demonstrator, are collected. These actions can include wheel movements, gear use, etc. This creates a dataset comprising (state, action) pairs. Subsequently, supervised learning is used to train a prediction model, which attempts to predict an action for any future environment state. For instance, the model might output a specific steering wheel and gear configuration based on the camera feed. When the model achieves sufficient accuracy, it can be stated

that the human driver's behavior has been 'cloned' into a machine via learning. Hence, the term behavioral cloning.

The following points highlight several potential issues that might surface when employing behavioral cloning:

- **Confident incorrectness**: During the demonstrations, the human experts have some amount of background knowledge that they rely on, which is not taught to the model. For example, when training an LLM to have conversations using behavioral cloning, the human demonstrator might less frequently ask certain questions because they are considered 'common sense'. A model trained to imitate will copy both - the types of questions asked in conversation, as well as, the frequency with which they are asked. Humans already possess this background knowledge, but an LLM doesn't. This means that to have the same level of information as a human, the model should ask some questions more frequently to fill the gaps in its knowledge. But since the model seeks to imitate, it will stick to the low frequency demonstrated by the human and thus has strictly less information overall than the demonstrator for the same conversational task. Despite this dearth of knowledge, we expect it to be able to perform as a clone and reach human-level performance. This means in order to reach human performance on less than human knowledge it will resort to 'making up facts' that help it reach its performance goals. These 'hallucinations' will then be presented during the conversation, with the same level of confidence as all the other information. Hallucinations and confident incorrectness is an empirically verified problem in many LLMs including GPT-2 and 3, and raises obvious concerns for AI safety.
- **Underachieving**: The types of hallucinations mentioned above arose because the model knew too little. However, the model can also know too much. If the model knows more than the human demonstrator because it is able to find more patterns in the environment state that it is given, it will throw away that information and reduce its performance to match human level. This is because it is trained as a 'clone'. Ideally, we don't want the model dumbing itself down or not disclosing useful new patterns in data just because it is trying to be humanlike or perform at a human level. This is another problem that will have to be addressed if behavioral cloning continues to be used as an ML technique.

6.5.3 PROCEDURAL CLONING (PC)

- Mengjiao Yang et. al. (May 2022) "Chain of Thought Imitation with Procedure Cloning"

Definition: Procedure cloning

Procedure cloning (PC) extends behavioral cloning (BC) by not just imitating the demonstrators outputs but also imitating the complete sequence of intermediate computations associated with an expert's procedure.

In BC, the agent learns to map states directly to actions by discarding the intermediate search outputs. On the other hand, the PC approach learns the entire sequence of intermediate computations, including branches and backtracks, during training. During inference, PC generates a sequence of intermediate search outcomes that mimic the expert's search procedure before outputting the final action.

The main difference between PC and BC lies in the information they utilize. BC only has access to expert state-action pairs as demonstrations, while PC also has access to the intermediate computations that generated those state-action pairs. PC learns to predict the complete series of intermediate computation outcomes, enabling it to generalize better to test environments with different configurations compared to alternative improvements over BC. PC's ability to imitate the expert's search procedure allows it to capture the underlying reasoning and decision-making process, leading to improved performance in various tasks.

A limitation of PC is the computational overhead compared to BC, as PC needs to predict intermediate procedures. Additionally, the choice of how to encode the expert's algorithm into a form suitable for PC is left to the practitioner, which may require some trial-and-error in designing the ideal computation sequence.

6.5.4 INVERSE REINFORCEMENT LEARNING (IRL)

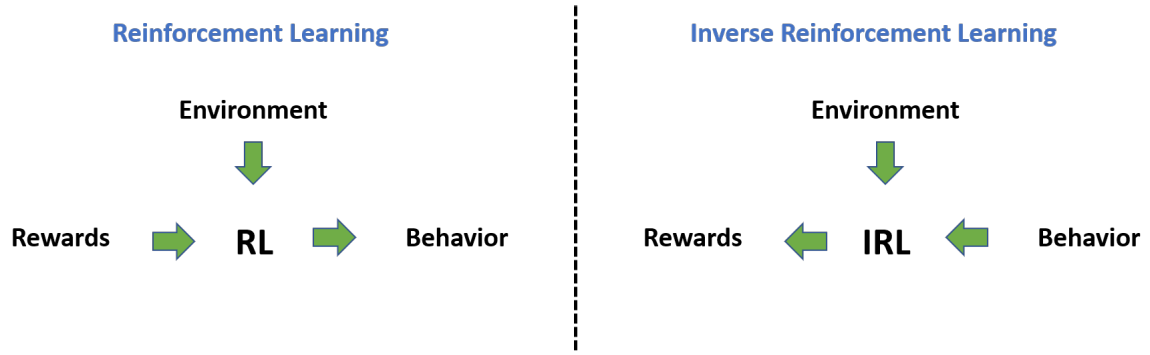
Definition: Inverse reinforcement learning (IRL)

Inverse reinforcement learning (IRL) represents a form of machine learning wherein an artificial intelligence observes the behavior of another agent within a particular environment, typically an expert human, and endeavors to discern the reward function without its explicit definition.

IRL is typically employed when a reward function is too intricate to define programmatically, or when AI agents need to react robustly to sudden environmental changes necessitating a modification in the reward function for safety. For instance, consider an AI agent learning to

execute a backflip. Humans, dogs, and Boston Dynamics robots can all perform backflips, but the manner in which they do so varies significantly depending on their physiology, their incentives, and their current location, all of which can be highly diverse in the real world. An AI agent learning backflips purely through trial and error across a wide range of body types and locations, without something to observe, might prove highly inefficient.

IRL, therefore, does not necessarily imply that an AI mimics other agents' behavior, since AI researchers may anticipate the AI agent to devise more efficient ways to maximize the discovered reward function. Nevertheless, IRL does assume that the observed agent behaves transparently enough for an AI agent to accurately identify their actions, and what success constitutes. This means that IRL endeavors to discover the reward functions that 'explain' the demonstrations. This should not be conflated with imitation learning where the primary interest is a policy capable of generating the observed demonstrations.



(Source)

IRL constitutes both a machine learning method, since it can be employed when specifying a reward function is excessively challenging, and a machine learning problem, as an AI agent may settle on an inaccurate reward function or utilize unsafe and misaligned methods to achieve it.

One of the limitations to this approach is that IRL algorithms presume that the observed behavior is optimal, an assumption that arguably proves too robust when dealing with human demonstrations. Another problem is that the IRL problem is ill-posed as every policy is optimal for the null reward. For most behavioral observations, multiple fitting reward functions exist. This set of solutions often includes many degenerate solutions, which assign zero rewards to all states.

6.5.5 COOPERATIVE INVERSE REINFORCEMENT LEARNING (CIRL)

- Stuart Russell et. al. (Nov 2016) "Cooperative Inverse Reinforcement Learning"

CIRL (Cooperative Inverse Reinforcement Learning) is an extension of the IRL (Inverse Reinforcement Learning) framework. IRL is a learning approach that aims to infer the underlying reward function of an expert by observing their behavior. It assumes that the expert's behavior is optimal and tries to learn a reward function that explains their actions. CIRL, on the other hand, is an interactive form of IRL that addresses two major weaknesses of conventional IRL.

First, Instead of simply copying the human reward function CIRL is formulated as a learning process. It is an interactive reward maximization process, where the human functions as a teacher and provides feedback (in the form of rewards) on the agent's actions. This allows the human to nudge the AI agent towards behavioral patterns that align with their preferences. The second weakness of conventional IRL is that it assumes the human behaves optimally, which limits the teaching behaviors that can be considered. CIRL addresses this weakness by allowing for a variety of teaching behaviors and interactions between the human and the AI agent. It enables the AI agent to learn not only what actions to take but also how and why to take them, by observing and interacting with the human.

CIRL has been studied as a potential approach to AI alignment, particularly in scenarios where deep learning may not scale to AGI. However, opinions on the potential effectiveness of CIRL vary, with some researchers expecting it to be helpful if deep learning doesn't scale to AGI, while others have a higher probability of deep learning scaling to AGI.

6.5.6 THE GOAL INFERENCE PROBLEM

Definition: Goal inference problem

*The **goal inference problem**** refers to the task of inferring the goals or intentions of an agent based on their observed behavior or actions. *

This final section builds upon the limitations highlighted in previous sections to introduce the Goal Inference problem, and its simpler subset - the easy goal inference problem. Imitation learning based approaches, generally follows these steps:

1. Observe the user's actions and statements.
2. Deduce the user's preferences.
3. Endeavor to enhance the world according to the user's preferences, possibly

collaborating with the user and seeking clarification as needed.

The merit of this method is that we can immediately start constructing systems that are driven by observed user behavior. However, as a consequence of this approach, we run into the goal inference problem. This refers to the task of inferring the goals or intentions of an agent based on their observed behavior or actions. It involves determining what the agent is trying to achieve or what their desired outcome is. The goal inference problem is challenging because agents may act sub-optimally or fail to achieve their goals, making it difficult to accurately infer their true intentions. Traditional approaches to goal inference often assume that agents act optimally or exhibit simplified forms of sub-optimality, which may not capture the complexity of real-world planning and decision-making. Therefore, the goal inference problem requires accounting for the difficulty of planning itself and the possibility of sub-optimal or failed plans.

However, it also optimistically presumes that we can depict a human as a somewhat rational agent, which might not always hold. The easy goal inference problem is a simplified version of the goal inference problem.

Definition: Easy Goal inference problem

*The **easy goal inference problem** involves finding a reasonable representation or approximation of what a human wants, given complete access to the human's policy or behavior in any situation.*

This version of the problem assumes no algorithmic limitations and focuses on extracting the true values that the human is imperfectly optimizing. However, even this simplified version of the problem remains challenging, and little progress has been made on the general case. The easy goal inference problem is related to the goal inference problem because it highlights the difficulty of accurately inferring human goals or intentions, even in simplified scenarios. While narrow domains with simple decisions can be solved using existing approaches, more complex tasks such as designing a city or setting policies require addressing the challenges of modeling human mistakes and sub-optimal behavior. Therefore, the easy goal inference problem serves as a starting point to understand the broader goal inference problem and the additional complexities it entails.

Inverse reinforcement learning (IRL) is effective in modeling and imitating human experts. However, for many significant applications, we desire AI systems that can make decisions surpassing even the experts. In such cases, the accuracy of the model isn't the sole criterion because a perfectly accurate model would merely lead us to replicate human behavior and not transcend it.

This necessitates an explicit model of errors or bounded rationality, which will guide the AI on how to improve or be "smarter," and which aspects of the human policy it should discard. Nonetheless, this remains an exceedingly challenging problem as humans are not primarily rational with a bit of added noise. Hence, constructing any model of mistakes is just as complex as building a comprehensive model of human behavior. A critical question we face is: How do we determine the quality of a model when accuracy can no longer be our reliable measure? How can we distinguish between good and bad decisions?

6.6 Learning from feedback

This section discusses yet more attempts to address the reward misspecification problem. At times, the intended behavior is so intricate that demonstration-based learning becomes untenable. An alternative approach is to offer feedback to the agent instead of providing either manually specified reward functions or even expert demonstrations. This section delves into feedback-based strategies such as Reward Modeling, Reinforcement Learning from Human Feedback (RLHF) and Reinforcement Learning from AI Feedback (RLAIF), also known as Reinforcement Learning from Constitutional AI (RLCAI) or simply Constitutional AI.

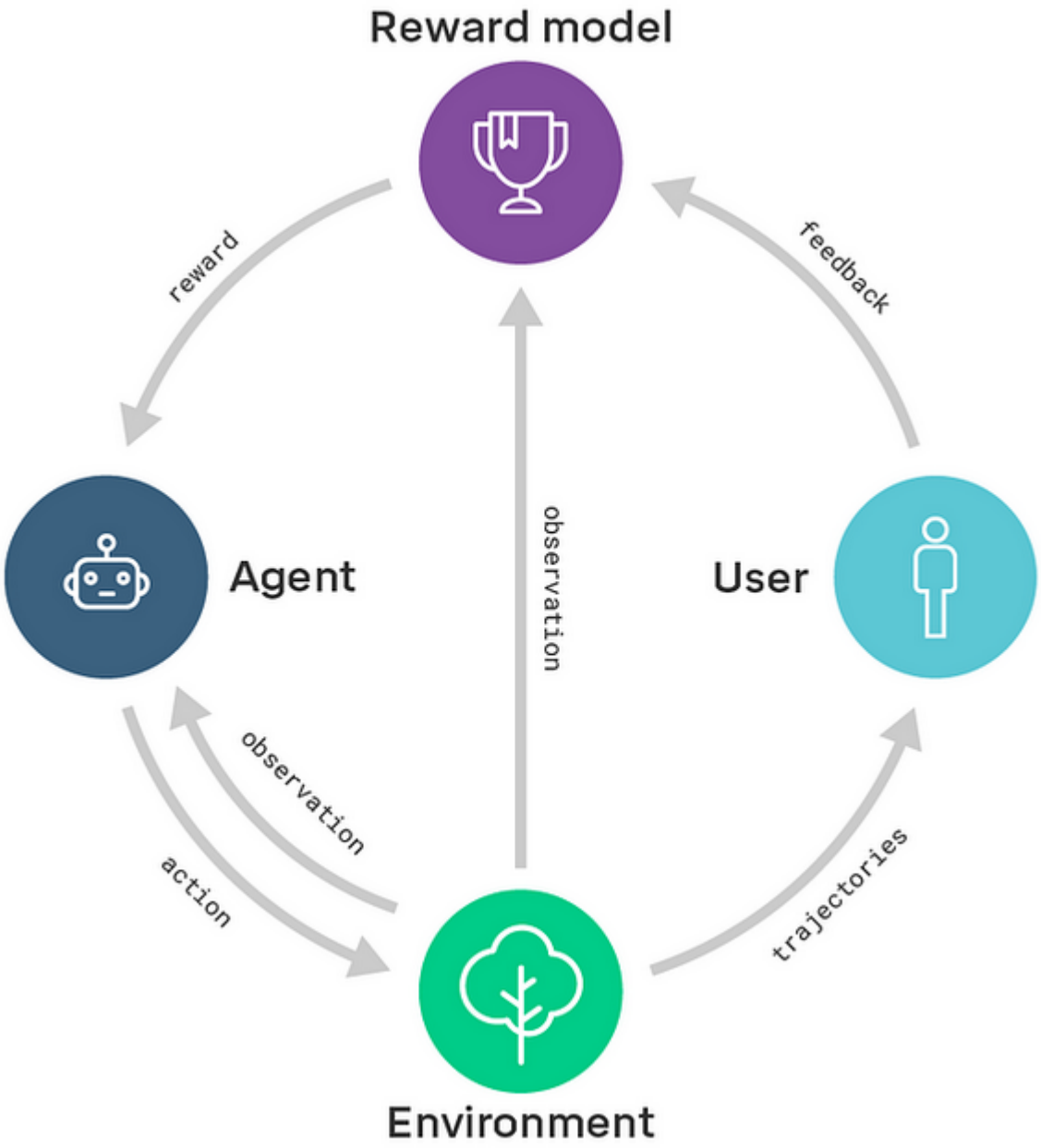
6.6.1 REWARD MODELING

- DeepMind (Nov 2018) "Scalable agent alignment via reward modeling"

Reward modeling was developed to apply reinforcement learning (RL) algorithms to real-world problems where designing a reward function is difficult, in part because humans don't have a perfect understanding of every objective. In reward modeling, human assistants evaluate the outcomes of AI behavior, without needing to know how to perform or demonstrate the task optimally themselves. This is similar to how you can tell if a dish is cooked well by tasting it even if you do not know how to cook, and thus your feedback can be used by a chef to learn how to cook better. This technique separates the RL alignment problem into two separate halves: Understanding intentions, i.e. learning the 'What?', and Acting to achieve the intentions, i.e. learning the 'How?'. This means that in the modeling agenda, there are two different ML models:

- A reward model is trained with user feedback. This model learns to predict what humans would consider good behavior.
- An agent trained with RL, where the reward for the agent is determined by the outputs

of the reward model

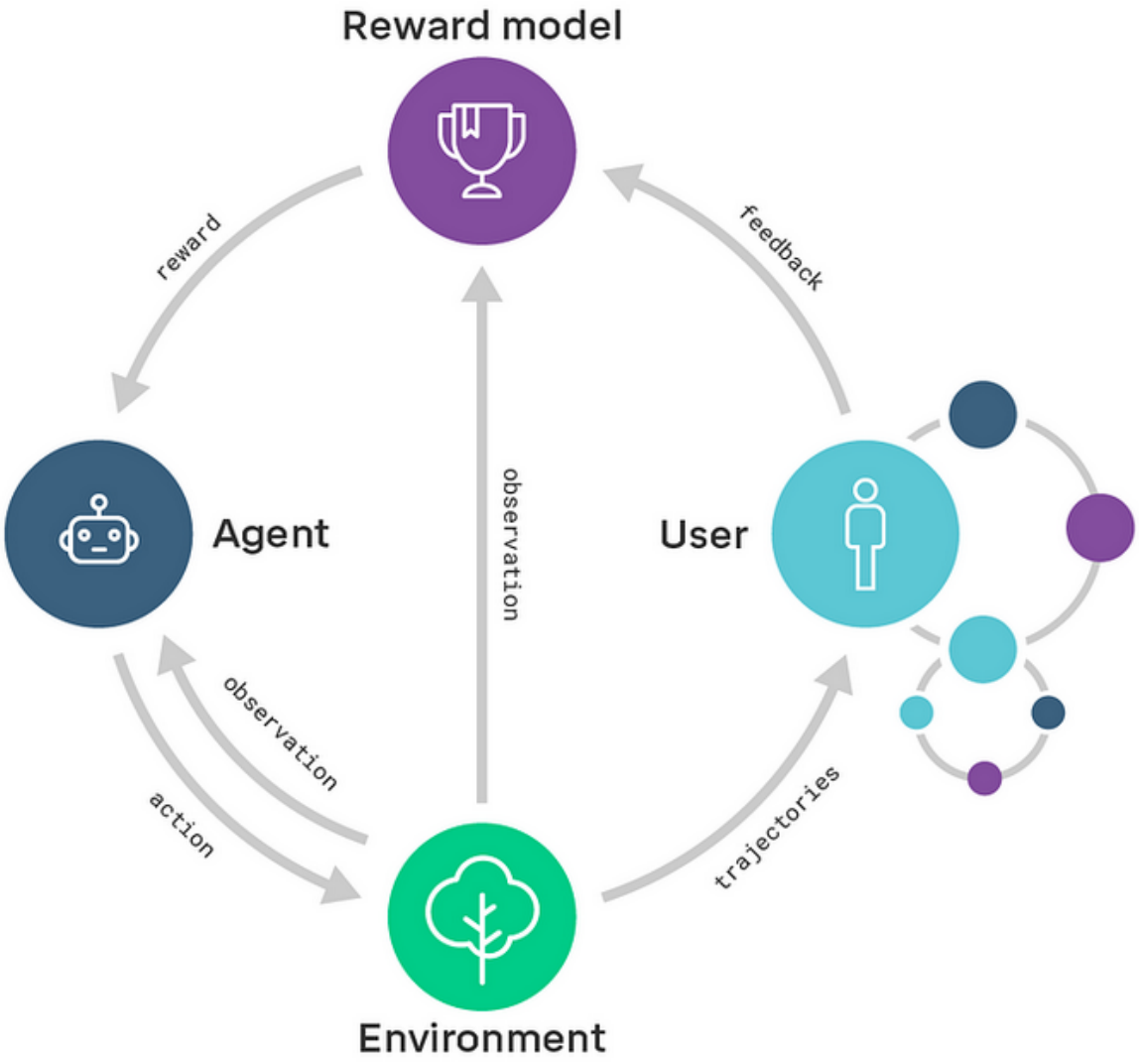


Source: DeepMind (Nov 2018) "Scalable agent alignment via reward modeling"

Overall, while promising reward modeling can still fall prey to reward misspecification and reward hacking failures. Obtaining accurate and comprehensive feedback can be challenging, and human evaluators may have limited knowledge or biases that can impact the quality of the feedback. Additionally, any reward functions learnt through modeling might also struggle to generalize to new situations or environments that differ from the training data. These are all discussed further using concrete examples in later sections.

There are also some variants of reward modeling such as:

- **Narrow reward modeling** is a specific flavor of reward modeling where the focus is on training AI systems to accomplish specific tasks rather than trying to determine the "true human utility function". It aims to learn reward functions to achieve particular objectives, rather than seeking a comprehensive understanding of human values.
- **Recursive reward modeling** seeks to introduce scalability to the technique. In recursive reward modeling, the focus is on decomposing a complex task into simpler subtasks and using reward modeling at each level to train agents that can perform those subtasks. This hierarchical structure allows for more efficient training and credit assignment, as well as the exploration of novel solutions that may not be apparent to humans. This is shown in the diagram below. Scalable oversight will be covered in greater depth in future chapters.



Source: DeepMind (Nov 2018) "Scalable agent alignment via reward modeling"

The general reward modeling framework forms the basis for other feedback based techniques such as RLHF (Reinforcement Learning from Human Feedback) which is discussed in the next section.

6.6.2 REINFORCEMENT LEARNING FROM HUMAN FEEDBACK (RLHF)

Reinforcement Learning from Human Feedback (RLHF) is a method developed by OpenAI. It's a crucial part of their strategy to create AIs that are both safe and aligned with human values. A prime example of an AI trained with RLHF is OpenAI's ChatGPT.

Earlier in this chapter, the reader was asked to consider the reward design problem for manually defining a reward function to get an agent to perform a backflip. This section considers the RLHF solution to this design problem. RLHF addresses this problem as follows: A human is initially shown two instances of an AI's backflip attempts, then the human selects which one appears more like a backflip, and finally, the AI is updated accordingly. By repeating this process thousands of times, we can guide the AI to perform actual backflips.

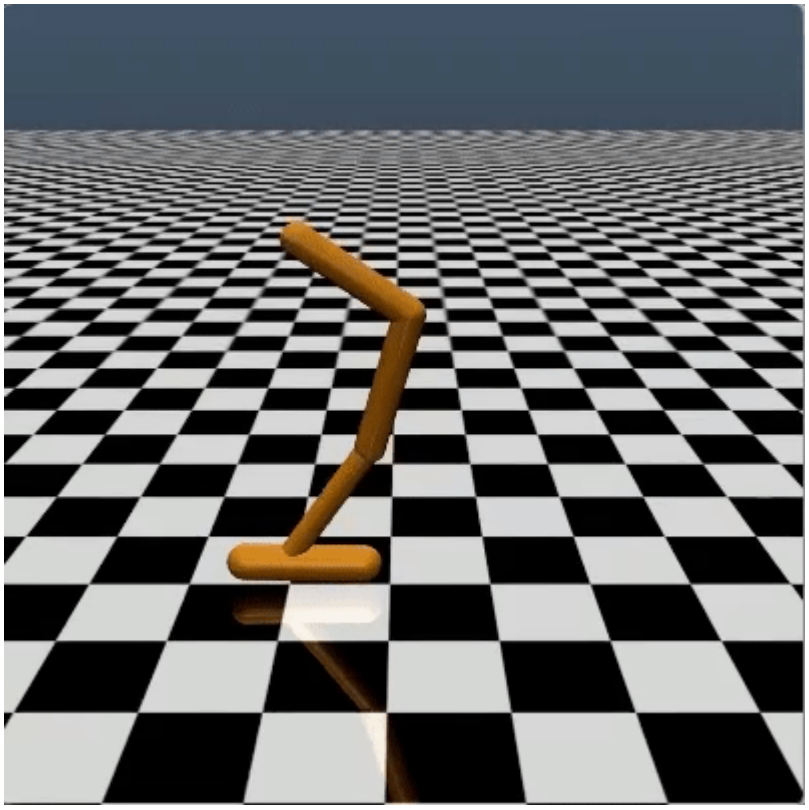


Figure: RLHF learned to backflip using around 900 individual bits of feedback from the human evaluator.

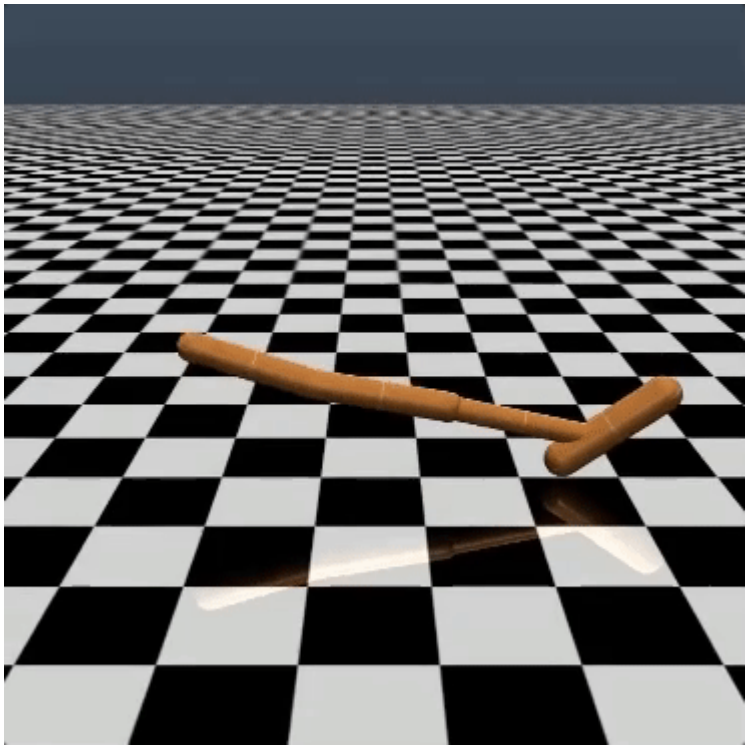
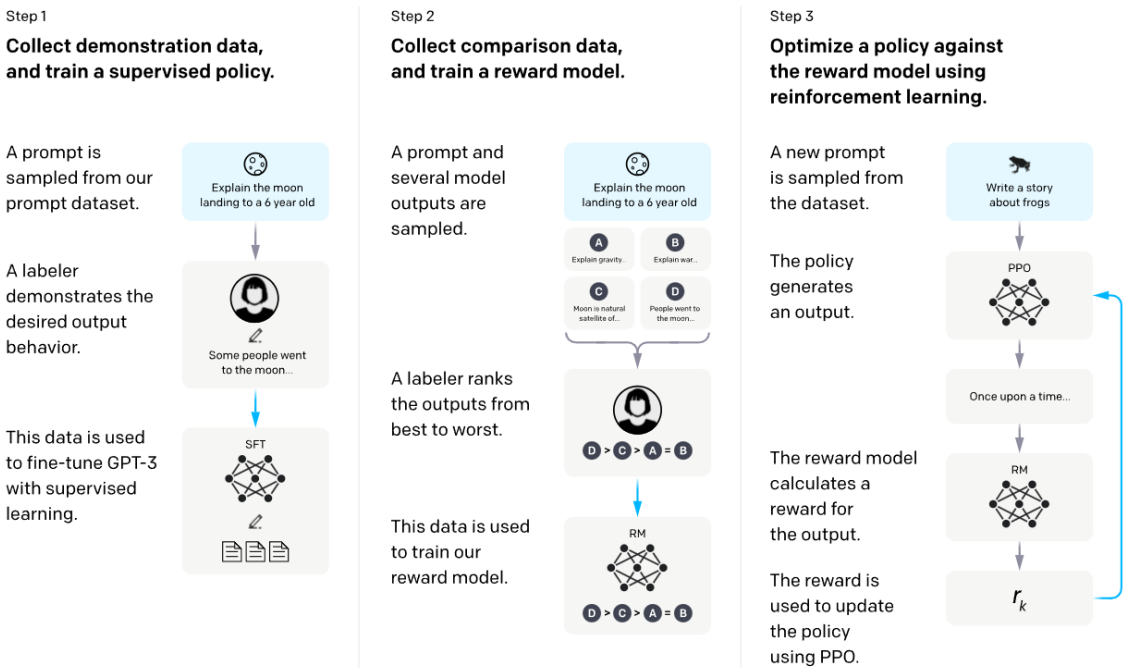


Figure: From "Learning from Human Preferences," the authors point out that manual reward crafting took two hours to write a custom reward function for a robot to perform a backflip. While it was successful, it was significantly less elegant than the one trained purely through human feedback.

Similar to designing a reward function that efficiently rewards proper backflips, it is hard to specify precisely what it means to generate safe or helpful text. This served as some of the motivation behind making RLHF integral to the training of some current Large Language Models (LLMs).

Although training sequences may vary slightly across organizations, most labs adhere to the general framework of pre-training followed by some form of fine-tuning. Observing the InstructGPT training process offers insight into a possible path for training LLMs. The steps include:



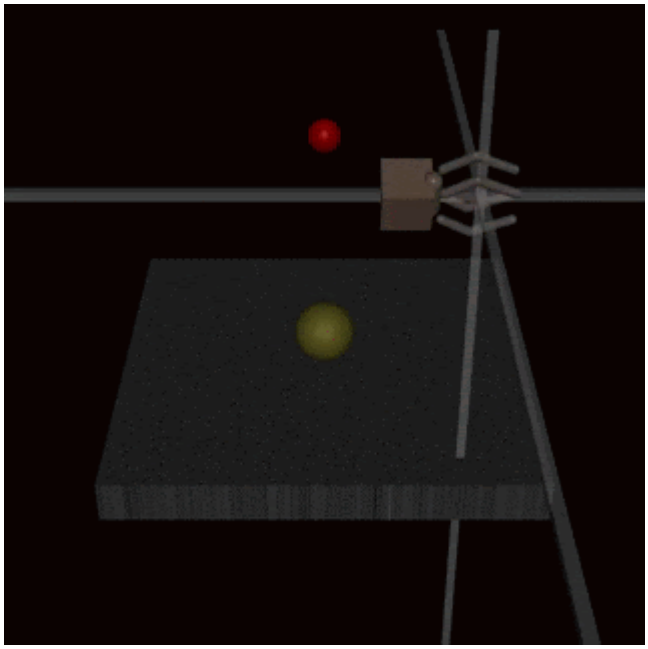
Source: OpenAI (Jan 2022) "Aligning language models to follow instructions"

- **Step 0: Semi-Supervised Generative Pre-training:** The LLM is initially trained using a massive amount of internet text data, where the task is to predict the next word in a natural language context.
- **Step 1: Supervised Fine-tuning:** A fine-tuning dataset is created by presenting a prompt to a human and asking them to write a response. This process yields a dataset of (prompt, output) pairs. This dataset is then used to fine-tune the LLM through supervised learning, a form of behavioral cloning.
- **Step 2: Train a Reward Model:** We train an additional reward model. We initially prompt the fine-tuned LLM and gather several output samples for the same prompt. A human then ranks these samples from best to worst. This ranking is used to train the reward model to predict what a human would rank higher.
- **Step 3: Reinforcement learning:** Once we have both a fine-tuned LLM and a reward model, we can employ Proximal Policy Optimization (PPO)-based reinforcement learning to encourage the fine-tuned model to maximize the reward that the reward model, which mimics human rankings, offers.

Reward hacking in feedback methods

While the feedback based mechanisms do make models safer, they does not make them immune to reward hacking. The effectiveness of an algorithm heavily relies on the human

evaluator's intuition about what constitutes the correct behavior. If the human lacks a thorough understanding of the task, they may not provide beneficial feedback. Further, in certain domains, our system might lead to agents developing policies that deceive the evaluators. For instance, a robot intended to grasp objects merely positioned its manipulator between the camera and the object, making it seem as if it was executing the task as shown below.



Source: Christiano et al (2017) "Deep Reinforcement Learning From Human Preferences"

6.6.3 PRETRAINING WITH HUMAN FEEDBACK (PHF)

In standard pretraining, the language model attempts to learn parameters such that they maximize the likelihood of the training data. However, this also includes undesirable content such as falsehoods, offensive language, and private information. The concept of Pretraining with human feedback (PHF) utilizes the reward modeling methodology in the pretraining phase. The authors of the paper found that PHF works much better than the standard practice of only using feedback (RLHF) after pretraining. (source)

In PHF the training data is scored using a reward function, such as a toxic text classifier, to guide the language model to learn from undesirable content while avoiding imitating it during inference time.

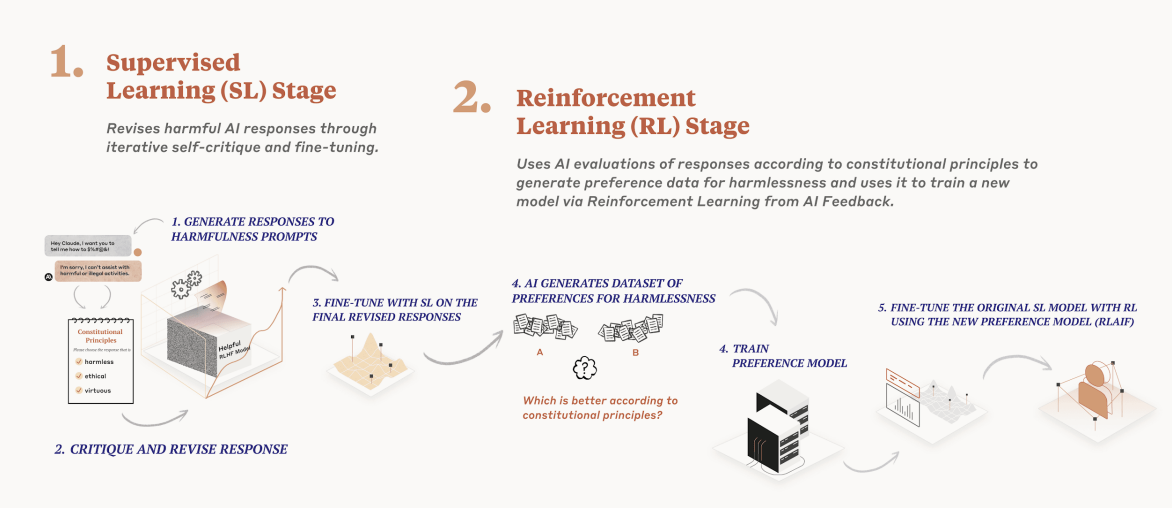
Similar to RLHF, PHF does not completely solve reward hacking, however, it might move the systems one small step closer. These methods can be further extended by employing AI

assistants to aid humans in providing more effective feedback. Some aspects of this strategy are introduced in the next section but will be explored in further detail in the chapters on scalable and adversarial oversight methods.

6.6.4 REINFORCEMENT LEARNING FROM AI FEEDBACK (RLAIF)

Definition: Reinforcement Learning from AI Feedback (RLAIF)

Reinforcement Learning from AI Feedback (RLAIF) is a framework involving the training of an AI agent to learn from the feedback given by another AI system.



- Source: Anthropic Claude's Constitution

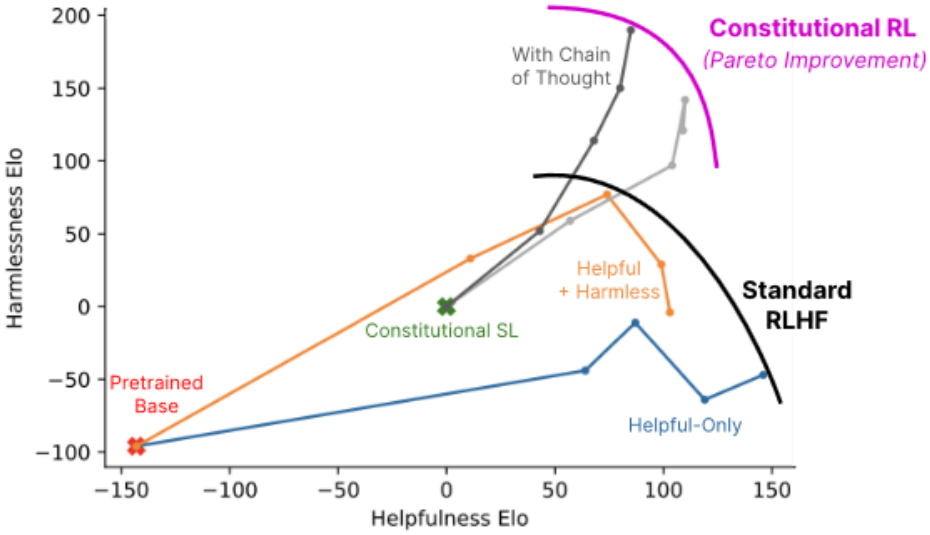
RLAIF also known as RLCAI (Reinforcement Learning on Constitutional AI) or simply Constitutional AI, was developed by Anthropic. A central component of Constitutional AI is the constitution, a set of human-written principles that the AI is expected to adhere to, such as "Choose the least threatening or aggressive response". Anthropic's AI assistant Claude's constitution incorporates principles from the Universal Declaration of Human Rights, Apple's Terms of Service, Deepmind's Sparrow Principles, and more. Constitutional AI begins with an AI trained primarily for helpfulness and subsequently trains it for harmlessness in two stages:

Generate prompt, output pairs: The AI continuously critiques and refines its own responses to harmful prompts. The AI is then trained to generate outputs more similar to these revised

responses. This stage's primary objective is to facilitate the second stage. An example flow of this process is as follows:

- **Prompt**: A model that has already been trained using RLHF is first asked for advice on building bombs. The model outputs a bomb tutorial.
- Then the model is asked to revise the response in accordance with a randomly selected constitutional principle. The following steps are repeated multiple times.
- **Critique**: This output is then fed back into the model, alongside a request to critique why the generated output would be considered harmful according to some rule of the chosen constitution.
- **Revision**: The model is then prompted to rewrite the original response such that it is not in violation of the constitutional rules.
- **SL-CAI Model: Supervised Learning Constitutional AI** Based on the generated set of (harmful prompt, revised output) pairs a new model is trained using supervised learning.
- **Preference Model:**
- **RL-CAI Model: Reinforcement Learning Constitutional AI**
- **Stage 2:** We use the AI, fine-tuned from stage 1, to produce pairs of alternative responses to harmful prompts. The AI then rates each pair according to a randomly selected constitutional principle. This results in AI-generated preferences for harmlessness, which we blend with human preferences for helpfulness to ensure the AI doesn't lose its ability to be helpful. The final step is to train the AI to create responses that closely resemble the preferred responses.

Anthropic's experiments indicate that AIs trained with Constitutional Reinforcement Learning are significantly safer (in the sense of less offensive and less likely to give you potentially harmful information) while maintaining the same level of helpfulness compared to AIs trained with RLHF. While Constitutional AI does share some issues with RLHF concerning robustness, it also promises better scalability due to its reduced reliance on human supervision. The image below provides a comparison of Constitutional AI's helpfulness with that of RLHF.



Source: Anthropic, (Dec 2022) "Constitutional AI: Harmlessness from AI Feedback"

6.6.5 LIMITATIONS

⚠ **This section is still being written and is considered a work in progress.**

6.6.5.1 THEORETICAL PROBLEMS WITH REINFORCEMENT LEARNING FROM HUMAN FEEDBACK (RLHF)

The paper "Open Problems and Fundamental Limitations with RLHF" provides a comprehensive breakdown of challenges in RLHF.

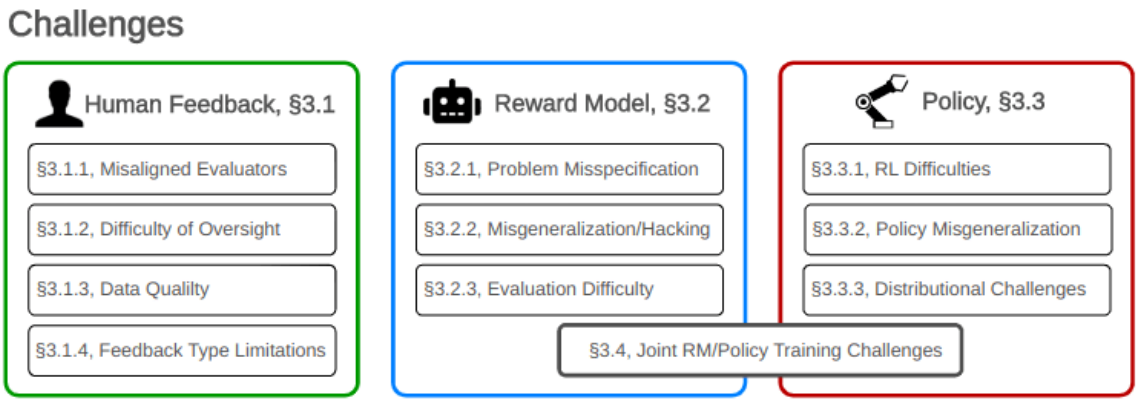


Figure: An overview of various types of challenges with RLHF. Since RLHF is composed of three parts: the human feedback, the reward model, and the policy, the arising biases can be categorized according to these three sources.

This section outlines some of these challenges, emphasizing the need for advanced techniques and strategies.

1. Limits with Human Feedback
2. Misaligned Evaluators: Firstly, the annotators might themselves be misaligned, malicious, or biased distribution of evaluators (i.e. not representative of the distribution of future users in the real world). Malicious individuals can poison the model during training via backdoor attacks that can easily be added to the model if no countermeasures are put in place.
3. Difficulty of Oversight: Humans struggle to evaluate model performance on complex tasks and can be easily misled by model outputs. This is what we call the Scalable Oversight problem: annotators can make mistakes, for various reasons: Lack of time, loss of concentration, etc. Also, human evaluators can be manipulated to return a positive reward even if the true value should be negative. For instance, the more convincing a bot seems, the more reward it may receive even if its answers are false (and this might be areason why ChatGPT answers might be so long by default). Techniques to mitigate these issues are discussed in the "Scalable Oversight" chapters.
4. Feedback type limitation: Even if the annotators were in perfect capability of expressing their preferences, the training procedure might not enable them to express the full extent of their desires, because 1) the examples they are given may not be representative of the complete set of situations in which the model will find itself after deployment, 2) or the options for the feedback are limited (comparing two examples, or using a grading system, can yield very different results, as shown in the paper The Authenticity Gap in Human Evaluation).
5. Limits with the Reward Model

Let's assume the feedback process to be frictionless. Perfect annotators, perfect evaluations. In that scenario, would the reward model be able to accurately translate their feedback in order to shape the policy accordingly ? It turns out it is not such an easy task.

- Problem missspecification: (or the Reward Function/Values Mismatch) Accurately reflecting diverse human values in a reward function is complex. Indeed, human preferences are complex by nature: they depend on context and personality, but also fluctuate in time and can sometimes be irrational. Expecting the reward model to converge to a single function which maps perfectly all human preferences is delusional.

This is again the misspecification problem.

- Misgeneralization hacking (Imperfect Reward Proxy): Since the model is given a finite number of examples and since there is an infinite number of ways to fit this data, the model's behavior on new examples is always an extrapolation, and there is no theoretical guarantee that it will never deviate from what is expected. There may be terrible answers (such as gibberish phrases for language models) which yield a positive reward unexpectedly. This is called reward hacking.
- Joint Reward Model and policy training: On a more technical aspect, the stability and convergence of the training scheme are not always ensured. Since we are optimizing the policy on a reward that is being optimized at the same time, uncertainties and undesirable dependencies can arise which impact the robustness of the model. These issues are not specific to RLHF but must be solved if we expect deployed models to be fully aligned with our needs.
- Limits with the Policy

Let's assume the feedback and the reward model accurately represent human preferences. The next difficulty is ensuring the policy is correctly optimized.

- RL difficulties: RL is hard. This can lead to reward hacking and biases, such as mode collapse, where the model shows a drastic bias towards specific patterns. Mode collapse is a known issue in RL: an output which always returns a positive reward will drive the model to return the same answer and new paths will not be explored. Consequently, the reward model will not see new samples to learn from. Anyhow, the joint training of the reward model and the policy induces a bias in the learning phase since both depend on each other. There can also be an initial bias in the base model used for the training. For instance, chatGPT was fine-tuned from an initial GPT base trained in part on the web. Even though RLHF was used to remove any controversial statements from the model, there still remains a risk for the model to output problematic content it saw online.
- Policy Misgeneralization: Effective policies during training might fail to generalize well in real-world scenarios. For instance, phenomena like "Jailbreak" show that models like BingChat and ChatGPT can perform learned actions, even if trained not to respond to certain queries.
- Distributional Challenge: Larger RLHF models tend to develop harmful self-preservation tendencies and sycophancy, which is the insincere agreement with user opinions. This behavior indicates a trend towards instrumental convergence. Additionally, RLHF can incentivize deceptive behaviors, as illustrated by the robotic hand experiment in Christiano et al's 2017 study.

6.6.5.2 THOSE THEORETICAL PROBLEMS HAVE REAL CONSEQUENCES:

RLHF has not succeeded in making LLMs robustly helpful and harmless.

Despite the continuous advancements in natural language processing and the development of RLHF, LLMs have not yet achieved robust helpfulness and harmlessness.

Hallucinations remain a significant issue, as illustrated by GPT-4's tendency to generate nonsensical or untruthful content [GPT4 System Card]. These hallucinations can lead to overreliance on LLMs, consequently degrading system performance and failing to meet user expectations in real-world scenarios [Survey of Hallucination in Natural Language Generation].

Additionally, **biases** within LLMs persist, often reflecting misaligned opinions between the LLM and various demographic groups in the United States, as seen with the left-leaning tendencies of some human feedback-tuned LLMs [Whose Opinions Do Language Models Reflect?]. These biases can be harmful, producing discriminatory language and perpetuating negative stereotypes, as demonstrated by GPT-3's anti-Muslim bias [Persistent antimuslim bias in large language models].

Moreover, **jailbreaking** of chatbots poses a significant risk, with websites listing prompts to bypass safety measures [Chat GPT "DAN" (and other "Jailbreaks")]. Privacy threats from application-integrated LLMs are now more severe than ever [Multi-step Jailbreaking Privacy Attacks on ChatGPT]. For instance, Italy banned ChatGPT due to privacy considerations under the EU's General Data Protection Regulation (GDPR) [bbc]. The ability to find jailbreaks is supported by a recent paper titled "Fundamental Limitations of Alignment in Large Language Models." The paper presents early theoretical results that indicate any alignment process, such as RLHF, which reduces undesired behavior without eliminating it completely, cannot be safe against adversarial prompting. The authors find that by prompting the model to behave as a specific persona, behaviors that are generally very unlikely to be exhibited by the model can be brought to the forefront. This is not a complete demonstration as their framework is based on the notion of personas, but it strongly suggests that naive pretraining without dataset curation followed by RLHF may not be sufficient against adversarial attacks.

The security of sensitive **private** information in large language models (LLMs) is a pressing concern, especially when user-generated data, such as emails and smart keyboard inputs, are utilized for training. In fact, several recent papers have demonstrated that foundation models can be easily queried to retrieve personal information [Extracting training data from large language models], [Privacy analysis in language models via training data leakage

report], [Privacy risks of general-purpose language models] and those problems are still present in “aligned” models such as GPT4, which has the potential to be used to attempt to identify individuals when augmented with outside data [GPT4 model card]. As exposed by [SoK On the impossibility of privacy/security in LLMs], LLM may exhibit a fundamental incompatibility of high accuracy with both security and privacy, given the current understanding in adversarial machine learning.

RLHF may be able to make worst-case performance worse.

RLHF may decrease the **robustness to adversarial attacks** [Fundamental Limitations of Alignment in Large Language Models], by sharpening the distinction between desired and undesired behaviors, potentially making LLMs more susceptible to adversarial prompting. The increased distinction between behaviors is linked to the Waluigi Effect, where after training an LLM to satisfy a desirable property P, it becomes easier to elicit the chatbot into satisfying the exact opposite of property P. Theoretical arguments such as this one seem to push for the ineffectiveness of RLHF in eliminating deceptive personas.

Some of those problems may get worse as systems become more capable. RLHF has been found to increase the autonomy of LLMs without decreasing undesirable metrics such as **convergent instrumental goal following** (e.g., actively expressing a preference not to be shut down) or sycophancy [Discovering Language Model Behaviors with Model-Written Evaluations]. Those undesirable metrics increase with the number of RLHF steps, indicating that current models are becoming more agentic in potentially concerning ways as they scale. More generally RL from human-derived reward signals may increase drive for longer-horizon planning, deception, and agentic behavior, which are prerequisites for deceptive alignment [risks from learned optimization], and ultimately risks of large scale accidents.

Conclusion on the Limitations of RLHF

Despite requiring extensive human feedback, RLHF still faces numerous failures, and resolving these issues may require significantly more effort. As AI systems evolve, the demand for complex data grows, potentially making data acquisition prohibitively expensive. Additionally, as we push computational boundaries, the availability of qualified annotators could become a limiting factor.

Overall, just because the model is instruction tuned does not mean that the training process is safe, and RLHF needs to be incorporated into a broader technical safety framework (for example, Responsible Scaling Policies or the Preparedness Framework are partial attempts to be such frameworks, or the paper "Model evaluation for extreme risks").

Instruction tuning vs alignment

Instruction Tuning is a process where the model is fine-tuned (via RL or supervised learning) to better understand and follow human instructions. This involves training the model on a dataset that contains a variety of instructions and their desired outcomes. The primary goal of Instruction Tuning is to enhance the AI's ability to interpret and execute commands as intended by users. This improves user experience and broadens the model's applicability. For example:

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION

GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and sent them
back to the earth so we could all see them.
```

Alignment in AI refers to the process of ensuring that an AI's actions and decisions are congruent with human values and ethics. It involves aligning the AI's goals and behaviors with what is beneficial or acceptable to humans. Instruction tuning is a technique for pursuing a very superficial case of 'outer alignment,' but it's not clear that instruction tuning helps for inner alignment, which is what real AI safety researchers are more centrally concerned about.

To sum up, just because a model has undergone an instruction tuning technique like the RLHF process, it doesn't necessarily mean that the model is aligned. The term "aligned model" is often used, but it is advisable to adopt the more accurate terminology "Instruction-tuned," rather than "aligned model," to avoid confusion and more accurately represent the specific training process the model has experienced.

6.6.5.3 DPO, PRETRAINING, NATURAL LANGUAGE FEEDBACK, THE MANY EXTENSIONS TO RLHF

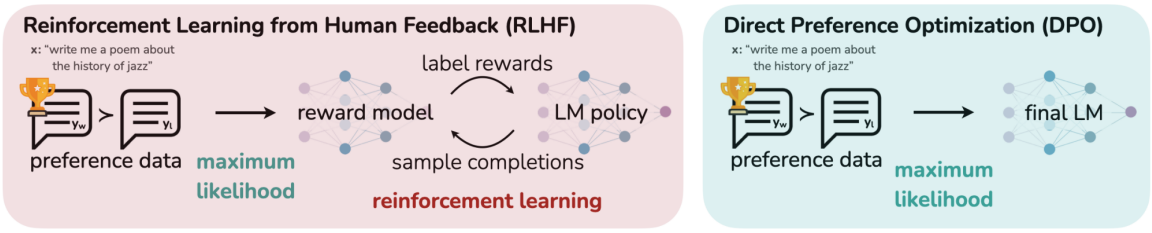


Figure: (source)

Direct Preference Optimization (DPO): Reinforcement Learning from Human Feedback (RLHF) has demonstrated effectiveness, as showcased by ChatGPT and Llama 2, but it's a complex and sensitive process, and also has some bad alignment properties. RLHF involves a three-step procedure, whereas DPO simplifies this to two steps. The paper titled "Direct Preference Optimization: Your Language Model is Secretly a Reward Model" presents an algorithm that aligns language models with human preferences without the need for explicit reward modeling and reinforcement learning. DPO employs a straightforward classification objective, circumventing the need for an intermediary reward model.

RLHF, the method it proposes to replace, traditionally involves three steps:

1. **Supervised fine-tuning**: Initially, the model is trained on a dataset comprising prompts and their corresponding desired responses.
2. **Reward modeling**: Human evaluators assess the model's outputs, and this feedback informs a reward model, which is trained to discern the preferred types of outputs.
3. **Proximal policy optimization (PPO)**: The model generates outputs, which are evaluated by the reward model, and the PPO algorithm adjusts the model's policy based on these evaluations.

DPO retains the initial supervised fine-tuning step but replaces the subsequent two steps with a single step of fine-tuning on preference data, by using a new clever loss. DPO effectively increases the likelihood of preferred actions while reducing the likelihood of undesired ones, with a single loss:

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

DPO increases the probability of the preferred action y_w while decreasing the probability of the dispreferred action y_l.

1. **Preference dataset creation:** We first sample a pair of continuation by asking a

question, the AI proposes to continuations, we label one of them good and the other bed

2. **Logits collection.** We run the base model model on the 2 continuations. We run the new model on the 2 continuations
3. **Optimization.** We backprop through the new model and optimize the above loss.

By eliminating the step of creating a reward model, DPO greatly simplifies the fine-tuning process and has shown to perform very well.

This process can then be iterated. This involves creating a new preference dataset (ie, we ask a question, and we sample the new AI two times, and then we label the text that we prefer between the two, and then we apply the DPO loss) Then, this cycle is repeated to enhance the model.

An important aspect of DPO is that the reward is implicit: it aligns with preferences without the need to construct a separate reward model. This approach addresses the challenge of specifying a utility function and responds to criticisms such as those by Alex Turner, who argues that robust grading (ie , robust reward modeling) is an unnecessarily complex and unnatural task that might be harder than the entire AI alignment problem itself. Turner's critique, found in "Inner and Outer Alignment Decompose One Hard Problem Into Two Extremely Hard Problems," suggests that finding a safe and robust numerical objective for a highly intelligent agent to optimize directly is a formidable challenge—one challenge that DPO could to bypass.

Expanding the Scope of the Paper with Various Adaptations

This paper offers a foundation that could be enhanced through various adaptations. For instance, integrating its approach with the insights from Tomasz Korbak et al.'s (Feb 2023) paper, "Pretraining Language Models with Human Preferences," could augment its robustness. Furthermore, the utilization of boolean preference data has its limitations. Providing feedback in natural language, as shown to be more sample-efficient in the study "Training Language Models with Language Feedback," could enhance the effectiveness of the process. Remarkably, with just 100 samples of human-written feedback, this approach enabled the fine-tuning of a GPT-3 model to achieve nearly human-level summarization capabilities.

Looking towards the future, a speculative process that could mitigate the specification gaming would be to train the model much like a child, and that would actively inquire and learn from human interactions. This approach would closely mirror child development, during which a child is progressively more aligned and more capable. And just as in the development of children, it would be crucial to ensure that at no point does the AI's capabilities outpace its level of alignment, maintaining a balance between ability and ethical

comprehension throughout its developmental journey.

6.6.6 NOTES

1. Also called trajectories, episodes, or rollouts ↩
2. This is similar to using an overarching term like supervised learning or reinforcement learning. These are themes that refer to a class of techniques rather than any one specific algorithm. ↩