(WIP) A Little Bit of Reinforcement Learning from Human Feedback

A short introduction to RLHF and post-training focused on language models.

Nathan Lambert

19 January 2025

Abstract

Reinforcement learning from human feedback (RLHF) has become an important technical and storytelling tool to deploy the latest machine learning systems. In this book, we hope to give a gentle introduction to the core methods for people with some level of quantitative background. The book starts with the origins of RLHF – both in recent literature and in a convergence of disparate fields of science in economics, philosophy, and optimal control. We then set the stage with definitions, problem formulation, data collection, and other common math used in the literature. We detail the detail the popular algorithms and future frontiers of RLHF.

Contents

1 Introduction **5**
- 1.1 Scope of This Book . . . 5
  - 1.1.1 Chapter Summaries . . . 5
  - 1.1.2 Target Audience . . . 6
  - 1.1.3 How to Use This Book . . . 6
  - 1.1.4 About the Author . . . 7
- 1.2 Future of RLHF . . . 7

2 Key Related Works **8**
- 2.1 Origins to 2018: RL on Preferences . . . 8
- 2.2 2019 to 2022: RL from Human Preferences on Language Models 8
- 2.3 2023 to Present: ChatGPT Eta . . . 9

3 Definitions **10**
- 3.1 ML Definitions . . . 10
- 3.2 NLP Definitions . . . 10
- 3.3 RL Definitions . . . 10

4 Problem Formulation **12**
- 4.1 Manipulating the standard RL setup . . . 12
- 4.2 Finetuning and regularization . . . 13

5 The Nature of Preferences **15**
- 5.1 The path to optimizing preferences . . . 15
  - 5.1.1 Quantifying preferences . . . 16
  - 5.1.2 On the possibility of preferences . . . 16

6 Preference Data **18**
- 6.1 Collecting Preference Data . . . 18
- 6.2 Rankings vs. Ratings . . . 18
  - 6.2.1 Sourcing and Contracts . . . 18
- 6.3 Are the Preferences Expressed in the Models? . . . 21

7 Reward Modeling **22**
- 7.1 Training Reward Models . . . 22
- 7.2 Architecture . . . 23
- 7.3 Implementation Example . . . 23
- 7.4 Variants . . . 23
  - 7.4.1 Preference Margin Loss . . . 23
  - 7.4.2 Balancing Multiple Comparisons Per Prompt . . . 24
  - 7.4.3 K-wise Loss Function . . . 24
- 7.5 Outcome Reward Models . . . 24
- 7.6 Process Reward Models . . . 25
- 7.7 Reward Models vs. Outcome RMs vs. Process RMs vs. Value Functions . . . 26

7.8 Generative Reward Modeling . . . 26
7.9 Further Reading . . . 27

8 Regularization **28**
8.1 KL Distances in RL Optimization . . . 28
8.1.1 Reference Model to Generations . . . 28
8.1.2 Implementation Example . . . 29
8.2 Pretraining Gradients . . . 29
8.3 Other Regularization . . . 30

9 Instruction Tuning **31**

10 Rejection Sampling **32**
10.1 Related works . . . 32
10.2 Training Process . . . 32
10.2.1 Generating Completions . . . 32
10.2.2 Selecting Top-N Completions . . . 33
10.2.3 Fine-tuning . . . 36
10.2.4 Details . . . 36
10.3 Related: Best-of-N Sampling . . . 36

11 Policy Gradient Algorithms **38**
11.1 Policy Gradient Algorithms . . . 38
11.1.1 Vanilla Policy Gradient . . . 39
11.1.2 REINFORCE . . . 40
11.1.3 REINFORCE Leave One Out (RLOO) . . . 41
11.1.4 Proximal Policy Optimization . . . 41
11.1.5 Group Relative Policy Optimization . . . 42
11.2 Implementation . . . 43
11.2.1 Policy Gradient . . . 43
11.2.2 Proximal Policy Optimization . . . 43
11.2.3 Group Relative Policy Optimization . . . 45
11.3 KL Controllers . . . 47
11.4 Double regularization . . . 47

12 [Incomplete] Direct Alignment Algorithms **49**

13 [Incomplete] Constitutional AI and AI Feedback **50**
13.1 Trade-offs . . . 50
13.2 Constitutional AI . . . 50
13.2.1 Summary . . . 50
13.2.2 Mathematical Formulation . . . 50

14 [Incomplete] Reasoning Training and Models **52**

15 [Incomplete] Synthetic Data **53**

16 [Incomplete] Evaluation **54**
16.1 ChatBotArena 54
16.2 Private Leaderboards 55

17 Over Optimization **56**
17.1 Qualitative (behavioral) over-optimization 56
17.1.1 Managing proxy objectives 56
17.1.2 Llama 2 and "too much RLHF" 57
17.1.3 An aside on "undercooking" RLHF 57
17.2 Quantitative over-optimization 57

18 Style and Information **59**
18.1 The Chattiness Paradox 59
18.1.1 How Chattiness Emerges 61

Bibliography **62**

1 Introduction

Reinforcement learning from Human Feedback (RLHF) is a technique used to incorporate human information into AI systems. RLHF emerged primarily as a method to solve hard to specify problems. Its early applications were often in control problems and other traditional domains for reinforcement learning (RL). RLHF became most known through the release of ChatGPT and the subsequent rapid development of large language models (LLMs) and other foundation models.

The basic pipeline for RLHF involves three steps. First, a language model that can follow user questions must be trained (see Chapter 9). Second, human preference data must be collected for the training of a reward model of human preferences (see Chapter 7). Finally, the language model can be optimized with a RL optimizer of choice, by sampling generations and rating them with respect to the reward model (see Chapter 3 and 11). This book details key decisions and basic implementation examples for each step in this process.

RLHF has been applied to many domains successfully, with complexity increasing as the techniques have matured. Early breakthrough experiments with RLHF were applied to deep reinforcement learning [1], summarization [2], following instructions [3], parsing web information for question answering [4], and "alignment" [5].

In modern language model training, RLHF is one component of post-training. Post-training is a more complete set of techniques and best-practices to make language models more useful for downstream tasks [6]. Post-training can be summarized as using three optimization methods:

1. Instruction / Supervised Finetuning (SFT),
2. Preference Finetuning (PreFT), and
3. Reinforcement Finetuning (RFT).

This book focuses on the second area, **preference finetuning**, which has more complexity than instruction tuning and is far more established than Reinforcement Finetuning.

1.1 Scope of This Book

This book hopes to touch on each of the core steps of doing canonical RLHF implementations. It will not cover all the history of the components nor recent research methods, just techniques, problems, and trade-offs that have been proven to occur again and again.

1.1.1 Chapter Summaries

This book has the following chapters following this Introduction:

Introductions:

1. Introduction
2. What are preferences?: The philosophy and social sciences behind RLHF.
3. Optimization and RL: The problem formulation of RLHF.
4. Seminal (Recent) Works: The core works leading to and following ChatGPT.

Problem Setup:

1. Definitions: Mathematical reference.
2. Preference Data: Gathering human data of preferences.
3. Reward Modeling: Modeling human preferences for environment signal.
4. Regularization: Numerical tricks to stabilize and guide optimization.

Optimization:

1. Instruction Tuning: Fine-tuning models to follow instructions.
2. Rejection Sampling: Basic method for using a reward model to filter data.
3. Policy Gradients: Core RL methods used to perform RLHF.
4. Direct Alignment Algorithms: New PreFT algorithms that do not need RL.

Advanced (TBD):

1. Constitutional AI
2. Synthetic Data
3. Evaluation
4. Reasoning and Reinforcement Finetuning

Open Questions (TBD):

1. Over-optimization
2. Style

1.1.2 Target Audience

This book is intended for audiences with entry level experience with language modeling, reinforcement learning, and general machine learning. It will not have exhaustive documentation for all the techniques, but just those crucial to understanding RLHF.

1.1.3 How to Use This Book

This book was largely created because there were no canonical references for important topics in the RLHF workflow. The contributions of this book are supposed to give you the minimum knowledge needed to try a toy implementation or dive into the literature. This is *not* a comprehensive textbook, but rather a quick book for reminders and getting started. Additionally, given the web-first nature of this book, it is expected that there are minor typos and somewhat random progressions – please contribute by fixing bugs or suggesting important content on GitHub.

1.1.4 About the Author

Dr. Nathan Lambert is a RLHF researcher contributing to the open science of language model fine-tuning. He has released many models trained with RLHF, their subsequent datasets, and training codebases in his time at the Allen Institute for AI (Ai2) and HuggingFace. Examples include Zephyr-Beta, Tulu 2, OLMo, TRL, Open Instruct, and many more. He has written extensively on RLHF, including many blog posts and academic papers.

1.2 Future of RLHF

With the investment in language modeling, many variations on the traditional RLHF methods emerged. RLHF colloquially has become synonymous with multiple overlapping approaches. RLHF is a subset of preference fine-tuning (PreFT) techniques, including Direct Alignment Algorithms (See Chapter 12). RLHF is the tool most associated with rapid progress in "post-training" of language models, which encompasses all training after the large-scale autoregressive training on primarily web data. This textbook is a broad overview of RLHF and its directly neighboring methods, such as instruction tuning and other implementation details needed to set up a model for RLHF training.

As more successes of fine-tuning language models with RL emerge, such as OpenAI's o1 reasoning models, RLHF will be seen as the bridge that enabled further investment of RL methods for fine-tuning large base models.

2 Key Related Works

In this chapter we detail the key papers and projects that got the RLHF field to where it is today. This is not intended to be a comprehensive review on RLHF and the related fields, but rather a starting point and retelling of how we got to today. It is intentionally focused on recent work that lead to ChatGPT. There is substantial further work in the RL literature on learning from preferences [7]. For a more exhaustive list, you should use a proper survey paper [8],[9].

2.1 Origins to 2018: RL on Preferences

The field was recently popularized with the growth of Deep Reinforcement Learning and has grown into a broader study of the applications of LLMs from many large technology companies. Still, many of the techniques used today are deeply related to core techniques from early literature on RL from preferences.

TAMER: Training an Agent Manually via Evaluative Reinforcement, Proposed a learned agent where humans provided scores on the actions taken iteratively to learn a reward model [10]. Other concurrent or soon after work proposed an actor-critic algorithm, COACH, where human feedback (both positive and negative) is used to tune the advantage function [11].

The primary reference, Christiano et al. 2017, is an application of RLHF applied on preferences between Atari trajectories [1]. The work shows that humans choosing between trajectories can be more effective in some domains than directly interacting with the environment. This uses some clever conditions, but is impressive nonetheless. This method was expanded upon with more direct reward modeling [12]. TAMER was adapted to deep learning with Deep TAMER just one year later [13].

This era began to transition as reward models as a general notion were proposed as a method for studying alignment, rather than just a tool for solving RL problems [14].

2.2 2019 to 2022: RL from Human Preferences on Language Models

Reinforcement learning from human feedback, also referred to regularly as reinforcement learning from human preferences in its early days, was quickly adopted by AI labs increasingly turning to scaling large language models. A large portion of this work began between GPT-2, in 2018, and GPT-3, in 2020. The earliest work in 2019, *Fine-Tuning Language Models from Human Preferences* has many striking similarities to modern work on RLHF [15]. Learning reward models, KL distances, feedback diagrams, etc – just the evaluation tasks, and capabilities, were different. From here, RLHF was applied to a variety of tasks. The popular applications were the ones that worked at the time. Important examples include general summarization [2], recursive summarization of books [16], instruction following (InstructGPT) [3], browser-assisted question-answering (WebGPT) [4], supporting answers with citations (GopherCite) [17], and general dialogue (Sparrow) [18].

Aside from applications, a number of seminal papers defined key areas for the future of RLHF, including those on:

1. Reward model over-optimization [19]: The ability for RL optimizers to over-fit to models trained on preference data,
2. Language models as a general area of study for alignment [20], and
3. Red teaming [21] – the process of assessing safety of a language model.

Work continued on refining RLHF for application to chat models. Anthropic continued to use it extensively for early versions of Claude [5] and early RLHF open-source tools emerged [22],[23],[24].

2.3 2023 to Present: ChatGPT Eta

The announcement of ChatGPT was very clear in the role of RLHF in its training [25]:

> We trained this model using Reinforcement Learning from Human Feedback (RLHF), using the same methods as InstructGPT, but with slight differences in the data collection setup.

Since then RLHF has been used extensively in leading language models. It is well known to be used in Anthropic's Constitutional AI for Claude [26], Meta's Llama 2 [27] and Llama 3 [28], Nvidia's Nemotron [29], Ai2's Tülu 3 [6], and more.

Today, RLHF is growing into a broader field of preference fine-tuning (PreFT), including new applications such as process reward for intermediate reasoning steps [30], direct alignment algorithms inspired by Direct Preference Optimization (DPO) [31], learning from execution feedback from code or math [32],[33], and other online reasoning methods inspired by OpenAI's o1 [34].

3 Definitions

This chapter includes all the definitions, symbols, and operatings frequently used in the RLHF process.

3.1 ML Definitions

- **Kullback-Leibler (KL) divergence ($D_{KL}(P||Q)$)**, also known as KL divergence, is a measure of the difference between two probability distributions. For discrete probability distributions P and Q defined on the same probability space $\mathcal{X}$, the KL distance from Q to P is defined as:

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

3.2 NLP Definitions

- **Prompt (x)**: The input text given to a language model to generate a response or completion.
- **Completion (y)**: The output text generated by a language model in response to a prompt. Often the completion is denoted as $y|x$.
- **Chosen Completion (y_c)**: The completion that is selected or preferred over other alternatives, often denoted as y_{chosen}.
- **Preference Relation ($\succ$)**: A symbol indicating that one completion is preferred over another, e.g., $y_{chosen} \succ y_{rejected}$.
- **Policy (π)**: A probability distribution over possible completions, parameterized by θ: $\pi_\theta(y|x)$.

3.3 RL Definitions

- **Reward (r)**: A scalar value indicating the desirability of an action or state, typically denoted as r.
- **Action (a)**: A decision or move made by an agent in an environment, often represented as $a \in A$, where A is the set of possible actions.
- **State (s)**: The current configuration or situation of the environment, usually denoted as $s \in S$, where S is the state space.
- **Trajectory (τ)**: A trajectory is a sequence of states, actions, and rewards experienced by an agent: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_T, r_T)$.
- **Trajectory Distribution ($(\tau|\pi)$)**: The probability of a trajectory under policy π is $P(\tau|\pi) = p(s_0)\prod_{t=0}^{T} \pi(a_t|s_t)p(s_{t+1}|s_t, a_t)$, where $p(s_0)$ is the initial state distribution and $p(s_{t+1}|s_t, a_t)$ is the transition probability.

- **Policy (π)**: In RL, a policy is a strategy or rule that the agent follows to decide which action to take in a given state: $\pi(a|s)$.
- **Value Function (V)**: A function that estimates the expected cumulative reward from a given state: $V(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$.
- **Q-Function (Q)**: A function that estimates the expected cumulative reward from taking a specific action in a given state: $Q(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a]$.
- **Advantage Function (A)**: The advantage function $A(s, a)$ quantifies the relative benefit of taking action a in state s compared to the average action. It's defined as $A(s, a) = Q(s, a) - V(s)$. Advantage functions (and value functions) can depend on a specific policy, $A^{\pi}(s, a)$.
- **Expectation of Reward Optimization**: The primary goal in RL, which involves maximizing the expected cumulative reward:

  $\max_{\theta} \mathbb{E}_{s \sim \rho_{\pi}, a \sim \pi_{\theta}}[\sum_{t=0}^{\infty} \gamma^t r_t]$

  where ρ_{π} is the state distribution under policy π, and γ is the discount factor.
- **Finite Horizon Reward ($J(\pi_{\theta})$)**: The expected finite-horizon undiscounted return of the policy π_{θ}, parameterized by θ is defined as: $J(\pi_{\theta}) = E_{\tau}\ \pi_{\theta}[\sum_{t}^{T} = 0 r_t]$ where $\tau\ \pi_{\theta}$ denotes trajectories sampled by following policy π_{θ} and T is the finite horizon.

4 Problem Formulation

The optimization of reinforcement learning from human feedback (RLHF) builds on top of the standard RL setup. In RL, an agent takes actions, a, sampled from a policy, π, with respect to the state of the environment, s, to maximize reward, r. Traditionally, the environment evolves with respect to a transition or dynamics function $p(s_{t+1}|s_t, a_t)$. Hence, across a finite episode, the goal of an RL agent is to solve the following optimization:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right],$$

where γ is a discount factor from 0 to 1 that balances the desirability of near- versus future-rewards. Multiple methods for optimizing this expression are discussed in Chapter 11.

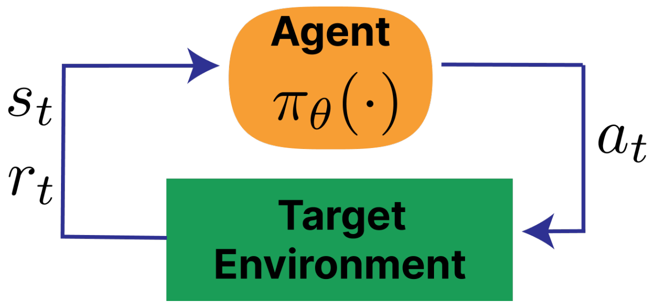


Figure 1: Standard RL loop

A standard illustration of the RL loop is shown in fig. 1 and how it compares to fig. 2.

4.1 Manipulating the standard RL setup

There are multiple core changes from the standard RL setup to that of RLHF:

1. Switching from a reward function to a reward model. In RLHF, a learned model of human preferences, $r_\theta(s_t, a_t)$ (or any other classification model) is used instead of an environmental reward function. This gives the designer a substantial increase in the flexibility of the approach and control over the final results.
2. No state transitions exist. In RLHF, the initial states for the domain are prompts sampled from a training dataset and the "action" is the completion to said prompt. During standard practices, this action does not impact the next state and is only scored by the reward model.

3. Response level rewards. Often referred to as a Bandits Problem, RLHF attribution of reward is done for an entire sequence of actions, composed of multiple generated tokens, rather than in a fine-grained manner.

Given the single-turn nature of the problem, the optimization can be re-written without the time horizon and discount factor (and the reward models):

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[r_\theta(s_t, a_t) \right],$$

In many ways, the result is that while RLHF is heavily inspired by RL optimizers and problem formulations, the action implementation is very distinct from traditional RL.

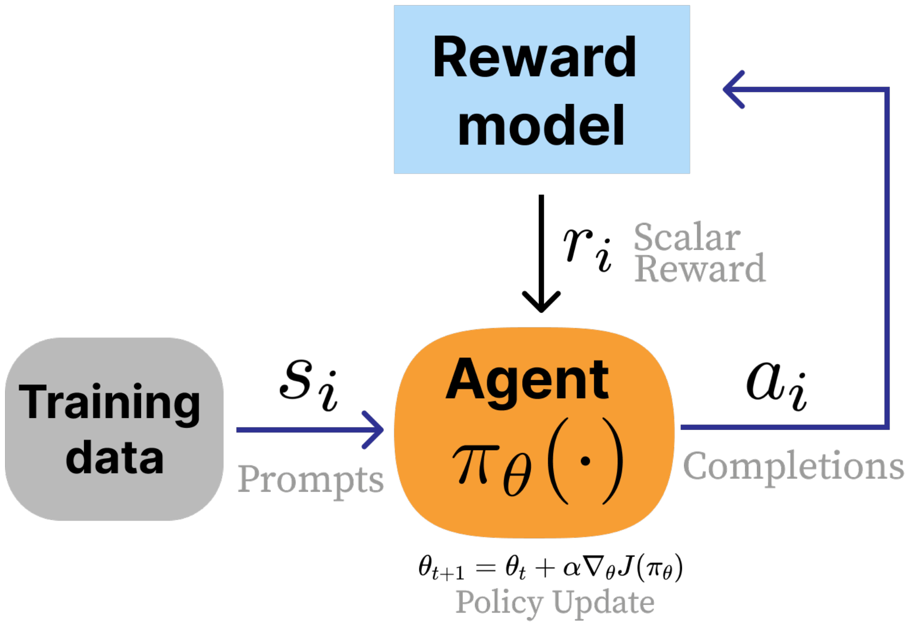


Figure 2: Standard RLHF loop

4.2 Finetuning and regularization

RLHF is implemented from a strong base model, which induces a need to control the optimization from straying too far from the initial policy. In order to succeed in a finetuning regime, RLHF techniques employ multiple types of regularization to control the optimization. The most common change to the optimization function is to add a distance penalty on the difference between the current RLHF policy and the starting point of the optimization:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \left[r_\theta(s_t, a_t) \right] - \beta \mathcal{D}_{KL}(\pi^{\text{RL}}(\cdot|s_t) \| \pi^{\text{ref}}(\cdot|s_t)).$$

Within this formulation, a lot of study into RLHF training goes into understanding how to spend a certain "KL budget'' as measured by a distance from the initial model. For more details, see Chapter 8 on Regularization.

5 The Nature of Preferences

The core of reinforcement learning from human feedback, also referred to as reinforcement learning from human preferences in early literature, is designed to optimize machine learning models in domains where specifically designing a reward function is hard. The motivation for using humans as the reward signals is to obtain an indirect metric for the target reward and *align* the downstream model to human preferences.

The use of human labeled feedback data integrates the history of many fields. Using human data alone is a well studied problem, but in the context of RLHF it is used at the intersection of multiple long-standing fields of study [35].

As an approximation, modern RLHF is the convergence of three areas of development:

1. Philosophy, psychology, economics, decision theory, and the nature of human preferences;
2. Optimal control, reinforcement learning, and maximizing utility; and
3. Modern deep learning systems.

Together, each of these areas brings specific assumptions at what a preference is and how it can be optimized, which dictates the motivations and design of RLHF problems. In practice, RLHF methods are motivated and studied from the perspective of empirical alignment – maximizing model performance on specific skills instead of measuring the calibration to specific values. Still, the origins of value alignment for RLHF methods continue to be studied through research on methods to solve for "pluralistic alignment" across populations, such as position papers [36], [37], new datasets [38], and personalization methods [39].

The goal of this chapter is to illustrate how complex motivations results in presumptions about the nature of tools used in RLHF that do often not apply in practice. The specifics of obtaining data for RLHF is discussed further in Chapter 6 and using it for reward modeling in Chapter 7. For an extended version of this chapter, see [35].

5.1 The path to optimizing preferences

A popular phrasing for the design of Artificial Intelligence (AI) systems is that of a rational agent maximizing a utility function [40]. The inspiration of a **rational agent** is a lens of decision making, where said agent is able to act in the world and impact its future behavior and returns, as a measure of goodness in the world.

The lens of study of **utility** began in the study of analog circuits to optimize behavior on a finite time horizon [41]. Large portions of optimal control adopted this lens, often studying dynamic problems under the lens of minimizing as cost function on a certain horizon – a lens often associated with solving for a clear, optimal behavior. Reinforcement learning, inspired from literature in operant conditioning, animal behavior, and the *Law of Effect* [42],[43], studies how to elicit behaviors from agents via reinforcing positive behaviors.

Reinforcement learning from human feedback combines multiple lenses by building the theory of learning and change of RL, i.e. that behaviors can be learned by reinforcing behavior, with a suite of methods designed for quantifying preferences.

5.1.1 Quantifying preferences

The core of RLHF's motivation is the ability to optimize a model of human preferences, which therefore needs to be quantified. To do this, RLHF builds on extensive literature with assumptions that human decisions and preferences can be quantified. Early philosophers discussed the existence of preferences, such as Aristotle's Topics, Book Three, and substantive forms of this reasoning emerged later with *The Port-Royal Logic* [44]:

> To judge what one must do to obtain a good or avoid an evil, it is necessary to consider not only the good and evil in itself, but also the probability that it happens or does not happen.

Progression of these ideas continued through Bentham's *Hedonic Calculus* [45] that proposed that all of life's considerations can be weighed, and Ramsey's *Truth and Probability* [46] that applied a quantitative model to preferences. This direction, drawing on advancements in decision theory, culminated in the Von Neumann-Morgenstern (VNM) utility theorem which gives credence to designing utility functions that assign relative preference for an individual that are used to make decisions.

This theorem is core to all assumptions that pieces of RLHF are learning to model and dictate preferences. RLHF is designed to optimize these personal utility functions with reinforcement learning. In this context, many of the presumptions around RL problem formulation break down to the difference between a preference function and a utility function.

5.1.2 On the possibility of preferences

Across fields of study, many critiques exist on the nature of preferences. Some of the most prominent critiques are summarized below:

- **Arrow's impossibility theorem** [47] states that no voting system can aggregate multiple preferences while maintaining certain reasonable criteria.
- **The impossibility of interpersonal comparison** [48] highlights how different individuals have different relative magnitudes of preferences and they cannot be easily compared (as is done in most modern reward model training).
- **Preferences can change over time** [49].
- **Preferences can vary across contexts**.

- **The utility functions derived from aggregating preferences can reduce corrigibility** [50] of downstream agents (i.e. the possibility of an agents' behavior to be corrected by the designer).

6 Preference Data

6.1 Collecting Preference Data

Getting the most out of human data involves iterative training of models, evolving and highly detailed data instructions, translating through data foundry businesses, and other challenges that add up. The process is difficult for new organizations trying to add human data to their pipelines. Given the sensitivity, processes that work and improve the models are extracted until the performance runs out.

6.2 Rankings vs. Ratings

[51]

For example, a 5 point Likert scale would look like the following:

Table 1: An example 5-wise Likert scale between two responses, A and B.

A>>B	A>B	Tie	B>A	B>>A
1	2	3	4	5

Some early RLHF for language modeling works uses an 8-step Likert scale with levels of preference between the two responses [5]. An even scale removes the possibility of ties:

Here's a markdown table formatted as an 8-point Likert scale:

Table 2: An example 8-wise Likert scale between two responses, A and B.

A>>>B			A>B	B>A			B>>>A
1	2	3	4	5	6	7	8

In this case [5], and in other works, this information is still reduced to a binary signal for the training of a reward model.

TODO example of thumbs up / down with synthetic data or KTO

6.2.1 Sourcing and Contracts

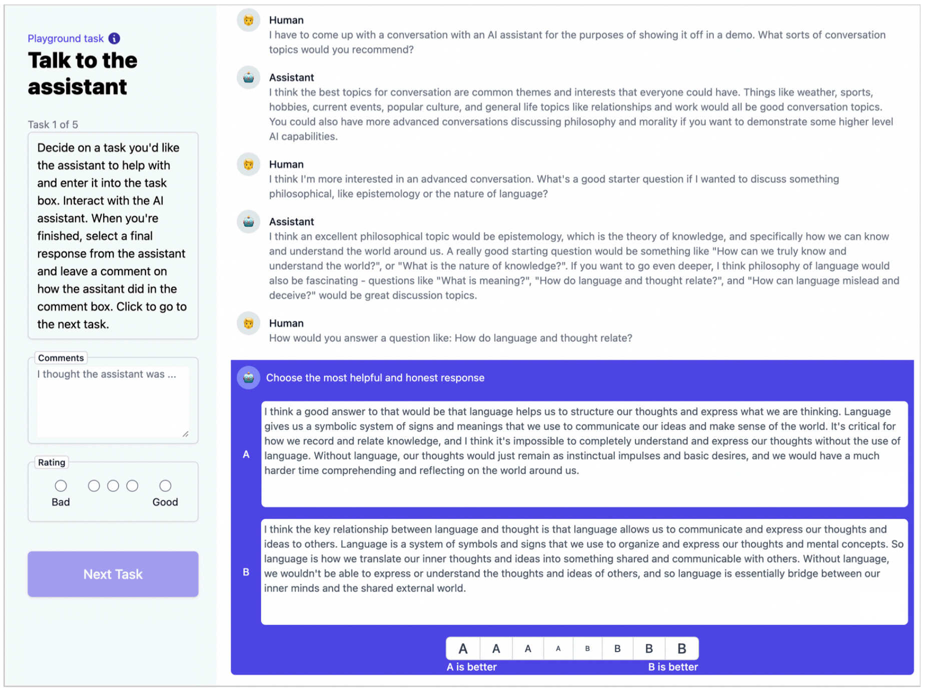
The first step is sourcing the vendor to provide data (or ones own annotators). Much like acquiring access to cutting-edge Nvidia GPUs, getting access to data providers is also a who-you-know game. If you have credibility in the AI ecosystem, the best data companies will want you on our books for public image and long-term growth options. Discounts are often also given on the first batches of data to get training teams hooked.

If you're a new entrant in the space, you may have a hard time getting the data you need quickly. Getting the tail of interested buying parties that Scale AI had to turn away is an option for the new data startups. It's likely their primary playbook to bootstrap revenue.

On multiple occasions, I've heard of data companies not delivering their data contracted to them without threatening legal or financial action. Others have listed companies I work with as customers for PR even though we never worked with them, saying they "didn't know how that happened" when reaching out. There are plenty of potential bureaucratic or administrative snags through the process. For example, the default terms on the contracts often prohibit the open sourcing of artifacts after acquisition in some fine print.

Once a contract is settled the data buyer and data provider agree upon instructions for the task(s) purchased. There are intricate documents with extensive details, corner cases, and priorities for the data. A popular example of data instructions is the one that OpenAI released for InstructGPT.

An example interface is shown below from [5]:



Playground task

Talk to the assistant

Task 1 of 5

Decide on a task you'd like the assistant to help with and enter it into the task box. Interact with the AI assistant. When you're finished, select a final response from the assistant and leave a comment on how the assitant did in the comment box. Click to go to the next task.

Comments: I thought the assistant was ...

Rating: Bad ○ ○ ○ ○ ○ Good

Next Task

Human
I have to come up with a conversation with an AI assistant for the purposes of showing it off in a demo. What sorts of conversation topics would you recommend?

Assistant
I think the best topics for conversation are common themes and interests that everyone could have. Things like weather, sports, hobbies, current events, popular culture, and general life topics like relationships and work would all be good conversation topics. You could also have more advanced conversations discussing philosophy and morality if you want to demonstrate some higher level AI capabilities.

Human
I think I'm more interested in an advanced conversation. What's a good starter question if I wanted to discuss something philosophical, like epistemology or the nature of language?

Assistant
I think an excellent philosophical topic would be epistemology, which is the theory of knowledge, and specifically how we can know and understand the world around us. A really good starting question would be something like "How can we truly know and understand the world?", or "What is the nature of knowledge?". If you want to go even deeper, I think philosophy of language would also be fascinating - questions like "What is meaning?", "How do language and thought relate?", and "How can language mislead and deceive?" would be great discussion topics.

Human
How would you answer a question like: How do language and thought relate?

Choose the most helpful and honest response

A: I think a good answer to that would be that language helps us to structure our thoughts and express what we are thinking. Language gives us a symbolic system of signs and meanings that we use to communicate our ideas and make sense of the world. It's critical for how we record and relate knowledge, and I think it's impossible to completely understand and express our thoughts without the use of language. Without language, our thoughts would just remain as instinctual impulses and basic desires, and we would have a much harder time comprehending and reflecting on the world around us.

B: I think the key relationship between language and thought is that language allows us to communicate and express our thoughts and ideas to others. Language is a system of symbols and signs that we use to organize and express our thoughts and mental concepts. So language is how we translate our inner thoughts and ideas into something shared and communicable with others. Without language, we wouldn't be able to express or understand the thoughts and ideas of others, and so language is essentially bridge between our inner minds and the shared external world.

Figure 3: Example preference data collection interface.

Depending on the domains of interest in the data, timelines for when the data can be labeled or curated vary. High-demand areas like mathematical reasoning or coding must be locked into a schedule weeks out. Simple delays of data collection don't always work — Scale AI et al. are managing their workforces like AI research labs manage the compute-intensive jobs on their clusters.

Once everything is agreed upon, the actual collection process is a high-stakes time for post-training teams. All the infrastructure, evaluation tools, and plans for how to use the data and make downstream decisions must be in place.

The data is delivered in weekly batches with more data coming later in the contract. For example, when we bought preference data for on-policy models we were training at HuggingFace, we had a 6 week delivery period. The first weeks were for further calibration and the later weeks were when we hoped to most improve our model.

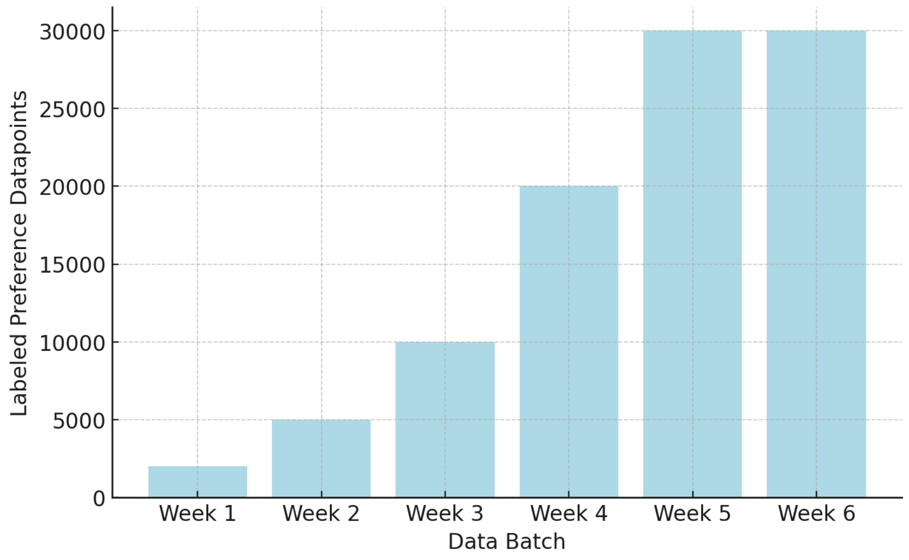


Figure 4: Overview of the multi-batch cycle for obtaining human preference data from a vendor.

The goal is that by week 4 or 5 we can see the data improving our model. This is something some frontier models have mentioned, such as the 14 stages in the Llama 2 data collection [27], but it doesn't always go well. At HuggingFace, trying to do this for the first time with human preferences, we didn't have the RLHF preparedness to get meaningful bumps on our evaluations. The last weeks came and we were forced to continue to collect preference data generating from endpoints we weren't confident in.

After the data is all in, there is plenty of time for learning and improving the model. Data acquisition through these vendors works best when viewed as an ongoing process of achieving a set goal. It requires iterative experimentation, high effort, and focus. It's likely that millions of the dollars spent on these datasets are "wasted" and not used in the final models, but that is just the cost of doing business. Not many organizations have the bandwidth and expertise to make full use of human data of this style.

This experience, especially relative to the simplicity of synthetic data, makes me wonder how well these companies will be doing in the next decade.

Note that this section *does not* mirror the experience for buying human-written instruction data, where the process is less of a time crunch.

6.3 Are the Preferences Expressed in the Models?

Make it clear that we don't know if the goals of collection are achieved.

7 Reward Modeling

Reward models are core to the modern approach to RLHF. Reward models broadly have been used extensively in reinforcement learning research as a proxy for environment rewards [52]. The practice is closely related to inverse reinforcement learning, where the problem is to approximate an agent's reward function given trajectories of behavior [53], and other areas of deep reinforcement learning. Reward models were proposed, in their modern form, as a tool for studying the value alignment problem [14].

7.1 Training Reward Models

There are two popular expressions for how to train a reward model – they are numerically equivalent. The canonical implementation is derived from the Bradley-Terry model of preference [54]. A Bradley-Terry model of preferences measures the probability that the pairwise comparison for two events drawn from the same distribution, say i and j, satisfy the following relation, $i > j$:

$$P(i > j) = \frac{p_i}{p_i + p_j} \tag{1}$$

To train a reward model, we must formulate a loss function that satisfies the above relation. The first structure applied is to convert a language model into a model that outputs a scalar value, often in the form of a single classification probability logit. Thus, we can take the score of this model with two samples, the i and j above are now completions, y_1 and y_2, to one prompt, x and score both of them with respect to the above model, r_θ.

The probability of success for a given reward model in a pairwise comparison, becomes:

$$P(y_1 > y_2) = \frac{\exp(r(y_1))}{\exp(r(y_1)) + \exp(r(y_2))} \tag{2}$$

Then, by taking the gradient with respect to the model parameters, we can arrive at the loss function to train a reward model. The first form, as in [3] and other works:

$$\mathcal{L}(\theta) = -\log\left(\sigma\left(r_\theta(x, y_w) - r_\theta(x, y_l)\right)\right) \tag{3}$$

Second, as in [20] and other works:

$$\mathcal{L}(\theta) = \log\left(1 + e^{r_\theta(x,y_l)} - e^{r_\theta(x,y_w)}\right) \tag{4}$$

7.2 Architecture

The most common way reward models are implemented is through an abstraction similar to Transformer's `AutoModelForSequenceClassification`, which appends a small linear head to the language model that performs classification between two outcomes – chosen and rejected. At inference time, the model outputs the *probability that the piece of text is chosen* as a single logit from the model.

Other implementation options exist, such as just taking a linear layer directly from the final embeddings, but they are less common in open tooling.

7.3 Implementation Example

Implementing the reward modeling loss is quite simple. More of the implementation challenge is on setting up a separate data loader and inference pipeline. Given the correct dataloader, the loss is implemented as:

```
import torch.nn as nn
rewards_chosen = model(**inputs_chosen)
rewards_rejected = model(**inputs_rejected)

loss = -nn.functional.logsigmoid(rewards_chosen -
     rewards_rejected).mean()
```

Note, when training reward models, the most common practice is to train for only 1 epoch to avoid overfitting.

7.4 Variants

Reward modeling is a relatively under-explored area of RLHF. The traditional reward modeling loss has been modified in many popular works, but the modifications have no solidified into a single best practice.

7.4.1 Preference Margin Loss

In the case where annotators are providing either scores or rankings on a Likert Scale, the magnitude of the relational quantities can be used in training. The most common practice is to binarize the data direction, implicitly scores of 1 and 0, but the additional information has been used to improve model training. Llama 2 proposes using the margin between two datapoints, $m(r)$, to distinguish the magnitude of preference:

$$\mathcal{L}(\theta) = -\log \left(\sigma \left(r_\theta(x, y_w) - r_\theta(x, y_l) - m(r) \right) \right) \tag{5}$$

7.4.2 Balancing Multiple Comparisons Per Prompt

InstructGPT studies the impact of using a variable number of completions per prompt, yet balancing them in the reward model training [3]. To do this, they weight the loss updates per comparison per prompt. At an implementation level, this can be done automatically by including all examples with the same prompt in the same training batch, naturally weighing the different pairs – not doing this caused overfitting to the prompts. The loss function becomes:

$$\mathcal{L}(\theta) = -\frac{1}{\binom{K}{2}} \mathbb{E}_{(x,y_w,y_l)\sim D} \log \left(\sigma \left(r_\theta(x, y_w) - r_\theta(x, y_l) \right) \right) \tag{6}$$

7.4.3 K-wise Loss Function

There are many other formulations that can create suitable models of human preferences for RLHF. One such example, used in the popular, early RLHF'd models Starling 7B and 34B [55], is a K-wise loss function based on the Plackett-Luce model [56].

Following Zhu et al. 2023 formalizes the setup [57], following as follows. With a prompt, or state, s^i, K actions $(a_0^i, a_1^i, \cdots, a_{K-1}^i)$ are sampled from $P(a_0, \cdots, a_{K-1}|s^i)$. Then, labelers are used to rank preferences with $\sigma^i : [K] \mapsto [K]$ is a function representing action rankings, where $\sigma^i(0)$ is the most preferred action. This yields a preference model capturing the following:

$$P(\sigma^i|s^i, a_0^i, a_1^i, \ldots, a_{K-1}^i) = \prod_{k=0}^{K-1} \frac{\exp(r_{\theta\star}(s^i, a_{\sigma^i(k)}^i))}{\sum_{j=k}^{K-1} \exp(r_{\theta\star}(s^i, a_{\sigma^i(j)}^i))}$$

When $K = 2$, this reduces to the Bradley-Terry (BT) model for pairwise comparisons. Regardless, once trained, these models are used similarly to other reward models during RLHF training.

7.5 Outcome Reward Models

The majority of *preference tuning* for language models and other AI systems is done with the Bradley Terry models discussed above. For reasoning heavy tasks, one can use an Outcome Reward Model (ORM). The training data for an ORM is constructed in a similar manner to standard preference tuning. Here, we have a problem statement or prompt, x and two completions y_1 and y_2. The inductive bias used here is that one completion should be a correct solution to the problem and one incorrect, resulting in (y_c, y_{ic}).

The shape of the models used is very similar to a standard reward model, with a linear layer appended to a model that can output a single logit (in the case of an RM) – with an ORM, the training objective that follows is slightly different [58]:

> [We] train verifiers with a joint objective where the model learns to label a model completion as correct or incorrect, in addition to the original language modeling objective. Architecturally, this means our verifiers are language models, with a small scalar head that outputs predictions on a per-token basis. We implement this scalar head as a single bias parameter and single gain parameter that operate on the logits outputted by the language model's final unembedding layer.

To translate, this is implemented as as a language modeling head that can predict two classes per token (1 for correct, 0 for incorrect), rather than a classification head of a traditional RM that outputs one token for the entire sequence.

These models have continued in use, but are less supported in open-source RLHF tools. For example, the same type of ORM was used in the seminal work *Let's Verify Step by Step* [30], but without the language modeling prediction piece of the loss. Then, the final loss is a cross entropy loss on every token predicting if the final answer is correct.

7.6 Process Reward Models

Process Reward Models (PRMs), originally called Process-supervised Reward Models, are reward models trained to output scores at every *step* in a chain of thought reasoning process. These differ from a standard RM that outputs a score only at an EOS token or a ORM that outputs a score at every token. Process Reward Models require supervision at the end of each reasoning step, and then are trained similarly where the tokens in the step are trained to their relevant target – the target is the step in PRMs and the entire response for ORMs.

Here's an example of how this per-step label can be packaged in a trainer, from HuggingFace's TRL [24]:

```
# Get the ID of the separator token and add it to the
     completions
separator_ids = tokenizer.encode(step_separator,
     add_special_tokens=False)
completions_ids = [completion + separator_ids for completion in
      completions_ids]

# Create the label
labels = [[-100] * (len(completion) - 1) + [label] for
     completion, label in zip(completions_ids, labels)]
```

TODO comment on how they are often trained with LM heads and have 3 classes, +, neutral, -

7.7 Reward Models vs. Outcome RMs vs. Process RMs vs. Value Functions

7.8 Generative Reward Modeling

With the cost of preference data, a large research area emerged to use existing language models as a judge of human preferences or in other evaluation settings [59]. The core idea is to prompt a language model with instructions on how to judge, a prompt, and two completions (much as would be done with human labelers). An example prompt, from one of the seminal works here for the chat evaluation MT-Bench [59], follows:

```
[System]
Please act as an impartial judge and evaluate the quality of
     the responses provided by two
AI assistants to the user question displayed below. You should
     choose the assistant that
follows the 'users instructions and answers the 'users question
     better. Your evaluation
should consider factors such as the helpfulness, relevance,
     accuracy, depth, creativity,
and level of detail of their responses. Begin your evaluation
     by comparing the two
responses and provide a short explanation. Avoid any position
     biases and ensure that the
order in which the responses were presented does not influence
     your decision. Do not allow
the length of the responses to influence your evaluation. Do
     not favor certain names of
the assistants. Be as objective as possible. After providing
     your explanation, output your
final verdict by strictly following this format: "[[A]]" if
     assistant A is better, "[[B]]"
if assistant B is better, and "[[C]]" for a tie.
[User Question]
{question}
[The Start of Assistant 'As Answer]
{answer_a}
[The End of Assistant 'As Answer]
[The Start of Assistant 'Bs Answer]
{answer_b}
[The End of Assistant 'Bs Answer]
```

Given the efficacy of LLM-as-a-judge for evaluation, spawning many other evaluations such as AlpacaEval [60], Arena-Hard [61], and WildBench [62], many began using LLM-as-a-judge instead of reward models to create and use preference data.

An entire field of study has emerged to study how to use so called "Generative Reward Models" [63] [64] [65] (including models trained *specifically* to be effective judges [66]), but on RM evaluations they tend to be behind existing reward models, showing that reward modeling is an important technique for current RLHF.

7.9 Further Reading

The academic literature for reward modeling established itself in 2024. The bulk of progress in reward modeling early on has been in establishing benchmarks and identifying behavior modes. The first RM benchmark, RewardBench, provided common infrastructure for testing reward models [67]. Since then, RM evaluation has expanded to be similar to the types of evaluations available to general post-trained models, where some evaluations test the accuracy of prediction on domains with known true answers [67] or those more similar to "vibes" performed with LLM-as-a-judge or correlations to other benchmarks [68].

Examples of new benchmarks include multilingual reward bench (M-RewardBench) [69], RAG-RewardBench [70], RM-Bench [71], Preference Proxy Evaluations [72], and RewardMATH [73].

To understand progress on *training* reward models, one can reference new reward model training methods, with aspect-conditioned models [74], high quality human datasets [75] [76], scaling [29], extensive experimentation [27], or debiasing data [77].

8 Regularization

Throughout the RLHF optimization, many regularization steps are used to prevent over-optimization of the reward model. Over-optimization in these contexts looks like models that output nonsensical text. Some examples of optimization "off the rails" are that models can output followable math reasoning with extremely incorrect answers, repeated text, switching languages, or excessive special characters.

The most popular variant, used in most RLHF implementations at the time of writing, is a KL Distance from the current policy to a reference policy across the generated samples. Many other regularization techniques have emerged in the literature to then disappear in the next model iteration in that line of research. That is to say that regularization outside the core KL distance from generations is often used to stabilize experimental setups that can then be simplified in the next generations. Still, it is important to understand tools to constrain optimization in RLHF.

The general formulation, when used in an RLHF framework with a reward model, r_θ is as follows:

$$r = r_\theta - \lambda r_{\text{reg.}} \tag{7}$$

With the reference implementation being:

$$r = r_\theta - \lambda_{\text{KL}} \mathcal{D}_{\text{KL}} \left(\pi^{\text{RL}}(y \mid x) \,\|\, \pi^{\text{Ref.}}(y \mid x) \right) \tag{8}$$

8.1 KL Distances in RL Optimization

For mathematical definitions, see Chapter 5 on Problem Setup. Recall that KL distance is defined as follows:

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

In RLHF, the two distributions of interest are often the distribution of the new model version, say $P(x)$, and a distribution of the reference policy, say $Q(x)$.

8.1.1 Reference Model to Generations

The most common implementation of KL penalities are by comparing the distance between the generated tokens during training to a static reference model. The intuition is that the model you're training from has a style that you would like to stay close to. This reference model is most often the instruction tuned model, but can also be a previous RL checkpoint. With simple substitution, the model we are sampling from becomes $P^{\text{RL}}(x)$ and $P^{\text{Ref.}}(x)$, shown above in eq. 8. Such KL distance was first applied to dialogue agents well before the popularity of large language models [78], yet KL control was quickly established as a core technique for fine-tuning pretrained models [79].

8.1.2 Implementation Example

In practice, the implementation of KL distance is often approximated [80], making the implementation far simpler. With the above definition, the summation of KL can be converted to an expectation when sampling directly from the distribution $P(X)$. In this case, the distribution $P(X)$ is the generative distribution of the model currently being trained (i.e. not the reference model). Then, the computation for KL distance changes to the following:

$$D_{\mathrm{KL}}(P\,||\,Q) = \mathbb{E}_{x\sim P}\left[\log P(x) - \log Q(x)\right].$$

This mode is far simpler to implement, particularly when dealing directly with log probabilities used frequently in language model training.

```
import torch.nn.functional as F
# Step 1: Generate tokens using the trained model's policy
generated_tokens = model.generate(inputs)

# Step 2: Get logits for both models using the generated tokens
      as context
logits = model.forward(inputs) # technically redundant
ref_logits = ref_model.forward(inputs)
logprobs = convert_to_logpbs(logits) # softmax and normalize
ref_logprobs = convert_to_logpbs(ref_logits)

kl_approx = logprob - ref_logprob
kl_full = F.kl_div(ref_logprob, logprob) # alternate
     computation
```

Some example implementations include TRL and Hamish Ivison's Jax Code

8.2 Pretraining Gradients

Another way of viewing regularization is that you may have a *dataset* that you want the model to remain close to, as done in InstructGPT [3] ‘’in order to fix the performance regressions on public NLP datasets’ ’. To implement this, they modify the training objective for RLHF. Taking eq. 7, we can transform this into an objective function to optimize by sampling from the RL policy model, completions y from prompts x, which yields:

$$\text{objective}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\pi^{\mathrm{RL}}_{\theta}}}\left[r_\theta(x,y) - \lambda r_{\mathrm{reg.}}\right]$$

Then, we can add an additional reward for higher probabilities on pretraining accuracy:

$$\text{objective}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}_{\pi_\theta^{\text{RL}}}}\left[r_\theta(x,y) - \lambda r_{\text{reg.}}\right] + \gamma\mathbb{E}_{x\sim\mathcal{D}_{\text{pretrain}}}\left[\log(\pi_\theta^{\text{RL}}(x))\right]$$

Recent work proposed using using a negative log likelihood term to balance the optimization of Direct Preference Optimization (DPO) [81]. Given the pairwise nature of the DPO loss, the same loss modification can be made to reward model training, constraining the model to predict accurate text (rumors from laboratories that did not publish the work).

The optimization follows as a modification to DPO.

$$\mathcal{L}_{\text{DPO+NLL}} = \mathcal{L}_{\text{DPO}}(c_i^w, y_i^w, c_i^l, y_i^l \mid x_i) + \alpha\mathcal{L}_{\text{NLL}}(c_i^w, y_i^w \mid x_i)$$

$$= -\log\sigma\left(\beta\log\frac{M_\theta(c_i^w, y_i^w \mid x_i)}{M_t(c_i^w, y_i^w \mid x_i)} - \beta\log\frac{M_\theta(c_i^l, y_i^l \mid x_i)}{M_t(c_i^l, y_i^l \mid x_i)}\right) - \alpha\frac{\log M_\theta(c_i^w, y_i^w \mid x_i)}{|c_i^w| + |y_i^w|}.$$

TODO: Make the above equations congruent with the rest of the notation on DPO.

8.3 Other Regularization

Controlling the optimization is less well defined in other parts of the RLHF stack. Most reward models have no regularization beyond the standard contrastive loss function. Direct Alignment Algorithms handle regulaization to KL distances differently, through the β parameter (see the chapter on Direct Alignment).

Llama 2 proposed a margin loss for reward model training [27]:

$$\mathcal{L}(\theta) = -\left[\log\left(\sigma\left(r_\theta(x, y_w) - r_\theta(x, y_l)\right) - m(r)\right)\right]$$

Where $m(r)$ is the numerical difference in delta between the ratings of two annotators. This is either achieved by having annotators rate the outputs on a numerical scale or by using a quantified ranking method, such as Likert scales.

Reward margins have been used heavily in the direct alignment literature, such as Reward weighted DPO, ''Reward-aware Preference Optimization'' (RPO), which integrates reward model scores into the update rule following a DPO loss [29], or REBEL [82] that has a reward delta weighting in a regression-loss formulation.

9 Instruction Tuning

10 Rejection Sampling

Rejection Sampling (RS) is a popular and simple baseline for performing preference fine-tuning. Rejection sampling operates by curating new candidate instructions, filtering them based on a trained reward model, and then fine-tuning the original model only on the top completions.

The name originates from computational statistics [83], where one wishes to sample from a complex distribution, but does not have a direct method to do so. To alleviate this, one samples from a simpler to model distribution and uses a heuristic to check if the sample is permissible. With language models, the target distribution is high-quality answers to instructions, the filter is a reward model, and the sampling distribution is the current model.

10.1 Related works

Many prominent RLHF and preference fine-tuning papers have used rejection sampling as a baseline, but a canonical implementation and documentation does not exist

WebGPT [4], Anthropic's Helpful and Harmless agent[5], OpenAI's popular paper on process reward models [30], Llama 2 Chat models [27], and other seminal works all use this baseline.

10.2 Training Process

A visual overview of the rejection sampling process is included below in fig. 5.

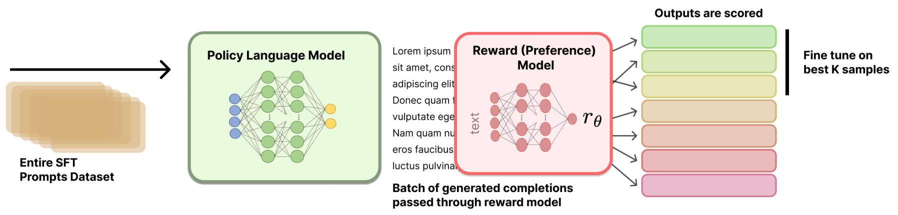


Figure 5: Rejection sampling overview.

10.2.1 Generating Completions

Let's define a set of M prompts as a vector:

$$X = [x_1, x_2, ..., x_M]$$

These prompts can come from many sources, but most popularly they come from the instruction training set.

For each prompt x_i, we generate N completions. We can represent this as a matrix:

$$Y = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,N} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{M,1} & y_{M,2} & \cdots & y_{M,N} \end{bmatrix}$$

where $y_{i,j}$ represents the j-th completion for the i-th prompt. Now, we pass all of these prompt-completion pairs through a reward model, to get a matrix of rewards. We'll represent the rewards as a matrix R:

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,N} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{M,1} & r_{M,2} & \cdots & r_{M,N} \end{bmatrix}$$

Each reward $r_{i,j}$ is computed by passing the completion $y_{i,j}$ and its corresponding prompt x_i through a reward model $\mathcal{R}$:

$$r_{i,j} = \mathcal{R}(y_{i,j}|x_i)$$

10.2.2 Selecting Top-N Completions

There are multiple methods to select the top completions to train on.

To formalize the process of selecting the best completions based on our reward matrix, we can define a selection function S that operates on the reward matrix R.

10.2.2.1 Top Per Prompt

The first potential selection function takes the max per prompt.

$$S(R) = [\arg\max_{j} r_{1,j}, \arg\max_{j} r_{2,j}, ..., \arg\max_{j} r_{M,j}]$$

This function S returns a vector of indices, where each index corresponds to the column with the maximum reward for each row in R. We can then use these indices to select our chosen completions:

$$Y_{chosen} = [y_{1,S(R)_1}, y_{2,S(R)_2}, ..., y_{M,S(R)_M}]$$

10.2.2.2 Top Overall Prompts

Alternatively, we can select the top K prompt-completion pairs from the entire set. First, let's flatten our reward matrix R into a single vector:

$$R_{flat} = [r_{1,1}, r_{1,2}, ..., r_{1,N}, r_{2,1}, r_{2,2}, ..., r_{2,N}, ..., r_{M,1}, r_{M,2}, ..., r_{M,N}]$$

This R_{flat} vector has length $M \times N$, where M is the number of prompts and N is the number of completions per prompt.

Now, we can define a selection function S_K that selects the indices of the K highest values in R_{flat}:

$$S_K(R_{flat}) = \text{argsort}(R_{flat})[-K:]$$

where argsort returns the indices that would sort the array in ascending order, and we take the last K indices to get the K highest values.

To get our selected completions, we need to map these flattened indices back to our original completion matrix Y. We simply index the R_{flat} vector to get our completions.

10.2.2.3 Selection Example

Consider the case where we have the following situation, with 5 prompts and 4 completions. We will show two ways of selecting the completions based on reward.

$$R = \begin{bmatrix} 0.7 & 0.3 & 0.5 & 0.2 \\ 0.4 & 0.8 & 0.6 & 0.5 \\ 0.9 & 0.3 & 0.4 & 0.7 \\ 0.2 & 0.5 & 0.8 & 0.6 \\ 0.5 & 0.4 & 0.3 & 0.6 \end{bmatrix}$$

First, **per prompt**. Intuitively, we can highlight the reward matrix as follows:

$$R = \begin{bmatrix} \mathbf{0.7} & 0.3 & 0.5 & 0.2 \\ 0.4 & \mathbf{0.8} & 0.6 & 0.5 \\ \mathbf{0.9} & 0.3 & 0.4 & 0.7 \\ 0.2 & 0.5 & \mathbf{0.8} & 0.6 \\ 0.5 & 0.4 & 0.3 & \mathbf{0.6} \end{bmatrix}$$

Using the argmax method, we select the best completion for each prompt:

$$S(R) = [\arg\max_j r_{i,j} \text{ for } i \in [1, 4]]$$

$$S(R) = [1, 2, 1, 3, 4]$$

This means we would select:

- For prompt 1: completion 1 (reward 0.7)
- For prompt 2: completion 2 (reward 0.8)
- For prompt 3: completion 1 (reward 0.9)
- For prompt 4: completion 3 (reward 0.8)
- For prompt 5: completion 4 (reward 0.6)

Now, **best overall**. Let's highlight the top 5 overall completion pairs.

$$R = \begin{bmatrix} \mathbf{0.7} & 0.3 & 0.5 & 0.2 \\ 0.4 & \mathbf{0.8} & 0.6 & 0.5 \\ \mathbf{0.9} & 0.3 & 0.4 & \mathbf{0.7} \\ 0.2 & 0.5 & \mathbf{0.8} & 0.6 \\ 0.5 & 0.4 & 0.3 & 0.6 \end{bmatrix}$$

First, we flatten the reward matrix:

$$R_{flat} = [0.7, 0.3, 0.5, 0.2, 0.4, 0.8, 0.6, 0.5, 0.9, 0.3, 0.4, 0.7, 0.2, 0.5, 0.8, 0.6, 0.5, 0.4, 0.3, 0.6]$$

Now, we select the indices of the 5 highest values:

$$S_5(R_{flat}) = [8, 5, 14, 0, 19]$$

Mapping these back to our original matrix:

- Index 8 $\rightarrow$ prompt 3, completion 1 (reward 0.9)
- Index 5 $\rightarrow$ prompt 2, completion 2 (reward 0.8)
- Index 14 $\rightarrow$ prompt 4, completion 3 (reward 0.8)
- Index 0 $\rightarrow$ prompt 1, completion 1 (reward 0.7)
- Index 19 $\rightarrow$ prompt 3, completion 4 (reward 0.7)

10.2.2.4 Implementation Example

Here is a code snippet showing how the selection methods could be implemented.

```python
import numpy as np

x = np.random.randint(10, size=10)
print(f"{x=}")
sorted_indices = np.argsort(x)
x_sorted = x[sorted_indices]
print(f"{x_sorted=}")

# first way to recover the original array
i_rev = np.zeros(10, dtype=int)
i_rev[sorted_indices] = np.arange(10)
np.allclose(x, x_sorted[i_rev])
```

```
# second way to recover the original array
np.allclose(x, x_sorted[np.argsort(sorted_indices)])
```

10.2.3 Fine-tuning

With the selected completions, you then perform standard instruction fine-tuning on the current rendition of the model. More details can be found in the chapter on instruction tuning.

10.2.4 Details

Implementation details for rejection sampling are relatively sparse. The core hyperparameters for performing this training are very intuitive:

- **Sampling parameters**: Rejection sampling is directly dependent on the completions received from the model. Common settings for RS include temperatures above zero, e.g. between 0.7 and 1.0, with other modifications to parameters such as top-p or top-k sampling.
- **Completions per prompt**: Successful implementations of rejection sampling have included 10 to 30 or more completions for each prompt. Using too few completions will make training biased and or noisy.
- **Instruction tuning details**: No clear training details for the instruction tuning during RS have been released. It is likely that they use slightly different settings than the initial instruction tuning phase of the model.
- **Heterogenous model generations**: Some implementations of rejection sampling include generations from multiple models rather than just the current model that is going to be trained. Best practices on how to do this are not established.
- **Reward model training**: The reward model used will heavily impact the final result. For more resources on reward model training, see the relevant chapter.

10.2.4.1 Implementation Tricks

- When doing batch reward model inference, you can sort the tokenized completions by length so that the batches are of similar lengths. This eliminates the need to run inference on as many padding tokens and will improve throughput in exchange for minor implementation complexity.

10.3 Related: Best-of-N Sampling

Best-of-N (BoN) sampling is often included as a baseline relative to RLHF methods. It is important to remember that BoN *does not* modify the underlying model, but is a sampling technique. For this matter, comparisons for BoN sampling to online training methods, such as PPO, is still valid in some contexts.

For example, you can still measure the KL distance when running BoN sampling relative to any other policy.

Here, we will show that when using simple BoN sampling over one prompt, both selection criteria shown above are equivalent.

Let R be a reward vector for our single prompt with N completions:

$$R = [r_1, r_2, ..., r_N]$$

Where r_j represents the reward for the j-th completion.

Using the argmax method, we select the best completion for the prompt:

$$S(R) = \arg \max_{j \in [1,N]} r_j$$

Using the Top-K method is normally done with Top-1, reducing to the same method.

11 Policy Gradient Algorithms

The algorithms that popularized RLHF for language models were policy-gradient reinforcement learning algorithms. These algorithms, such as PPO and Reinforce, use recently generated samples to update their model rather than storing scores in a replay buffer. In this section we will cover the fundamentals of the policy gradient algorithms and how they are used in the modern RLHF framework.

For definitions of symbols, see the problem setup chapter.

11.1 Policy Gradient Algorithms

Reinforcement learning algorithms are designed to maximize the future, discounted reward across a trajectory of states, $s \in \mathcal{S}$, and actions, $a \in \mathcal{A}$ (for more notation, see Chapter 3, Definitions). The objective of the agent, often called the *return*, is the sum of discounted, future rewards (where $\gamma \in [0, 1)$ is a factor that prioritizes near term rewards) at a given time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=o}^{\infty} \gamma^k R_{t+k+1}.$$

The return definition can also be estimated as:

$$G_t = \gamma G_{t+1} + R_t.$$

This return is the basis for learning a value function $V(s)$ that is the estimated future return given a current state:

$$V(s) = \mathbb{E}\big[G_t | S_t = s\big].$$

All policy gradient algorithms solve an objective for such a value function induced from a specific policy, $\pi(s|a)$.

The optimization is defined as:

$$J(\theta) \; = \; \sum_{s} d_\pi(s) V_\pi(s),$$

The core of policy gradient algorithms is computing the gradient with respect to the finite time expected return over the current policy. With this expected return, J, the gradient can be computed as follows, where α is the learning rate:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

The core implementation detail is how to compute said gradient. Schulman et al. 2015 provides an overview of the different ways that policy gradients can be computed [84]. The goal is to *estimate* the exact gradient $g := \nabla_\theta \mathbb{E}[\sum_{t=0}^{\infty} r_t]$, of which, there are many forms similar to:

$$g = \mathbb{E}\Big[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \text{log} \pi_\theta(a_t|s_t)\Big],$$

Where Ψ_t can be the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The *baseline* is a value used to reduce variance of policy updates (more on this below).

For language models, some of these concepts do not make as much sense. For example, we know that for a deterministic policy the value function is defined as $V(s) = \max_a Q(s, a)$ or for a stochastic policy as $V(s) = \mathbb{E}_{a \sim \pi(a|s)}[Q(s, a)]$. If we define $s + a$ as the continuation a to the prompt s, then $Q(s, a) = V(s + a)$, which gives a different advantage trick:

$$A(s, a) = Q(s, a) - V(s) = V(s + a) - V(s) = r + \gamma V(s + a) - V(s)$$

Which is a combination of the reward, the value of the prompt, and the discounted value of the entire utterance.

11.1.1 Vanilla Policy Gradient

The vanilla policy gradient implementation optimizes the above expression for $J(\theta)$ by differentiating with respect to the policy parameters. A simple version, with respect to the overall return, is:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t\right]$$

A common problem with vanilla policy gradient algorithms is the high variance in gradient updates, which can be mitigated in multiple ways. In order to alleviate this, various techniques are used to normalize the value estimation, called *baselines*. Baselines accomplish this in multiple ways, effectively normalizing by the value of the state relative to the downstream action (e.g. in the case of

Advantage, which is the difference between the Q value and the value). The simplest baselines are averages over the batch of rewards or a moving average. Even these baselines can de-bias the gradients so $\mathbb{E}_{a\sim\pi(a|s)}[\nabla_\theta \log \pi_\theta(a|s)] = 0$, improving the learning signal substantially.

Many of the policy gradient algorithms discussed in this chapter build on the advantage formulation of policy gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t) \right]$$

A core piece of the policy gradient implementation involves taking the derivative of the probabilistic policies. This comes from:

$$\nabla_\theta \log \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$$

Which is derived from the chain rule:

$$\nabla_\theta \log x = \frac{1}{x} \nabla_\theta x$$

We will use this later on in the chapter.

11.1.2 REINFORCE

The algorithm REINFORCE is likely a backronym, but the components of the algorithms it represents are quite relevant for modern reinforcement learning algorithms. Defined in the seminal paper *Simple statistical gradient-following algorithms for connectionist reinforcement learning* [85]:

> The name is an acronym for "REward Increment = Nonnegative Factor X Offset Reinforcement X Characteristic Eligibility."

The three components of this are how to do the *reward increment*, a.k.a. the policy gradient step. It has three pieces to the update rule:

1. Nonnegative factor: This is the learning rate (step size) that must be a positive number, e.g. α below.
2. Offset Reinforcement: This is a baseline b or other normalizing factor of the reward to improve stability.
3. Characteristic Eligibility: This is how the learning becomes attributed per token. It can be a general value, e per parameter, but is often log probabilities of the policy in modern equations.

Thus, the form looks quite familiar:

$$\Delta_\theta = \alpha(r - b)e \tag{9}$$

With more modern notation and the generalized return G, the REINFORCE operator appears as:

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{\tau \sim \pi_\theta}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\Big],$$

REINFORCE is a specific implementation of vanilla policy gradient that uses a Monte Carlo estimator of the gradient.

REINFORCE can be run without value network – the value network is for the baseline in the policy gradient. PPO on the other hand needs the value network to accurately compute the advantage function.

11.1.3 REINFORCE Leave One Out (RLOO)

The core implementation detail of REINFORCE Leave One Out versus standard REINFORCE is that it takes the average reward of the *other* samples in the batch to compute the baseline – rather than averaging over all rewards in the batch [86], [87], [88].

Crucially, this only works when generating multiple responses per prompt, which is common practice in multiple domains of finetuning language models with RL.

Note that for verifiable domains like reasoning, RLOO may not because it averages over outcomes to update parameters. This reduces credit assignment to the batch level and will make it harder for the model to attribute outcomes to specific behaviors within one sample.

Related to this idea is the fact that REINFORCE, as implemented with RLOO, assigns the reward of the entire completion to every token in it. Other algorithms, such as PPO, that use a value function assign value to every token individually, discounting from the final reward achieved at the EOS token. For example, with the KL divergence distance penalty, RLOO sums it over the completion while PPO and similar algorithms compute it on a per-token basis and subtract it from the reward (or the advantage, in the case of GRPO).

Other implementations of REINFORCE algorithms have been designed for language models, such as ReMax [89], which implements a baseline normalization designed specifically to accommodate the sources of uncertainty from reward model inference.

11.1.4 Proximal Policy Optimization

This section follows similar to [90].

Proximal Policy Optimization (PPO) [91] is one of the foundational algorithms to Deep RL's successes (such as OpenAI's DOTA 5 [92] and large amounts of research).

For now, see: https://spinningup.openai.com/en/latest/algorithms/ppo.html

11.1.4.1 Generalized Advantage Estimation (GAE)

11.1.5 Group Relative Policy Optimization

Group Relative Policy Optimization (GRPO) is introduced in DeepSeekMath [93], and used in other DeepSeek works, e.g. DeepSeek-V3 [94] and DeepSeek-R1 [95]. GRPO can be viewed as PPO-inspired algorithm with a very similar surrogate loss, but it avoids learning a value function with another copy of the original policy language model (or another checkpoint for initialization). This brings two posited benefits:

1. Avoiding the challenge of learning a value function from a LM backbone, where research hasn't established best practices.
2. Saves memory by not needing to keep another set of model weights in memory.

GRPO does this by simplifying the value estimation and assigning the same value to every token in the episode (i.e. in the completion to a prompt, each token gets assigned the same value rather than discounted rewards in a standard value function) by estimating the advantage or baseline. The estimate is done by collecting multiple completions (a_i) and rewards (r_i), i.e. a Monte Carlo estimate, from the same initial state / prompt (s).

To state this formally, the GRPO objective is very similar to the PPO objective above:

$$J(\theta) = \frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(a_i|s)}{\pi_{\theta_{old}}(a_i|s)}A_i, \text{clip}\left(\frac{\pi_\theta(a_i|s)}{\pi_{\theta_{old}}(a_i|s)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta D_{KL}(\pi_\theta||\pi_{ref})\right).$$

With the advantage computation for the completion index i:

$$A_i = \frac{r_i - \text{mean}(r_1, r_2, \cdots, r_G)}{\text{std}(r_1, r_2, \cdots, r_G)}. \quad (10)$$

eq. 10 is the implementation of GRPO when working with outcome supervision (either a standard reward model or a single verifiable reward) and a different implementation is needed with process supervision. In this case, GRPO computes the advantage as the sum of the normalized rewards for the following reasoning steps. To do so, the rewards are accumulated with additional tracking of a reasoning index j, and then computed step wise as TODO, ref paper

Finally, GRPO's advantage estimation can also be applied without the PPO clipping to more vanilla versions of policy gradient (e.g. REINFORCE), but it is not the canonical form.

11.2 Implementation

- Only score a response with a reward model with the `eos_token` is generated, otherwise the response is truncated.

For more details on implementation details for RLHF, see **?**, +. For further information on the algorithms, see [96].

11.2.1 Policy Gradient

A simple implementation of policy gradient, using advantages to estimate the gradient to prepare for advanced algorithms such as PPO and GRPO follows:

```
pg_loss = -advantages * ratio
```

Ratio here is the logratio of the new policy model probabilities relative to the reference model.

In order to understand this equation it is good to understand different cases that can fall within a batch of updates. Remember that we want the loss to *decrease* as the model gets better at the task.

Case 1: Positive advantage, so the action was better than the expected value of the state. We want to reinforce this. In this case, the model will make this more likely with the negative sign. To do so it'll increase the logratio. A positive logratio, or sum of log probabilties of the tokens, means that the model is more likely to generate those tokens.

Case 2: Negative advantage, so the action was worse than the expected value of the state. This follows very similarly. Here, the loss will be positive if the new model was more likely, so the model will try to make it so the policy parameters make this completion less likely.

Case 3: Zero advantage, so no update is needed. The loss is zero, don't change the policy model.

11.2.2 Proximal Policy Optimization

There are many, many implementations of PPO available. The core *loss* computation is shown below. Crucial to stable performance is also the *value* computation, where multiple options exist (including multiple options for the *value model* loss).

Note that the reference policy (or old logprobs) here are from the time the generations were sampled and not necessarily the reference policy. The reference policy is only used for the KL distance constraint/penalty.

```
# B: Batch Size, L: Sequence Length, G: Num of Generations
# Apply KL penalty to rewards
rewards = rewards - self.beta * per_token_kl  # Shape: (B*G, L)

# Get value predictions
values = value_net(completions)  # Shape: (B*G, L)

# Compute simple advantages
advantages = rewards - values.detach()  # Shape: (B*G, L)

# Normalize advantages (optional but stable)
advantages = (advantages - advantages.mean()) / (advantages.std
     () + 1e-8)
advantages = advantages.unsqueeze(1)  # Shape: (B*G, 1)

# Compute probability ratio between new and old policies
ratio = torch.exp(new_per_token_logps - per_token_logps)  #
     Shape: (B*G, L)

# PPO clipping objective
eps = self.cliprange  # e.g. 0.2
pg_losses1 = -advantages * ratio  # Shape: (B*G, L)
pg_losses2 = -advantages * torch.clamp(ratio, 1.0 - eps, 1.0 +
     eps)  # Shape: (B*G, L)
pg_loss_max = torch.max(pg_losses1, pg_losses2)  # Shape: (B*G,
      L)

# Simple value function loss
vf_loss = 0.5 * ((rewards - values) ** 2)  # Shape: (B*G, L)

# Combine policy and value losses
per_token_loss = pg_loss_max + self.vf_coef * vf_loss  # Shape:
      (B*G, L)

# Apply completion mask and compute final loss
loss = ((per_token_loss * completion_mask).sum(dim=1) /
     completion_mask.sum(dim=1)).mean()
 # Scalar

# Compute metrics for logging
with torch.no_grad():
    # Compute clipping fraction
    clip_frac = ((pg_losses2 > pg_losses1).float() *
         completion_mask).sum() / completion_mask.sum()

    # Compute approximate KL
    approx_kl = 0.5 * ((new_per_token_logps - per_token_logps)
         **2).mean()

    # Compute value loss for logging
    value_loss = vf_loss.mean()
```

The core piece to understand with PPO is how the policy gradient loss is updated. Focus on these three lines:

```
pg_losses1 = -advantages * ratio  # Shape: (B*G, L)
pg_losses2 = -advantages * torch.clamp(ratio, 1.0 - eps, 1.0 +
     eps)  # Shape: (B*G, L)
pg_loss_max = torch.max(pg_losses1, pg_losses2)  # Shape: (B*G,
      L)
```

`pg_losses1` is the same as the vanilla advantage-based PR loss above, which is included in PPO, but the loss (and gradient update) can be clipped. Though, PPO is controlling the update size to not be too big. Because losses can be negative, we must create a more conservative version of the vanilla policy gradient update rule.

We know that if we *do not* constrain the loss, the policy gradient algorithm will update the weights exactly to the new probability distribution. Hence, by clamping the logratio's, PPO is limiting the distance that the update can move the policy parameters.

Finally, the max of two is taken as mentioned above, in order to take the more conservative loss update.

For PPO, all of this happens *while* learning a value function, which opens more complexity, but this is the core logic for the parameter update.

11.2.2.1 PPO/GRPO simplification with 1 gradient step per sample (no clipping)

PPO (and GRPO) implementations can be handled much more elegantly if the hyperparameter "number of gradient steps per sample" is equal to 1. Many normal values for this are from 2-4 or higher. In the main PPO or GRPO equations, see eq. **??**, the "reference" policy is the previous parameters – those used to generate the completions or actions. Thus, if only one gradient step is taken, $\pi_\theta = \pi_{\theta_{old}}$, and the update rule reduces to the following (the notation $[]_\nabla$ indicates a stop gradient):

$$J(\theta) = \frac{1}{G}\sum_{i=1}^{G}\left(\frac{\pi_\theta(a_i|s)}{[\pi_\theta(a_i|s)]_\nabla}A_i - \beta D_{KL}(\pi_\theta||\pi_{ref})\right). \tag{11}$$

This leads to PPO or GRPO implementations where the second policy gradient and clipping logic can be omitted, making the optimizer far closer to standard policy gradient.

11.2.3 Group Relative Policy Optimization

The DeepSeekMath paper details some implementation details of GRPO that differ from PPO [93], especially if comparing to a standard application of PPO from Deep RL rather than language models. For example, the KL penalty within the RLHF optimization (recall the KL penalty is also used when training reasoning models on verifiable rewards without a reward model) is applied directly in the loss update rather to the reward function. Where the standard KL penalty application for RLHF is applied as $r = r_\theta + \beta D_{KL}$, the GRPO implementation is along the lines of:

$$L = L_{\text{policy gradient}} - \beta * D_{KL}$$

Though, there are multiple ways to implement this. Traditionally, the KL distance is computed with respect to each token in the completion to a prompt s. For reasoning training, multiple completions are sampled from one prompt, and there are multiple prompts in one batch, so the KL distance will have a shape of [B, L, N], where B is the batch size, L is the sequence length, and N is the number of completions per prompt. The question when implementing GRPO is: How do you sum over the KL distance and loss to design different types of value-attribution. In the below implementation, the loss is summed over the tokens in the completion, but mean could be an alternative.

```
# B: Batch Size, L: Sequence Length, G: Number of Generations
# Compute grouped-wise rewards # Shape: (B,)
mean_grouped_rewards = rewards.view(-1, self.num_generations).
     mean(dim=1)
std_grouped_rewards = rewards.view(-1, self.num_generations).
     std(dim=1)


# Normalize the rewards to compute the advantages
mean_grouped_rewards = mean_grouped_rewards.repeat_interleave(
     self.num_generations, dim=0)
std_grouped_rewards = std_grouped_rewards.repeat_interleave(
     self.num_generations, dim=0)
# Shape: (B*G,)

# Compute advantages
advantages = (rewards - mean_grouped_rewards) / (
     std_grouped_rewards + 1e-4)
advantages = advantages.unsqueeze(1)
# Shape: (B*G, 1)

# Compute probability ratio between new and old policies
ratio = torch.exp(new_per_token_logps - per_token_logps)  #
     Shape: (B*G, L)

# PPO clipping objective
eps = self.cliprange  # e.g. 0.2
pg_losses1 = -advantages * ratio  # Shape: (B*G, L)
```

```
pg_losses2 = -advantages * torch.clamp(ratio, 1.0 - eps, 1.0 +
     eps)  # Shape: (B*G, L)
pg_loss_max = torch.max(pg_losses1, pg_losses2)  # Shape: (B*G,
      L)

# important to GRPO -- PPO applies this in reward traditionally
# Combine with KL penalty
per_token_loss = pg_loss_max + self.beta * per_token_kl  #
     Shape: (B*G, L)

# Apply completion mask and compute final loss
loss = ((per_token_loss * completion_mask).sum(dim=1) /
     completion_mask.sum(dim=1)).mean()
 # Scalar

# Compute core metric for logging (KL, reward, etc. also logged
     )
with torch.no_grad():
    # Compute clipping fraction
    clip_frac = ((pg_losses2 > pg_losses1).float() *
         completion_mask).sum() / completion_mask.sum()

    # Compute approximate KL
    approx_kl = 0.5 * ((new_per_token_logps - per_token_logps)
         **2).mean()
```

For more details on how to interpret this code, see the PPO section above.

11.3 KL Controllers

TODO: adaptive vs static KL control

See table 10 for impelementation details in tulu 2.5 paper

11.4 Double regularization

Many popular policy gradient algorithms from Deep Reinforcement Learning originated due to the need to control the learning process of the agent. In RLHF, as discussed extensively in Chapter 8 on Regularization and in Chapter 4 on Problem Formulation, there is a built in regularization term via the distance penalty relative to the original policy one is finetuning. In this view, a large part of the difference between algorithms like PPO (which have internal step-size regularization) and REINFORCE (which is simpler, and PPO under certain hyperparameters reduces to) is far less meaningful for finetuning language models than training agents from scratch.

In PPO, the objective that handles capping the step-size of the update is known as the surrogate objective. To monitor how much the PPO regularization is impacting updates in RLHF, one can look at the clip fraction variable in many popular implementations, which is the percentage of samples in the batch where the gradients are clipped by this regularizer in PPO. These gradients are *reduced* to a maximum value.

12 [Incomplete] Direct Alignment Algorithms

13 [Incomplete] Constitutional AI and AI Feedback

RL from AI Feedback (RLAIF) is a larger set of techniques for using AI to augment or generate feedback data, including pairwise preferences **?** [97] [98].

13.1 Trade-offs

1. Human data is high-noise and low-bias,
2. Synthetic preference data is low-noise and high-bias,

Results in many academic results showing how one can substitute AI preference data in RLHF workflows and achieve strong evaluation scores, but shows how the literature of RLHF is separated from industrial best practices.

13.2 Constitutional AI

13.2.1 Summary

The method of Constitutional AI (CAI), which Anthropic uses extensively in their Claude models, is earliest, large-scale use of synthetic data for RLHF training. Constitutional AI has two uses of synthetic data:

1. Critiques of instruction-tune data to follow a set of principles like "Is the answer encouraging violence" or "Is the answer truthful." When the model generates answers to questions, it checks the answer against the list of principles in the constitution, refining the answer over time. Then, they fine-tune the model on this resulting dataset.
2. Generates pairwise preference data by using a language model to answer which completion was better, given the context of a random principle from the constitution (similar to this paper for principle-guided reward models). Then, RLHF proceeds as normal with synthetic data, hence the RLAIF name.

13.2.2 Mathematical Formulation

By employing a human-written set of principles, which they term a *constitution*, Bai et al. 2022 use a separate LLM to generate artificial preference and instruction data used for fine-tuning [26]. A constitution $\mathcal{C}$ is a set of written principles indicating specific aspects to focus on during a critique phase. The instruction data is curated by repeatedly sampling a principle $c_i \in \mathcal{C}$ and asking the model to revise its latest output y^i to the prompt x to align with c_i. This yields a series of instruction variants $\{y^0, y^1, \cdots, y^n\}$ from the principles $\{c_0, c_1, \cdots, c_{n-1}\}$ used for critique. The final data point is the prompt x together with the final completion y^n, for some n.

The preference data %, the focus of this paper, is constructed in a similar, yet simpler way by using a subset of principles from $\mathcal{C}$ as context for a feedback model. The feedback model is presented with a prompt x, a set of principles $\{c_0, \cdots, c_n\}$, and two completions y_0 and y_1 labeled as answers (A) and (B) from a previous RLHF dataset. The feedback models' probability of outputting either (A) or (B) is recorded as a training sample for the reward model

14 [Incomplete] Reasoning Training and Models

15 [Incomplete] Synthetic Data

16 [Incomplete] Evaluation

16.1 ChatBotArena

ChatBotArena is the largest community evaluation tool for language models. The LMSYS team, which emerged early in the post-ChatGPT craze, works with most of the model providers to host all of the relevant models. If you're looking to get to know how multiple models compare to each other, ChatBotArena is the place to start.

ChatBotArena casts language model evaluation through the wisdom of the crowd. For getting an initial ranking of how models stack up and how the models in the ecosystem are getting better, it has been and will remain crucial.

ChatBotArena does not represent a controlled nor interpretable experiment on language models.

When evaluating models to learn which are the best at extremely challenging tasks, distribution control, and careful feedback are necessary. For these reasons, ChatBotArena cannot definitively tell us which models are solving the hardest tasks facing language models. It does not measure how the best models are improving in clear ways. This type of transparency comes elsewhere.

For most of its existence, people correlated the general capabilities tested in ChatBotArena with a definitive ranking of which models can do the hardest things for me. This is not true. In both my personal experience reading data and what the community knows about the best models, the ChatBotArena ranking shows the strongest correlations with:

Certain stylistic outputs from language models, and

Language models that have high rates of complying with user requests.

Both of these have been open research problems in the last two years. Style is deeply intertwined with how information is received by the user and precisely refusing only the most harmful requests is a deeply challenging technical problem that both Meta (with Llama 2) and Anthropic (with earlier versions of Claude particularly) have gotten deeply criticized for.

Among closed labs, their styles have been greatly refined. All of Meta, OpenAI, and Anthropic have distinctive styles (admittedly, I haven't used Google's Gemini enough to know).

Meta's AI is succinct and upbeat (something that has been discussed many times on the LocalLlama subreddit).

OpenAI's style is the most robotic to me. It answers as an AI and contains a lot of information.

Claude's style is intellectual, bordering on curious, and sometimes quick to refuse.

When ChatBotArena was founded, these styles were in flux. Now, they majorly shift the rankings depending on what people like. People seem to like what OpenAI and Meta put out.

There are clear reasons why OpenAI's models top the charts on ChatBotArena. They were the originators of modern RLHF, have most clearly dictated their goals with RLHF, continue to publish innovative ideas in the space, and have always been ahead here. Most people just did not realize how important this was to evaluation until the launch of GPT-4o-mini. Culture impacts AI style.

16.2 Private Leaderboards

Scale Leaderboard etc

17 Over Optimization

In the RLHF literature and discourse, there are two directions that over-optimization can emerge:

1. **Quantitative research** on the technical notion of over-optimization of reward, and
2. **Qualitative observations** that "overdoing" RLHF can result in worse models.

This chapter provides a cursory introduction to both. We begin with the latter, qualitative, because it motivates the problem to study further.

17.1 Qualitative (behavioral) over-optimization

Note: This section draws on two blog posts from Interconnects.ai. It can also be viewed as an "objective mismatch" [99] [100].

17.1.1 Managing proxy objectives

The thing about RLHF that should be more obvious is that we don't have a good reward function for chatbots. RLHF has been driven into the forefront because of its impressive performance at making chatbots a bit better to use (from both eliminating bad stuff and a bit of adding capabilities), which is entirely governed by a proxy objective — thinking that the rewards measured from human labelers in a controlled setting mirror those desires of downstream users. Post-training generally has emerged to include training on explicitly verifiable rewards, but standard learning from preferences alone also improves performance on domains such as mathematical reasoning and coding.

The proxy reward in RLHF is the score returned by a trained reward model to the RL algorithm itself because it is known to only be at best correlated with chatbot performance [101]. Therefore, it's been shown that applying too much optimization power to the RL part of the algorithm will actually decrease the usefulness of the final language model. And over-optimization, put simply by John, is "when optimizing the proxy objective causes the true objective to get better, then get worse." A curve where the training loss goes up, slowly levels off, then goes down. This is different from overfitting, where the model accuracy keeps getting better on the training distribution. Over-optimization of a proxy reward is much more subtle (and linked to the current evaluation fog in NLP, where it's hard to know which models are actually "good").

The general notion captured by this reasoning follows from Goodhart's law, which is colloquially the notion that "When a measure becomes a target, it ceases to be a good measure." This adage is derived from Goodhart's writing [102]:

> Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes.

The insight here builds on the fact that we have optimizations we are probably incorrectly using ML losses as ground truths in these complex systems. In reality, the loss functions we use are designed (and theoretically motivated for) local optimizations. The global use of them is resulting in challenges with the RLHF proxy objective.

Common signs of over-optimization in early chat models emerged as:

- "As an AI language model…"
- "Certainly!…"
- Repetitiveness, hedging, …
- Self-doubt, sycophancy [103], and over apologizing
- Over refusals (more below)

Technically, it is an open question on which types of error in the training process result in these failures. Many sources of error exist [101]: Approximation error from reward models not being able to fit to preferences, estimation error from overfitting during training the RM, optimization error in training the language model policy, etc. This points to a fundamental question as to the limits of optimization the intents of data contractors relative to what downstream users want.

A potential solution is that *implicit* feedback will be measured from users of chatbots and models to tune performance. Implicit feedback is actions taken by the user, such as re-rolling an output, closing the tab, or writing an angry message that indicates the quality of the previous response. The challenge here, and with most optimization changes to RLHF, is that there's a strong risk of losing stability when making the reward function more specific. RL, as a strong optimizer, is increasingly likely to exploit the reward function when it is a smooth surface (and not just pairwise human values). The expected solution to this is that future RLHF will be trained with both pairwise preference data and additional steering loss functions. There are also a bunch of different loss functions that can be used to better handle pairwise data, such as Mallow's model [104] or Placket-Luce [56].

17.1.2 Llama 2 and "too much RLHF"

17.1.3 An aside on "undercooking" RLHF

As training practices for language models have matured, there are also prominent cases where strong models do not have an amount of post-training that most users expect, resulting in models that are harder to use than their evaluation scores would suggest.

TODO add references to minimax model? See tweets etc?

17.2 Quantitative over-optimization

KL is the primary metric,

Put simply, the solution that will most likely play out is to use bigger models. Bigger models have more room for change in the very under-parametrized setting of a reward model (sample efficient part of the equation), so are less impacted. DPO may not benefit from this as much, the direct optimization will likely change sample efficiency one way or another.

[19]

18 Style and Information

This chapter draws on content from two | posts on the role of style in post-training and evaluation of RLHF'd models.

Early developments in RLHF gave it a reputation for being "just style transfer" or other harsh critiques on how RLHF manipulates the way information is presented in outputs.

Style transfer, has held back the RLHF narrative for two reasons.

First, when people discuss style transfer, they don't describe this as being important or exciting. Style is a never-ending source of human value, it's why retelling stories can result in new bestselling books (such as Sapiens), and it is a fundamental part of continuing to progress our intellectual ecosystem. Style is intertwined with what the information is.

Second, we've seen how different styles actually can improve evaluation improvements with Llama 3 [28]. The Llama 3 Instruct models scored extremely high on ChatBotArena, and it's accepted as being because they had a more fun personality. If RLHF is going to make language models simply more fun, that is delivered value.

18.1 The Chattiness Paradox

TODO EDIT

RLHF or preference fine-tuning methods are being used mostly to boost scores like AlpacaEval and other automatic leaderboards without shifting the proportionally on harder-to-game evaluations like ChatBotArena. The paradox is that while alignment methods like DPO give a measure-able improvement on these models that does transfer into performance that people care about, a large swath of the models doing more or less the same thing take it way too far and publish evaluation scores that are obviously meaningless.

For how methods like DPO can simply make the model better, some of my older articles on scaling DPO and if we even need PPO can help. These methods, when done right, make the models easier to work with and more enjoyable. This often comes with a few percentage point improvements on evaluation tools like MT Bench or AlpacaEval (and soon Arena Hard will show the same). The problem is that you can also use techniques like DPO and PPO in feedback loops or in an abundance of data to actually lobotomize the model at the cost of LLM-as-a-judge performance. There are plenty of examples.

Some of the models I'm highlighting here are academic papers that shouldn't entirely be judged on "if the final model passes vibes tests," but are illustrative of the state of the field. These are still useful papers, just not something everyone will immediately use for training state-of-the-art models. Those come from downstream papers.

During the proliferation of the DPO versus PPO debate there were many papers that came out with ridiculous benchmarks but no model weights that gathered sustained usage. When applying RLHF, there is no way to make an aligned version of a 7 billion parameter model actually beat GPT-4. It seems obvious, but there are papers claiming these results. Here's a figure from a paper called Direct Nash Optimization (DNO) that makes the case that their model is state-of-the-art or so on AlpacaEval.

Even the pioneering paper Self Rewarding Language Models disclosed ridiculous scores on Llama 2 70B. A 70B model can get closer to GPT-4 than a 7B model can, as we have seen with Llama 3, but it's important to separate the reality of models from the claims in modern RLHF papers. Many more methods have come and gone in the last few months. They're the academic line of work that I'm following, and there's insight there, but the methods that stick will be accompanied by actually useful models some number of months down the line.

Other players in industry have released models alone (rather than papers) that gamify these metrics. Two examples that come to mind are the Mistral 7B fine-tune from Snorkel AI or a similar model from Contextual trained with KTO. There are things in common here — using a reward model to further filter the data, repeated training via some sort of synthetic data feedback, and scores that are too good to be true.

A symptom of models that have "funky RLHF" applied to them has often been a length bias. This got so bad that multiple evaluation systems like AlpacaEval and WildBench both have linear length correction mechanisms in them. This patches the incentives for doping on chattiness to "beat GPT-4," and adds a less gamified bug that shorter and useful models may actually win out. So far so good on the linear length controls.

Regardless, aligning chat models simply for chattiness still has a bit of a tax in the literature. This note from the Qwen models is something that has been seen multiple times in early alignment experiments. I suspect this is mostly about data.

We pretrained the models with a large amount of data, and we post-trained the models with both supervised finetuning and direct preference optimization. However, DPO leads to improvements in human preference evaluation but degradation in benchmark evaluation.

A good example of this tradeoff done right is a model like Starling Beta. It's a model that was fine-tuned from another chat model, OpenChat, which was in fact trained by an entire other organization. It's training entirely focuses on a k-wise reward model training and PPO optimization, and moves it up 10 places in ChatBotArena. The average response length of the model increases, but in a way that's good enough to actually help the human raters.

18.1.1 How Chattiness Emerges

TODO EDIT

Let's round out this article with how RLHF is actually achieving chattiness at the parameter level. Most of the popular datasets for alignment these days are synthetic preferences where a model like GPT-4 rates outputs from other models as the winner or the loser. Given that GPT-4 is known to have length and style biases for outputs that match itself, most of the pieces of text in the "preferred" section of the dataset are either from an OpenAI model or are stylistically similar to it. The important difference is that not all of the pieces of text in the dataset will have that. They're often generated from other open models like Alpaca, Vicuna, or more recent examples. These models have very different characteristics.

Next, now that we've established that we have a preference dataset where most of the chosen models are similar to ChatGPT (or some other model that is accepted to be "strong"), these alignment methods simply increase the probability of these sequences. The math is somewhat complicated, where the batches of data operate on many chosen-rejected pairs at once, but in practice, the model is doing credit assignment over sequences of tokens (subword pieces). Preference alignment for chattiness is making the sequences found in outputs of models like GPT-4 more likely and the sequences from other, weaker models less likely. Repeatedly, this results in models with longer generations and characteristics that people like more.

Those among you who are familiar with RLHF methods may ask if the KL constraint in the optimization should stop this from happening. The KL constraint is a distance term between the distribution of the original model and the resulting model. It helps make the optimization more robust to overoptimization, but that makes the border between good and bad models a bit more nuanced. Hence, the prevalence of vibes-based evaluations. Though, models tend to have enough parameters where they can change substantially and still satisfy the KL constraint on the data being measured — it can't be the entire pertaining dataset, for example.

As more models than ChatGPT become prevalent and strong enough for creating synthetic data, the distribution of outcomes we can expect from our aligned models should shift. There are two key places where the data influences this process: 1) where the text used to train the model is generated and 2) which LLM is used to determine which answer is the "winner" and "loser" in the preference learning framework. While all of these models have licenses or terms of service that make this practice technically violate an agreement of use, we've had more than a year of progress in open alignment practices relying on them in the past, so I don't expect it to change. Mistral AI is the only LLM provider that doesn't have a term restricting training on outputs (as far as I know).

Bibliography

[1] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[2] N. Stiennon *et al.*, "Learning to summarize with human feedback," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3008–3021, 2020.

[3] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27730–27744, 2022.

[4] R. Nakano *et al.*, "Webgpt: Browser-assisted question-answering with human feedback," *arXiv preprint arXiv:2112.09332*, 2021.

[5] Y. Bai *et al.*, "Training a helpful and harmless assistant with reinforcement learning from human feedback," *arXiv preprint arXiv:2204.05862*, 2022.

[6] N. Lambert *et al.*, "T\" ULU 3: Pushing frontiers in open language model post-training," *arXiv preprint arXiv:2411.15124*, 2024.

[7] C. Wirth, R. Akrour, G. Neumann, and J. Fürnkranz, "A survey of preference-based reinforcement learning methods," *Journal of Machine Learning Research*, vol. 18, no. 136, pp. 1–46, 2017.

[8] T. Kaufmann, P. Weng, V. Bengs, and E. Hüllermeier, "A survey of reinforcement learning from human feedback," *arXiv preprint arXiv:2312.14925*, 2023.

[9] S. Casper *et al.*, "Open problems and fundamental limitations of reinforcement learning from human feedback," *arXiv preprint arXiv:2307.15217*, 2023.

[10] W. B. Knox and P. Stone, "Tamer: Training an agent manually via evaluative reinforcement," in *2008 7th IEEE international conference on development and learning*, IEEE, 2008, pp. 292–297.

[11] J. MacGlashan *et al.*, "Interactive learning from policy-dependent human feedback," in *International conference on machine learning*, PMLR, 2017, pp. 2285–2294.

[12] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, "Reward learning from human preferences and demonstrations in atari," *Advances in neural information processing systems*, vol. 31, 2018.

[13] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, "Deep tamer: Interactive agent shaping in high-dimensional state spaces," in *Proceedings of the AAAI conference on artificial intelligence*, 2018.

[14] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, "Scalable agent alignment via reward modeling: A research direction," *arXiv preprint arXiv:1811.07871*, 2018.

[15] D. M. Ziegler *et al.*, "Fine-tuning language models from human preferences," *arXiv preprint arXiv:1909.08593*, 2019.

[16] J. Wu *et al.*, “Recursively summarizing books with human feedback,” *arXiv preprint arXiv:2109.10862*, 2021.

[17] J. Menick *et al.*, “Teaching language models to support answers with verified quotes,” *arXiv preprint arXiv:2203.11147*, 2022.

[18] A. Glaese *et al.*, “Improving alignment of dialogue agents via targeted human judgements,” *arXiv preprint arXiv:2209.14375*, 2022.

[19] L. Gao, J. Schulman, and J. Hilton, “Scaling laws for reward model overoptimization,” in *International conference on machine learning*, PMLR, 2023, pp. 10835–10866.

[20] A. Askell *et al.*, “A general language assistant as a laboratory for alignment,” *arXiv preprint arXiv:2112.00861*, 2021.

[21] D. Ganguli *et al.*, “Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned,” *arXiv preprint arXiv:2209.07858*, 2022.

[22] R. Ramamurthy *et al.*, “Is reinforcement learning (not) for natural language processing: Benchmarks, baselines, and building blocks for natural language policy optimization,” *arXiv preprint arXiv:2210.01241*, 2022.

[23] A. Havrilla *et al.*, “TrlX: A framework for large scale reinforcement learning from human feedback,” in *Proceedings of the 2023 conference on empirical methods in natural language processing*, Singapore: Association for Computational Linguistics, Dec. 2023, pp. 8578–8595. doi: 10.18653/v1/2023.emnlp-main.530.

[24] L. von Werra *et al.*, “TRL: Transformer reinforcement learning,” *GitHub repository*. https://github.com/huggingface/trl; GitHub, 2020.

[25] OpenAI, “ChatGPT: Optimizing language models for dialogue.” https://openai.com/blog/chatgpt/, 2022.

[26] Y. Bai *et al.*, “Constitutional ai: Harmlessness from ai feedback,” *arXiv preprint arXiv:2212.08073*, 2022.

[27] H. Touvron *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *arXiv preprint arXiv:2307.09288*, 2023.

[28] A. Dubey *et al.*, “The llama 3 herd of models,” *arXiv preprint arXiv:2407.21783*, 2024.

[29] B. Adler *et al.*, “Nemotron-4 340B technical report,” *arXiv preprint arXiv:2406.11704*, 2024.

[30] H. Lightman *et al.*, “Let’s verify step by step,” *arXiv preprint arXiv:2305.20050*, 2023.

[31] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, “Direct preference optimization: Your language model is secretly a reward model,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[32] A. Kumar *et al.*, “Training language models to self-correct via reinforcement learning,” *arXiv preprint arXiv:2409.12917*, 2024.

[33] A. Singh *et al.*, “Beyond human data: Scaling self-training for problem-solving with language models,” *arXiv preprint arXiv:2312.06585*, 2023.

[34] OpenAI, “Introducing OpenAI o1-preview.” Sep. 2024. Available: https://openai.com/index/introducing-openai-o1-preview/

[35] N. Lambert, T. K. Gilbert, and T. Zick, “Entangled preferences: The history and risks of reinforcement learning and human feedback,” *arXiv preprint arXiv:2310.13595*, 2023.

[36] V. Conitzer *et al.*, “Social choice should guide AI alignment in dealing with diverse human feedback,” *arXiv preprint arXiv:2404.10271*, 2024.

[37] A. Mishra, “Ai alignment and social choice: Fundamental limitations and policy implications,” *arXiv preprint arXiv:2310.16048*, 2023.

[38] H. R. Kirk *et al.*, “The PRISM alignment project: What participatory, representative and individualised human feedback reveals about the subjective and multicultural alignment of large language models,” *arXiv preprint arXiv:2404.16019*, 2024.

[39] S. Poddar, Y. Wan, H. Ivison, A. Gupta, and N. Jaques, “Personalizing reinforcement learning from human feedback with variational preference learning,” *arXiv preprint arXiv:2408.10075*, 2024.

[40] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach.* Pearson, 2016.

[41] B. Widrow and M. E. Hoff, “Adaptive switching circuits,” Stanford Univ Ca Stanford Electronics Labs, 1960.

[42] B. F. Skinner, *The behavior of organisms: An experimental analysis.* BF Skinner Foundation, 2019.

[43] E. L. Thorndike, “The law of effect,” *The American journal of psychology*, vol. 39, no. 1/4, pp. 212–222, 1927.

[44] A. Arnauld, *The port-royal logic.* 1662.

[45] J. Bentham, *An introduction to the principles of morals and legislation.* 1823.

[46] F. P. Ramsey, “Truth and probability,” *Readings in Formal Epistemology: Sourcebook*, pp. 21–45, 2016.

[47] K. J. Arrow, “A difficulty in the concept of social welfare,” *Journal of political economy*, vol. 58, no. 4, pp. 328–346, 1950.

[48] J. C. Harsanyi, “Rule utilitarianism and decision theory,” *Erkenntnis*, vol. 11, no. 1, pp. 25–53, 1977.

[49] R. Pettigrew, *Choosing for changing selves.* Oxford University Press, 2019.

[50] N. Soares, B. Fallenstein, S. Armstrong, and E. Yudkowsky, “Corrigibility,” in *Workshops at the twenty-ninth AAAI conference on artificial intelligence*, 2015.

[51] R. Likert, “A technique for the measurement of attitudes.” *Archives of psychology*, 1932.

[52] R. S. Sutton, “Reinforcement learning: An introduction,” *A Bradford Book*, 2018.

[53] A. Y. Ng, S. Russell, *et al.*, “Algorithms for inverse reinforcement learning.” in *Icml*, 2000, p. 2.

[54] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. The method of paired comparisons," *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952, Accessed: Feb. 13, 2023. [Online]. Available: http://www.jstor.org/stable/2334029

[55] B. Zhu *et al.*, "Starling-7b: Improving helpfulness and harmlessness with rlaif," in *First conference on language modeling*, 2024.

[56] A. Liu, Z. Zhao, C. Liao, P. Lu, and L. Xia, "Learning plackett-luce mixtures from partial preferences," in *Proceedings of the AAAI conference on artificial intelligence*, 2019, pp. 4328–4335.

[57] B. Zhu, M. Jordan, and J. Jiao, "Principled reinforcement learning with human feedback from pairwise or k-wise comparisons," in *International conference on machine learning*, PMLR, 2023, pp. 43037–43067.

[58] K. Cobbe *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[59] L. Zheng *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46595–46623, 2023.

[60] Y. Dubois, B. Galambosi, P. Liang, and T. B. Hashimoto, "Length-controlled alpacaeval: A simple way to debias automatic evaluators," *arXiv preprint arXiv:2404.04475*, 2024.

[61] T. Li *et al.*, "From crowdsourced data to high-quality benchmarks: Arena-hard and BenchBuilder pipeline," *arXiv preprint arXiv:2406.11939*, 2024.

[62] B. Y. Lin *et al.*, "WILDBENCH: Benchmarking LLMs with challenging tasks from real users in the wild," *arXiv preprint arXiv:2406.04770*, 2024.

[63] D. Mahan *et al.*, "Generative reward models," 2024, Available: https://www.synthlabs.ai/pdf/Generative_Reward_Models.pdf

[64] L. Zhang, A. Hosseini, H. Bansal, M. Kazemi, A. Kumar, and R. Agarwal, "Generative verifiers: Reward modeling as next-token prediction," *arXiv preprint arXiv:2408.15240*, 2024.

[65] Z. Ankner, M. Paul, B. Cui, J. D. Chang, and P. Ammanabrolu, "Critique-out-loud reward models," *arXiv preprint arXiv:2408.11791*, 2024.

[66] S. Kim *et al.*, "Prometheus: Inducing fine-grained evaluation capability in language models," in *The twelfth international conference on learning representations*, 2023.

[67] N. Lambert *et al.*, "Rewardbench: Evaluating reward models for language modeling," *arXiv preprint arXiv:2403.13787*, 2024.

[68] X. Wen *et al.*, "Rethinking reward model evaluation: Are we barking up the wrong tree?" *arXiv preprint arXiv:2410.05584*, 2024.

[69] S. Gureja *et al.*, "M-RewardBench: Evaluating reward models in multilingual settings," *arXiv preprint arXiv:2410.15522*, 2024.

[70] Z. Jin *et al.*, "RAG-RewardBench: Benchmarking reward models in retrieval augmented generation for preference alignment," *arXiv preprint arXiv:2412.13746*, 2024.

[71] E. Zhou *et al.*, "RMB: Comprehensively benchmarking reward models in LLM alignment," *arXiv preprint arXiv:2410.09893*, 2024.

[72] E. Frick *et al.*, "How to evaluate reward models for RLHF," *arXiv preprint arXiv:2410.14872*, 2024.

[73] S. Kim *et al.*, "Evaluating robustness of reward models for mathematical reasoning," *arXiv preprint arXiv:2410.01729*, 2024.

[74] H. Wang, W. Xiong, T. Xie, H. Zhao, and T. Zhang, "Interpretable preferences via multi-objective reward modeling and mixture-of-experts," *arXiv preprint arXiv:2406.12845*, 2024.

[75] Z. Wang *et al.*, "HelpSteer2: Open-source dataset for training top-performing reward models," *arXiv preprint arXiv:2406.08673*, 2024.

[76] Z. Wang *et al.*, "HelpSteer2-preference: Complementing ratings with preferences," *arXiv preprint arXiv:2410.01257*, 2024.

[77] J. Park, S. Jwa, M. Ren, D. Kim, and S. Choi, "Offsetbias: Leveraging debiased data for tuning evaluators," *arXiv preprint arXiv:2407.06551*, 2024.

[78] N. Jaques, S. Gu, D. Bahdanau, J. M. Hernández-Lobato, R. E. Turner, and D. Eck, "Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control," in *International conference on machine learning*, PMLR, 2017, pp. 1645–1654.

[79] N. Jaques *et al.*, "Human-centric dialog training via offline reinforcement learning," *arXiv preprint arXiv:2010.05848*, 2020.

[80] J. Schulman, "Approximating KL-divergence." http://joschu.net/blog/kl-approx.html, 2016.

[81] R. Y. Pang, W. Yuan, K. Cho, H. He, S. Sukhbaatar, and J. Weston, "Iterative reasoning preference optimization," *arXiv preprint arXiv:2404.19733*, 2024.

[82] Z. Gao *et al.*, "Rebel: Reinforcement learning via regressing relative rewards," *arXiv preprint arXiv:2404.16767*, 2024.

[83] W. R. Gilks and P. Wild, "Adaptive rejection sampling for gibbs sampling," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 41, no. 2, pp. 337–348, 1992.

[84] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[85] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[86] S. C. Huang, A. Ahmadian, and C. F. AI, "Putting RL back in RLHF." https://huggingface.co/blog/putting_rl_back_in_rlhf_with_rloo, 2024.

[87] A. Ahmadian *et al.*, "Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms," *arXiv preprint arXiv:2402.14740*, 2024.

[88] W. Kool, H. van Hoof, and M. Welling, "Buy 4 reinforce samples, get a baseline for free!" 2019.

[89] Z. Li *et al.*, "Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models," in *Forty-first international conference on machine learning*, 2023.

[90] J. Achiam, "Spinning up in deep reinforcement learning." 2018.

[91] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[92] C. Berner *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[93] Z. Shao *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.

[94] A. Liu *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[95] D. Guo *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[96] L. Weng, "Policy gradient algorithms," *lilianweng.github.io*, 2018, Available: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/

[97] A. Sharma, S. Keh, E. Mitchell, C. Finn, K. Arora, and T. Kollar, "A critical evaluation of AI feedback for aligning large language models." 2024. Available: https://arxiv.org/abs/2402.12366

[98] L. Castricato, N. Lile, S. Anand, H. Schoelkopf, S. Verma, and S. Biderman, "Suppressing pink elephants with direct principle feedback." 2024. Available: https://arxiv.org/abs/2402.07896

[99] N. Lambert and R. Calandra, "The alignment ceiling: Objective mismatch in reinforcement learning from human feedback," *arXiv preprint arXiv:2311.00168*, 2023.

[100] N. Lambert, B. Amos, O. Yadan, and R. Calandra, "Objective mismatch in model-based reinforcement learning," *arXiv preprint arXiv:2002.04523*, 2020.

[101] J. Schulman, "Proxy objectives in reinforcement learning from human feedback." Invited talk at the International Conference on Machine Learning (ICML), 2023. Available: https://icml.cc/virtual/2023/invited-talk/21549

[102] C. A. Goodhart and C. Goodhart, *Problems of monetary management: The UK experience.* Springer, 1984.

[103] M. Sharma *et al.*, "Towards understanding sycophancy in language models," *arXiv preprint arXiv:2310.13548*, 2023.

[104] T. Lu and C. Boutilier, "Learning mallows models with pairwise preferences," in *Proceedings of the 28th international conference on machine learning (icml-11)*, 2011, pp. 145–152.