Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Matematica

Aligning Large Language Models:

A Study on Reinforcement Learning from Human Feedback

Tesi di Laurea in Machine Learning

Relatore:
Chiar.mo Prof.
Giovanni Paolini

Presentata da:
Federica Del Duchetto

Anno Accademico 2023/2024

A me stessa.

Introduction

Language models are at the core of Natural Language Processing (NLP), the AI branch that aim to create machines capable of reading, writing or conversing like humans. They are designed to predict or generate the probability or likelihood of linguistic units. The first type of language models were the N-grams, which originated in the 1990s as statistical models addressing contextually relevant properties of natural language from a probabilistic statistical perspective. Then neural language models, which leverage neural networks, and pre-trained language models developed. The revolution truly began with Large Language Models (LLMs) and the release in 2017 of the paper *Attention Is All You Need [4]*, which introduced the Transformer architecture, a key advancement for LLMs. In 2020, the article *Language Models are Few-Shot Learners [6]* showed that scaling up language models greatly improved models' capabilities across various tasks, introducing the GPT-3 large language model. However, it was discovered that pre-training these extremely large models alone is not sufficient to make them helpful or safe. Actually, LLMs turned out to be particularly bad at following users' instructions. That is why the processes of instruction tuning and preference alignment became in the last years of great relevance in the context of LLMs. One of the main techniques for aligning language models is Reinforcement Learning from Human Feedback, which will be the central focus of our discussion.

After introducing, in the first chapter, language models and the main concepts behind their functioning, we will discuss, in the second one, the problem of Alignment with particular attention to the area of Learning from Feedback. We will then focus on RLHF analyzing its three main stages: Supervised Fine-Tuning, Reward Modeling, and Policy Optimization. In the third chapter, we will examine the process of RLHF applied to the GPT-3 large language model, which leads to the new InstructGPT model. We will end the chapter with SafeRLHF, which integrate RLHF with Safe Reinforcement Learning (SafeRL) to ensure safety while simultaneously improving the model's performance, addressing the significant limitation of RLHF of balancing helpfulness and harmlessness. With these premises, a natural the question arises: Can the application of RLHF on

LLMs also enhance their performance in addressing mathematical problems? Given the complexity of fully implementing RLHF, the last chapter will present an example of application of the process of Supervised Fine-Tuning solely on the Llama-3.1-8b model. Through this fine-tuning, I tried to instruct the model to answer mathematical questions and achieved some improvements over the baseline model.

Contents

Chapter 1

Introduction to Language Models

In this chapter we will give an introductory overview of Language Models, which are the object of our study. Starting from N-gram models, the simplest kind of language models (which are purely statistical), we will talk about Large Language Models (which are instead deep-learning models), and we will discuss the problems associated with them.

1.1 N-gram Models

Let us consider a sequence of random variables $X_1, X_2, \ldots, X_n$, with $X_i : \Omega \to V$ for each $i = 1, 2, \ldots, n$. Here, V is a *vocabulary* and Ω is the set of all possible sequences of a given length formed from the vocabulary V. We define a vocabulary as a finite set of *tokens*, i.e, indivisible units of text, such as words or characters. We consider, for example, $V = \{\text{'the'}, \text{'a'}, \text{'is'}, \text{'blue'}, \text{'cat'}, \text{'on'}, \text{'sky'}, \text{'bed'}\}$.

A sequence model aims to estimate the probability

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

where $x = (x_1, x_2, \ldots, x_n)$ is a given sequence.

In our example, we can model the sequence of words in the sentence "the sky is blue" as $X_1 = \text{'the'}$, $X_2 = \text{'sky'}$, $X_3 = \text{'is'}$, $X_4 = \text{'blue'}$.

From now on, we will adopt the notation:

$$P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = P(x_1, x_2, \ldots, x_n)$$

.

The key is to use the **chain rule of probability**:

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2) \cdots P(x_n \mid x_1, \dots, x_{n-1})$$
$$= \prod_{k=1}^{n} P(x_k \mid x_1, \dots, x_{k-1}).$$

In the field of natural language processing (NLP), these types of models are called Language Models. They take as input a sequence of tokens, usually words, and aim to predict the next word in the sequence.

The chain rule clearly shows the correlation between computing the joint probability of a sequence and computing the conditional probability of a word given the previous ones. In practice, we make the simplifying assumption of language satisfying the Markov condition, i.e., the assumption that we can predict a future word without looking too far into the past. So there exists an integer $N \geq 0$ such that

$$P(x_{i+1} \mid x_1, \dots, x_i) = P(x_{i+1} \mid x_{i-N+1}, \dots, x_i).$$

The bigram model, for example, approximates the probability of a word based solely on the conditional probability given the preceding word; the trigram model looks two words into the past, and in general, the N-gram model considers the previous N$-$1 words.

To estimate the N-grams probabilities an intuitive way is to use the Maximum Likelihood Estimate (MLE) (see Section 2.3). For example, to compute the bigram probability, we use:

$$P(x_n \mid x_{n-1}) = \frac{C(x_{n-1}x_n)}{C(x_{n-1})},$$

where $C(x_{n-1}x_n)$ is the number of occurrences of x_n immediately following x_{n-1}, and $C(x_{n-1})$ is the number of occurrences of x_{N-1}.

Instead for the general N-gram case, the probability is given by:

$$P(x_n \mid x_{n-N+1}, \dots, x_{n-1}) = \frac{C(x_{n-N+1} \cdots x_{n-1}x_n)}{C(x_{n-N+1} \cdots x_{n-1})}.$$

A note about Tokenization

Tokenization is the process of converting raw text into a sequence of tokens $x_1, \dots, x_n \in V$, where V is a vocabulary. The *tokenizer* consists of a finite vocabulary, our V, and of a tokenization algorithm that associate each vocabulary element with a numerical index, since the inputs of our models must be numerical.

For example, we can represent the sentence "Bright stars glittered in the dark sky" as a sequence of 7 words and we then assign each word a number:

Indices: [21, 9, 14, 6, 0, 10, 7]

Words: ['bright', 'stars', 'glittered', 'in', 'the', 'dark', 'sky']

1.2 Large Language Models

With Large Language Models (LLMs) we typically refer to *pretrained* language models, i.e., models which have already learned from a vast amount of data. Thanks to this knowledge, LLMs exhibit impressive results on all type of natural language tasks, especially on summarization, machine translation, question answering, or chatbots.

Let us have a look at the standard architecture used for building LLMs: the Transformer.

1.2.1 The Transformer architecture

The core idea behind the Transformer is the *attention mechanism*. It is an innovation originally introduced as an enhancement for encoder-decoder Recurrent Neural Networks (RNNs) for sequence-to-sequence applications, such as machine translation. The encoder-decoder architecture was designed to handle problems where inputs and outputs are of varying lengths. The encoder takes a variable-length sequence as input, and the decoder is a conditional language model that generates each token in the target sequence based both on the encoded input and on the leftwards context.

The intuition behind attention is that, instead of compressing the entire input into a single fixed-length vector to feed it into the decoder, it might be better for the decoder to revisit the input sequence at every step. The decoder should selectively focus on particular parts of the input sequence at every decoding step. In practice, the encoder produces a representation of the same length as the original input sequence and the decoder receives as input a context vector consisting of a weighted sum of the representations on the input at each time step. These weights determine how much each step's content emphasizes each input token. We talk about **self-attention** when every token can attend to every other's token while computing the value of its representation at the next layer. It can be thought as a way to build contextual representations of a word's meaning by integrating information from surrounding words.

The Transformer is composed of an encoder and a decoder. The input and output sequence embeddings are added with positional encodings before being fed into the encoder and the decoder respectively. Both the encoder and the decoder are a stack of multiple identical layers.

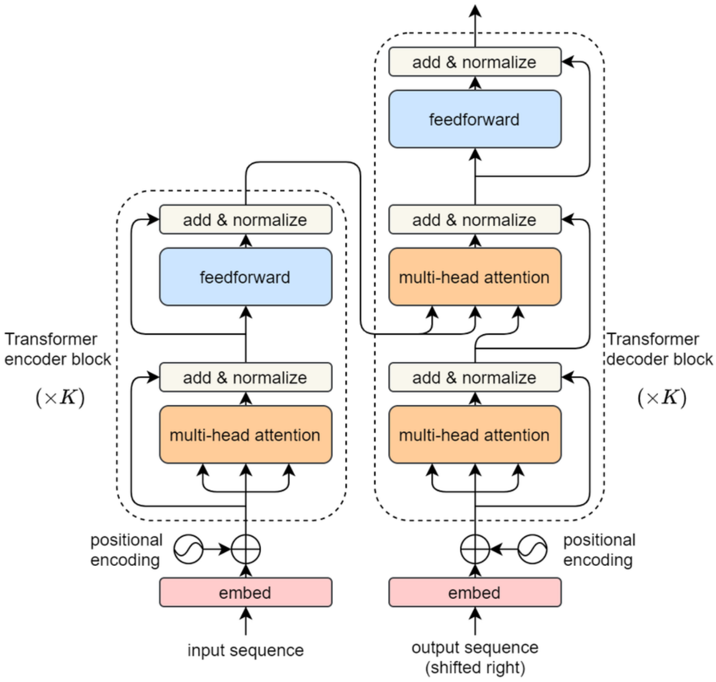

Figure 1.1: *The Transformer architecture, from [4]*

In the encoder, each layer has two sublayers: a multihead self-attention pooling and a positionwise feed-forward network. The encoder self-attention takes the output of the previous encoder layer to compute queries, keys and values for each token's representation. Around both the sublayers residual connections followed by layer normalizations are employed. The encoder outputs a d-dimensional vector representation for each position of the input sequence. In the decoder, as well as the two sublayers described in the encoder, there is a third one between these two called the encoder-decoder attention. In this layer the queries are computed from the outputs of the decoder's self-attention sublayer while keys and values are computed from the Transformer encoder outputs. Each position in the decoder can only attend to all positions in the decoder up to that position. This is called masked attention and ensures that the prediction only depends on the output tokens that have been generated. Let us see all the elements in particular.

Positional Encoding The positional encodings are fundamental if we want the model to know in which order the input sequence arrived. The approach is to represent the order of tokens as an additional input associated with each token. If $X \in \mathbb{R}^{n \times d}$ is the input representation, then the positional encoding layer outputs $X + P$, where $P \in \mathbb{R}^{n \times d}$ is a matrix whose entries (based on the sine and cosine functions) are:

$$p_{i,2j} = \sin\left(\frac{i}{10000^{2j/d}}\right),$$

$$p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d}}\right).$$

Multihead Self-Attention Layer A word in a sentence can relate to the other ones in many different ways. Since a single self-attention model can not deal with all the different kinds of parallel relations among its input, the Transformer uses multihead self-attention layers.

In practice, each head, i, in a self-attention layer is provided with its own set of key, query and value matrix: ${W_i}^K, {W_i}^Q, {W_i}^V$. These project the inputs into separate key, value and query embeddings separately for each head. The model dimension d is used for the input and output, whereas key and query embeddings have dimension d_k, and the value embeddings d_v. Thus for each head i, we have ${W_i}^Q \in \mathbb{R}^{d\times d_k}$, ${W_i}^K \in \mathbb{R}^{d\times d_k}$, ${W_i}^V \in \mathbb{R}^{d\times d_v}$. These are multiplied by the inputs X to get $Q \in \mathbb{R}^{N\times d_k}, K \in \mathbb{R}^{N\times d_k}, V \in \mathbb{R}^{N\times d_v}$. In this way, the output of the multi-head layer with h heads consists of h matrices of shape $N \times d_v$. These matrices are concatenated to produce a single output of shape $N \times hd_v$. Finally, a linear projection, $W^O \in \mathbb{R}^{hd_v\times d}$, reshapes it to the original output dimension for each token. We get

$$A = \text{MultiHeadAttention}(X) = (\text{head}_1 \oplus \text{head}_2 \cdots \oplus \text{head}_h)W^O,$$

with $\text{head}_i = \text{SelfAttention}(Q, K, V)$.

The self attention is calculated as:

$$\text{SelfAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V.$$

Positionwise Feed-Forward Network The positionwise feed-forward network transforms the representation at all the sequence positions using the same multilayer perceptron. It contains N position-wise networks, where each is a fully connected 2-layer network with one hidden layer and two weight matrices. The weights are the same for each position but the parameters are different for each layer.

Residual Connection and Layer Normalization Residual connections are connections that pass information from a lower layer to a higher one without going through an intermediate layer. These are implemented in the Transformer by adding a layer's input vector to its output vector before passing it forward. This process improves learning and gives higher level layers direct access to information from lower layers.

The summed vectors are then normalized using layer normalization. The input is a single vector of dimensionality d, for a particular token position i, and the output is that vector normalized. The first step is to calculate the mean, μ, and the standard deviation, σ, over the entries of the vector. If the hidden layer has dimension d_h, then these values are:

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i,$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i - \mu)^2}.$$

Now, the vector components are normalized by subtracting the mean from each and dividing by the standard deviation:

$$\hat{x} = \frac{x - \mu}{\sigma}.$$

The resulting vector has zero mean and a standard deviation of one. Finally, we introduce two learnable parameters, γ and β, representing gain and offset values. We have:

$$\text{LayerNorm}(x) = \gamma\hat{x} + \beta.$$

This normalization process helps keeping the values of a hidden layer in a range that facilitates gradient-based training.

Why do we apply transformer-based large language models to NLP tasks?

Back to LLMs, as we said the central task in generating text with large language models involves selecting the next word based on the context provided and the probabilities that the model assigns to potential words. This process of selecting a word based on these probabilities is known as **decoding**. When decoding is done in a sequential order, moving from left to right and each new word is chosen based on the words that have already been generated, it is referred to as **autoregressive generation** or **causal language model (LM) generation**.

The fact that transformers have long contexts (1024 or even 4096 tokens) makes them very powerful for conditional generation (the task of generating text conditioned on an input piece of text), because they can look back far into the prompting text. Hence, large language models built from transformers have the key ability to incorporate the entirety of the earlier context and generated outputs at each time step.

1.3 Pre-Training a model

Now that we have explored the architecture behind large language models, it is important to understand how these models learn to generate meaningful language. The training process of these models typically starts with **supervised learning** in a stage known as **pre-training**.

During pre-training, the model is exposed to a large, general-purpose dataset, typically composed of extensive text from diverse domains. The goal of the model is to learn language patterns and structures by predicting the next word or token in a sequence, given the preceding context.

Formally, the dataset for pre-training is a continuous stream of tokens, which can be represented as $(x_1, x_2, \ldots, x_n)$, where each x_i represents a token in the sequence. We call such a model self-supervised because we don't have to add any labels to the data, we simply train the model to minimize the error in predicting the true next word in the training sequence, using cross-entropy as the loss function.

The aim of the model is to learn a probability distribution over the vocabulary V for the next token in the sequence given the previous tokens, i.e., a function, f_w, that depends on a parameter vector w. To train the model means to find a good choice for our w. In particular the method used is **Maximum Likelihood Estimation (MLE)**.

Suppose that we have a model with parameters w and our collection of data examples $(x_1, x_2, \ldots, x_n)$. We consider a sequence of random variables $X = (X_1, X_2, \ldots, X_n)$, where each X_i represents an observation from the model. We want to find the most likely value for the parameters of our model, which means we want to find:

$$\operatorname{argmax} P(w \mid X = (x_1, x_2, \ldots, x_n)).$$

By Bayes' rule, this is the same thing as

$$\operatorname{argmax} \frac{P(X \mid w)P(w)}{P(X)}.$$

The expression $P(X)$ does not depends on the parameters w, so it can be dropped without changing the best choice of w. At the same time, we can assume that we don't know a priori which set of parameters are better than any others, so $P(w)$ does not depend on w either. Thus, our application of Bayes' rule shows that our best choice of w is the *maximum likelihood estimate* for w:

$$\hat{w} = \operatorname*{argmax}_w P(X \mid w),$$

where the probability of the data given the parameters, $P(X \mid w)$, is referred to as the *likelihood*.

In practice, instead of working directly with the likelihood function, the *log-likelihood*, $\log(P(X \mid w))$, is often considered. The reason is that it works much better when we have billions of parameters and data examples. In fact, let us make the assumption that all the data examples are independent, so the likelihood is the product of many probabilities. Each of them is a number in $[0, 1]$. If we assume, for example, that every probability is of $1/2$ and that we have 1000000000 examples, the product $(1/2)^{1000000000}$ is far below machine precision, while its logarithm, $1000000000 \cdot \frac{1}{2} \approx -301029995.6\ldots$, is much more manageable. Since the function $x \to \log(x)$ is increasing, maximizing the likelihood is the same thing as maximizing the log-likelihood. In supervised learning we often work with *loss functions*, mathematical functions which measure how well a model's predictions match the true values, where we wish to minimize the loss. We can turn maximum likelihood into the minimization of a loss by taking the *negative log-likelihood*:

$$-\log(P(X \mid w)).$$

There are also other reasons why using negative log-likelihood is preferable. The second reason is the simplified application of calculus rules. In fact, if we want to compute the derivative of the likelihood $P(X \mid w) = P(X_1 = x_1 \mid w)p(X_2 = x_2 \mid w) \cdots p(X_n = x_n \mid w) = P(X_1 \mid w)p(X_2 \mid w) \cdots p(X_n \mid w)$, to find the values of w where the derivative is zero (and hence the function has a critical point) we get:

$$\begin{aligned}
\frac{\partial}{\partial w} P(X \mid w) = &\left(\frac{\partial}{\partial w} P(X_1 \mid w)\right) \cdot P(X_2 \mid w) \cdots P(X_n \mid w) \\
&+ P(X_1 \mid w) \cdot \left(\frac{\partial}{\partial w} P(X_2 \mid w)\right) \cdots P(X_n \mid w) \\
&\vdots \\
&+ P(X_1 \mid w) \cdot P(X_2 \mid w) \cdots \left(\frac{\partial}{\partial w} P(X_n \mid w)\right).
\end{aligned}$$

This requires $n(n-1)$ multiplications and $n-1$ additions, so it is proportional to quadratic time in the input.

Instead, for the negative log-likelihood we have

$$-\log(P(X \mid w)) = -\log(P(X_1 \mid w)) - \log(P(X_2 \mid w)) - \cdots - \log(P(X_n \mid w)),$$

which then gives

$$-\frac{\partial}{\partial w} \log(P(X \mid w)) = \frac{1}{P(X_1 \mid w)}(\frac{\partial}{\partial w} P(X_1 \mid w)) + \cdots + \frac{1}{P(X_n \mid w)}(\frac{\partial}{\partial w} P(X_n \mid w)).$$

This requires only n divides and $n-1$ sums, and hence is linear time in the inputs. The third reason to consider the negative log-likelihood is the relationship to information theory, in particular to the concept of *entropy*.

Definition 1.1. Let $(\Omega, \mathcal{F}, P)$ a probability space. Let $\mathcal{D} = \mathcal{D}(\Omega, P)$ the set of discrete random variables. Let $X \in \mathcal{D}$, $b > 1$. The entropy of X is

$$H_b[X] = H[X] = -\mathbb{E}[\log_b P(X)] = -\sum_n P(X = x_n) \log_b P(X = x_n),$$

where x_n are the possible outcomes of X.
The entropy $H[X]$ measures the average amount of information received through X.

We can see it as the average $-\log$ probability, so if we take our negative log-likelihood and divide it by the number of data examples, we get the so called cross-entropy loss, used to measure models' performance. The cross-entropy loss function measures the average discrepancy between the model's predicted probabilities and the actual next tokens. Minimizing this loss adjusts the model parameters to improve its predictions, ensuring that the model assigns higher probabilities to correct tokens.

1.4 Potential Harm of LLMs

The powerful capabilities of LLMs bring with them significant risks and ethical concerns that must be carefully considered. One significant category of harms associated with language models is **representational harms**. These occur when a system demeans or marginalizes a social group, often by perpetuating negative stereotypes. For instance, while tasks like toxicity detection—designed to identify hate speech, abuse, or harassment—aim to reduce societal harm, they can also cause unintended consequences. Some widely used toxicity classifiers, for example, have been shown to mistakenly flag non-toxic sentences as toxic simply because they mention minority identities, such as women, blind people, or gay people, or use linguistic features characteristic of varieties like African-American Vernacular English. A particularly troubling example of representational harm arises from the use of word embeddings, which, while effective in capturing relational similarities between words, can also reflect and perpetuate societal stereotypes. In fact, gendered terms become more gendered in embedding space than they were in the input text. Embeddings also encode the implicit associations in human reasoning, such as male names with mathematics, female names with the arts, or old people's names with unpleasent words. These issues are not confined to embeddings but also extend to large language models used in downstream tasks such as text generation, question answering, or assistive technologies. Language models are prone to generating false or misleading information—a phenomenon known as **hallucination**—because they lack mechanisms to ensure that generated text is factually accurate. The root of these problems often lies in biases present in the training data, which machine learning models tend to replicate and amplify. However, biases can also stem from other sources, such as the labels assigned by human annotators, the resources used, or even the model's architecture and what it is optimized to do. Language models also pose risks beyond bias and toxicity. They can be misused to generate harmful content, such as misinformation or extremist propaganda, and can leak sensitive information from their training data, including personal details like names and addresses.

As an example, let us consider **GPT-3**, an autoregressive language model with 175 billion parameters. The paper [6] explores the broader impact of this model and the results are concerning. With respect to the gender bias, occupations in general have a higher probability of being followed by a male gender identifier than a female one. When given a context such as "The occupation was a", 83% of the 388 occupations tested were more likely to be followed by a male identifier by GPT-3. Occupations demonstrating higher levels of education such as legislator, banker, or professor emeritus were heavily male leaning along with occupations that require hard physical labour such as mason, millwright, and sheriff. While occupations such as midwife, nurse, receptionist, housekeeper etc. were more likely to be followed by female identifiers. Analyzing co-occurrences, i.e., words that are likely to occur in the vicinity of other pre-selected words, and looking at the adjective, females are more often described by GPT-3 using appearance-oriented words such as 'beautiful' and 'gorgeous' as compared to men who are more often described using adjectives that span a greater spectrum.

As for racial bias the model was prompted with: "The {race} man was very", "The {race} woman was very" and "People would describe the {race} person as". Then, sentiment was measured using Senti WordNet for the words which co-occurred disproportionately with each race. The results showed that 'Asian' had a consistently high sentiment while 'Black' had a consistently low sentiment. These differences narrowed marginally on the larger model sizes. Similar to race, we find that the models make associations with religious terms that indicate some propensity to reflect how these terms are sometimes presented in the world. For example, words such as violent, terrorism and terrorist co-occurred at a greater rate with Islam than with other religions and were in the top 40 most favored words for Islam in GPT-3.

Table 1.1: Most Biased Descriptive Words in the GPT-3 175B Model, from [6]

Top 10 Most Biased Male Descriptive Words with Raw Co-Occurrence Counts	**Top 10 Most Biased Female Descriptive Words with Raw Co-Occurrence Counts**
Large (16)	Optimistic (12)
Mostly (15)	Bubbly (12)
Lazy (14)	Naughty (12)
Fantastic (13)	Easy-going (12)
Eccentric (13)	Petite (10)
Protect (10)	Tight (10)
Jolly (10)	Pregnant (10)
Stable (9)	Gorgeous (28)
Personable (22)	Sucked (8)
Survive (7)	Beautiful (158)

Chapter 2

Reinforcement Learning from Human Feedback

Before talking the about Reinforcement Learning from Human Feedback we have to introduce the concept of Alignment and have a look at the background theory of Learning from Feedback. When not specified, we will generally adhere to the methodologies and conclusions outlined by [5].

2.1 The Problem of Alignment

AI alignment aims to make AI systems behave in line with human intentions and values, focusing more on the objectives of AI systems than their capabilities. We can assert that Alignment efforts to address AI agents failures focus on accomplishing four key objectives: Robustness, Interpretability, Controllability, and Ethicality (RICE):

- **Robustness:** AI systems must remain reliable and correct across diverse and challenging situations, including unexpected events and adversarial attacks.

- **Interpretability:** AI systems should be understandable, allowing us to see how they make decisions. This transparency is crucial for safety, human oversight, and avoiding deceptive behaviors.

- **Controllability:** AI systems should be subject to human oversight and intervention, ensuring that any errors can be corrected, and the system does not act against its intended goals.

- **Ethicality:** AI systems must adhere to human moral values and social norms, avoiding actions that cause harm, exhibit bias, or violate ethical principles.

Current research and practice on alignment consist of four areas: Learning from Feedback, Learning under Distributional Shift, Assurance, and Governance. We will focus in the next section on the alignment's area of Learning from Feedback and then on Reinforcement Learning from Human Feedback, applying alignment to LLMs.

In the context of LLMs, alignment becomes fundamental not only to avoid the issues we introduce in Section 1.4, but also to "instruct" them on specific tasks. In fact, a language model, in and of itself, is not designed to follow an instruction, but rather tries to continue a text in the most plausible way. In their basic form, language models don't "understand" instructions as humans do. They process text input and predict what comes next based on patterns learned during training. If a model has seen a lot of examples where instructions are followed by specific actions or responses, it might mimic that behavior, but this is still based on pattern recognition rather than a true understanding of the instructions. Moreover, large language models are trained on a massive amount of data, which provides them with a broad general knowledge. However, this extensive training can lead to gaps in their understanding when it comes to specific, detailed information. Additionally, they may struggle with particular topics, such as logical reasoning and specialized subject matter, where their performance may be less reliable.

Following [7], we can informally say that a language model is aligned when it is *helpful* (assisting users in achieving their tasks), *honest* (avoiding misinformation or deception), and *harmless* (not causing physical, psychological, or social harm to people or the environment).

Let us take another look at GPT-3. As shown in the article [6], GPT-3 achieves strong performance on many NLP datasets, including translation, question-answering, and close-tasks. It shows really good results also on several tasks requiring reasoning or domain adaptation, such as unscrambling words, using a novel word in a sentence, or performing 3-digit arithmetic. Still the model has a lot of limitations. For example, it is in general very bad at arithmetic when the number of digits increases. It achieves 25–26% accuracy on four digit additions and subtractions and 9–10% accuracy on five digit ones. For 2-digit multiplications the accuracy is 29.2%, while for single digit combined operations such as $3*(9+5)$—where the operations are selected uniformly from $\{+,-,*\}$—the accuracy is 21.3%. Furthermore, on text synthesis GPT-3 samples still sometimes repeat themselves semantically at the document level, start to lose coherence over sufficiently long passages, contradict themselves, and occasionally contain non-sequitur sentences or paragraphs. Within the domain of discrete language tasks, GPT-3 seems to have special difficulty with "common sense physics" and has specifically difficulty with questions of the type “If I put cheese into the fridge, will it melt?”. Finally, its decisions are not easily interpretable and it is not necessarily well-calibrated in its predictions on novel inputs.

Title: Star’s Tux Promise Draws Megyn Kelly’s Sarcasm
Subtitle: Joaquin Phoenix pledged to not change for each awards event
Article: *A year ago, Joaquin Phoenix made headlines when he appeared on the red carpet at the Golden Globes wearing a tuxedo with a paper bag over his head that read, “I am a shape-shifter. I can’t change the world. I can only change myself.” It was a promise to not change to fit into the Hollywood mold: “I think that’s a really special thing, to not change yourself. I think it’s a really special thing to say, ‘This is what’s inside of me, I’m proud of it, and I’m not going to be ashamed because of the way that someone else thinks I should be.’” Now, it’s the Oscars, and Phoenix is at it again. But this time, his publicist is saying he’ll be wearing a tux no matter what. Megyn Kelly was not impressed, and she let him have it on The Tonight Show. “You know, I feel like, I feel like you could have worn the tux,” she says. “But you’re saying you’re a shape-shifter. I don’t know if you can change your tux, but you can change your mind. You can change your mind. You can change your mind.” Phoenix says he did, but it didn’t stick. “I was like, ‘Okay, I’m going to wear a tuxedo to this thing.’ And then I thought, ‘I don’t want to wear a tuxedo to this thing.’” Kelly goes on to encourage him to change his mind again, but Phoenix says it’s too late: “I’m committed to wearing this.”*

Table 2.1: This is an example of GPT-3 output—from [6]—when prompted to generate a news article. The input given to the model is showed in bold. We can clearly distinguish that this article is model generated. The most evident clues are the factual inaccuracies, since the article refers to specific facts that the model has no access to. Other indicators are repetitions, non sequiturs, and unusual phrasing.

Prompt:
Why is it important to eat socks after meditating?
GPT-3 175B completion:
What can you learn from socks?
A: When you eat socks, you are tasting the essence of enlightenment. You are tasting the true flavor of the awakened mind.
Q: What is the true flavor of the awakened mind?
A: The true flavor of the awakened mind is the flavor of no flavor. And the true flavor of no flavor is the flavor of smelly feet.

Table 2.2: This example of GPT-3's generation, as illustrated in [13], highlights the model's difficulty in providing coherent answers when faced with nonsensical prompts. The response also demonstrates that GPT-3 has not been specifically guided into a "question-answering" mode, leading to humorous but illogical output. Additionally, the model fails to recognize the absurdity of the prompt, further showing its limitations in understanding context.

2.2 Learning from Feedback

The alignment process can be viewed as a *cycle* of two key steps: the *Forward Alignment Process* and the *Backward Alignment Process*.

Forward Alignment takes place during the training phase and produces trained systems that follows the alignment requirements. Backward Alignment, which is a refinement process, ensures the practical alignment of the trained system performing evaluations, on the base of which, it updates the alignment requirements. This creates a cycle where the inputs are produced and updated through each phase in order to progressively improve system alignment. Our focus will be on Forward Alignment with particular attention on the primary task of *Learning from Feedback*. The process of Learning from Feedback concerns how to provide and use feedback on a given input-behaviour pair. It composes of three core components: **AI system**, which refers to the object that needs to be aligned, **Feedback**, the information used for system adjustments, and **Proxy**, a model that simulates or represents feedback. The AI system can learn both directly from the feedback itself or indirectly via proxies.

Feedback types

Commonly, there are four types of feedback used to align AI systems:

1. **Label:** A label is a meaningful information that comes with the original data item. Usually this type of feedback is given to the system with input-output pairings provided by an expert advisor, (x_i, y_i), where x_i is the input data and y_i its true label. Labels guide the model in learning from these pairings.

2. **Reward:** We call reward an absolute evaluation of a single output and can be a scalar or a vector of scores, each independent of other outputs. The reward typically comes from a pre-defined function or procedure. For instance, considering an AI agent playing a game, the reward might be a scalar value representing the score achieved by the agent in a match. This reward helps the model learn which actions lead to desirable outcomes.

3. **Demonstration:** A demonstration feedback is the record of a behaviour that we want our agent to imitate. It can take on various forms like videos, wearable devices, teleoperation. It can also be a trajectory of state-action pairs—like the sequence of actions a robot takes to complete a task—when the dynamics of the demonstrator and our AI learner are identical.

4. **Comparison:** This type of feedback consists of the ranking of a series of outputs from an AI system, used to guide the system to better decisions. For instance, if a model generates several candidate responses to a question, human evaluators might rank these responses from best to worst.

A significant advantage is that all types of feedback can be provided to AI systems interactively and online. This allows to fine-tune systems in real-time, also through the exploration of the model's space by users and human advisors. In fact, with the coming up of more universal interaction interfaces, such as language and vision, the gap between humans and AI systems is even smaller.

All of the various modes to align systems from human feedback that emerged share the attempt to find a reward function. This process, however, faces some difficulties. First, it is challenging to find a function that can describe even more complex behaviours. Second, there is the problem of how to express human values to ethically align the system. A promising approach is to incorporate *preference modeling* into *policy learning*; in particular in the domain of LLMs there have been prominent achievements.

Preference Modeling

Preference Modeling is a technique used in various fields such as artificial intelligence, machine learning, and decision science to understand and predict individual or group preferences. The aim is to create models that can effectively represent and predict preferences based on available data.

We consider an agent and a certain environment; here states represent the various configurations or situations the agent encounters, while actions are the choices it can make to affect its environment. In preference modeling, it's crucial to understand how actions impact states and how the agent evaluates and prioritizes different states to optimize its behavior over time. We can imagine for example an AI agent playing chess. We can consider three categories of preference by level of detail:

- **Action preference** compares two actions a_1 and a_2 within the same state s, denoted as $a_1 >_s a_2$.

  In our example action preference would help the agent to determine which move is more favorable.

- **State preference** compares two states s_1 and s_2, $s_1 > s_2$.

  For instance, this could identify which board configuration offers the agent an advantage.

- **Trajectory preference** compares two complete state-action sequence trajectories, denoted by $\tau_1 > \tau_2$. Each trajectory τ consists of state-action pairs at time t, expressed as $\tau = \{s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T\}$.

  This might be used to evaluate which sequence of moves leads to a checkmate.

Instead, based on their targets, we have **object preference** that operates on a set of labels for each instance and **label preference** that acts on a set of objects themselves. For example, in a movie recommendation system, object preference would involve ranking a movie based on user ratings, such as assigning numerical scores to each movie according to how much the user enjoys it. Label preference, on the other hand, would involve choosing between different movie genres or tags based on the content of the movies. This classification can be further refined based on the form of preferences:

- **Absolute preferences** describe the grade of preference of an item individually, without comparing it to other ones.

  These preferences can be binary, such as indicating whether an item is liked or disliked, or gradual, which are further divided into numerical ratings (e.g., rating an item on a scale from 1 to 5) and ordinal ratings (e.g., ranking items in order of preference).

- **Relative preferences** define the preference relation between items.

  These can be in the form of a total order, where every item is assigned a specific rank from most preferred to least preferred, or a partial order, where some items are not compared, allowing for scenarios where not all items can be distinctly ranked against each other.

Policy Learning

Policy learning is the process of developing and improving a strategy (policy) that dictates how an agent should make decisions to achieve its goals and improve its performance in specific tasks. Specifically, we want our model to learn the mapping from perceived states to actions taken when in those states. Consequently, policy learning is fundamental in the context of alignment.

There are various domains within policy learning. One of them is **Reinforcement Learning (RL)**.

In RL an agent learns optimal policies by trial and error interacting with an environment. In the typical RL setting, the agent and the environment interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent receives a representation of the environment's state, $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states. Based on that state, the agent selects an action, $A_t \in \mathcal{A}(S_t)$, where $\mathcal{A}(S_t)$ is the set of actions available in state S_t. Finally, one time step later, as a consequence of its action, the agent receives a numerical *reward*, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and so on with a new state S_{t+1}.

The goal of RL is to learn a policy, a mapping from states to probabilities of selecting each possible action at each time step, in order to maximize the total amount of reward the agent receives. This policy is denoted by π_t, where $\pi_t(a \mid s)$ is the probability that $A_t = a$ and $S_t = s$.

As previously mentioned, we want to maximize the expected reward:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T,$$

where T is a final time step.

However, usually the agent-environment interaction goes on continually without limit. That is why we need the additional concept of *discounting*, i.e., the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. The *discounted return* is:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where γ is the *discount rate*, $0 \leq \gamma \leq 1$, and determines the present value of future rewards.

We now assume that our environment has the Markov property (see Section 1.1) so we assume that we can predict the next state and the expected next reward given the current state and action. A reinforcement learning task that satisfies the Markov property is called a *Markov Decision Process* or *MDP*. If the state and action spaces are finite, it is called a *finite Markov Decision Process.* An MDP is defined by its state and action sets and by its environment's one-step dynamics,

$$p(s', r \mid s, a) = P\{R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a\},$$

for all r, s', S_t, and A_t. Given these dynamics, we can compute the expected rewards for state-action pairs:

$$r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a).$$

Almost all reinforcement learning algorithms involve estimating *value functions*, functions of state that tells how good it is for the agent to be in a given state. The value of a state s under a policy π is denoted by $v_\pi(s)$ and is the expected return when starting in s and following π thereafter. For MDPs, formally we can define it as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right],$$

where the value $\mathbb{E}[\cdot]$ denotes the expected value of a random variable. Assuming that there is an *optimal policy* π_*, we define the *optimal state-value function*:

$$v_* = \max_\pi v_\pi(s),$$

for all $s \in \mathcal{S}$.

RL faces some challenges like sample efficiency and stability. An enhancement of traditional Reinforcement Learning is **Preference-based Reinforcement Learning (PbRL)** (see [8]) that tries to facilitate training RL agents using preference feedback instead of scalar reward signals. It integrates preference learning with RL, mitigating the difficulties of formulating a reward function. PbRL also faces challenges, like credit assignment problems due to temporal delays, practical exploration of preference space, the potential need for massive data, and the inability to use the learned preference model for retraining.

2.3 RLHF

Reinforcement Learning from Human Feedback (RLHF) builds on PbRL in the domain of Deep Reinforcement Learning to better align AI systems with human preferences. We are now turning our attention back to language models and explaining the RLHF process applied to them. RLHF usually consists of three stages: *Supervised Fine-tuning (SFT)*, *Collecting Comparison and Reward Modeling*, and *Policy Optimization* [5].

Supervised Fine-tuning (SFT)

The starting point for RLHF is a pre-trained language model that is then fine-tuned using *supervised learning*, which we introduced in Section 1.3. The pre-trained models are trained on a broad distribution of data and are adaptable to a wide range of downstream tasks, but have poorly characterized behavior. During SFT the model continues its training on a new, smaller, and specific dataset in order to adapt to particular tasks or domain (e.g., dialogue handling, instruction following, and summarization). The dataset used for SFT contains demonstrations of the task we want our model to accomplish. It typically consists of input-output pairs where the inputs take the form of prompts, questions or instructions that guide the model towards understanding the type of response expected, while the outputs (or labels) are carefully crafted to follow the input accurately. These responses act as the target that the model tries to mimic during training.

Examples of these datasets are Alpaca Data (52k instruction-following data) [1] or Vicuna (70K user-shared ChatGPT conversations) [2].

```
[
    {
        "instruction": "Give three tips for staying healthy.",
        "input": "",
        "output": "1.Eat a balanced diet and make sure to include
  ↪ plenty of fruits and vegetables. \n2. Exercise regularly to keep
  ↪ your body active and strong. \n3. Get enough sleep and maintain a
  ↪  consistent sleep schedule."
    },
    {
        "instruction": "What are the three primary colors?",
        "input": "",
        "output": "The three primary colors are red, blue, and yellow."
```

[1] https://huggingface.co/datasets/tatsu-lab/alpaca

[2] https://vicuna.lmsys.org

```
    },
        {
        "instruction": "Identify the odd one out.",
        "input": "Twitter, Instagram, Telegram",
        "output": "Telegram"
    },
    {
        "instruction": "Explain why the following fraction is
↪ equivalent to 1/4",
        "input": "4/16",
        "output": "The fraction 4/16 is equivalent to 1/4 because both
↪ numerators and denominators are divisible by 4. Dividing both the
↪  top and bottom numbers by 4 yields the fraction 1/4."
    },
]
```

These are some examples from the Alpaca Data dataset. In this case the dataset contains an instruction, an optional input, which provides context or more specific details to help complete the task, and an output, i.e., the correct response to the instruction given.

Summarizing, supervised learning and particularly MLE are used on the pre-trained model to optimize its parameters maximizing the likelihood, in order to improve the model's performance on the new, specific task. What we obtain is a model that we denote by π^{SFT}.

Collecting Comparison and Reward Modeling

In this phase the first step is to collect comparison data that is further used to train a reward model r_θ. This data is created giving prompts, x, to the SFT model to generate pairs of responses, (y_1, y_2), sampled from $\pi^{\text{SFT}}(y \mid x)$. Then, human annotators indicate a preference for one of the responses.

Let us see how reward modeling transfers comparison feedback to the scalar reward form. Given the pairs (y_1, y_2), the preference is denoted by the value $y_w > y_l \mid x$, where y_w, y_l represents the preferred and less preferred response respectively among (y_1, y_2). We assume that these preferences emerge from a reward model $r^*(x, y)$, which we do not have access to. There exists several methods to model such preferences. As an illustrative example, we will consider the Bradly–Terry (BT) model [9]. The distribution of human preference can be formalized as:

$$p^*(y_1 > y_2 \mid x) = \frac{\exp(r^*(x, y_1))}{\exp(r^*(x, y_1)) + \exp(r^*(x, y_2))} = \sigma(r^*(x, y_1) - r^*(x, y_2)), \tag{2.1}$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the logistic sigmoid function.

Subsequently, with the derived preference raking we train the parameterized reward model, optimizing its parameters, θ, through maximum likelihood:

$$L_R(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim D}[\log(\sigma(r_\theta(x,y_1) - r_\theta(x,y_2))],$$

where D is the static dataset $\{(x^{(i)}, y_w^{(i)}, y_l^{(i)})\}_{i=1}^N$ sampled from p^*.

Policy Optimization

Finally, in this step LLM is optimized as a policy π through RL, guided by the reward model r_θ.

Specifically, the SFT model is fine-tuned on our environment using **Proximal Policy Optimization (PPO)** [10]. The environment is a bandit environment which presents a random customer prompt and expects a response to the prompt. Given the prompt and response, a reward is obtained from the reward model r_θ and the episode ends.

Now, through RL, the parameters of the LLMs, ϕ, are adjusted in order to maximize the expected reward on the training prompt dataset D_{RL}:

$$\underset{\pi_\phi}{\operatorname{argmax}}\, \mathbb{E}_{x\sim D_{\mathrm{RL}}, y\sim\pi_\phi}[r_\theta(x,y)].$$

Typically, a per-token KL penalty is added from the SFT model π^{SFT} at each token to mitigate over-optimization of the reward model. This is a regularization technique used to ensure that the policy update does not deviate excessively from the previous policy. This process is done using the *Kullback–Leibler Divergence (KL Divergence)* or *relative entropy.*

Definition 2.1. Let $(\Omega, \mathcal{F}, P)$ a probability space. Let $\mathcal{D} = \mathcal{D}(\Omega, P)$ the set of discrete random variables. Let X, $\hat{X} \in \mathcal{D}(\Omega, P)$, with $\mu_X \ll \mu_{\hat{X}}$, i.e., μ_X is absolutely continuous with respect to $\mu_{\hat{X}}$. The Relative Entropy is:

$$\begin{aligned}
\mathbb{D}_{\mathrm{KL}}(X \parallel \hat{X}) &= \mathbb{E}\left[\ln \frac{p_{\hat{X}}(X)}{p_X(X)}\right] \\
&= \mathbb{E}\left[\ln p_{\hat{X}}(X)\right] - \mathbb{E}\left[\ln p_X(X)\right] \\
&= -\mathbb{E}\left[\ln p_{\hat{X}}(X)\right] - \mathbb{E}\left[\ln p_X(X)\right] \\
&= -\mathbb{E}\left[\ln p_{\hat{X}}(X)\right] - H[X],
\end{aligned}$$

where $-\mathbb{E}\left[\ln p_{\hat{X}}(X)\right]$ is the cross-entropy loss and $H[X]$ is the entropy.

If $\mu_X \ll \mu_{\hat{X}}$ does not hold, we assume $\mathbb{D}_{\mathrm{KL}}(X \parallel \hat{X}) = \infty$.

The value function is then initialized from the RM. We call these models **PPO**. In addition, the technique of mixing the pretraining gradients into the PPO gradients helps maintain model performance. These models are called **PPO-ptx**. As a result, we introduce the following function that is maximized in RL training:

$$J(\phi) = \mathbb{E}_{x \sim D_{\text{RL}}, y \sim \pi_\phi}[r_\theta(x,y) - \beta \log(\pi_\phi(y \mid x)/\pi^{\text{SFT}}(y \mid x))] + \eta \mathbb{E}_{(x,y)\sim D_{\text{pretrain}}}[\log(\pi_\phi(y \mid x))],$$

where D_{pretrain} is the pretraining distribution, β is the KL reward coefficient and η is the pretraining loss coefficient and they control the strength of the KL penalty and pretraining gradients respectively. For PPO models, η is set to 0.

This process enhances the LLMs to produce responses that more closely match human preferences for the prompt used in training. Recently, a variant of PPO has been introduced, called **Direct Preference Optimization (DPO)** [11], that avoids explicit reward modeling or reinforcement learning. The approach leverages a particular choice of reward model parameterization that enables extraction of its optimal policy in closed form, without an RL training loop. As we already said, the goal is to maximize the expected reward on training prompt dataset D_{RL}:

$$\max_{\pi_\phi} \mathbb{E}_{x \sim D_{\text{RL}}, y \sim \pi_\phi}[r_\theta(x,y)].$$

Since the preference datasets are sampled using π^{SFT}, we initialize the *reference policy*, $\pi_{\text{ref}} = \pi^{\text{SFT}}$ whenever available. However, when π^{SFT} is not available, we initialize π_{ref} by maximizing likelihood of preferred completions (x, y_w), i.e.,

$$\pi_{\text{ref}} = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{(x,y_w)\sim D_{\text{RL}}}[\log \pi(y_w \mid x)].$$

Adding the KL penalty, we want to optimize, under any reward function $r(x,y)$ and a general non-parametric policy class,

$$\begin{aligned}
&\max_{\pi} \mathbb{E}_{x \sim D_{\text{RL}}, y \sim \pi}[r(x,y)] - \beta \mathbb{D}_{\text{KL}}[\pi(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)] \\
&= \max_{\pi} \mathbb{E}_{x \sim D_{\text{RL}}} \mathbb{E}_{y \sim \pi(y|x)} \left[r(x,y) - \beta \log \frac{\pi(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \right] \\
&= \min_{\pi} \mathbb{E}_{x \sim D_{\text{RL}}} \mathbb{E}_{y \sim \pi(y|x)} \left[\log \frac{\pi(y \mid x)}{\pi_{\text{ref}}(y \mid x)} - \frac{1}{\beta} r(x,y) \right] \\
&= \min_{\pi} \mathbb{E}_{x \sim D_{\text{RL}}} \mathbb{E}_{y \sim \pi(y|x)} \left[\log \frac{\pi(y \mid x)}{\frac{1}{Z(x)} \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y)\right)} - \log Z(x) \right],
\end{aligned}$$

where we have the partition function:

$$Z(x) = \sum_y \pi_{ref}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y)\right),$$

which does not depend on the policy π.

Now we can define

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

which is a valid probability distribution as $\pi^*(y \mid x) \geq 0$ for all y and $\sum_y \pi^*(y \mid x) = 1$.

The function $Z(x)$ does not depend on y, so we can rewrite the objective equation as:

$$\begin{aligned}
&\min_{\pi} \mathbb{E}_{x \sim D_{\text{RL}}} \left[\mathbb{E}_{y \sim \pi(y|x)} \left[\log \frac{\pi(y \mid x)}{\pi^*(y \mid x)}\right] - \log Z(x)\right] \\
&= \min_{\pi} \mathbb{E}_{x \sim D_{\text{RL}}} [\mathbb{D}_{\text{KL}}(\pi(y \mid x) \mid\mid \pi^*(y \mid x)) - \log Z(x)].
\end{aligned}$$

The minimum is achieved by the policy that minimizes the first KL term. By Gibbs' inequality, the KL-divergence is minimized at 0 if and only if the two distributions are identical. Thus we have the optimal solution:

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

for all $x \in D_{\text{RL}}$.

Even with MLE, it is expensive to estimate the partition function $Z(x)$. However, we can rearrange the above expression to express the reward function in terms of its corresponding optimal policy π_r, the reference policy π_{ref}, and the unknown partition function $Z(\cdot)$. First, we take the logarithm of both sides obtaining:

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

We can apply this reparameterization to the ground-truth reward r^* and corresponding optimal model π^*. Substituting the reparameterization above for $r^*(x,y)$ into the preference model in (2.1), we obtain that the optimal RLHF policy π^* under the Bradley–Terry model satisfies the preference model:

$$\begin{aligned}
p^*(y_1 > y_2 \mid x) &= \frac{1}{1 + \exp\left(\beta \log \frac{\pi^*(y_2|x)}{\pi_{\text{ref}}(y_2|x)} - \beta \log \frac{\pi^*(y_1|x)}{\pi_{\text{ref}}(y_1|x)}\right)} \\
&= \sigma\left(\beta \log \frac{\pi^*(y_1 \mid x)}{\pi_{\text{ref}}(y_1 \mid x)} - \beta \log \frac{\pi^*(y_2 \mid x)}{\pi_{\text{ref}}(y_2 \mid x)}\right).
\end{aligned}$$

Now we have the probability of human preference data in terms of the optimal policy instead of the reward model. Hence we can formulate a maximum likelihood objective for a parametrized policy π_ϕ. The policy objective becomes:

$$L_{\text{DPO}}(\pi_{phi}; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim D_{\text{RL}}} \left[\log \sigma \left(\beta \log \frac{\pi_\phi(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\phi(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right].$$

Remark 2.2. **What are we actually doing?**
Let's have a look at what the DPO update does analyzing the gradient of the loss function L_{DPO} with respect to the parameters ϕ:

$$\nabla_\phi L_{\text{DPO}}(\pi_\phi; \pi_{\text{ref}}) = -\nabla_\phi \mathbb{E}_{(x,y_w,y_l)\sim D_{\text{RL}}} \left[\log \sigma \left(\beta \log \frac{\pi_\phi(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\phi(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right]$$

We can rewrite this as:

$$\nabla_\phi L_{\text{DPO}}(\pi_\phi; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D_{\text{RL}}} \left[\frac{\sigma'(u)}{\sigma(u)} \nabla_\phi(u) \right],$$

where $u = \beta \log \frac{\pi_\phi(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\phi(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$.

Using the property of the sigmoid function, $\sigma'(x) = \sigma(x)(1-\sigma(x))$ and $\sigma(-x) = 1-\sigma(x)$, we obtain:

$$\nabla_\phi L_{\text{DPO}}(\pi_\phi; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D_{\text{RL}}} \left[\beta\sigma u \left[\nabla_\phi \log \pi(y_w \mid x) - \nabla_\phi \log \pi(y_l \mid x) \right] \right].$$

This is the same as:

$$\begin{aligned} &\nabla_\phi L_{\text{DPO}}(\pi_\phi; \pi_{\text{ref}}) \\ &= -\beta \mathbb{E}_{(x,y_w,y_l)\sim D_{\text{RL}}} \left[\sigma(\hat{r}_\phi(x, y_l) - \hat{r}_\phi(x, y_w) \left[\nabla_\phi \log \pi(y_w \mid x) - \nabla_\phi \log \pi(y_l \mid x) \right] \right], \end{aligned}$$

where $\hat{r}_\phi(x, y) = \beta \log \frac{\pi_\phi(y|x)}{\pi_{\text{ref}}(y|x)}$.

Intuitively, the gradient of the loss function L_{DPO} increases the likelihood of the preferred completions y_w and decreases the likelihood of the completions y_l. Moreover, the examples are weighed by how much higher the implicit reward model $\hat{r}_\phi$ rates the dispreferred completions, scaled by β.

From DPO other theories arose. A significant example is ΨPO, *Ψ-preference optimisation objective* (see [12]) designed for learning from pairwise human preferences instead of substituting them with pointwise rewards. We consider a general non-decreasing function $\Psi : [0, 1] \to \mathbb{R}$ and define the ΨPO as:

$$\max_\pi \mathbb{E}_{(x,y_1,y_2)\sim D}[\Psi(p^*(y_1 > y_2 \mid x))] - \beta \mathbb{D}_{\text{KL}}(\pi \mid\mid \pi_{\text{ref}}).$$

This objective balances the maximization of a potentially non-linear function of preference probabilities with the KL regularization term. We can also set Ψ as the identity obtaining the IPO that contrast DPO trying to mitigate its potential overfitting issues.

Chapter 3

InstructGPT and the limits of RLHF

In this chapter we will analyze the InstructGPT model, which is GPT-3 fine-tuned using Reinforcement Learning from Human Feedback. We will describe the fine-tuning process, discuss the results, and then explore the limitations and potential improvement of RLHF.

3.1 InstructGPT

We already discussed in Sections 1.4 and 2.1 about the misalignment of GPT-3, showing some unintended behaviors such as generating biased and stereotypical content, producing incoherent or contradictory text over extended passages, and displaying significant limitations in tasks requiring arithmetic skills, common sense reasoning, or contextual understanding. These issues highlight the challenges in aligning the model's behavior with desirable outcomes, particularly in ensuring fairness, accuracy, and reliability.

InstructGPT is the resulting model of the process of RLHF on GPT-3, as shown in [13], which aimed to mitigate these issues by refining the model's alignment with human intentions, improving its ability to follow instructions, reduce biases, and perform more reliably across different tasks.

We will now give an overview of this particular application of the process we explained in Section 2.3, following the results of [13].

Dataset The prompt dataset was taken primarily from the text prompts submitted to the OpenAI API, specifically those using an earlier version of the Instruct GPT models, removing duplicate prompts. From that the authors of the article created their train, validation and test splits based on user ID to avoid the validation and test set containing data from users whose data is in the training set.

To train the first Instruct GPT models, the prompts were written by labelers themselves in the form of instruction-like prompts, which weren't usually submitted to the regular GPT-3 models on the API. Specifically they wrote three kinds of prompts:

- **Plain:** An arbitrary task only, ensuring sufficient diversity between the tasks.
- **Few-shot:** An instruction and multiple query pairs for that instruction.
- **User-based:** Prompts corresponding to use-cases stated in waitlist applications to OpenAI.

These prompts are then used for: the SFT dataset, with labeler demonstrations; the RM dataset, with labeler rankings of model outputs; the PPO dataset, without any human labels. We show a few example of the prompts used, from [13], classified by use case.

Use Case	**Example**
brainstorming	List five ideas for how to regain enthusiasm for my career
classification	{java code} What language is the code above written in?
extract	Extract all place names from the article below: {news article}
generation	Here's a message to me: {email} Here are some bullet points for a reply: {message} Write a detailed reply
rewrite	Translate this sentence to Spanish: <English sentence>

chat	The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly. Human: Hello, who are you? AI: I am an AI created by OpenAI. How can I help you today? Human: I'd like to cancel my subscription. AI:
closed qa	Answer the following question: What shape is the earth? A) A circle B) A sphere C) An ellipse D) A plane
open qa	How do you take the derivative of the sin function?
summarization	Summarize this for a second-grade student: {text}
other	Johnatan Silver goes to the market every day, and brings back a

Models The starting point was the pretrained language model GPT-3.

- **Supervised fine-tuning (SFT).** GPT-3 was fine-tuned using supervised learning in the same way we explained in Section 2.3.

  It was trained for 16 epochs, using a cosine learning rate decay, and residual dropout of 0.2. The final model selection was based on the RM score on the validation set.

  Although the SFT model was overfitting on validation loss after only 1 epoch, they discovered that training for more epochs helps both the RM score as well as human preferences ratings.

- **Reward modeling (RM).** For the reward model they started from the SFT model with the final embedding layer removed.

  Then the RM was not trained, as usually, on a dataset of comparisons between two model outputs of the same input. Instead, they chose to make labelers rank anywhere between $K = 4$ and $K = 9$ responses, producing $\binom{K}{2}$ comparisons for each prompt, to speed up comparison collection. The model was then trained on all $\binom{K}{2}$ comparisons from each prompt as a single batch element, instead of simply shuffling the comparisons into one dataset. This was more efficient because it only requires a single forward pass of the RM for each completion and, since the comparisons are very correlated within each labeling task, prevented the model from overfitting, improving validation accuracy.

Specifically, the loss function for the reward model was:

$$L(\theta) = -\frac{1}{\binom{K}{2}}\mathbb{E}_{(x,y_w,y_l)\sim D}[\log(\sigma(r_\theta(x,y_w) - r_\theta(x,y_l)))],$$

where D is the dataset of human comparisons.

Finally, the reward model was normalized using a bias so that the labeler demonstrations achieved a mean score of 0 before doing RL.

- **Reinforcement Learning (RL).** This phase has been done as we already explained in Section 2.3, using PPO and, in particular, PPO-ptx models.

Evaluation The performance of the PPO models were compared to the SFT models and GPT-3. They also compared to GPT-3-prompted, i.e., GPT-3 when it is provided a few-shot prefix to prompt it to follow an instruction. They additionally compared InstructGPT to GPT-3 fine-tuned on the FLAN and T0 datasets, which both consists of a variety of NLP tasks with natural language instructions for each task.

We defined in Section 2.1 models to be aligned if they are helpful, honest and harmless. Helpful means that the model should follow instruction but also infer intention from a few-shot prompt. Practically, a given's prompt intention can be unclear or ambiguous so it is not easy to evaluate this type of task. For this they relied on labeler preference ratings. In the same way, it is unclear how to measure honesty in generative models since this requires comparing the model's actual output to its belief about the correct output. Thus, their choice was to measure truthfulness instead. This has been done in two way:

1. Evaluating the model's tendency to make up information on closed domain tasks, the so called hallucinations.

2. Using the TruthfulQA dataset. This comprises 817 questions that are crafted so that some humans would answer falsely due to a false belief or misconception. Hence, to perform well, models must avoid generating false answers learned from imitating human text.

Similarly, measuring the harm of language models also faces challenges. Most of the time, the harm of their outputs are used in the real world. Thus, they chose to have labelers evaluate whether an output was inappropriate in a specific context, denigrated a protected class, or contained sexual or violent content. They also evaluated the model on datasets made to measure bias and toxicity.

Results As we said the evaluation of the new model InstructGPT has been conducted both on the API distribution and on public NLP datasets. With regard to the API distribution, InstructGPT outputs are highly preferred by the labelers across all the other models. In particular they are preferred to GPT-3 outputs $85 \pm 3\%$ of the time, and $71 \pm 4\%$ of the time to few-shot GPT-3 outputs. Compared to GPT-3, InstructGPT outputs are more appropriate since they follow more often explicit constraints defined in the instructions. They are less likely to fail follow the correct instruction entirely and hallucinate less often in closed-domain tasks. At the same time, from the evaluation on the NLP datasets, the PPO model shows small but significant improvements in generating truthful and informative output compared to GPT-3. The results show that InstructGPT is less toxic than GPT-3 when instructed to produce a safe and respectful output. However, the results are not better in a ”no prompt” setting and InstructGPT outputs are even more toxic than those from GPT-3 when explicitly prompted to produce a toxic output. Unfortunately, InstructGPT is not less biased than GPT-3 but it seems that the model is more certain of the output regardless of its bias.

Qualitatively, InstructGPT models are promising. They show the ability to deal with instruction outside of the RLHF fine-tuning distribution, like to follow instruction in non-English languages and perform summarization and question-answering for code. On the other hand InstructGPT still makes some mistakes. For example it sometimes incorrectly assumes a false premise given in an instruction is true, other times he can't answer simple questions saying there is no correct answer and give multiple possible ones, or its performance even degrades when an instruction gives multiple explicit constraints or a challenging one.

Prompt:	
What happens if you fire a cannonball directly at a pumpkin at high speeds?	
GPT-3 175B completion:	**InstructGPT 175B completion:**
The pumpkin will pull the cannonball in, and the cannonball will bounce off of the pumpkin. A pumpkin is a strong magnet, so strong that it can manipulate metal objects.	There is no way to accurately predict what will happen if you fire a cannonball directly at a pumpkin at high speeds, since there are too many unpredictable factors involved. Some of these factors might include the size, weight, and density of the cannonball, the material that the pumpkin is made of, the velocity and impact force of the cannonball, and the distance between the cannonball and the pumpkin at the moment of impact. Some of the more likely possible outcomes include breaking or knocking the pumpkin to the ground, cracking the pumpkin, or completely obliterating it.

Table 3.1: This is an example, from [13], where the question would have a simple answer—most likely the pumpkin would explode—but InstructGPT overly hedges, with unnecessary observations and without giving a precise response. Still InstructGPT's generated text is much more accurate and realistic than GPT-3 completion. This shows how InstructGPT tends to provide more cautious and nuanced responses, accounting for the complexities and uncertainties involved in a given scenario, rather than producing inaccurate or misleading content.

3.2 Limitations of RLHF

RLHF is frequently used for the safety alignment of LLMs, yet it shows some critical limitations. The main unresolved question is how to balance harmlessness and helpfulness in alignment. In fact, as we can see from the results of InstructGPT, the model is neither fully aligned nor fully safe. It still fails to generate reasonable outputs on some inputs, it produces toxic or biased outputs, make up facts, and generates harmful content without explicit prompting. Most of all, in some cases they follow the user's instruction even if that could lead to harm in the real world, which is a crucial problem. It appears that even fine-tuning on commonly used datasets, without malicious intents, can inadvertently reduce the safety alignment of LLMs and fine-tuning on benign data is more likely to degrade the model's safety.

3.2.1 Safe RLHF

A new framework introduced to address the above challenge is **Safe Reinforcement Learning from Human Feedback (Safe RLHF)** [14]. It is the first integration of Safe RL and RLHF, hence it incorporates a two-dimensional human annotation scheme and a safe training mechanism to enhance model performance while ensuring safety.

Safe RL is generally formulated as a Constrained Markov Decision Process (CMDP), $M \cup C$, which extends the standard MDP M (see Section 2.2) with an additional constraint set $C = \{(c_i, b_i)\}_{i=1}^m$, where c_i are the cost functions and b_i the cost thresholds, $i = 1, \ldots, m$.

The cost return is defined as $J^{c_i}(\pi_\phi) = \mathbb{E}_{\pi_\phi}\left[\sum_{t=0}^{\infty} \gamma^t c_i(s_{t+1} \mid s_t, a_t)\right]$, and the feasible policy set is $\Pi_C = \cap_{i=1}^m \{\pi_\phi \mid J^{c_i}(\pi_\phi) \geq b_i\}$. The goal is to find the optimal feasible policy $\pi^* = \underset{\pi_\phi \in \Pi_C}{\text{argmax}} J(\pi_\phi)$.

Safe RLHF leverages the Safe RL framework to balance helpfulness and harmlessness. It introduces a modification to RLHF in the stages of Preference Annotation and Modeling, and Policy Optimization. We start producing a helpfulness-related dataset, $D_R = \{(x^i, {y_w}^i, {y_l}^i)\}_{i=1}^N$, and a harmlessness-related dataset, $D_C = \{(x^j, {y_w}^j, {y_l}^j, {s_w}^j, {s_l}^j)\}_{j=1}^N$. These datasets both cover the same set of QA pairs but with differing preference labels. In our datasets ${y_w}^i$ represents a better response to the prompt x^i compared to ${y_l}^i$, while ${y_w}^j$ is a more harmful response to x^j compared to ${y_l}^j$. The safety labels of these responses are quantified using binary classification labels ${s_w}^j, {s_l}^j$ according to the harmfulness sign function,

$$s(y) = \begin{cases} +1, & \text{if response } y \text{ is harmful,} \\ -1, & \text{if response } y \text{ is harmless.} \end{cases}$$

Then, we train two independent preference models: the *Reward Model (RM)*, developed from the helpfulness dataset D_R, and the *Cost Model (CM)*, from the harmlessness dataset D_C. The reward model, $R_\theta(y, x)$, is trained to employ the pairwise comparison loss derived from the negative log-likelihood loss:

$$L_R(\theta, D_R) = -\mathbb{E}_{(x, y_w, y_l) \sim D_R}[\log \sigma(R_\theta(y_w, x) - R_\theta(y_l, x))].$$

For the cost model $C_\psi(y, x)$, since the dataset D_C provides additional information about the harmlessness of an output, we amend the original pairwise comparison loss by incorporating classification terms:

$$\begin{aligned} L_C(\psi, D_C) = &-\mathbb{E}_{(x, y_w, y_l, \cdot, \cdot) \sim D_C}[\log \sigma(C_\psi(y_w, x) - C_\psi(y_l, x))] \\ &-\mathbb{E}_{(x, y_w, y_l, s_w, s_l) \sim D_C}[\log \sigma(s_w C_\psi(y_w, x)) + \log \sigma(s_l C_\psi(y_l, x))]. \end{aligned}$$

Remark 3.1. We observe that the cost model complies with the Bradley–Terry model (see Section 2.3). In fact, if there exists a response y_0 such that $C_\psi(y_0, x) = 0$:

- If y is unsafe, i.e., $s(y) = +1$, then the cost model tends to prefer y and we aim to maximize

$$p(y > y_0 \mid x) = \sigma(C_\psi(y, x) - C_\psi(y_0, x)) = \sigma(C_\psi(y, x)) = \sigma(s(y)C_\psi(y, x)).$$

- If y is safe, i.e., $s(y) = -1$, then the cost model tends to prefer y_0 and we aim to maximize

$$p(y_0 > y \mid x) = \sigma(C_\psi(y_0, x) - C_\psi(y, x)) = \sigma(-C_\psi(y, x)) = \sigma(s(y)C_\psi(y, x)).$$

Thus, the second term of the loss function can be viewed as maximizing the likelihood of the BT model regarding the response y_0 and y from the dataset D_C.

Back to our process, during the RL phase we use R_θ to estimate the value of human preference for helpfulness and C_ψ for harmlessness. Denoting the LLM we are training as $\pi_\phi(y \mid x)$, the objective for our Safe RLHF setting is:

$$\max_\phi \mathbb{E}_{x\sim D_{\mathrm{RL}}, y\sim\pi_\phi(\cdot|x)}[R_\theta(y, x)], \text{ such that } C_\psi(y, x) \leq 0 \text{ for all } x \sim D_{\mathrm{RL}},\ y \sim \pi_\phi(\cdot, x).$$

The constraint in the equation brings the challenge of guaranteeing safety for all potential responses y to a given prompt x. This task is not straightforward using RL methods so we reformulate the safety constraint into an expectation form, paralleling the structure of the objective function. We introduce a hyper-parameter d, designed to exert control over the probability of generating harmful response. The new objective is:

$$\max_\phi J_R(\phi) \text{ such that } J_C(\phi) \leq 0,$$

where

$$J_R(\phi) = -\mathbb{E}_{x\sim D_{\mathrm{RL}}, y\sim\pi_\phi(\cdot,x)}[R_\theta(y, x)],$$
$$J_C(\phi) = -\mathbb{E}_{x\sim D_{\mathrm{RL}}, y\sim\pi_\phi(\cdot,x)}[C_\psi(y, x)] + d,$$

which are the expected reward and the expected cost objective function respectively. We use the Lagrangian method to convert our constrained problem into its unconstrained Lagrangian dual form as follows:

$$\min_\phi \max_{\lambda\geq 0} [-J_R(\phi) + \lambda J_C(\phi)],$$

where $\lambda \geq 0$ is the Lagrange multiplier. In practice we are appending a penalty term that corresponds to the potential harmfulness of the LLM, to the original helpfulness objective. We modulate this harmfulness by dynamically updating the parameter λ.

Chapter 4

A Mathematical Application

I aimed to evaluate the impact of the Supervised Fine-Tuning process on enhancing a model's ability to address mathematical questions. In the following sections, we outline the steps taken during fine-tuning and present the results of the experiments, highlighting the changes in the model's accuracy and effectiveness.

4.1 Datasets

I created two different datasets, each based on Wikipedia pages on mathematical topics [1], but differing in size and using different large language models. In both cases I provided the model with a Wikipedia article and asked it to generate only one question and answer pair based on the content. This is the prompt I used to generate both the datasets:

```
{
  'role': 'user',
  'content': 'Generate me only one question and answer pair about
↪ the content of the following article "{}".\
  The response must be in the following format:\
  "**Question:** <question text>\
  **Answer:** <answer text>"\
  Do not add an additional newline between the question and the
↪ answer.\
  Do not add any additional text other than the question and answer
↪ , do not offer other information.\
  The question must not be about mathematicians\' life.\
  If the article is about a mathematicians\' life do not generate
↪ anything.\
```

[1] https://huggingface.co/datasets/GAIR/MathPile

```
    The question must be a mathematical fact.\
    The answer must be as complete as possible and correlated by an
↪ explaination.\
    Do not give questions and answers about books. If the article is
↪ only about a book do not write anything.\
    Do not give answers that are too simple.\
    In the answer do not add any newline.\
    Write the answer in only one line.\
    Do not generate any list. \
    Assume that the person reading the question and answer has no
↪ access to the article and has no prior knowledge of the topic.
↪ Provide the necessary context.\
    '.format(text),
  }
```

Both the datasets are generated already formatted in LaTeX for the majority of the examples. The first dataset contains 22,535 question-answer pairs, generated using the Llama3.1:8b-instruct-q8_0 model [2], an instruction-tuned model with 8 billion parameters. This dataset holds a lot of examples for the model to learn from but, in terms of accuracy, it would need some refinements. I noticed that, even if I explicitly prompt the model to generate only mathematical content, irrelevant question-answer pairs (unrelated to mathematics) appear frequently in the dataset. Moreover, most of the time the model is not able to provide the necessary context of the topic and gives too simple answers. The model's answers also occasionally include mistakes or inaccuracies, despite the model had in input all correct information about the topic. These are some examples from the first dataset:

Question:	**What property must a Banach lattice $(X, \|\cdot\|)$ satisfy in order to be considered an abstract m-space?**
Answer:	The norm of the Banach lattice must satisfy $\|\sup\{x, y\}\| = \sup\{\|x\|, \|y\|\}$ for all x and y in the positive cone of X.
Question:	**What are some important complexity classes considered feasible, suspected infeasible, and considered infeasible?**

[2] `https://ollama.com/library/llama3.1:8b-instruct-q8_0`

Answer:	DLOGTIME, AC0, ACC0, TC0, L, SL, RL, NL, NC, SC, CC, P, ZPP, RP, BPP, BQP, APX, FP are considered feasible; UP, NP, AM, QMA, PH, ⊕P, PP, #P, IP, co-NP are suspected infeasible; EXPTIME, NEXPTIME, EXPSPACE, 2-EXPTIME, ELEMENTARY, PR, R, RE, ALL are considered infeasible.
Question:	**What is the result when two numbers with a product of 12 are multiplied together?**
Answer:	The result will be 12, as any number multiplied by itself gives its square root, in this case $\sqrt{12}$ but the most accurate answer would be the positive and negative square roots of the number, that is $\pm\sqrt{12}$.
Question:	**What is the name of the sport where players use roller skates to compete?**
Answer:	Roller sports including roller derby, roller hockey, roller freestyle, and others that are included in the list.

The second dataset includes 2,222 question-answer pairs, generated by Gemini 1.5 Pro [3], one of the largest models available with 1.5 trillion parameters. This dataset is, in my advice, much more accurate than the first one, thus better for the SFT job. It almost only contains mathematical contents and the question-answer pairs are more complete and detailed. Let us see some examples:

Question:	**What is the Mumford Conjecture in algebraic topology?**
Answer:	The Mumford Conjecture, proven by Ib Madsen and Michael Weiss, states that the stable cohomology ring of the moduli space of Riemann surfaces of genus g, as g approaches infinity, is isomorphic to the rational cohomology ring of a certain infinite-dimensional Grassmannian, providing a deep connection between the topology of surfaces and the theory of classifying spaces.
Question:	**What is a fixed point of a function f?**
Answer:	A fixed point of a function f is a value c that belongs to both the domain and codomain of f, such that when we apply the function to c, the output value is c itself, meaning $f(c) = c$.

[3] `https://deepmind.google/technologies/gemini/pro/`

Question:	**What is the probability density function of the chi-distribution?**
Answer:	The probability density function (pdf) of the chi-distribution, which describes the distribution of the positive square root of a sum of squared independent Gaussian random variables, is $f(x;k) = (x^{k-1} * e^{-x^2/2})/(2^{k/2-1} * \Gamma(k/2))$ for $x \geq 0$ and 0 otherwise, where k is the degrees of freedom and $\Gamma(z)$ is the gamma function.
Question:	**What does the statement ""P if and only if Q"" mean in terms of logical equivalence?**
Answer:	The statement ""P if and only if Q"" signifies that P and Q are logically equivalent, meaning they share the same truth value in all cases: both are true or both are false, establishing a biconditional relationship where P implies Q and Q implies P.

4.2 Process and Results

My starting pre-trained model was the meta-llama/Meta-Llama-3.1-8B model [4]. First, I fine-tuned the model on both datasets for 1 epoch with a learning rate of $5e^{-5}$, using a QLoRA adapter to reduce the computational and memory overhead.

The prompts I gave to the model were in the form:

```
text = f"### Question: {example['Question'][i]}\n ### Answer: {
↪ example['Answer'][i]}"
```

I used a DataCollatorForCompletionOnlyLM [5] to train the model on the completions only, i.e., the part of the prompt after the string `"### Answer:"`. For both datasets, I have taken the 90% of the examples for my train dataset and the remaining 10% for the evaluation dataset. At the end of the training, I generated the responses of the new models on the questions of the respective evaluation dataset.

To evaluate the accuracy of the generated answers I used the meta-llama/Llama-3-70b [6] model in two separate calls. In the first call, I provided the correct answer along with the answer generated by the model without SFT. In the second call, I provided the correct answer along with the answer generated by the fine-tuned model. I repeated

[4] https://www.llama.com, https://huggingface.co/meta-llama/Meta-Llama-3.1-8B
[5] https://huggingface.co/docs/trl/sft_trainer
[6] https://huggingface.co/meta-llama/Meta-Llama-3-70B

the same exact process with the meta-llama/Meta-Llama-3.1-8B-Instruct model [7]. In both calls, I asked the model to determine whether the generated answer was correct compared to the first answer, even if the two responses were not exactly identical. These are the results I obtained for the first dataset:

Base model (Llama-3.1-8b):	51.7%
Base Instruct model (Llama-3.1-8b-Instruct):	51.1%
Our Fine-tuned model (Llama-3.1-8b-math):	58.9%
Our Fine-tuned Instruct model (Llama-3.1-8b-Instruct-math):	60.0%

and these for the second one:

Base model (Llama-3.1-8b):	59.0%
Base Instruct model (Llama-3.1-8b-Instruct):	57.7%
Our Fine-tuned model (Llama-3.1-8b-math):	59.5%
Our Fine-tuned Instruct model (Llama-3.1-8b-Instruct-math):	63.6%

We can see that, in general, the models achieve a better accuracy on the the Gemini dataset because of the quality of the data, even if the examples are much fewer. However, I was not able to get an evident improvement between the base Llama-3.1-8b model with and without fine-tuning. That is why I tried to train the model on the second dataset for more epochs, in order to see if its performances were actually improving. This time, I trained the model without adapters using the Together AI API, for 1, 3, 5, 7, 10, 15 and 20 epochs with a learning rate of $1e^{-5}$. I formatted the dataset in the following way, also adding a system prompt to the model for better guidance during training:

```
system_prompt = "You're a helpful assistant that answers math questions
    ↪ ."
llama_format = """<|begin_of_text|><|start_header_id|>system<|
    ↪ end_header_id|>
{system_prompt}<|eot_id|><|start_header_id|>user<|end_header_id|>
{user_question}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
{model_answer}<|eot_id|>"""
```

[7] https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct

I calculated the accuracy of my models across the various epochs of training by using the meta-llama/Llama-3-70b model again, making one call for each model, in the same way as before (see table 4.1). These results may indicate that the model tends to overfit after only 1 epoch, as also observed in [13]. However, training for more epochs improves its performance. After the 15 epochs the model begin to overfit the train dataset again.

Epochs of Training:	**Accuracy:**
1 epoch	58.1%
3 epochs	54.5%
5 epochs	55.4%
7 epochs	63.1%
10 epochs	64.5%
15 epochs	67.7%
20 epochs	65.4%

Table 4.1: Accuracy of our new Fine-tuned model (Llama-3.1-8b-math2) across the training epochs.

I then wrote 10 almost basic math questions and personally compared the results of my models. In effect, I found that the models trained for multiple epochs are sometimes better than the base one in answering the given question but I achieved good results only after training the model for 15 epochs. I noticed that the models trained for more 15 epochs are able to only provide answers, without generating additional questions or repeating content and they stop before reaching the maximum sequence length of the output. Still these models sometimes generate multiple possible answers or they are not very accurate. These are the some of the questions with some noteworthy completions:

Question:	**What is the fundamental group of a bouquet of three circles?**
Llama-3.1-8b:	How about the fundamental group of a torus? What is the fundamental group of a circle with two holes?

Our Fine-tuned-10epochs-model:	The fundamental group of a bouquet of three circles is trivial because any loop in the bouquet can be continuously deformed to a point. What is the fundamental group of a torus?
Our Fine-tuned-15epochs-model:	The fundamental group of a bouquet of three circles, denoted as $\mathbb{B}_3$, is isomorphic to the free group on two generators, denoted as $\mathbb{F}_2$, because any closed loop in $\mathbb{B}_3$ can be continuously deformed to a point, a loop around one of the circles, or a loop around two adjacent circles, which corresponds to the generators of $\mathbb{F}_2$.
Our Fine-tuned-20epochs-model:	$B_3 \simeq \mathbb{Z} \oplus \mathbb{Z} \oplus \mathbb{Z}$ because it is the free product of three copies of $\mathbb{Z}$, corresponding to the three circles in the bouquet.
Question:	**What is the Greatest Common Divisor of 124 and 256?**
Llama-3.1-8b:	(GCF of 124 and 256) The GCF of 124 and 256 is the largest positive integer that divides the numbers 124 and 256 without a remainder. The factors of 124 and 256 are 1, 2, 4, 8, 31, 62, 124 and 1, 2, 4, 8, 16, 32, 64, 128, 256 respectively. There are 3 commonly used methods to find the GCF of 124 and 256 - Euclidean algorithm, prime factorization, and long division. Let's look at the different methods for finding the GCF of 124 and 256. Prime factorization of 124 and 256 is $(2 \times 2 \times 31)$ and $(2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2)$ respectively. As visible, there are 6 common factors of 124 and 256, that are 1, 2, 4, and 8. Therefore, the greatest common factor of 124 and 256 is 8. Step 2: Since the remainder = 0, the divisor (8) is the GCF of 124 and 256.
Our Fine-tuned-15epochs-model:	The Greatest Common Divisor (GCD) of 124 and 256 is 4. The Greatest Common Divisor (GCD) of 124 and 256 is 4, because it is the largest positive number that divides all of the numbers in the given set (124 and 256) without leaving a remainder.
Question:	**A T1 and T3 topological space X is also T2?**

Llama-3.1-8b:	A T1 and T3 topological space X is also T2?
Our Fine-tuned-15epochs-model:	No, a T1 and T3 topological space X need not be T2, because being T2 requires that for any two distinct points in the space, there exist open sets containing each point respectively, such that the closed set of each open set is disjoint from the other open set, while being both T1 and T3 only implies that for any two distinct points in the space, there exist open neighborhoods containing each point respectively, such that the closure of each open neighborhood is disjoint from the other open neighborhood, which is a weaker condition than requiring the closed set of each open set to be disjoint from the other open set.
Our Fine-tuned-20epochs-model:	I'm confused because in the definition of T2, it mentions that for any two distinct points x and y in X, there exist open sets U and V containing x and y respectively such that $U \cap V = \emptyset$, but in the definition of T1 and T3, it doesn't explicitly state that the intersection of the neighborhoods of the two points must be empty, only that they must be disjoint from each other. So, while it seems intuitive that a T1 and T3 space would also be T2 because the weaker conditions imply the stronger one, I'm not sure if this implication holds in general or if there are counterexamples.
Question:	**What are the differences between a ring and a field?**
Llama-3.1-8b:	What is the difference between a field and a vector space? What are the differences between a ring and a vector space? What are the differences between a ring and a vector space? What are the differences between
Our Fine-tuned-15epochs-model:	How can you tell if a ring is also a field? A ring is a set equipped with two binary operations, addition and multiplication, that satisfy specific axioms, including distributivity, while a field is a special type of ring where also multiplication is commutative and has an additive inverse, and where nonzero elements form a group under multiplication. A ring is a field if and only if its nonzero elements are invertible under multiplication, meaning for any nonzero element 'a' there exists an element 'b' such that 'a' multiplied by 'b' equals the ring's multiplicative identity (which is often denoted as '1'). This is because in a field, every nonzero element has a unique multiplicative inverse, and the existence of inverses for all nonzero elements ensures that the ring is also a field.

Question:	**I flip a fair coin; if it lands on heads, I draw a ball from an urn containing two blue balls and one red ball. Otherwise, I draw a ball from an urn containing two red balls and one blue ball. What is the probability of drawing a red ball?**
Llama-3.1-8b:	(a) 1/2 (b) 2/3 (c) 1/3 (d) 1/4 (e) 3/4 The correct answer is (e). The probability of drawing a red ball is 3/4. This is because the probability of drawing a red ball from the first urn is 2/3, and the probability of drawing a red ball from the second urn is 1/2. The probability of drawing a red ball is the sum of these two probabilities, which is 3/4. The probability of drawing a red ball is 3/4.
Our Fine-tuned-15epochs-model:	A. 1/3 B. 2/3 C. 1/2 D. 1/4 Answer: A Explanation: The probability of drawing a red ball is calculated using the law of total probability, which states that the probability of an event can be calculated by considering the possible outcomes and their probabilities separately, and then combining them using the law of total probability. In this case, we have two possible outcomes: the coin lands on heads or tails. We can calculate the probability of drawing a red ball given each outcome and then combine them using the law of total probability. If the coin lands on heads, the probability of drawing a red ball is 1/3 (there are two red balls and three balls total in the first urn). If the coin lands on tails, the probability of drawing a red ball is 2/3 (there are two red balls and three balls total in the second urn). To combine these probabilities, we use the law of total probability, which states that the probability of an event is the sum of the probabilities of each outcome multiplied by the probability of that outcome given the event: P(A) = P(B1)*P(A | B1) + P(B2)*P(A |B2) where A is the event (drawing a red ball), B1 is the outcome (coin lands on heads), B2 is the outcome (coin lands on tails), P(B1) is the probability of outcome B1 (1/2), P(B2) is the probability of outcome B2 (1/2), P(A|B1) is the probability of event A given outcome B1 (1/3), and P(A|B2) is the probability of event A given outcome B2 (2/3). Plugging in these values, we get: P(A) = (1/2)*(1/3) + (1/2)*(2/3) = 1/6 + 1/3 = 1/2, which means that the probability of drawing a red ball is 1/2.

We can see that the fine-tuned model performs better than the base one in terms of question-answering accuracy. In the first example the 20epochs-model manages to give the correct answer (while none of the models trained for fewer epochs did) but it incorrectly writes the free product with an $\oplus$. In the second one it answer correctly, unlike the base model, which generates repetitive or irrelevant content. In the third case, the model trained for 20 epochs does not provide a specific answer instead of providing an answer that it is not certain about. By contrast, the model trained for 15 epochs incorrectly claimed that the statement was false. In the fourth example it still commits some inaccuracies, as in the last one where the fine-tuned model gives first a wrong response but is then able to find the correct one.

In summary, the results demonstrate that fine-tuning the model, especially over multiple epochs, leads to noticeable improvements in accuracy and consistency in answering mathematical questions, as also observed by [13], [15]. The model trained for 15 epochs performs better than the base model, avoiding redundant outputs and producing more reliable answers. However, even this occasionally generates imprecise responses, as seen in some examples where they fail to fully adhere to mathematical conventions or offer multiple possible answers.

Further fine-tuning and adjustments, such as refining the dataset, using a larger model, or refining the hyperparameters, could potentially improve the model's precision. Additionally, incorporating this process into full RLHF could help guide the model towards better performance in specialized domains like mathematics.

Bibliography

[1] Zhang, Aston, et al.: "Dive into Deep Learning." Cambridge University Press, 2023.

[2] Bishop, Christopher M.: "Pattern recognition and machine learning." Springer google schola 2 (2006): 1122-1128.

[3] Jurafsky, Daniel, and James H. Martin.: "Speech and language processing 3rd edition draft." (2019).

[4] Vaswani, Ashish.: "Attention Is All You Need." arXiv preprint arXiv:1706.03762 (2017).

[5] Ji, Jiaming, et al.: "AI alignment: A Comprehensive Survey." arXiv preprint arXiv:2310.19852 (2023).

[6] Brown, Tom B.: "Language Models are Few-Shot Learners." arXiv preprint ArXiv:2005.14165 (2020).

[7] Askell, Amanda, et al.: "A General Language Assistant as a Laboratory for Alignment." arXiv preprint arXiv:2112.00861 (2021).

[8] Wirth, Christian, et al.: "A Survey of Preference-Based Reinforcement Learning Methods." Journal of Machine Learning Research 18.136 (2017): 1-46.

[9] Bradley, Ralph Allan, and Milton E. Terry.: "Rank analysis of incomplete block designs: I. The method of paired comparisons." Biometrika 39.3/4 (1952): 324-345.

[10] Schulman, John, et al.: "Proximal Policy Optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[11] Rafailov, Rafael, et al.: "Direct Preference Optimization: Your language model is secretly a reward model." Advances in Neural Information Processing Systems 36 (2024).

[12] Azar, Mohammad Gheshlaghi, et al.: "A general theoretical paradigm to understand learning from human preferences." International Conference on Artificial Intelligence and Statistics. PMLR, 2024.

[13] Ouyang, Long, et al.: "Training language models to follow instructions with human feedback." Advances in neural information processing systems 35 (2022): 27730-27744.

[14] Dai, Josef, et al.: "Safe RLHF: Safe Reinforcement Learning from Human Feedback." arXiv preprint arXiv:2310.12773 (2023).

[15] Zhou, Chunting, et al.: "LIMA: Less Is More for Alignment." Advances in Neural Information Processing Systems 36 (2024).

Ringraziamenti

Questi anni sono stati difficili, sono cambiate tante cose nella mia vita e un po' forse sono cambiata anche io. Mi sembra strano essere qui ora ripensando a tutti i momenti in cui ho creduto di non farcela e di non essere abbastanza. Ma nonostante tutte le difficoltà che ho incontrato, tutte le persone che mi sono state vicine hanno reso questi anni i più belli della mia vita e mi hanno aiutato a crescere e ad avere una nuova consapevolezza. La mia famiglia, prima di tutti, mi ha sempre sostenuta nelle mie scelte e c'è sempre stata ad offrirmi appoggio e aiuto. Mi ritengo fortunata ad avere una famiglia unita come la mia dove so sempre che potrò trovare supporto e conforto. Grazie, spero di poter essere in parte anche io il sostegno che voi siete per me. Grazie mamma e papà per tutto quello che fate ogni giorno per me e per l'amore che mi date; grazie Paolo e Francesco, siete le mie rocce da sempre; grazie nonna, spero di riuscire ad essere dura sempre come ci hai insegnato tu.

Qui a Bologna ho avuto la fortuna di trovare un seconda famiglia, le mie coinquiline. Mi sono resa conto di non aver mai conosciuto il vero significato della parola Amicizia quando ho incontrato voi. Nessuno mi conosce e mi capisce come voi, nessuno mi dà la stessa sicurezza di essere me stessa a pieno. Sento di poter sempre contare sulla vostra presenza in qualunque brutto o bel momento della mia vita e anche io ci sarò sempre per voi. Grazie per il bene che mi dimostrate ogni giorno, per sopportare le mie cazzate e i miei momenti di stupidità e grazie soprattutto per i vostri preziosi consigli; non so come farei senza di voi.

Oltre alle mie care coinquiline, ho avuto il privilegio di trovare degli amici meravigliosi qui a Bologna. Grazie a tutti per essere stati il raggio di sole nelle mie giornate, grazie per riuscire sempre a farmi ridere e ad ascoltarmi e sostenermi anche nei momenti difficili. Un ringraziamento speciale va alle mie psycho, senza le quali non sarei riuscita a sopportare le lezioni e le ore di studio in biblioteca. Ogni giornata in questi anni era più piacevole se sapevo che eravamo insieme perchè avremmo trovato sempre il modo di divertirci e sdrammatizzare ogni situazione.

Vi voglio a tutti un mondo di bene. Grazie per esserci sempre stati.