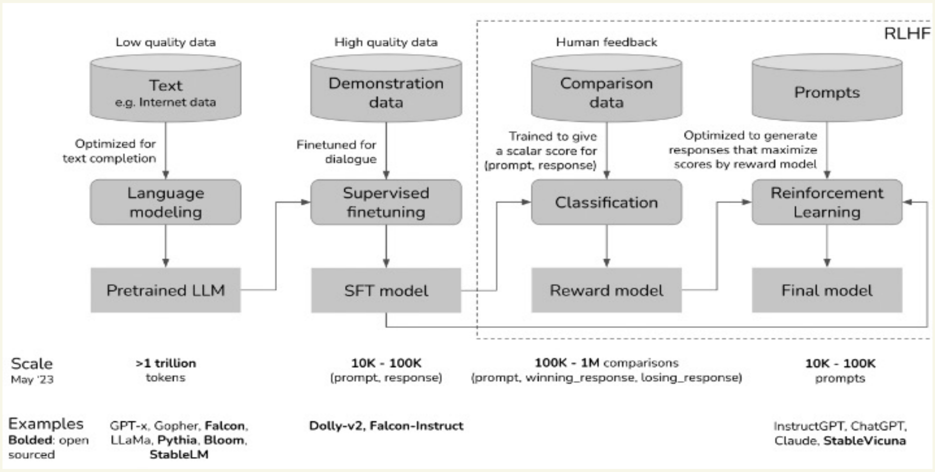
RLHF: Reinforcement Learning from Human Feedback

RLHF: Reinforcement Learning from Human Feedback


Low quality data
Text e.g. Internet data
Optimized for text completion
Language modeling
Pretrained LLM
High quality data
Demonstration data
Finetuned for dialogue
Supervised finetuning
SFT model
RLHF
Human feedback
Comparison data
Trained to give a scalar score for (prompt, response)
Classification
Reward model
Prompts
Optimized to generate responses that maximize scores by reward model
Reinforcement Learning
Final model
Scale May '23
>1 trillion tokens
10K - 100K (prompt, response)
100K - 1M comparisons (prompt, winning_response, losing_response)
10K - 100K prompts
Examples Bolded: open sourced
GPT-x, Gopher, Falcon, LLaMa, Pythia, Bloom, StableLM
Dolly-v2, Falcon-Instruct
InstructGPT, ChatGPT, Claude, StableVicuna

1. The pretrained model is an untamed monster because it was trained on indiscriminate data scraped from the Internet: think clickbait, misinformation, propaganda, conspiracy theories, or attacks against certain demographics.
2. This monster was then finetuned on higher quality data – think StackOverflow, Quora, or human annotations – which makes it somewhat socially acceptable.
3. Then the finetuned model was further polished using RLHF to make it customer-appropriate, e.g. giving it a smiley face.

so pertaining is most resource intensive phase(98%). Think of SFT and RLHF as unlocking the capabilities that pertained model has but hard for users to access via prompting alone.

Phase 1. Pertaining
The result of the pretraining phase is a large language model (LLM), often known as the pretrained model. Examples include GPT-x (OpenAI), Gopher (DeepMind), LLaMa (Meta).

a language model encode statistical info about language , for simplicity it tell us how likely something to appear given a context.

Data bottleneck for pertaining

language model like GPT-4 uses so much data that there's a realistic concern that we'll run out of Internet data in the next few years. these new LLMs might just be trained on data generated by existing LLMs.

Phase -2 Supervised fine-tuning (SFT) for dialogue

The goal of SFT is to optimize the pretrained model to generate the responses that users are looking for. During SFT, we show our language model examples of how to appropriately respond to prompts of different use cases. Unlike traditional data labeling, demonstration data is generated by highly educated labelers who pass a screen test.

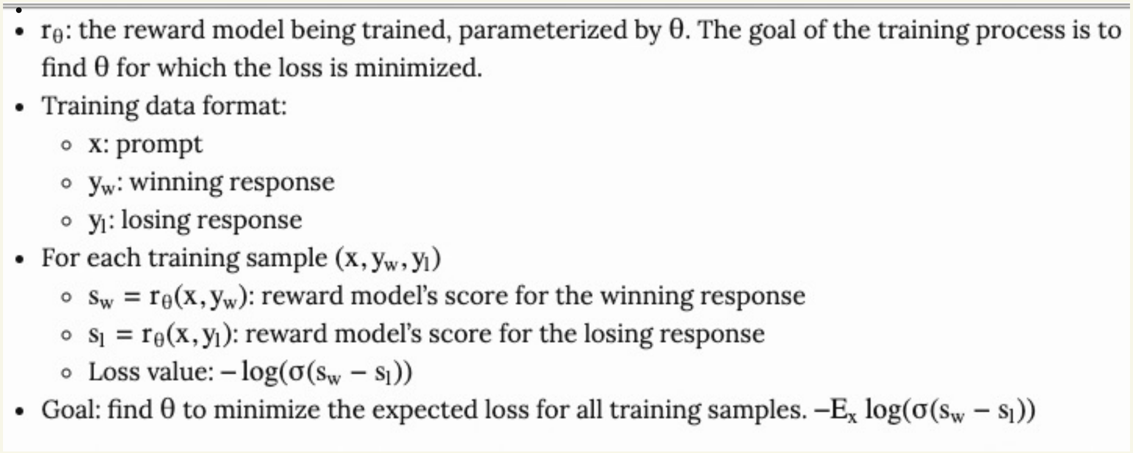
Prompt **-** Serendipity means the occurrence and development of events by chance in a happy or beneficial way. Use the word in a sentence.
Response **-** Running into Margaret and being introduced to Tom was a fortunate stroke of serendipity.

Phase 3 - RLHF

- Dialogues are flexible. Given a prompt, there are many reasonable responses, some are better than others. Demonstration data tells the model what responses are reasonable for a given context, but doesn't tell the model how good or how bad a response is.
- The idea: what if we have a scoring function that, if given a prompt and a response, outputs a score for how good that response is? Then we use this scoring function to further train our LLMs towards giving responses with high scores. That's exactly what RLHF does. RLHF consists of two parts:
    1. Train a reward model to act as a scoring function.
    2. Optimize LLM to generate responses for which the reward model will give high scores.

3.1. Reward model (RM)

- The RM's job is to output a score for a pair of (prompt, response). Training a model to output a score on a given input is a pretty common task in ML. You can simply frame it as a classification or a regression task.
- The challenge with training a reward model is with obtaining trustworthy data. Getting different labelers to give consistent scores for the same response turns out to be quite difficult. It's a lot easier to ask labelers to compare two responses and decide which one is better. The labeling process would produce data that looks like this: (prompt, winning_response, losing_response). This is called comparison data.
- For InstructGPT, the objective is to maximize the difference in score between the winning response and the losing response
- different ways to initialize an RM: e.g. training an RM from scratch or starting with the SFT model as the seed. Starting from the SFT model seems to give the best performance



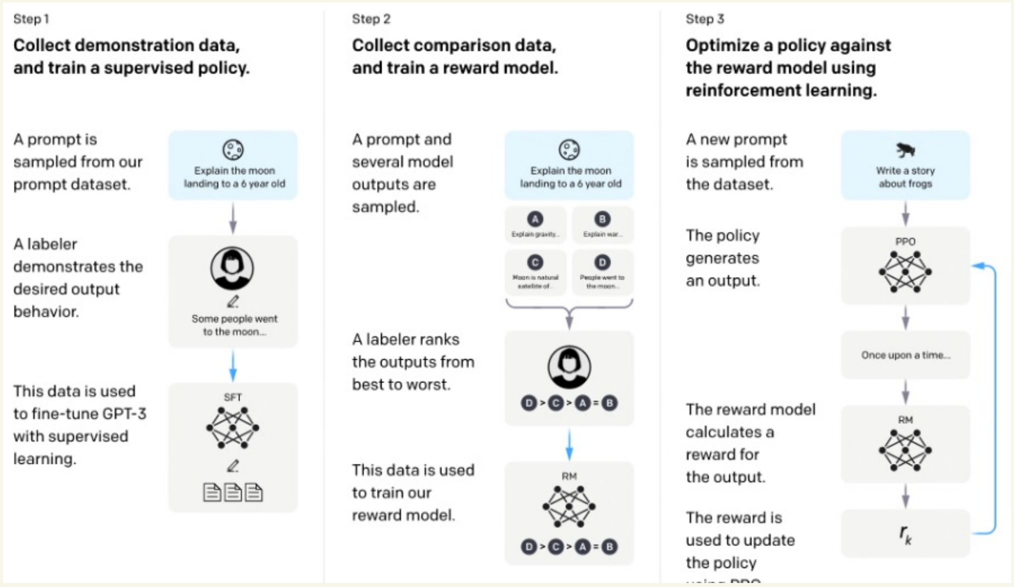
- r_θ: the reward model being trained, parameterized by θ. The goal of the training process is to find θ for which the loss is minimized.
- Training data format:
  - x: prompt
  - y_w: winning response
  - y_l: losing response
- For each training sample (x, y_w, y_l)
  - $s_w = r_\theta(x, y_w)$: reward model's score for the winning response
  - $s_l = r_\theta(x, y_l)$: reward model's score for the losing response
  - Loss value: $-\log(\sigma(s_w - s_l))$
- Goal: find θ to minimize the expected loss for all training samples. $-E_x \log(\sigma(s_w - s_l))$

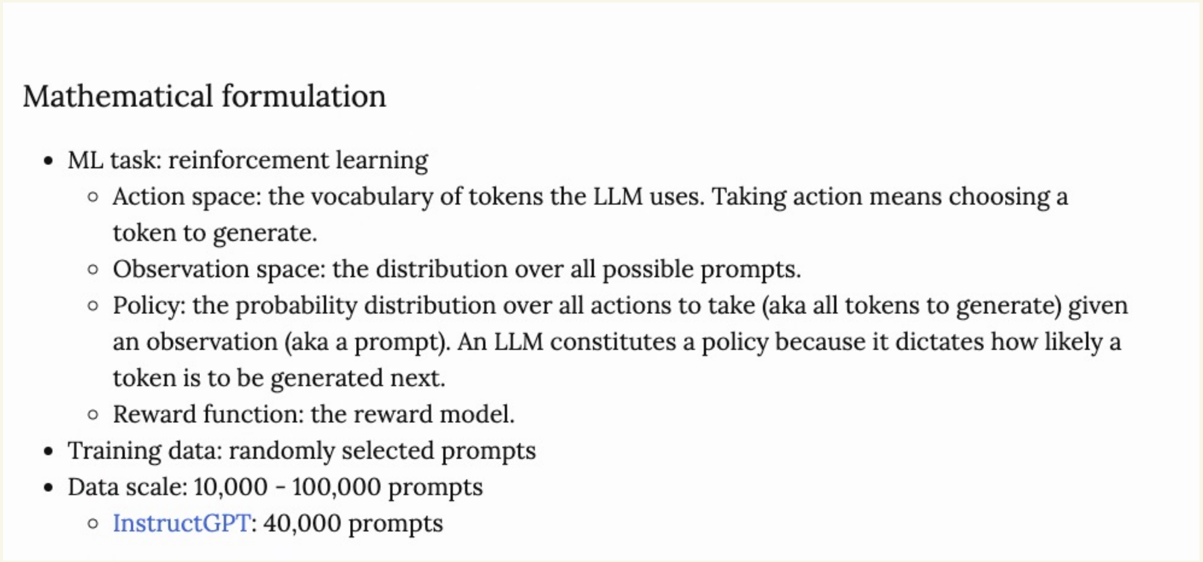
3.2. Finetuning using the reward model

we will further train the SFT model to generate output responses that will maximize the scores by the RM. Today, most people use Proximal Policy Optimization (PPO), a reinforcement learning algorithm

- During this process, prompts are randomly selected from a distribution – e.g. we might randomly select among customer prompts. Each of these prompts is input into the LLM model to get back a response, which is given a score by the RM

- OpenAI also found that it's necessary to add a constraint: the model resulting from this phase should not stray too far from the model resulting from the SFT phase. For many of those unknown (prompt, response) pairs, the RM might give an extremely high or low score by mistake. Without this constraint, we might bias toward those responses with extremely high scores, even though they might not be good responses.


Step 1
Collect demonstration data, and train a supervised policy.
A prompt is sampled from our prompt dataset.
Explain the moon landing to a 6 year old
A labeler demonstrates the desired output behavior.
Some people went to the moon...
This data is used to fine-tune GPT-3 with supervised learning.
SFT
Step 2
Collect comparison data, and train a reward model.
A prompt and several model outputs are sampled.
Explain the moon landing to a 6 year old
A Explain gravity...
B Explain war...
C Moon is natural satellite of...
D People went to the moon...
A labeler ranks the outputs from best to worst.
D > C > A = B
This data is used to train our reward model.
RM
D > C > A = B
Step 3
Optimize a policy against the reward model using reinforcement learning.
A new prompt is sampled from the dataset.
Write a story about frogs
The policy generates an output.
PPO
Once upon a time...
The reward model calculates a reward for the output.
RM
The reward is used to update the policy
r_k



Mathematical formulation

- ML task: reinforcement learning
  - Action space: the vocabulary of tokens the LLM uses. Taking action means choosing a token to generate.
  - Observation space: the distribution over all possible prompts.
  - Policy: the probability distribution over all actions to take (aka all tokens to generate) given an observation (aka a prompt). An LLM constitutes a policy because it dictates how likely a token is to be generated next.
  - Reward function: the reward model.
- Training data: randomly selected prompts
- Data scale: 10,000 - 100,000 prompts
  - InstructGPT: 40,000 prompts

RLHF and hallucination

- The first hypothesis,, is that LLMs hallucinate because they "lack the understanding of the cause and effect of their actions"
- The second hypothesis is that hallucination is caused by the mismatch between the LLM's internal knowledge and the labeler's internal knowledge. During SFT, LLMs are trained to mimic responses written by humans. If we give a response using the knowledge that we have but the LLM doesn't have, we're teaching the LLM to hallucinate.
- Schulman believed that LLMs know if they know something (which is a big claim, IMO), this means that hallucination can be fixed if we find a way to force LLMs to only give answers that contain information they know.

Why RL?

why RL (Reinforcement Learning) is better than learning from demonstrations (a.k.a supervised learning) for training language models. Shouldn't learning from demonstrations be sufficient?

- **pre-training** -> trying to predict next word , By doing so, it also acquires some sort of an internal representation of the language. After this process, the model is very capable of generating texts, and providing natural continuations to given text prefixes, but it is not good at "communicating". For example, when prompted with a question, it might either answer it OR it might generate a series of additional questions, OR it might say this is an important question that was raised in the context of
- **supervised-training -** also called learning from demonstrations, or "instruction tuning". we collect a bunch of human authored texts that have a form of a question or an instruction, followed by the desired output. By continuing to train the model on the same "predict the next token given the prefix" objective, but this time on this collection of instruction-output pairs, the model learns to respond to instructions by performing them.
- **Reinforcement Learning (RL)** In the reinforcement learning setup, we provide the model with the instructions, but not with their human authored answers. Instead, the model should generate its own answer. A scoring mechanism (for example, a human) reads the

generated answers, and tells the model if its good or not

Why RL? when RL is much harder Than supervised training.

1. The language model generates a sequence of tokens, and only gets a score at the end of the sequence. The signal is weak: which parts of the answer are good and which are bad?
2. we need a scoring mechanism to score the answers and, in the context of language-based tasks, it is hard to come up with an automatic scorer

Diversity argument -> instruction tuning in the context of language generation models is that we teach the learner to replicate the exact answer given by the demonstrator, while the reality of human language is that there are many different ways to convey the same message.

Theoretical argument -> supervised learning allow only positive feedback while RL also allow negative feedback.

The core argument ->

There are (at least) three modes of interaction with a language model: (a) **text-grounded**: we provide the model with a text and an instruction ("summarize this text", "based on this text, what is the population of Israel", "what are the chemical names mentioned in this text", "translate this text to spanish", etc), and expect the answer to be fully grounded in the provided text. (b) **knowledge-seeking**: we provide the model with a question or instruction, and expect a (truthful) answer based on the model's internal knowledge ("What are common causes of flu"). (c) **creative**: we provide the model with a question or instruction, and expect some creative output. ("Write a story about...")

The argument for RL is based on interaction type (b): knowledge-seeking queries in which we expect a truthful (or confident) answer, and the ability of the model to say "I don't know" or refuse to answer in situations in which it is uncertain.

The core issue is that **we want to encourage the model to answer based on its internal knowledge, but we don't know what this internal knowledge contains** n supervised training, we present the model with a question and its correct answer, and train the model to replicate the provided answer. There are two cases: (1) the model "knows" the answer. In this case, the supervised training correctly pushes it to associate the answer with the question, hopefully pushing it to perform similar steps to answer similar questions in the future. This is the desired behavior. (2) the model *does not know* the answer. In this case, the supervised training pushes the model to associate the answer with the question anyhow. Now, there are two options. It may push the model to memorize this particular question-answer pair. This is not harmful, but also not very effective, as our aim is for the model to generalize and learn to answer any question, not only the ones in the instructions training data. We want the model to generalize. But if we are succeed in training the model to generalize in these cases, then we essentially teaches the model to make stuff up! it actively encourages the model to "lie". This is bad.

In contrast to the supervised setting, the RL setting does not actively encourage the model to lie: even if the model does initally guess some answers correctly and learns a "making stuff up" behavior by mistake, in the long run it will get bad scores for made up answers (which are likely to be incorrect) and learn to adopt a policy that relies on its internal knowledge, or abstain.

teaching to abstain -> One way around this is to start with some supervised training learning to produce "I don't know" answers in some cases, and then continuing the process with RL

RL and Truthfulness

in SFT we do behaviour cloning. .SFT model hallucinate because it don’t whether it is allowed to do that .if you are telling model to replicate the instruct -answer in SFT and that is not in model knowledge then you are teaching model to hallucinate. Labeled know about something but not LLM

and if you tell model to say I don't know in SFT then model may withhold info that it know.like labeled don't know the answer but our model know but right answer is I don't know acc. To labeller.
does model know about its uncertainty -> yes

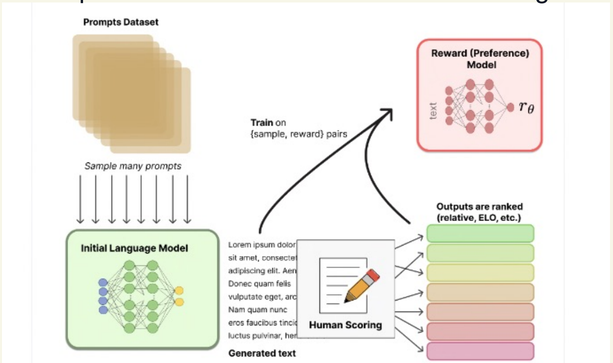
How to fix this with RL:
reward for (right, refuse, wrong).

Illustrating Reinforcement Learning from Human Feedback (RLHF)

There are many applications such as writing stories where you want creativity, pieces of informative text which should be truthful, or code snippets that we want to be executable. Writing a loss function to capture these attributes seems intractable and most language models are still trained with a simple next token prediction loss.

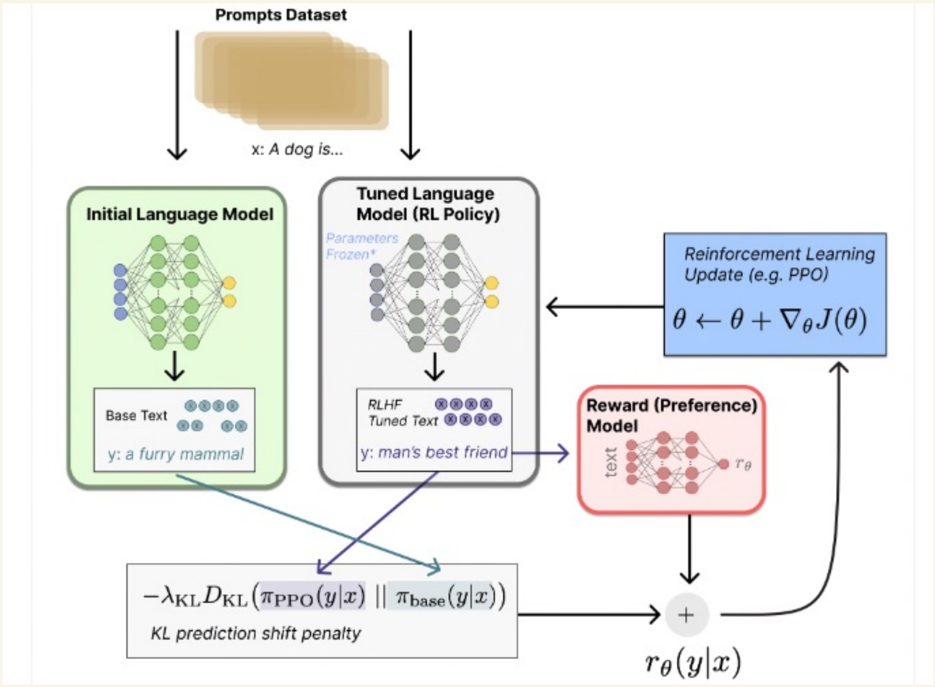
Wouldn't it be great if we use human feedback for generated text as a measure of performance or go even one step further and use that feedback as a loss to optimize the model?; use methods from reinforcement learning to directly optimize a language model with human feedback

Human annotators are used to rank the generated text outputs from the LM. One may initially think that humans should apply a scalar score directly to each piece of text in order to generate a reward model, but this is difficult to do in practice. The differing values of humans cause these scores to be uncalibrated and noisy. Instead, rankings are used to compare the outputs of multiple models and create a much better regularized dataset.


Prompts Dataset
Sample many prompts
Initial Language Model
Lorem ipsum dolor sit amet, consecte adipiscing elit. Aen Donec quam felis vulputate eget, arc Nam quam nunc eros faucibus tinci luctus pulvinar, her
Generated text
Human Scoring
Train on {sample, reward} pairs
Reward (Preference) Model
text
r_θ
Outputs are ranked (relative, ELO, etc.)

the **policy** is a language model that takes in a prompt and returns a sequence of text (or just probability distributions over text). The **action space** of this policy is all the tokens corresponding to the vocabulary of the language model (often on the order of 50k tokens) and the **observation space** is the distribution of possible input token sequences, which is also quite large given previous uses of RL (the dimension is approximately the size of vocabulary ^ length of the input token sequence). The **reward function** is a combination of the preference model and a constraint on policy shift.

Given a prompt, *x*, from the dataset, the text *y* is generated by the current iteration of the fine-tuned policy. Concatenated with the original prompt, that text is passed to the preference model, which returns a scalar notion of "preferability" r, . **In addition, per-token probability distributions from the RL policy are compared to the ones from the initial model to compute a penalty on the difference between them.** The KL divergence term penalizes the RL policy from moving substantially away from the initial pretrained model with each training batch, which can be useful to make sure the model outputs reasonably coherent text snippets. Without this penalty the optimization can start to generate text that is gibberish but fools the reward model to give a high reward. In practice, the KL divergence is approximated via sampling from both distributions .R = r(theta) - lambda* kl



Prompts Dataset
x: A dog is...
Initial Language Model
Tuned Language Model (RL Policy)
Parameters Frozen*
Reinforcement Learning Update (e.g. PPO)
$\theta \leftarrow \theta + \nabla_\theta J(\theta)$
Base Text
y: a furry mammal
RLHF Tuned Text
y: man's best friend
Reward (Preference) Model
text
r_θ
$-\lambda_{KL} D_{KL}(\pi_{PPO}(y|x) \,||\, \pi_{base}(y|x))$
KL prediction shift penalty
+
$r_\theta(y|x)$

When deploying a system using RLHF, gathering the human preference data is quite expensive due to the direct integration of other human workers outside the training loop. RLHF performance is only as good as the quality of its human annotations, which takes on two varieties: human-generated text, such as fine-tuning the initial LM in InstructGPT, and labels of human preferences between model outputs. The second challenge of data for RLHF is that human annotators can often disagree, adding a substantial potential variance to the training data without ground truth.

Direct Preference Optimization (DPO)

when models like ChatGPT first popped on the scene, the foremost approach involved training a reward model first and optimising the LLM policy. when models like ChatGPT first popped on the scene, the foremost approach involved training a reward model first and optimising the LLM policy.

DPO is a method introduced to achieve precise control over LLMs. Reinforcement Learning from Human Feedback (RLHF) was based on training a Reward Model and then using Proximal Policy Optimization (PPO) to align the language model's output with human preferences. This method, while effective, was complex and unstable
The DPO pipeline can be broken down into two main stages:

1. Supervised Fine-tuning (SFT): This is the initial step where the model is fine-tuned on the dataset(s) of interest.

2. Preference Learning: After SFT, the model undergoes preference learning using preference data, ideally from the same distribution as the SFT examples

DPO directly defines the preference loss as a function of the policy. This means that there's no need to train a reward model first.
During the fine-tuning phase, DPO uses the LLM as a reward model. It optimizes the policy using a binary cross-entropy objective, leveraging human preference data to determine which responses are preferred and which are not. By comparing the model's responses to the preferred ones, the policy is adjusted to enhance its performance.

Understanding Preference Data in NLP

Preference data is a curated set of options or alternatives related to a specific prompt. The goal is to rank these options from the most preferred to the least preferred.
The dataset should comprise three key entries: prompt, chosen, and rejected.

- ***prompt***: Contains the context inputs.
- ***chosen***: Houses the corresponding chosen responses.
- ***rejected***: Lists the corresponding negative (or rejected) responses.

The LLM is then fine-tuned to maximize the likelihood of generating preferred completions and minimize the likelihood of generating dispreferred ones.