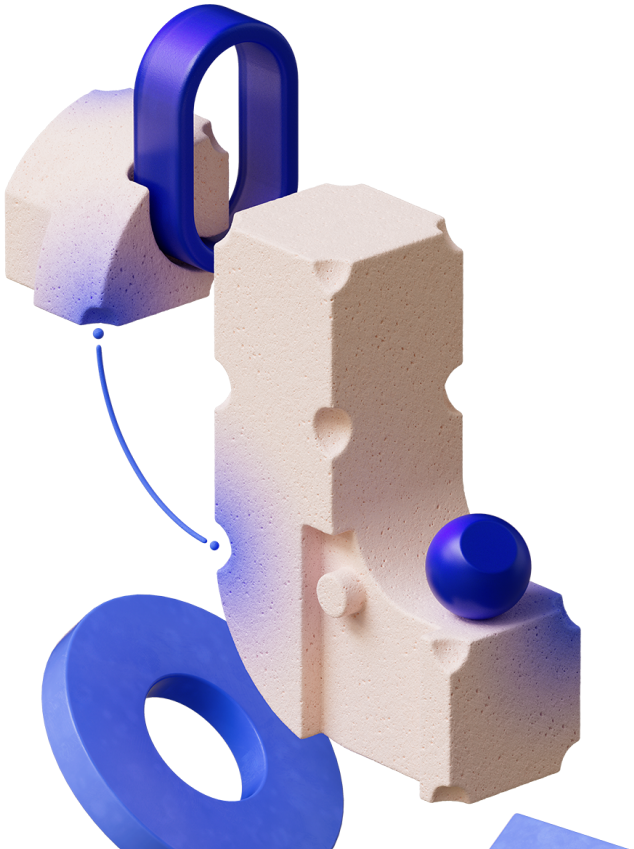
Google DeepMind



Reinforcement Learning with Human Feedback

Nikita Saxena
Research Engineer

Agenda

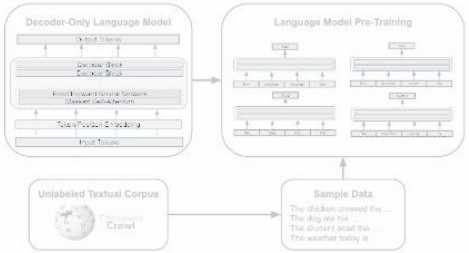
Introduction	01
Policy Optimization for Model Alignment	02
Beyond RLHF	03

Introduction

1

Alignment

Pre-Training



SFT

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

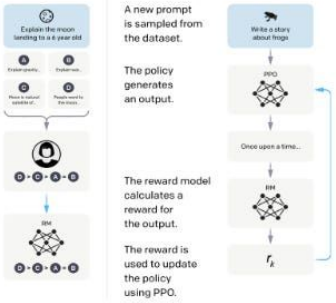
This data is used to fine-tune GPT-3 with supervised learning.

RLHF

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

01 Overview

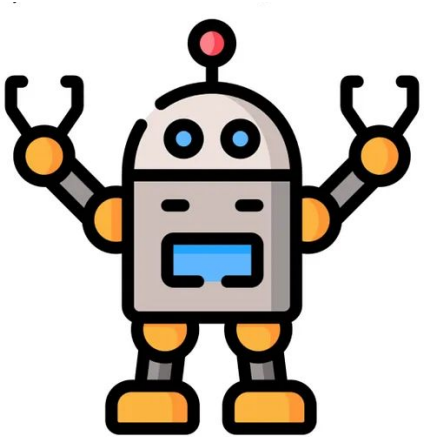
Quick Recap

We have a Supervised Fine-Tuned (SFT) Model

- Vast general knowledge.
- Learned the format of a helpful assistant.

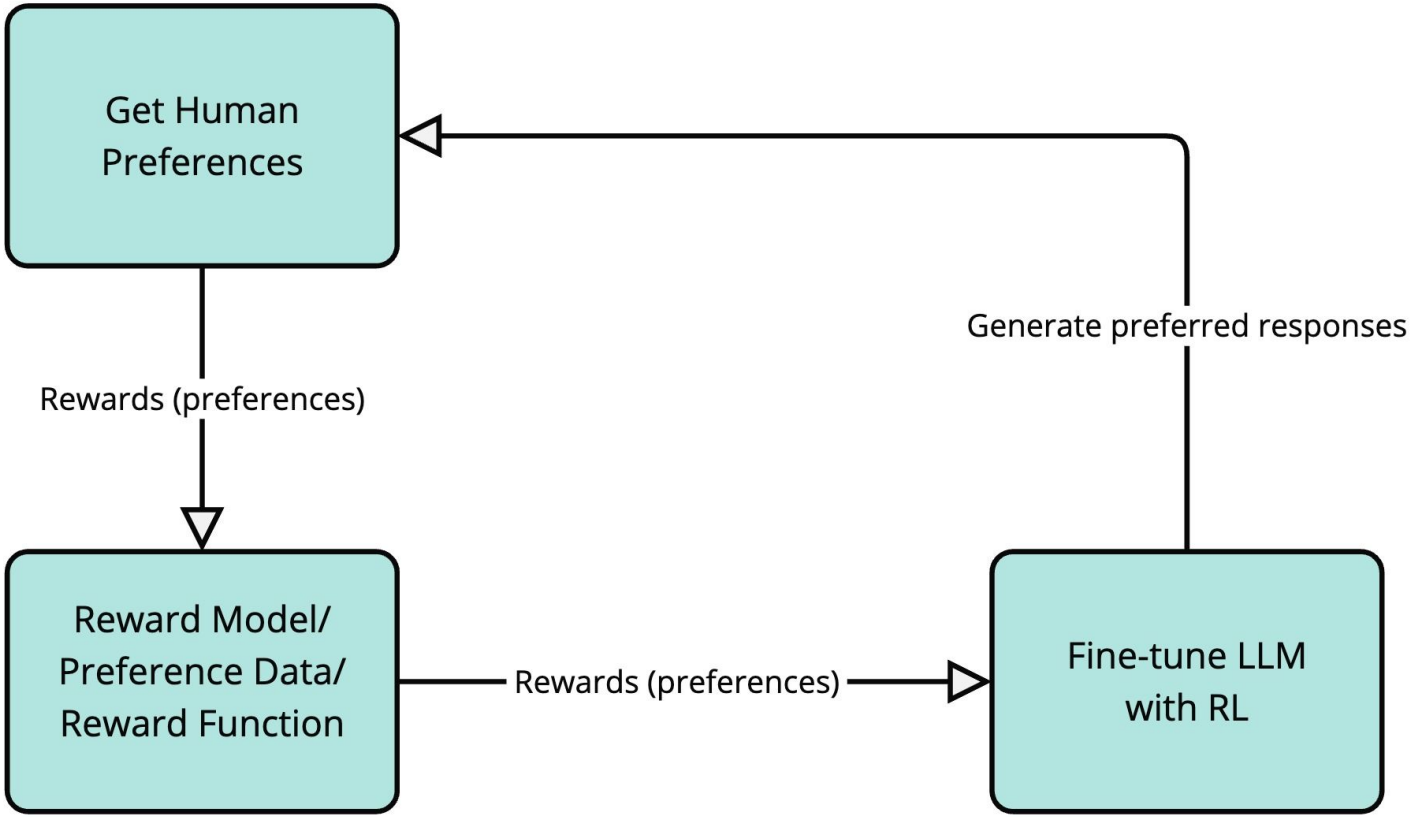
But... there are still major limitations:

- The "One Right Answer" Problem: SFT teaches the model that for any given prompt, there is a single, "correct" demonstrated response. The real world is much more nuanced.
- Scalability & Quality: Creating high-quality, detailed demonstrations for every possible scenario is incredibly expensive and difficult. It's much easier for a human to judge a good response than to write one from scratch.
- Subtlety is Lost: SFT struggles to capture subtle human preferences. What makes one good answer slightly better, safer, or more creative than another? This is hard to express in a single demonstration.



Benefits of RLHF

Benefit	Description
Improved Control	RL allows us to have more control over the kind of text LLMs generate. We can guide them to produce text that is more aligned with specific goals, like being helpful, creative, or concise.
Enhanced Alignment with Human Values	It's hard to write down rules for "what makes a good answer," but humans can easily judge and compare responses. RLHF lets the model learn from these human judgments.
Mitigating Undesirable Behaviors	RL can be used to reduce negative behaviors in LLMs, such as generating toxic language, spreading misinformation, or exhibiting biases. By designing rewards that penalize these behaviors, we can nudge the model to avoid them.



Step 1

Collect demonstration data, and train a supervised policy.

SFT

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

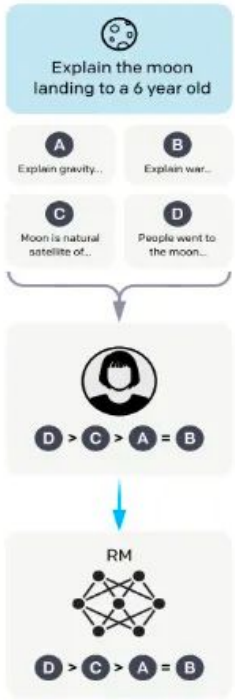
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using reinforcement learning.

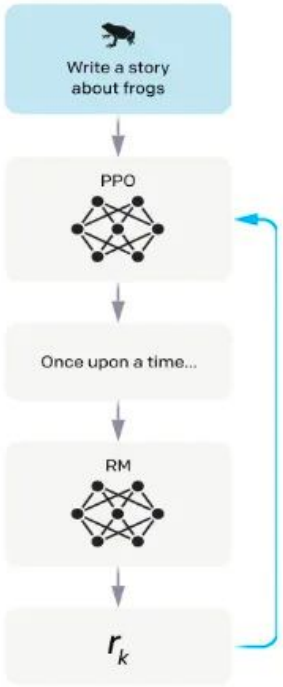
RLHF

A new prompt is sampled from the dataset.

The policy generates an output.

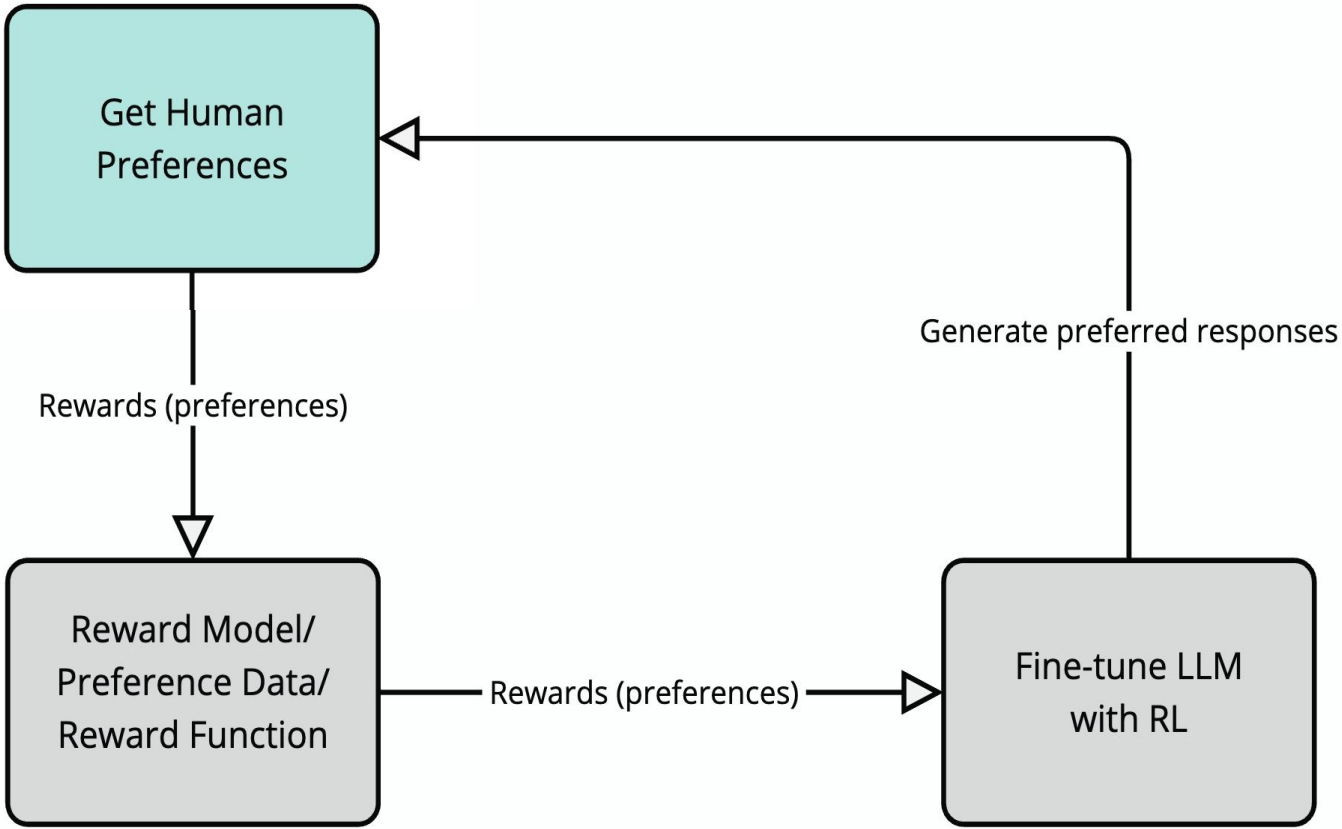
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



02

Human Preference Data Collection



Overview

Goal

Translate subjective human judgments about LLM outputs (text, images, etc.) into a structured dataset that a Reward Model can learn from. Dataset collection in the order of thousands.

Data Collection

- The SFT Model generates multiple responses to various prompts.
- Human labelers evaluate these responses.

Evaluation Approaches: Likert Scale (Absolute Scoring)

How

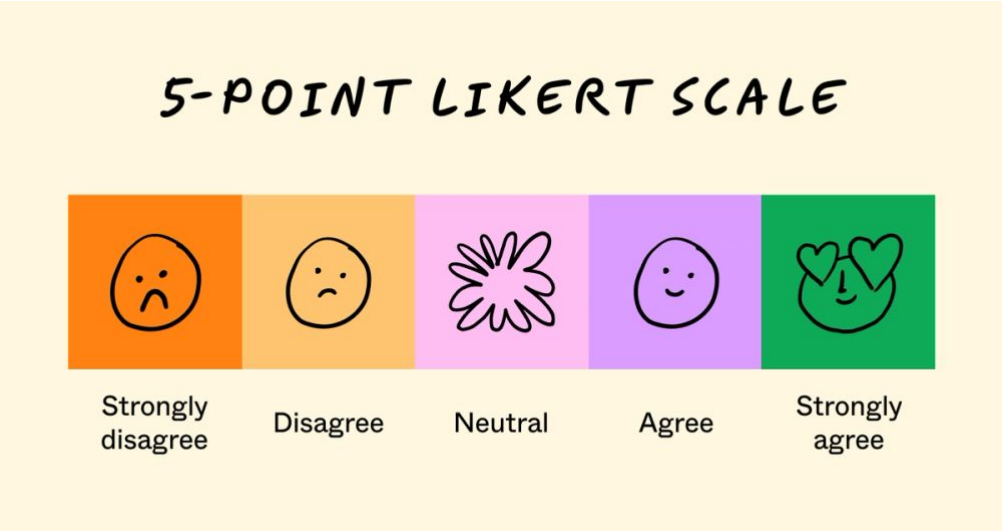
Humans rate responses against predefined criteria (e.g., helpfulness, realism, safety) on a fixed scale (e.g., 1-5 stars, "Strongly Disagree" to "Strongly Agree").

Example (Images)

Rate image on "Photorealism (1-5)," "Aesthetics (1-5)," "Prompt Adherence (1-5)."

Pro: Granular feedback on multiple aspects.

Con: Can be subjective; scale interpretation varies.



Evaluation Approaches:
Side-by-Side (SxS) Comparisons (Relative Scoring)

How

Humans are shown two (or more) outputs generated from the *same prompt* and asked to choose the best one, or rank them.

Example

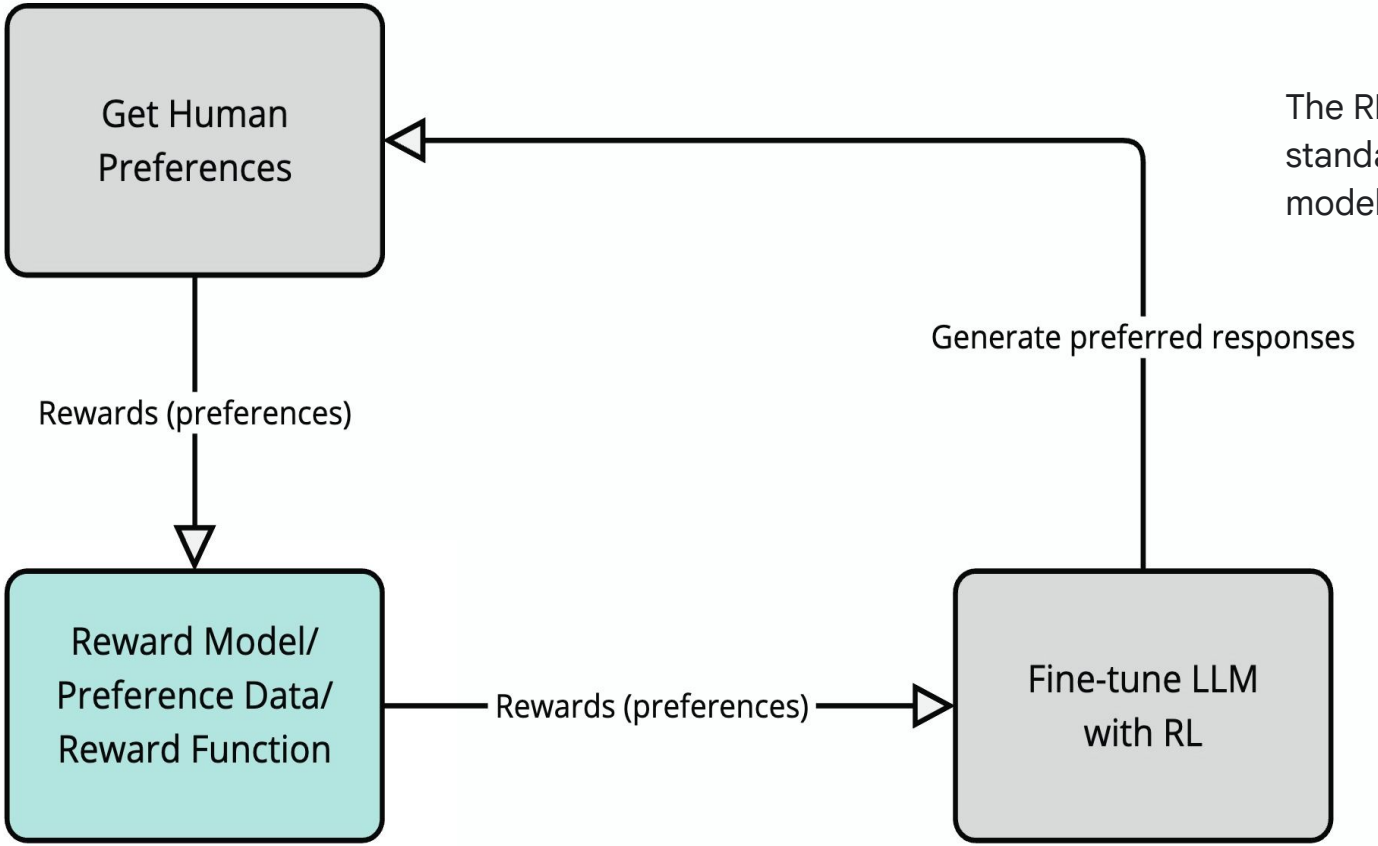
Given two images for prompt "a cat wearing a hat," select "Image A is better," "Image B is better," or "Roughly Equal."

Pro: Often more intuitive and reliable for humans; directly elicits relative preference.

Con: Less detail on *why* one is better without further questions.

02

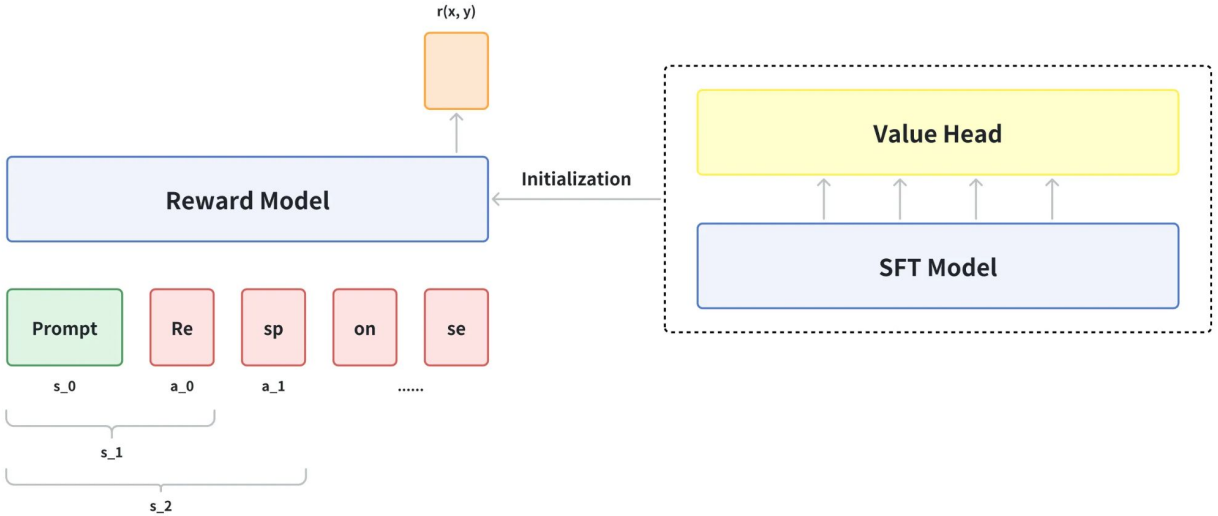
Train the Reward Model



The RM is trained like a standard supervised learning model.

Model Architecture

Often, a pre-trained Large Language Model (e.g., a copy of the SFT model, or another suitable LLM) is used as the base. The final layer of this LLM is modified or replaced with a linear layer that outputs a single scalar value (the reward score) instead of a sequence of tokens.



Input Data Preparation

For SxS Comparison Data (Most Common for RLHF)

Each human judgment "Response A is preferred over Response B (for Prompt P)" becomes a training instance. This typically involves pairs of (prompt, chosen_response) and (prompt, rejected_response).

For Likert Scale Data

Each (prompt, response, human_score) becomes a training instance.

Training: For Likert Scale Data

The RM processes the (prompt, response) to get a predicted score:

```
predicted_score = RM(prompt, response)
```

Loss Function

Typically Mean Squared Error (MSE) between the predicted_score and the human_score.

$$\text{loss} = (\text{predicted\_score} - \text{human\_score})^2$$

Training: For SxS Comparison Data

The RM processes both the chosen and rejected responses (for the same prompt) to get their respective scores:

```
score_chosen = RM(prompt, chosen_response)
score_rejected = RM(prompt, rejected_response)
```

Loss Function

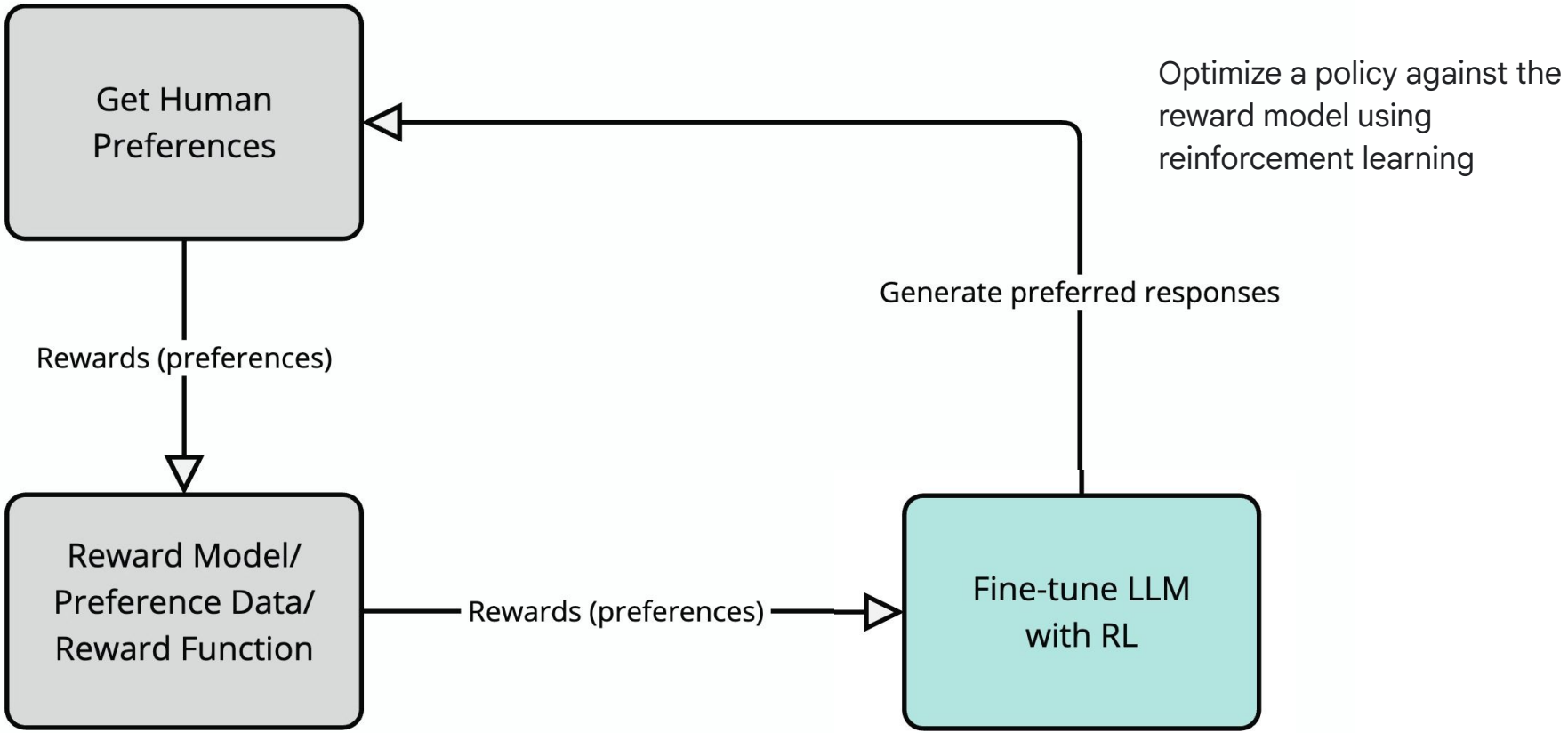
A pairwise ranking loss is commonly used. The objective is to maximize the margin between the score of the preferred response and the score of the dispreferred response.

```
loss = -log(sigmoid(score_chosen - score_rejected))
```

This encourages `score_chosen` to be significantly higher than `score_rejected`.

02

Fine-tuning the LLM

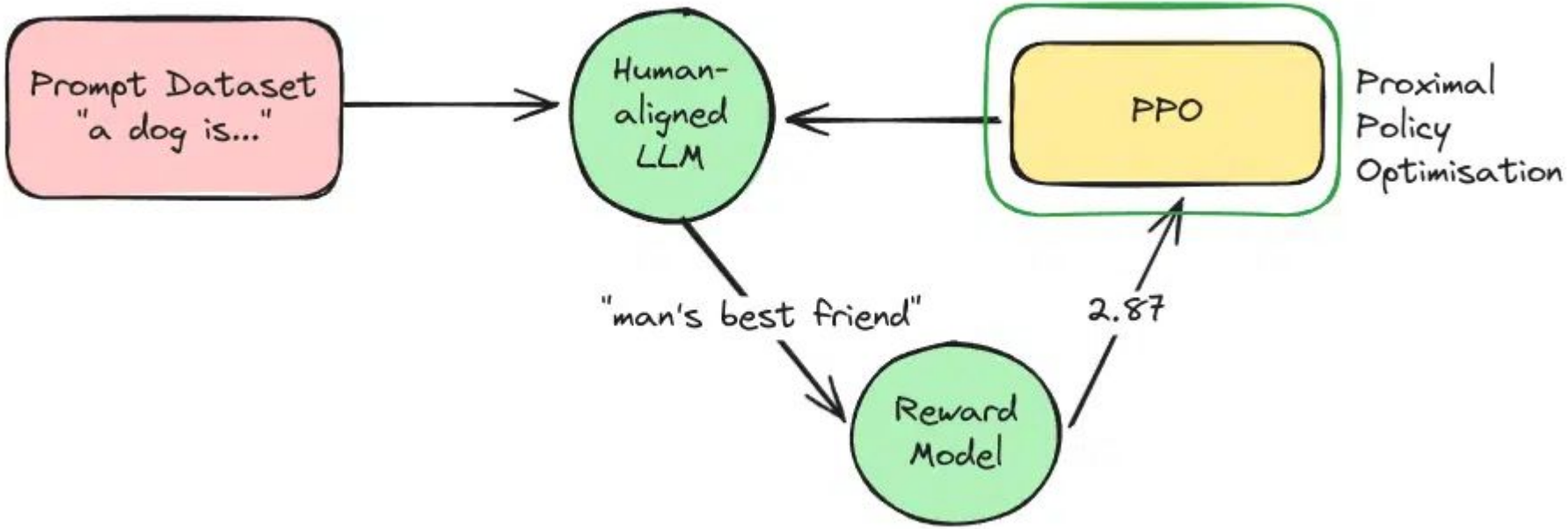


Policy Optimization for Model Alignment

2

01

Proximal Policy Optimization



Policy Gradient Loss: L^{policy}

Intuition

- Maximises the probability of taking good actions.
- Calculated by comparing the new policy π to the old policy π_o using importance sampling.
- Clipping
  - Ensures that updates remain close to the original policy.
  - Controls the step size of updates hence preventing large updates that could hurt the performance of the agent, thus ensuring the “proximal” policy optimisation.

Mathematical Formulation

$$L^{policy} = E[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

- r□(θ) is the ratio of the probabilities according to the new and old policies,
- A_t is the advantage function that indicates how much better an action is compared to the average of all actions,
- ϵ is a hyper-parameter that controls the degree of trust region in PPO

“

Imagine our Actor-Critic Mario is learning a difficult jump. His current policy is pretty good. One time, by sheer fluke, he tries a wild, spinning jump. A glitch in the game sends him flying through a wall, and he lands right at the flagpole, getting a massive, unprecedented score.

A "naive" Actor-Critic would be overwhelmed by this. The Critic would give a gigantic positive signal. The Actor would react by thinking, "That wild spin is the secret! I must do it every time!" It would change its policy to a 100% chance of doing that wild spin.

What happens next? He tries the spin again, but there's no glitch. He slams into the wall and dies. He has "fallen off a performance cliff" because he overreacted to one lucky event.

“

PPO prevents this by acting like a very wise and cautious coach. Its core rule is: "We will not change our strategy too much in a single update, no matter how good or bad the last result was.”

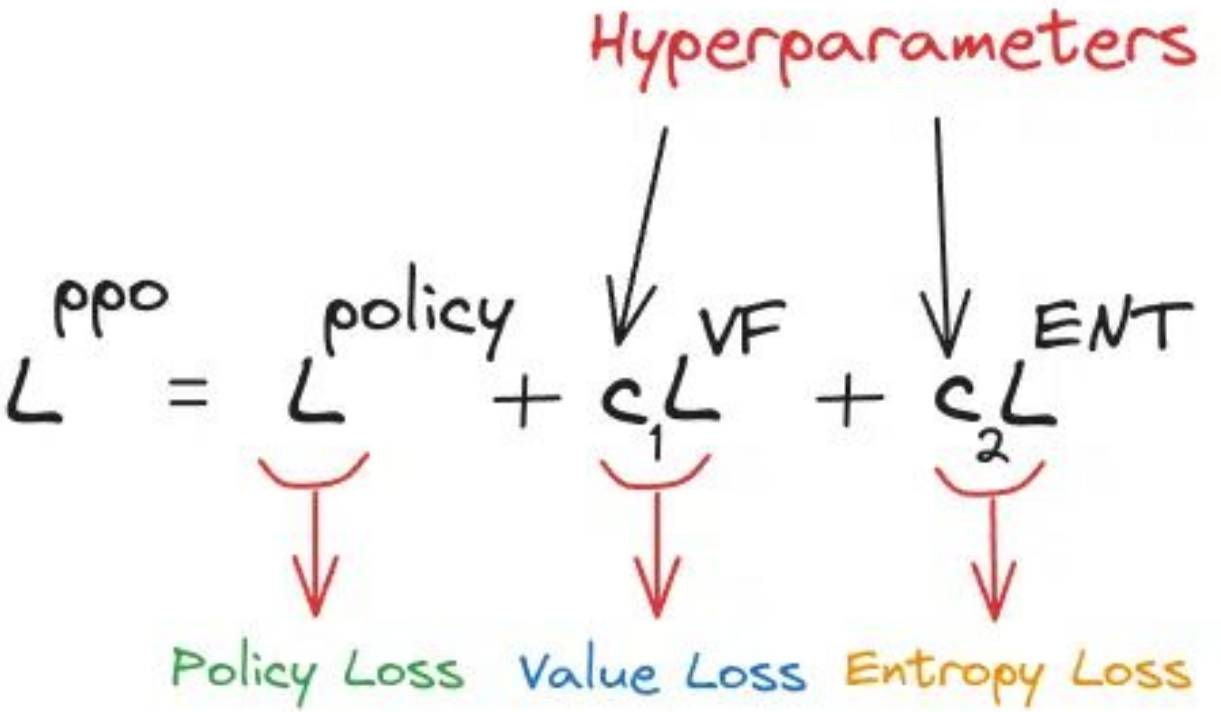
The PPO coach sees the same lucky, glitchy run. Instead of getting overly excited, the coach says, "Okay, that was a huge score, which is interesting. But it was probably a fluke. We can't throw away our entire playbook based on that. Let's incorporate a little bit of that new move, but not too much

“

PPO enforces this by "clipping" the policy update. It looks at how much the Actor wants to change the policy. If the proposed change is too big (e.g., "let's go from a 10% chance of a wild spin to a 90% chance!"), PPO clips it and says, "Nope. The maximum we will change this update is 5%. You can go from 10% to 15%, and we'll see how that works out first.

PPO leads to much smoother, more reliable, and more stable training. RL-Mario makes small, incremental improvements, testing the waters with each change. He doesn't take wild risks in his learning process.

Objective Function



Value Function Loss: L^{vf}

Intuition

- Minimises the error in value estimation.
- The value function predicts the future reward from a state, leading to better decision making by the agent.

Mathematical Formulation

$$L^{value} = c_1 * E_t[(V_w(s_t)V_t)^2]$$

- $V_w(s_t)$ is the estimated ‘value’ of the current state,
- V_t is the observed return

Entropy Bonus: L^{ENT}

Intuition

- Encourages exploration by adding a bonus for more random policies.
- Entropy of a policy is a measure of its randomness. It is used to improve exploration by discouraging premature convergence to a sub-optimal policy.

Mathematical Formulation

$$L^{entropy} = c_2 E_t[S\pi_\theta]$$

- $V_w(s_t)$ is the estimated 'value' of the current state,
- $S\pi_\theta$ is the entropy bonus which is a measure of the randomness of the policy

Significant Computational Overload

PPO Iterations

The PPO phase involves numerous iterations of:

1. Generating responses with the current policy (LLM).
2. Scoring these responses with the Reward Model.
3. Calculating advantages.
4. Performing policy updates.

Each step consumes considerable computational resources, especially with large LLMs

Large-Scale Reward Models:

Effective Reward Models (RMs) often need to be comparable in size and complexity to the original LLM they are evaluating.

Training and running inferences with these large RMs during PPO updates contribute significantly to computational demands.

This requires substantial GPU memory and processing power.

02

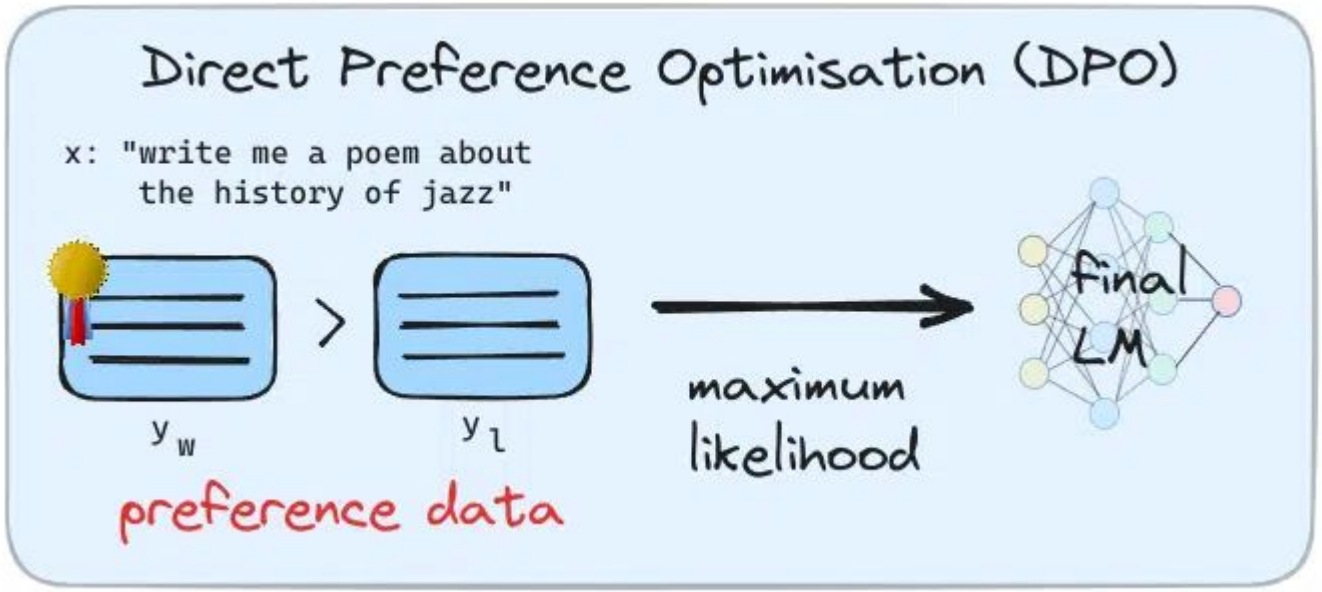
Direct Preference Optimization

Overview

DPO streamlines the alignment process by **eliminating the explicit need for a separate, trained reward model.** The language model (policy) uses these preference pairs to directly adjust its parameters.

The model learns to increase the likelihood of generating the *preferred* response and decrease the likelihood of the *dispreferred* response for a given prompt.

It essentially learns an **implicit reward function from the preference data** as part of its own update rule.



“

Imagine we want to train a 'Stuntman Mario' whose goal is to be as flashy and entertaining as possible.

1. Gather the Preference Album:

We start with a basic Mario and generate thousands of pairs of short gameplay clips. A human judge looks at each pair and creates a massive "preference album."

- Page 1: Clip A (a fancy wall-jump) is preferred over Clip B (a simple jump).
- Page 2: Clip C (sliding down a wall) is preferred over Clip D (just falling).

...and so on for thousands of pages.

“

The DPO Update

DPO-Mario sits down with this album. For each page, it does a very simple calculation:

- It looks at the preferred clip (e.g., Clip A, the wall-jump). It asks: "Based on my current instincts, how likely was I to do that?"
- It looks at the rejected clip (e.g., Clip B, the simple jump). It asks: "And how likely was I to do that?"

The 'Tug-of-War' on Instincts

The DPO algorithm then directly adjusts Mario's policy. It performs a "tug-of-war":

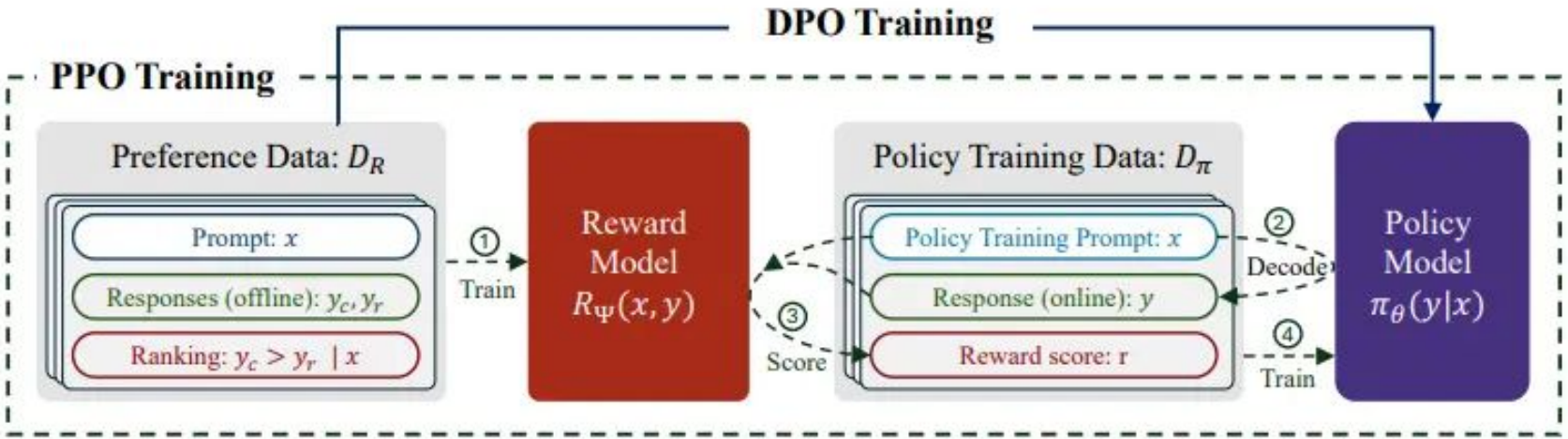
- It PULLS his instincts towards the preferred behavior, slightly increasing the probability that he will perform the actions from Clip A in the future.
- It PUSHES his instincts away from the rejected behavior, slightly decreasing the probability that he will perform the actions from Clip B.

“

DPO skips building a "judge" AI (reward model).

It lets Mario learn directly from a "preference album," nudging his instincts to be more like the chosen examples and less like the rejected ones.

DPO vs PPO



Objective Function

DPO reframes preference learning as a simple classification problem. The model should learn to assign a higher probability to the preferred completion y_w than the dispreferred completion y_l, given the prompt x.

To maximize the probability of observing the human preferences in our dataset, we minimize the negative log-likelihood:

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

- D: The dataset of human preferences.
- E : Expectation (average over the dataset).
- π_θ: The policy we are optimizing.
- π_{ref}: The fixed reference policy (e.g., the SFT model).
- β: Temperature parameter (hyperparameter, often set to 0.1 - 0.5).

$$\left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right)\right]$$

- **Inside the sigmoid:**This is the difference between the implicit reward of the winning response (y_w) and the losing response y_l.
- **Maximizing this difference**: The loss aims to make the policy π_θ assign a much higher log-probability (relative to π_{ref}) to y_w compared to y_l
- **β**: Controls how much the optimized policy π_θ can deviate from the reference policy π_{ref}. A higher **β** allows more deviation.

The DPO loss directly trains the language model to increase the relative log-probability of preferred responses over dispreferred ones, using a fixed reference model to regularize the learning and avoid drastic deviations. It cleverly bypasses explicit reward modeling by optimizing this preference likelihood directly.

03

Group Relative Policy Optimization

“

The PPO Way (The Single Coach)

The PPO coach looks at Mario's last game and says, "Let's take that strategy and make it a little bit better." It's always comparing to the immediate past.

The GRPO Solution (The Committee of Coaches)

GRPO is like replacing the single coach with a "committee of coaches," where each coach represents one of Mario's recent successful strategies.

“

Suppose our GRPO-Mario has just completed five successful runs of the level. His five past version, each with a slightly different strategy, form the committee:

- Speedrunner Mario: Very fast, but a bit risky.
- Cautious Mario: Slower, but never gets hit.
- Coin-Collector Mario: Gets a high score from coins, but isn't the fastest.
- Enemy-Stomper Mario: Also gets a high score, but from stomping everything.
- Balanced Mario: A mix of all the above.

“

The GRPO committee convenes. Their goal is to create a new, updated policy for the next game. Instead of just trying to improve on the last strategy (Balanced Mario), they ask a more robust question:

"How can we create a new strategy that is a guaranteed improvement over the average performance of this entire group?"

They are optimizing the policy "relative" to the "group." They might take the speed of Speedrunner Mario, the safety of Cautious Mario, and blend them into a new, superior strategy.

“

GRPO makes Mario learn from a "committee" of his recent successful past selves, not just his single last self. This makes the learning process more stable and less likely to be fooled by one-time lucky events.

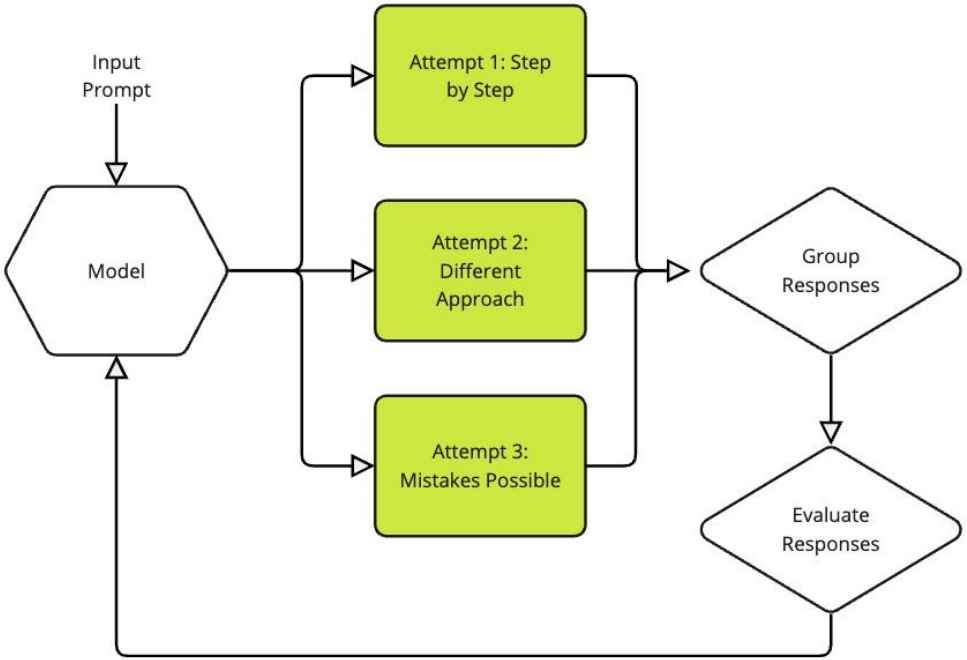
Group Formation: Creating Multiple Solutions

When given a prompt, the model doesn't just generate one response; instead, it creates multiple attempts at solving the same problem.

Imagine you're teaching a model to solve math problems. For a question about counting chickens on a farm, the model might generate several different solutions:

- One solution might break down the problem step by step
- Another might use a different but equally valid approach
- Some attempts might contain mistakes or less efficient solutions

All these attempts are kept together as a group, much like having multiple students' solutions to compare and learn from.



Preference Learning

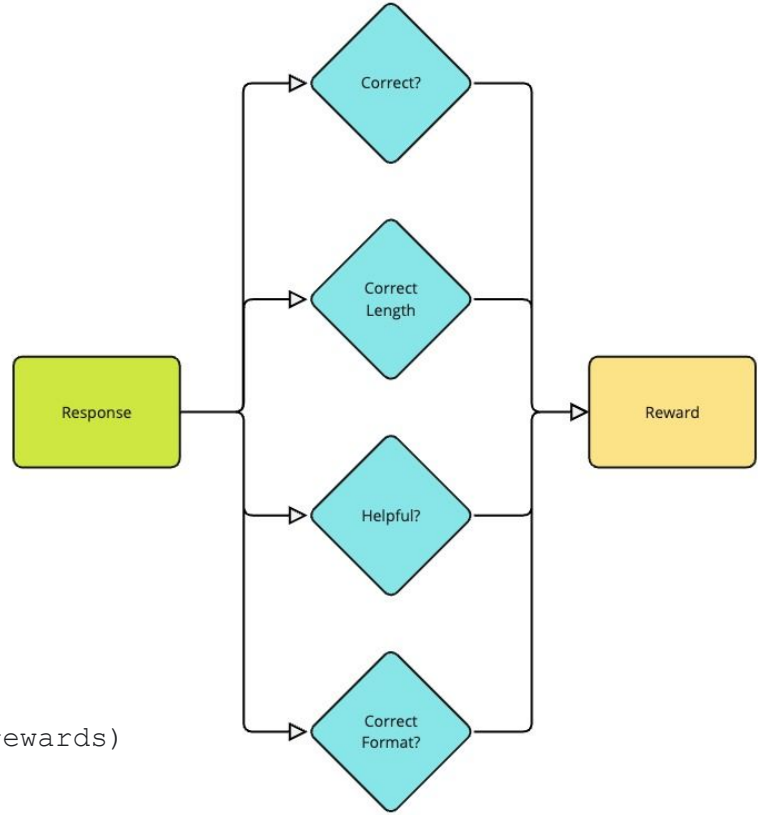
GRPO can use any function or model to evaluate the quality of a solution (what makes it different from other methods).

Example: Length function to reward shorter responses or a solver to reward accurate math solutions.

Instead of just giving absolute scores, GRPO normalizes the rewards within each group. It uses a simple but effective formula for group relative advantage estimation:

```
advantage = (reward - mean(group_rewards)) / std(group_rewards)
```

This normalization is like grading on a curve, but for AI. It helps the model understand which solutions within the group were better or worse compared to their peers, rather than just looking at absolute scores.



The evaluation process looks at various aspects of each solution: Is the final answer correct? Did the solution follow proper formatting (like using the right XML tags)? Does the reasoning match the answer provided?

Optimization

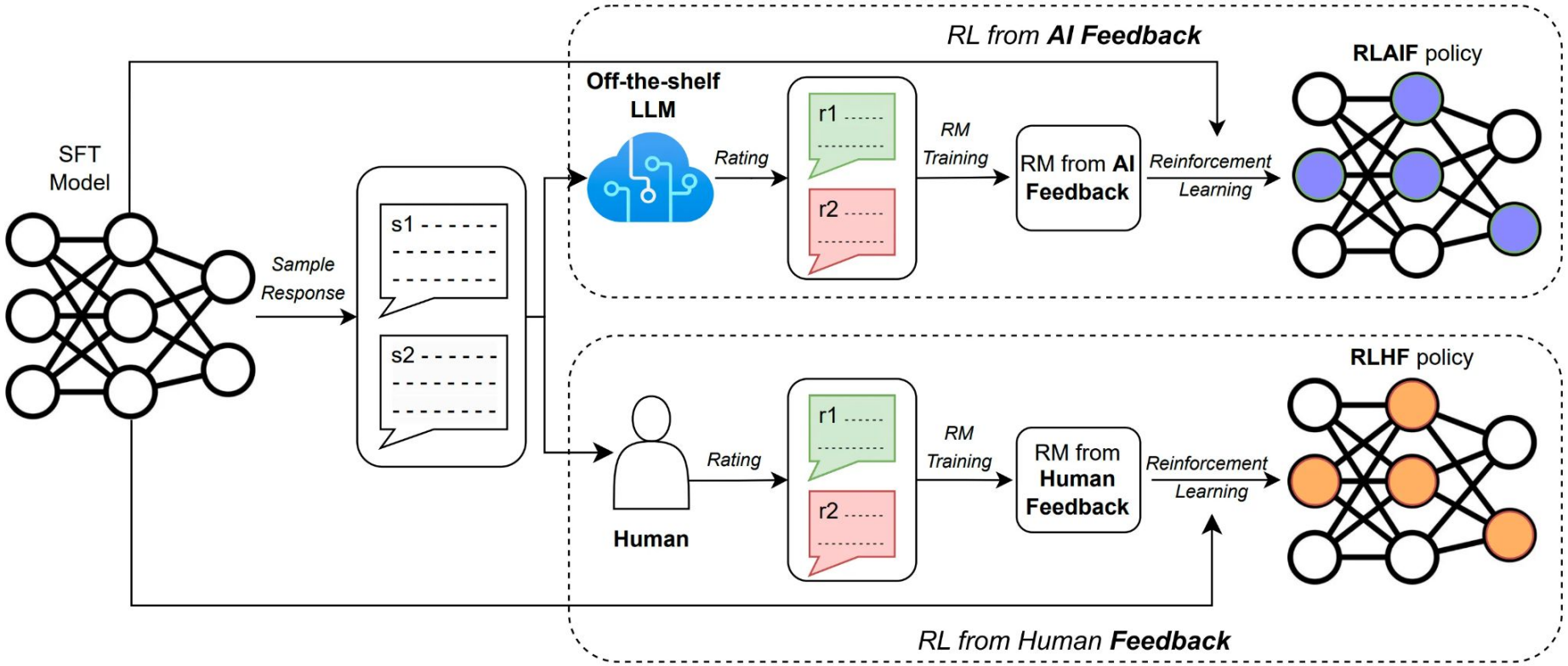
Teaches the model to improve based on what it learned from evaluating the group of solutions. It includes a safety mechanism **(called KL divergence penalty)** that prevents the model from changing too drastically all at once

This approach proves more stable than traditional methods because:

- It looks at multiple solutions together rather than comparing just two at a time
- The group-based normalization helps prevent issues with reward scaling

Beyond RLHF

RL-AI-F



Advantages

Scalability & Cost Efficiency

- Reduced Dependence on Human Annotators
  The AI Feedback Model can generate vast amounts of preference data much faster and cheaper than humans.
- Enables Training on Larger Scales
  Critical for training increasingly massive LLMs and for tasks requiring extensive feedback.
- Continuous Feedback
  AI can provide feedback 24/7, unlike human teams

Flexibility & Adaptability

- Altering Principles
  Changing the desired behavior is often as simple as updating the AI Feedback Model's instructions or fine-tuning it on new/revised criteria.
- Switching Domains/Tasks
  Adapting to new domains or tasks can involve reconfiguring or fine-tuning the AI Feedback Model, rather than starting human data collection from scratch.

References

1. Part 1: Key Concepts in RL — Spinning Up documentation
2. A Deep Dive into Policy Optimization Algorithms & Frameworks for Model Alignment - SuperAGI
3. Understanding the DeepSeek R1 Paper - Hugging Face LLM Course
4. A (Long) Peek into Reinforcement Learning | Lil'Log

Google DeepMind

Practical: Build your own RL Game

Colab Notebook

tme: 5 mins

Thank you.

Nikita Saxena
Research Engineer

Google DeepMind