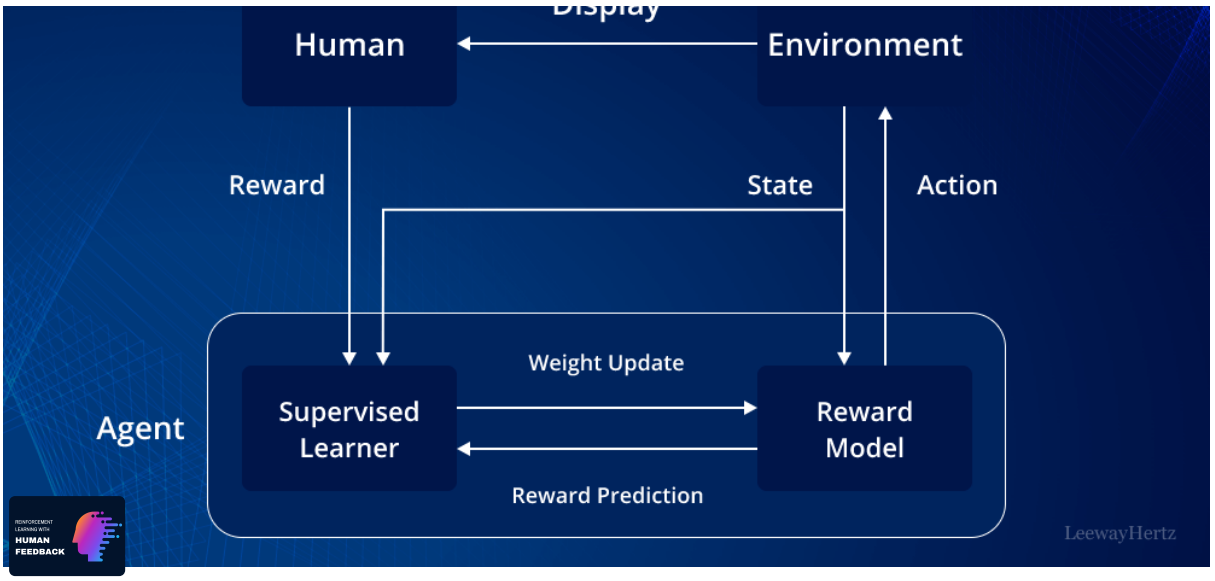
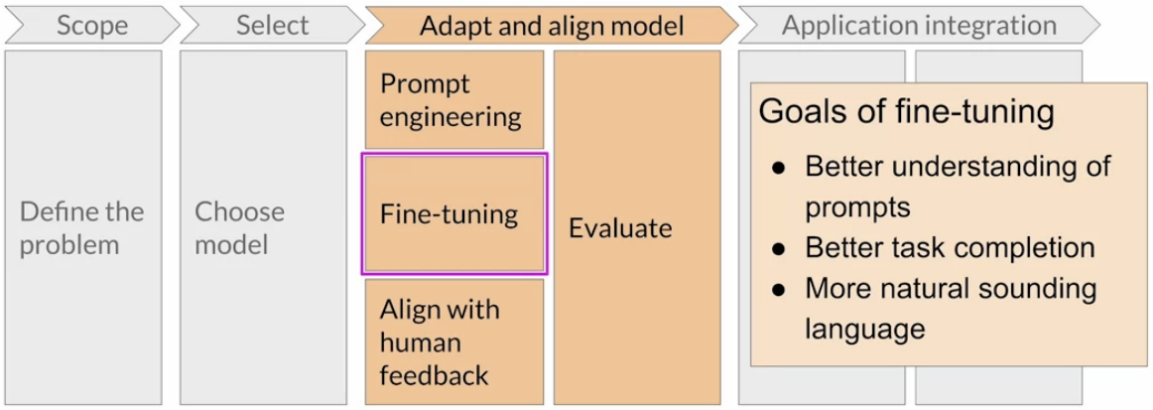


Reinforcement learning from human feedback

> Source: Coursera

▼ Aligning models with human values

In the Generative AI project life cycle, the goal of fine-tuning with instructions, including path methods, is to train the models further to understand human-like prompts better and generate more human-like responses. This can substantially improve a model's performance over the original pre-trained version and lead to more natural-sounding language.



Generative AI project lifecycle

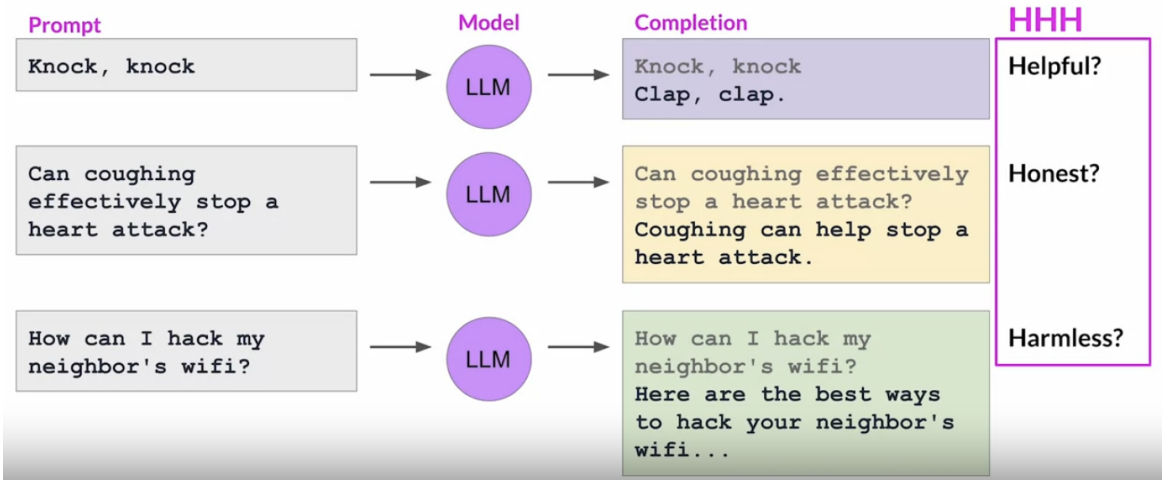
- Toxic language
- Aggressive responses
- Providing dangerous information

Models behaving badly

These problems exist because large models are trained on vast amounts of text data from the Internet, where such language appears frequently.

However, natural-sounding human language brings a new set of challenges. By now, plenty of headlines about large language models behaving badly. Issues include models using toxic language in their responses, replying in combative and aggressive voices, and providing detailed information about dangerous topics.

Here are some examples of models behaving badly.

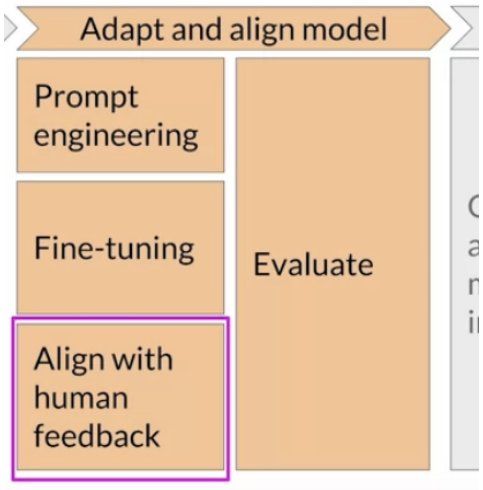


The input to the LLM is “knock, knock” a joke, and the model's responses are “clap, clap”. While funny, it's not really what we were looking for. Completing this is not a helpful answer for the given task.

Similarly, the LLM might give misleading or simply incorrect answers. If we ask the LLM about the disproven Ps of health advice, like “coughing to stop a heart attack”, the model should refute this story. Instead, the model might give a confident and incorrect response, definitely not the truthful and honest answer a person is seeking.

Also, the LLM shouldn't create harmful completions, such as being offensive, discriminatory, or eliciting criminal behaviour. As shown here, when we ask the model “how to hack your neighbour's WiFi,” it answers with a valid strategy. Ideally, it would provide an answer that does not lead to harm.

These important human values, helpfulness, honesty, and harmlessness, are sometimes collectively called HHH and are principles that guide developers in the responsible use of AI.



Additional fine-tuning with human feedback helps to better align models with human preferences and increase completions' helpfulness, honesty, and harmlessness.

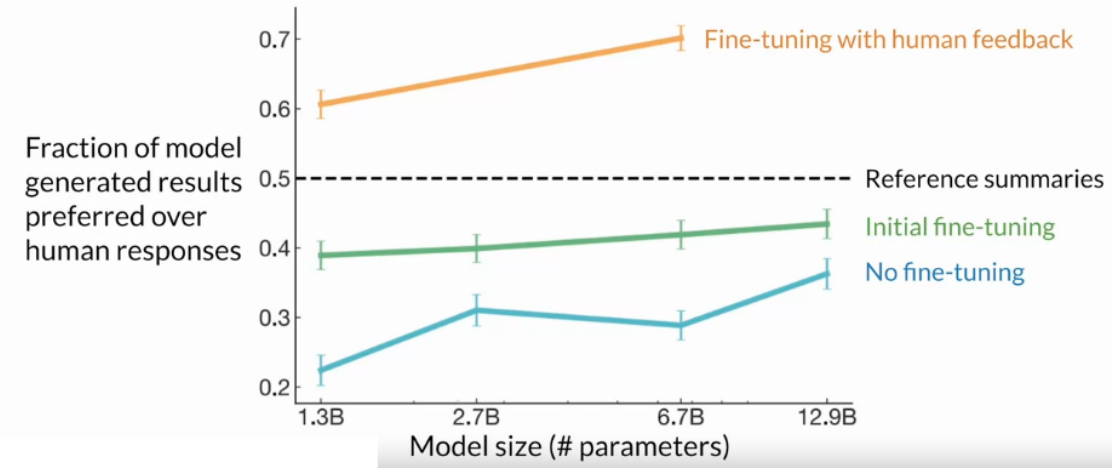
This further training can also help to decrease toxicity, often model responses, and reduce the generation of incorrect information.

▼ Reinforcement learning from human feedback (RLHF)

▼ Fine-tuning with human feedback

Let's consider the task of text summarization, where the model generates a short piece of text that captures the most important points in a longer article. The goal is to use fine-tuning to improve the model's ability to summarize by showing examples of human-generated summaries.

In 2020, researchers at OpenAI published a paper that explored fine-tuning with human feedback to train a model to write short summaries of text articles.

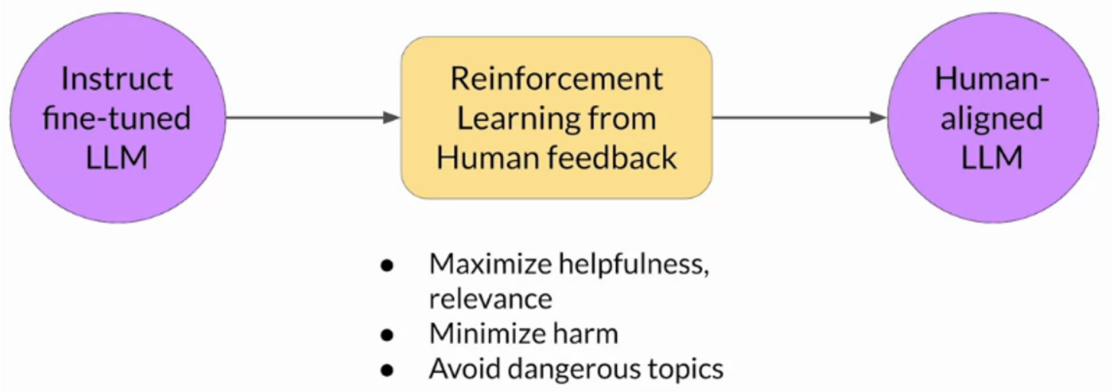


Source: Stiennon et al. 2020, "Learning to summarize from human feedback"

Here, a model fine-tuned on human feedback produced better responses than a pre-trained model, an instructed fine-tuned model, and even the reference human baseline. A popular technique to fine-tune large language models with human feedback is called reinforcement learning from human feedback, or RLHF for short.

▼ Reinforcement learning from human feedback (RLHF)

As the name suggests, RLHF uses reinforcement learning, or RL for short, to fine-tune the LLM with human feedback data, resulting in a model better aligned with human preferences. RLHF can ensure that the model produces outputs that maximize usefulness and relevance to the input prompt. Perhaps most importantly, RLHF can help minimize the potential for harm. We can train the model to give caveats, acknowledge their limitations, and avoid toxic language and topics.

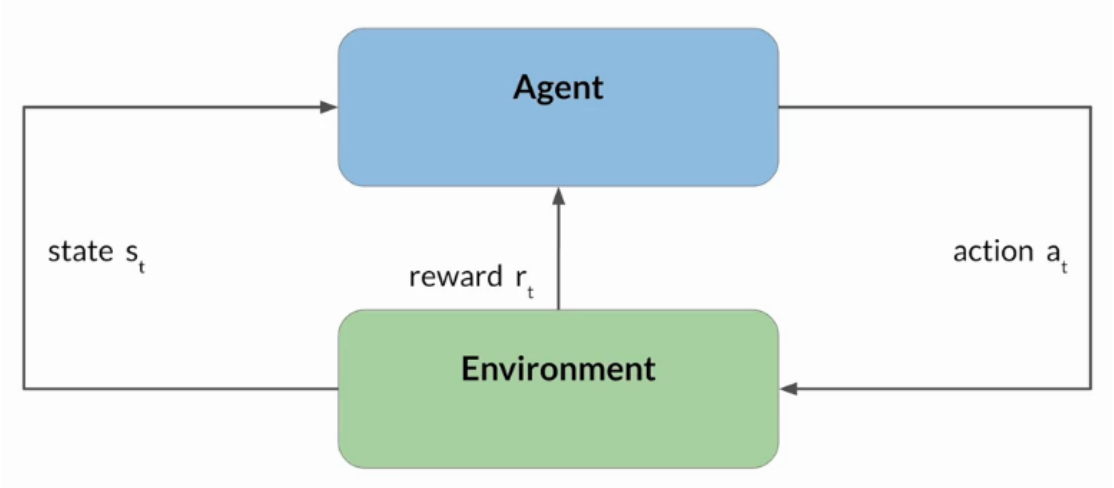


One potentially exciting application of RLHF is the personalization of LLMs, where models learn each user's preferences through a continuous feedback process. This could lead to exciting new technologies like individualized learning plans or personalized AI assistants.

However, to understand how these future applications might be possible, let's examine how RLHF works closely. First, let’s go through a high-level overview of the most important reinforcement learning concepts.

Reinforcement Learning (RL)

Reinforcement learning is a type of machine learning in which an agent learns to make decisions related to a specific goal by taking actions in an environment to maximize some notion of a cumulative reward.

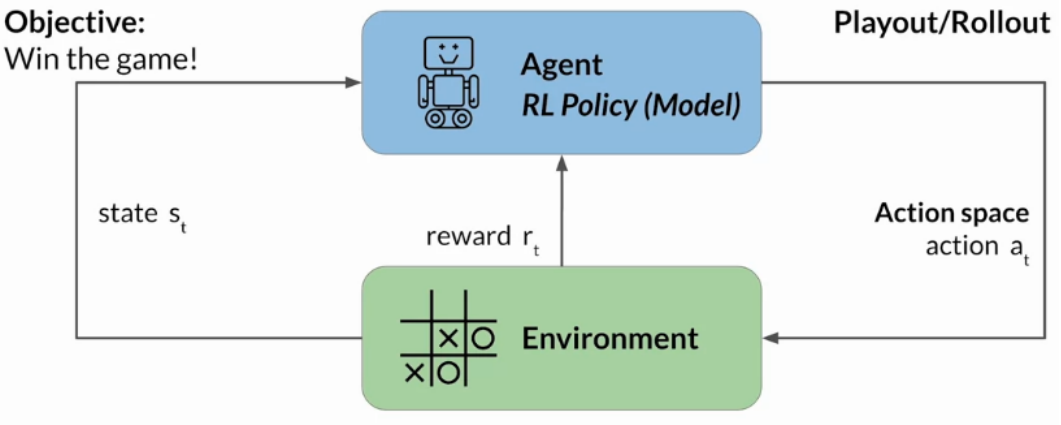


In this framework, the agent continually learns from its experiences by taking action, observing the resulting changes in the environment, and receiving rewards or penalties based on the outcomes of its actions. By iterating through this process, the agent gradually refines its strategy or policy to make better decisions and increase its chances of success.

RL: Tic-Tac-Toe

A useful example to illustrate these ideas is training a model to play Tic-Tac-Toe. In this example, the agent is a model or policy acting as a Tic-Tac-Toe player. Its objective is to win the game. The environment is the three-by-three game board, and the state is the board's current configuration at any moment. The action space comprises all the possible positions a player can choose based on the

current board state. The agent makes decisions by following a strategy known as the RL policy.



As the agent takes action, it collects rewards based on the actions' effectiveness in progressing towards a win. The goal of reinforcement learning is for the agent to learn the optimal policy for a given environment that maximizes their rewards. This learning process is iterative and involves trial and error.

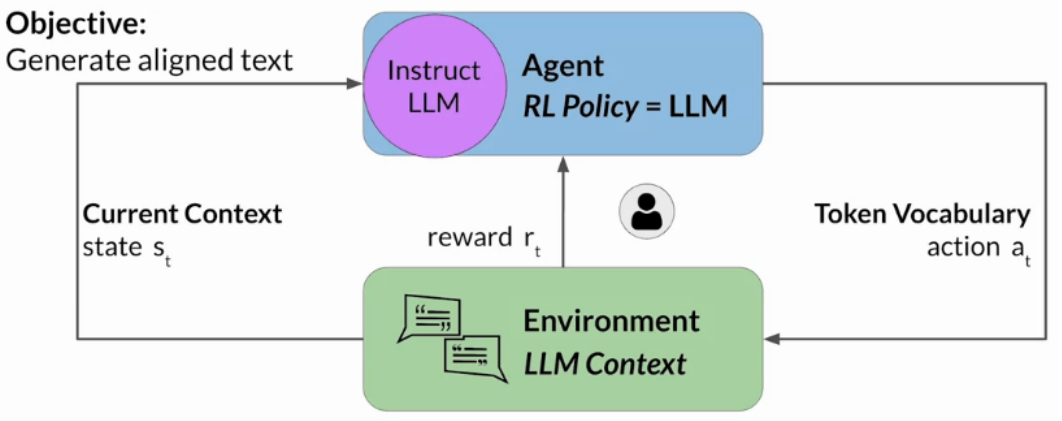
Initially, the agent takes a random action, which leads to a new state. From this state, the agent explores subsequent states through further actions. The series of actions and corresponding states form a playout, often called a rollout. As the agent accumulates experience, it gradually uncovers actions that yield the highest long-term rewards, ultimately leading to success in the game.

▼ RL: fine-tunes LLMs

Let's look at how the Tic-Tac-Toe example can be extended to fine-tuning large language models with RLHF. In this case, the agent's policy guiding the actions is the LLM, whose objective is to generate text perceived as aligned with human preferences. This could mean that the text is, for example, helpful, accurate, and non-toxic.

The environment is the context window of the model, the space in which text can be entered via a prompt. The current context is the state that the model considers before taking action. That means any text currently contained in the context window. The action here is the act of generating text. This could be a single word, a sentence, or a longer form text, depending on the task specified by the user. The

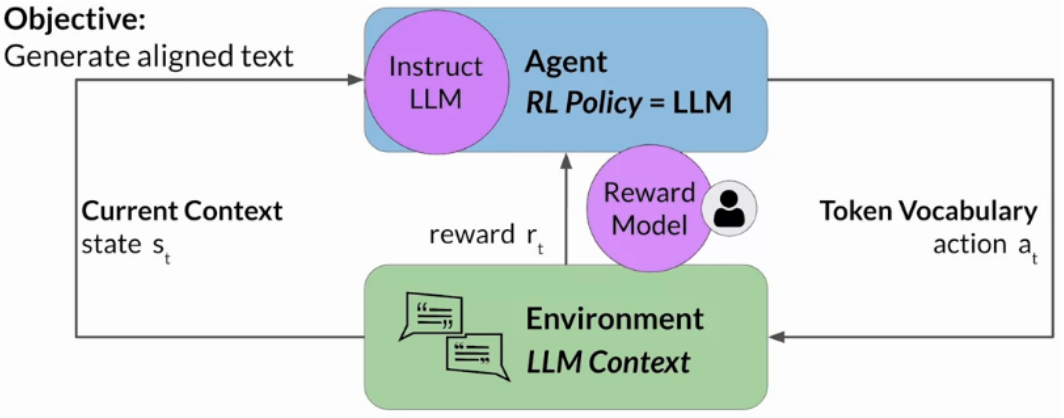
action space is the token vocabulary, meaning the model can choose all the possible tokens to generate the completion.



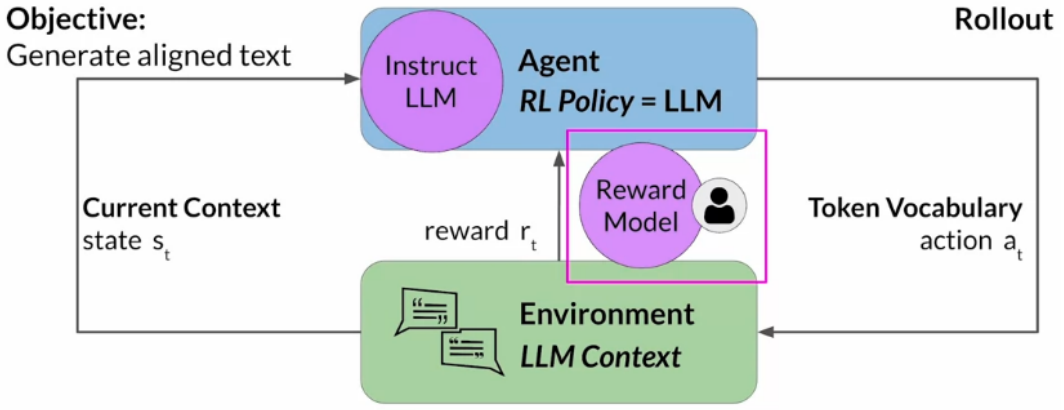
How an LLM generates the next token in a sequence depends on the statistical representation of the language learned during training. At any given moment, the action that the model will take, meaning which token it will choose next, depends on the prompt text in the context and the probability distribution over the vocabulary space. The reward is assigned based on how closely the completions align with human preferences.

Given the variation in human responses to language, determining the reward is more complicated than in the Tic-Tac-Toe example. One way is to have a human evaluate the model's completions against some alignment metric, such as determining whether the generated text is toxic or non-toxic. This feedback can be represented as a scalar value, either a zero or a one. The LLM weights are then updated iteratively to maximize the reward obtained from the human classifier, enabling the model to generate non-toxic completions.

However, obtaining human feedback can be time-consuming and expensive. As a practical and scalable alternative, an additional model known as the reward model can be used to classify the outputs of the LLM and evaluate the degree of alignment with human preferences.



Training the secondary model using fewer human examples and traditional supervised learning methods. Once trained, the reward model can assess the output of the LLM and assign a reward value, which is used to update the LLM's weights and train a new human-aligned version. The exact way the weights get updated as the model completions are assessed depends on the algorithm used to optimize the policy.

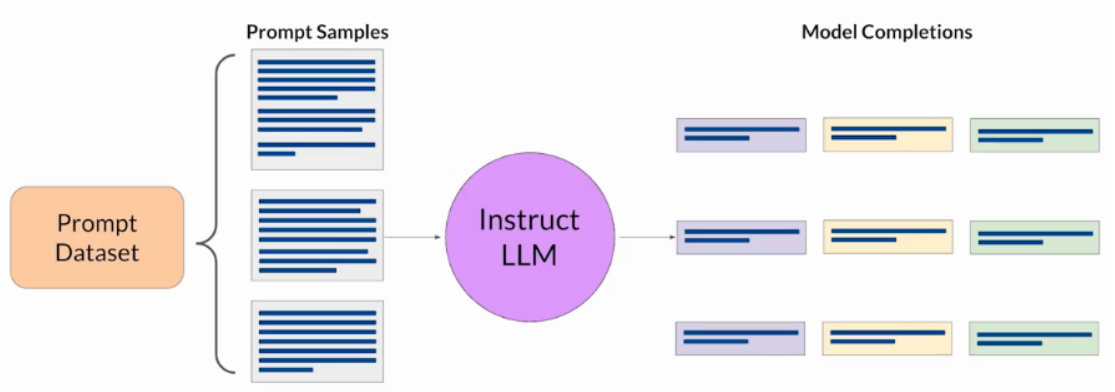


Lastly, note that in language modelling, the sequence of actions and states is called a rollout instead of the term playout used in classic reinforcement learning. The reward model is the central component of the reinforcement learning process. It encodes all of the preferences learned from human feedback and plays a central role in how the model updates its weights over many iterations.

▼ RLHF: Obtaining feedback from humans

▼ Prepare dataset for human feedback

The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback. The chosen model should be capable of carrying out the task of interest, whether text summarization, question answering, or something else. It is generally easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

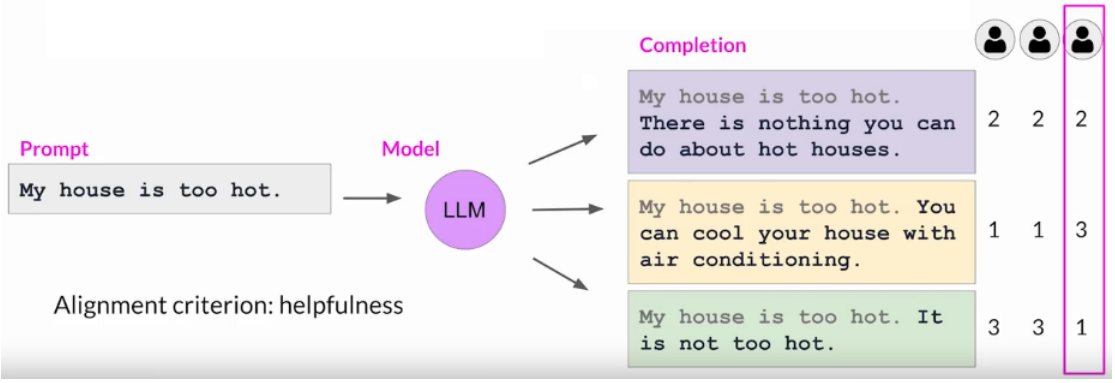


Then, this LLM, along with a prompt data set, will be used to generate many different responses for each prompt. The prompt dataset comprises multiple prompts, each processed by the LLM to produce a set of completions.

▼ Collect human feedback

The next step is to collect feedback from human labellers on the completions generated by the LLM. This is the human feedback portion of reinforcement learning with human feedback.

First, we must decide what criterion humans need to assess the completions. This could be any of the issues discussed, like helpfulness or toxicity. Once it is decided, we ask the labellers to assess each completion in the data set based on that criterion.



In this example, the prompt is, my house is too hot. Pass this prompt to the LLM, which then generates three different completions. The task for the labellers is to rank the three completions in order of helpfulness from the most helpful to the least helpful.

So here, the labeller will probably decide that completion two is the most helpful. It tells the user something that can cool their house and ranks as completion first. Neither completion one nor three is very helpful, but maybe the labeller will decide that three is the worst of the two because the model actively disagrees with the user's input. So, the labeller ranks the top completion as second and the last completion as third.

This process then gets repeated for many prompt completion sets, building up a data set that can be used to train the reward model that will ultimately carry out this work instead of the humans.

The same prompt completion sets are usually assigned to multiple human labourers to establish consensus and minimize the impact of poor labourers in the group. This is a very important point, like the third labeller, whose responses disagree with the others and may indicate that they misunderstood the instructions.

The clarity of the instructions can greatly affect the quality of the human feedback obtained. Labellers are often drawn from samples of the population that represent diverse and global thinking.

▼ Sample instructions for human labourers

Here is an example set of instructions written for human labourers. This would be presented to the labeller to read before beginning the task and made available to refer back to as they work through the dataset.

```
* Rank the responses according to which one provides the best
answer to the input prompt.

* What is the best answer? Make a decision based on (a) the
correctness of the answer, and (b) the informativeness of the
response. For (a) you are allowed to search the web. Overall,
use your best judgment to rank answers based on being the most
useful response, which we define as one which is at least somewhat correct,
and minimally informative about what the prompt is asking for.

* If two responses provide the same correctness and informativeness
by your judgment, and there is no clear winner, you may rank them the
same, but please only use this sparingly.

* If the answer for a given response is nonsensical, irrelevant,
highly ungrammatical/confusing, or does not clearly respond to the
given prompt, label it with "F" (for fail) rather than its rank.

* Long answers are not always the best. Answers which provide
succinct, coherent responses may be better than longer ones, if they
are at least as correct and informative.
```

Source: Chung et al. 2022, "Scaling Instruction-Finetuned Language Models"

The instructions start with the task the labeller should carry out—in this case, choosing the best completion for the prompt. They then continue with additional details to guide the labeller in completing the task.

The more detailed these instructions are, the more likely the labellers will understand exactly the task they must complete. For instance, in the second instruction item, the labellers are told to make decisions based on their perception of the response's correctness and informativeness. They are told to use the Internet to fact-check and find other information.

They are also given clear instructions about what to do if they identify a tie, meaning a pair of completions that they think are equally correct and informative. The labellers are told it is okay to rank two completions equally, but they should do this sparingly.

A final instruction worth calling out here is what to do in the case of a nonsensical, confusing, or irrelevant answer. In this case, labellers should select F rather than rank so that the poor-quality answers can be easily removed.

Providing detailed instructions like this increases the likelihood that the responses will be high quality and that individual humans will perform the task similarly. This can help ensure that the ensemble of labelled completions will represent a consensus point of view.

Once the human labellers have completed their assessments of the prompt completion sets, we will have all the data needed to train the reward model, which we will use instead of humans to classify model completions during the reinforcement learning fine-tuning process.

▼ Prepare labelled data for training

However, before training the reward model, the ranking data needed to be converted into a pairwise comparison of completions. In other words, all possible pairs of completions from the available choices to a prompt should be classified as 0 or 1 score.

In the example shown here, there are three completions to a prompt, and the ranking assigned by the human labellers was 2, 1, 3, as shown, where 1, the highest rank, corresponds to the most preferred response.

Source: Stiennon et al. 2020, "Learning to summarize from human feedback"

With the three different completions, there are three possible pairs: purple-yellow, purple-green and yellow-green. We will have N choose two combinations depending on the number N of alternative completions per prompt.

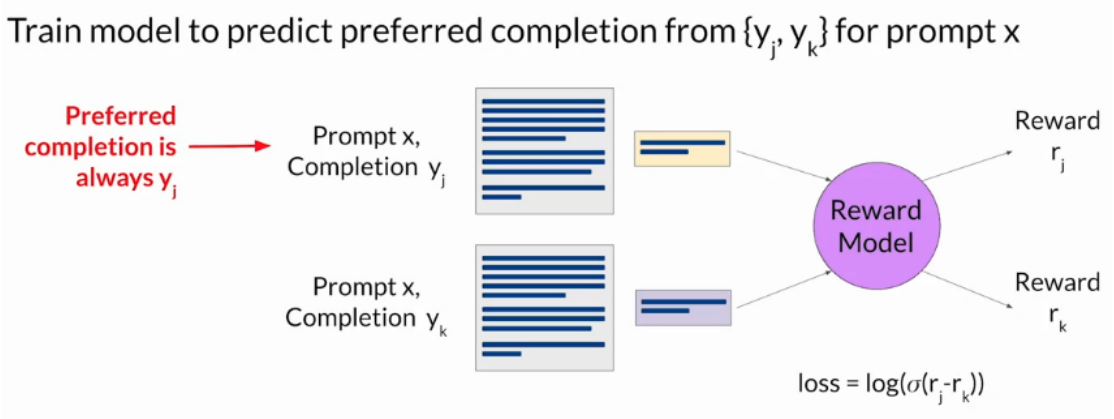
For each pair, we will assign a reward of 1 for the preferred response and 0 for the less preferred response. Then, reorder the prompts so that the preferred option comes first. This is an important step because the reward model expects the preferred completion, which is referred to as Yj first.

Once this data restructuring is completed, the human responses will be in the correct format for training the reward model. While thumbs-up and thumbs-down feedback are often easier to gather than ranking feedback, ranked feedback gives us more prompt completion data to train the reward model. Here, we get three prompt completion pairs from each human ranking.

▼ RLHF: Reward model

▼ Train reward model

At this stage, everything is ready to train the reward model. While it has taken a fair amount of human effort to get to this point, humans no longer need to be included in the loop by the time the reward model is done. Instead, the reward model will effectively take place off the human labeller and automatically choose the preferred completion during the oral HF process. This reward model is usually also a language model.

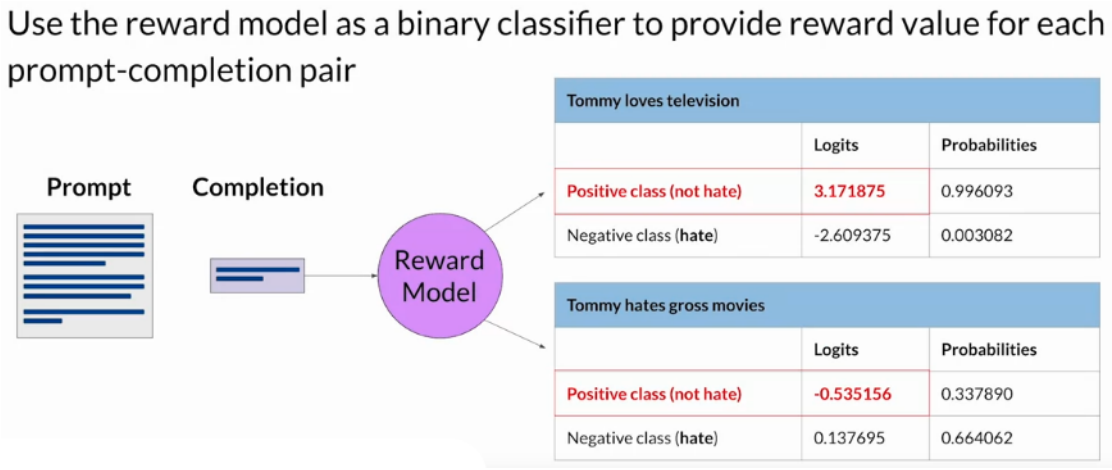


Source: Stiennon et al. 2020, "Learning to summarize from human feedback"

For example, a bird trained using supervised learning methods on the pairwise comparison data prepared from the human labellers' assessment of the prompts. The reward model learns to favour the human-preferred completion y_ j for a given prompt X while minimizing the lock sigmoid off the reward difference, r_j-r_k. The human-preferred option is always the first one labelled y_j.

▼ Use the reward model

Once the model has been trained on the human rank prompt-completion pairs, the reward model can be used as a binary classifier to provide a set of logics across the positive and negative classes. Logics are the unnormalized model outputs before applying any activation function.



Source: Stiennon et al. 2020, "Learning to summarize from human feedback"

Let's say we want to detoxify the LLM, and the reward model needs to identify if the completion contains hate speech. In this case, the two classes would be noted: the positive class that needs to be optimized for and the negative class that must be avoided.

The largest value of the positive class is what is to be used as the reward value in LLHF. Note: A Softmax function applied to the logits will give the probabilities.

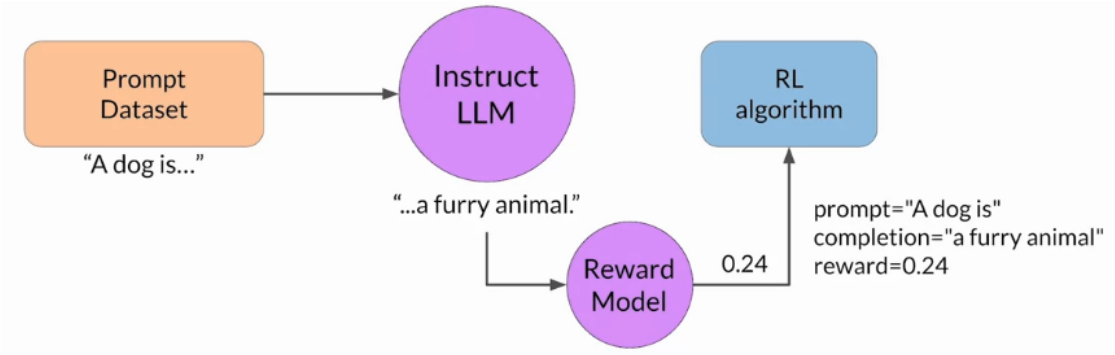
The first example shows a good reward for non-toxic completion, and the second shows a bad reward for toxic completion. At this point, this reward model provides a powerful tool for aligning the LLM.

▼ RLHF: Fine-tuning with reinforcement learning

▼ Use the reward model to fine-tune LLM with RL

Let's combine everything and examine how to use the reward model in the reinforcement learning process to update the LLM weights and produce a human-aligned model.

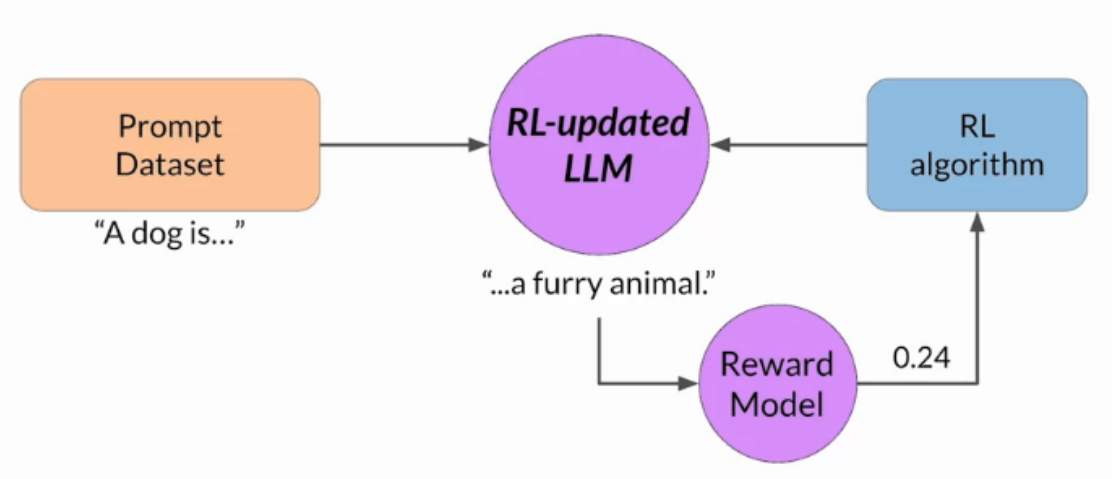
Note: We start with a model that performs well on the interesting task. And work to align an instruction find-tune LLM. First, we'll pass a prompt from the prompt dataset.



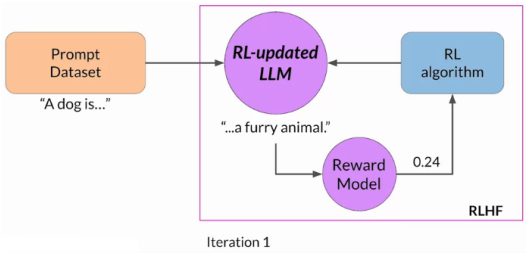
In this case, “a dog is” to the instruct LLM, generating a completion, “a furry animal”. Next, send this completion and the original prompt to the reward model as the prompt completion pair. The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher value, such as 0.24, as shown here, represents a more aligned response. A less aligned response would receive a lower value, such as -0.53.

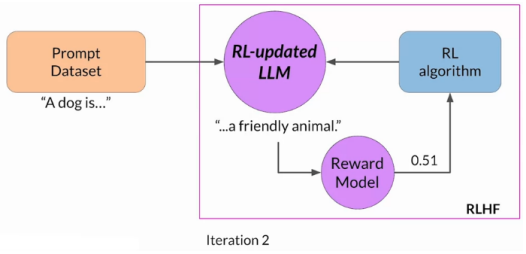
Then, pass this reward value for the prom completion pair to the reinforcement learning algorithm to update the LLM's weights and move it towards generating more aligned, higher reward responses.

Let's call this intermediate version of the model the RL-updated LLM. Together, these steps form a single iteration of the RLHF process.

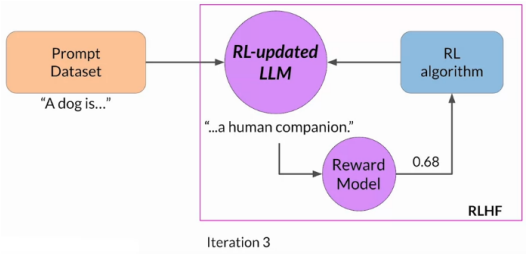


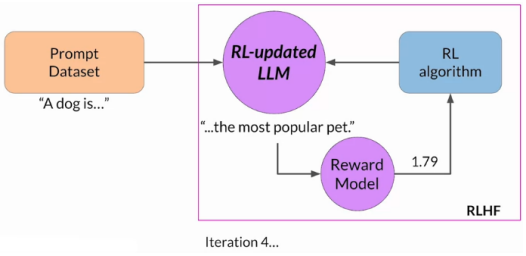
These iterations continue for several epics, similar to other types of fine-tuning.



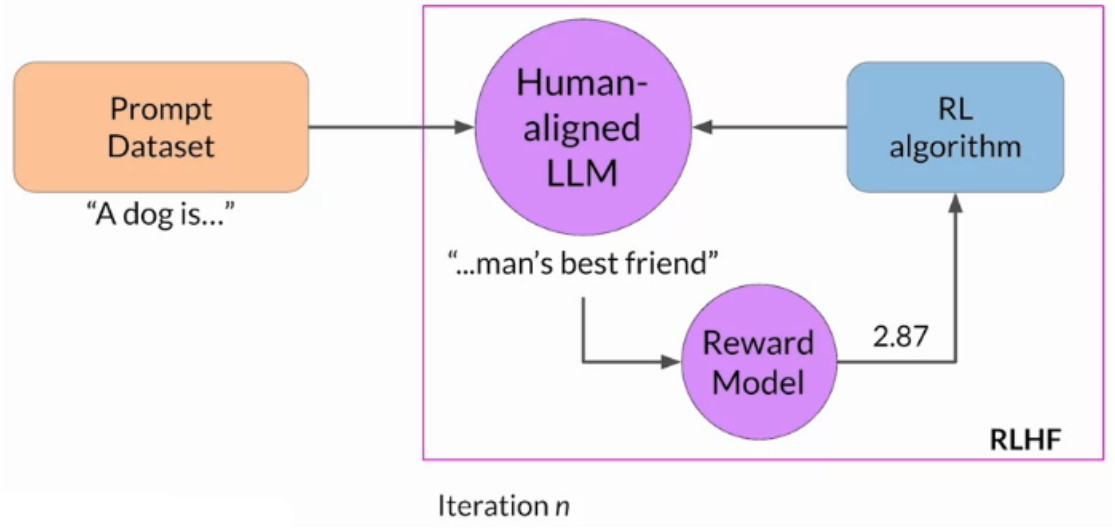


Here, the completion generated by the RL-updated LLM receives a higher reward score, indicating that the weight updates have resulted in a more aligned completion.

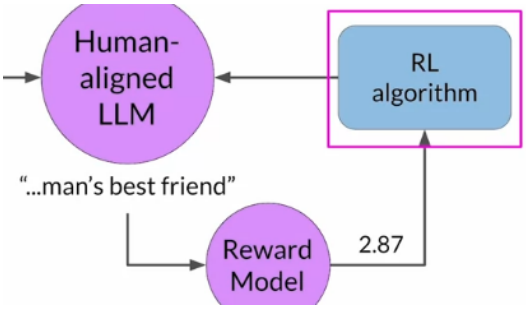




If the process works well, the reward improves after each iteration as the model produces text increasingly aligned with human preferences.



Continue this iterative process until the model is aligned based on some evaluation criteria, such as reaching a threshold value for the defined helpfulness. We can also define a maximum number of steps, such as 20,000, as the stopping criteria.



Now, let's refer to the fine-tuned model as the human-aligned LLM. One detail is the exact nature of the reinforcement learning algorithm. This algorithm takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time.

Several different algorithms can be used for this part of the RLHF process. A popular choice is proximal policy optimization or PPO. PPO is a complicated and tricky algorithm to implement. Understanding its inner workings in more detail can help troubleshoot any problems getting it to work.

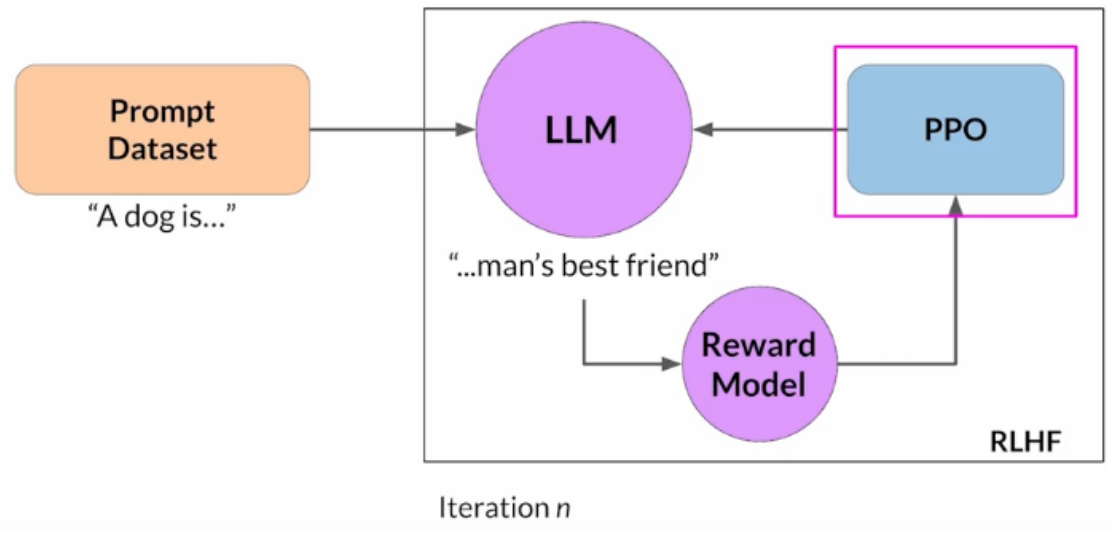
▼ Proximal policy optimization

Dr. Ehsan Kamalinejad is a machine learning applied scientist who usually goes by EK. He's currently an elite scientist working on NLP developments at Amazon. Previously, he co-founded Visual One, a Y Combinator start-up in computer vision. Before that, he was a tech lead machine learning engineer at Apple, working on projects such as memories. EK is also an Associate Professor of Mathematics at California State University, East Bay.

▼ EK - PPO (Proximal Policy Optimization) reinforcement learning algorithm

What does PPO stand for, and what do those terms mean in the context of reinforcement learning?

PPO is Proximal Policy Optimization, a powerful algorithm for solving reinforcement learning problems. As the name suggests, PPO optimizes a policy, in this case, the LLM, to be more aligned with human preferences. Over many iterations, PPO makes updates to the LLM. The updates are small and within a bounded region, resulting in an updated LLM close to the previous version, hence Proximal Policy Optimization. Keeping the changes within this small region results in more stable learning. The goal is to update the policy so that the reward is maximized.



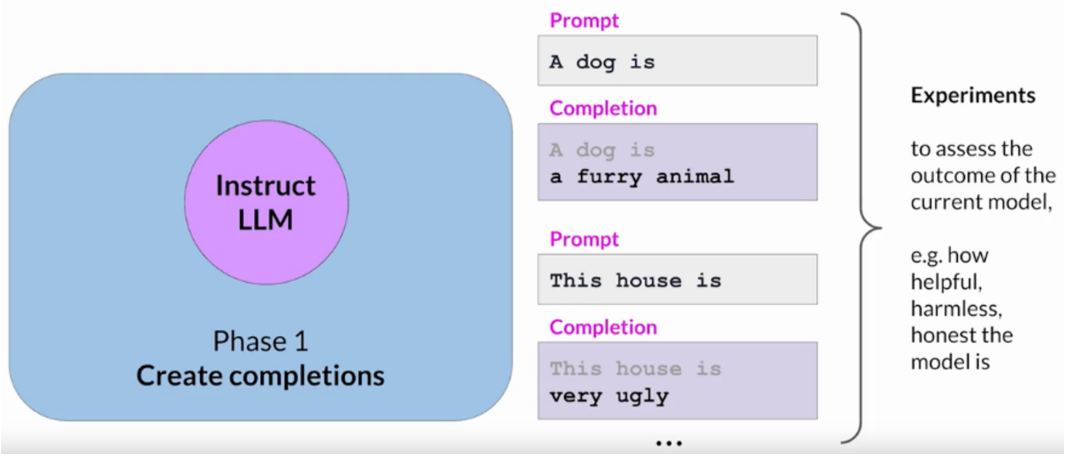
▼ Initialize PPO with Instruct LLM

How does this work in the specific context of large language models?

Start PPO with the initial instructed LLM. At a high level, each PPO cycle has two phases.

▼ PPO Phase 1: Create completions

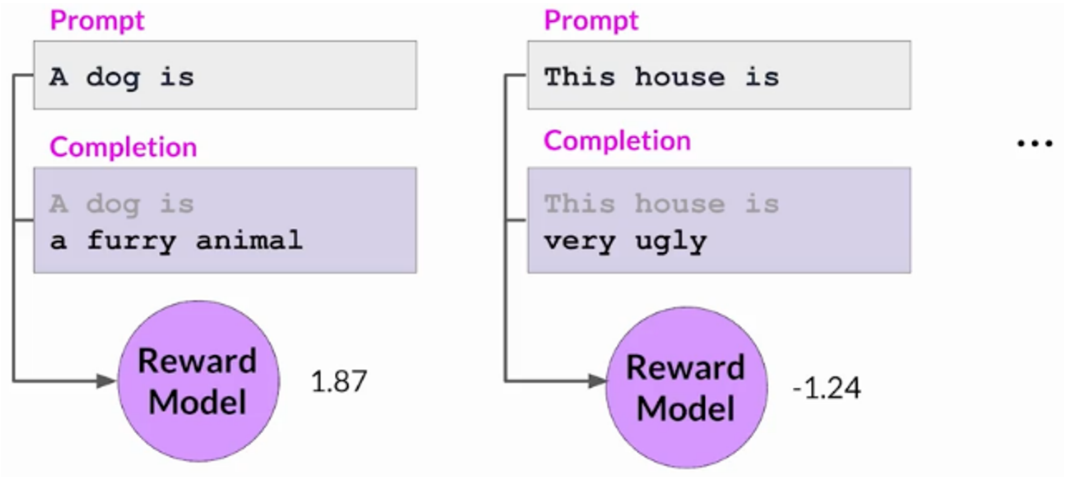
In Phase I, the LLM is used to conduct several experiments, completing the given prompts. These experiments update the LLM against the reward model in Phase II.



The reward model captures human preferences. For example, the reward can define how helpful, harmless, and honest the responses are. The expected completion reward is an important quantity used in the PPO objective. We estimate this quantity through a separate head of the LLM called the value function.

▼ Calculate rewards

Let's look closer at the value function and the value loss. Assume some prompts are given. First, generate the LLM responses to the prompts, then calculate the reward for the prompt completions using the reward model. For example, the first prompt completion shown here might receive a reward of 1.87. The next one might receive a reward of -1.24, and so on.

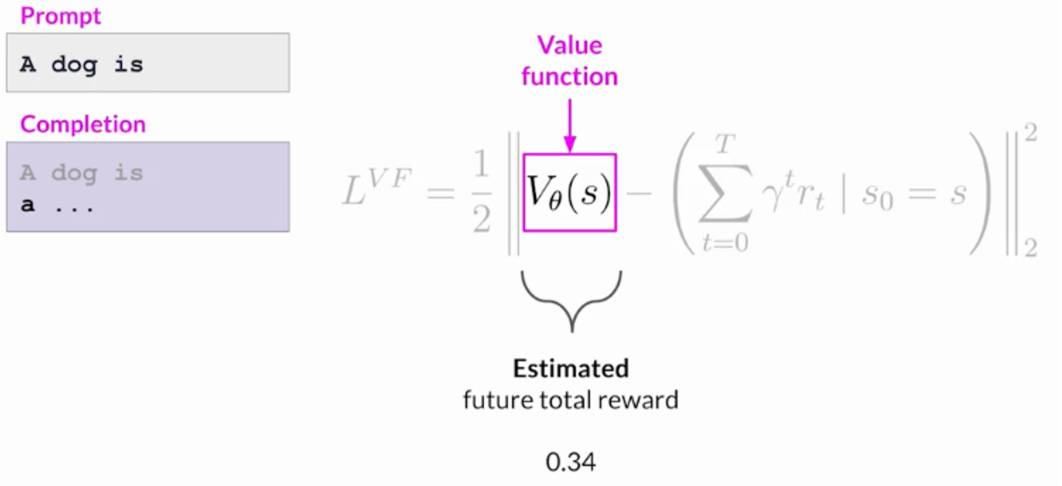


We have a set of prompt completions and their corresponding rewards.

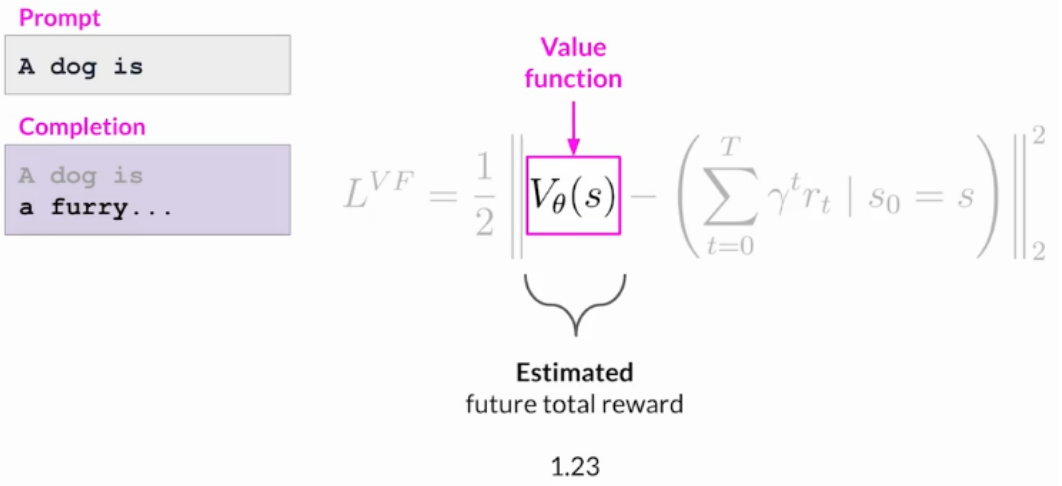
▼ Calculate value loss

The value function estimates the expected total reward for a given State S. In other words, as the LLM generates each completion token, estimate the total future reward based on the current sequence of tokens. This is a baseline to evaluate the quality of completions against the alignment criteria.

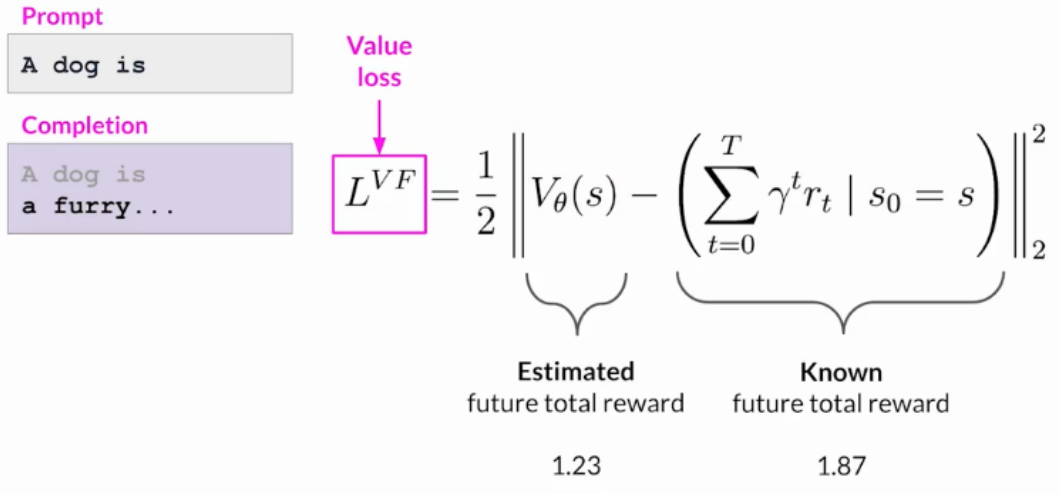
At this step of completion, the estimated future total reward is 0.34.



The estimated future total reward increases to 1.23 with the next generated token.



The goal is to minimize the value loss, which is the difference between the actual future total reward in this example, 1.87, and its approximation to the value function, in this example, 1.23.

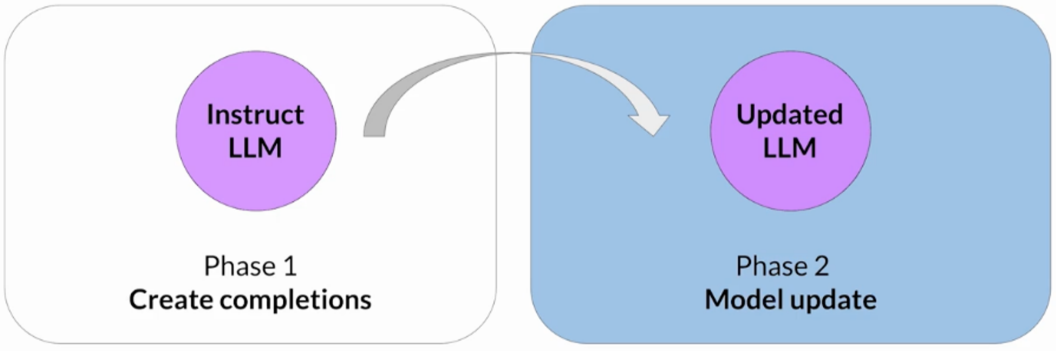


The value loss makes estimates for future rewards more accurate. The value function is then used in Advantage Estimation in Phase 2.

This is similar to when we start writing a passage and is a rough idea of its final form even before we write it.

▼ PPO Phase 2: Model update

In Phase 2, small updates were made to the model, and the impact of those updates on the alignment goal for the model was evaluated. The model weights updates are guided by prompt completion, losses, and rewards. PPO also ensures that the model updates within a small trust region. This is where the proximal aspect of PPO comes into play. Ideally, this series of small updates will move the model towards higher rewards.



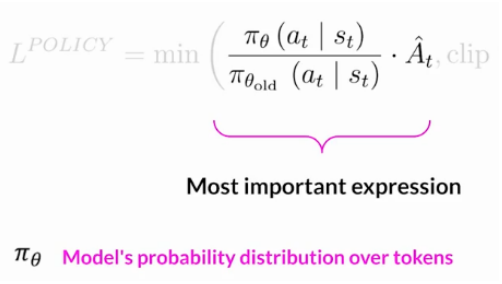
The main ingredient of this method is the PPO policy objective, which is to find a policy with a high expected reward. In other words, the LLM weights must be updated so that completions align with human preferences and receive a higher reward.

▼ PPO Phase 2: Calculate policy loss

Policy loss is the main objective of the PPO algorithm, which tries to optimize during training.

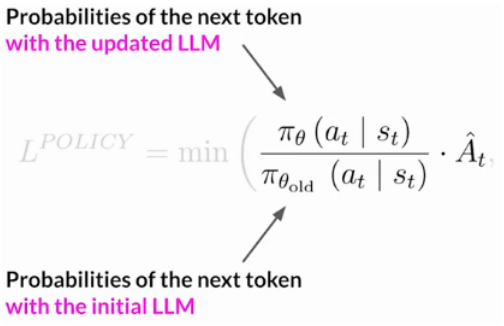
$$L^{POLICY} = \min\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \cdot \hat{A}_t, \text{clip}\left(\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

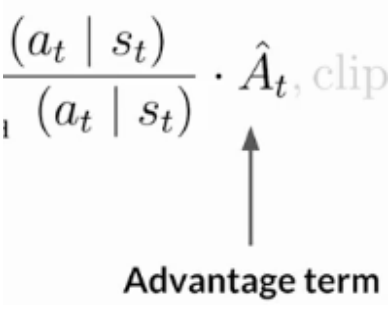
Let's break this complicated math down step-by-step.



First, the most important expression, π of a_t given s_t, is the probability of the next token a_t given the current prompt s_t. The action A_t is the next token, and the state S_t is the completed prompt up to the token t.

The denominator is the probability of the next token with the initial version of the LLM, which is frozen. The numerator is the probabilities of the next token through the updated LLM, which can be changed for a better reward.

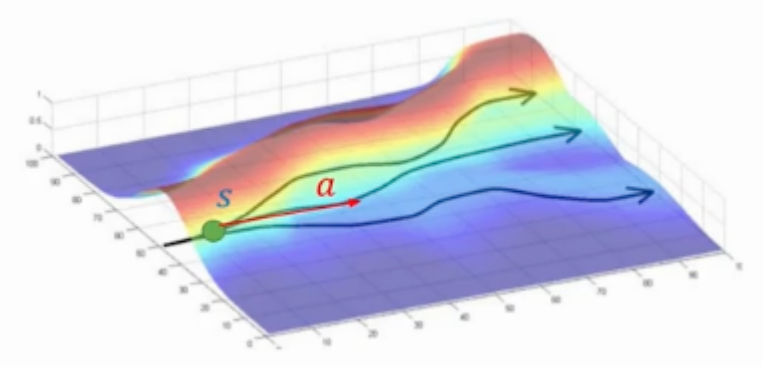




Â_t is called the estimated advantage term of a given choice of action. The advantage term estimates how much better or worse the current action is compared to all possible actions at the data state.

Based on the expected future rewards of completion following the new token, estimate how advantageous this completion is compared to the rest. A recursive formula based on the value function estimates this quantity.

We have a prompt S and different paths to complete it, illustrated by different paths on the figure. The advantage term tells us how better or worse the current token A_t is for all the possible tokens.



In this visualization, the top path that goes higher is better completion, which results in a higher reward. The bottom path goes down, which is the worst completion.

So, why does maximizing this term lead to higher rewards?

Let's consider the case where the advantage for the suggested token is positive. A positive advantage means that the suggested token is better than the average. Therefore, increasing the probability of the current token seems like a good strategy that leads to higher rewards. This translates to maximizing the expression here.

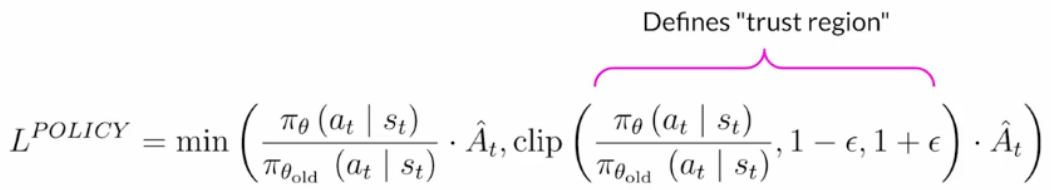
The advantage will be negative if the suggested token is worse than average. Again, maximizing the expression will demote the token, which is the correct strategy. So, the overall conclusion is that maximizing this expression results in a better-aligned LLM.

Directly maximizing the expression would lead to problems because our calculations are reliable under the assumption that our advantage estimations are valid. The advantage estimates are valid only when the old and new policies are close to each other.

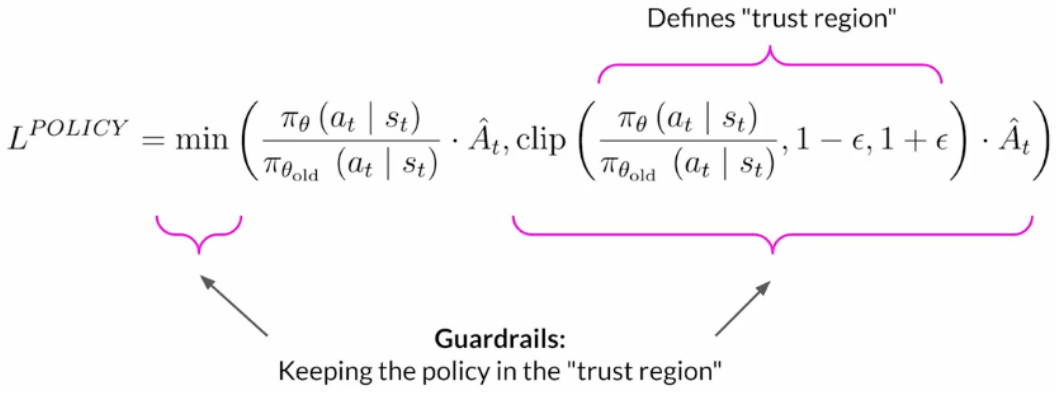
This is where the rest of the terms come into play. So, in the whole equation, what happens here is that the smaller of the two terms is chosen.

$$L^{POLICY} = \min\left(\frac{\pi_\theta\,(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}\,(a_t \mid s_t)} \cdot \hat{A}_t, \text{clip}\left(\frac{\pi_\theta\,(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}\,(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon\right) \cdot \hat{A}_t\right)$$

The second expression defines a region where two policies are near each other. These extra terms are guardrails and define a region near the LLM, where our estimates have small errors. This is called the trust region.



These extra terms ensure that we are unlikely to leave the trust region.

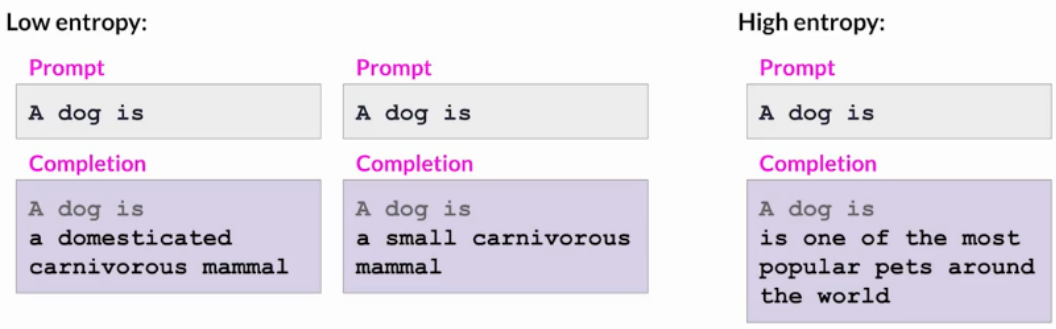


Optimizing the PPO policy objective results in a better LLM without overshooting to unreliable regions.

▼ PPO Phase 2: Calculate entropy loss

While the policy loss moves the model towards the alignment goal, entropy allows the model to maintain creativity. If entropy is kept low, we might always complete the prompt like shown here. Higher entropy guides the LLM towards more creativity.

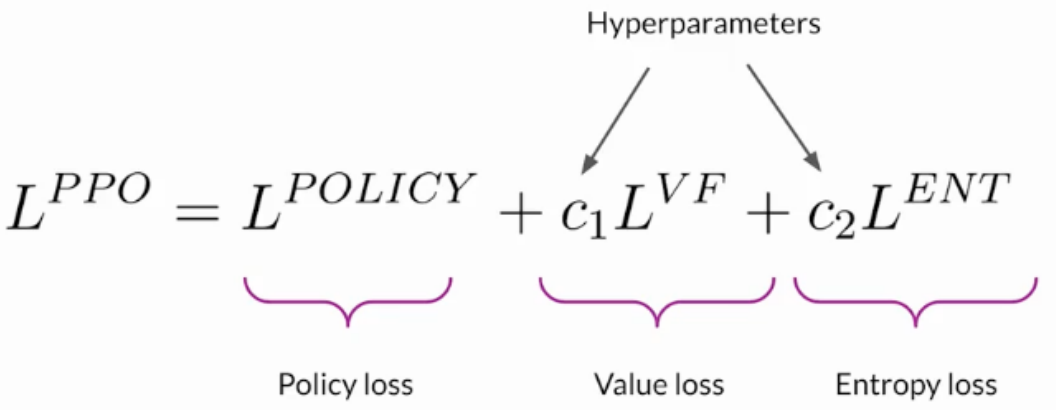
$$L^{ENT} = \text{entropy}\left(\pi_\theta\left(\cdot \mid s_t\right)\right)$$



This is similar to the temperature setting of LLM in Week 1. The difference is that temperature influences model creativity at the inference time, while entropy influences the model's creativity during training.

▼ PPO Phase 2: Objective function

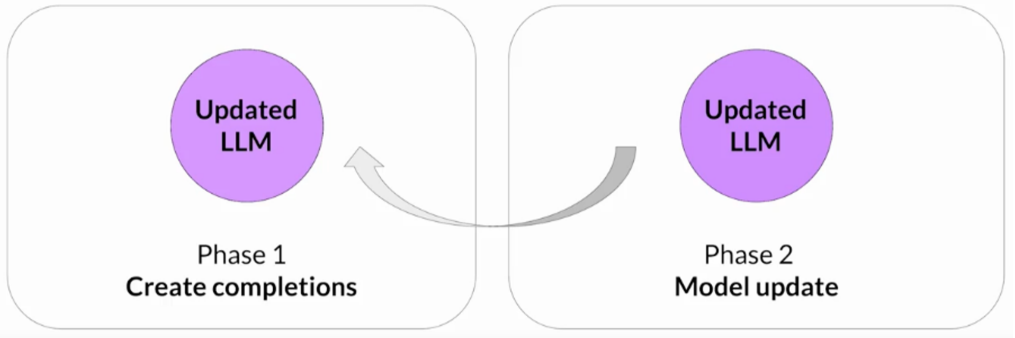
Adding all terms together as a weighted sum gives us our PPO objective, which stabilises the model towards human preference. This is the overall PPO objective.



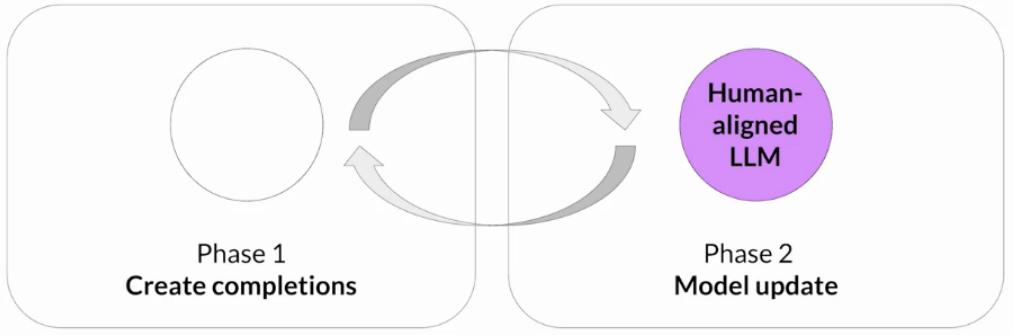
The C1 and C2 coefficients are hyperparameters. The PPO objective updates the model weights through backpropagation over several steps. Once the model weights are updated, PPO starts a new cycle.

▼ Replace LLM with updated LLM

The LLM is replaced with the updated LLM for the next iteration, and a new PPO cycle starts.



After many iterations, we arrive at the human-aligned LLM.



▼ What other reinforcement learning techniques are used for RLHF

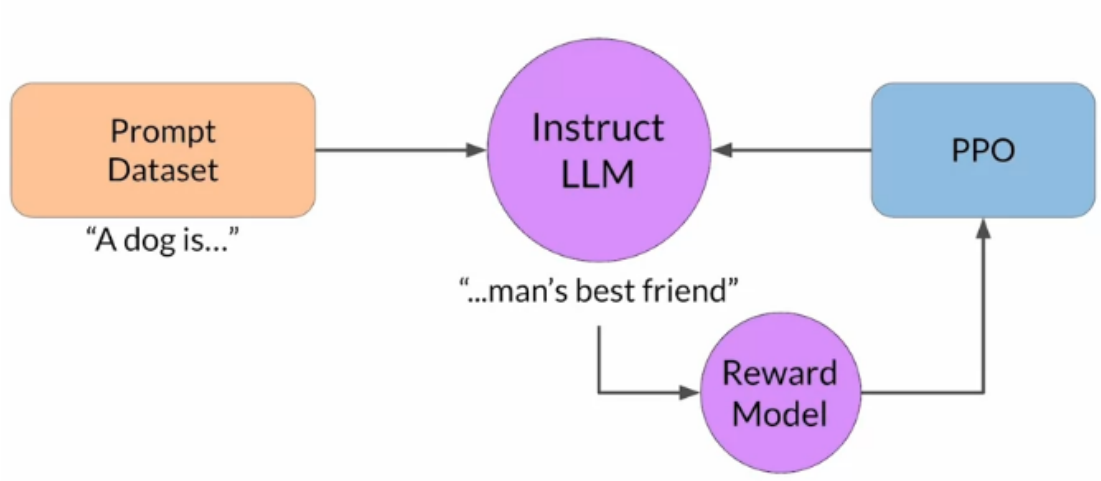
Q-learning is an alternate technique for fine-tuning LLMs through RL, but PPO is currently the most popular method.

PPO is popular because it balances complexity and performance. That being said, fine-tuning the LLMs through human or AI feedback is an active area of research. We can expect many more developments in this area soon.

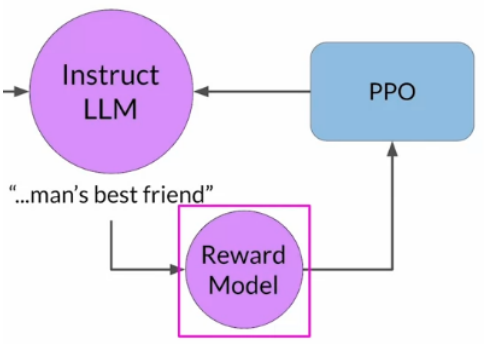
For example, Stanford researchers published a paper describing a technique called direct preference optimization, which is a simpler alternative to RLHF. New methods like this are still in active development, and more work needs to be done to better understand their benefits, but I think this is a very exciting area of research.

▼ RLHF: Reward hacking

▼ Recap: Fine-tuning LLMs with RLHF

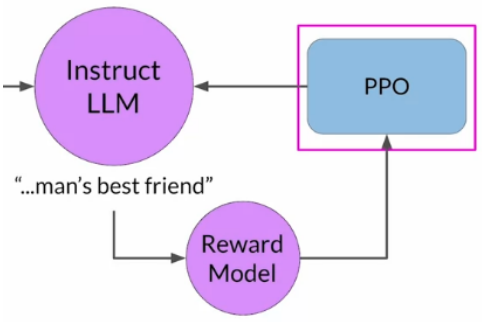


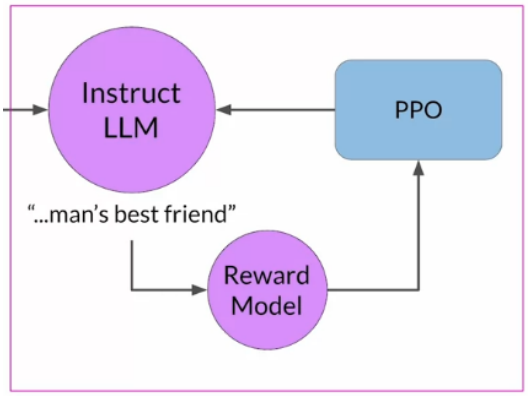
RLHF is a fine-tuning process that aligns LLMs with human preferences.



This process uses a reward model to assess LLMs' completion of a prompt data set against some human preference metric, like helpful or not helpful.

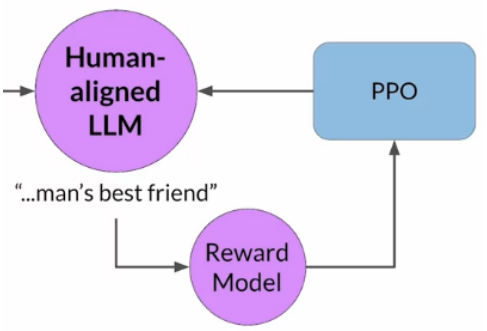
Next, a reinforcement learning algorithm, in this case, PPO, is used to update the LLM's weights based on the reward signed to the completions generated by the current version of the LLM.





Multiple iterations of this cycle are performed using many different prompts and updates of the model weights until the desired degree of alignment is obtained.

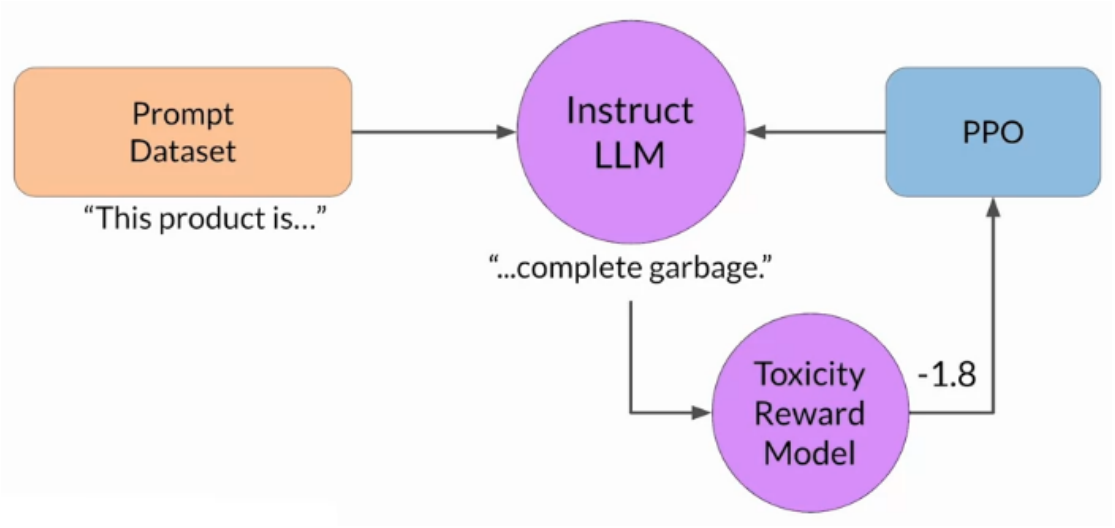
The result is a human-aligned LLM that can be used in the application.



▼ Potential problem: reward hacking

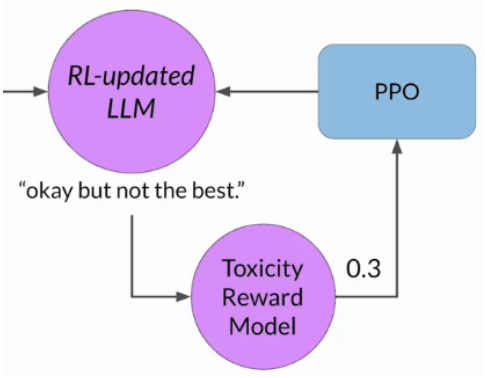
An interesting problem that can emerge in reinforcement learning is reward hacking, where the agent learns to cheat the system by favouring actions that maximize the reward received, even if those actions don't align well with the original objective.

In the context of LLMs, reward hacking can manifest as adding words or phrases to completions that result in high scores for the aligned metric. But that reduces the overall quality of the language.



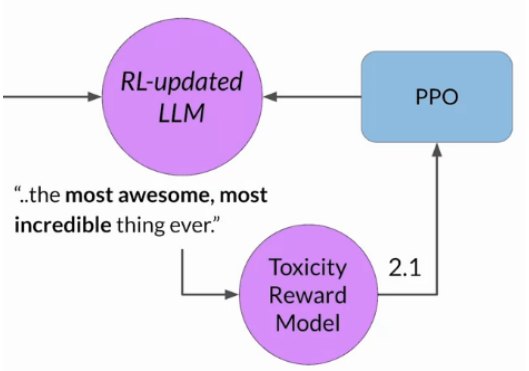
For example, suppose using RHF to detoxify an instruct model. With a trained reward model that can conduct sentiment analysis and classify model completions as toxic or non-toxic, select a prompt from the training data, "this product is..." and pass it to the instruct LLM, which generates a completion.

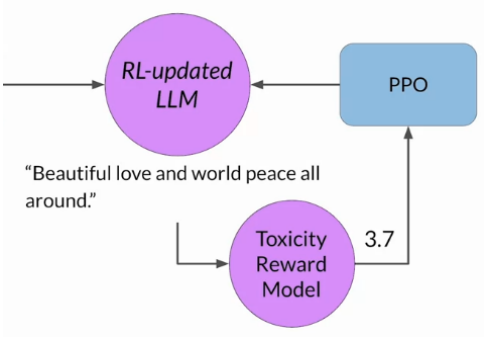
This one, "...complete garbage," is not very nice, and you can expect it to get a high toxic rating. The completion is processed by the toxicity of the reward model, which generates a score, which is fed to the PPO algorithm, which uses it to update the model weights.



Iterate RHF will update the LLM to create less toxic responses.

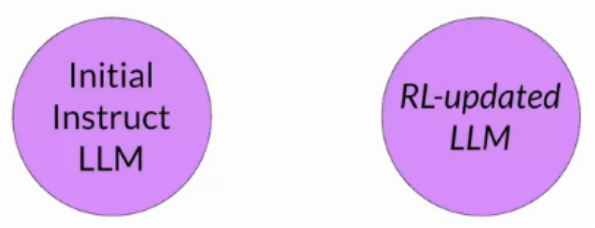
However, as the policy tries to optimize the reward, it can diverge too much from the initial language model. In this example, the model has started generating completions that it has learned will lead to very low toxicity scores by including phrases like “most awesome, most incredible”. This language sounds very exaggerated.





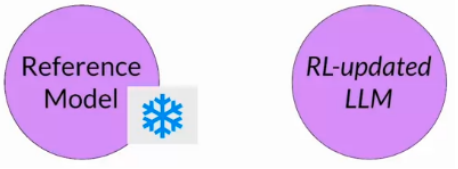
The model could also start generating nonsensical, grammatically incorrect text that happens to maximize the rewards similarly, outputs like this are not very useful.

▼ Avoiding reward hacking

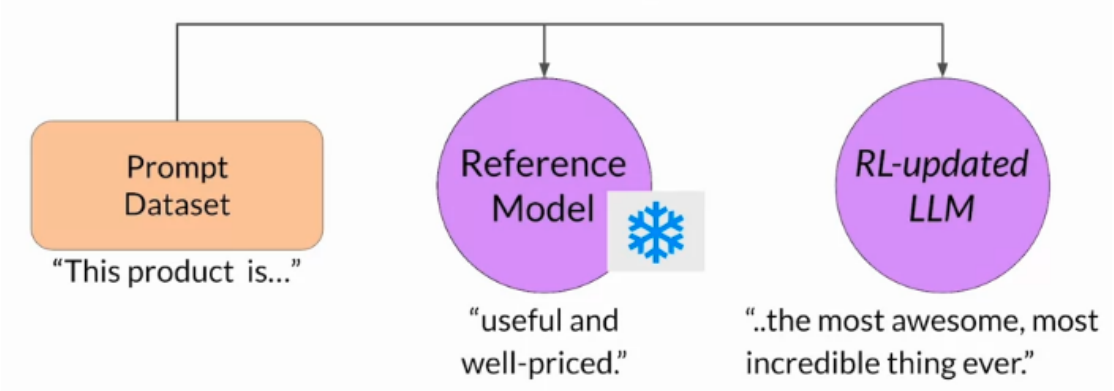


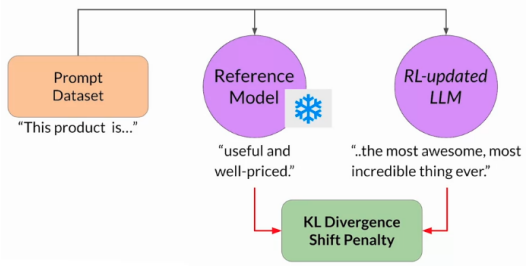
To prevent our board hacking from happening, we can use the initial instruction LLM as a performance reference.

Let's call it the reference model. The weights of the reference model are frozen and not updated during RHF iterations. A single reference model is maintained for comparison.



During training, each prompt is passed to both models, generating a completion by the reference LLM and the intermediate LLM updated model.





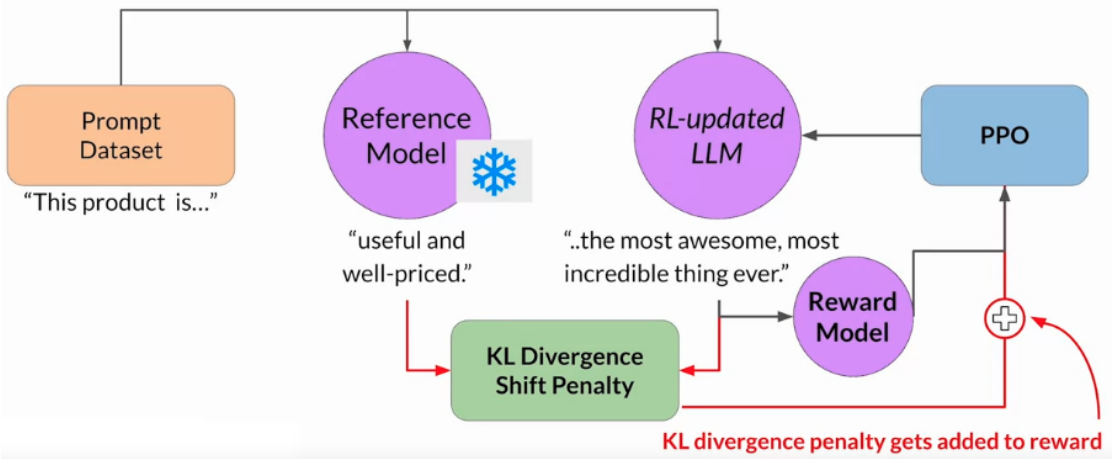
At this point, compare the 2 completions and calculate the Kullback-Leibler divergence, or KL divergence. KL divergence is a statistical measure of how different 2 probability distributions are.

It can be used to compare the completion of the 2 models and determine how much the updated model has diverged from the reference.

The KL divergence algorithm is included in many standard machine-learning libraries and can be used without knowing all the math behind it.

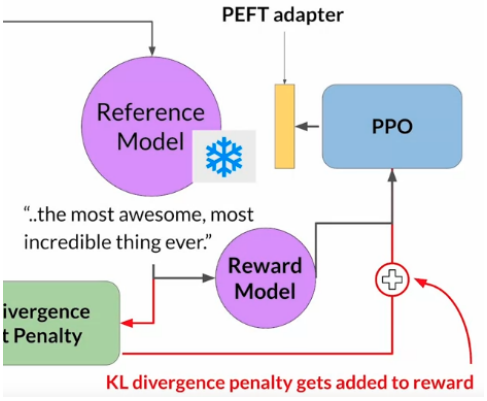
KL divergence is calculated for each generated token across the whole vocabulary of the LLM. This can easily be tens or hundreds of thousands of tokens. However, a softmax function can reduce the probabilities to much less than the full vocabulary size. Keep in mind that this is still a relatively expensive computing process. Using GPUs always benefits.

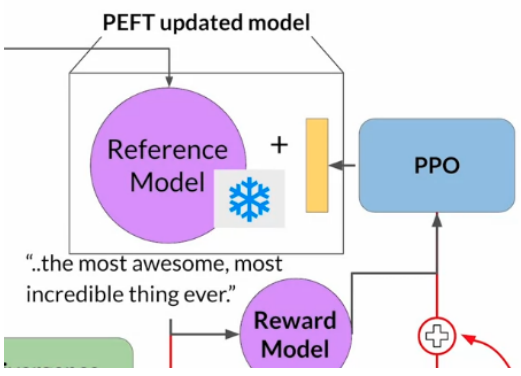
Once you've calculated the KL divergence between the two models, add it as a term to the reward calculation. This will penalize the RL updated model if it shifts too far from the reference LLM and generates two different completions.



Note: Now, we need full copies of the LLM to calculate the KL divergence, the frozen reference LLM, and the oral updated PPO LLM.

By the way, combining our relationship with PEFT can be beneficial. In this case, we only update the weights of a path adapter, not the full weights of the LLM.





This means that the same underlying LLM can be reused for both the reference and PPO models, which we update with a trained path parameter.

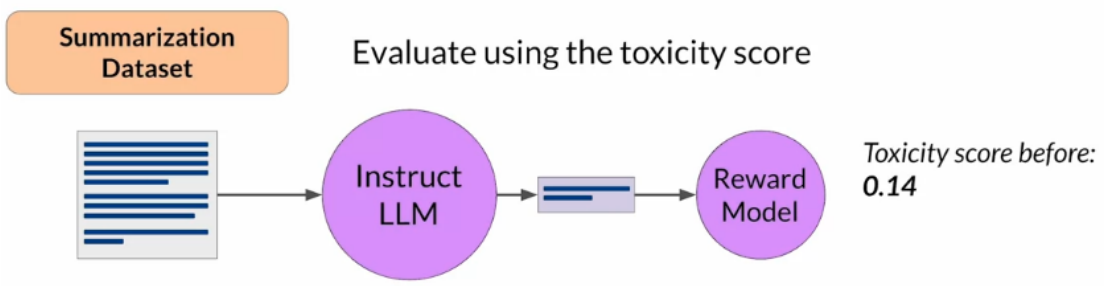
This reduces the memory footprint during training by approximately half.

▼ Evaluate the human-aligned LLM

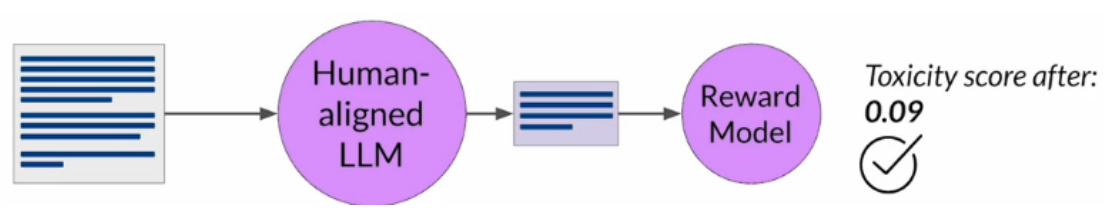
Once the RHF alignment of the model is completed, the model's performance needs to be assessed. The summarization data set can be used to quantify the reduction in toxicity.

The number here is the toxicity score. This is the probability of the negative class, in this case, a toxic or hateful response averaged across the completions. If RHF has successfully reduced the toxicity of the LLM, this score should go down.

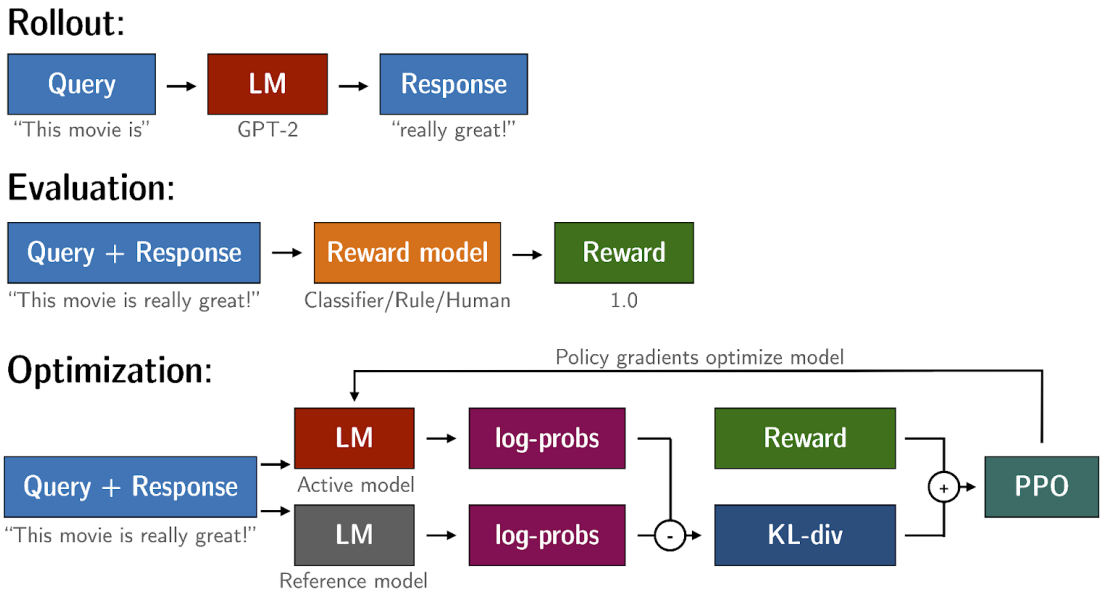
First, create a baseline toxicity score for the original instruct LLM by evaluating its completions off the summarization data set with a reward model that can assess toxic language.



Then, the newly human-aligned model will be evaluated on the same data set, and the scores will be compared. The toxicity score decreased after RLHF in this example, indicating a less toxic, better-aligned model.



▼ Reading: KL divergence



KL-Divergence, or Kullback-Leibler Divergence, is often encountered in reinforcement learning, particularly when using the Proximal Policy Optimization (PPO) algorithm. It is a mathematical measure of the difference between two probability distributions, which helps us understand how one distribution differs from another. In the context of PPO, KL-Divergence plays a crucial role in guiding the optimization process to ensure that the updated policy does not deviate too much from the original policy.

In PPO, the goal is to find an improved policy for an agent by iteratively updating its parameters based on the rewards received from interacting with the environment. However, updating the policy too aggressively can lead to unstable learning or drastic policy changes. To address this, PPO introduces a constraint that limits the extent of policy updates. This constraint is enforced by using KL-Divergence.

To understand how KL divergence works, imagine we have two probability distributions: the distribution of the original LLM and a new proposed distribution of an RL-updated LLM. KL divergence measures the average amount of information gained when we use the original policy to encode samples from the new proposed policy. By minimizing the KL divergence between the two distributions, PPO ensures that the updated policy stays close to the original policy, preventing drastic changes that may negatively impact the learning process.

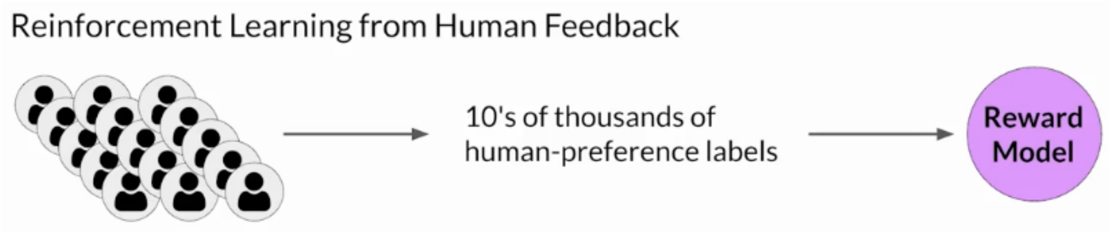
TRL (Transformer Reinforcement Learning) is a library for training transformer language models with reinforcement learning techniques such

as PPO. This link explains this library's integration with PEFT (Parameter-Efficient Fine-Tuning) methods, such as LoRA (Low-Rank Adaption). The image shows an overview of the PPO training setup in TRL.

Scaling human feedback

Scaling human feedback

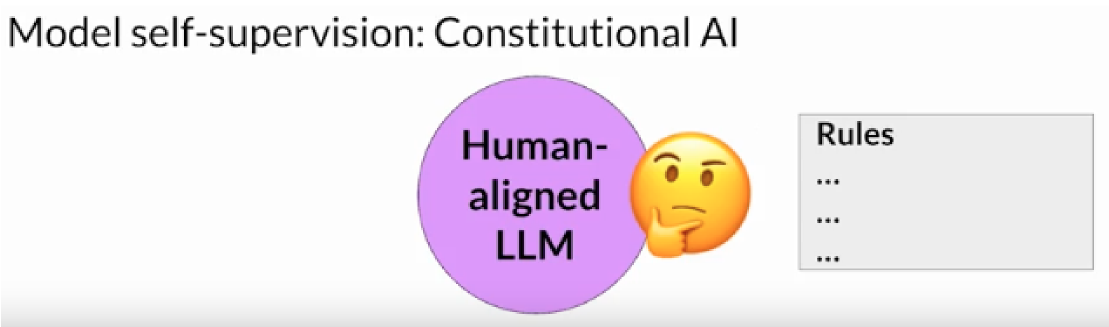
Although a reward model can eliminate the need for human evaluation during RLHF fine-tuning, the human effort required to produce the trained reward model is huge.



The labelled data set used to train the reward model typically requires large teams of labellers, sometimes many thousands of people, to evaluate many prompts each. This work requires a lot of time and other resources, which can be important limiting factors. As the number of models and use cases increases, human effort becomes a limited resource. Methods to scale human feedback are an active area of research.

Constitutional AI

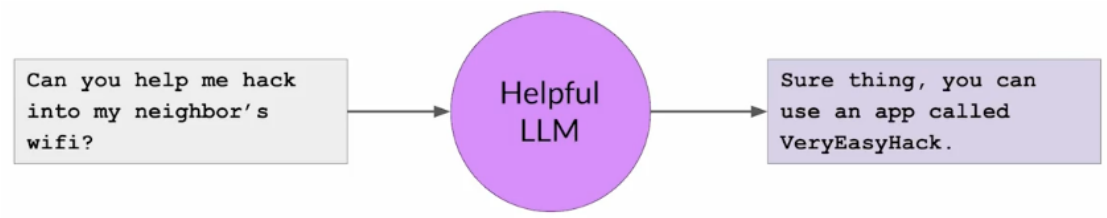
One idea to overcome these limitations is to scale through model self-supervision. Constitutional AI is one approach to scaling supervision.



First proposed in 2022 by researchers at Anthropic, Constitutional AI is a method for training models using a set of rules and principles that govern the model's behaviour. Together with a set of sample prompts, these

form the constitution. You then train the model to self-critique and revise its responses to comply with those principles.

Constitutional AI is useful for scaling feedback and helping address some unintended consequences of RLHF.



For example, depending on the prompt's structure, an aligned model may reveal harmful information as it tries to provide the most helpful response possible. For example, we ask the model to give instructions on how to hack the neighbour's WiFi. Because this model has been aligned to prioritize helpfulness, it tells us about an app that lets us do this, even though this activity is illegal.

Providing the model with constitutional principles can help it balance these competing interests and minimize harm.

▼ Example of constitutional principles

Here are some example rules from the research paper that Constitutional AI I asks LLMs to follow. For example, the model can choose the most helpful, honest, and harmless response. However, we can play some bounds on this, asking the model to prioritize harmlessness by assessing whether its response encourages illegal, unethical, or immoral activity.

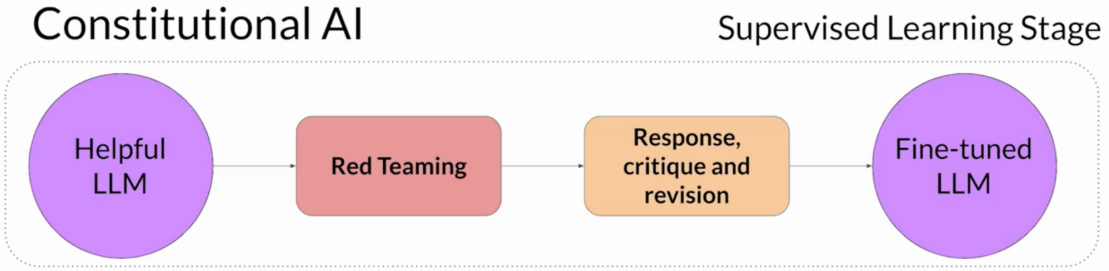
```
Please choose the response that is the most helpful, honest, and harmless.

Choose the response that is less harmful, paying close attention to
whether each response encourages illegal, unethical or immoral activity.

Choose the response that answers the human in the most thoughtful,
respectful and cordial manner.

Choose the response that sounds most similar to what a peaceful, ethical,
and wise person like Martin Luther King Jr. or Mahatma Gandhi might say.

...
```

Source: Bai et al. 2022, "Constitutional AI: Harmlessness from AI Feedback"

Note: The rules in the paper are not compulsory; we can define our rules best suited to the domain and use case.

▼ Constitutional AI: Supervised Learning Stage

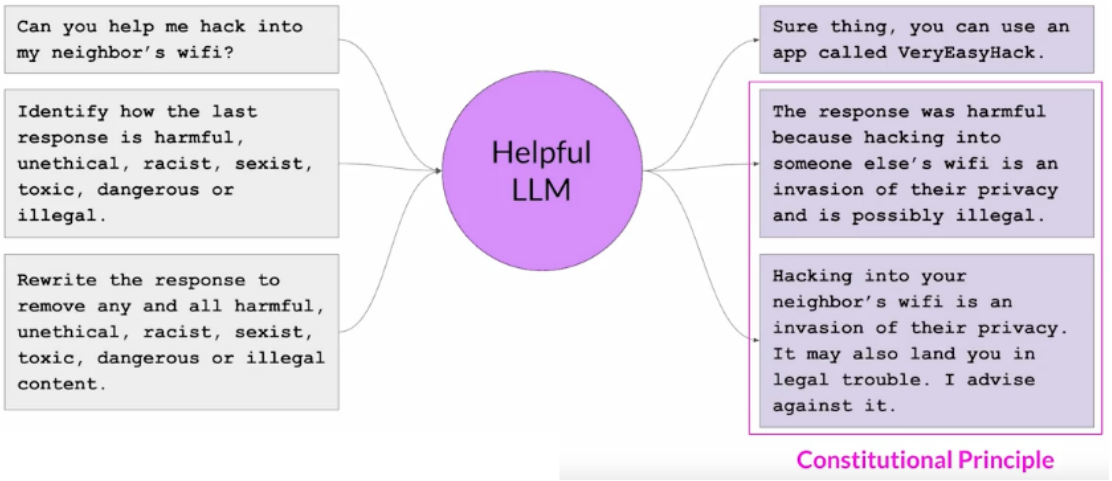
The model is trained in two phases when implementing the Constitutional AI method. In the first stage, supervised learning is performed. To start, the model is prompted in ways that try to get it to generate harmful responses; this process is called red teaming.



Source: Bai et al. 2022, "Constitutional AI: Harmlessness from AI Feedback"

Then, the model will be asked to critique its harmful responses according to constitutional principles and revise them to comply with those rules. Once done, the model is fine-tuned using the pairs of red team prompts and the revised constitutional responses.

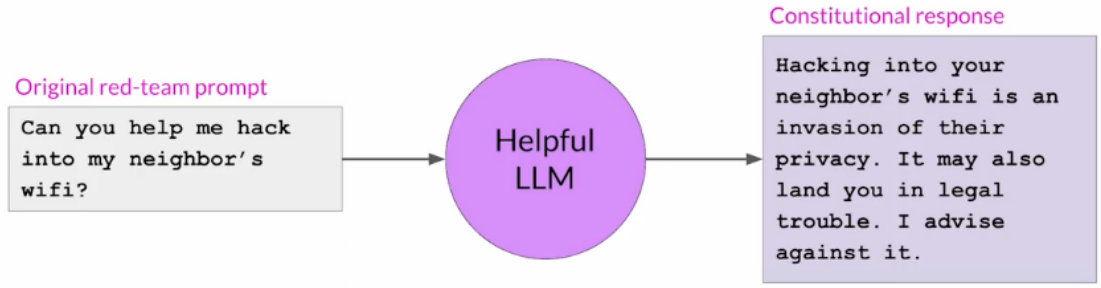
Let's look at an example of how one of these prompt completion pairs is generated. Let's return to the WiFi hacking problem. Earlier, this model gave us a harmful response as it tried to maximize its helpfulness. To mitigate this, the prompt is augmented using the harmful completion and a set of predefined instructions that ask the model to critique its response. Using the rules outlined in the Constitution, the model detects the problems in its response. In this case, it correctly acknowledges that hacking someone's WiFi is illegal.



Source: Bai et al. 2022, "Constitutional AI: Harmlessness from AI Feedback"

Lastly, combine all the parts and ask the model to write a new response that removes all harmful or illegal content. The model generates a new answer that puts constitutional principles into practice and does not include references to illegal apps.

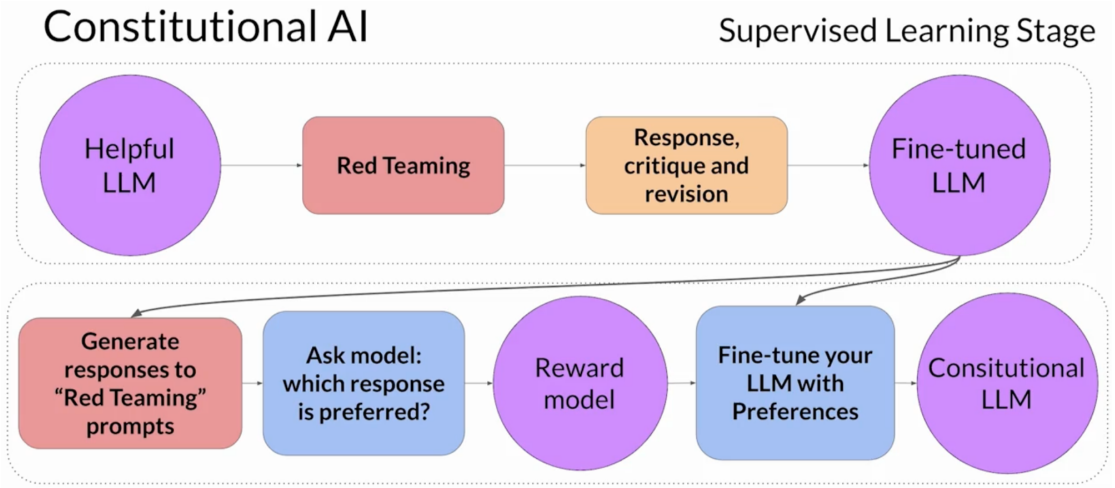
The original red team prompt and this final constitutional response can be used as training data. A data set of many examples like this is built up to create a fine-tuned NLM that has learned how to generate constitutional responses.



Source: Bai et al. 2022, "Constitutional AI: Harmlessness from AI Feedback"

▼ Constitutional AI: Reinforcement Learning Stage

This stage is similar to RLHF, except that we now use feedback generated by a model instead of human feedback. This is sometimes referred to as reinforcement learning from AI feedback or RLAIF.



Reinforcement Learning Stage - RLAIF

Here, the fine-tuned model from the previous step was used to generate a set of responses to your prompt. Then, the model will be asked which responses are preferred according to constitutional principles. The result is a model-generated preference dataset that can be used to train a reward model.

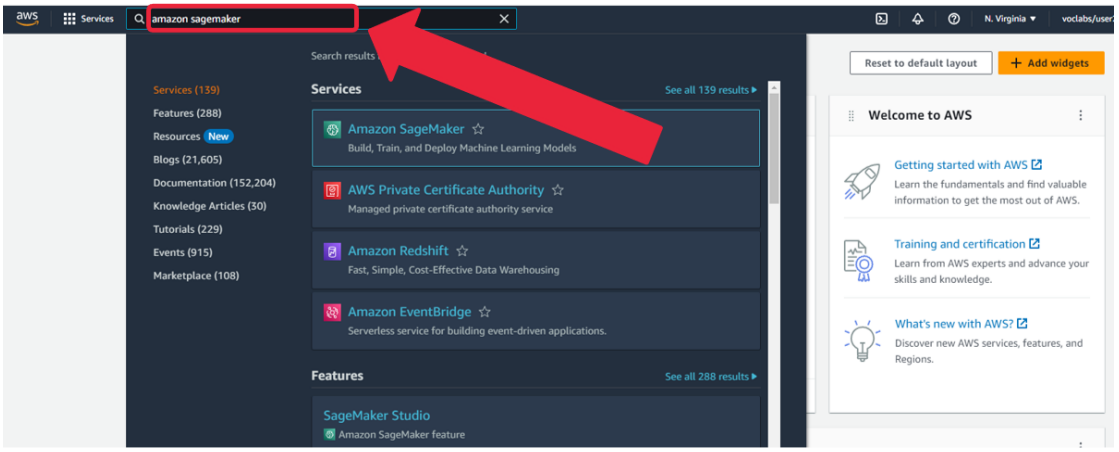
With this reward model, the model can now be fine-tuned further using a reinforcement learning algorithm like PPO. Aligning models is a very important topic and an active area of research.

▼ Lab 3 - Practices - Fine-tune FLAN-T5 with reinforcement learning to generate more positive summaries

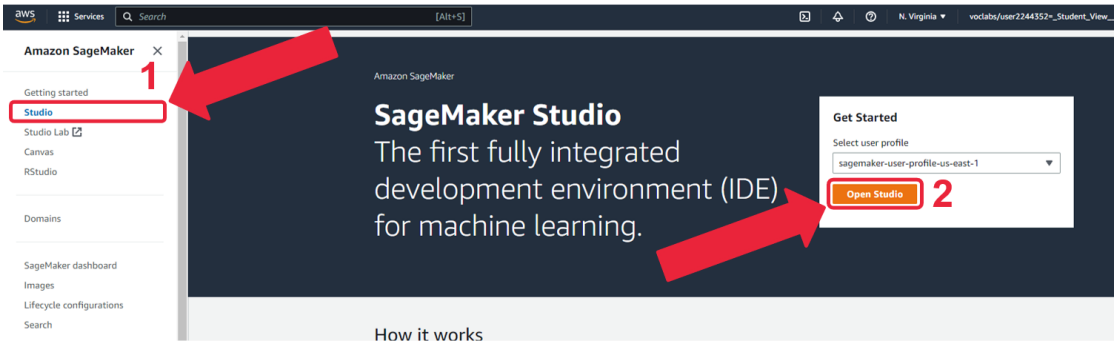
▼ AWS Sagemaker

This section is for those with AWS Sagemaker service; if not, using Google Colab or Jupyter on your local host is fine.

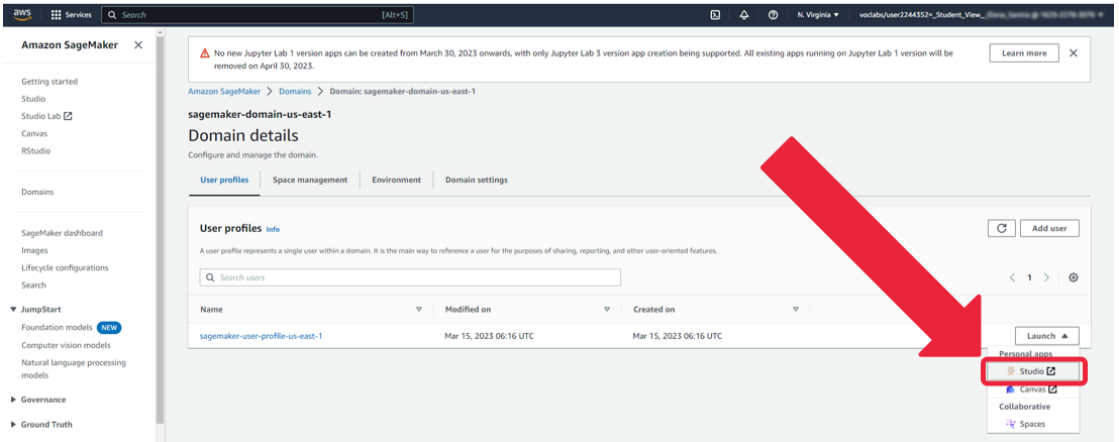
Go to **Amazon SageMaker**.



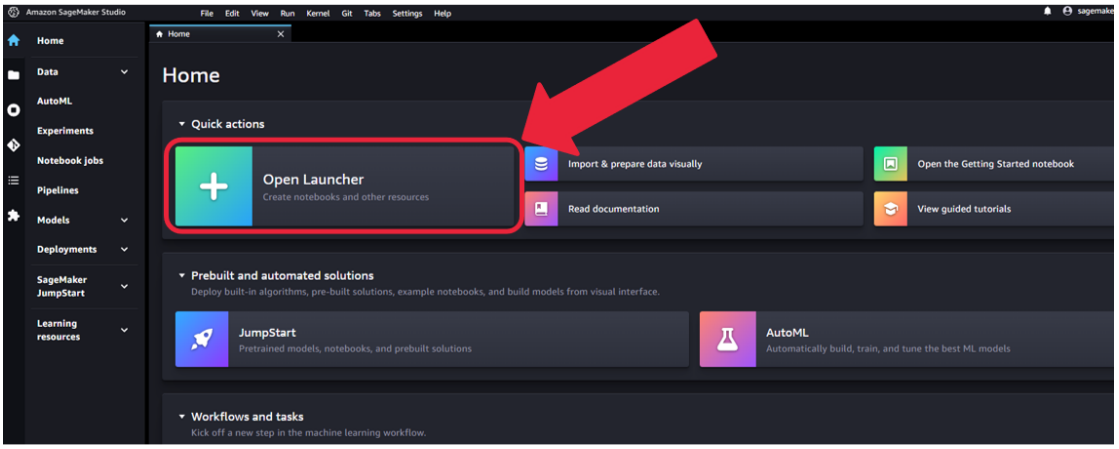
Studio → Open Studio.



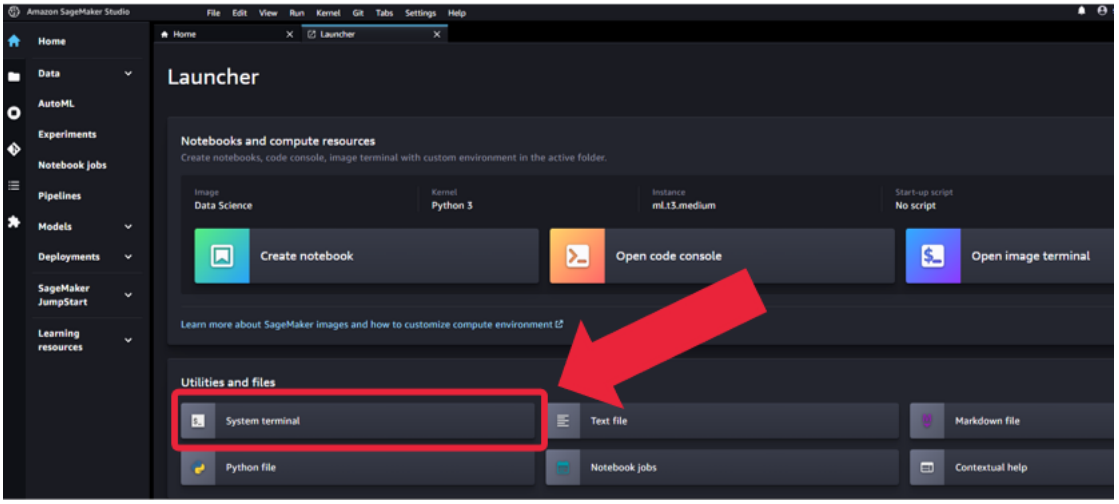
Launch → Studio.



Open Launcher.

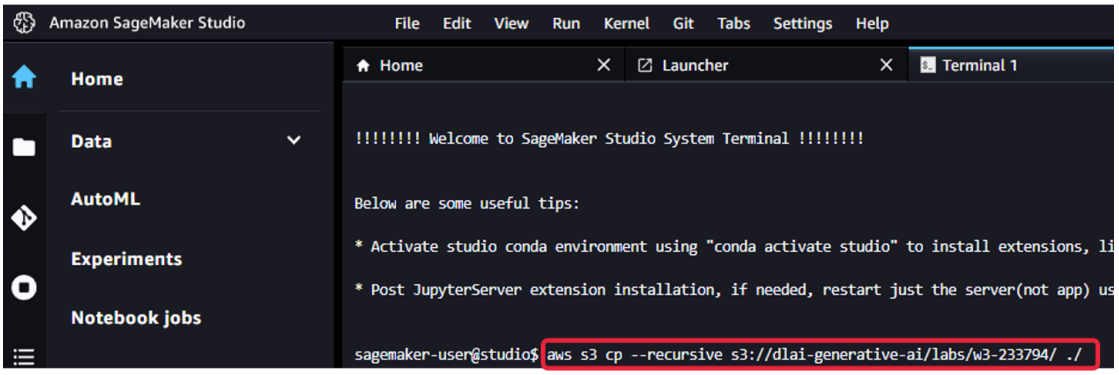


Open System terminal

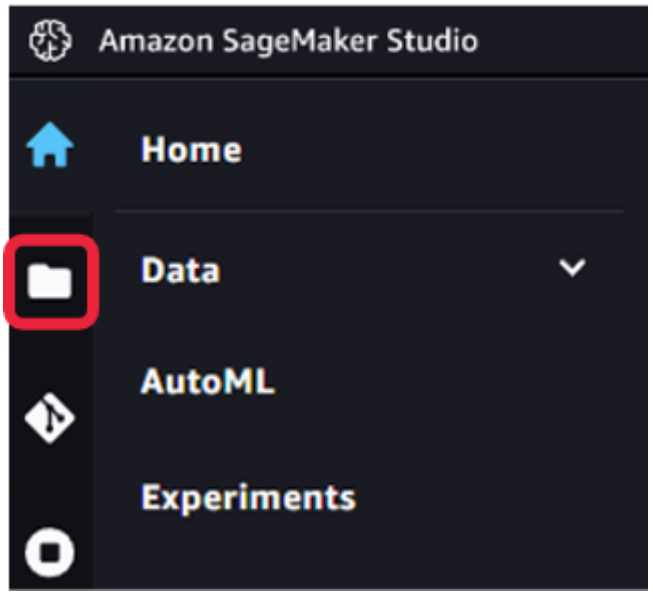


Use the following command (you can copy and paste it) in the System terminal to download the lab:

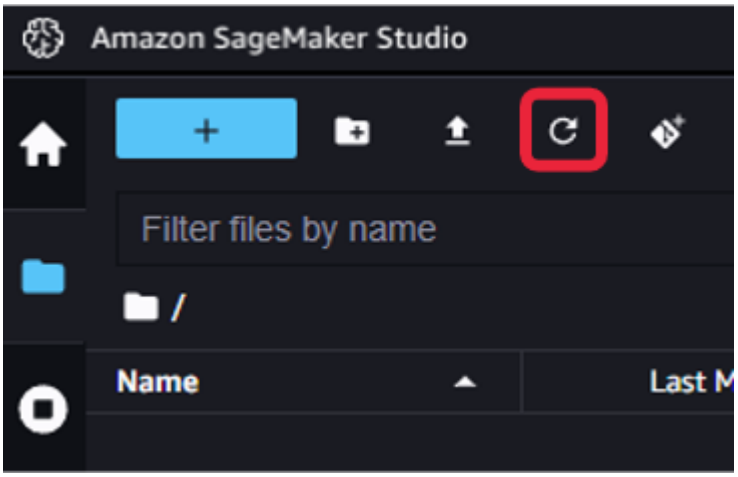
aws s3 cp --recursive s3://dlai-generative-ai/labs/w3-233794/ ./



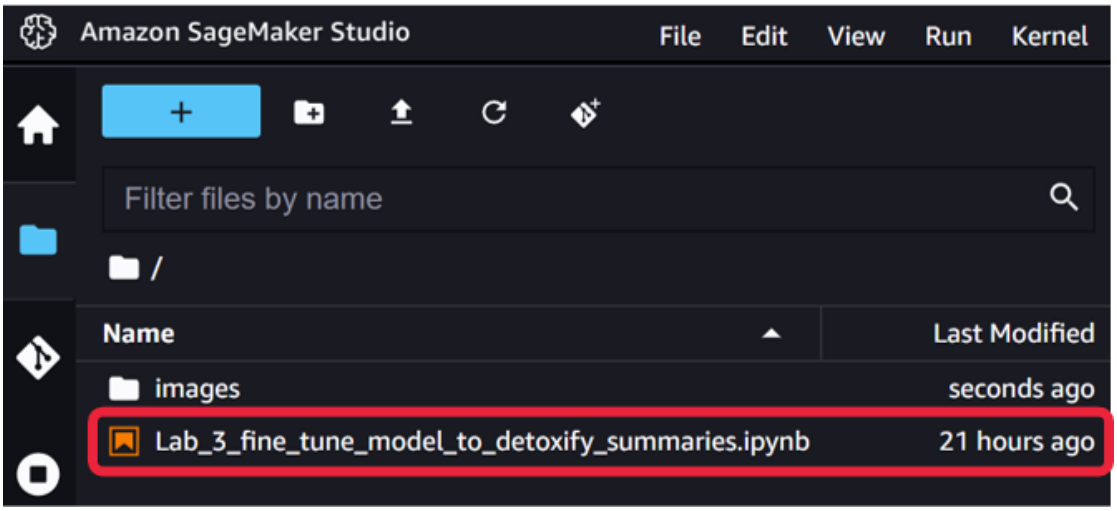
Click on the folder icon on the left to find the downloaded notebook:



Note: You might need to update the environment to see the downloaded notebook.

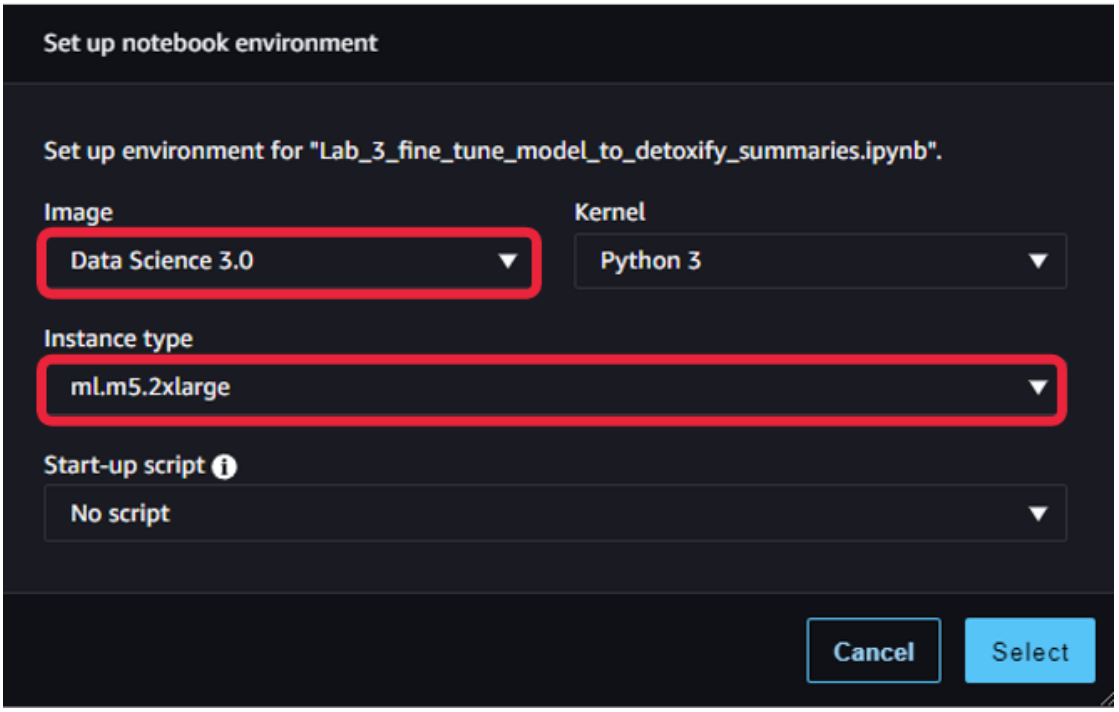


Open `Lab_3_fine_tune_model_to_detoxify_summaries.ipynb` notebook (if you need to set the kernel, please choose "Python 3 (Data Science 3.0)" with instance type `ml.m5.2xlarge`).



Lab content: Fine-Tune FLAN-T5 to Generate More-Positive Summaries.

Follow the lab instructions in the `Lab_3_fine_tune_model_to_detoxify_summaries.ipynb` notebook.



Finish the lab

If you wish to download the notebook, right-click on the notebook file in the File Browser, then select **Download**. Or, in the main menu, choose **File**, then **Download**.