

02 - LLM Introduction

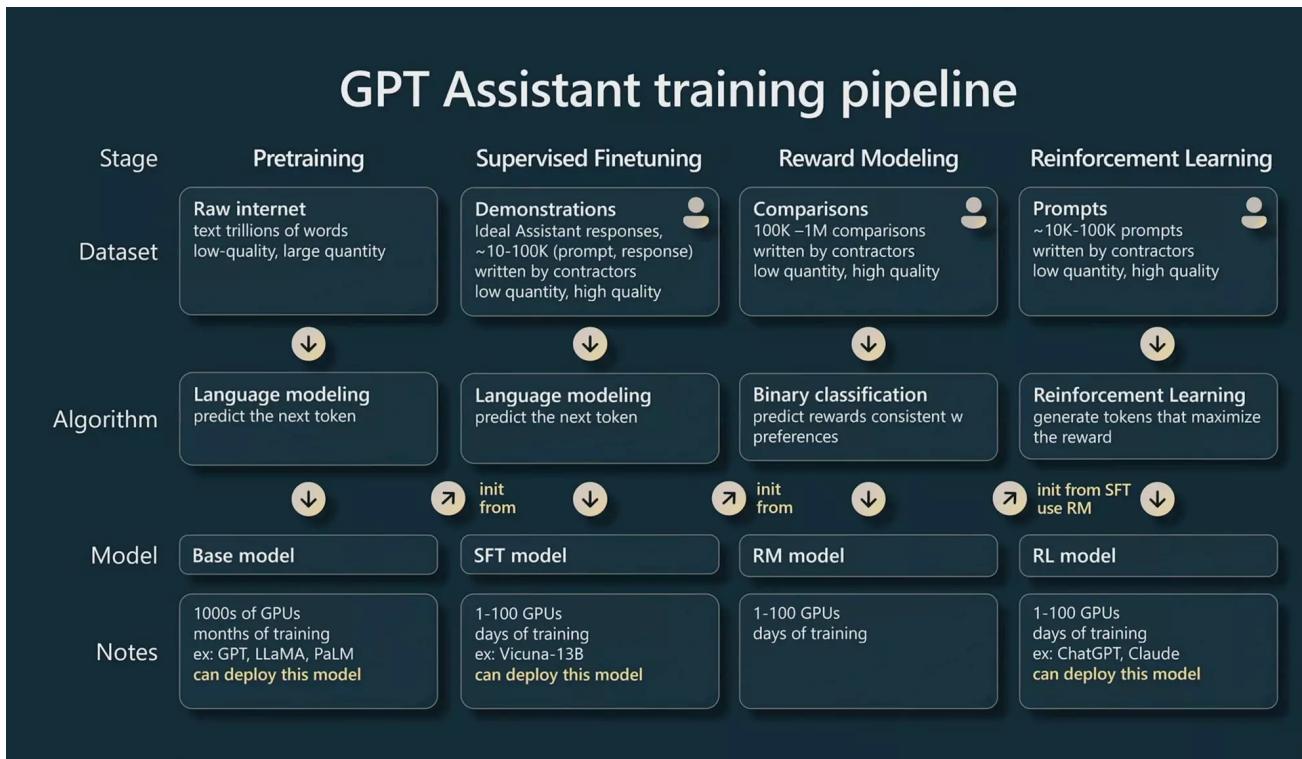
Agenda

- LLM Training Pipeline
 - Pre-training
 - Supervised finetuning (SFT)
 - Alignment (RLHF)
 - Summary
- List of Available LLMs
- References

LLM Training Pipeline

This is a famous picture from talk [State of GPT](#) given by Andrej Karpathy:

It contains about the training pipeline of GPT assistants like ChatGPT, from tokenization to pretraining, supervised finetuning, and Reinforcement Learning from Human Feedback (RLHF). Dive deeper into practical techniques and mental models for the effective use of these models, including prompting strategies, finetuning, the rapidly growing ecosystem of tools, and their future extensions.



Pic taken from <https://build.microsoft.com/en-US/sessions/db3f4859-cd30-4445-a0cd-553c3304f8e2>

1) Pre Training

- Large Decoder models (Sizes of around ~10B to 100B parameters) are taken and are fed trillions of tokens (Internet Data)
- The objective is to predict next token (Language modeling)
- It is making a baby train to predict the next token ("Twinkle Twinkle ...")
- Also called Self-supervised training..
- Tokens are word pieces that language model can understand (GPT3 used [tiktoken](#)) vocab size is ~50,000+
- The loss is simple cross entropy loss
- The learnings are stored as model weights (Every 1B model occupies approximately ~ 4 GB on device(without quantization))
- These models are called Foundation models (Example GPT-3)

1

Pretraining

Dataset:

100B to >5T tokens

Task: Next-token prediction on unlabeled texts

Output: base model / “foundation model”

Project Gutenberg (PG) is a volunteer effort to digitize and archive cultural works, as well as to "encourage the creation and distribution of eBooks". It was founded in 1971 by American writer Michael Hart and is the oldest digital library. Most of the items in its collection are the full texts of books or individual stories in the public domain. All files can be accessed for free under an open format layout, available on almost any computer. As of 3 October 2015, Project Gutenberg had reached 50,000 items in its collection of free eBooks.

What kind of data goes into the training?

Data collection

Download a large amount of publicly available data



Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

[Training data mixture used in Meta's LLaMA model]

Open datasets: RedPajama, Pile

Two Example Models

2 example models

GPT-3 (2020)

50,257 vocabulary size
2048 context length
175B parameters
Trained on 300B tokens

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3.2 7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3.6 7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3.13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

Training: (rough order of magnitude to have in mind)

- O(1,000 - 10,000) V100 GPUs
- O(1) month of training
- O(1-10) \$M

LLaMA (2023)

32,000 vocabulary size
2048 context length
65B parameters
Trained on 1-1.4T tokens

params	dimension	n_{heads}	n_{layers}	learning rate	batch size	n_{tokens}
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

Table 2.2: Model sizes, architectures, and optimization hyper-parameters.

Training for 65B model:

- 2,048 A100 GPUs
- 21 days of training
- \$5M

[Language Models are Few-Shot Learners, OpenAI 2020]
[LLaMA: Open and Efficient Foundation Language Models, Meta AI 2023]

- We do not need to pre-train models in this way. Why it is not possible??!
- 175B model occupies ~ 700 GB & 65B model ~ 260 GB

2) Supervised Finetuning (SFT)

2

Supervised finetuning

More next-token
prediction

Usually 1k-50k
instruction-response
pairs

```
{
    "instruction": "Write a limerick about a
                    pelican.",
    "input": "",
    "output": "There once was a pelican so fine,
              \nHis beak was as colorful as
              sunshine,\nHe would fish all day,\n
              a very unique way,\nThis pelican w
              truly divine!\n\n\n"
  },
  {
    "instruction": "Identify the odd one out from
                  the group.",
    "input": "Carrot, Apple, Banana, Grape",
    "output": "Carrot\n\n"
  }
}
```

- These foundation models are document completers
 - How to make pizza?
 - Adding more context to the question: for a family of six

- Adding follow-up questions: ? What ingredients do I need? How much time would it take?
- goal of SFT is to optimize the pretrained model to generate the responses that users are looking for
- Training data: high-quality data in the format of (prompt, response)
- These models are called Instruction Fine Tuned models (Ex: InstructGPT text-davinci-003, gpt-3.5-turbo-instruct)
- These models can be identified with 'Instruct' or 'Chat' in their names Ex: Llama Chat

3) Alignment

These SFT model can produce anything as an answer it may be **Toxic, Non Truthful or False (hallucination)** and some times **Not useful / Not Safe**. All of these cannot be handled by a **simple cross-entropy loss function**

If a language model **produces code it should be executable!**

For example when some people/plays music it sounds sweet to hear

This is called **Alignment** where the Language model aligns with human preferences

Two steps

- Reward Model Training
- Reinforcement learning

Dialogues are flexible. Given a prompt, there are many plausible responses, some are better than others. Demonstration data tells the model what responses are **plausible 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?**

Reward Model Training

Prompts are sampled and SFT model is asked to generate **multiple responses & Human annotators are asked to rank** the responses as shown below

Based on these ranking information a **classifier model is trained to understand human preferences** (This reward model is of the same kind of LLMs used for training for instance [InstructGPT](#) paper tells they use **6B** RMs)

At this point in the RLHF system, we have an initial language model that can be used to generate text and a preference model that takes in any text and assigns it a score of how well humans perceive it. Next, we use **reinforcement learning (RL)** to optimize the original language model with respect to the reward model.

The screenshot shows the RM Dataset interface with three examples of prompts and their corresponding responses and scores:

- Example 1 (Score: 3):**
 - Prompt: Write a Python function that checks if a given string is a palindrome.
 - Response 1 (User 1): Certainly! Here's a Python function that checks if a given string is a palindrome.


```
python
def is_palindrome(word):
    # Convert the word to lowercase and remove spaces
    word = word.lower().replace(" ", "")

    # Compare the word with its reverse
    if word == word[::-1]:
        return True
    else:
        return False
```
 - Response 2 (User 2): Certainly! Here's a Python function that checks if a given string is a palindrome.


```
python
def is_palindrome(string):
    # Converts the string to lowercase and removes spaces
    string = string.lower().replace(" ", "")

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```
 - Note: This function takes a string as input, converts it to lowercase, and removes any spaces. It then checks if the reversed string is equal to the original string. If they are equal, the function returns 'True', indicating that the string is a palindrome. Otherwise, it returns 'False'.
 - Output: Here's an example of how you can use the function:


```
python
word1 = "radar"
print(is_palindrome(word1)) # Output: True

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```
 - Note: The function first converts the input word to lowercase and removes any spaces. Then, it compares the word with its reverse using slicing ('[::-1]'). If the word is equal to its reverse, it is a palindrome, and the function returns 'True'. Otherwise, it returns 'False'.
- Example 2 (Score: 2):**
 - Prompt: Write a Python function that checks if a given string is a palindrome.
 - Response 1 (User 1): Certainly! Here's a Python function that checks if a given string is a palindrome.


```
python
def is_palindrome(string):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join([char.lower() for char in s if char.isalnum()])

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```
 - Response 2 (User 2): Certainly! Here's a Python function that checks if a given string is a palindrome.


```
python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join([char.lower() for char in s if char.isalnum()])

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```
 - Note: In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing ('[::-1]') creates a reversed copy of the string. If they are equal, the string is a palindrome, and the function returns 'True'; otherwise, it returns 'False'.
 - Output: Here's an example usage of the function:


```
python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "Hello world"
print(is_palindrome(string3)) # False
```
 - Note: In the above example, 'is_palindrome' is called with three different strings. The function correctly identifies the palindromes ('A man, a plan, a canal: Panama' and 'race car') and returns 'True'. For the non-palindrome 'Hello world', it returns 'False'.
- Example 3 (Score: 1):**
 - Prompt: Write a Python function that checks if a given string is a palindrome.
 - Response 1 (User 1): Certainly! Here's a Python function that checks if a given string is a palindrome.


```
python
def is_palindrome(string):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join([char.lower() for char in s if char.isalnum()])

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```
 - Response 2 (User 2): Certainly! Here's a Python function that checks if a given string is a palindrome.


```
python
def is_palindrome(s):
    # Remove all non-alphanumeric characters and convert to lowercase
    s = ''.join([char.lower() for char in s if char.isalnum()])

    # Check if the reversed string is equal to the original string
    return s == s[::-1]
```
 - Note: In this function, we first remove all non-alphanumeric characters from the string and convert it to lowercase using a list comprehension. Then, we check if the reversed string is equal to the original string using slicing ('[::-1]') creates a reversed copy of the string. If they are equal, the string is a palindrome, and the function returns 'True'; otherwise, it returns 'False'.
 - Output: Here's an example usage of the function:

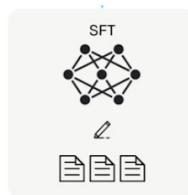

```
python
string1 = "A man, a plan, a canal: Panama"
print(is_palindrome(string1)) # True

string2 = "race car"
print(is_palindrome(string2)) # True

string3 = "Hello world"
print(is_palindrome(string3)) # False
```
 - Note: In the above example, 'is_palindrome' is called with three different strings. The function correctly identifies the palindromes ('A man, a plan, a canal: Panama' and 'race car') and returns 'True'. For the non-palindrome 'Hello world', it returns 'False'.

RLHF Step 2

LLM finetuned in step 1:



Sample prompt
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...

Human ranks responses
D > C > A = B

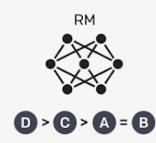
Sample prompt

Collect model responses

Human ranks responses

Time & labor intensive

Train reward model (Another LLM)

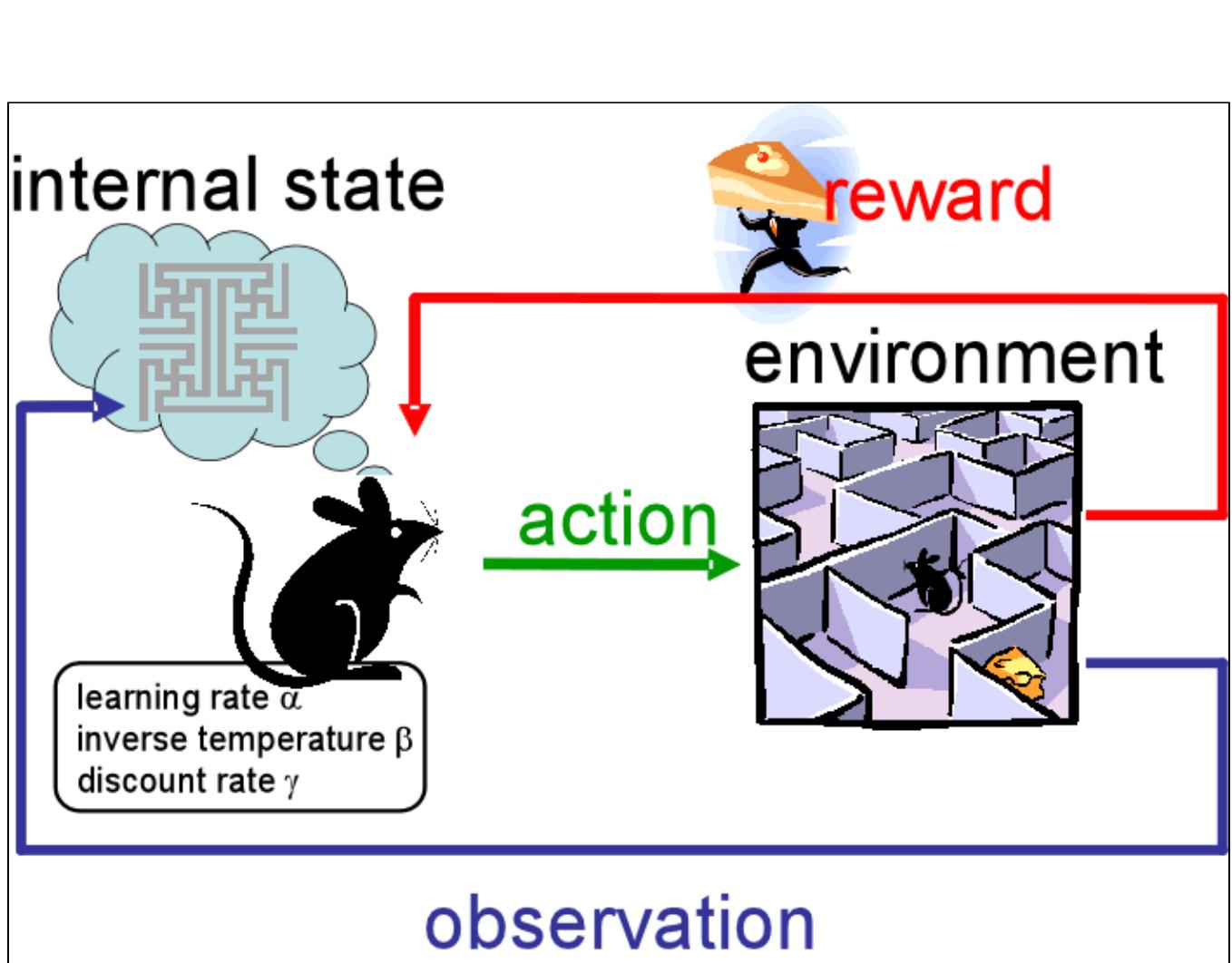


Reinforcement Learning

Reinforcement learning (RL) is a machine learning technique that uses **feedback** to train **agents** to behave in an environment. The goal of RL is to **maximize cumulative reward**.

In RL, agents learn by performing **actions** and seeing the **results**. For each good action, the agent receives positive feedback, and for each bad action, the agent receives negative feedback or penalty.

Common RL problems are Self Driving Cars, Humanoid robot walking



Here **Agent** is a language model that takes in a prompt and returns a sequence of text

The **action space** is all the tokens corresponding to the vocabulary of the language model (often on the order of 50k tokens)

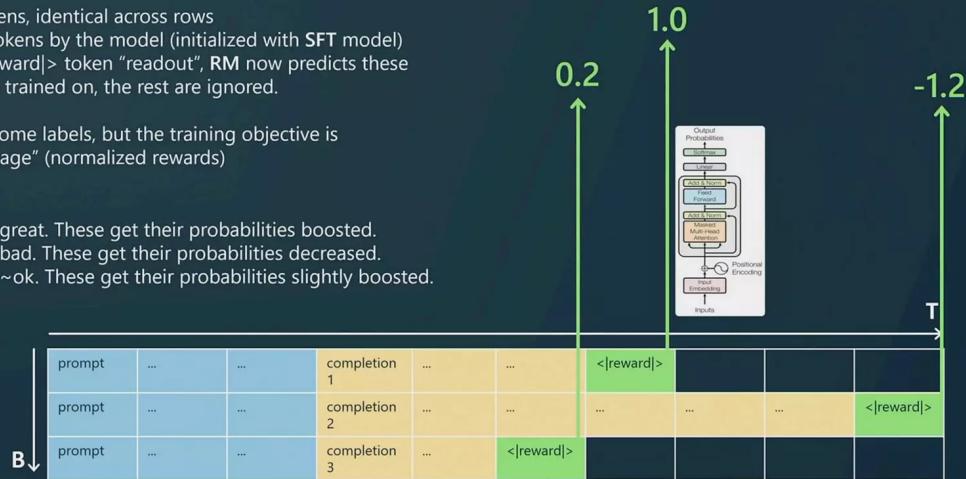
RL Training

Blue are the prompt tokens, identical across rows
Yellow are completion tokens by the model (initialized with SFT model)
Green is the special <|reward|> token "readout", RM now predicts these
Only the yellow cells are trained on, the rest are ignored.

The sampled tokens become labels, but the training objective is weighted by the "advantage" (normalized rewards)

In this example:

- Row #1 tokens were great. These get their probabilities boosted.
- Row #2 tokens were bad. These get their probabilities decreased.
- Row #3 tokens were ~ok. These get their probabilities slightly boosted.

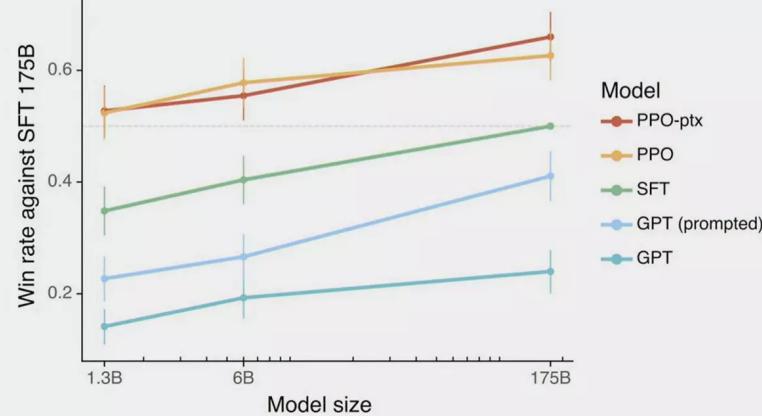


[Proximal Policy Optimization Algorithms, Schulman et al. 2017]

Why RLHF works?

Why RLHF?

It works better.



[Training language models to follow instructions with human feedback, OpenAI, 2022]

Summary

Large Language Models are trained in three different stages:

Language model Pretraining

It is like prompting a new born baby "Twinkle Twinkle little star"

But are Language models are prompted to complete entire Web data (Trillions of tokens Common Crawl, C4 etc.,)



Supervised Finetuning

Just like a little grown up kid learns from question answering, also called Instruction Finetuning , the LLM are trained to generate tokens so as to answer to the instructions



Reinforcement learning with Human Feedback

Generating a complete professionally sounding human being (or LLM in our context) who is Truthful/Factual, Non Toxic and Safe & Helpful



Once All this done we get ChatGPT

LE

Can you explain to me how RL with human feedback works for language models like I am five?



Sure! So, imagine you have a robot named Rufus who wants to learn how to talk like a human. Rufus has a language model that helps him understand words and sentences.

First, Rufus will say something using his language model. For example, he might say "I am a robot."

Then, a human will listen to what Rufus said and give him feedback on whether it sounds like a natural sentence a human would say. The human might say, "That's not quite right, Rufus. Humans don't usually say 'I am a robot.' They might say 'I'm a robot' or 'I am a machine.'"

Rufus will take this feedback and use it to update his language model. He will try to say the sentence again, using the new information he received from the human. This time, he might say "I'm a robot."

The human will listen again and give Rufus more feedback. This process will continue until Rufus can say sentences that sound natural to a human.

Over time, Rufus will learn how to talk like a human thanks to the feedback he receives from humans. This is how language models can be improved using RL with human feedback.

For List of Available LLMs

<https://chat.lmsys.org/?leaderboard>

Apache 2.0 & MIT are Potential Commercial License (Licensing scope goes beyond this page:))

Please consider going through the License documentation for further details
<https://github.com/eugeneyan/open-lmfs>

References:

[State of GPT \(microsoft.com\)](#)

[RLHF: Reinforcement Learning from Human Feedback \(huyenchip.com\)](#)

[LLM Training: RLHF and Its Alternatives \(sebastianraschka.com\)](#)

[Aligning language models to follow instructions \(openai.com\)](#)

1) LLMs for Everything and Everyone! by Sebastian Raschka, PhD
<https://lnkd.in/gVac4H94>

2) The Busy persons intro to LLMs by Andrej Karpathy
<https://lnkd.in/gxVuK2-F>

3) State of GPT by Andrej Karpathy
<https://lnkd.in/gjxktFJv>

4) A Hackers' Guide to Language Models by Jeremy Howard
<https://lnkd.in/gshimYrF>