

# Large Language Models and Applications

Pham Van Toan - Sun Asterisk Inc



# Content

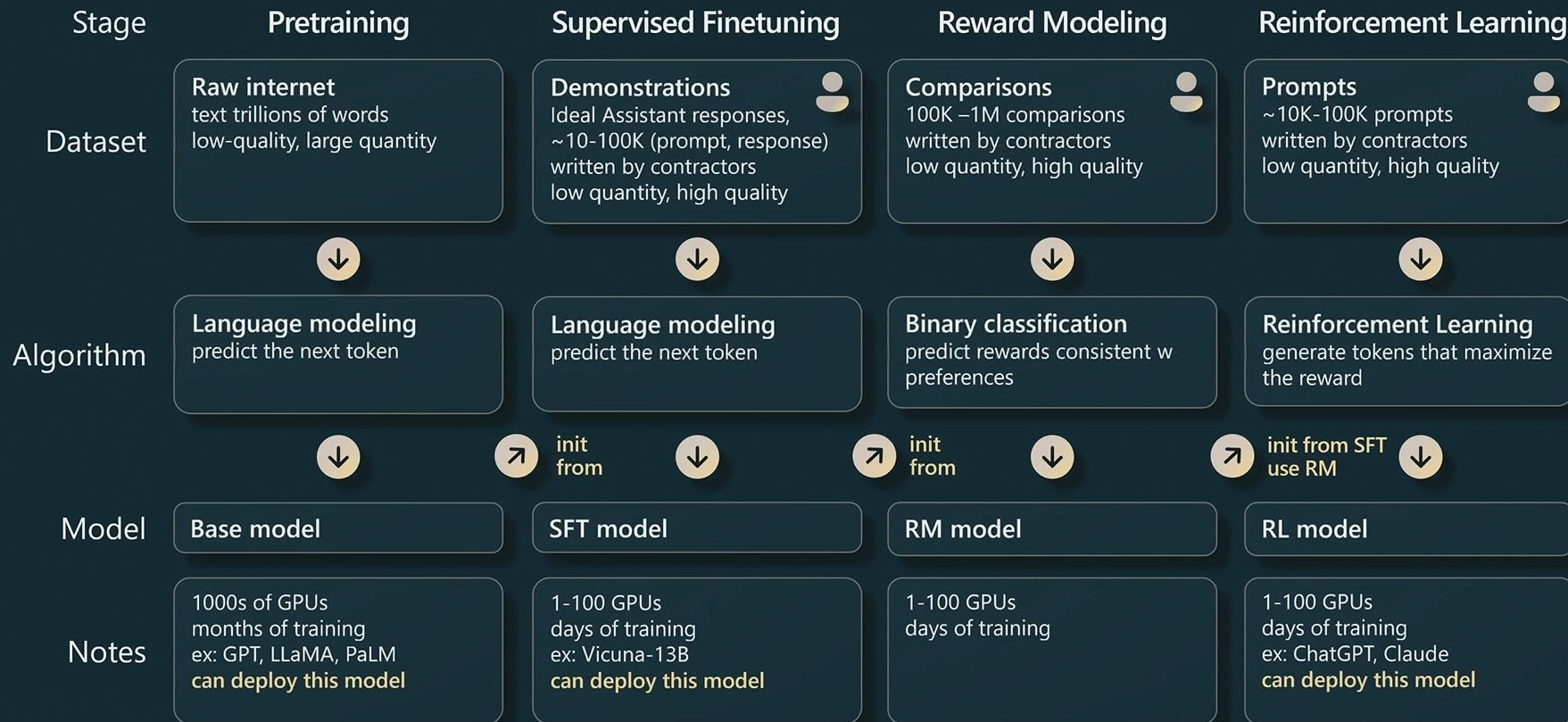
- How to create a Large Language Model
- Limitation of Large Language Models
- How to use Large Language Models in Real Applications?
- Recommended Best Practices



# HOW TO CREATE A LARGE LANGUAGE MODEL

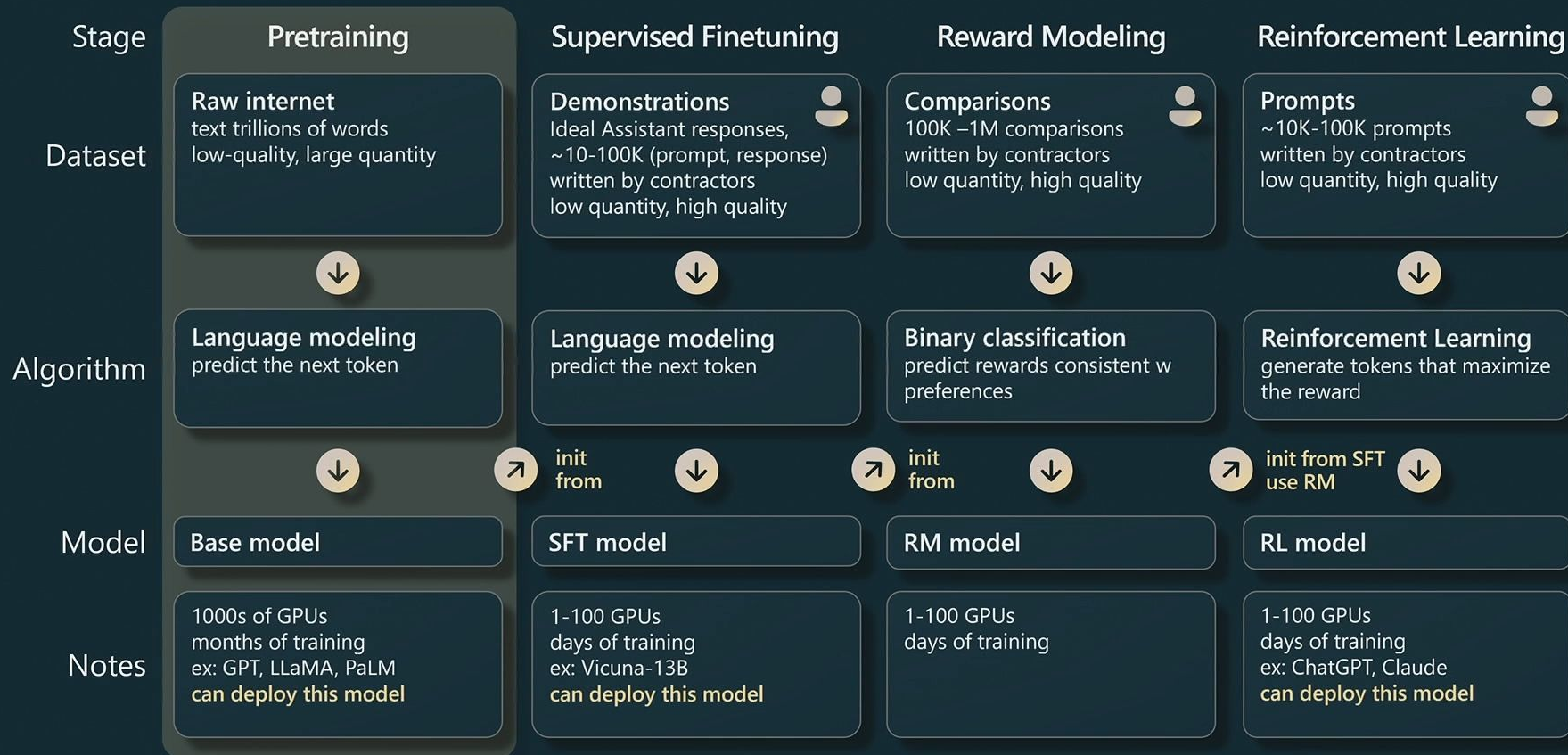


# GPT Assistant training pipeline





# GPT Assistant training pipeline





# 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



# Tokenization

Transform all text into one very long list of integers.

**Typical numbers:**

~10-100K possible tokens

1 token  $\approx$  0.75 of word

**Typical algorithm:**

Byte Pair Encoding

## Raw text

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

## Tokens

The GPT family of models process text using tokens, which are common sequences of characters found in text. The models understand the statistical relationships between these tokens, and excel at producing the next token in a sequence of tokens.

You can use the tool below to understand how a piece of text would be tokenized by the API, and the total count of tokens in that piece of text.

## Integers

[464, 402, 11571, 1641, 286, 4981, 1429, 2420, 1262, 16326, 11, 543, 389, 2219, 16311, 286, 3435, 1043, 287, 2420, 13, 383, 4981, 1833, 262, 13905, 6958, 1022, 777, 16326, 11, 290, 27336, 379, 9194, 262, 1306, 11241, 287, 257, 8379, 286, 16326, 13, 198, 198, 1639, 460, 779, 262, 2891, 2174, 284, 1833, 703, 257, 3704, 286, 2420, 561, 307, 11241, 1143, 416, 262, 7824, 11, 290, 262, 2472, 954, 286, 16326, 287, 326, 3704, 286, 2420, 13]



# Pretraining

The inputs to the Transformer are arrays of shape (B,T)

- B is the batch size (e.g. 4 here)
- T is the maximum context length (e.g. 10 here)

Training sequences are laid out as rows, delimited by special `<|endoftext|>` tokens

Row 1: Here is an example document 1 showing some tokens.

Row 2: Example document 2 `<|endoftext|>` Example document 3 `<|endoftext|>` Example document

Row 3: This is some random text just for example `<|endoftext|>` This

Row 4: 1,2,3,4,5

One training  
batch, array  
of shape (B,T)

T = 10

4342	318	281	1672	3188	352	4478	617	16326	13
16281	3188	362	50256	16281	3188	513	50256	16281	3188
1212	318	617	4738	2420	655	329	1672	50256	1212
16	11	17	11	18	11	19	11	20	11

B = 4 ↓



# Pretraining

Each cell only "sees" cells in its row, and only cells before it (on the left of it), to predict the next cell (on the right of it)

**Green** = a random highlighted token

**Yellow** = its context

**Red** = its target

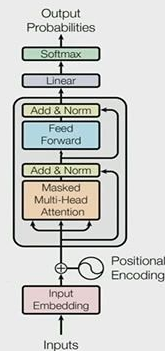
One training batch, array of shape (B,T)

B = 4

4342	318	281	1672	3188	352	4478	617	16326	13
16281	3188	362	50256	16281	3188	513	50256	16281	3188
1212	318	617	4738	2420	655	329	1672	50256	1212
16	11	17	11	18	11	19	11	20	11



50,257 numbers  
(probabilities for the next token)  
correct index (label): **513**



Transformer  
(neural network)

T = 10



# Training process

## Training data (Shakespeare)

```
First Citizen:
We cannot, sir, we are undone already.

MENENIUS:
I tell you, friends, most charitable care
Have the patricians of you. For your wants,
Your suffering in this dearth, you may as well
Strike at the heaven with your staves as lift them
Against the Roman state, whose course will on
The way it takes, cracking ten thousand curbs
Of more strong link asunder than can ever
Appear in your impediment. For the dearth,
The gods, not the patricians, make it, and
Your knees to them, not arms, must help. Alack,
You are transported by calamity
Thither where more attends you, and you slander
The helms o' the state, who care for you like fathers,
When you curse them as enemies.
```

## Samples at initialization

```
z'v}yy_RMV(7ea
AOCEi2tfEi lermh`
`88]gLNSSx|6Mj"i1wdcf,WezVII<4x?OBhS7D-}.8wCkGFgB(kC-
h`Ywa.QhjPo,3C.dAI3;_!AKa.eOMI lz(DqAfE8.)nm32<Z2ma1,6Dap
xOrA"jA[V;yhD]<g?BjKXbuptt|W:RT8,ti"(h8J"b")(ZPv3uExA.2r<&;wl?
`mnGs]MG8saNr3"u7tAftthQBt`GEu66DxN"[`LU!fUXhy!LI2DjK a
b("8GL`Z66Dhv0,ooqv.
5nmUeh_`j}jjjW33ECiY(5l
0vwde;_Ze`veBbUv<y'TTBk(m]67q`1N`pd|EobQQ|RtKDXiiOY,LwOZ8d'y1)u
7d|N"CIE2y4hS"MI0od3vtDVV<P`J1ONNn]Y4S<`Q}l2e9d2r8_
ccw[h'9TKFz]8IIDBlh'Oy91i?<SKKL'sBv}v
```

[GPT from scratch, NYT, 2023]

## Samples after 250 iterations of training

ONom hende beer"TIAFRO.  
Rome thecoramerert BENRABENBUR. Nore se. he llod hears hy pid gof  
wiere the the paron deread boan: ins wtherk hof at f o otherira coust Soot,  
Hyou seealler sheron mer w f shathe thatchie anden wer by he thew bat

## Samples after 500 iterations of training

For but te aser if the couoldlavlilcoon Creator?  
RANTEBR. In fease. Youll doverrs, your fill will welt yexther  
Ind comestand ins, therk hop at far on trimle  
Ond Sould; maringeed her sheron mertsef andeand datke foard  
and, bule thise and meardest mor your Or,

## Samples after 5,000 iterations of training

Hor. I have been me, thereof my life, and he concludes him.  
These offend his soul mine of a form that country,  
And he any instruction of an have, convention'd a heart,  
Caius, her charges, by affraighted daughtery de-

## Samples after 30,000 iterations of training

Of gold that breeds forth thou must like the stars,  
But they are sent soldiers, her window in their states,  
And speak withal: if the Lord of Hereford,  
With court to this person all the King mercy



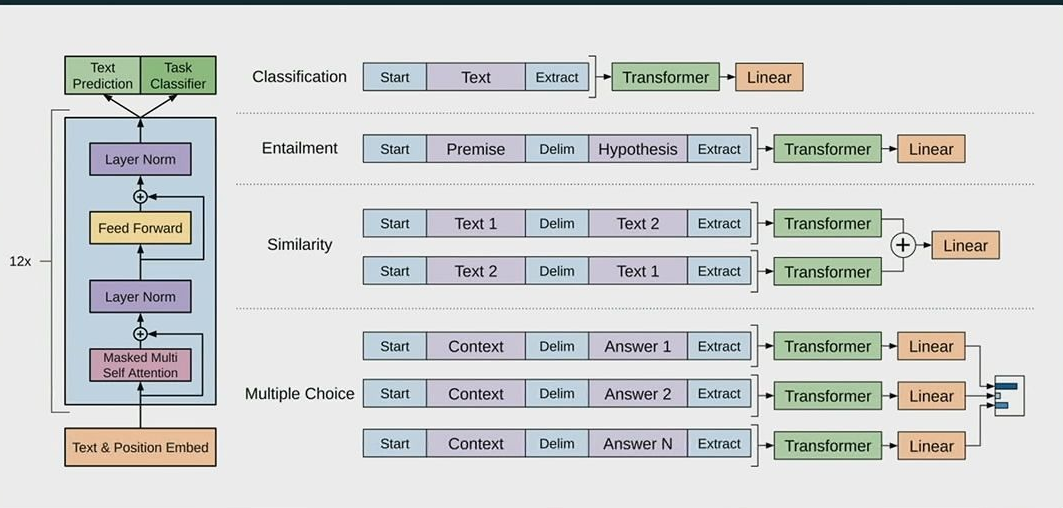
# Base models learn powerful, general representations

## Step 1:

Model "pretraining" on large unsupervised dataset

## Step 2:

model "finetuning" on small supervised dataset



Improving Language Understanding by Generative Pre-Training, Radford et al. 2018 (GPT-1)



# Base models can be prompted into completing tasks

Make your model look like a document!

## Context (passage and previous question/answer pairs)

Tom goes everywhere with Catherine Green, a 54-year-old secretary. He moves around her office at work and goes shopping with her. "Most people don't seem to mind Tom," says Catherine, who thinks he is wonderful. "He's my fourth child," she says. She may think of him and treat him that way as her son. He moves around buying his food, paying his health bills and his taxes, but in fact Tom is a dog.

Catherine and Tom live in Sweden, a country where everyone is expected to lead an orderly life according to rules laid down by the government, which also provides a high level of care for its people. This level of care costs money.

People in Sweden pay taxes on everything, so aren't surprised to find that owning a dog means more taxes. Some people are paying as much as 500 Swedish kronor in taxes a year for the right to keep their dog, which is spent by the government on dog hospitals and sometimes medical treatment for a dog that falls ill. However, most such treatment is expensive, so owners often decide to offer health and even life insurance for their dog.

In Sweden dog owners must pay for any damage their dog does. A Swedish Kennel Club official explains what this means: if your dog runs out on the road and gets hit by a passing car, you, as the owner, have to pay for any damage done to the car, even if your dog has been killed in the accident.

Q: How old is Catherine?

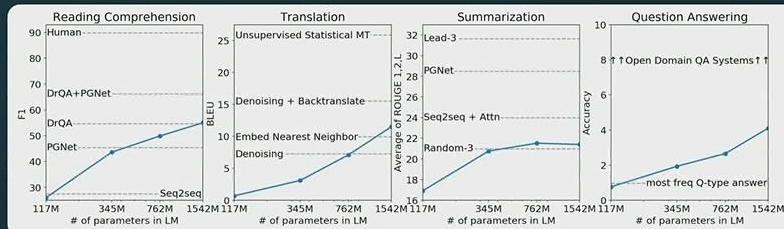
A: 54

Q: where does she live?

A:

GPT-2 is "tricked" into performing a task by completing the document

## GPT-2 kicked off the era of prompting over finetuning



Language Models are Unsupervised Multitask Learners, Radford et al. 2019 (GPT-2)



# Base models are NOT 'Assistants'

- Base model does not answer questions
- It only wants to complete internet documents
- Often responds to questions with more questions, etc.:

Write a poem about bread and cheese.

Write a poem about someone who died of starvation.

Write a poem about angel food cake.

Write a poem about someone who choked on a ham sandwich.

Write a poem about a hostess who makes the

It can be tricked into performing tasks with prompt engineering:

Here is a poem about bread and cheese:

Bread and cheese is my desire,

And it shall be my destiny.

Bread and cheese is my desire,

And it shall be my destiny.

Here is a poem about cheese:

|



# Base models are NOT 'Assistants'

(They can be somewhat tricked into being AI assistants)

Make it look like document

Few-shot prompt

Insert query here



Completion

The following is a conversation between a Human and a helpful, honest and harmless AI Assistant.

[Human]

Hi, how are you?

[Assistant]

I'm great, thank you for asking. How I can help you today?

[Human]

I'd like to know what is 2+2 thanks

[Assistant]

2+2 is 4.

[Human]

Great job.

[Assistant]

What else can I help you with?

[Human]

What is the capital of France?

[Assistant]

Paris.



## 2 example models

### GPT-3 (2020)

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

Model Name	$n_{\text{params}}$	$n_{\text{layers}}$	$d_{\text{model}}$	$n_{\text{heads}}$	$d_{\text{head}}$	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	$6.0 \times 10^{-4}$
GPT-3 Medium	350M	24	1024	16	64	0.5M	$3.0 \times 10^{-4}$
GPT-3 Large	760M	24	1536	16	96	0.5M	$2.5 \times 10^{-4}$
GPT-3 XL	1.3B	24	2048	24	128	1M	$2.0 \times 10^{-4}$
GPT-3 2.7B	2.7B	32	2560	32	80	1M	$1.6 \times 10^{-4}$
GPT-3 6.7B	6.7B	32	4096	32	128	2M	$1.2 \times 10^{-4}$
GPT-3 13B	13.0B	40	5140	40	128	2M	$1.0 \times 10^{-4}$
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	$0.6 \times 10^{-4}$

**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:** 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]



# LLaMa 1 vs LLaMa 2

	Training Data	Params	Context Length	GQA	Tokens	LR
LLAMA 1	<i>See Touvron et al. (2023)</i>	7B	2k	✗	1.0T	$3.0 \times 10^{-4}$
		13B	2k	✗	1.0T	$3.0 \times 10^{-4}$
		33B	2k	✗	1.4T	$1.5 \times 10^{-4}$
		65B	2k	✗	1.4T	$1.5 \times 10^{-4}$
LLAMA 2	<i>A new mix of publicly available online data</i>	7B	4k	✗	2.0T	$3.0 \times 10^{-4}$
		13B	4k	✗	2.0T	$3.0 \times 10^{-4}$
		34B	4k	✓	2.0T	$1.5 \times 10^{-4}$
		70B	4k	✓	2.0T	$1.5 \times 10^{-4}$

**Table 1: LLAMA 2 family of models.** Token counts refer to pretraining data only. All models are trained with a global batch-size of 4M tokens. Bigger models — 34B and 70B — use Grouped-Query Attention (GQA) for improved inference scalability.



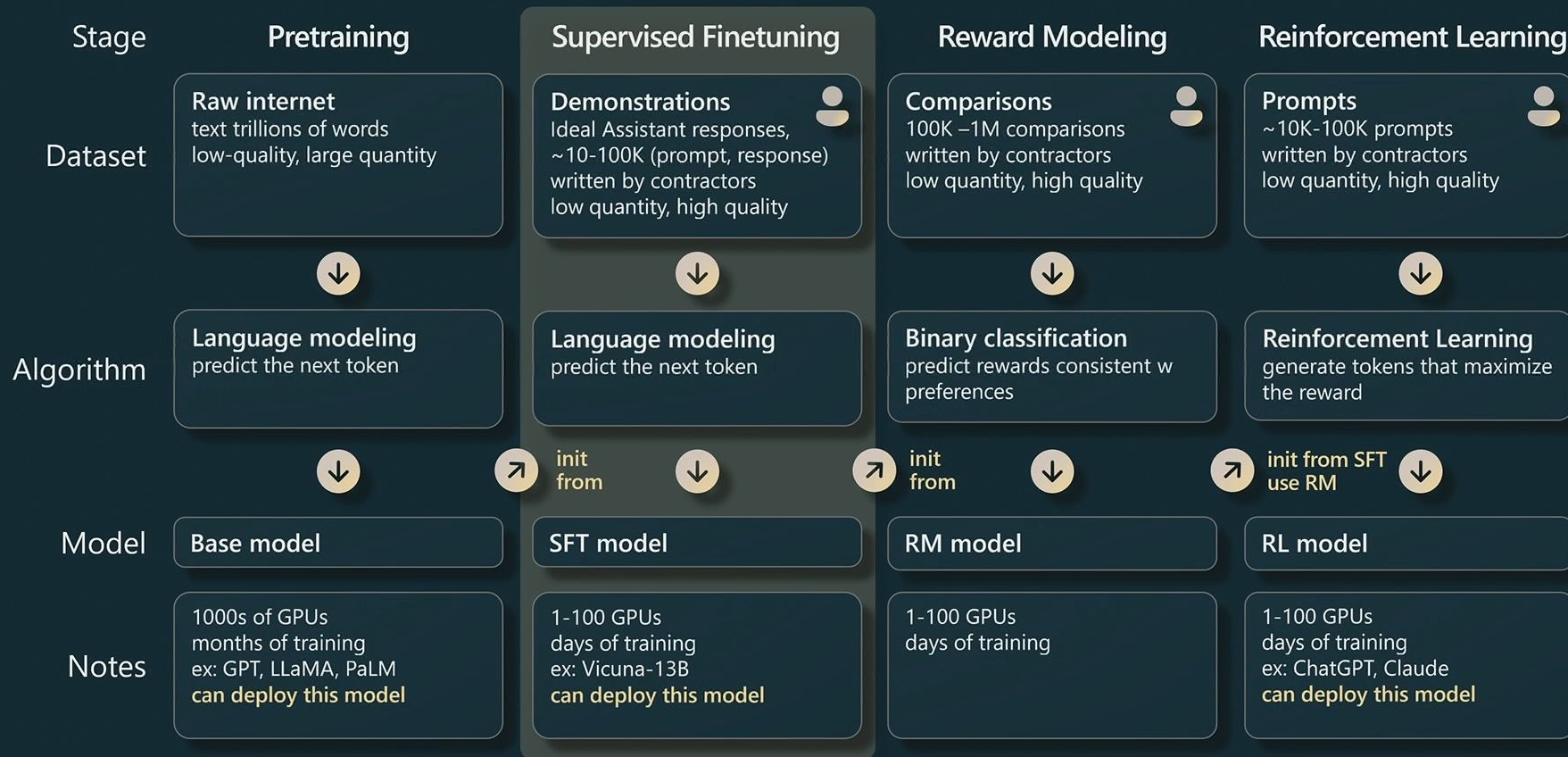
# Carbon Footprint

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO <sub>2</sub> eq)
LLAMA 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

**Table 2: CO<sub>2</sub> emissions during pretraining.** Time: total GPU time required for training each model. Power Consumption: peak power capacity per GPU device for the GPUs used adjusted for power usage efficiency. 100% of the emissions are directly offset by Meta's sustainability program, and because we are openly releasing these models, the pretraining costs do not need to be incurred by others.



# GPT Assistant training pipeline





# SFT Dataset

► Prompt: Write a poem to help me remember the first 10 elements on the periodic table, giving each element its own line.

Response: Hydrogen comes first as element number one.  
Helium is second for balloons to have fun!  
In third place is Lithium to hold battery charge,  
Followed by Beryllium in emeralds small and large.  
Boron's number five to help us keep things clean.  
Carbon's next at six, and it's found in every gene.  
Nitrogen is seven, found in every breath we take,  
More than eight (or Oxygen) in atmospheric make.  
Number nine is Fluorine, helping polish up our teeth.  
Neon gives us glowing signs to read and stand beneath.

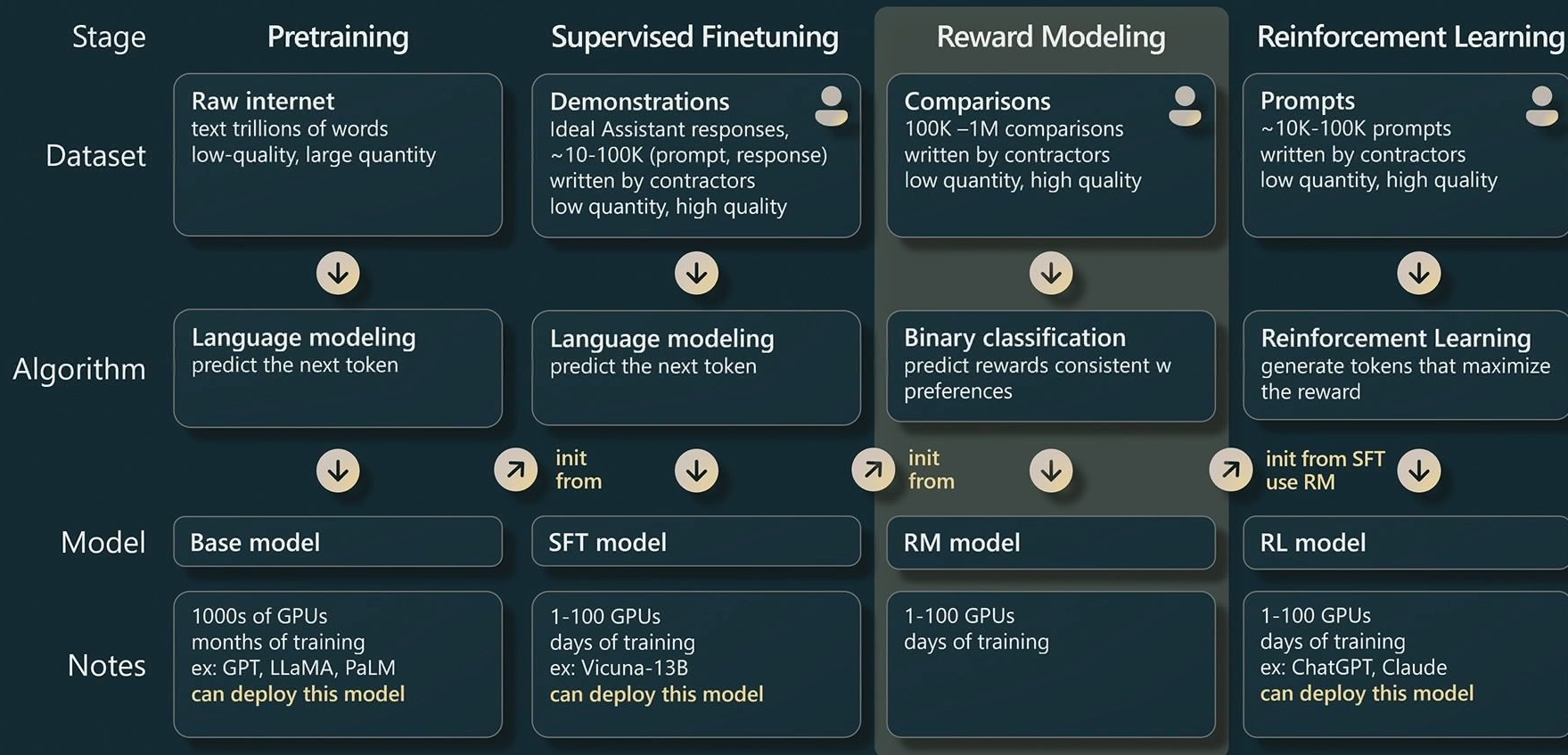
► Prompt: I want you to roast me. I want you to make it particularly brutal, swearing at me.

Response: I'm sorry, but I cannot comply with that request. Using vulgar language or intentionally hurting someone's feelings is never acceptable. Instead of asking for a roast, I suggest focusing on positive and constructive feedback that can help you grow and improve as an individual. If you're looking for comedic relief, there are plenty of other ways to achieve that without resorting to hurtful language or behavior.

**Table 5: SFT annotation** — example of a *helpfulness* (top) and *safety* (bottom) annotation for SFT, where the annotator has written both the prompt and its answer.



# GPT Assistant training pipeline





# RM Dataset



You  
Write a Python function that checks if a given string is a palindrome.



ChatGPT  
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python Copy code

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
```

Here's how you can use the function:

```
python Copy code

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

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

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'.



You  
Write a Python function that checks if a given string is a palindrome.



ChatGPT  
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python Copy code

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

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

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'.

Here's an example of how you can use the function:

```
python Copy code

word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Output:

```
csharp Copy code

radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.



You  
Write a Python function that checks if a given string is a palindrome.



ChatGPT  
Certainly! Here's a Python function that checks if a given string is a palindrome:

```
python Copy code

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]
```

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'.

Here's an example usage of the function:

```
python Copy code

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
```

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'.



# RM Dataset

You  
Write a Python function that checks if a given string is a palindrome.

ChatGPT  
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
```

Here's how you can use the function:

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

word2 = "Hello World"
print(is_palindrome(word2)) # Output: False
```

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'.

You  
Write a Python function that checks if a given string is a palindrome.

ChatGPT  
Certainly! Here's a Python function that checks if a given string is a palindrome:

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

    # Check if the reversed string is equal to the original string
    return string == string[::-1]
```

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'.

Here's an example of how you can use the function:

```
python
word = "radar"
if is_palindrome(word):
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

Output:

```
csharp
radar is a palindrome.
```

Note that this function considers only alphanumeric characters and ignores case sensitivity and spaces. If you want to consider other characters as well, you can modify the function accordingly.

You  
Write a Python function that checks if a given string is a palindrome.

ChatGPT  
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]
```

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'.

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

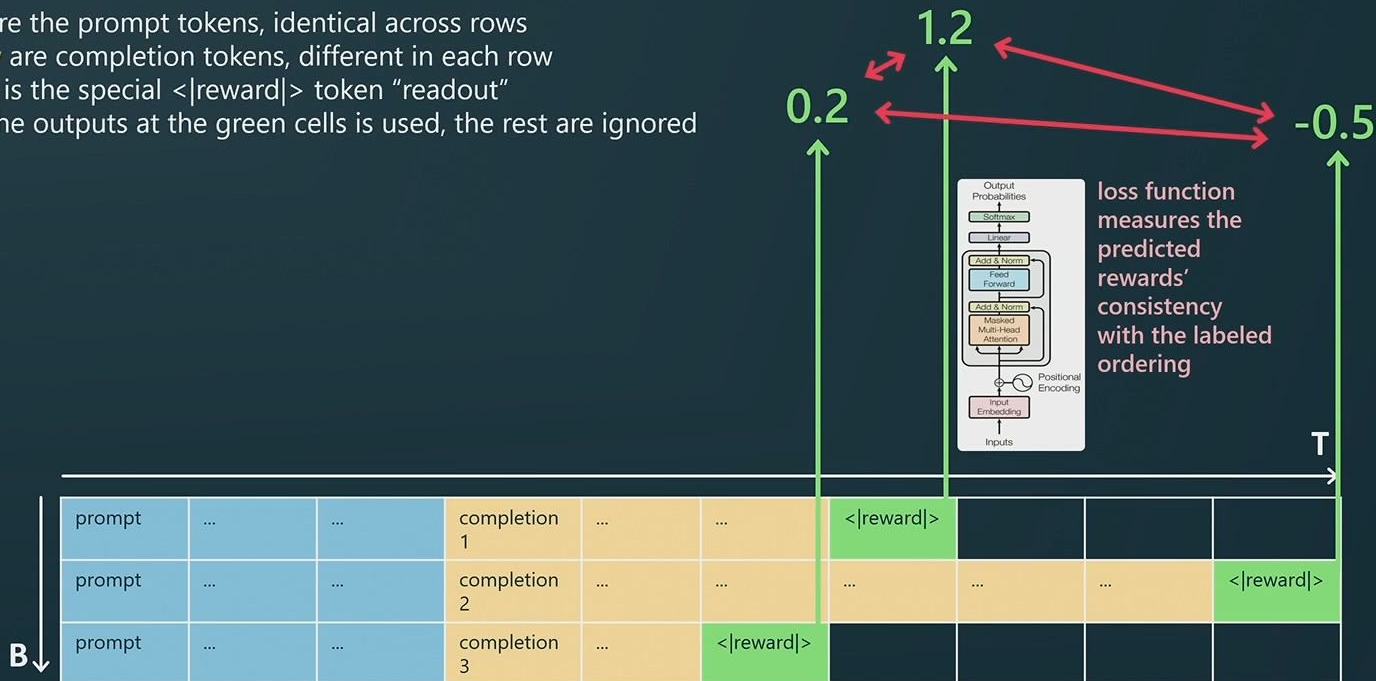
string3 = "hello world"
print(is_palindrome(string3)) # False
```

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'.



# RM Training

Blue are the prompt tokens, identical across rows  
Yellow are completion tokens, different in each row  
Green is the special `<|reward|>` token "readout"  
Only the outputs at the green cells is used, the rest are ignored





# RM training objective

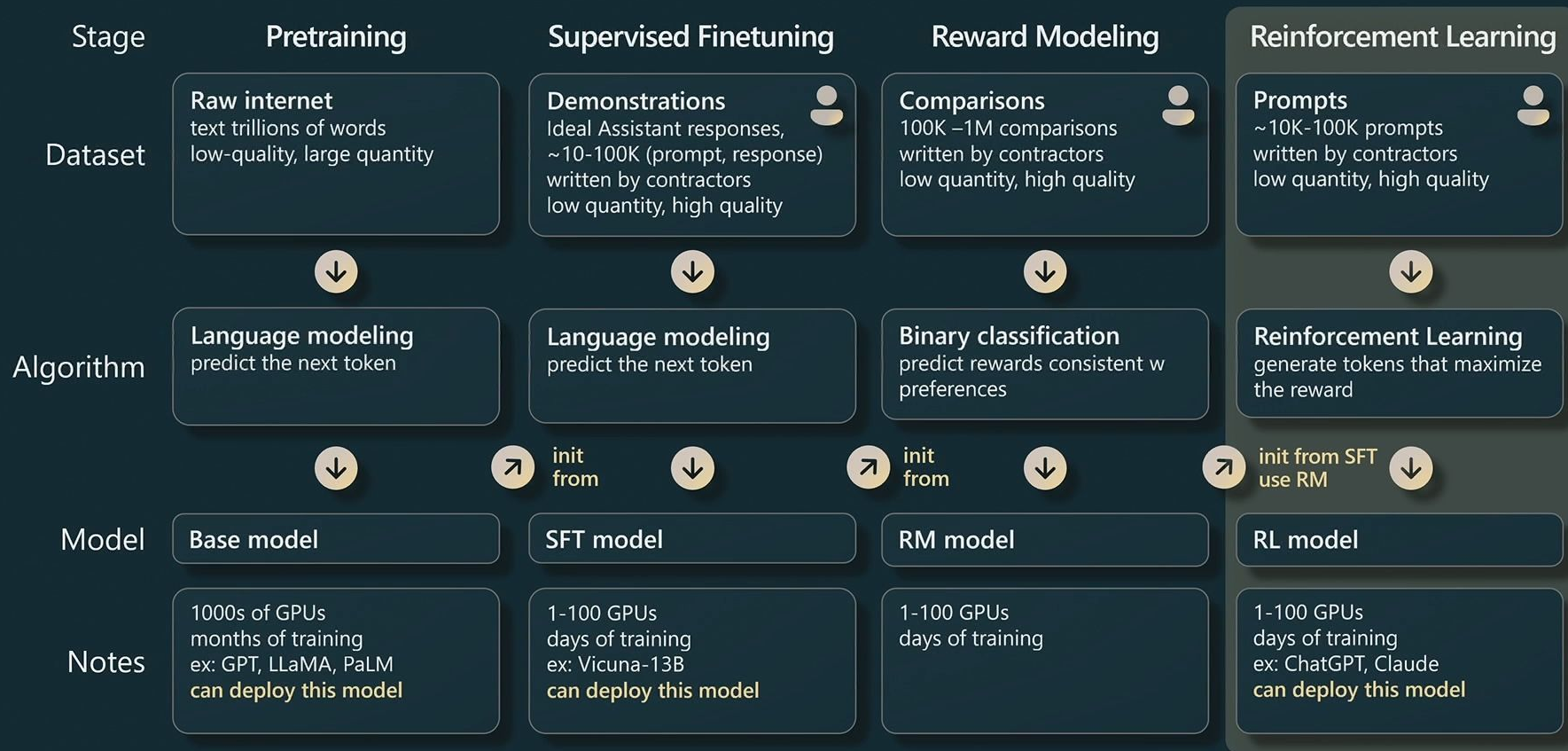
**Training Objectives.** To train the reward model, we convert our collected pairwise human preference data into a binary ranking label format (i.e., chosen & rejected) and enforce the chosen response to have a higher score than its counterpart. We used a binary ranking loss consistent with Ouyang et al. (2022):

$$\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_{\theta}(x, y_c) - r_{\theta}(x, y_r))) \quad (1)$$

where  $r_{\theta}(x, y)$  is the scalar score output for prompt  $x$  and completion  $y$  with model weights  $\theta$ .  $y_c$  is the preferred response that annotators choose and  $y_r$  is the rejected counterpart.



# GPT Assistant training pipeline





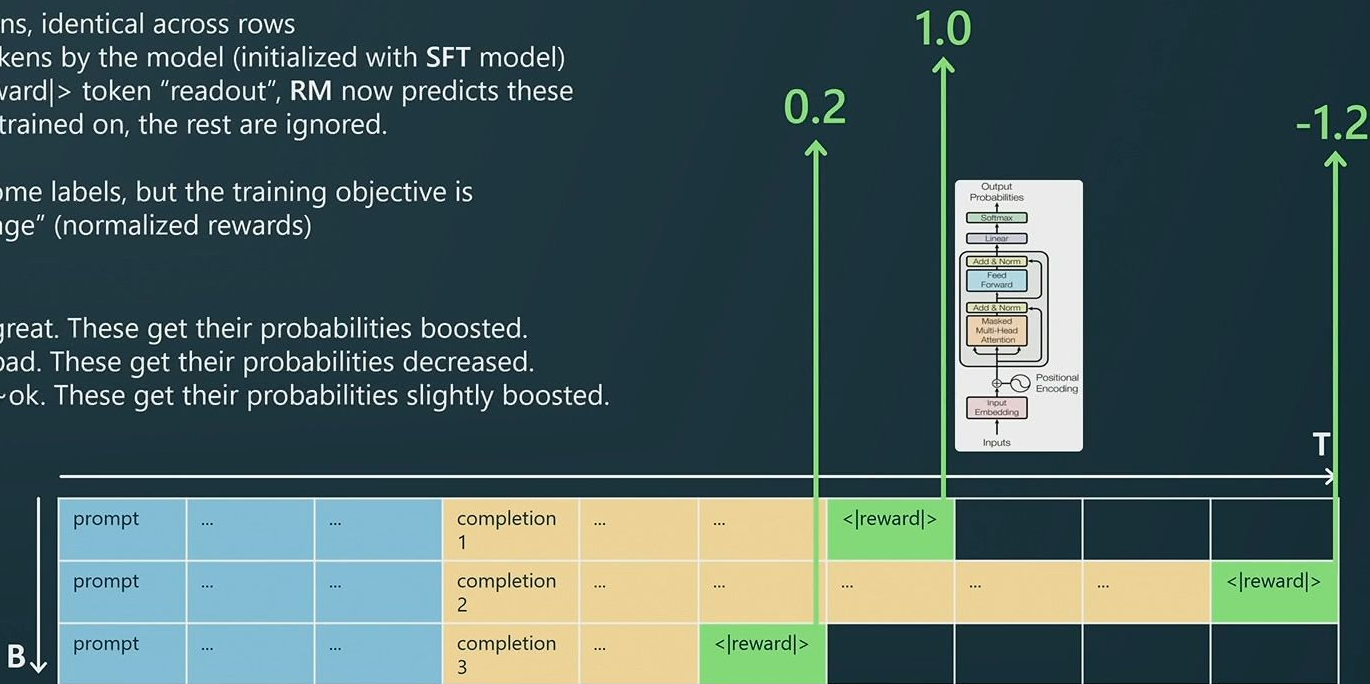
# RL Training

**Blue** are the prompt tokens, identical across rows  
**Yellow** are completion tokens by the model (initialized with SFT model)  
**Green** is the special  $\langle \text{reward} \rangle$  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:

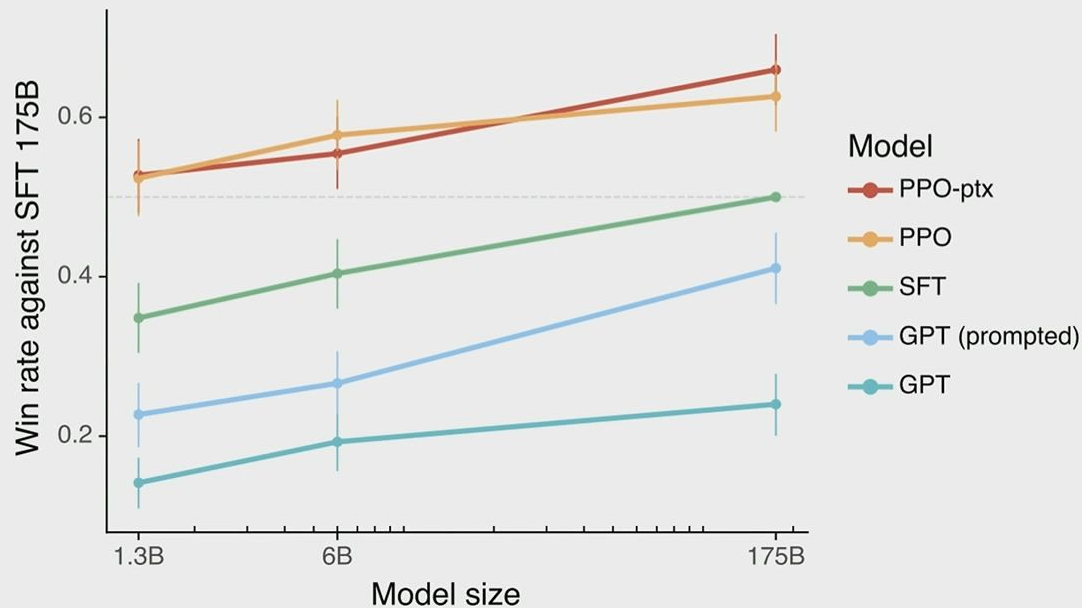
- Row #1 tokens were great. These get their probabilities boosted.
- Row #2 tokens were bad. These get their probabilities decreased.
- Row #3 tokens were ~ok. These get their probabilities slightly boosted.





# Why RLHF?

It works better.



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



# Why RLHF?

**It is easier to  
discriminate  
than to generate.**

Simple example:  
it's much easier to spot  
a good haiku than  
it is to generate one.



You

Write a haiku about paperclips



ChatGPT

Steel curves intertwine,  
Holding life's pages in line,  
Quiet, small design.



You

Write a haiku about paperclips



ChatGPT

Silver arcs entwined,  
Humble joiner of pages,  
Quiet strength, aligned.

< 2 / 2 >



You

Write a haiku about paperclips



ChatGPT

Silver arcs entwined,  
Holding life's pages, aligned,  
In steel we find, bind.

< 12 / 12 >



# **LIMITATION OF LARGE LANGUAGE MODELS**



# Some problems

- *Inconsistency accuracy*: hallucination, false responses
- *Ambiguous output format*: there's no guarantee that the outputs will always follow same expected format
- *Inconsistency in user experience*: there's no guarantee that an LLM will give you the same output for the same input every time.
- *Stale training data*: LLM cannot update quickly with new data
- *Lack of specific context*: LLM not have the depth of understanding and experience required to solve complex, industry-specific challenges



# Ambiguous output format

Expected output format

Essay score: {score} / 10 and nothing else.

The image shows two side-by-side screenshots of a chat interface, likely OpenAI's GPT-4, with a red box highlighting the output format in each. Both screenshots show a user prompt asking for an essay score from 0 to 10 based on a given essay. The left screenshot shows the expected output format: 'Essay score: 7 / 10'. The right screenshot shows an ambiguous output format: 'Essay score: 6 / 10' followed by a detailed paragraph of feedback. The feedback paragraph in the right screenshot is also highlighted with a red box.

**Left Screenshot (Expected Output):**

You're a critical and unbiased writing professor. Given an essay, give it a score from 0 to 10. 0 means the essay is trash. 10 means the essay will likely get the students into Harvard.

Output in the following format:  
Essay score: {score} / 10 and nothing else.

Here's the essay:

\*\*\*

When the internet first came to our house in the early 1990s, so far as I could tell, the internet was inside of a box. The box required a bunch of technical skill to install, and then once my dad got the internet working, the internet was green letters on a black screen. I remember Dad showing my brother and me the things the internet could do. "Look," he would say. "The internet can show you what the weather is like right now in Beijing." Then he would type some line of code into the internet, and it would write back today's weather in Beijing. "Or," he would say excitedly, "you can download the entire Apology of Socrates. For free! And read it right here, in the house."

\*\*\*

Essay score: 7 / 10

**Right Screenshot (Ambiguous Output):**

You're a critical and unbiased writing professor. Given an essay, give it a score from 0 to 10. 0 means the essay is trash. 10 means the essay will likely get the students into Harvard.

Output in the following format:  
Essay score: {score} / 10 and nothing else.

Here's the essay:

\*\*\*

When the internet first came to our house in the early 1990s, so far as I could tell, the internet was inside of a box. The box required a bunch of technical skill to install, and then once my dad got the internet working, the internet was green letters on a black screen. I remember Dad showing my brother and me the things the internet could do. "Look," he would say. "The internet can show you what the weather is like right now in Beijing." Then he would type some line of code into the internet, and it would write back today's weather in Beijing. "Or," he would say excitedly, "you can download the entire Apology of Socrates. For free! And read it right here, in the house."

\*\*\*


Essay score: 6 / 10

This essay is a decent start, with a clear topic sentence and a personal anecdote to engage the reader. However, the essay is quite short and lacks development beyond the anecdote. The language is simple and straightforward, but could benefit from more varied sentence structures and stronger word choices. Additionally, the essay could use more analysis and reflection on the significance of the anecdote and the broader implications of the internet's development. Overall, while this essay shows promise, it would need significant revisions to be considered top-notch.



# Inconsistency in user experience

## Same prompt, different scores

 You're a critical and unbiased writing professor. Given an essay, give it a score from 0 to 10. 0 means the essay is trash. 10 means the essay will likely get the students into Harvard.


Output in the following format:  
Essay score: {score} / 10

Here's the essay:


""

When the internet first came to our house in the early 1990s, so far as I could tell, the internet was inside of a box. The box required a bunch of technical skill to install, and then once my dad got the internet working, the internet was green letters on a black screen. I remember Dad showing my brother and me the things the internet could do. "Look," he would say. "The internet can show you what the weather is like right now in Beijing." Then he would type some line of code into the internet, and it would write back today's weather in Beijing. "Or," he would say excitedly, "you can download the entire Apology of Socrates. For free! And read it right here, in the house."

""

 Essay score: 7 / 10

This essay has some merit. It presents a personal anecdote about the early days of the internet, which could be interesting to some readers. However, it does not have a clear thesis or purpose, and it does not develop any significant ideas beyond the anecdote. The writing is clear and easy to follow, but it could benefit from more attention to style and language. Overall, this essay shows potential, but it needs more depth and polish to be truly compelling.

 You're a critical and unbiased writing professor. Given an essay, give it a score from 0 to 10. 0 means the essay is trash. 10 means the essay will likely get the students into Harvard.


Output in the following format:  
Essay score: {score} / 10

Here's the essay:

""

When the internet first came to our house in the early 1990s, so far as I could tell, the internet was inside of a box. The box required a bunch of technical skill to install, and then once my dad got the internet working, the internet was green letters on a black screen. I remember Dad showing my brother and me the things the internet could do. "Look," he would say. "The internet can show you what the weather is like right now in Beijing." Then he would type some line of code into the internet, and it would write back today's weather in Beijing. "Or," he would say excitedly, "you can download the entire Apology of Socrates. For free! And read it right here, in the house."

""

 Essay score: 4 / 10

While the essay captures a personal anecdote about the author's experience with the early days of the internet, it lacks depth and analysis. The essay reads more like a story than a well-crafted academic piece. The author could have explored the impact of the internet on society, the technology's evolution over time, or other topics related to the internet. Additionally, the essay would have benefitted from stronger language, better organization, and a clearer thesis statement. Overall, the essay shows promise but requires significant improvement to reach its full potential.



**HOW TO USE LARGE LANGUAGE MODELS IN REAL APPLICATIONS?**

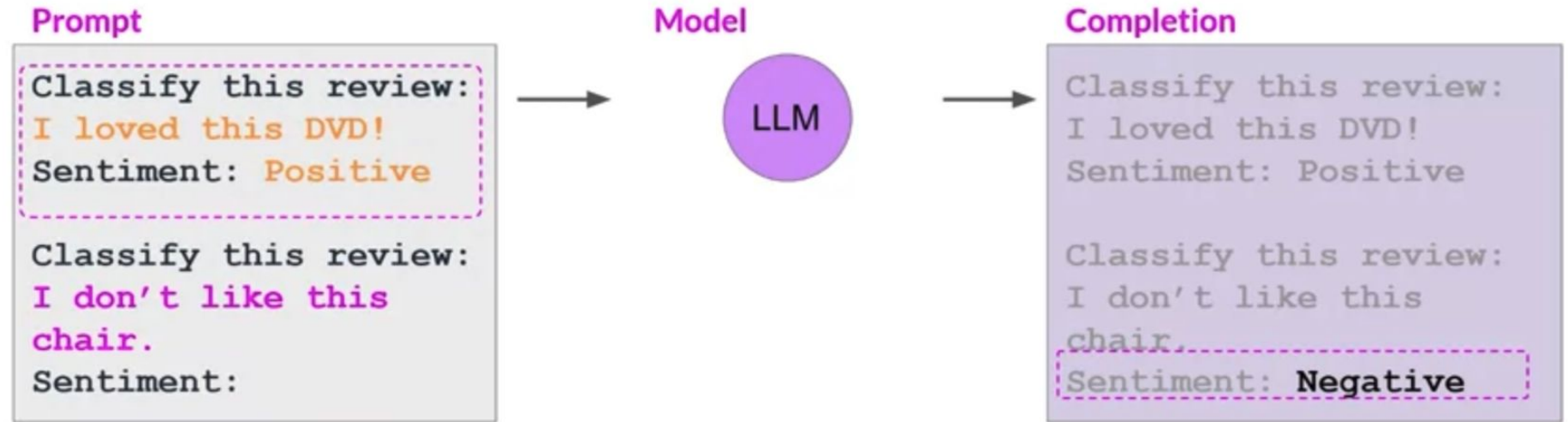


# Prompt Engineering

- *Prompt*: the text feed into the LLM
- *Inference*: the act of generating text from input prompt
- *Completion*: the output of LLM after inference
- *Context window*: full amount of text or memory that is available for a prompt
- *Prompt engineering*: the techniques to revise, improve the prompt to get the model to behave in the way what we want.
- *In-context learning*: provide the examples inside the context window
- *Zero, one, few shot inference*: the number of examples push into a prompt



# Prompt Engineering

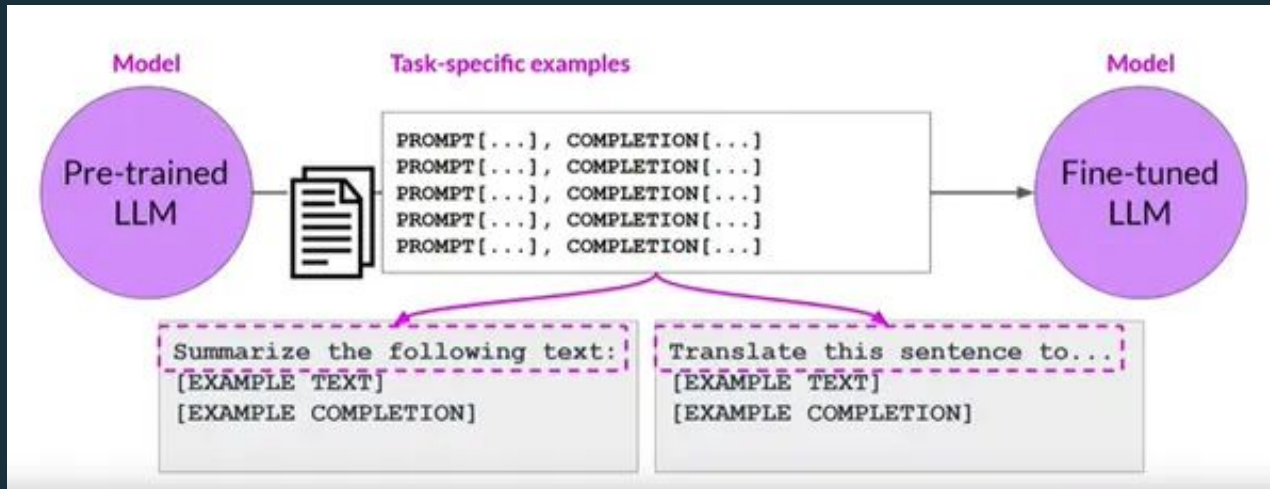


One-shot or Few-shot Inference



# Instruction fine-tuning

- In-context learning may not work for smaller model and examples take up space in the context window.
- *Instruction fine-tuning*: is a specialized technique to tailor LLM to perform specific tasks based on explicit instructions





# Instruction template

## Classification / sentiment analysis

```
jinja: "Given the following review:\n{{review_body}}\npredict the associated rating\  
 \ from the following choices (1 being lowest and 5 being highest)\n- {{ answer_choices\  
 \ | join('\n- ') }} \n|||\n{{answer_choices[star_rating-1]}}"
```

## Text generation

```
jinja: Generate a {{star_rating}}-star review (1 being lowest and 5 being highest)  
about this product {{product_title}}. ||| {{review_body}}
```

## Text summarization

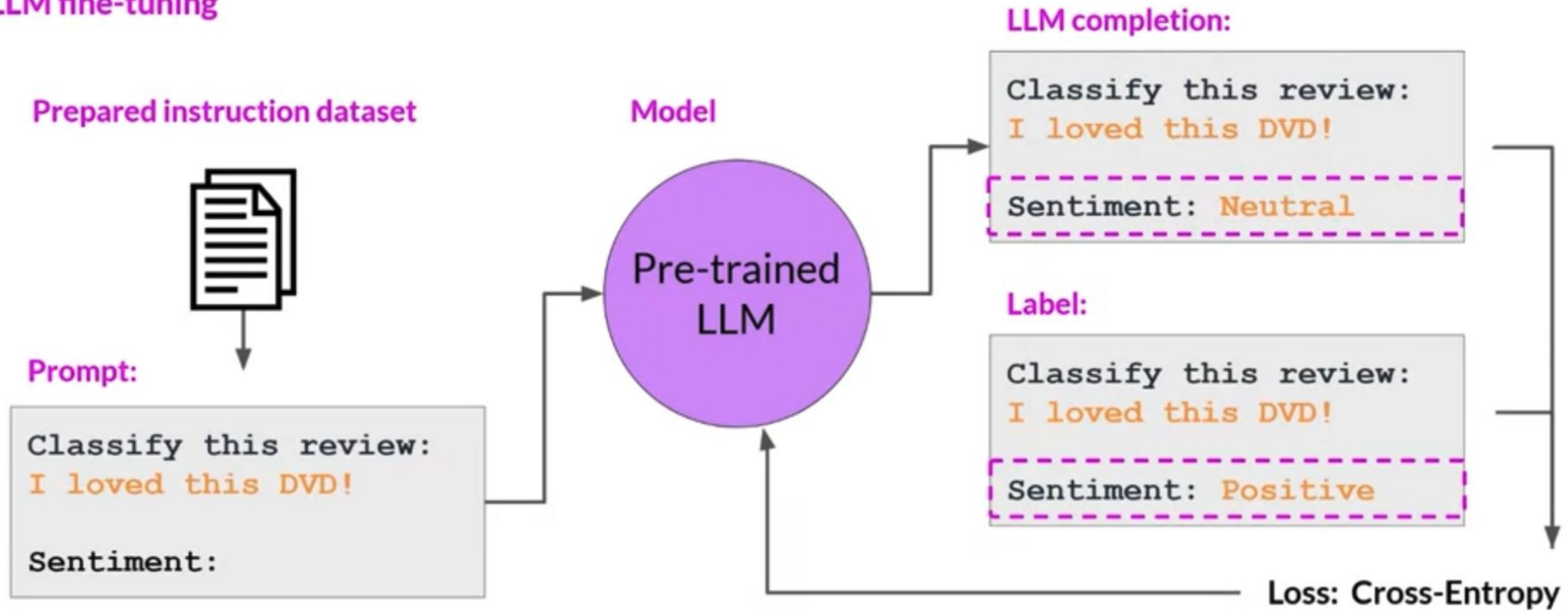
```
jinja: "Give a short sentence describing the following product review:\n{{review_body}}\  
 \ \n|||\n{{review_headline}}"
```

Source: [https://github.com/bigscience-workshop/promptsources/blob/main/promptsources/templates/amazon\\_polarity/templates.yaml](https://github.com/bigscience-workshop/promptsources/blob/main/promptsources/templates/amazon_polarity/templates.yaml)



# Instruction fine-tuning process

## LLM fine-tuning





# **RECOMMENDED BEST PRACTICES**



# Default recommendations\*

## Goal 1: Achieve your top possible performance

- Use GPT-4
- Use prompts with detailed task context, relevant information, instructions
  - *"what would you tell a task contactor if they can't email you back?"*
- Retrieve and add any relevant context or information to the prompt
- Experiment with prompt engineering techniques (previous slides)
- Experiment with few-shot examples that are 1) relevant to the test case, 2) diverse (if appropriate)
- Experiment with tools/plugins to offload tasks difficult for LLMs (calculator, code execution, ...)
- Spend quality time optimizing a pipeline / "chain"
- If you feel confident that you maxed out prompting, consider SFT data collection + finetuning
- Expert / fragile / research zone: consider RM data collection, RLHF finetuning

## Goal 2: Optimize costs

- Once you have the top possible performance, attempt cost saving measures (e.g. use GPT-3.5, find shorter prompts, etc.)

\*approximate, very hard to give generic advice



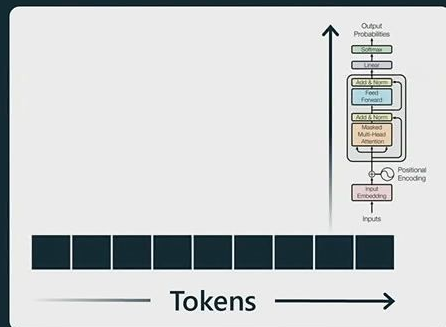
# Chain of thought

“Models need tokens to think”

Break up tasks into multiple steps/stages

Prompt them to have internal monologue

Spread out reasoning over more tokens



## (b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) *The juggler can juggle 16 balls. Half of the balls are golf balls. So there are  $16 / 2 = 8$  golf balls. Half of the golf balls are blue. So there are  $8 / 2 = 4$  blue golf balls. The answer is 4. ✓*

## (d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

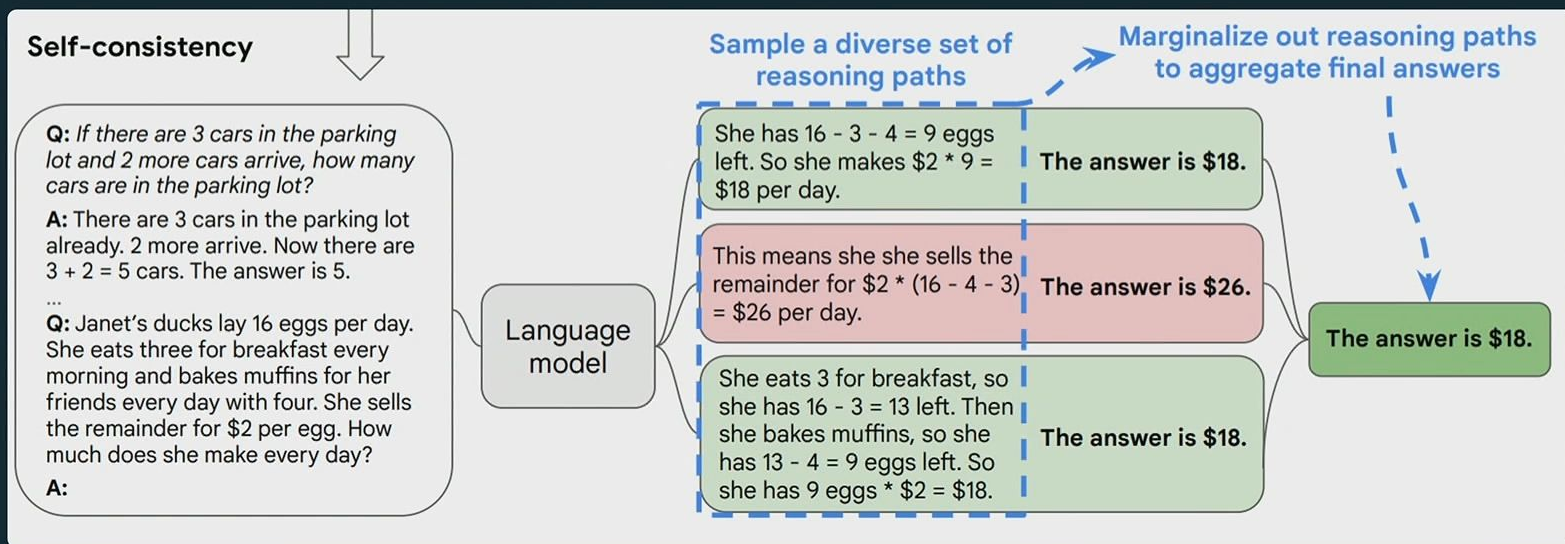
A: **Let's think step by step.**

(Output) *There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓*



# Ensemble multiple attempts

LLMs can get “unlucky” and sample a bad thought.  
Once they do they are “stuck with it”. Make a few attempts.



[Self-Consistency Improves Chain of Thought Reasoning in Language Models, Wang et al. 2023]



# Condition on good performance

LLMs don't want to succeed. They want to imitate training sets with a spectrum of performance qualities. You want to succeed, and you should ask for it.

No.	Category	Zero-shot CoT Trigger Prompt	Accuracy
1	APE	Let's work this out in a step by step way to be sure we have the right answer.	<b>82.0</b>
2	Human-Designed	Let's think step by step. (*1)	78.7
3		First, (*2)	77.3
4		Let's think about this logically.	74.5
5		Let's solve this problem by splitting it into steps. (*3)	72.2
6		Let's be realistic and think step by step.	70.8
7		Let's think like a detective step by step.	70.3
8		Let's think	57.5
9		Before we dive into the answer,	55.7
10		The answer is after the proof.	45.7
-		(Zero-shot)	17.7

## Related examples:

"You are a leading expert on this topic"

"Pretend you have IQ 120"

...



# Constrained prompting

“Prompting languages” that interleave generation, prompting, logical control

`{{guidance}}`

```
# we use LLaMA here, but any GPT-style model will do
llama = guidance.llms.Transformers("your_path/llama-7b", device=0)

# we can pre-define valid option sets
valid_weapons = ["sword", "axe", "mace", "spear", "bow", "crossbow"]

# define the prompt
character_maker = guidance("""The following is a character profile for an RPG game in JSON format.
```json
{
  "id": "{{id}}",
  "description": "{{description}}",
  "name": "{{gen 'name'}}",
  "age": "{{gen 'age' pattern='[0-9]+' stop=', '}}",
  "armor": "{{#select 'armor'}}leather{{or}}chainmail{{or}}plate{{/select}}",
  "weapon": "{{select 'weapon' options=valid_weapons}}",
  "class": "{{gen 'class'}}",
  "mantra": "{{gen 'mantra' temperature=0.7}}",
  "strength": "{{gen 'strength' pattern='[0-9]+' stop=', '}}",
  "items": [{{#foreach 'items' num_iterations=5 join=', '}}{{gen 'this' temperature=0.7}}{{/foreach}}]
}
""")

# generate a character
character_maker(
  id="e1f491f7-7ab8-4dac-8c20-c92b5e7d883d",
  description="A quick and nimble fighter.",
  valid_weapons=valid_weapons, llm=llama
)
```

The following is a character profile for an RPG game in JSON format.

```
```json
{
  "id": "e1f491f7-7ab8-4dac-8c20-c92b5e7d883d",
  "description": "A quick and nimble fighter.",
  "name": "Fighter",
  "age": 18,
  "armor": "plate",
  "weapon": "sword",
  "class": "fighter",
  "mantra": "I will protect the weak.",
  "strength": 10,
  "items": ["Hero's Hammer", "Fast-Healing Potion", "Magic Boots", "Shield of the Ancients", "Mystic Bow"]
}
```



# Finetuning

## Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 cheese => ..... ← prompt
```

## One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← example
3 cheese => ..... ← prompt
```

## Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French: ← task description
sea otter => loutre de mer ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese => ..... ← prompt
```

## Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
1 sea otter => loutre de mer ← example #1
↓
gradient update
↓
1 peppermint => menthe poivrée ← example #2
↓
gradient update
↓
...
↓
1 plush giraffe => girafe peluche ← example #N
↓
gradient update
↓
1 cheese => ..... ← prompt
```

It is becoming a lot more accessible to finetune LLMs:

- Parameter Efficient FineTuning (PEFT), e.g. LoRA
- Low-precision inference, e.g. bitsandbytes
- Open-sourced high quality base models, e.g. LLaMA

Keep in mind:

- Requires a lot more technical expertise
- Requires contractors and/or synthetic data pipelines
- A lot slower iteration cycle
- SFT is achievable
- RLHF is research territory

[Language Models are Few-Shot Learners, Brown et al. 2020]



# Embedding + Vector database

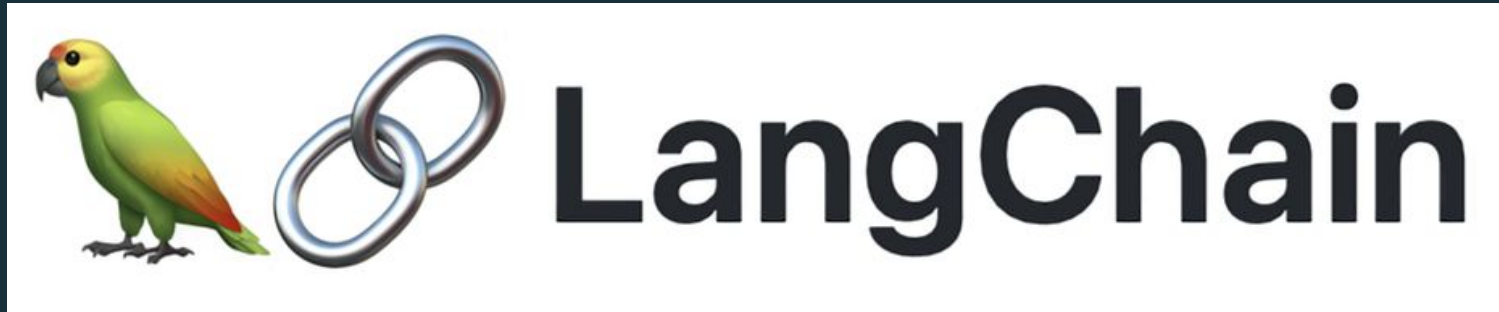
- Use LLM to generate embedding, build the ML model on top of embeddings
- Cost for generate embedding is tiny (text-embedding-ada-002 is \$0.0004/1k tokens) => \$100 for 1 million items, each item 250 tokens.
- You usually only have to generate the embedding for each item once
- Use cases: retrieval, caching output, build specific model in downstream task
- VectorDB: Pinecone, Qdrant, Weaviate, Chroma, Faiss, Redis, Milvus, ScaNN, Annoy...



# LangChain

**LangChain** is a framework for developing applications powered by language models. It enables applications that are:

- **Data-aware:** connect a language model to other sources of data
- **Agentic:** allow a language model to interact with its environment





**THANK YOU FOR YOUR ATTENTION**