
Pretraining and post-training

Key takeaways

1. Pretraining runs Thursday's loop on trillions of tokens of web text and code. The result continues documents; it does not answer you.
2. Post-training runs the same loop and loss on about a million curated conversations. The data is what makes the model act like an assistant.
3. More is not better by itself. At fixed compute, which data you keep matters more than how much you had.

Common misunderstandings of what a model is trained on

Which claim would you defend right now? Choose before the evidence.

“More training always makes the model better.”

“A pretrained model is already a chatbot; the chat interface is only a wrapper around it.”

“Filtering the web throws away knowledge, so the best dataset is all of it.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

The loop learns $f: x \rightarrow y$ from $(x, y) \sim D$

REPRISE

Thursday fixed the algorithm. Today's question is D , the data the examples (x, y) are drawn from: the same loop fed different data makes a different model.



Before 2019: a dataset per task

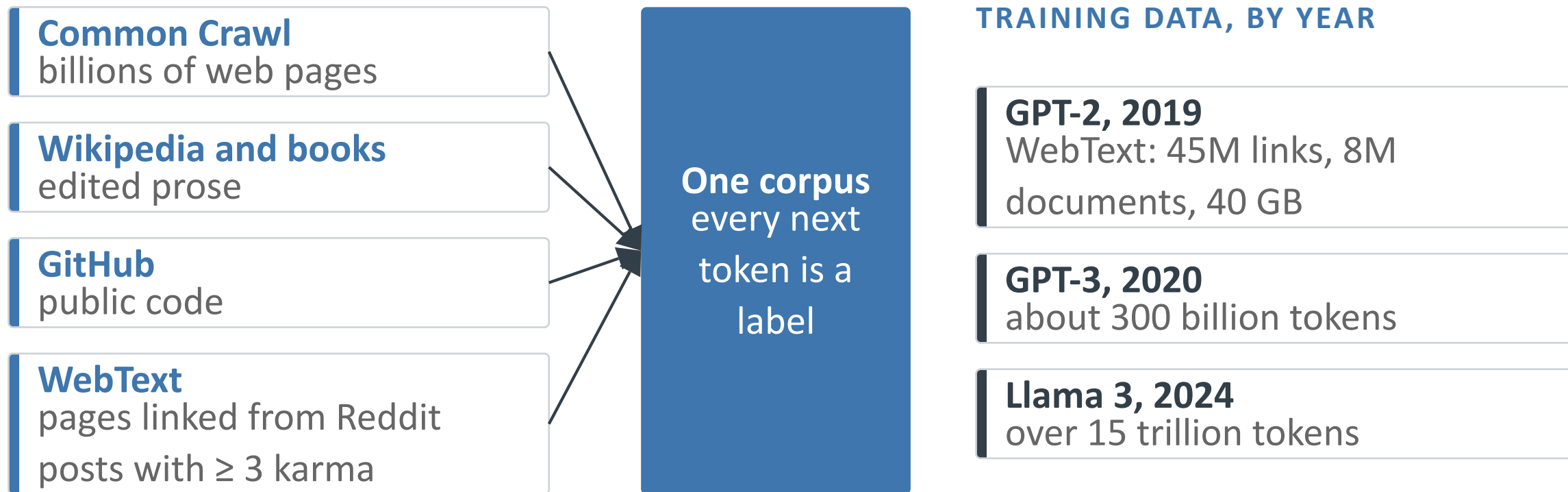
Each task got its own labeled dataset and its own model, and a person produced every label.

Task	Dataset and size	Who made each label
Sentiment	IMDB: 50,000 movie reviews	the reviewer's star rating, as positive or negative
Reading comprehension	SQuAD: about 100,000 questions on Wikipedia	crowdworkers wrote each question and marked its answer
Translation	WMT: millions of sentence pairs	human translators
Topic classification	Reuters-21578: 21,578 news stories	Reuters staff tagged each story's topics

Each model could do one thing, and only after people labeled examples of it.

After 2019: everything written down is data

Next-token prediction on web text transfers to tasks nobody labeled. The label is the next token, so every page is training data.



Instapoll

Instapoll · Review · Nov 12 A

A model has been pretrained on trillions of tokens of web text and code, and nothing else has been done to it. You type “Write a Python function that reverses a string.” and let it generate. What happens most often?

- A. It answers you the way an assistant would: acknowledges the request, writes the function, and explains it
- B. It continues your line as if it were part of a document: a tutorial page, documentation, or code that then stops
- C. It produces nothing useful, because without instructions it has no way to start an answer
- D. It writes the function, but in whichever programming language was most common in its training data

A base model continues the document

Nothing in its data was a conversation with you, so a base model writes what a page would hold next: documentation, or a tutorial. Neither replies to you.

GPT-2 (2019): web text only

Write a Python function that reverses a string.

This is useful for reversing a string from one character to another, or for reverse-complementary operations. It can also be used to reverse a string that contains a punctuation character.

...

Qwen2.5 base (2024)

Write a Python function that reverses a string.

```
def
reverse_string(string):
    return string[::-1]
```

```
reverse_string('hello')
<|endoftext|>
```

Qwen2.5-Instruct

Certainly! Below is a Python function that reverses a given string:

```
```python
def reverse_string(s):
 """
 Reverses the input
 string.
```

...

Only the post-trained model replies: it addresses you, then answers.

# What the base model is missing

This is not ChatGPT or Claude yet. The model has never seen a dialogue with turns, a place to stop, or a reader waiting for an answer.

## What pretraining gave it

### Language and code

grammar, idioms, APIs, many languages

### Facts and formats

what documents, lists, and answers look like

### The shape of an answer

from forum threads and Q&A pages

## What it never saw

### Who is speaking

no user turn, no assistant turn

### When to stop

a web page just keeps going

### What you want

that a request expects a response, and in what tone

# Post-training, step 1: instruction tuning

Wrap each conversation in a chat template whose special tokens mark the turns, then run the same loss on the reply: an answer, then a stop token.

```
<|begin_of_text|>
```

```
<|start_header_id|>user<|end_header_id|>
```

```
Write a Python function that reverses a string.
```

```
<|eot_id|>
```

```
<|start_header_id|>assistant<|end_header_id|>
```

```
def reverse(s): return s[::-1]
```

```
<|eot_id|>
```

Prompt: the model reads these tokens, and no loss is computed on them

**Target: next-token loss on every token, the stop token included**

InstructGPT (2022): about 13,000 demonstrations written by hired labelers. DeepSeek-V3 (2024): 1.5 million examples, many generated by models and then filtered.

# Preferred behavior is not the same as scale

Human raters preferred the 1.3B InstructGPT to the 175B GPT-3. Post-training changed what the model was optimized for, not how much it knew.

System	Parameters	Training emphasis	Observed preference
GPT-3	175B	next-token pretraining	less preferred
InstructGPT	1.3B	instruction tuning + human-feedback RL	more preferred

- A model 130× smaller was preferred: post-training changed what it was optimized for.
- This is about helpfulness on one prompt distribution, not capability in general.
- Parameter count and conversational polish are both poor predictors of task performance.

# Same weights, same loss, different data

Pretraining learns the world from trillions of tokens. Post-training learns to behave from far fewer curated examples.

	Pretraining	Post-training (instruction tuning)
<b>Data</b>	web pages, books, code	conversations in a chat template
<b>Size</b>	trillions of tokens	thousands to millions of examples
<b>Label</b>	the next token in the text	an ideal reply someone wrote
<b>Run length</b>	weeks to months	a small fraction of that
<b>It learns</b>	language, code, and facts	turns, answering, stopping

The same next-token machinery runs in both. The data decides what the model becomes.

# More data is not automatically better

At fixed compute, which data you keep beats how much you had, and repeating data past a few passes stops paying.

**CURATION: SAME MODEL AND COMPUTE,  
DIFFERENT DATA**

ImageNet zero-shot accuracy of a CLIP ViT-L/14 trained with OpenAI's recipe and compute:

OpenAI's own 400M pairs  75.5%

1.4B kept of a 12.8B pool  79.2%

DataComp kept 11% of its pool and won:  
stricter subsets trained better models.

**REPETITION:  
PASSES OVER THE SAME DATA**

**Up to about 4 passes**  
nearly as good as fresh data

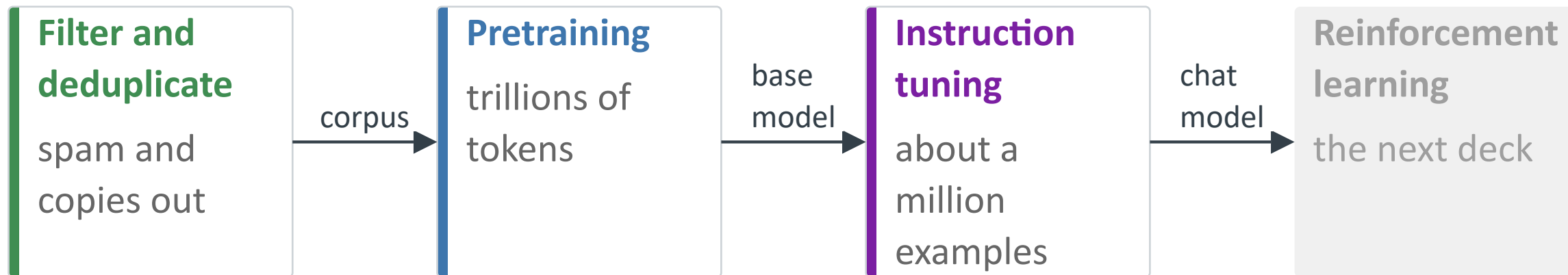
**Many more passes**  
each extra pass adds less, toward nothing

**Post-training, too**  
tuning hard on a narrow set costs skills  
elsewhere

This is why curation, filtering, and deduplication are where the work is.

# The pipeline, end to end

Raw web pages become a filtered corpus, then a base model, then a chat model, then the model in your terminal. Each arrow carries what the next stage consumes.



Pretraining, instruction tuning, and reinforcement learning all update the same weights with the same loop. Only D changes.

# Key Takeaways

1. Pretraining runs the next-token loss on everything written down, trillions of tokens for months, because the data labels itself. The weights absorb the statistics of the web, so the model continues documents and has no notion of being asked anything.
2. Post-training runs the same loss on about a million conversations in a chat template, so the model learns that an assistant turn follows a user turn and ends with a stop token. Same weights, same loop: the data is the difference, and the InstructGPT result is what it buys.
3. More is not better by itself. DataComp's filtered 1.4B subset beat OpenAI's data at the same compute, repeated data pays for only a few passes, and over-tuning narrows a model. Curation, filtering, and deduplication are where the work is.

# Curate a batch

10 min

- In pairs: you may keep four documents for pretraining. Choose, and write one filter rule that would select them automatically.
- What would your rule wrongly drop from the whole web? Name one example.
- Pick two documents that could become instruction-tuning pairs, and write the (instruction, response) for one of them.
- Compare with a neighbor: did you keep the Reddit argument? Defend either answer.

## EIGHT CANDIDATE DOCUMENTS

- a** A Stack Overflow answer with working code
- b** An SEO page repeating “cheap flights” 40 times
- c** A Wikipedia paragraph on the Krebs cycle
- d** The MIT license, in countless repositories
- e** A page of an open linear algebra textbook
- f** A Reddit argument about a referee’s call
- g** A broken machine translation from Japanese
- h** A GitHub README: install steps, an example