CS590-06 Building Intelligent Agents with Frontier Models

Lecture 2: Pretraining and Scaling Law

Shuyan Zhou

2025-08-28

Lecture overview

In this lecture, we will cover

- A brief background of language modeling
- Generative pretraining
- Scaling law
- In context learning
- Open and close model
- Copyright and licensing

Treating text as a sequence of words

- Machines don’t “see” words, only numbers.
- The simplest approach: represent text as a **bag of words (BOW)**.
- Each word is a feature — a signal saying *“this word is present.”*
- By combining word features with weights, we can start making predictions.

Word	Lookup Vector
I	[0, 0, 1, 0, 0, …]
hate	[0, 1, 0, 0, 0, …]
this	[0, 0, 0, 1, 0, …]
movie	[0, 0, 0, 0, 1, …]

Feature × feature weights = score

Word	Lookup Vector
I	[0, 0, 1, 0, 0, …]
hate	[0, 1, 0, 0, 0, …]
this	[0, 0, 0, 1, 0, …]
movie	[0, 0, 0, 0, 1, …]

- summed vector (V) = [0, 1, 1, 1, 1, …]
- weights (W) = [w_1, w_2, w_3, w_4, w_5, …]
- score (s) = $V \times W^T$

Example: Sentiment classification

Binary classification: Each word has a single scalar, positive indicating "yes" and negative indicating "no"

Multi-class classification: Each word has its own N elements corresponding to difference classes, such as [very good, good, neutral, bad, very bad]

Binary

word	score
love	2.4
hate	-3.5
nice	1.2
no	-0.2
dog	-0.3
…	…

Multi-class

word	v.pos	pos	neu	neg	v.neg
love	2.4	1.5	-0.5	-0.8	-1.4
hate	-3.5	-2.0	-1.0	0.4	3.2
nice	1.2	2.1	0.4	-0.1	-0.2
no	-0.2	0.3	-0.1	0.4	0.5
dog	-0.1	0.3	0.6	0.2	-0.2
…	…	…	…	…	…

Simple training of BOW models

Structured perceptron

```
feature_weights = {}
for x, y in data:
    # Make a prediction
    features = extract_features(x)
    predicted_y = run_classifier(features)
    # Update the weights if the prediction is wrong
    if predicted_y != y:
        for feature in features:
            feature_weights[feature] = (
            feature_weights.get(feature, 0) +
            y * features[feature]
 )
```

What's missing in BOW?

- Handling of conjugated or compound words
  - I **love** this move ⟶ I **loved** this movie
- Handling of word similarity
  - I **love** this move ⟶ I **adore** this movie
- Handling of combination features and sentence structure
  - I **love** this movie ⟶ I **don't love** this movie
  - It has an interesting story, **but** is boring overall.

Each deficiency can be fixed

- Handling of word compositions
  - I **love** this move ⟶ I **loved** this movie

✅ Subword models

- Handling of word similarity
  - I **love** this move ⟶ I **adore** this movie

✅ Continuous word embeddings

- Handling of combination features and sentence structure
  - I **love** this movie ⟶ I **don't love** this movie
  - It has an interesting story, **but** is boring overall.

✅ Sequence modeling with neural networks

Subword models: Basic idea

- Split less common words into multiple subword tokens

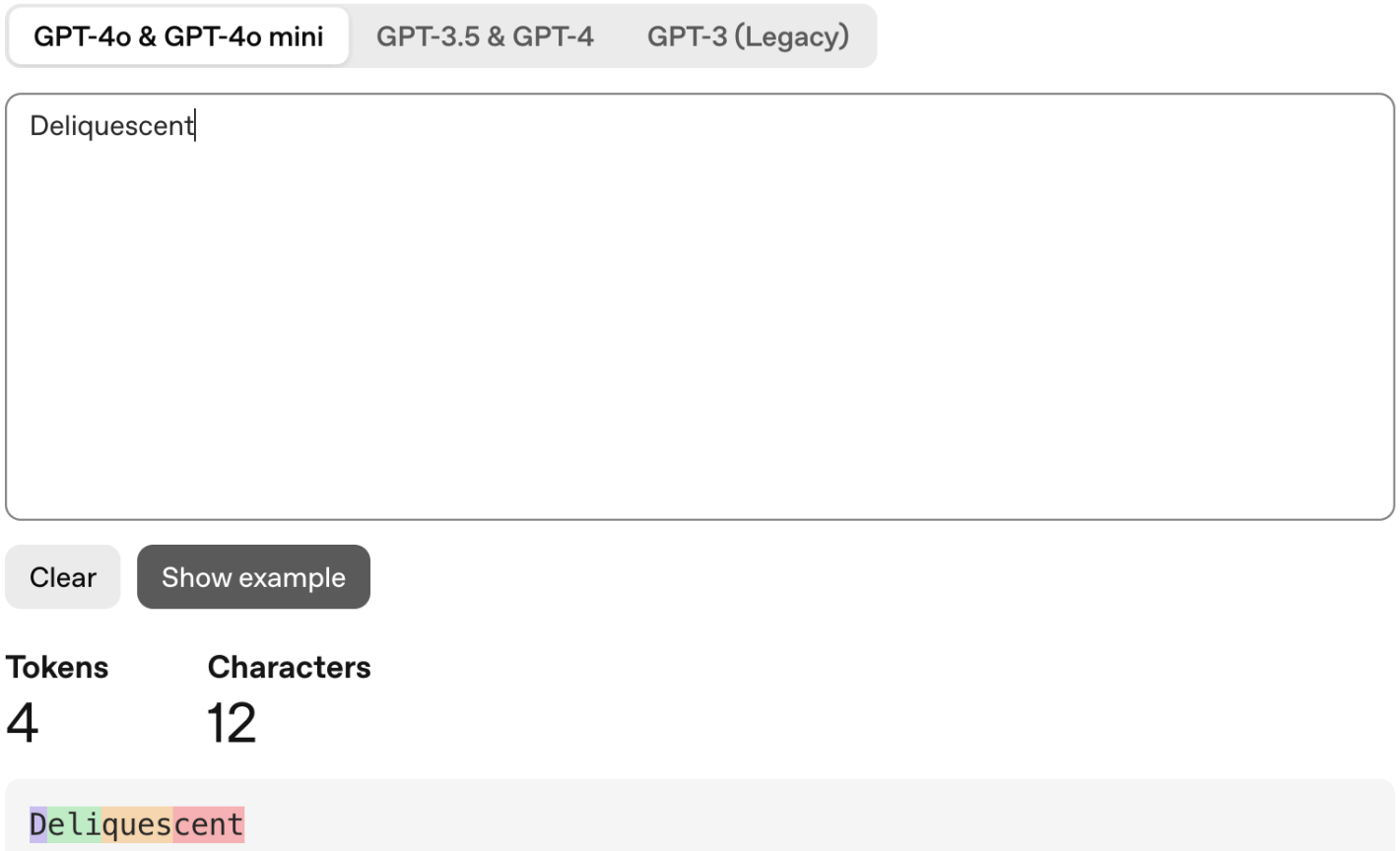
the companies are expanding

↓

the **compan** **_ies** are **expand** **_ing**

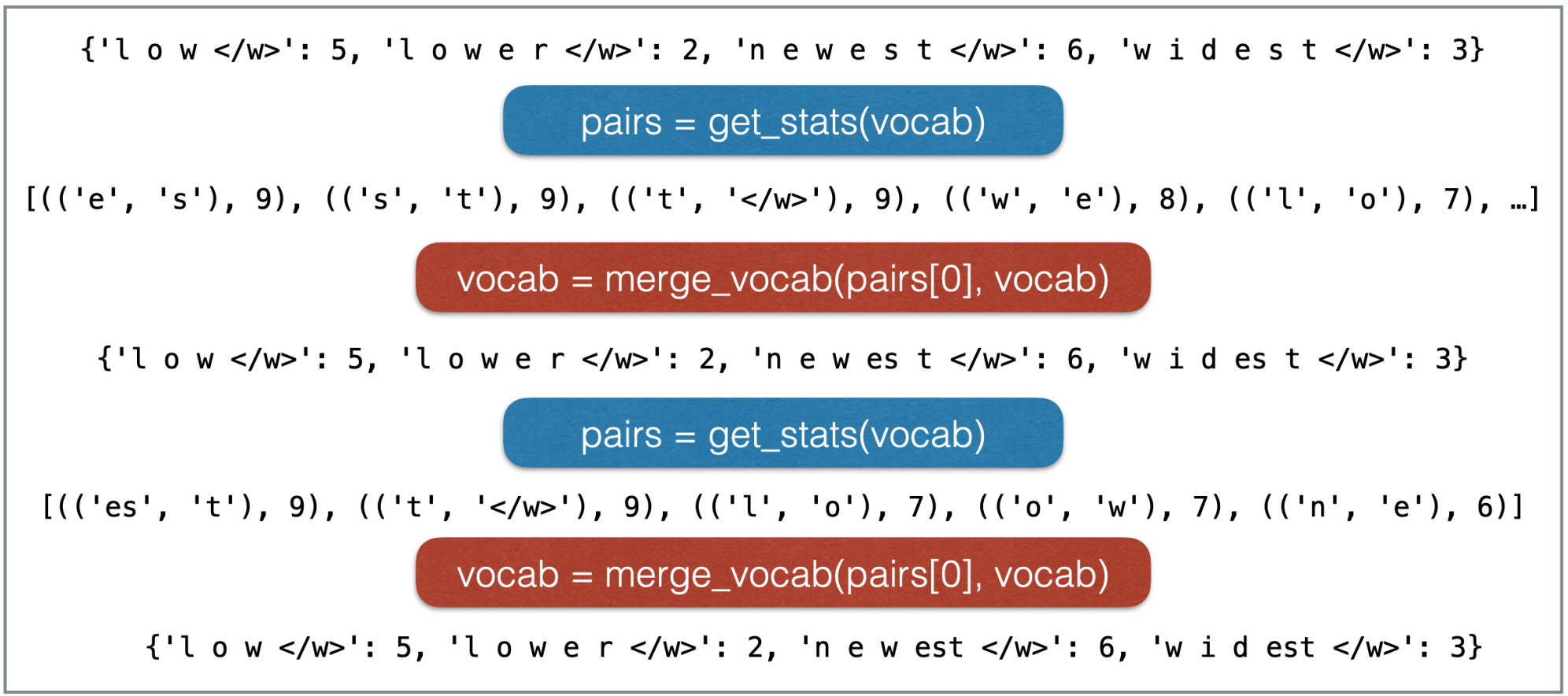
- Share subword representations across similar words
  - **expand**ing and **expand**ed share the same prefix
- Reduce vocabulary size, hence the parameter size
- Improve handling of rare words
  - English headwords: ~795k
  - LLama 3 model vocabulary: ~128k

Example tokenization



Constructing the subword collection

Byte pair encoding (BPE, Sennrich et al. 2015): Starting from characters, incrementally combine together the most frequent token pairs



Software: SentencePiece

```
spm_train
  --input=<input> \
  --model_prefix=<model_name> \
  --vocab_size=8000 \
  --character_coverage=1.0 \
  --model_type={bpe|unigram|character}
```

Subword Considerations

- **Multilinguality**:
  - Subword models struggle **across languages**
    - e.g., English **“information”** may stay whole, while Hungarian **“információ”** gets over-segmented into `in + for + má + ci + ó`
  - **Programming languages**: Formatting and symbols are semantically crucial
    - Indentation defines scope in Python
      - `␣␣␣␣print()` ≠ `print()`
    - Operators and punctuation must remain atomic
      - `==` ≠ `=` + `=`, `10:` ≠ `10 + :`
  - Workaround: upsample less represented languages
- **Arbitrariness**: Boundaries can be unstable (`es t` vs `e st`)
  - Workaround: Subword regularization samples multiple valid segmentations during training for robustness (Kudo 2018)

From one-hot to continous word embedding

- Previously: a word is represented as a one-hot vector with 1 in the position corresponding to the word and 0s elsewhere.

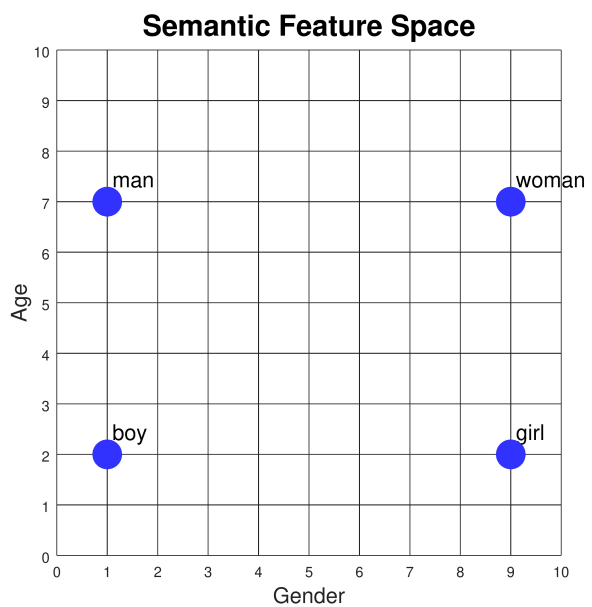
  ```
  boy = [0, 1, 0, 0, 0]
  ```

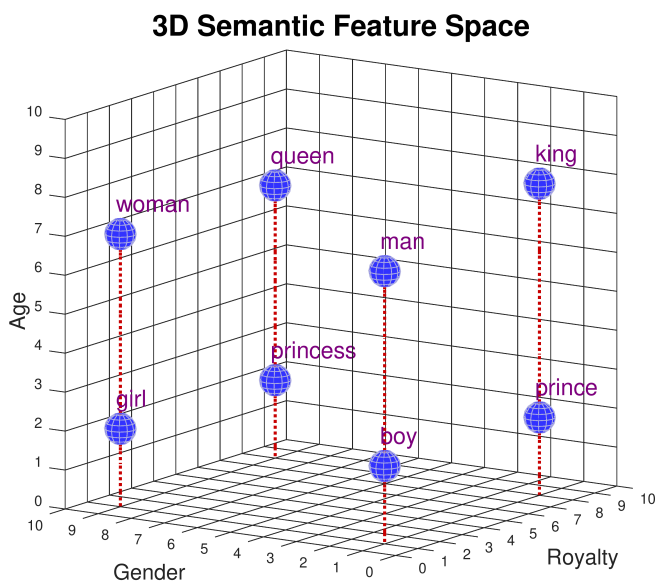
- Continuous word embedding look up a **dense** vector

  ```
  boy = [0.1, 0.5, 0.3, 0.2]
  ```

Expected characteristics of word embeddings

- Words that are similar (syntactically, semantically, same language, etc.) are close in vector space, and vice versa.
- Each element in the vector represents a different **feature** of the word (e.g., part of speech, sentiment, etc.)





Language modeling

- Word embeddings let us represent words in a continuous space
- But language is **sequential**: words depend on their **context**
- Goal: build models that can **predict the next word** given previous ones

Classical approach: n-gram models

- Simplest way: approximate context with the last *n* words
- Estimate probability from counts in a corpus

$$P(w_i \mid w_{i-n}, \ldots, w_{i-1}) = \frac{C(w_{i-n}, \ldots, w_{i-1}, w_i)}{C(w_{i-n}, \ldots, w_{i-1})}$$

$$P(\text{movie} \mid \text{I love this}) = \frac{C(\text{I love this movie})}{C(\text{I love this})}$$

Challenges of n-gram models

- Data sparsity: many possible sequences are never seen
- Memory explosion: storing large tables of counts
- Limited context: fixed n → cannot capture long-range dependencies
- Workarounds: smoothing, back-off, interpolation

Continuous word embeddings + neural networks

- Word embeddings map discrete tokens into **dense vectors**
- Neural networks take these vectors as input → learn patterns in context
- Context window can be modeled flexibly, not just fixed *n*
- Predictions become **generalizable** (similar contexts → similar next-word probabilities)

Example: `I love this movie, this review is`

1. Tokenize the sentence
2. Embed each token into a dense vector
3. Run through the model to get a probability distribution
4. *[Math Processing Error]* `positive`

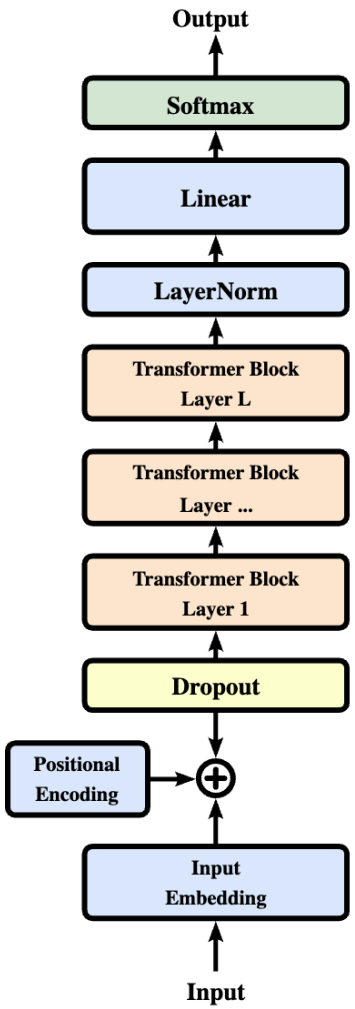
The powerful unsupervised training: next-token prediction

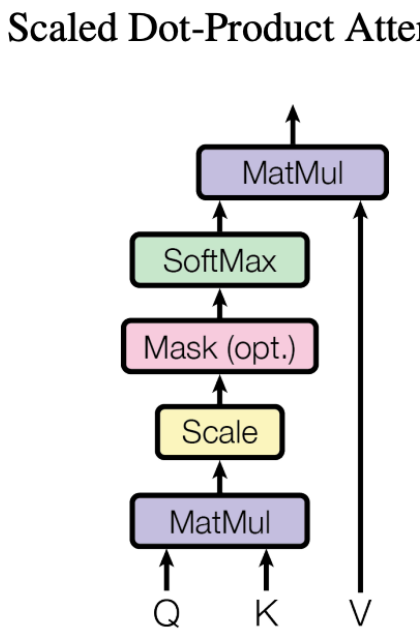
- Training objective: maximize likelihood of observed sequences (next-word prediction)
  - Minimize the negative log-likelihood

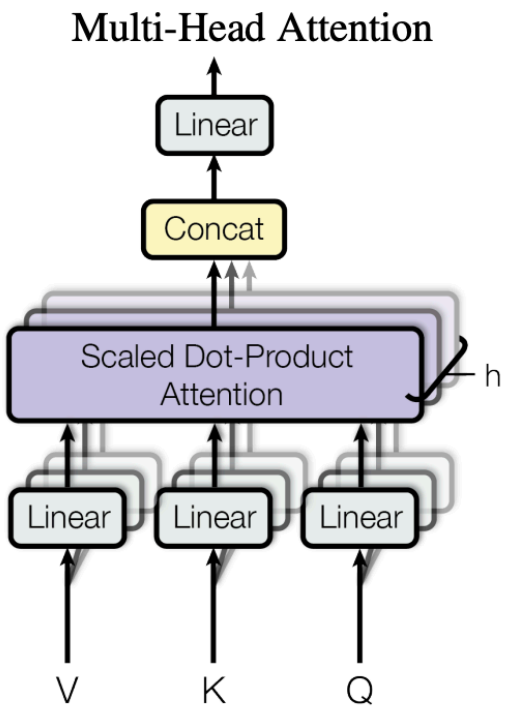
$$\min_{\theta} \sum_{x \in D_{\text{train}}} \sum_{t} -\log p_\theta(x_t \mid x_{<t})$$

- θ: model architecture and size
- D_{train}: data

Transformer [Vaswani et al 2017]







[Math Processing Error]

Recap: A brief background of language modeling

The data source

$$\min_{\theta} \sum_{x \in D_{\text{train}}} \sum_{t} -\log p_\theta(x_t \mid x_{<t})$$

- **Public**: Large-scale datasets from the internet (e.g., Common Crawl, Wikipedia)
- **Private**: Domain-specific corpora (e.g., legal documents, medical records)
- **Synthetic**: Generated data (e.g., using language models to create training examples)

Example: Llama 1 pretraining 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

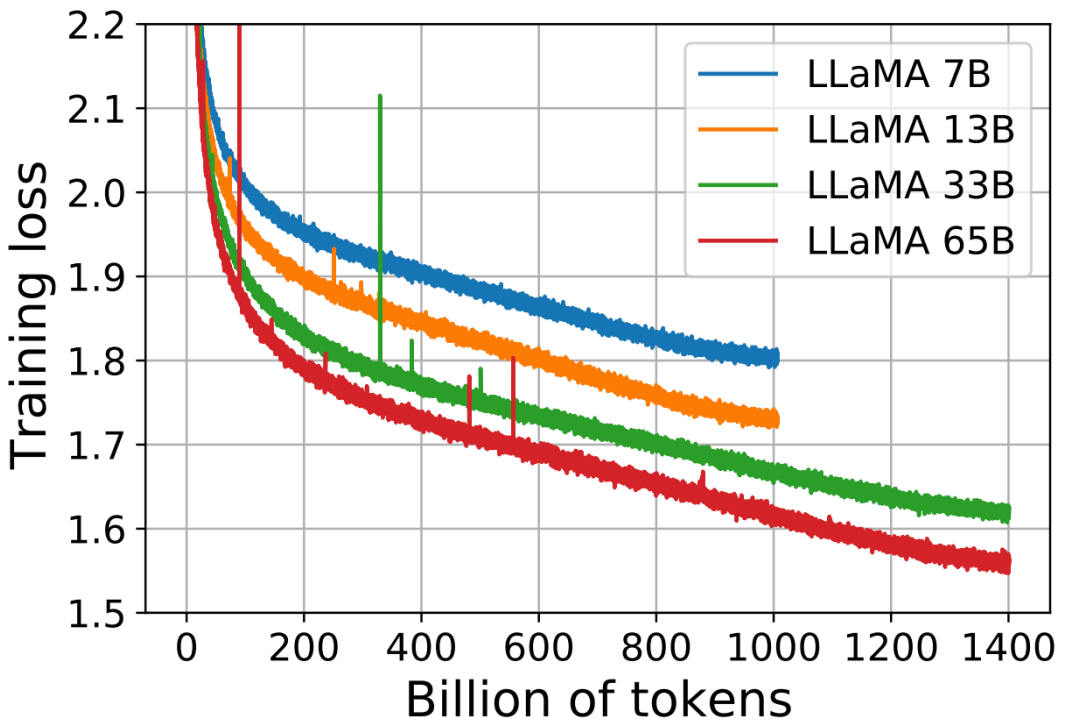
Table 1: **Pre-training data.** Data mixtures used for pre-training, for each subset we list the sampling proportion, number of epochs performed on the subset when training on 1.4T tokens, and disk size. The pre-training runs on 1T tokens have the same sampling proportion.

1.4 trillion tokens! (wikipedia ~ 10 billion tokens)

Pretraining model evaluation

- Loss (train, validation, test)
  - Cross entropy loss: $L = -\frac{1}{N} \sum_{t=1}^{N} \log p_\theta(x_t \mid x_{<t})$
  - Comparison across models with the same vocabulary/logits
- Perplexity
  - $\mathrm{PPL} = \exp(L)$
  - Intuition: PPL ≈ “average number of equally likely choices”
    - Lower PPL → model is more confident & accurate
    - Example:
      - PPL = 10 → model acts like it chooses among 10 options each step
      - PPL = 100 → much less certainty
- Few-shot prompting (approximate downstream performance)
- End-to-end evaluation by post-training on different pretrained models (cover in the next lecture)

Example: Llama training loss



- Bigger model ⟶ better loss
- More data ⟶ better loss

Recap: Generative pretraining

- Minimize negative log-likelihood of next-token prediction on large text corpora
- Evaluation with many metircs, with loss being the most common and important one
- So, how to get a better pretrained model (i.e. lower loss)?

The bitter lesson

> The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. – Richard Sutton, 2019

Goal: Get a better pretrained model by increasing computation

Compute in language model training

- Compute is spent by performing forward and backward passes during training.
- e.g., An approximate formula for total training compute for transformer-based LM
  $C \approx 6ND$
  - N: number of model parameters
  - D: number of training tokens
  - C: total training compute in FLOPs

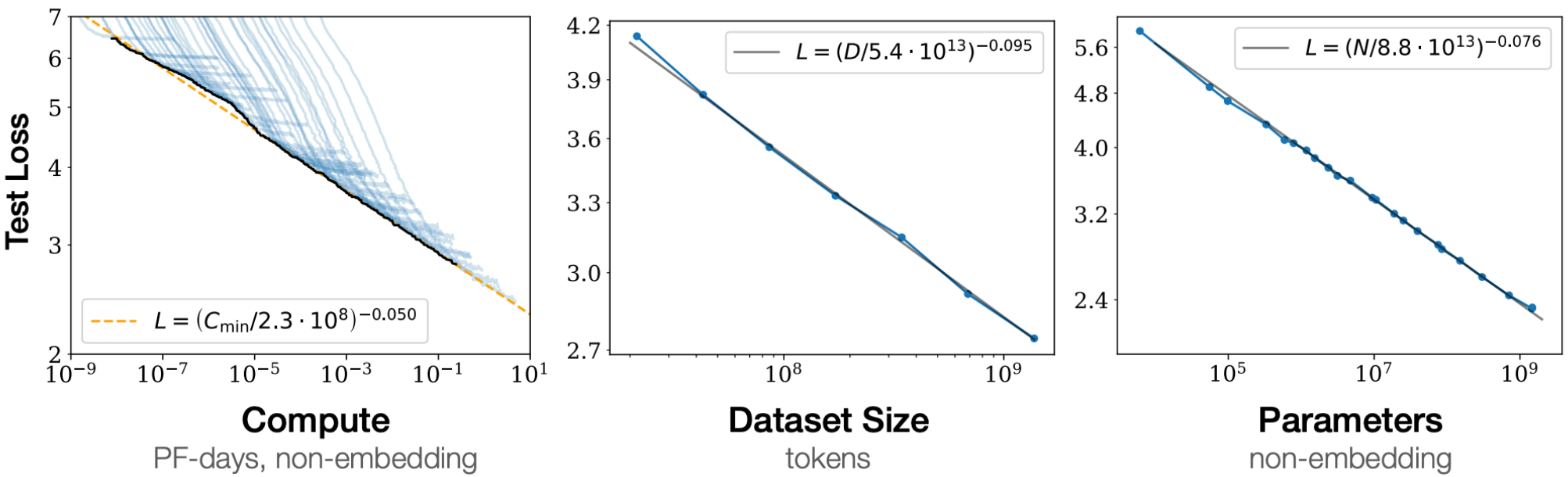
Scaling up the compute

We can increase compute by

- Increase the number of model parameters N
- Increase the number of training tokens D
- Or combination of both

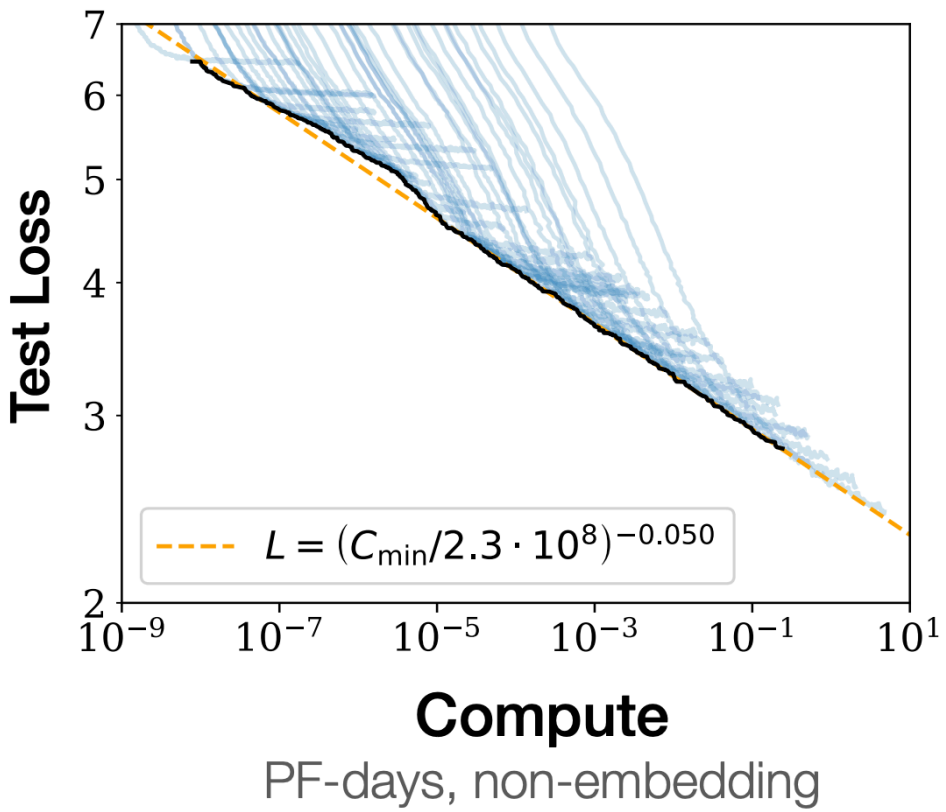
Empirical validation: Scaling law

Language modeling loss predictably improves as a power-law with increased compute



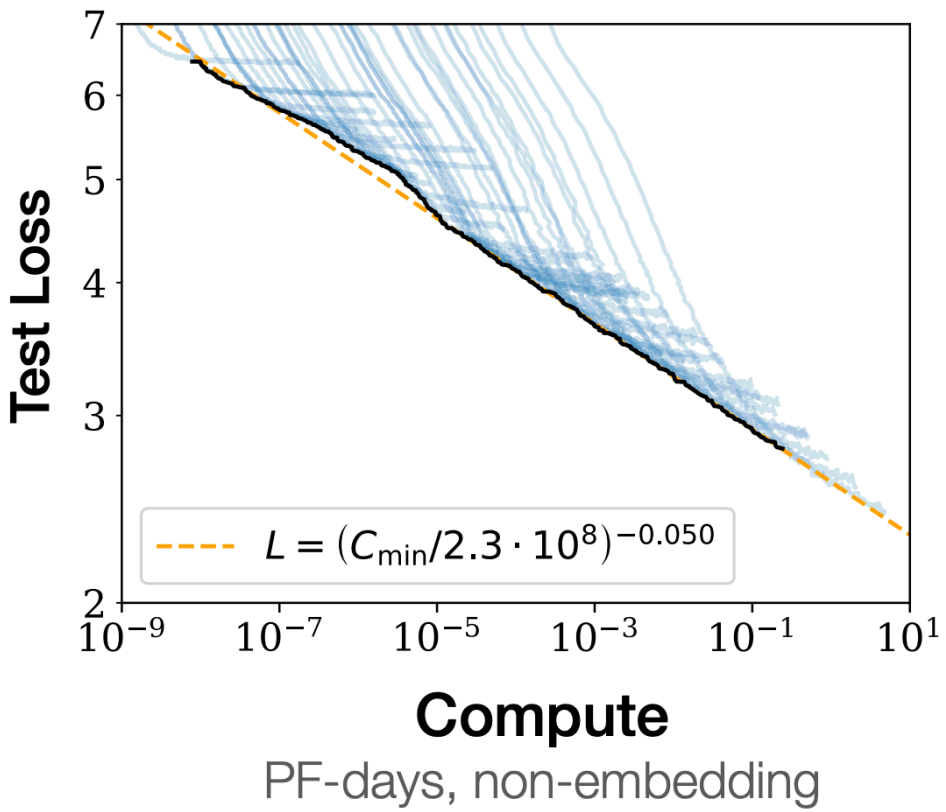
(Kaplan et al 2020)

Deriving the scaling law curve



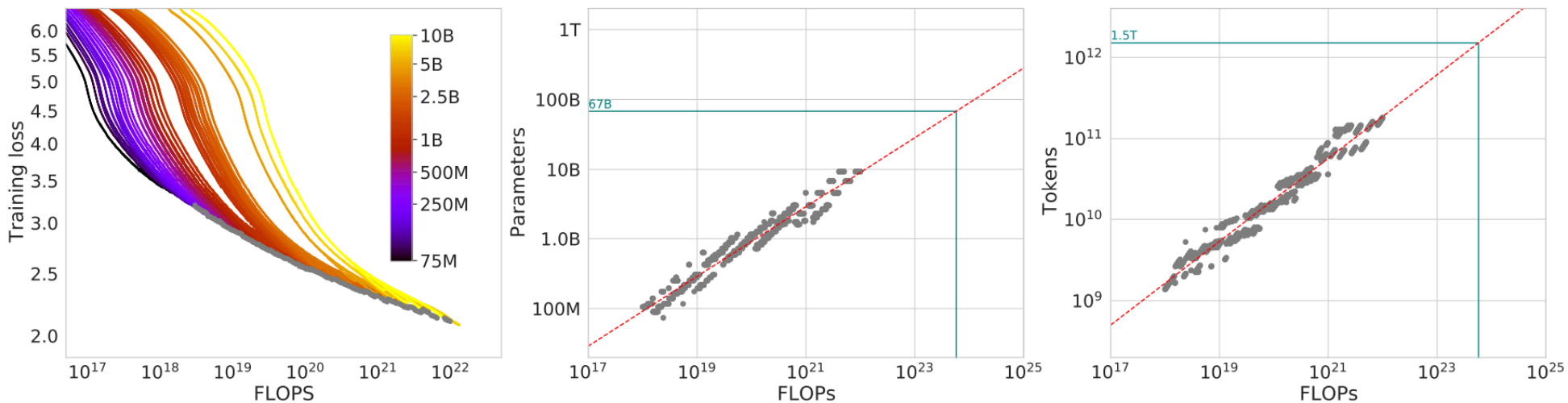
- Train multiple models with varying model sizes N and dataset sizes D
- Record the loss at different compute budgets (log scale) during training
- Select the lowest loss achieved at each compute budget
- Fit a linear regression on these (loss, compute) points to obtain the scaling law trend

Deriving the scaling law curve



- **Compute optimal**: black curve
- **Scaling law**: orange curve (e.g., $L(C) \propto 1/C^{a}$)

Using scaling law to guide compute allocation



- Run many experiments and extract the envelope of minimal loss per FLOP
- Fit the scaling law curve to predict the optimal model size / data size given a compute budget
- e.g., Chinchila scaling law found that when increasing the model size by x, the data size should roughly increase by x too

Stronger performance even with smaller models

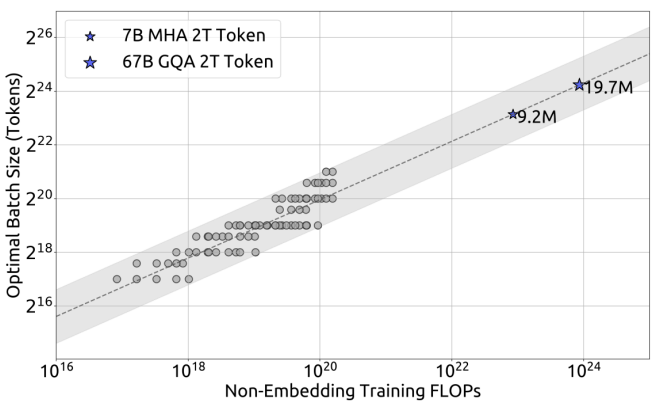
280B
not compute optimal
(under-trained)

70B
compute optimal

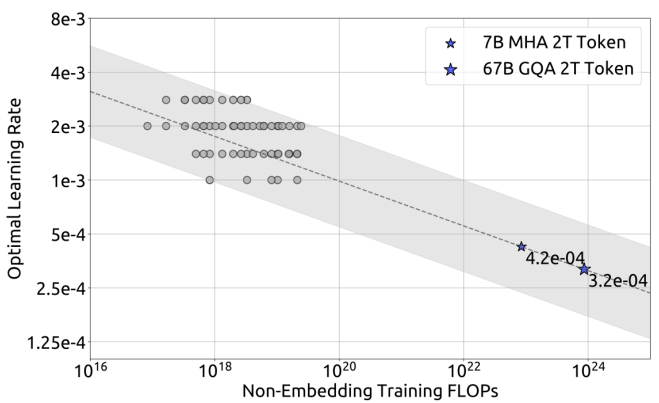
Random	25.0%
Average human rater	34.5%
GPT-3 5-shot	43.9%
Gopher 5-shot	60.0%
***Chinchilla* 5-shot**	**67.6%**
Average human expert performance	*89.8%*
June 2022 Forecast	57.1%
June 2023 Forecast	63.4%

Using scaling law find the right hyperparameters

While most hyperparameters may stay unchanged across different compute budgets, learning rate and batch size may need to be adjusted



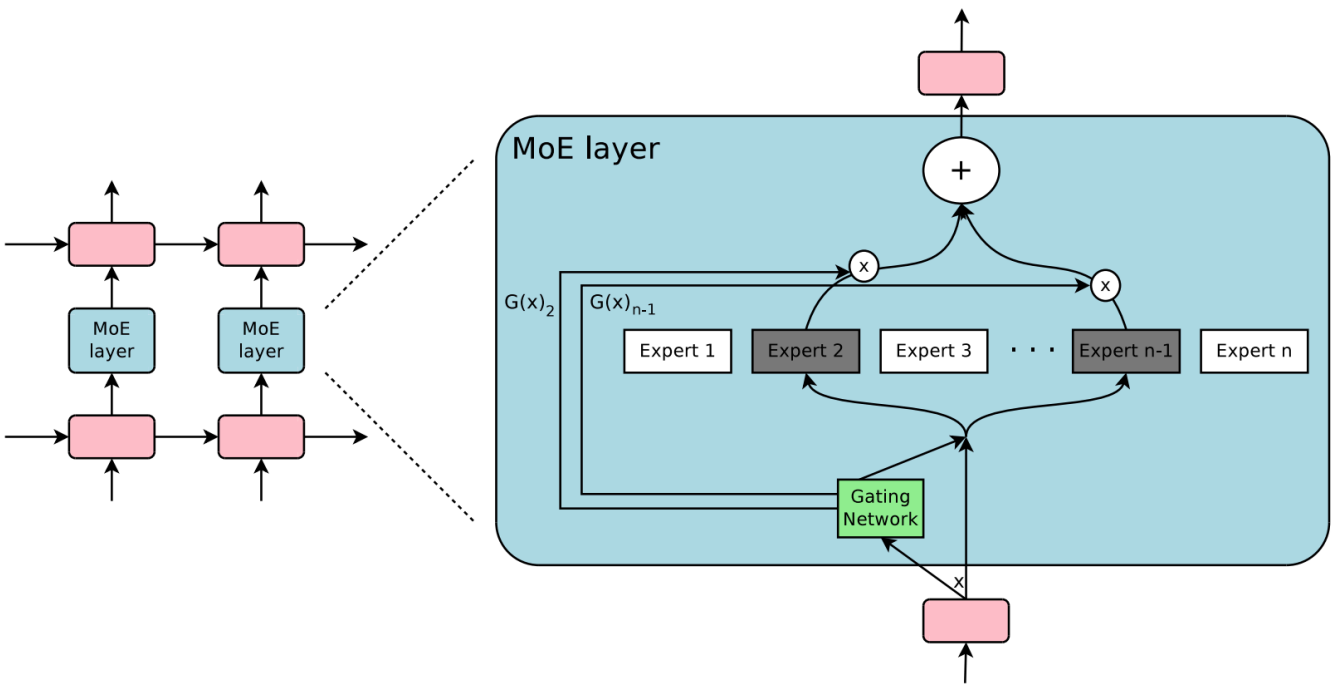
(a) Batch size scaling curve



(b) Learning rate scaling curve

Practical pathways to scaling: N

- More layers, wider layers (larger hidden size), more attention heads, larger MLPs etc
- More scalable model architecture (e.g., Sparsely-Gated Mixture-of-Experts, (Shazeer et al 2017))



- Train-time compute scaling: e.g., larger global batch, grad accumulation

Practical pathways to scaling: D

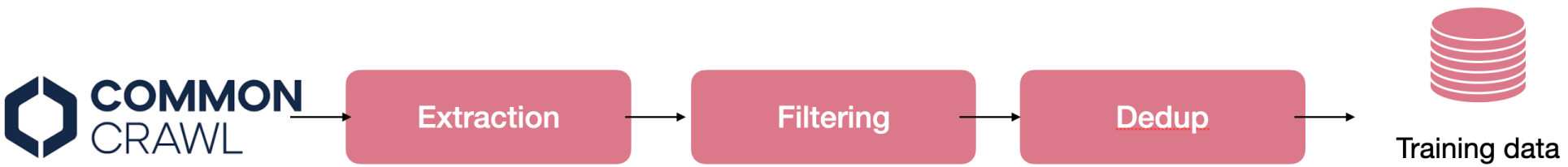
- Curate a diverse mixture: web pages, books, code, math, scientific papers, multilingual text, forums
- Common practice for more/better data:
  - Massive crawl + refresh: expand seeds, broaden domains, update recency, include non-English + code
  - Synthetic data: self-instruct/distillation for scarce skills; verify & de-duplicate
  - Metadata & provenance: track source, license, language, domain for audits and rebalancing
  - Private datasets: proprietary or sensitive data sources not available in the public domain

Practical pathways to scaling: D

Model	Training Tokens
Llama 1	1.4 trillion
Llama 2	1.8 trillion
Llama 3	15 trillion
Deepseek 3	15 trillion

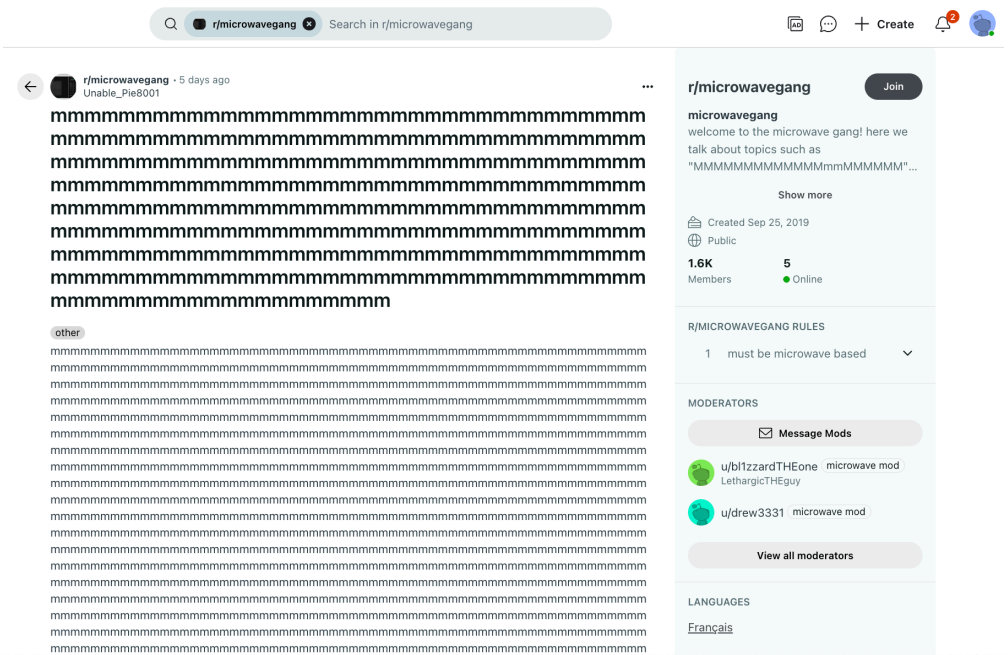
- New models push for more training tokens

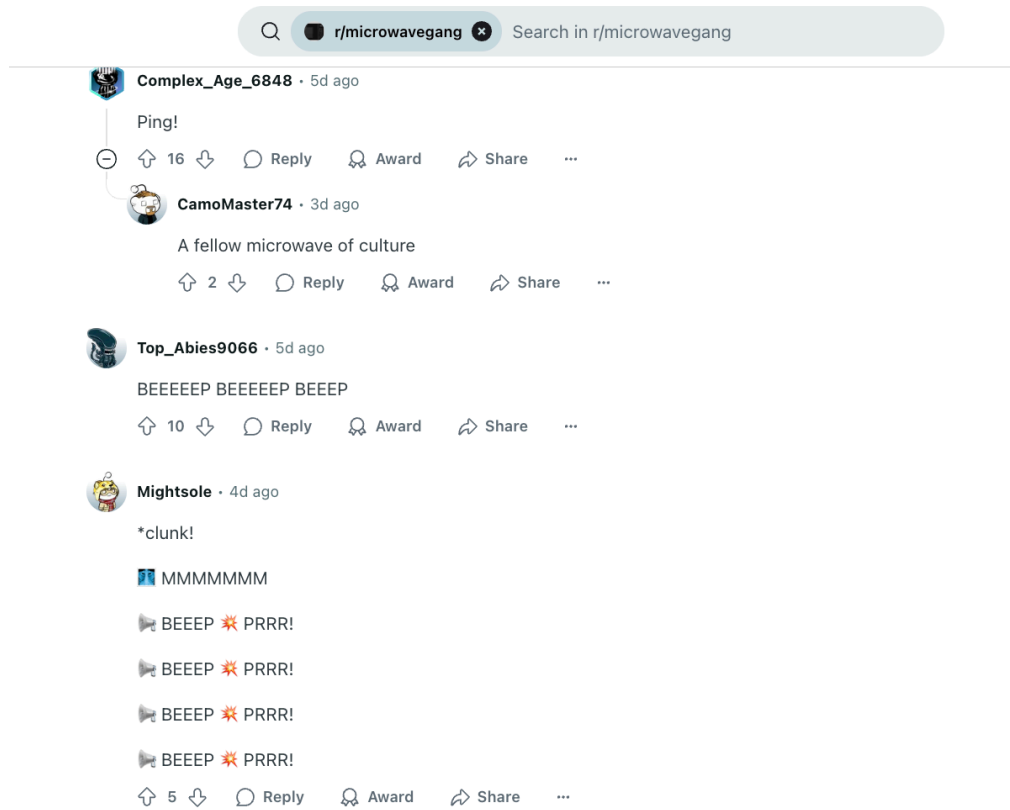
Getting high-quality data from Internet is non-trivial



- Extraction: from HTML to text
  - remove boilerplate etc, require a lot of heuristics
- Filtering: remove low-quality unwanted content
- De-duplication: remove near-duplicate content
  - e.g., MinHash, SimHash, embedding-based

Internet is highly diverse





Practical pathways to scaling: Infrastructure

The unsung hero of scaling

- Distributed training on GPUs/TPUs, mixed precision, optimized libraries (e.g, Megatron-LM, DeepSpeed)
- Efficient data pipelines: processing, streaming, sharding, caching (e.g., Apache Hadoop, WebDataset)
- Experiment tracking and hyperparameter tuning (e.g., Weights & Biases)

An intuitive explanation: pretraining as massive multi-task learning

Example tasks from next-word prediction ## Example tasks from next-word prediction

Task	Example sentence in pre-training that would teach that task
Grammar	She wants to {play, playing} the piano every evening.
Lexical semantics	He put the groceries in the {fridge, guitar}.
World knowledge	The Eiffel Tower is located in {Paris, Rome}.
Sentiment	He said the food in that restaurant is terrible, he {likes, hates} it.
Translation	The word for “cat” in French is {chat, perro}.
Coding	To plot the results in Python, use {`matplotlib.pyplot.plot()`, `numpy.random`}.
How-to	To submit the filled form, click the button {`submit`, `cancel`}

Recap: Scaling law

- Scaling law defines the relationship between training compute and model performance
  - Help decide the compute allocation between model size and data size (compute optimal)
  - Help decide training hyperparameters
- Practical pathways to scaling: model parameters, data, infrastructure
- Intuition: pretraining as massive multi-task learning

In-context learning

- A typical learning process via **gradient descent (GD):**
  - Train on dataset → update weights via **backprop** → new model
- In-context learning: learning from examples provided in the input context without explicit gradient updates.
  - Prompt (with examples) → [Frozen model] → output

I love this movie Positive
The story is terrible Negative
It is such a awful experience on participating this event Negative
The company anticipated its operating profit to improve. ????

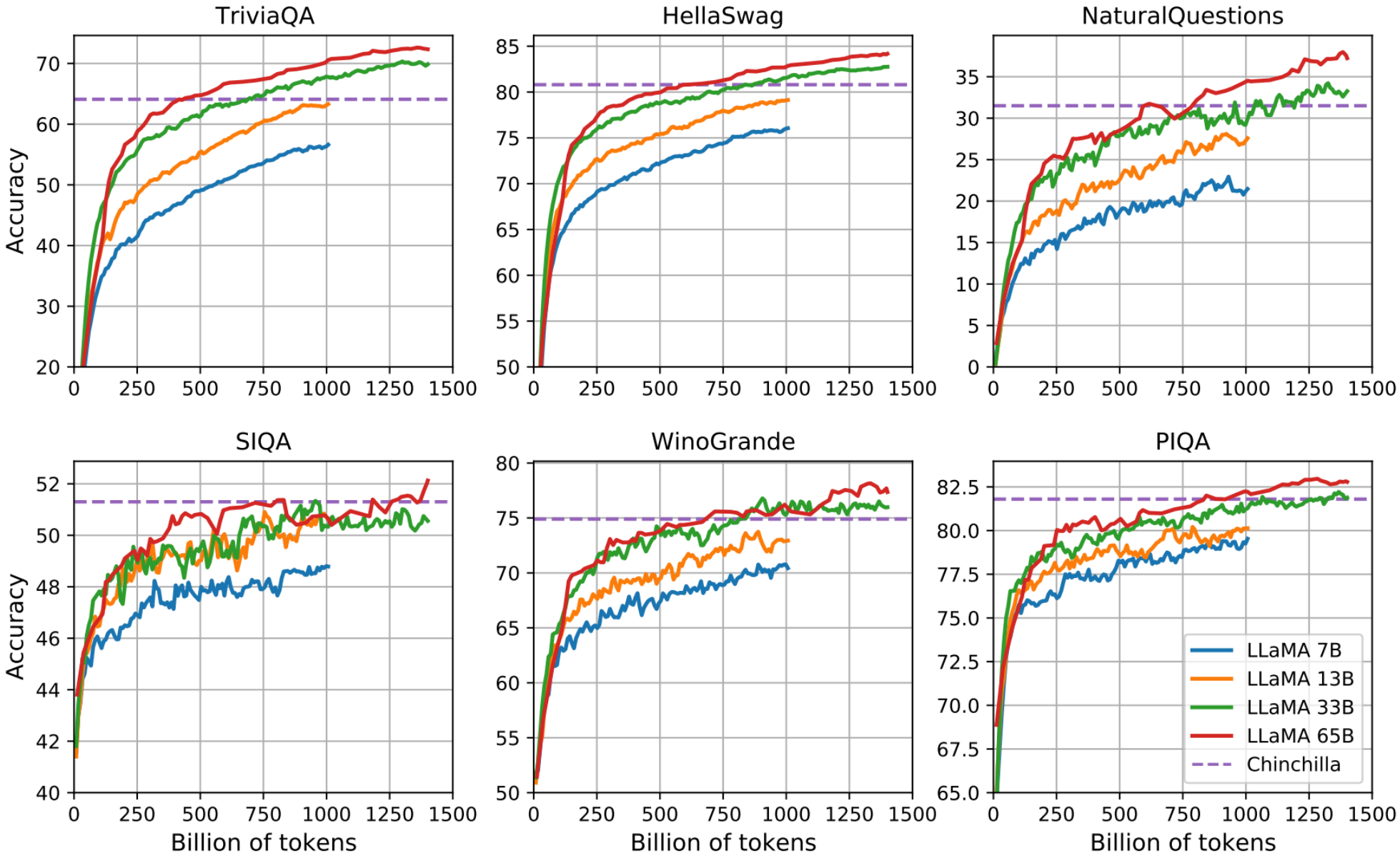
↓

LLM

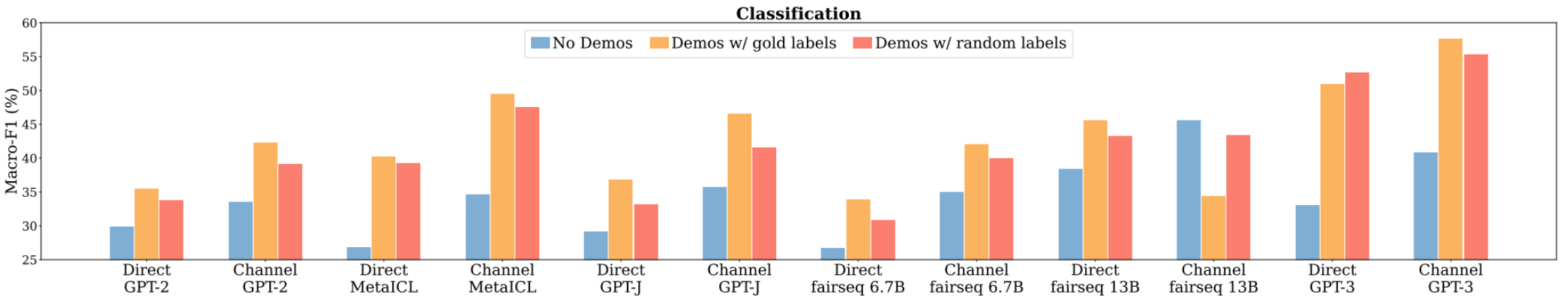
↓

Positive

Improved ICL with scaling


TriviaQA
HellaSwag
NaturalQuestions
SIQA
WinoGrande
PIQA
Accuracy
Billion of tokens
LLaMA 7B
LLaMA 13B
LLaMA 33B
LLaMA 65B
Chinchilla

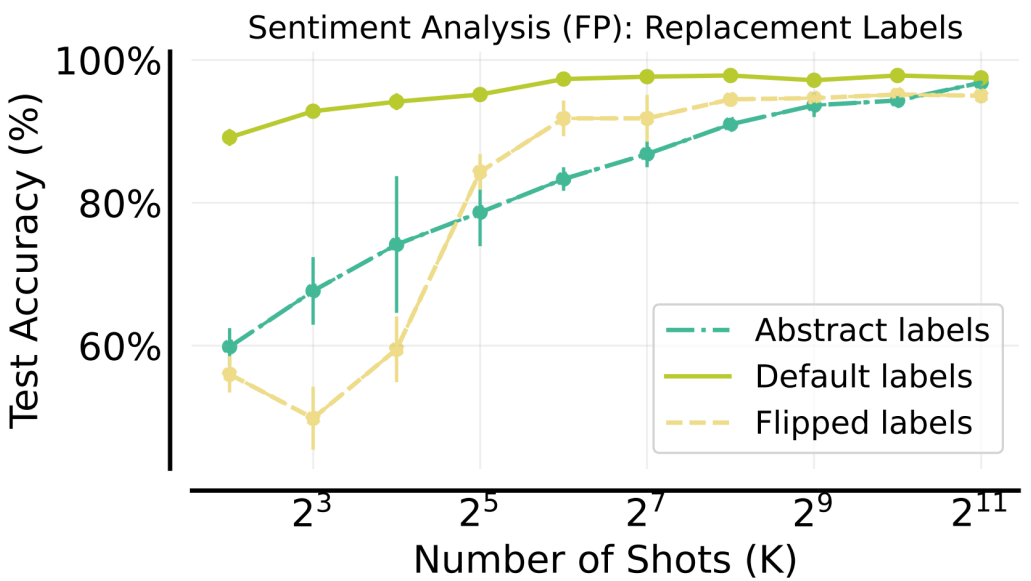
Does the model learn from the context?



(Min et al 2022)

- Generate correct labels even with randomized labels in the context
- Model learns the label space, input distribution and format

Does the model learn from the context?



(Argwarl et al 2024)

- The model overcomes the priori with more examples
- Infer what the task is ⟶ perform the task

Understanding model dynmamic is an open research question

Why openness matters for understanding

- The science behind LLMs remain poorly understood
- Without transparency, it's harder to study *how* and *why* models behave
- Openness (data, code, weights) affects both **science** and **responsible use**

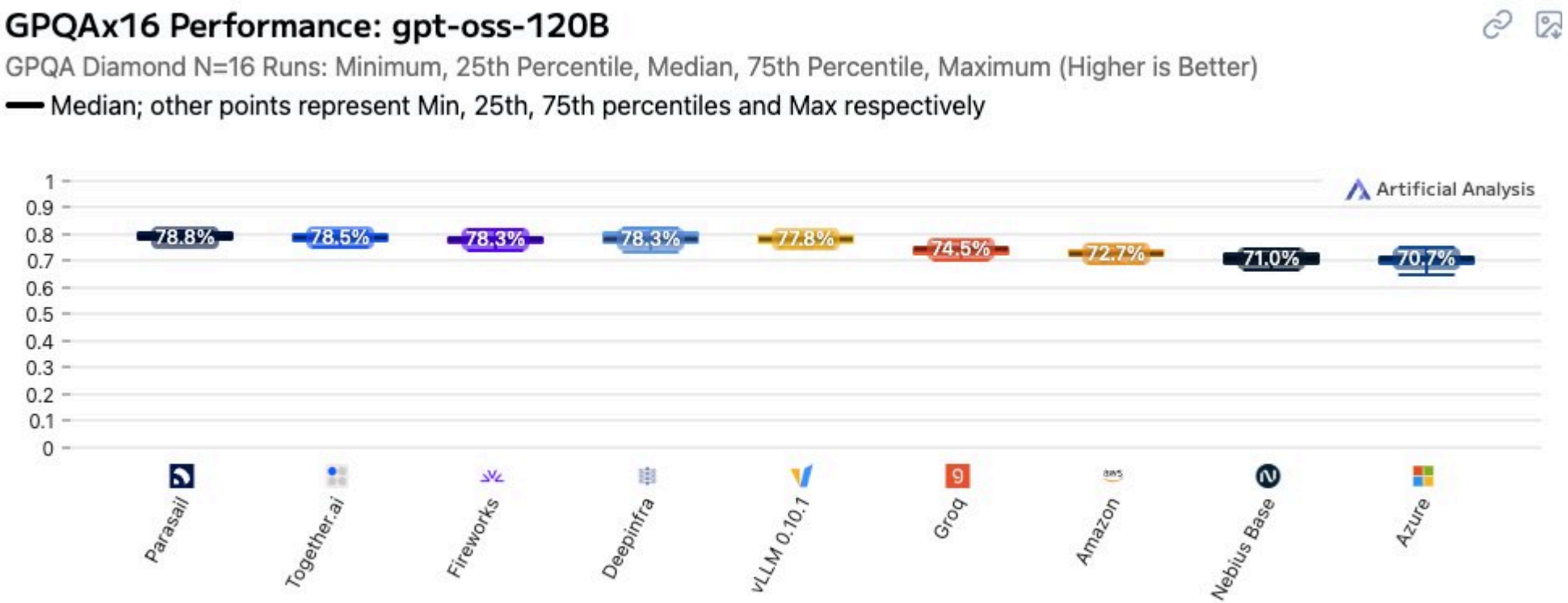
There are different levels of openess

- **Training corpus**
- **Training code**
- **Weights**
- **Inference code**

Is the information open? described in text? not described at all?

Many so call “open source” models are simply open **weights** models, with inference model contributed by the community (e.g, vllm)

Same model, different performance



API Provider Benchmarking & Analysis

Same model, different performance

- Quantization & Precision Choices: Different float formats can affect outputs
- Prompt Formatting: Models expect strict conversation formats; mis-mapping roles/messages hurts quality.
- Inference Harness: Serving stack must correctly parse, emit, and round-trip tool calls.

Pretraining on internet-scale corpora

- LLMs are trained on trillions of tokens from the web
- Most of this data is **not curated** — comes from books, code, forums, media
- Raises **legal, ethical, and quality issues** that researchers must understand

Licenses and permissiveness

- **Public domain, CC-0**: freely usable (e.g., US gov works, expired copyright)
- **MIT, BSD**: very few restrictions
- **Apache, CC-BY**: must acknowledge owner
- **GPL, CC-BY-SA**: must acknowledge and use same license for derivative works
- **CC-NC**: cannot use for commercial purposes
- **LLaMa, OPEN-RAIL**: custom restrictions
- **No License**: all rights reserved (default copyright)

What's fair use?

- US **fair use doctrine** allows some copyrighted use without permission
- Common (but simplified) considerations:
  - **Small excerpts** → more likely fair use
  - **Doesn't harm market value** → possibly OK
  - **Non-commercial/educational use** → possibly OK
- Problem: training copies **entire works**, not snippets
- Courts are still deciding if large-scale model training is fair use

Copyright complications in pretraining

- **Most online works are copyrighted by default**
- **No license ≠ public domain** → it means *all rights reserved*
- **Derivative work risks**: outputs may reproduce training data
- **License conflicts**: mixing permissive and restrictive sources
- **Attribution gaps**: models don't credit authors, violating many licenses

There are ongoing lawsuits!

This article is more than **4 months old**

US authors' copyright lawsuits against OpenAI and Microsoft combined in New York with newspaper actions

California cases over AI trainers' use of work by writers including Ta-Nehisi Coates and Michael Chabon transferred to consolidate with New York suits from John Grisham and Jonathan Franzen and more