Natural Language Processing

Lecture : Large Language Models & Pretraining

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Large language models

©Stanford HAI

In deep learning **pretraining** refers to the process of optimizing a neural network before it is further trained/tuned and applied to the tasks of interest.

This approach is based on the general assumption that a model pre-trained on one task can be **adapted** to perform another task, called **transfer learning**.

As a result, we do not need to train a deep, complex neural network from scratch on tasks with limited labeled data. Instead, we can make use of tasks where supervision signals are easier to obtain.

Large language models

In this lecture we introduce the idea of **pretraining** in the context of LM: learning knowledge about language and the world from vast amounts of text, using a LM objective function.

We call the result of this process **large language models**, or LLMs for short.

At time of writing, all LLMs are based on the transformer.

We will see that almost any NLP task can be modeled as next word prediction in a LLM.

Today LLMs exhibit remarkable performance on all sorts of NLP tasks, and are at the basis of most applications in generative AI.

Large language models

The decoder component of a transformer can be turned into a language model, using an additional learnable linear layer called language modeling head.

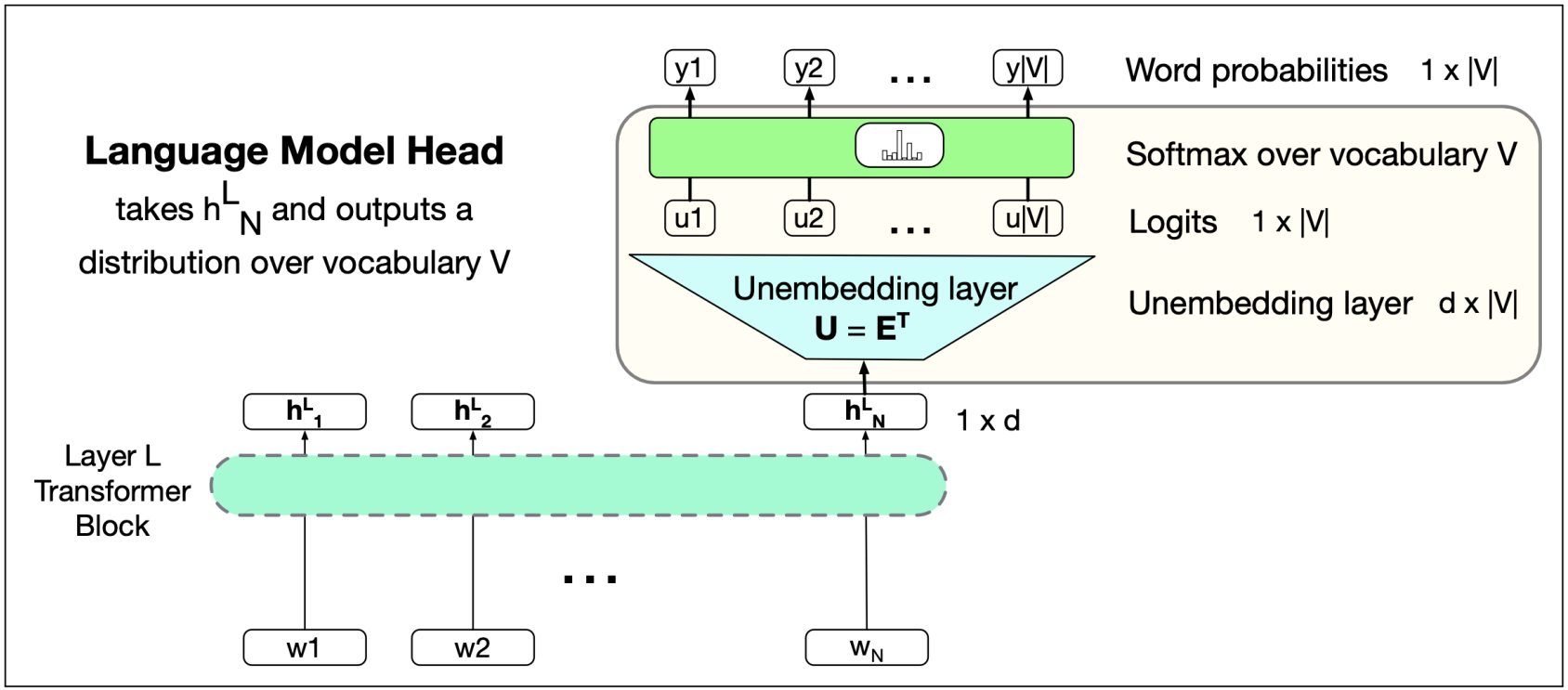
The **language modeling head**

- takes the output of the topmost transformer layer for token N
- uses it to predict the upcoming word at position $N + 1$

Compare to MLM in BERT, where the language modeling head predicts the masked word.

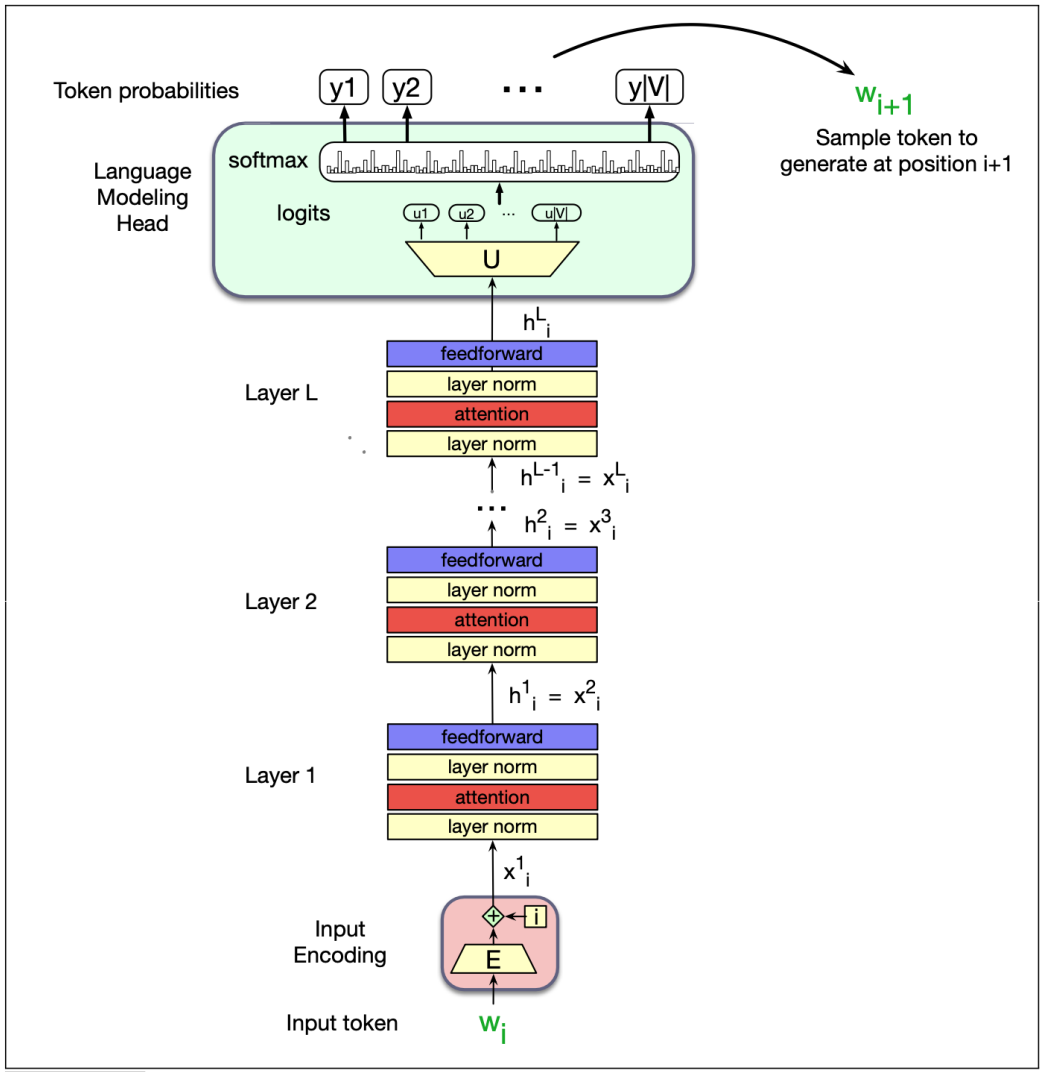
Large language models

Decoder component of the transformer combined with language modeling head.



Large language models

Full stack view of the language modeling head.



Large language models

Most commonly, the language modeling head is **tied** to the embedding matrix, that is, $\mathbf{U} = \mathbf{E}^\top$.

Recall that in weight tying, we use the same weights for two different matrices in the model as a form of regularization.

A softmax layer turns the logits $\mathbf{u}$ into the probabilities $\mathbf{y}$ over the vocabulary

$$\begin{aligned} \mathbf{u} &= \mathbf{E}^\top \mathbf{h}_N^L \\ \mathbf{y} &= \text{softmax}(\mathbf{u}) \end{aligned}$$

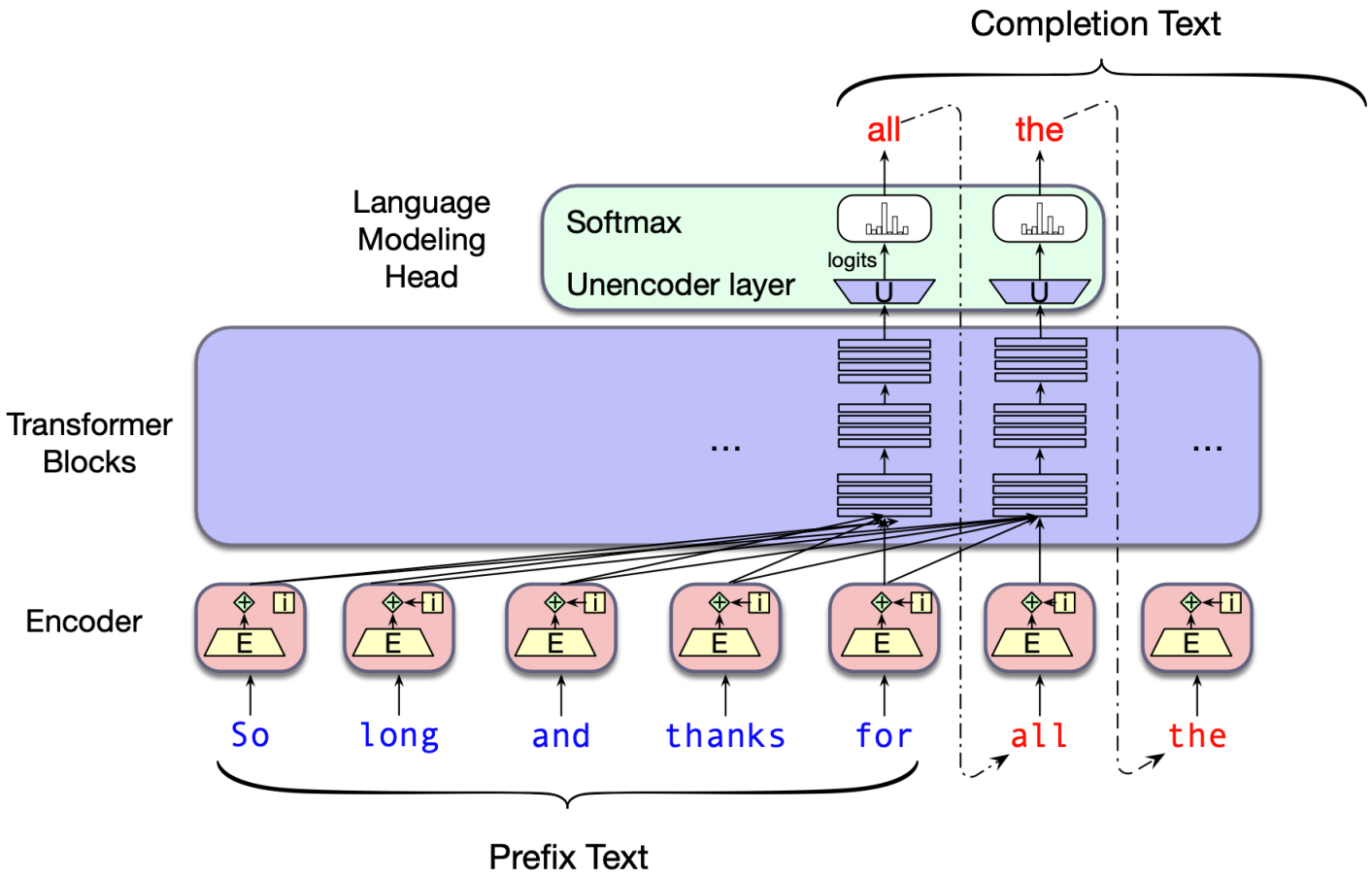
Instead of computing the probability of a sentence, we will **sample** words with the resulting distribution $\mathbf{y}$ to generate text.

Text completion

Jerry Zhang on Unsplash

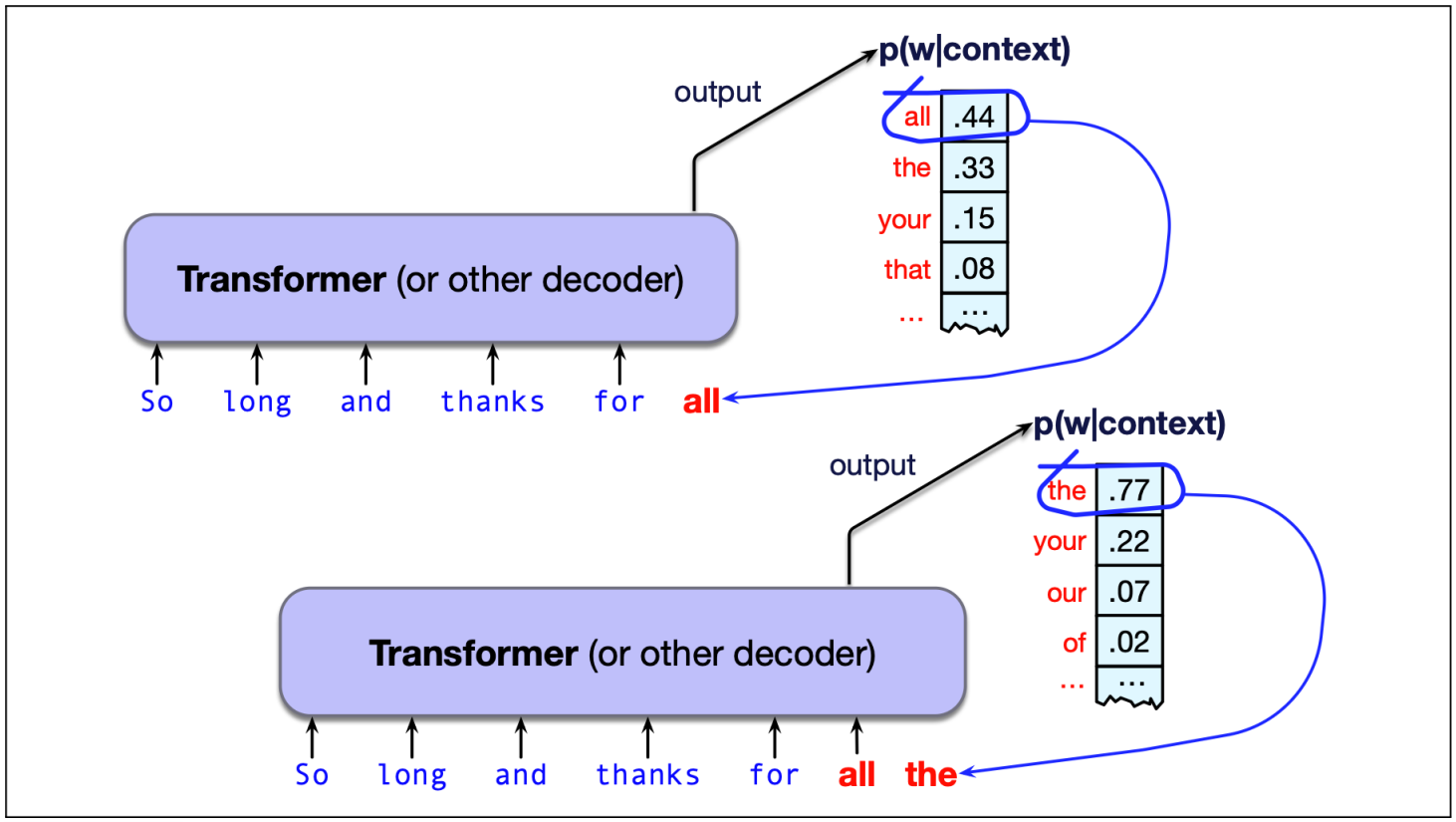
Text completion

Text completion (also called conditional generation) is the task of generating text conditioned on an input piece of text.



Text completion

Two-steps view of text completion. The result is a left-to-right (also called autoregressive) language model.



Text completion

A transformer used for this kind of unidirectional causal language model is often called a **decoder-only** model.

This is because the model constitutes roughly half of the encoder-decoder architecture of the transformer.

Text completion

We can cast several complex NLP tasks as text completion.

Example : We can cast **sentiment analysis** as word prediction by giving a language model a context like

The sentiment of the sentence 'I like Jackie Chan' is:

and by comparing the following conditional probability of the word **positive** and the word **negative**

$P(\text{positive} \mid \text{The sentiment of the sentence 'I like Jackie Chan' is:})$

$P(\text{negative} \mid \text{The sentiment of the sentence 'I like Jackie Chan' is:})$

Text completion

Example : We can cast the task of **question answering** as text completion by giving a language model a context like

> Q: Who wrote the book 'The Origin of Species'?
> A:

and expect that the most likely completion is

> Q: Who wrote the book 'The Origin of Species'?
> A: Charles Darwin

Text completion

Example : We can cast **text summarization** as text completion by giving a large language model a text, followed by the token **tl;dr;** which is short for 'too long; didn't read'.

This token is sufficiently frequent in training data before a summary, and hence the LLM will interpret the token as instructions to generate a summary.

Recall that the context is the size of the transformer's context window, which can be quite large, like 32K tokens (and much larger with special long-context architectures).

Text completion

Example : Similarly, for **machine translation** we can use the context

English: I missed the bus today. French:'

and the LLM may be able to fill in the blank with a French translation.

Text completion

Few-shot learning is the ability of an LLM to solve a problem on the basis of a few examples.

Example :
sea otter ⇒ loutre de mer
peppermint ⇒ menthe poivrée
cheese ⇒ ??

Zero-shot learning is the ability of an LLM to solve a new problem solely on the basis of natural language instructions

Example :
translate English to French
cheese ⇒ ??

Sampling

Sampling

Decoding problem for LLM: the task of choosing the single word to generate next, based on the context.

For decored-only LLM, this happens left-to-right and is called in several ways

- autoregressive generation
- causal generation
- masked attention generation

For languages like Arabic in which we read from right to left, decoding is instead right-to-left.

Sampling

One simple strategy is **greedy decoding**: we always generate the most likely word given the context

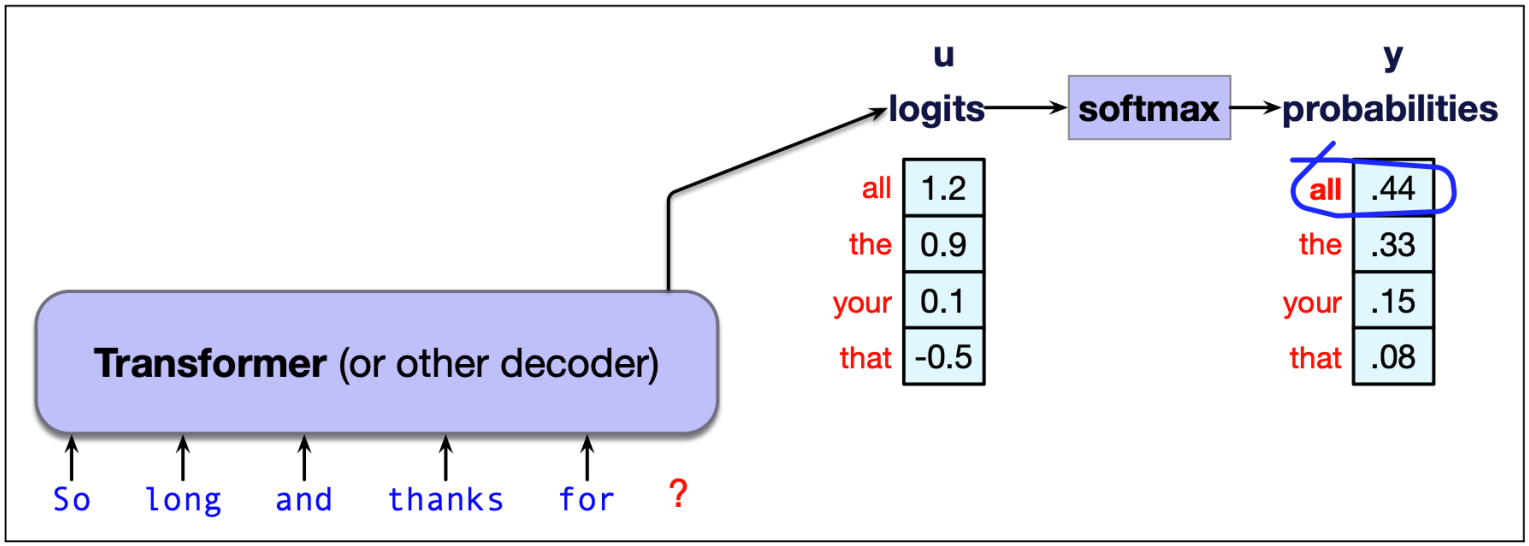
$$\hat{w}_t = \underset{w \in V}{\text{argmax}} \; P(w \mid \mathbf{w}_{<t})$$

Greedy decoding has **two problems**

- it makes a choice that is locally optimal; however, this may not result in the best solution overal
- the system is deterministic: if the context is identical, we always generate exactly the same text

Sampling

Greedy decoding: choose the highest probability word.



Sampling

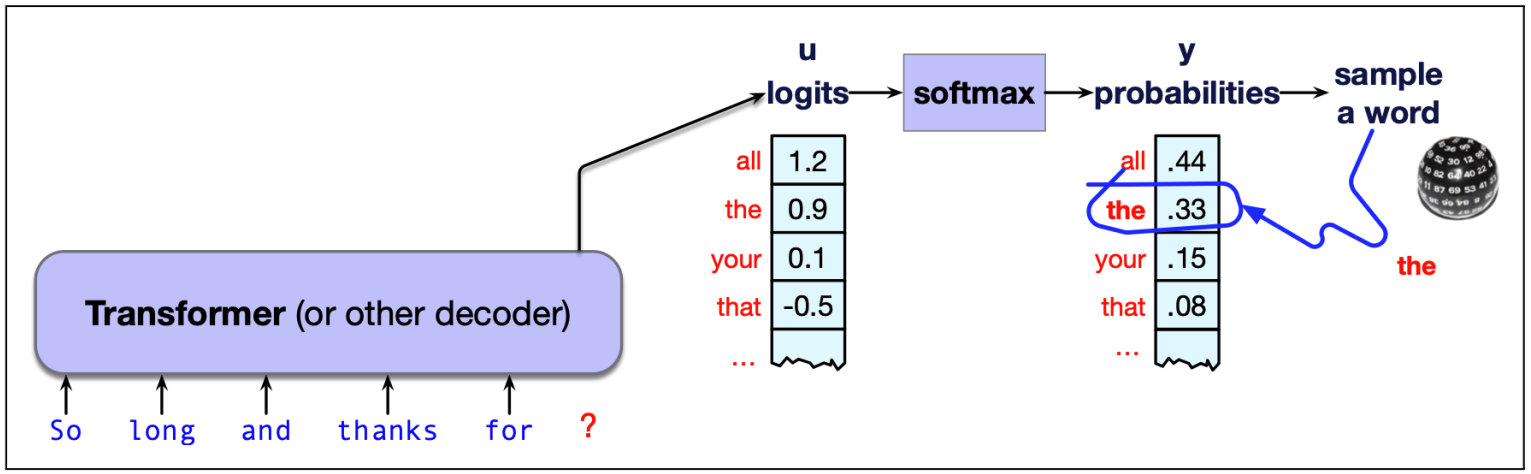
The most common decoding method for LLM is **random sampling**. This means that we iteratively choose the next word according to its probability given the context.

Let $w_1, w_2, \ldots, w_N$ be a sequence of words. We write $x \sim p(x)$ to mean 'choose x by sampling from the distribution $p(x)$.

$$
\begin{aligned}
&\mathrm{i} \leftarrow 1 \\
&w_i \sim \mathrm{p}(w) \\
&\textbf{while } w_i \text{ != EOS} \\
&\quad \mathrm{i} \leftarrow \mathrm{i} + 1 \\
&\quad w_i \sim \mathrm{p}(w_i \mid w_{<i})
\end{aligned}
$$

Sampling

Random sampling: we randomly chose a word according to its probability.



Sampling

Even though random sampling generates sensible, high-probable words, it sometimes generates weird sentences.

The **problem** with random sampling is that

- there are many odd, low-probability words in the tail of the distribution
- if you add up all the rare words, they constitute a large enough portion of the distribution that they get chosen often enough

Sampling

We need sampling methods that enable trading off two important factors in generation.

See also discussion in the LM lecture.

Quality: methods that emphasize the most probable words produce accurate and coherent text, but also more repetitive.

Diversity: methods that slightly increase the weight of middle-probability words produce more creative and diverse text, but less factual or incoherent.

Sampling

Top-k sampling is a simple generalization of greedy decoding

- truncate the word distribution to the top k most likely words
- renormalize to produce a proper probability distribution p_k
- randomly sample from within these k words according to p_k

When $k = 1$, top-k sampling is identical to greedy decoding.

When $k > 1$, top-k generates more diverse but still high quality text.

Sampling

Sometimes the top k words represent only a small part of the probability mass (the probability distribution is rather flat).

An alternative technique called **top-p sampling** or **nucleus sampling** keeps the top p percent of the probability mass.

We compute the top-p vocabulary $V^{(p)}$, defined as the **smallest** set of words such that

$$\sum_{w \in V^{(p)}} P(w \mid \mathbf{w}_{<t}) \geqslant p.$$

We then truncate P to $V^{(p)}$, renormalize, and sample words accordigly.

Sampling

In **temperature sampling** we don't truncate the distribution, but instead reshape it by means of a temperature parameter $\tau > 0$

$$\frac{\exp(\frac{\mathbf{y}_i}{\tau})}{\sum_j \exp(\frac{\mathbf{y}_j}{\tau})}$$

where $\mathbf{V}_i$ denotes the i-th row of $\mathbf{V}$.

This allows balancing the coherence / diversity trade-off:

- low τ produces peaky distribution (high coherence)
- large τ produces flat distribution (high diversity)

Sampling

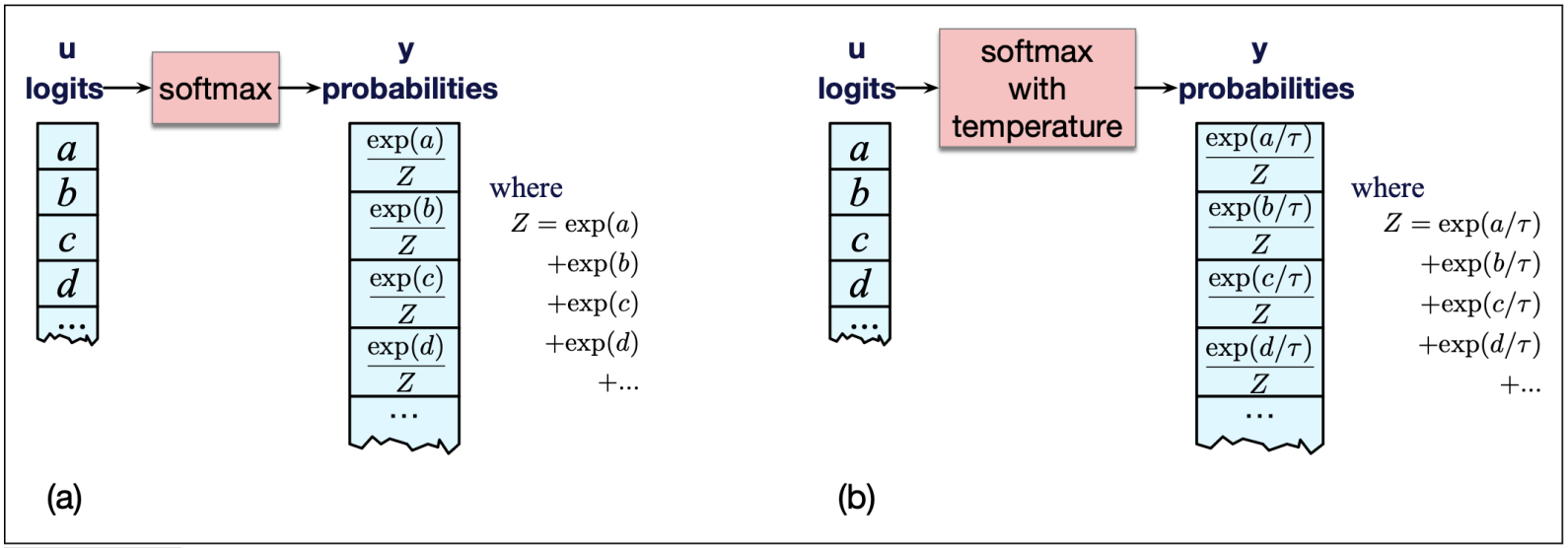
For **low-temperature sampling** we divide the logits by a temperature parameter $\tau \in (0, 1]$ before we normalize through the softmax. This smoothly increases the probability of the most probable words and decreases the probability of the rare words.

As τ approaches 0, the probability of the most likely word approaches 1.

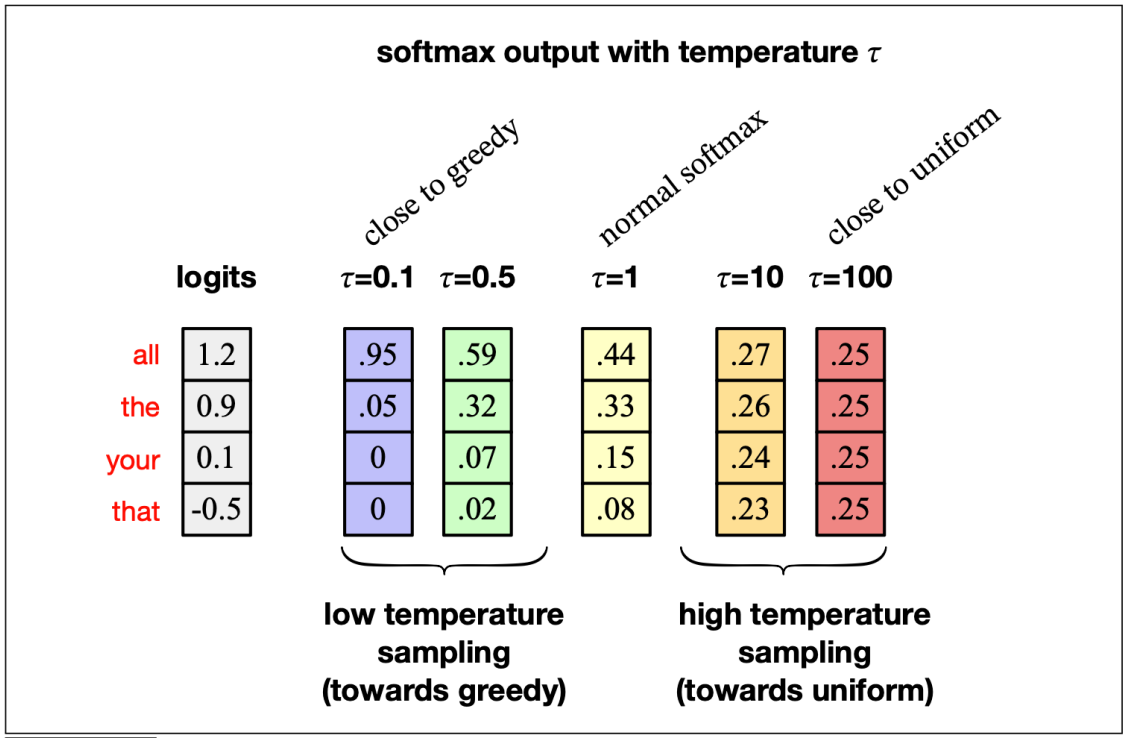
For **high-temperature sampling**, we use $\tau > 1$. This flattens the word probability distribution, rather than making it more spiky.

Sampling

Adding temperature scaling to the softmax by first dividing by the temperature parameter τ.



Sampling


softmax output with temperature τ
close to greedy
normal softmax
close to uniform
logits
τ=0.1
τ=0.5
τ=1
τ=10
τ=100
all
the
your
that
1.2
0.9
0.1
-0.5
.95
.05
0
0
.59
.32
.07
.02
.44
.33
.15
.08
.27
.26
.24
.23
.25
.25
.25
.25
low temperature sampling (towards greedy)
high temperature sampling (towards uniform)

Key-Value cache

Key-Value cache

Transformer's attention vectors can be efficiently computed in parallel at **training time**, via two matrix multiplications

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}$$

We can't do the same at **inference time**, because we need to generate the next tokens one at a time.

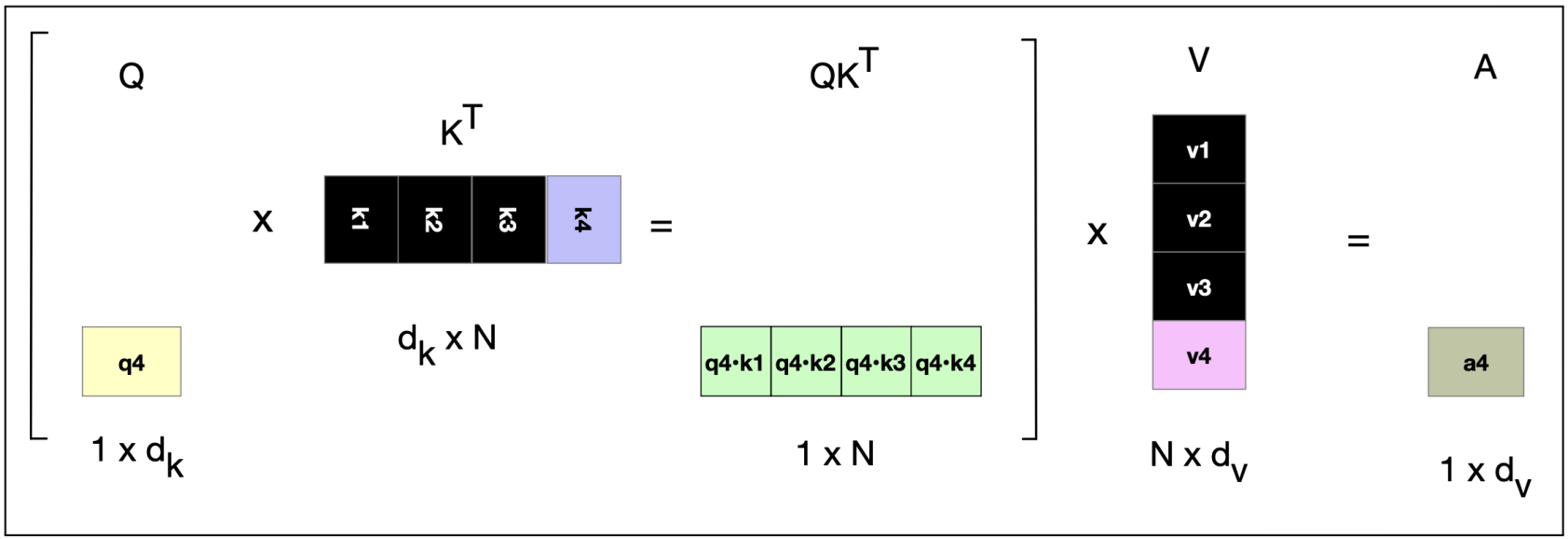
Key-Value cache

For a new token $\mathbf{x}_i$, we need to compute its **query**, **key**, and **values** by multiplying by $\mathbf{W}^{\mathbf{Q}}$, $\mathbf{W}^{\mathbf{K}}$, and $\mathbf{W}^{\mathbf{V}}$, respectively.

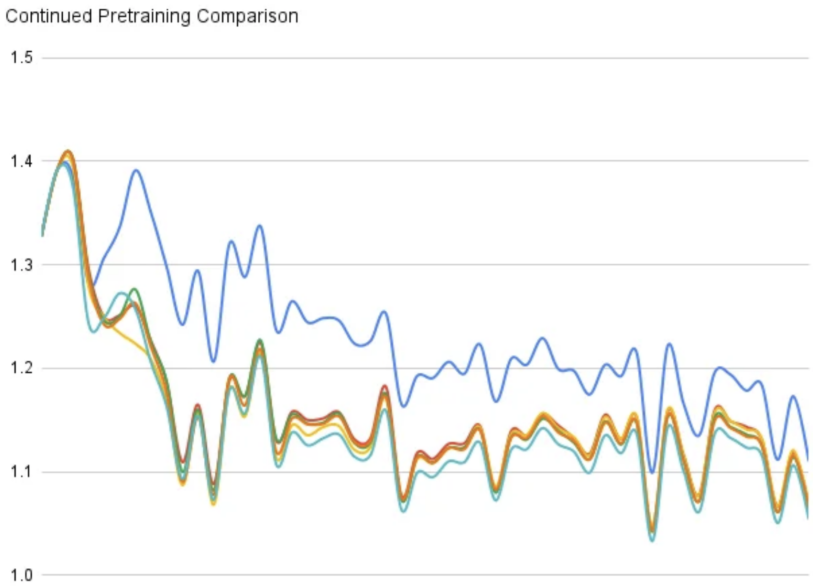
It is a waste of time to recompute key and value vectors for all the prior tokens $\mathbf{x}_{<i}$, since we already computed these vectors at prior steps!

Key-Value cache

Example : Attention computation at the 4-th token. Black vectors need not be recomputed



Pretraining



Pretraining

The process of training a transformer as a language model involves vast amounts of text and compute, and only few large corporations have the necessary resources for this.

For these reasons, the process is called **pretraining**.

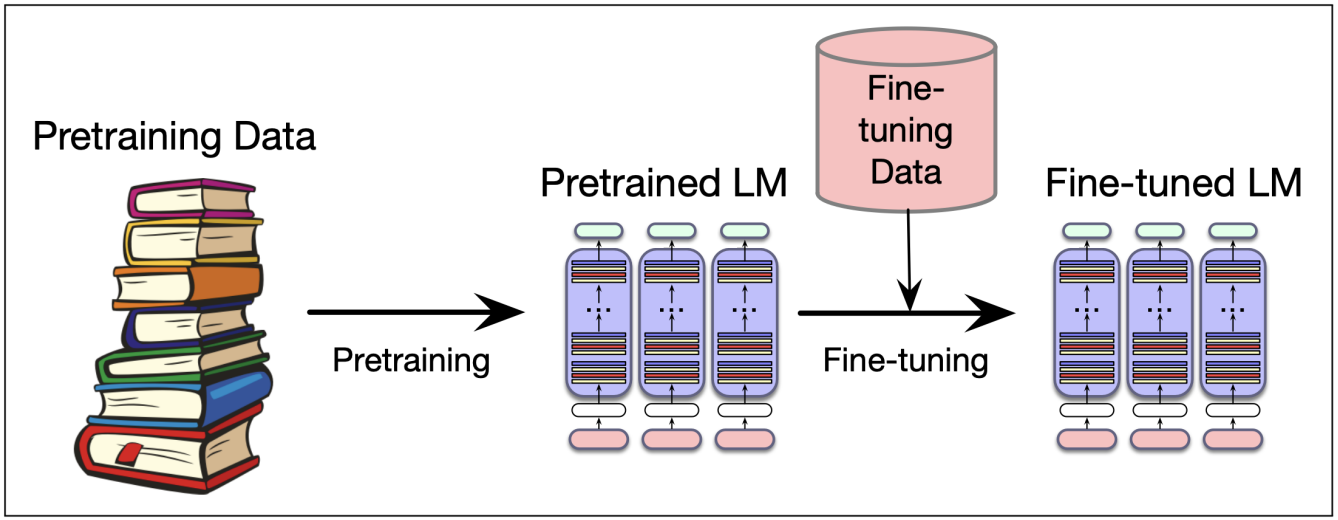
As the result of pretraining, the language model acquires

- language knowledge
- general knowledge about real world

Pretraining

To apply a pre-trained LLM to a new domain or task that might not have appeared sufficiently in the pretraining data, we need to run additional training passes.

In contrast to pretraining, this process is called **fine-tuning** and requires specialized data.



Fine-tuning will be introduced and discussed in the next lecture.

Pretraining

To train a transformer as a language model, we use the algorithm we discussed for LM, called **self-supervision** (or self-training).

We train on a corpus of text, and at each time step t we ask the model to predict the next word.

We call such a model **self-supervised** because we don't have to add any special gold labels to the data: the natural sequence of words is its own supervision!

Pretraining

Let $ind(w)$ be the index of word $w \in V$.

Recall that at time step t the **true distribution** $\mathbf{y}_t$ for the next word w_{t+1} is a 1-hot vector of size $|V|$ with

- $\mathbf{y}_t[ind(w_{t+1})] = 1$
- $\mathbf{y}_t[k] = 0$ everywhere else

At time step t, the **predicted distribution** (also called the empirical distribution) for the word at position $t+1$ is the distribution realized by the model, which we denote as $\hat{\mathbf{y}}_t$.

For a fixed t, the **cross-entropy** (CE) loss measures the difference between the predicted distribution and the true distribution

$$L_{\text{CE}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\sum_{w \in V} \mathbf{y}_t[ind(w)] \log \hat{\mathbf{y}}_t[ind(w)]$$

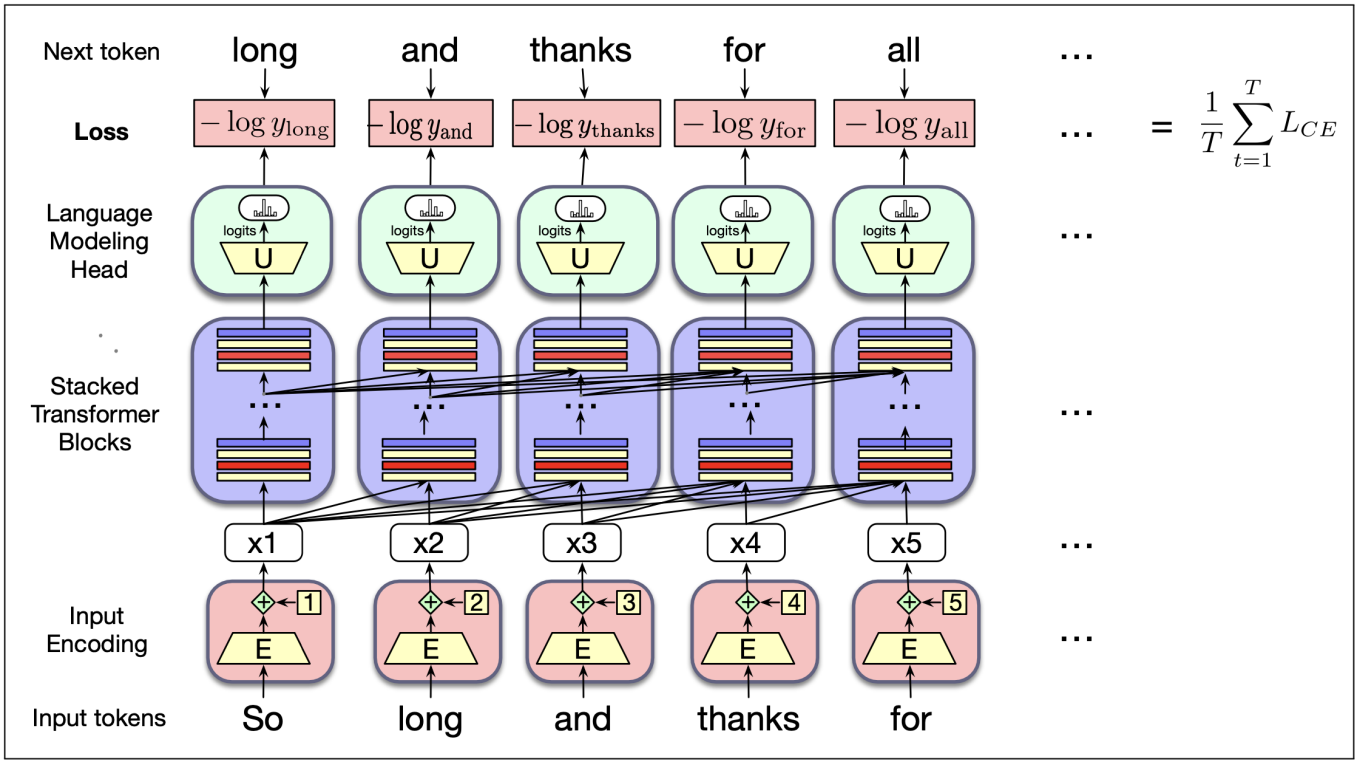
Replacing the definition of $\mathbf{y}_t$ we derive

$$L_{\text{CE}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[ind(w_{t+1})]$$

The loss for a training sequence is the average CE loss over the entire sequence.

Pretraining

At the next step $t+1$, we ignore what the model predicted at step t and instead use the correct sequence of tokens $w_{1:t+1}$: this is called **teacher forcing**.



Pretraining

With transformers, all training items can be processed in parallel.

Large models are trained by filling the full **context window** (4096 tokens for GPT4, 8192 for Llama 3) with text.

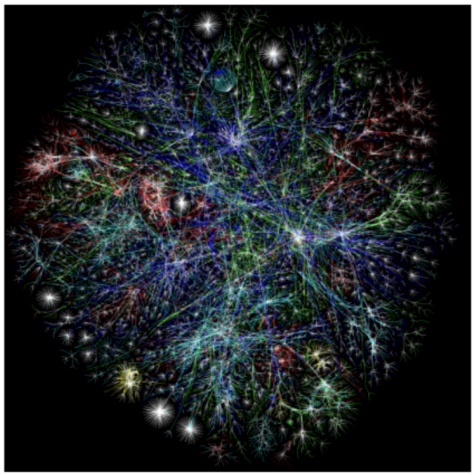
If documents are shorter, multiple documents are packed into the window with a special end-of-text token between them.

The **batch size** is usually quite large (the largest GPT-3 model uses 3.2 million tokens).

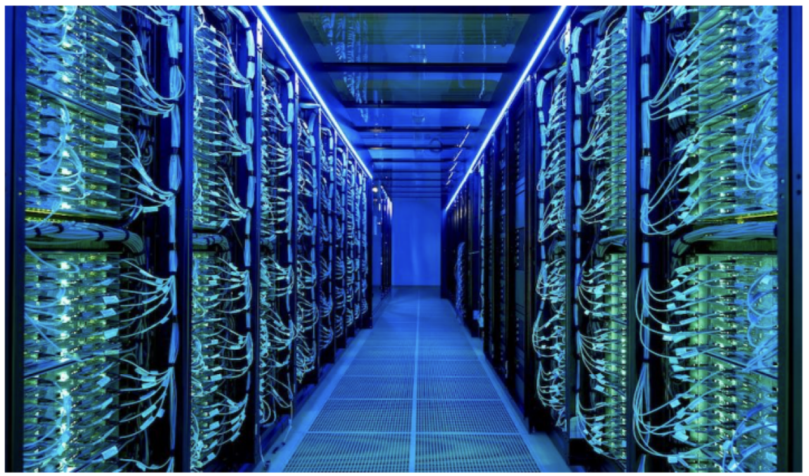
Pretraining

Pretraining of an LLM requires an extremely large amount of resources, that only very large corporations can afford.

Example : LLMs with 70 billion parameters



Chunk of the internet,
~10TB of text



6,000 GPUs for 12 days, ~$2M
~1e24 FLOPS

A. Karpathy, Intro to LLMs

Training corpora

Rohit Tandon on Unsplash

Training corpora

Large language models are mainly trained on text scraped from the web and filtered.

C4 (Colossal Clean Crawled Corpus): 156 billion tokens of English; provided by Common Crawl, a public organization that maintains a series of snapshots of the entire web.

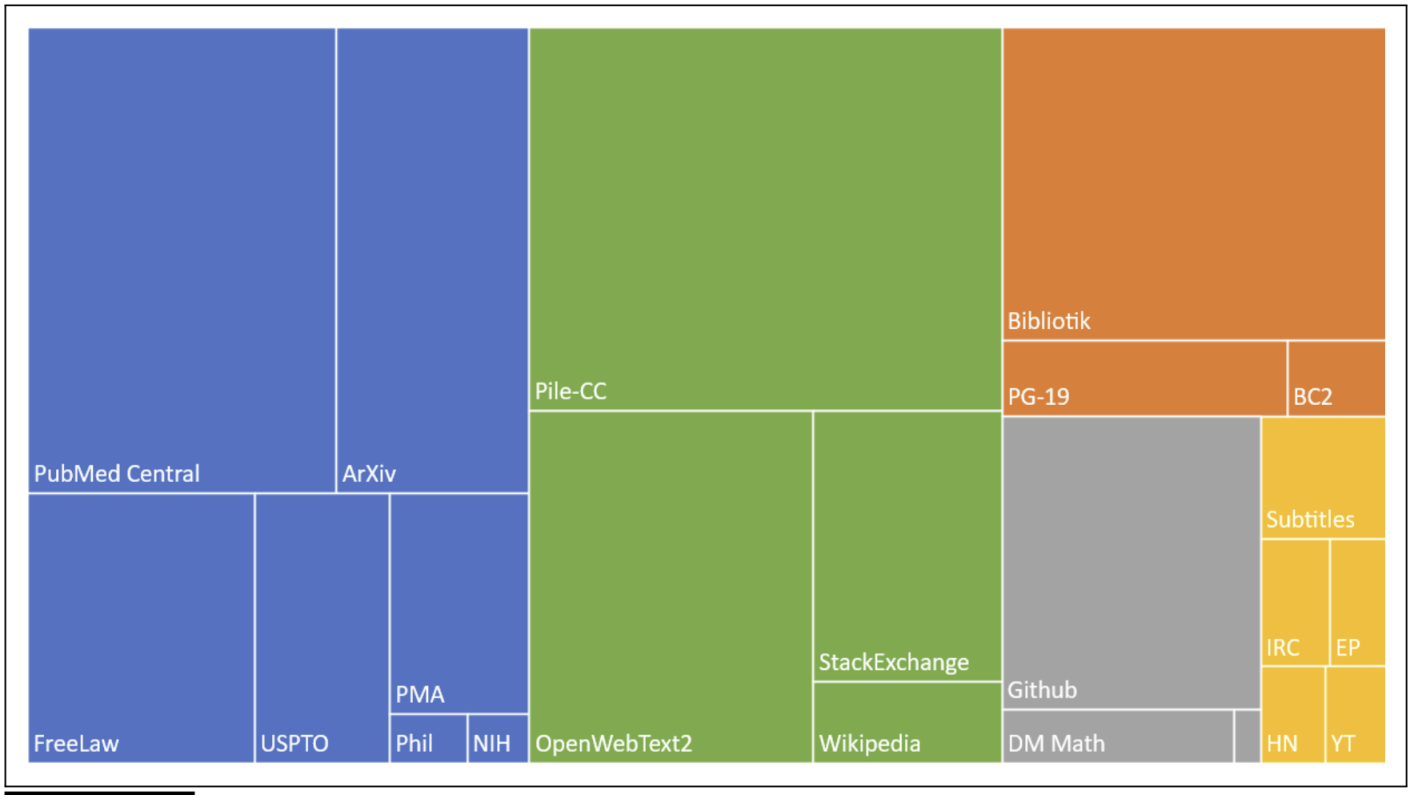
FineWeb: 15-trillion tokens, provided by HaggingFace; sourced from 96 Common Crawl snapshots, widely considered one of the most high-quality open-source pretraining corpora.

The Pile: 825 GB English text corpus, including books and Wikipedia.

Dolma: open corpus of English, 3 trillion tokens, consists of web text, academic papers, code, books, encyclopedic materials, and social media.

Training corpora

The Pile corpus: academic (blue), internet and Wiki (green), prose and books (orange), dialogue (yellow), and miscellanea (gray).



Training corpora

Pretraining data filtered for both quality and safety.

Quality filtering cleans text, removes duplicate documents (deduplication), personal identifiable information, boilerplate text, etc.

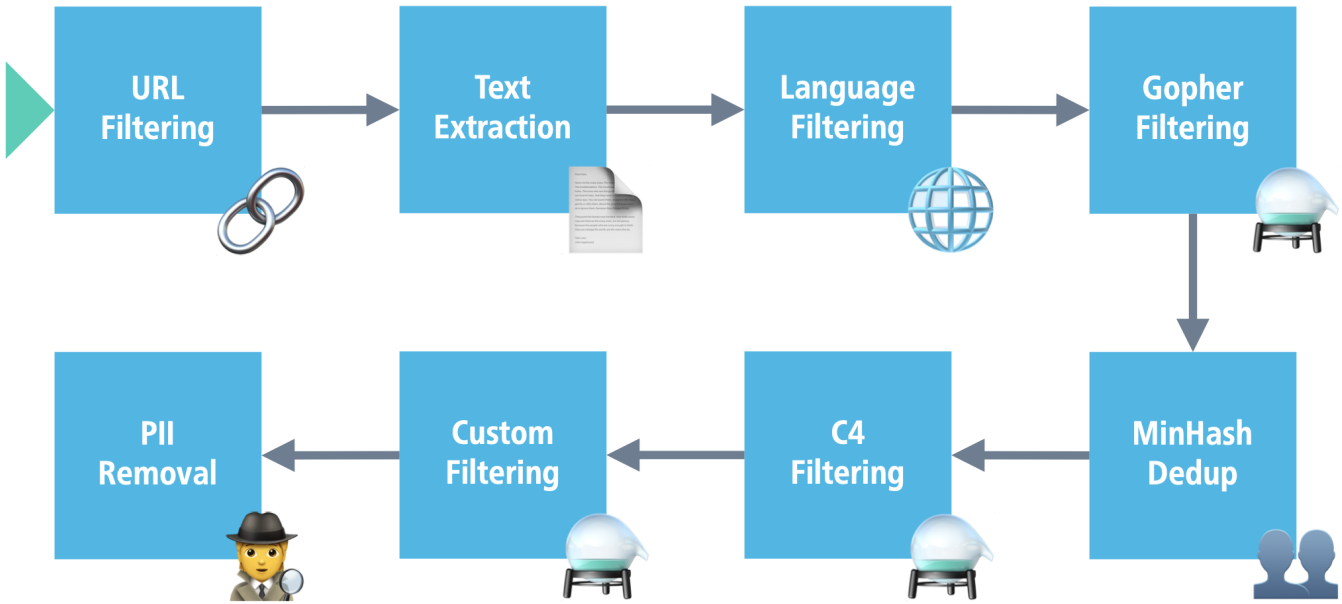
Generally, quality filtering improves LLM performance.

Safety filtering deals with toxicity detection, adult content, etc.

Issues with pretraining: copyright, data consent, privacy.

Training corpora

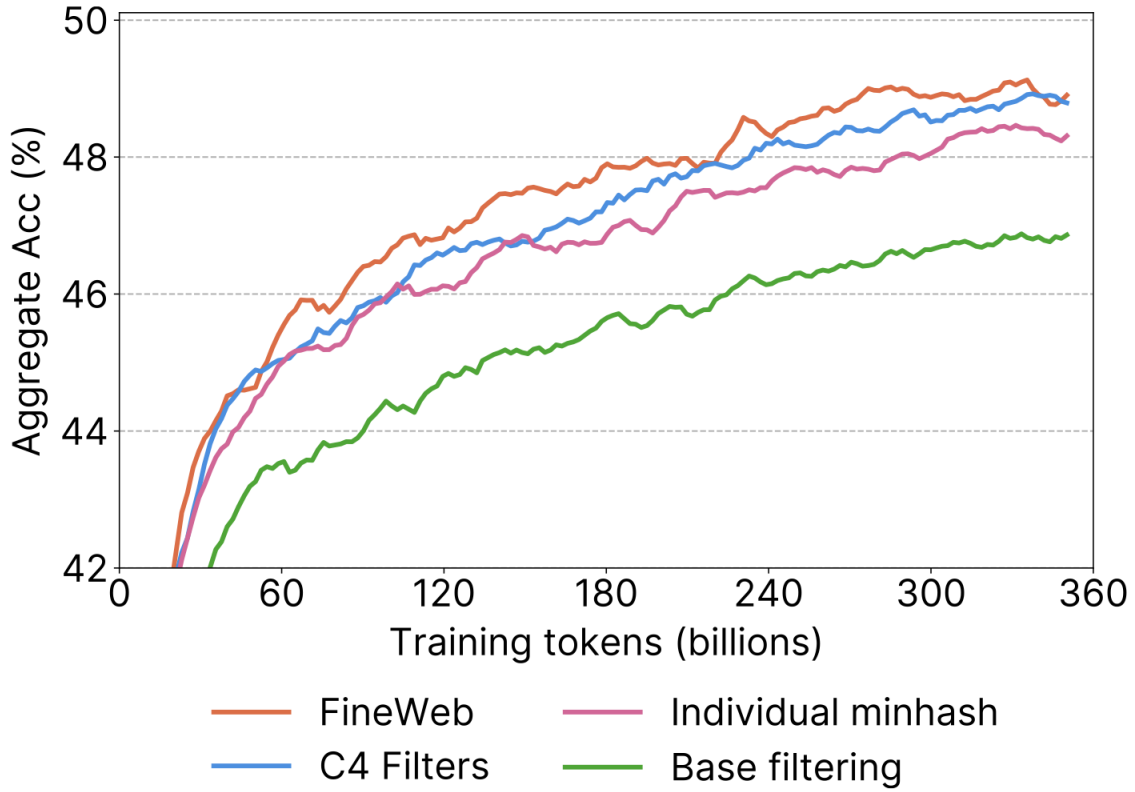
The FineWeb filtering pipeline



Penedo *et al.*, 2024

Training corpora

Impact of filtering on LLM accuracy



Penedo *et al.*, 2024

Scaling laws for LLMs

Daniel Leone on Unsplash

Scaling laws for LLMs

The loss L of a LLM scales as a **power-law** with each of the three factors below, if the other two are not bottlenecking

- N: the number of parameters of the neural model (excluding embeddings)
- D: the amount of data available for the training process
- C: the computing power used to train the neural model

For optimal performance all three factors must be scaled up in tandem. These relationships are known as **scaling laws**.

Other architectural details, such as network width, depth, or number of heads, have **minimal** effects within a wide range.

Scaling laws for LLMs

Power-laws have the form $f(x) = ax^{-k}$, k some constant

- early increase in resources leads to large increase in performance (strong loss decrease)
- late increase in resources leads to small increase in performance (small loss decrease)

When plotted in log-scale, scaling lows become downward sloping **linear** functions.

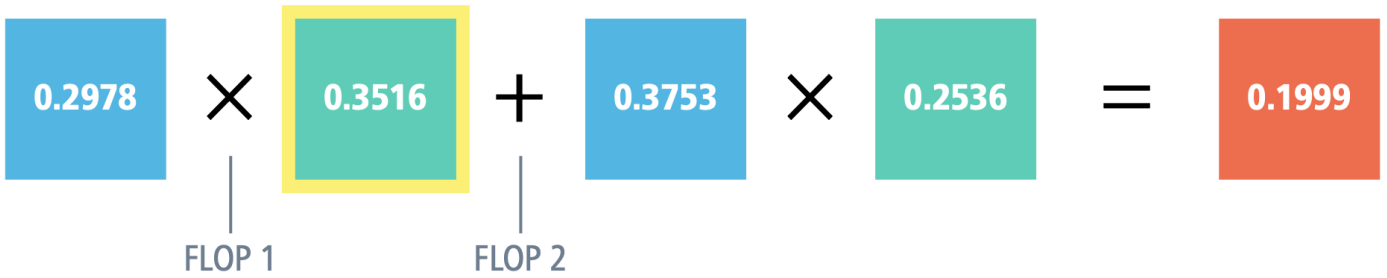
Scaling laws allow us to determine the **optimal allocation** of data and computational resources.

Scaling laws for LLMs

For a single token and a single parameter, the number F of FLOPs (floating point operations) at training time is (approximately)

$$F = F_{\text{fwd}} + F_{\text{bwd}}$$
$$F_{\text{fwd}} = 2$$
$$F_{\text{bwd}} = 4$$


parameter
token 1
0.2978
0.3753
0.3981
0.0685
×
0.3516
0.6527
0.2536
0.7357
=
0.1999
0.4705
0.1573
0.3102


0.2978
×
0.3516
+
0.3753
×
0.2536
=
0.1999
FLOP 1
FLOP 2

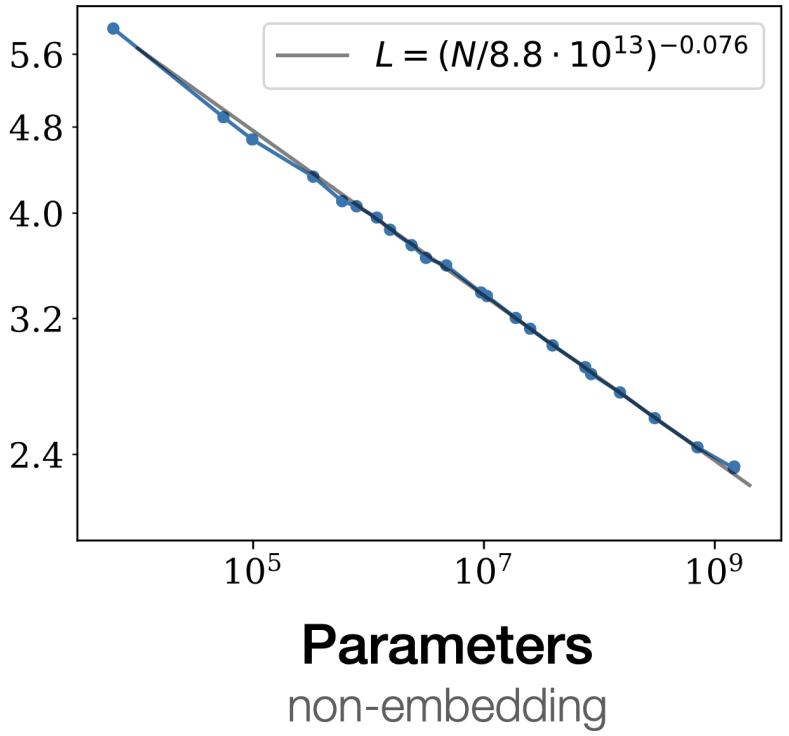
Scaling laws for LLMs

We can then relate total cost C to N and D

$$\begin{aligned} C &= (F_{\text{fwd}} + F_{\text{bwd}})ND \\ &= 6ND \end{aligned}$$

The standard unit for measuring computational cost C is PetaFLOP-days, that is, the number of **floating point** operations per day (which depends on the architecture).

Scaling laws for LLMs

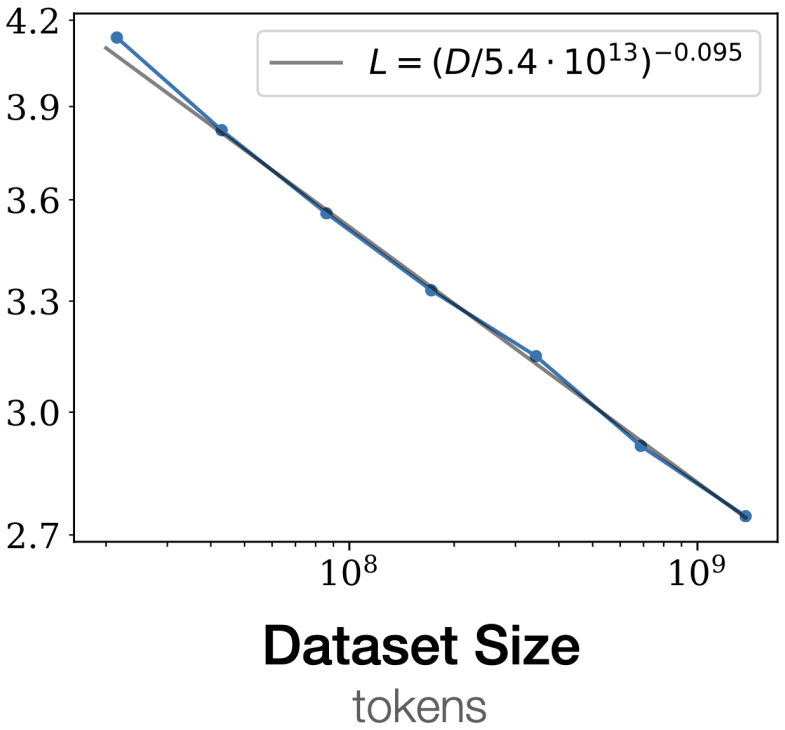


Kaplan *et al.*, 2020

For models with a limited number of parameters, trained to convergence on sufficiently large datasets

$$L(N) \sim \left(\frac{N_c}{N}\right)^{\alpha_N}$$

Scaling laws for LLMs

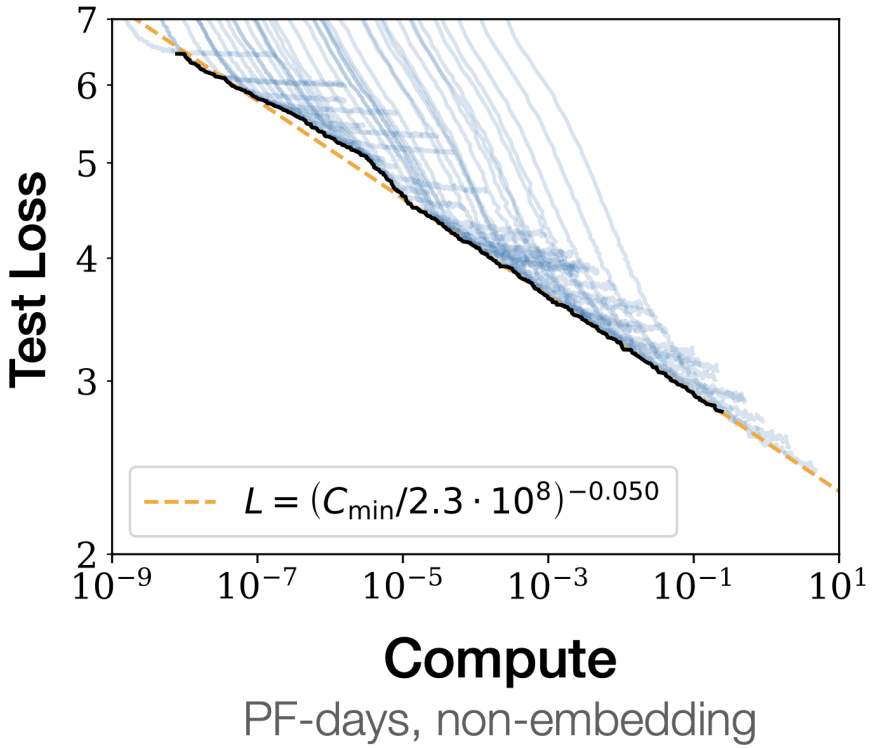


Kaplan *et al.*, 2020

For models trained with a limited dataset with early stopping

$$L(D) \sim \left(\frac{D_c}{D}\right)^{\alpha_D}$$

Scaling laws for LLMs



Kaplan *et al.*, 2020

When training with a limited amount of compute

$$L(C_{\text{min}}) \sim \left(\frac{C_c^{\text{min}}}{C_{\text{min}}}\right)^{\alpha_C^{\text{min}}}$$

Scaling laws for LLMs

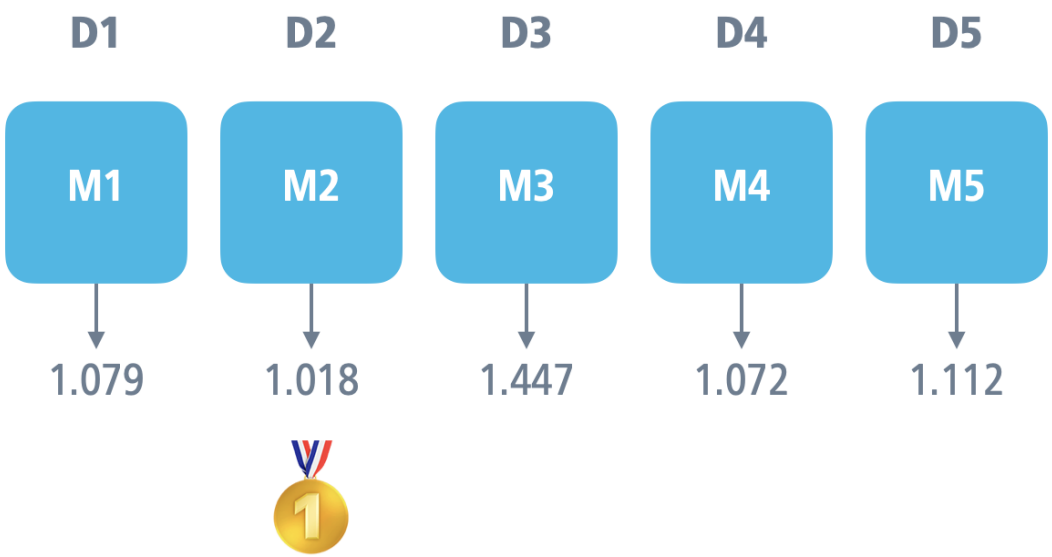
Loss of a model while varying N and D simultaneously

$$L(N, D) \sim \left[\left(\frac{N_c}{N} \right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D} \right]^{\alpha_D}$$

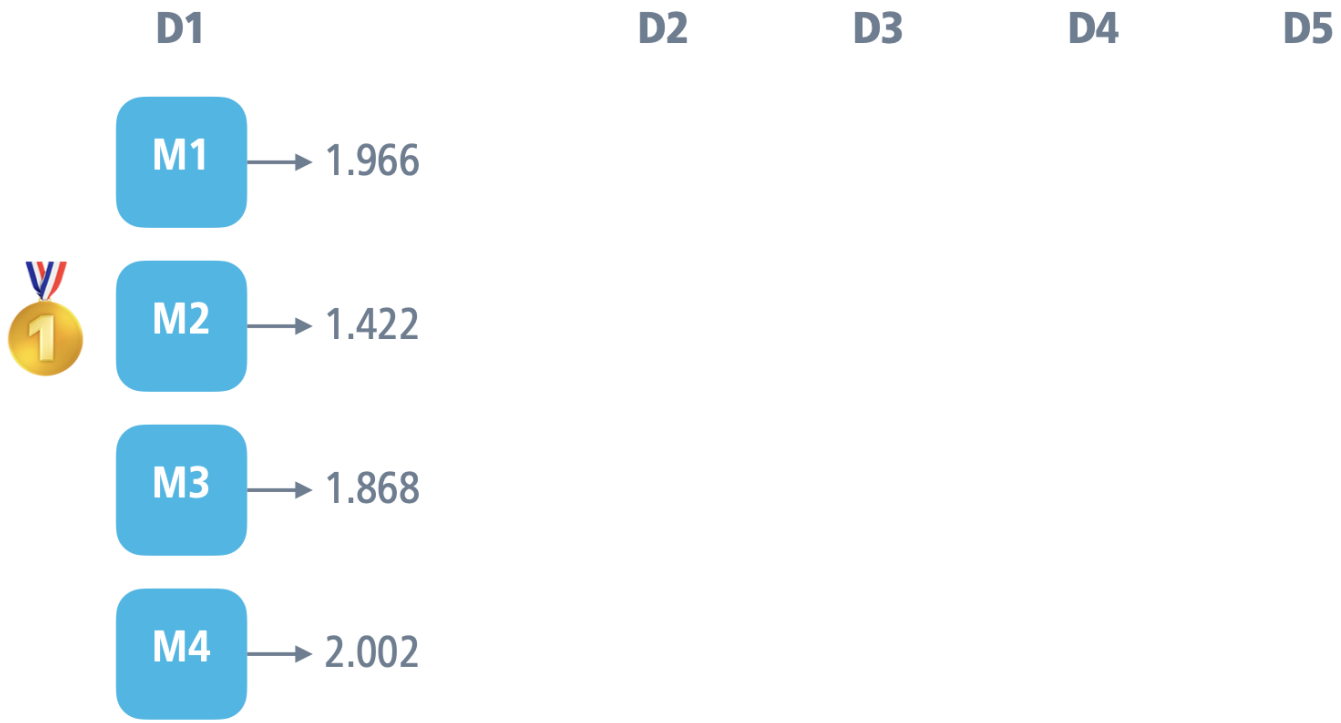
This is used to derive, for a given N, the minimum size of D that is needed to avoid overfitting.

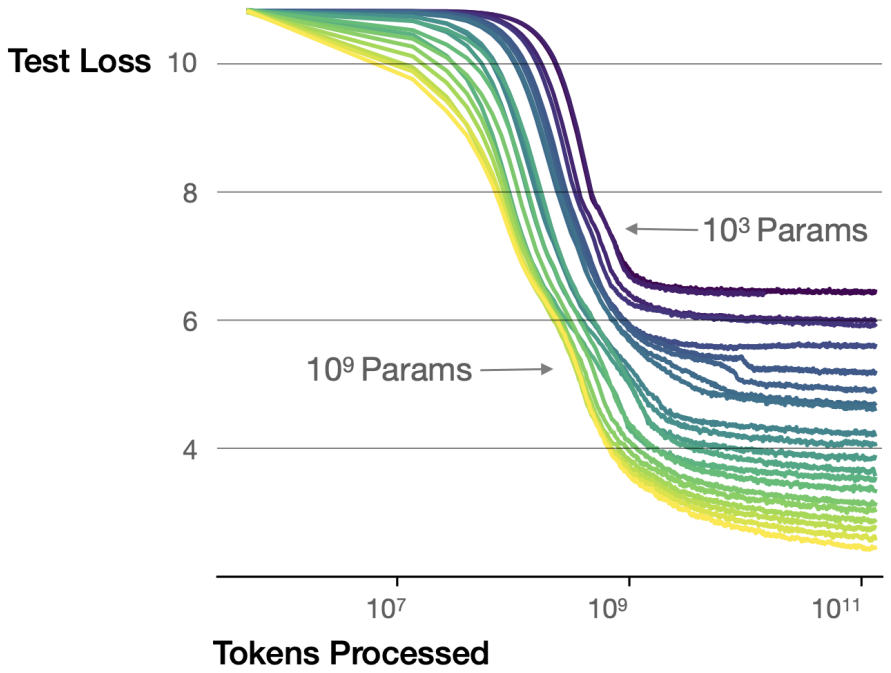
We can now use scaling laws to improve training strategies.

Old paradigm: train a few models, select the best one



New paradigm: train many small models, up-scale the best one.





Larger models require fewer samples to reach the same performance.

Overview of LLMs

Swapnil Vithaldas on Unsplash

Overview of LLMs

A language model is called **large language model** (LLM) when

- it employs a Transformer neural architecture
- it uses a number of parameters greater than one billion
- it has been trained on a very large amount of text

LLMs are at the basis of modern applications of **generative artificial intelligence**.

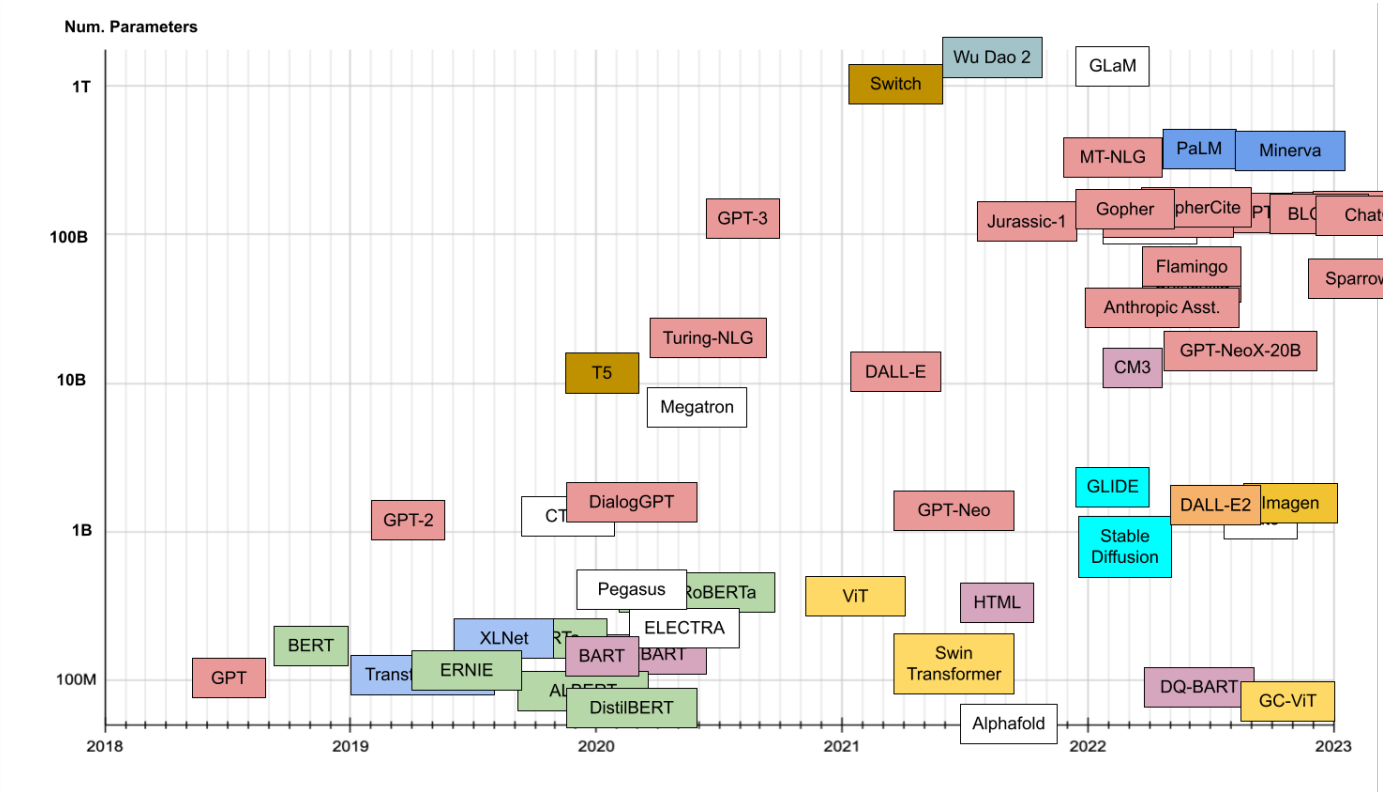
Overview of LLMs

Most popular corporations producing LLM:

- OpenAI (GPT-3, GPT-4, GPT-4)
- Google (T5, Palm, Gemini 3)
- Meta (Llama 3, Llama 4, opensource)
- Anthropic (Claude)
- DeepSeek (DeepSeek V3)
- xAI (Grok)

Overview of LLMs

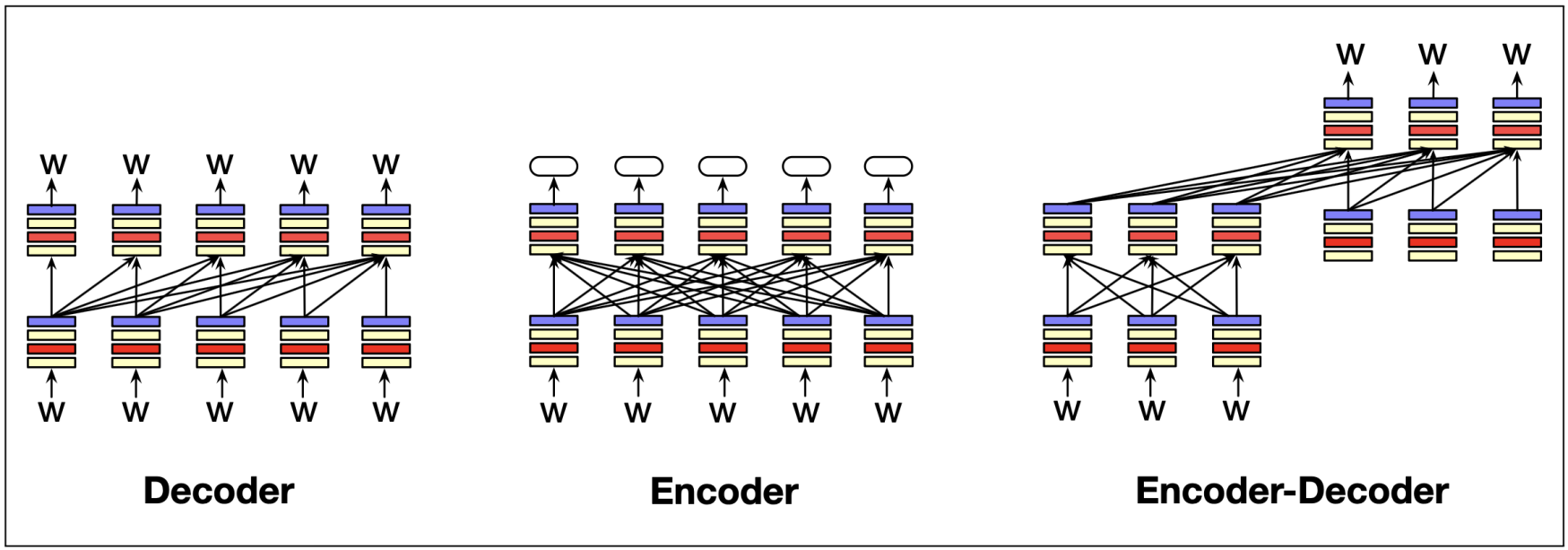
A timeline for LLMs plotted against number of parameters



https://amatriain.net/blog/transformer-models-an-introduction-and-catalog-2d1e9039f376/

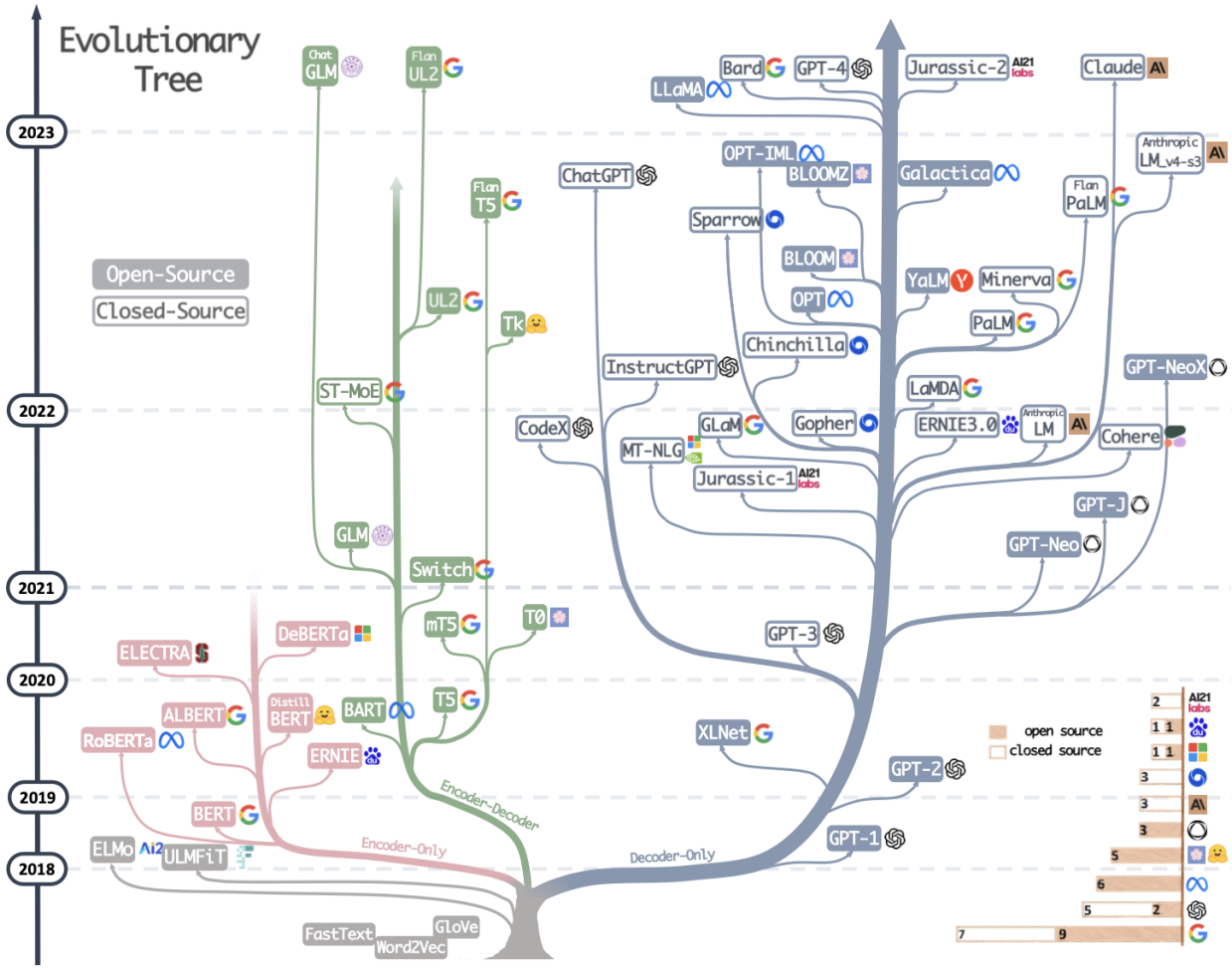
Overview of LLMs

Classification for LLMs:
encoder-only, decoder-only, encoder-decoder.



Overview of LLMs

Classification for LLMs:
encoder-only, decoder-only, encoder-decoder.



Yang et al., Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond 3

Multi-lingual LLMs

Konstantin Kleine on Unsplash

Multi-lingual LLMs

Multilingual large language models (MLLMs) are typically based on the encoder part of the transformer architecture.

A MLLM is pre-trained using large amounts of unlabeled data from multiple languages, ranging from 10 to 100.

Very useful in **transfer learning** (see later), where low resource languages can benefit from high resource languages.

Multi-lingual LLMs

Most popular MLLM

- mBERT (multimingual BERT) and several variants
- XLM (cross-lingual language model)
- mT5 (multilingual T5)
- BLOOM, the largest multilingual open-source model to date

  Trained using data from 46 languages; developed in 2022 by several groups, including HuggingFace, Microsoft, NVIDIA and PyTorch

Training of MLLMs

During pretraining, MLLMs use two different sources of data

- large monolingual corpora in individual languages
- parallel corpora between some languages

Objective functions can be based on monolingual data alone by

- pooling together monolingual corpora from multiple languages
- applying masked language modeling (MLM)

Surprisingly, the encoder learns word representations across languages that are **aligned** (close vectors), without the need for any parallel corpora.

Training of MLLMs

Objective functions based on parallel corpora **explicitly** force representations of corresponding words across languages to be close to each other in the multilingual encoder space.

Using MLM to predict a masked word `[MASK]` for language A the system can

- rely on words for language A surrounding `[MASK]`
- rely on an **aligned** word that has not been masked, that is, a word representing a translation in language B of the masked word

Training of MLLMs

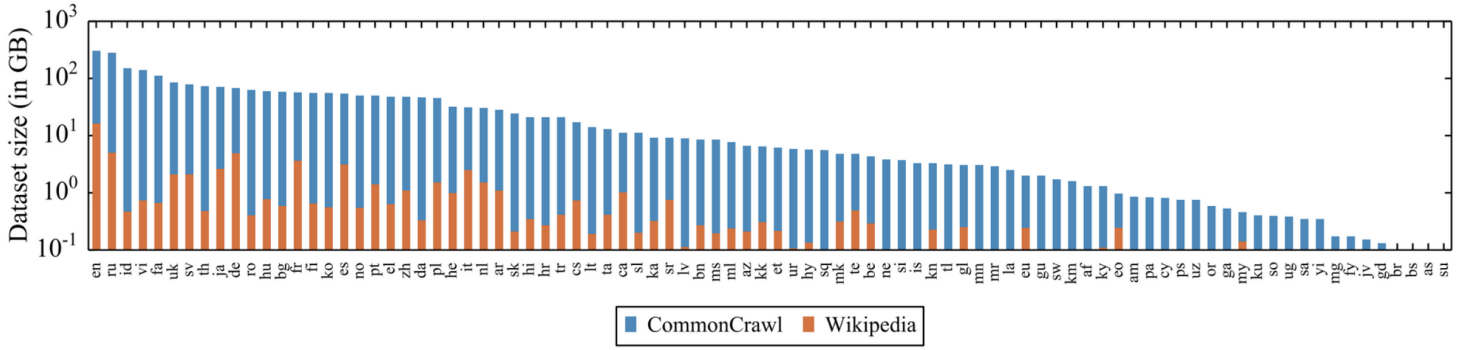
Since parallel corpora are generally much smaller than monolingual corpora, the parallel objectives are often used in conjunction with monolingual models.

Curse of multilinguality: Similar to models that are trained on many tasks, the more languages a model is pre-trained on, the less model capacity is available to learn representations for each language.

Multi-lingual LLMs

English language is overrepresented.

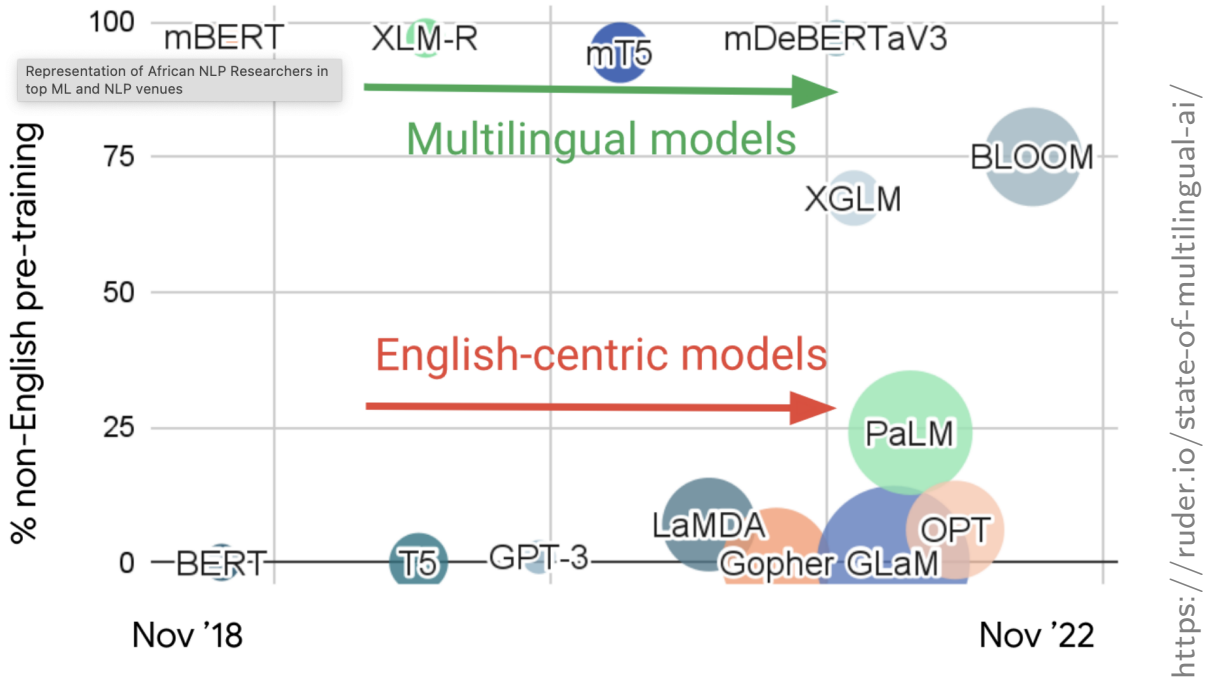
Amount of data in GB (log-scale) for the 88 languages that appear in CommonCrawl and Wikipedia.



Conneau *et al.*, 2020

Multi-lingual LLMs

The largest recent models are not becoming significantly more multilingual.



Miscellanea

Kristine Rosenblatt, Kristine's Kitchen

Evaluation of LLM

Standard intrinsic evaluation measure is **perplexity**: good models assign higher probabilities to unseen data.

Already introduced in a previous lecture.

Caveat: hard to exactly compare perplexities produced by two language models if they use very different tokenizers.

Extrinsic evaluation discussed in later lecture for specific nlp tasks.

Other important factors in LLM evaluation: pretraining cost, environmental impact, and fairness.

Emergent abilities

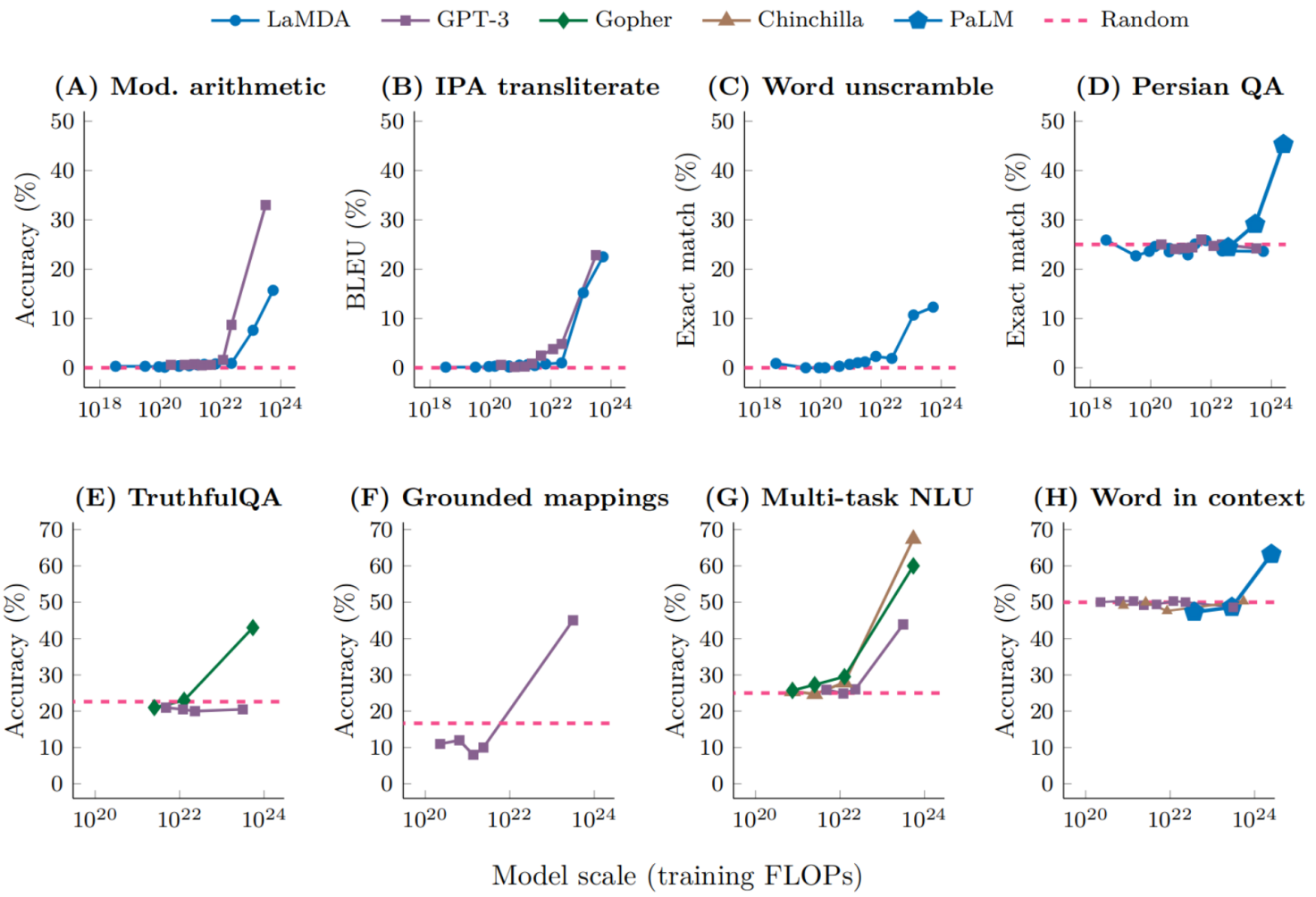
Emergent abilities of large language models are characterized by

- sharpness: transitioning seemingly instantaneously from not present to present
- unpredictability: transitioning at seemingly unforeseeable model scales

This issue is still under debate and controversial.

It has been argued that emergent abilities might be induced by wrong choice of metrics.

Emergent abilities



Wei et al., Emergent abilities of large language models, 2022

Potential harms from LLM

LLM exhibit many potential harms

- hallucination: see next slide
- copyright violation
- toxic language and hate speech
- information leak from training data

Finding ways to mitigate these harms is an important current research area in NLP.

It is extremely important that LLM include **model cards**, giving full information on the corpora used to train them.

Hallucinations

Hallucination refers to a phenomenon where a LLM generates text that is incorrect, nonsensical, or not real.

Since LLMs are not databases or search engines, they would not cite where their response is based on.

Since LLMs need to be creative, they must be able to invent scenarios that are not factual.

To avoid hallucinations use **controlled generation**: providing enough details and constraints in the prompt to the model.

Prompting presented in a later lecture.

Mixture of Experts

Mixture of Experts (MoE) is a machine learning technique where multiple expert networks (learners) are used to divide a problem space into homogeneous regions.

MoE differs from ensemble techniques in that typically **only one model** will be run, rather than combining results from all models.

This technique has been recently used in LLMs. The result is a sparsely-activated model with a large number of parameters but a constant computational cost.