ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Intro to LLMs

Đinh Viết Sang
Foundation Models Labs
BKAI

ONE LOVE. ONE FUTURE.

Contents

Language Models and LLMs

Language Models

- A probabilistic model p is a probability distribution over a sequence of tokens $\{w_1, w_2, \ldots, w_n\}$, where each token comes from some vocabulary V.

Conditional probability:
$p(w \mid w_1, w_2), \forall w \in V$

$$p(w_1, w_2, w_3, \ldots, w_n) =$$

$$p(w_1)p(w_2 \mid w_1)p(w_3 \mid w_1, w_2) \times \cdots \times p(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

Sentence: "the cat sat on the mat"

$$P(\text{the cat sat on the mat}) = P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat})$$
$$*P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on})$$
$$*P(\text{mat}|\text{the cat sat on the})$$

LLMs: the model is implemented as a very large neural network!

Tokenization

- A **tokenizer** converts any string into a sequence of tokens.

  the mouse ate the cheese $\Rightarrow$ [*the*, *mouse*, *ate*, *the*, *cheese*]

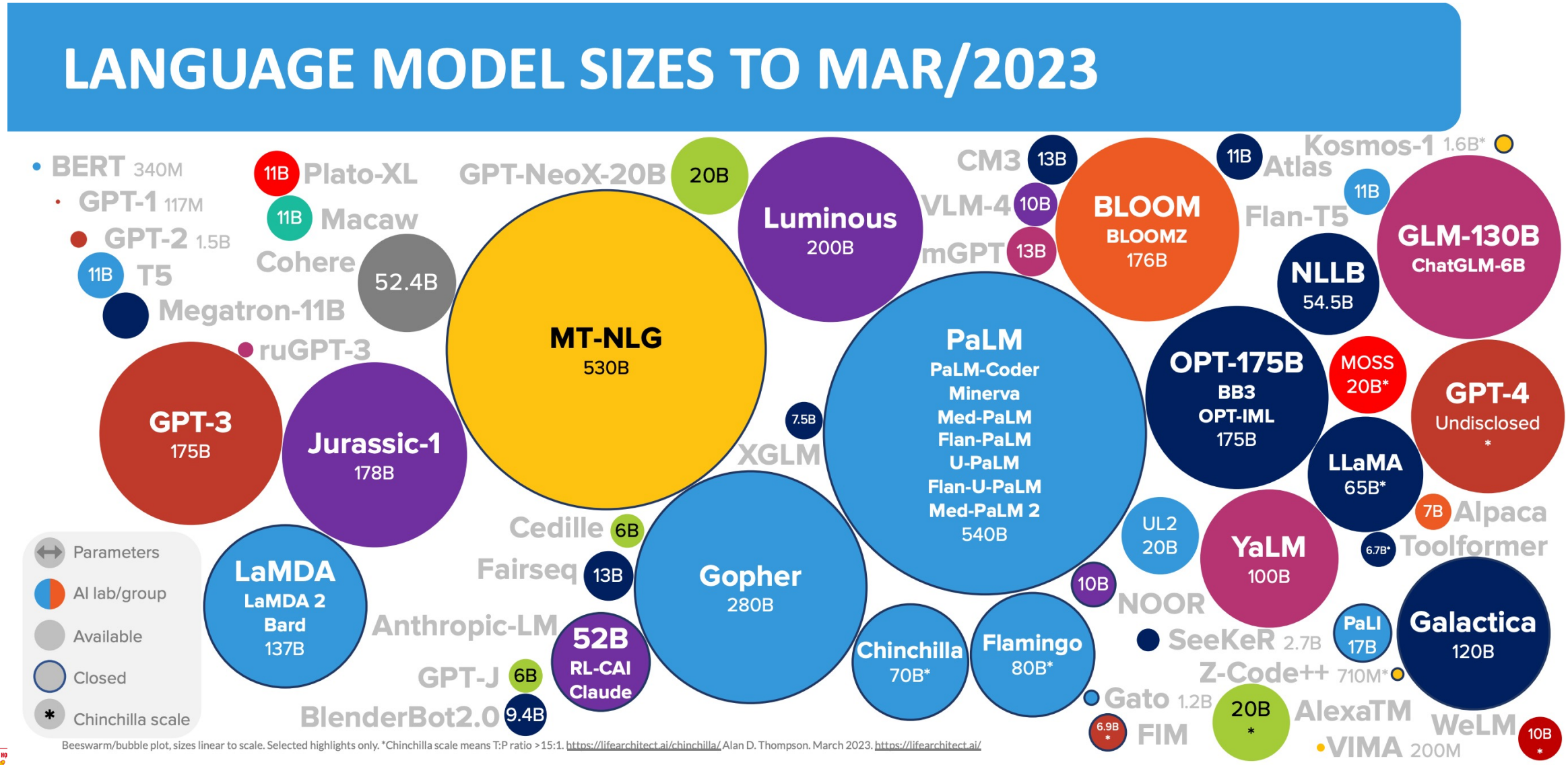
- **Split by spaces**

```
text.split(' ')
```

- **Byte pair encoding**
  - Input: a training corpus (sequence of characters).
  - Initialize the vocabulary $\mathcal{V}$ be the set of characters.
  - While we want to still grow $\mathcal{V}$:
    - Find the pair of elements $x, x' \in \mathcal{V}$ that co-occur the most number of times.
    - Replace all occurrences of x, x' with a new symbol xx'.
    - Add xx' to $\mathcal{V}$.

Unicode: Run BPE on bytes instead of 144,697 Unicode characters

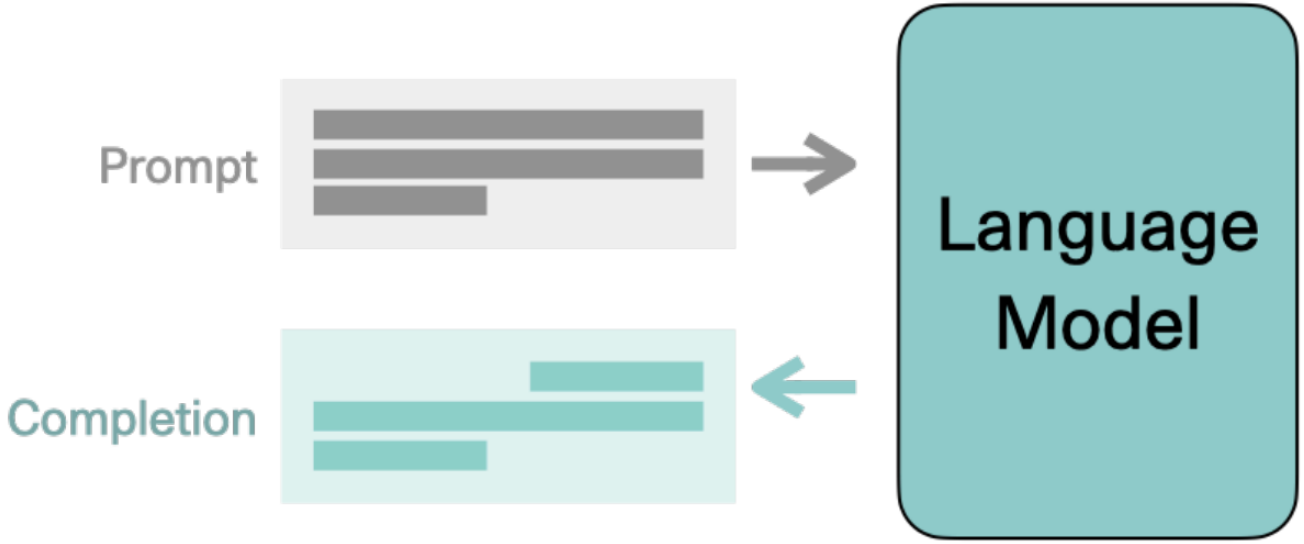
How large are “large” LMs?



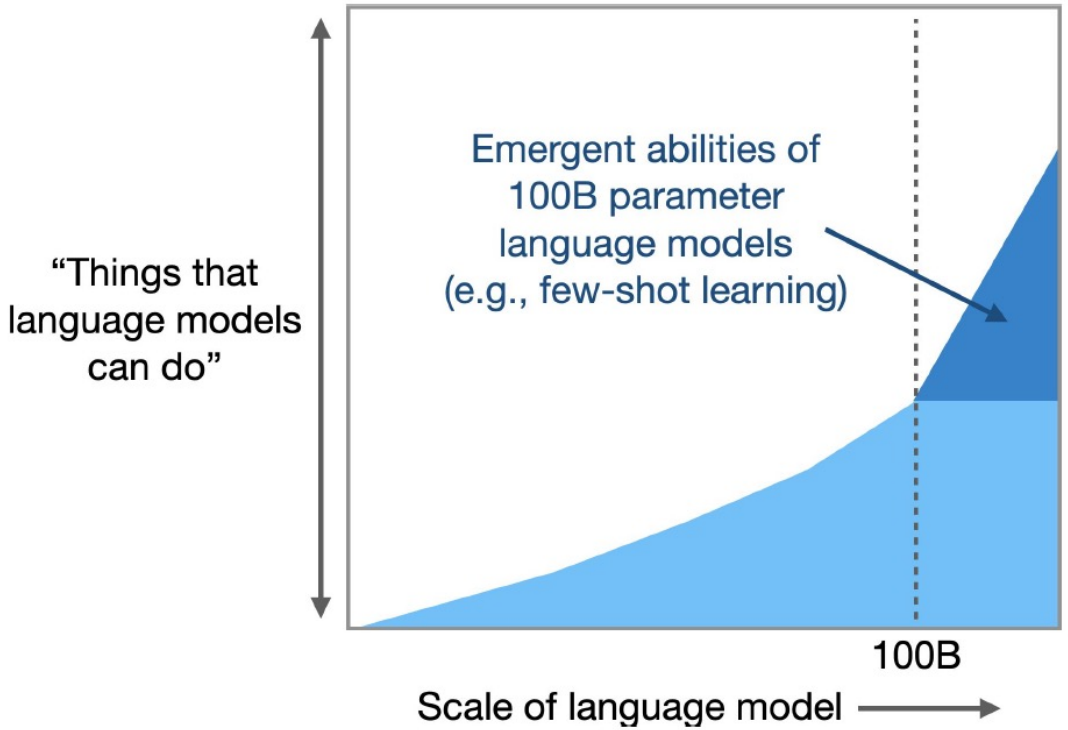
https://lifearchitect.ai/models

Why LLMs?

- One single model to solve many NLP tasks



- Emergent properties



Attentions and Transformers

Attention

- A more general form: use a set of keys and values $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_n, \mathbf{v}_n), \mathbf{k}_i \in \mathbb{R}^{d_k}, \mathbf{v}_i \in \mathbb{R}^{d_v}$, keys are used to compute the attention scores and values are used to compute the output vector

- Attention always involves the following steps:
  - Computing the **attention scores** $\mathbf{e} = g(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}^n$
  - Taking softmax to get **attention distribution** α:

$$\alpha = \text{softmax}(\mathbf{e}) \in \mathbb{R}^n$$

  - Using attention distribution to take **weighted sum** of values:

$$\mathbf{a} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \in \mathbb{R}^{d_v}$$

Self-attention

A self-attention layer maps a sequence of input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_1}$ to a sequence of *n* vectors: $\mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_2}$

Step #1: Transform each input vector into three vectors: query, key, and value vectors

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q \in \mathbb{R}^{d_q} \qquad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K \in \mathbb{R}^{d_k} \qquad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V \in \mathbb{R}^{d_v}$$

$$\mathbf{W}^Q \in \mathbb{R}^{d_1 \times d_q} \qquad \mathbf{W}^K \in \mathbb{R}^{d_1 \times d_k} \qquad \mathbf{W}^V \in \mathbb{R}^{d_1 \times d_v}$$

Step #2: Compute pairwise similarities between keys and queries; normalize with softmax

For each $\mathbf{q}_i$, compute attention scores and attention distribution:

$$e_{i,j} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}, \forall j = 1, \ldots, n$$

$$\alpha_i = \mathrm{softmax}(\mathbf{e}_i)$$

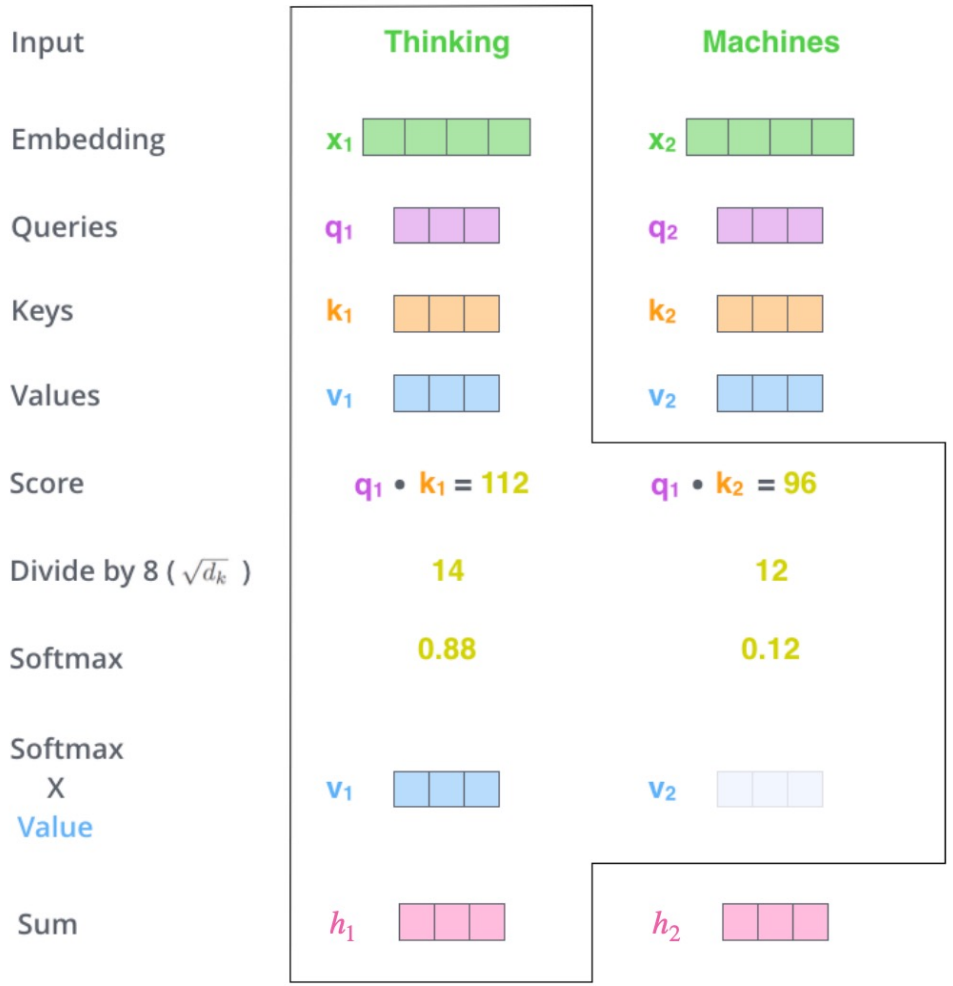
$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{n} \exp(e_{i,k})}$$

Self-attention

A self-attention layer maps a sequence of input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_1}$ to a sequence of *n* vectors: $\mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_2}$

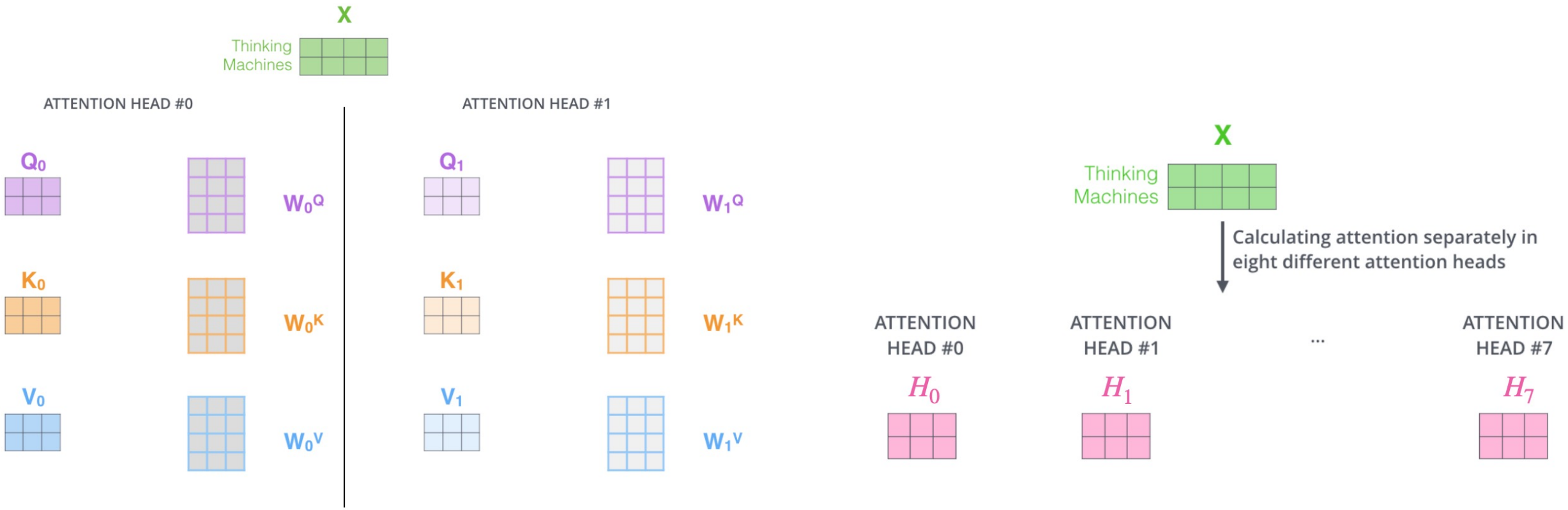
Step #3: Compute output for each input as weighted sum of values

$$\mathbf{h}_i = \sum_{j=1}^{n} \alpha_{i,j} \mathbf{v}_j \in \mathbb{R}^{d_v}$$



Multi-head Attention

- It is better to use multiple attention functions instead of one!
  - Each attention function ("head") can focus on different positions.



Positional Encoding

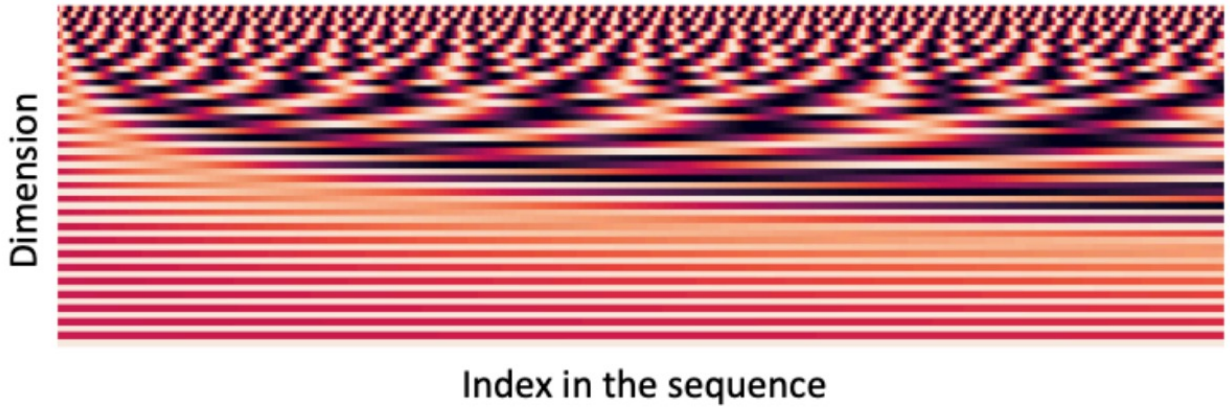
- Unlike RNNs, self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values

- Solution: Add **"positional encoding"** to the input embeddings: $\mathbf{p}_i \in \mathbb{R}^d$ for $i = 1,2,\ldots,n$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{p}_i$$

- **Sinusoidal position encoding**: sine and cosine functions of different frequencies:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



Adding Nonlinearities

- There are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors

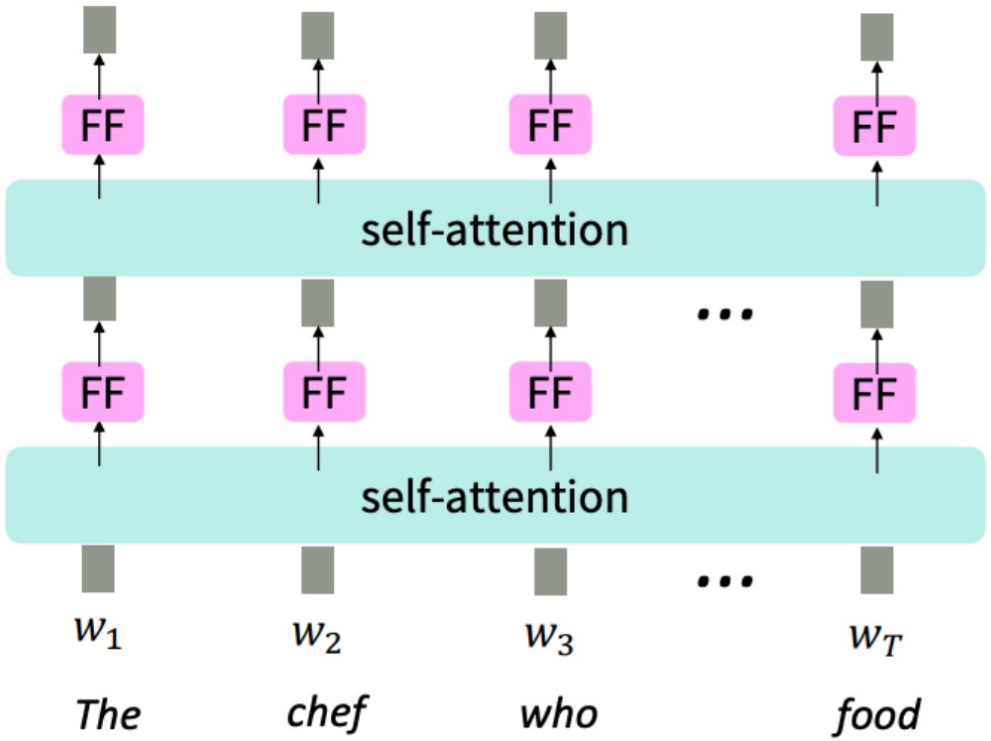
- Simple fix: add a feed-forward network to post-process each output vector

$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W_2} + \mathbf{b}_2$$

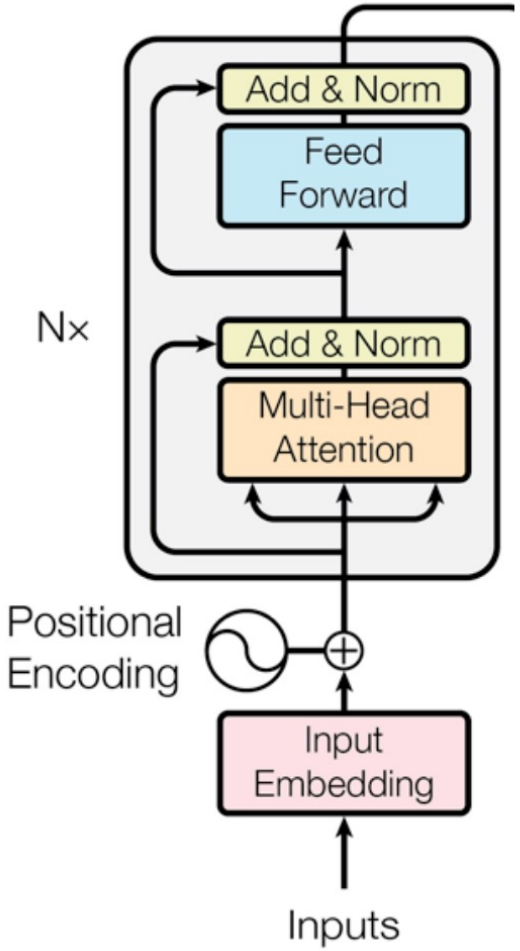
$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}, \mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$$

$$\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}, \mathbf{b}_2 \in \mathbb{R}^{d}$$

In practice, they use $d_{ff} = 4d$



Transformer Encoder



From the bottom to the top:

- Input embedding
- Positional encoding
- A stack of Transformer encoder layers

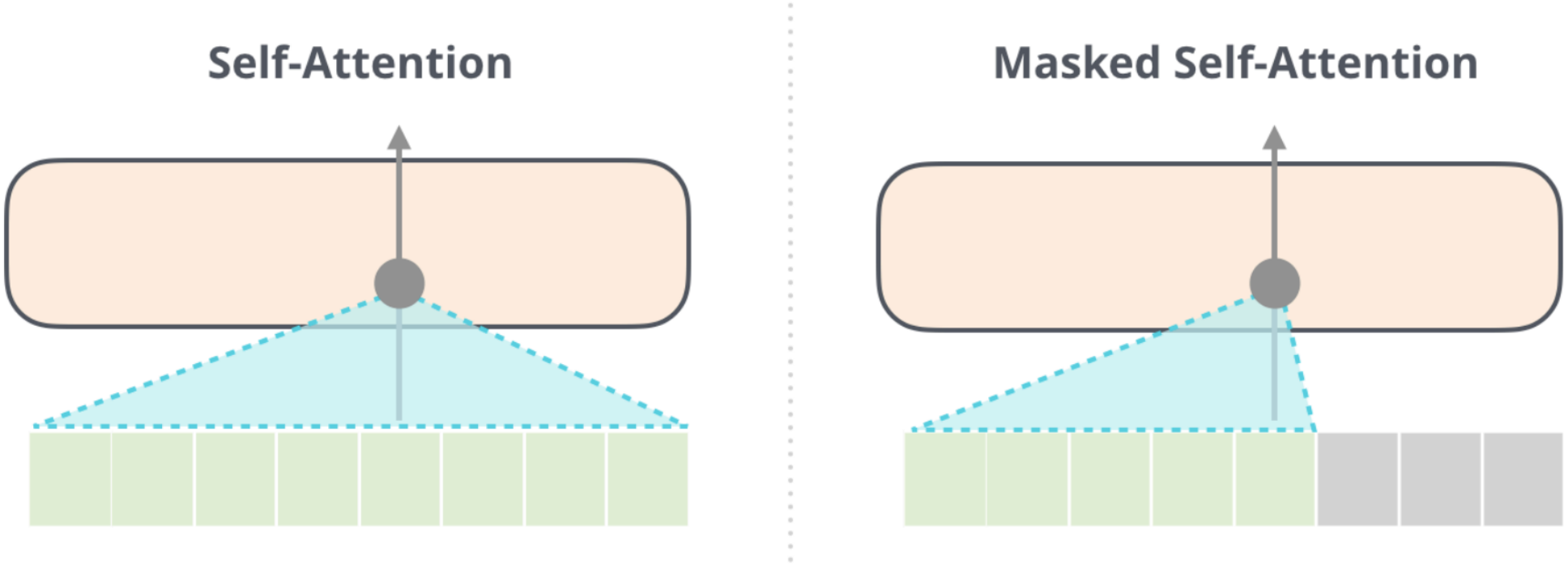
Transformer encoder is a stack of N layers, which consists of two sub-layers:

- Multi-head attention layer
- Feed-forward layer

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_1} \longrightarrow \mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_2}$$

Masked (casual) Self-attention

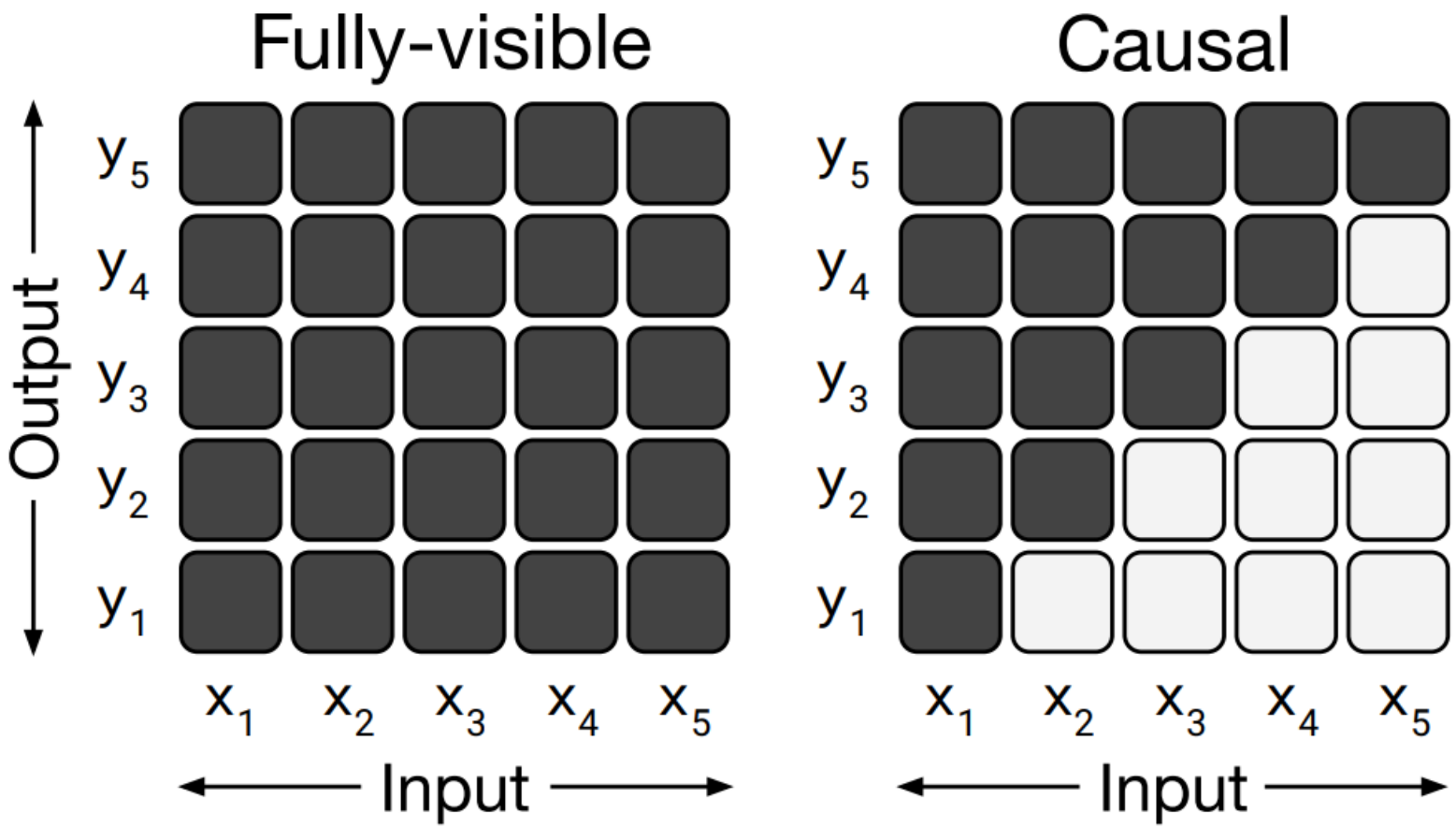
- Key: You can't see the future for the decoder!



- Solution: for every $\mathbf{q}_i$, only attend to $\{(\mathbf{k}_j, \mathbf{v}_j)\}, j \leq i$

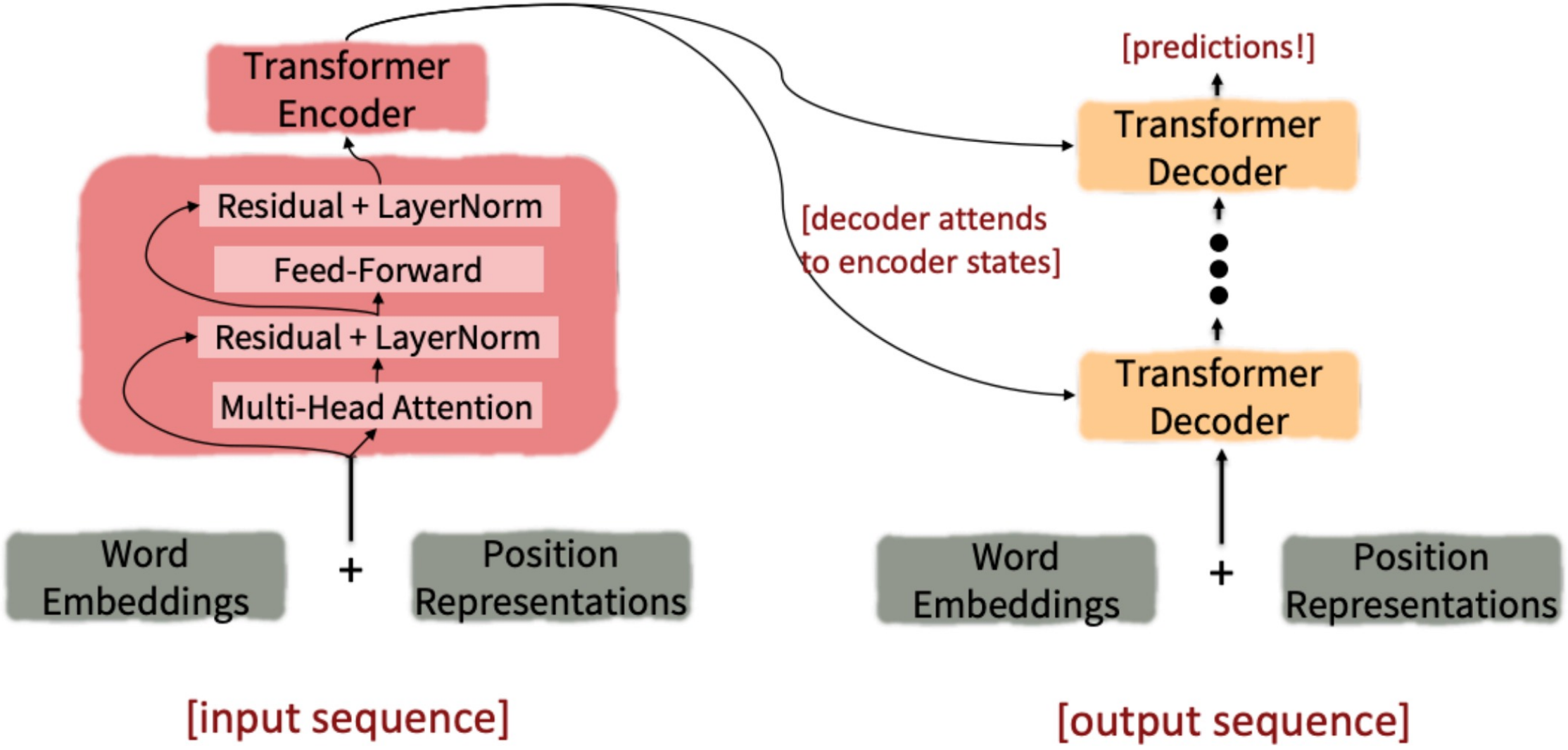
Masked (casual) Self-attention

- Key: You can't see the future for the decoder!



(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

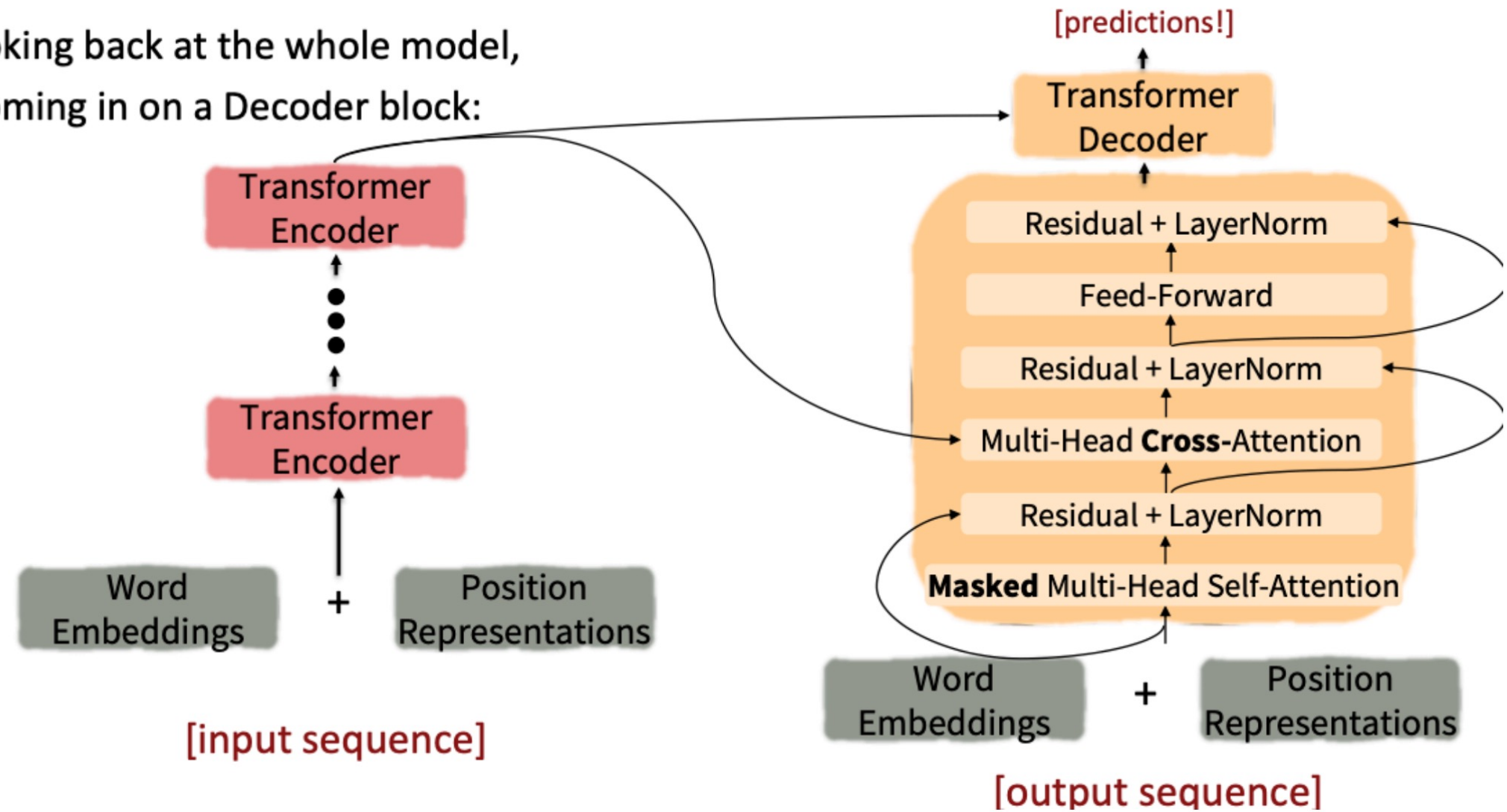
Transformer Encoder-decoder



Transformer Encoder-decoder

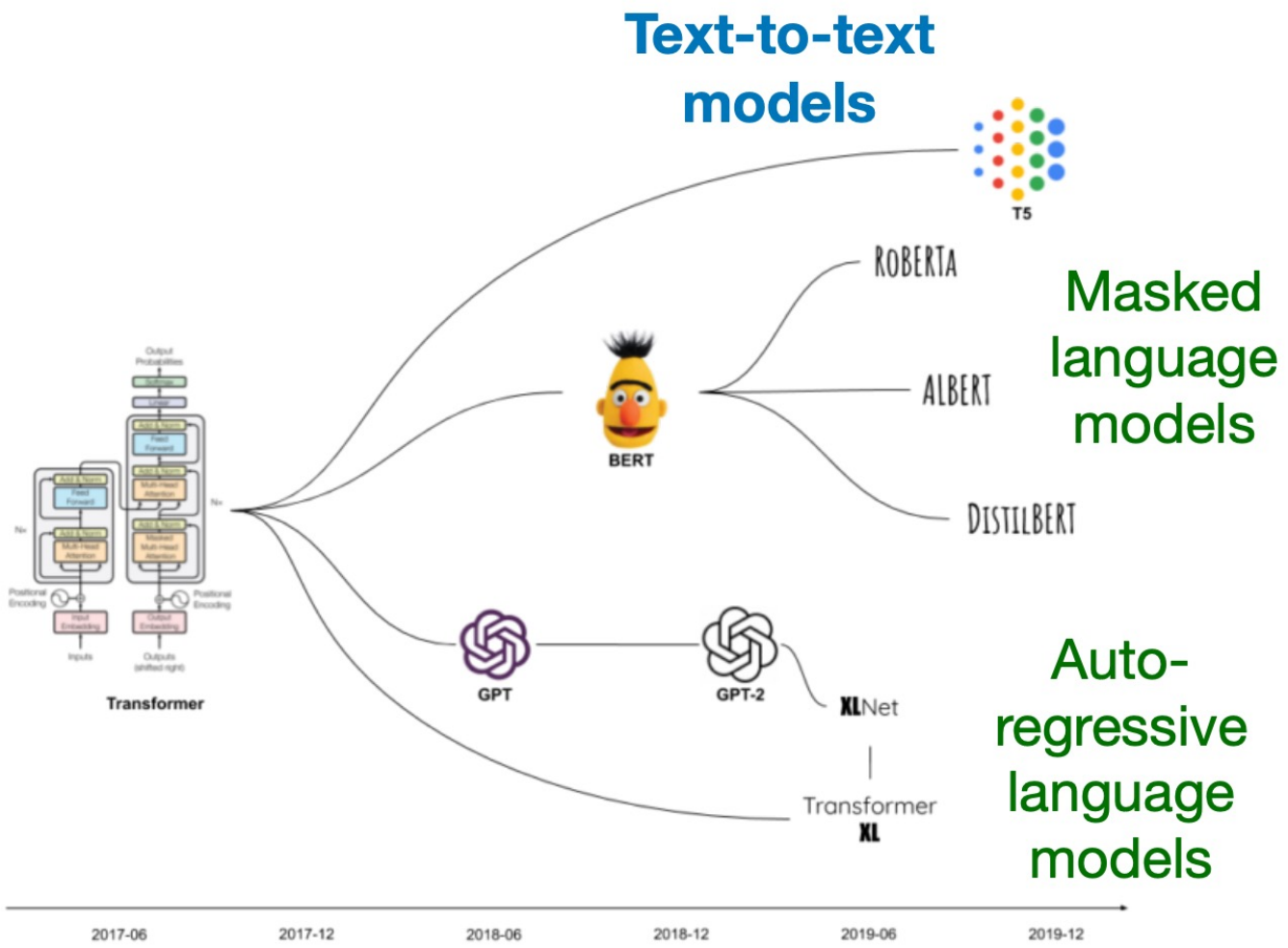
Looking back at the whole model,

zooming in on a Decoder block:



LLMs Taxonomy

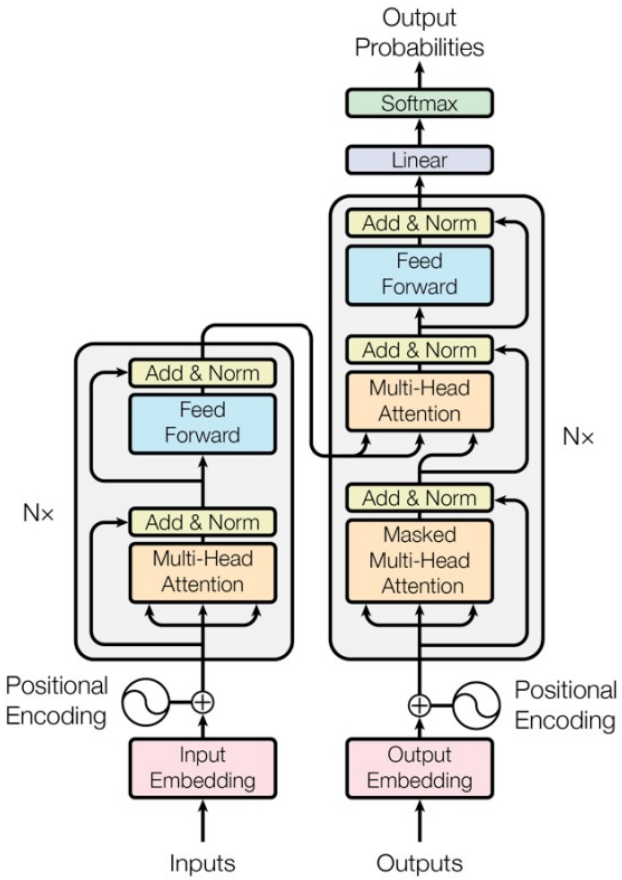
LLMs Taxonomy



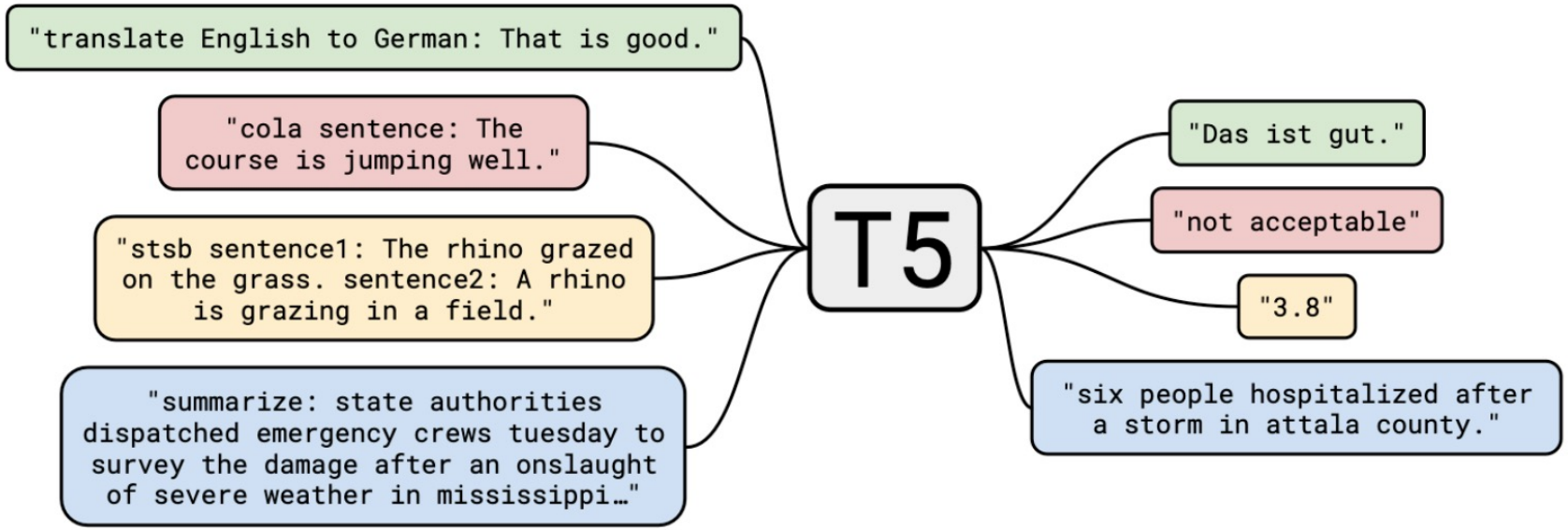
- Masked language models = Transformer encoder
- Autoregressive language models = Transformer decoder
- **Text-to-text models = Transformer encoder-decoder**

https://princeton-nlp.github.io/cos484

LLMs Taxonomy: Encoder-Decoder Models

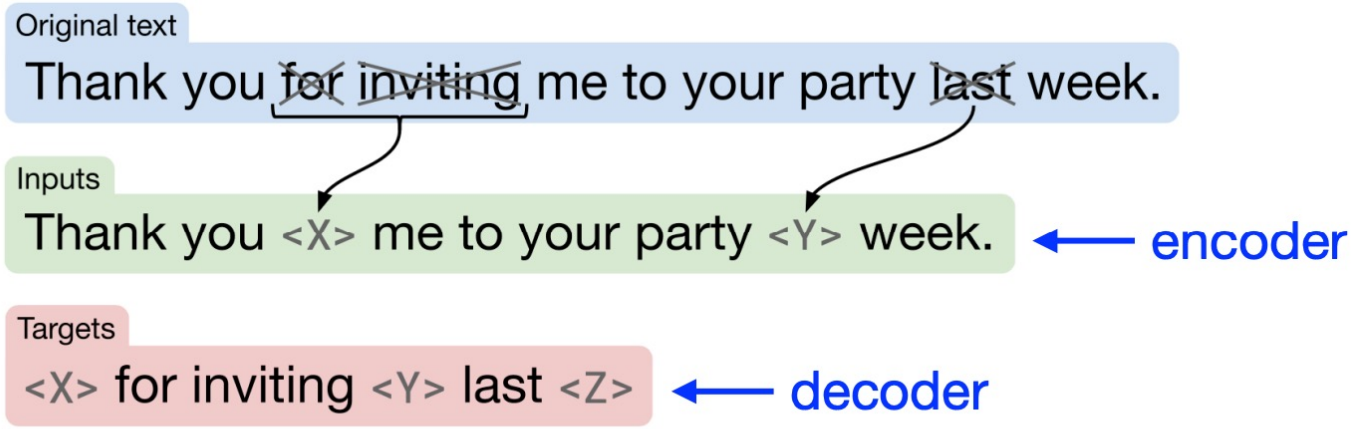


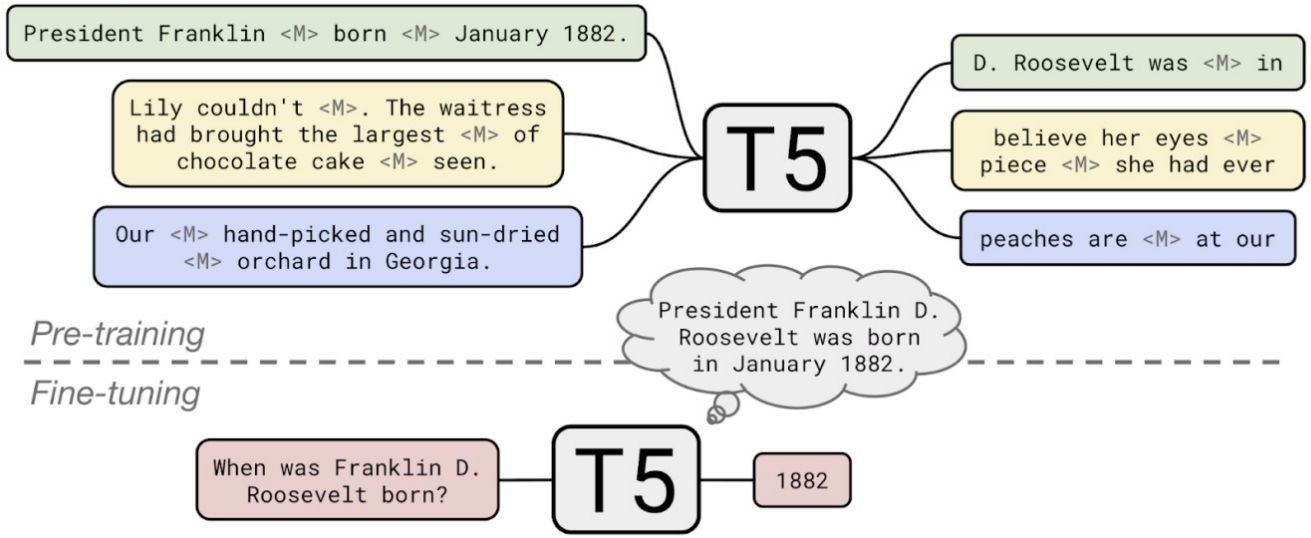
T5 = Text-to-Text Transfer Transformer



(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

LLMs Taxonomy: Encoder-Decoder Models





T5 comes in different sizes:

- t5-small.
- t5-base.
- t5-large.
- t5-3b.
- t5-11b.

(Raffel et al., 2020): Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

LLMs Taxonomy: Encoder-Only Models (Masked Language Modeling)

- Q: Why we can't do language modeling with bidirectional models?

- Solution: Mask out k% of the input words, and then predict the masked words

store gallon

the man went to [MASK] to buy a [MASK] of milk

k = 15% in practice

MLM: 80-10-10 corruption

For the 15% predicted words,

- 80% of the time, they replace it with [MASK] token

  went to the store ⟶ went to the [MASK]

- 10% of the time, they replace it with a random word in the vocabulary

  went to the store ⟶ went to the running

- 10% of the time, they keep it unchanged

  went to the store ⟶ went to the store

Why? Because [MASK] tokens are never seen during fine-tuning

Next Sentence Prediction (NSP)

- Motivation: many NLP downstream tasks require understanding the relationship between two sentences (natural language inference, paraphrase detection, QA)
- NSP is designed to reduce the gap between pre-training and fine-tuning

[CLS]: a special token always at the beginning

[SEP]: a special token used to separate two segments

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

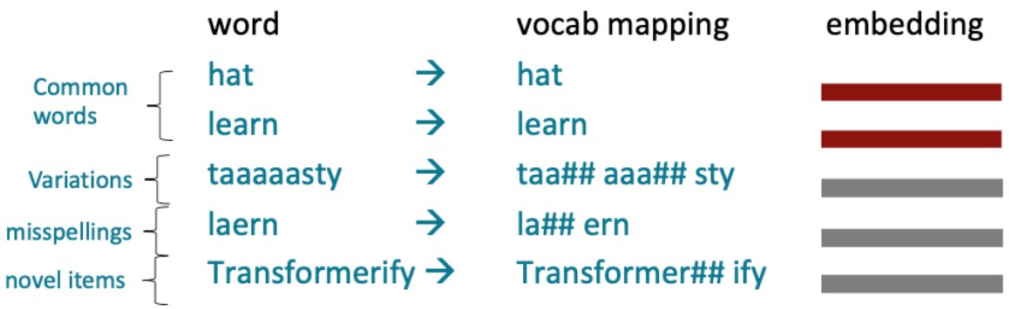
penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

They sample two contiguous segments for 50% of the time and another random segment from the corpus for 50% of the time

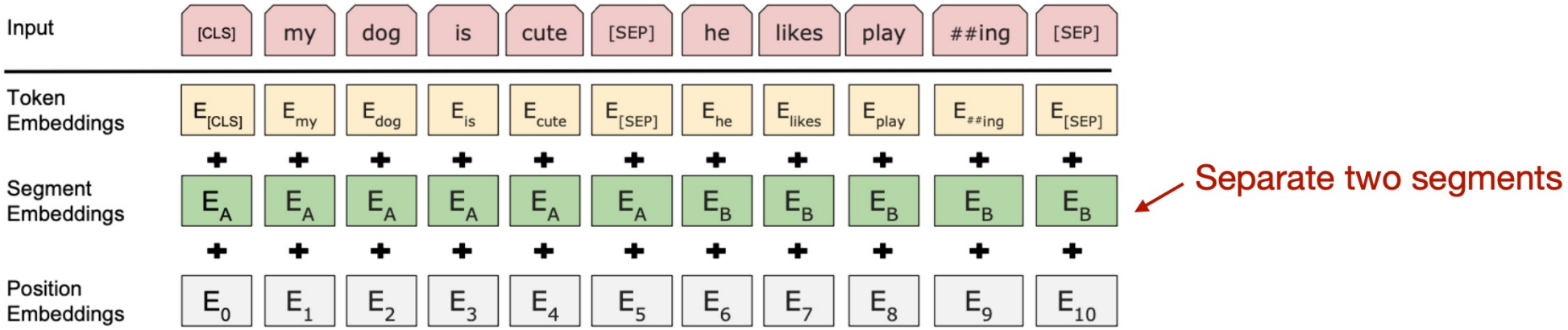
BERT pre-training

- Vocabulary size: 30,000 workpieces (common sub-word units) (Wu et al., 2016)



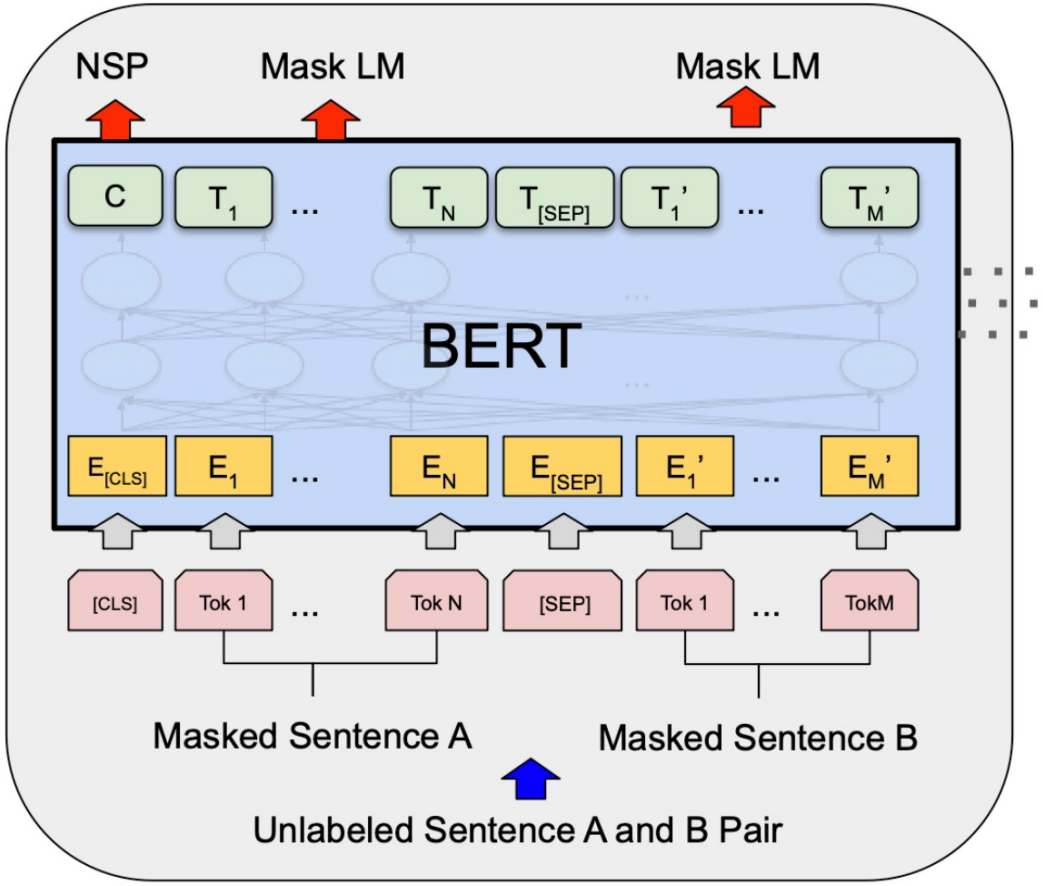
(Image: Stanford CS224N)

- Input embeddings:



https://www.cs.princeton.edu/courses/archive/fall22/cos597G

BERT pre-training

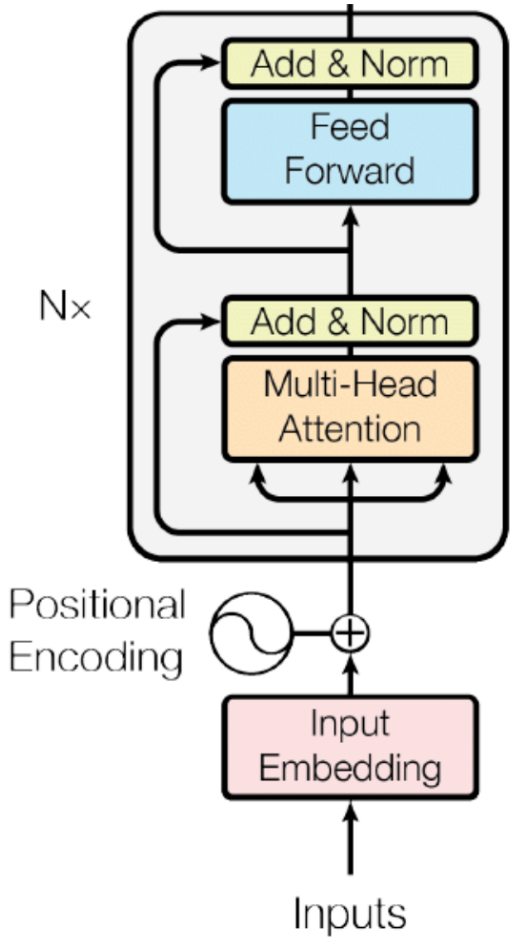


Pre-training

- MLM and NSP are trained together
- [CLS] is pre-trained for NSP
- Other token representations are trained for MLM

https://www.cs.princeton.edu/courses/archive/fall22/cos597G

BERT pre-training



- BERT-base: 12 layers, 768 hidden size, 12 attention heads, 110M parameters

Same as OpenAI GPT

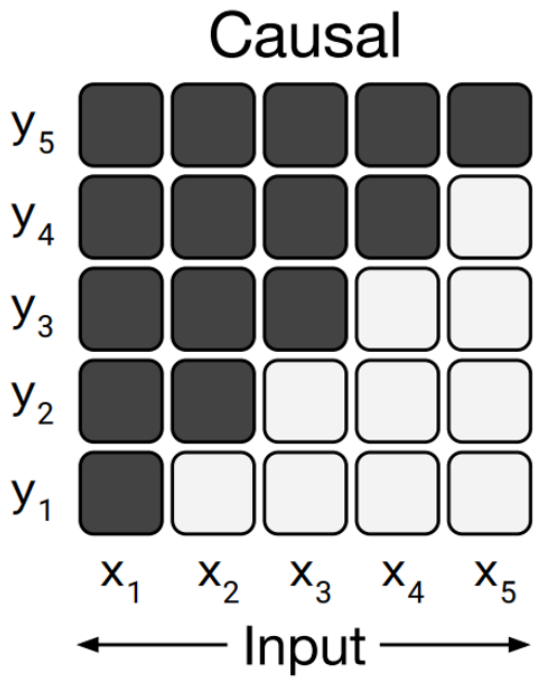
- BERT-large: 24 layers, 1024 hidden size, 16 attention heads, 340M parameters

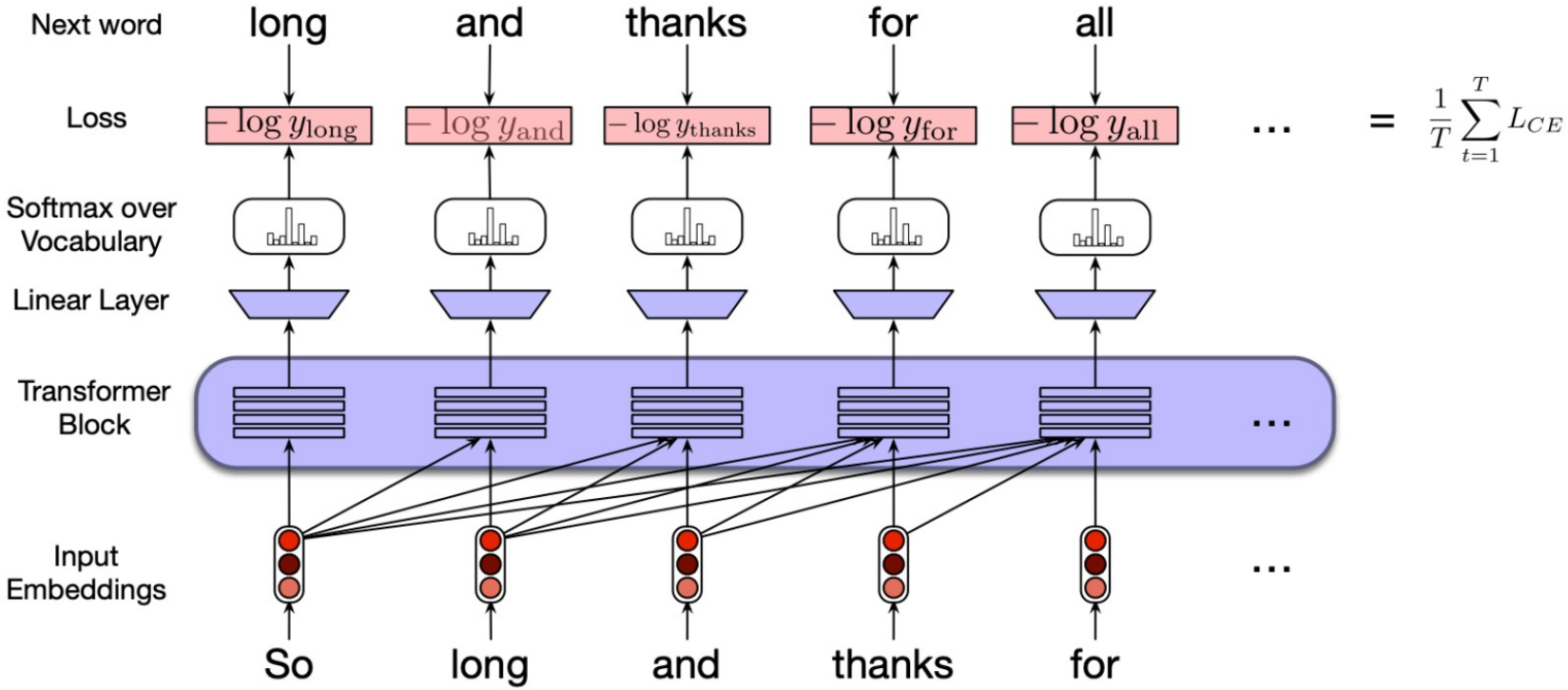
OpenAI GPT was trained on BooksCorpus only!

- Training corpus: Wikipedia (2.5B) + BooksCorpus (0.8B)
- Max sequence size: 512 word pieces (roughly 256 and 256 for two non-contiguous sequences)
- Trained for 1M steps, batch size 128k

LLMs Taxonomy: Decoder-Only Models (Autoregressive Language Modeling)

- Use a **Transformer decoder** (unidirectional; left-to-right)
- Use **language modeling** as a pre-training objective
- Trained on longer segments of text (**512 BPE tokens**), not just single sentences

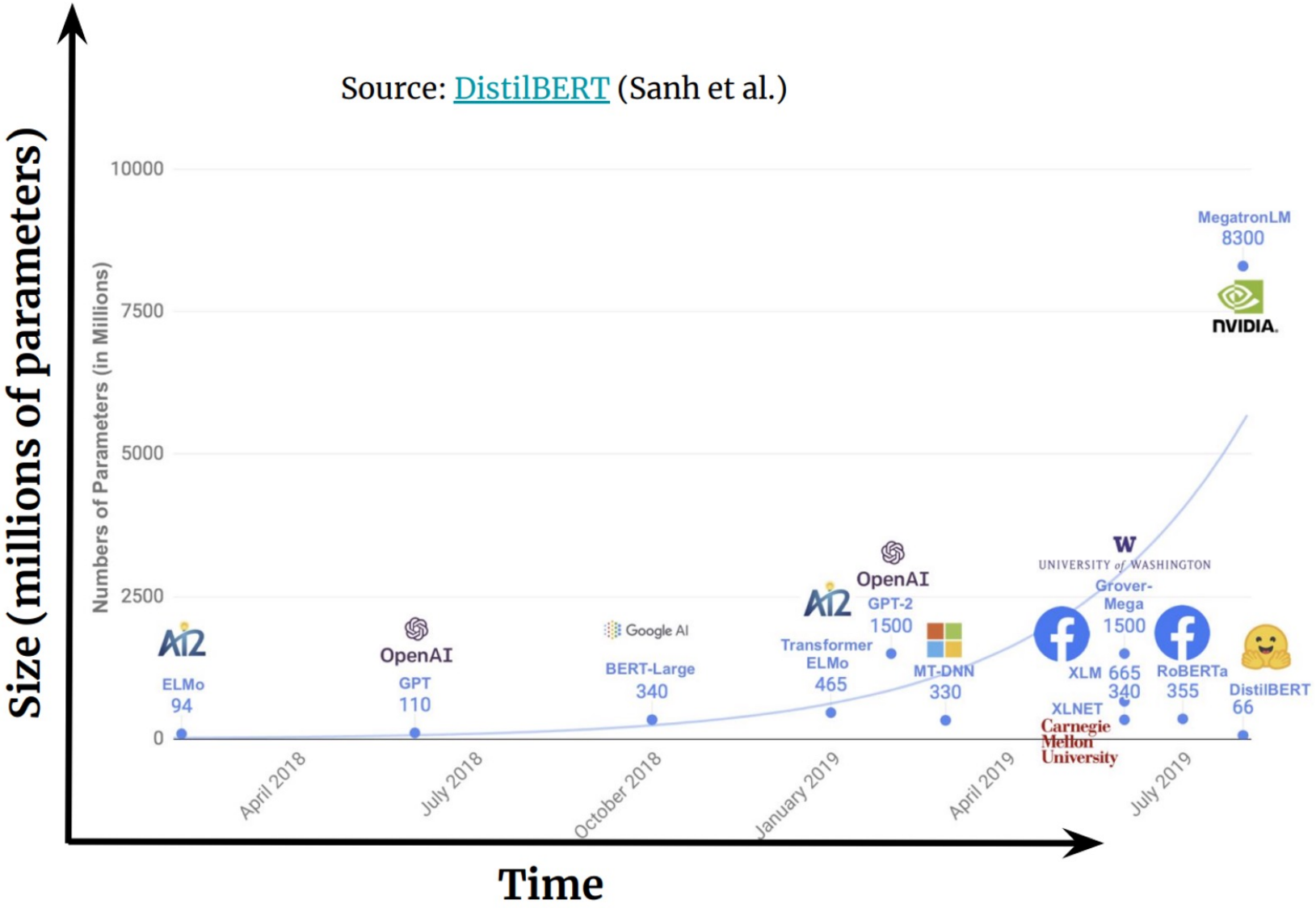




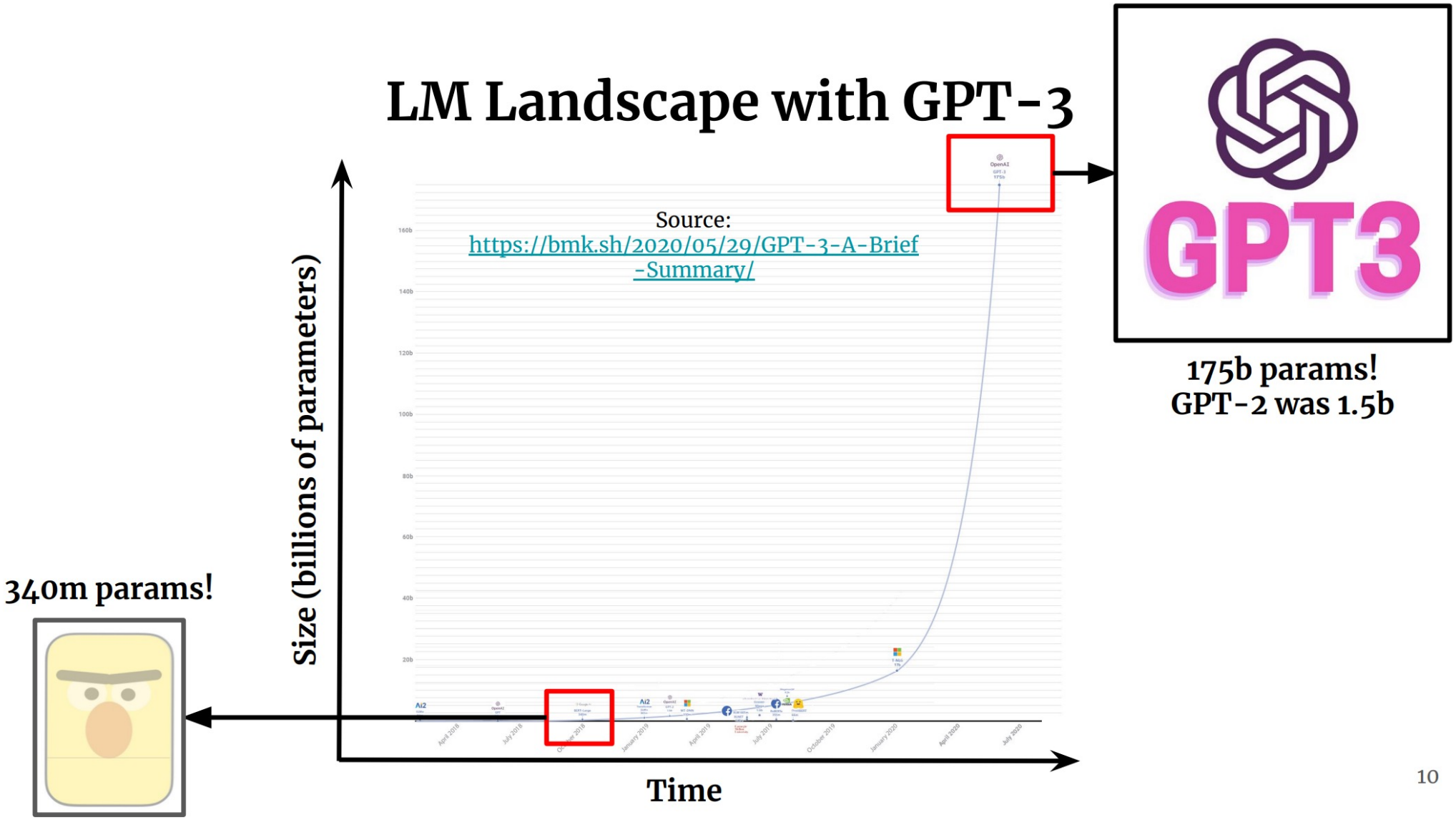
(Radford et al, 2018): Improving Language Understanding by Generative Pre-Training

Scaling laws

Scaling: LM Landscape with GPT-3

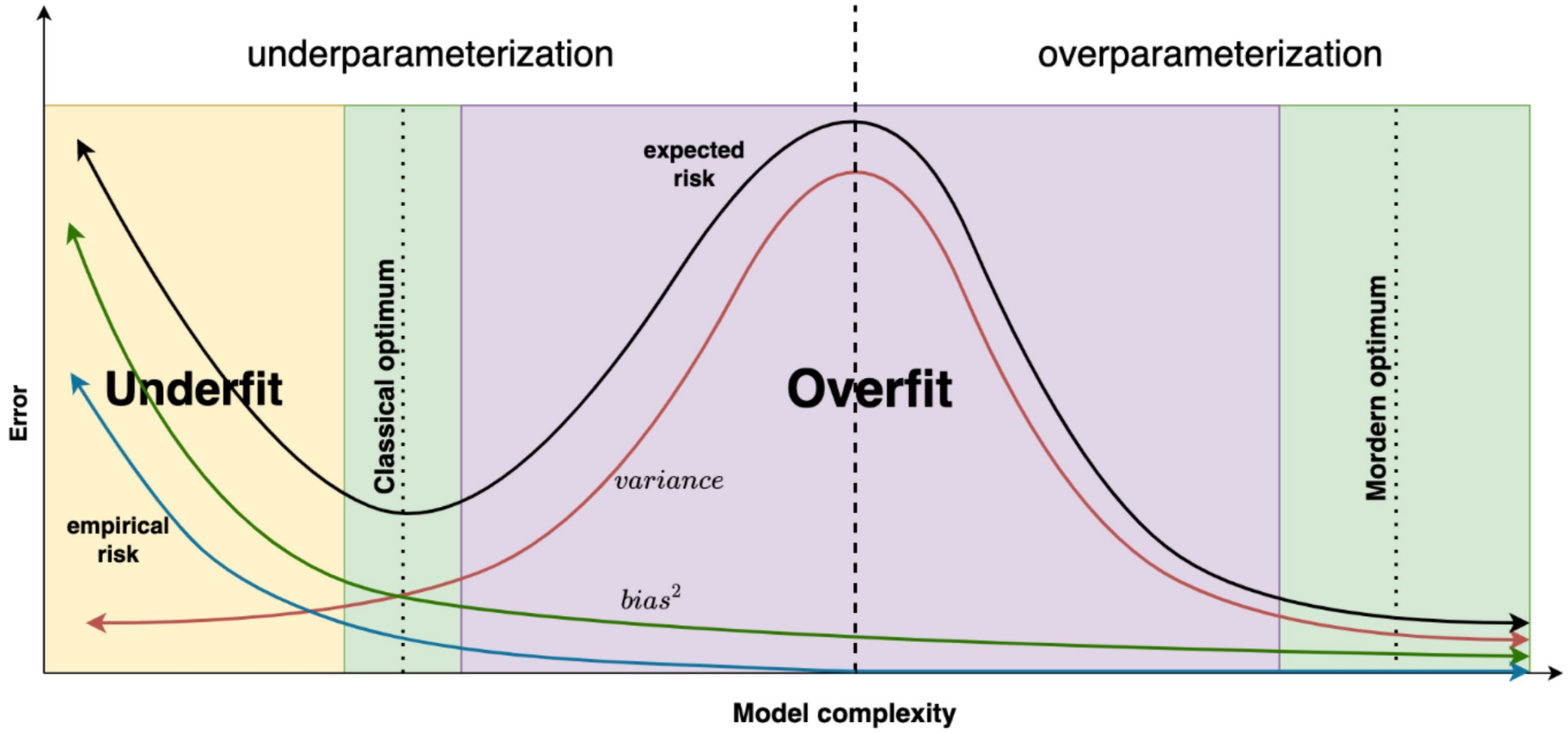


Scaling: LM Landscape with GPT-3



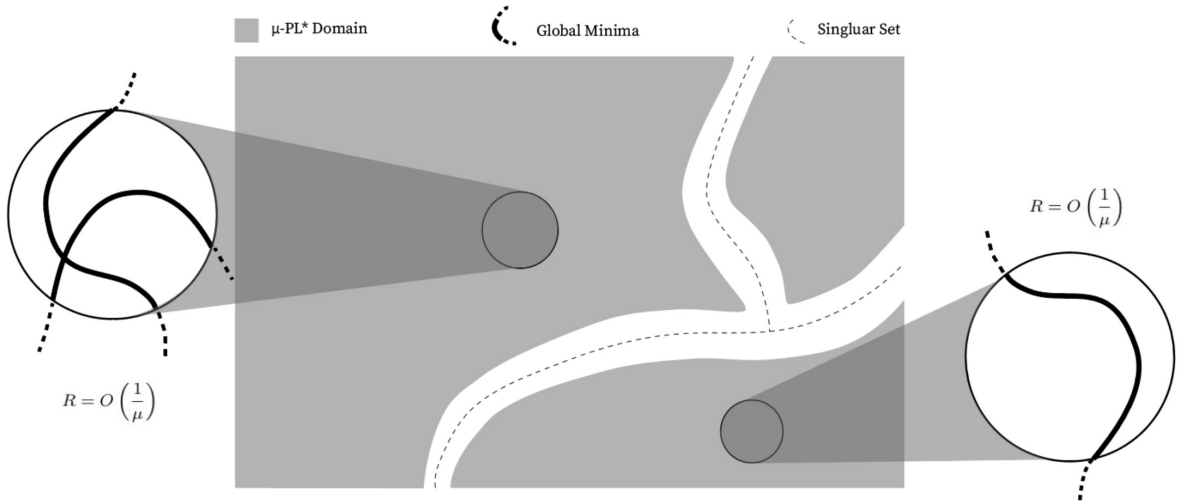
Why scaling?

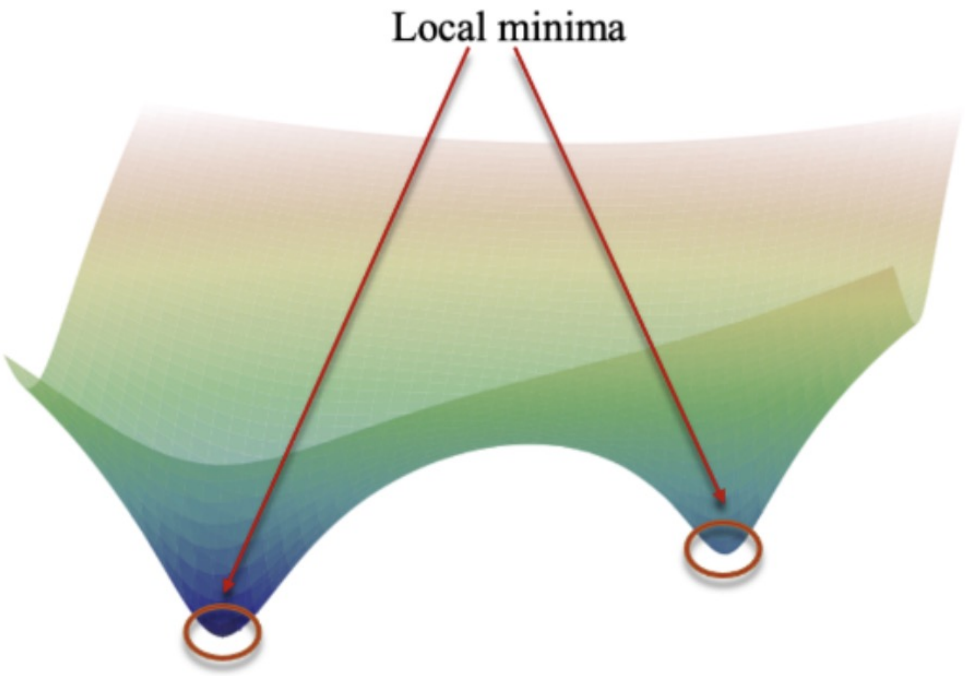
- Double Descent



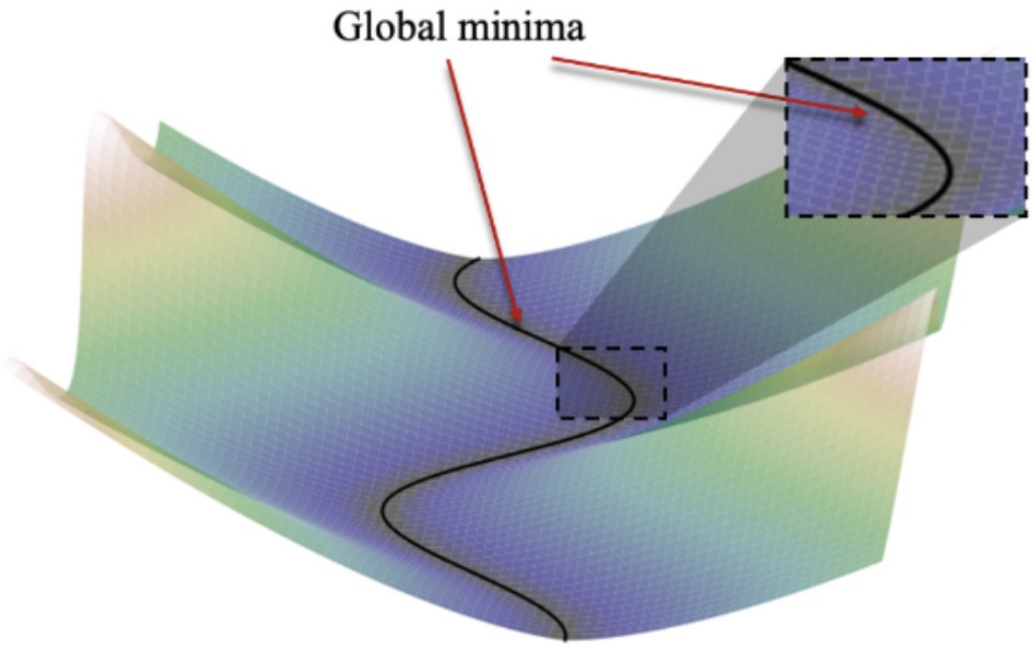
Why scaling?

- All minima are global





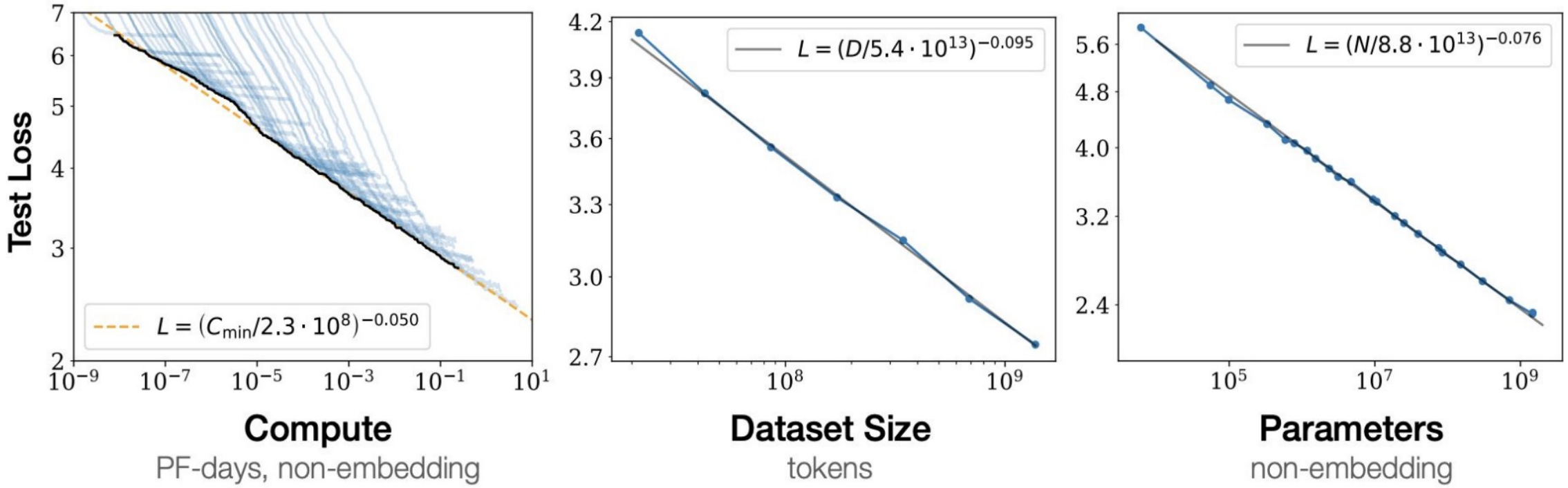
(a) Loss landscape of under-parameterized models



(b) Loss landscape of over-parameterized models

Liu et al. Loss landscapes and optimization in over-parameterized non-linear systems and neural networks, 2022

Scaling laws



Joint data-model scaling laws

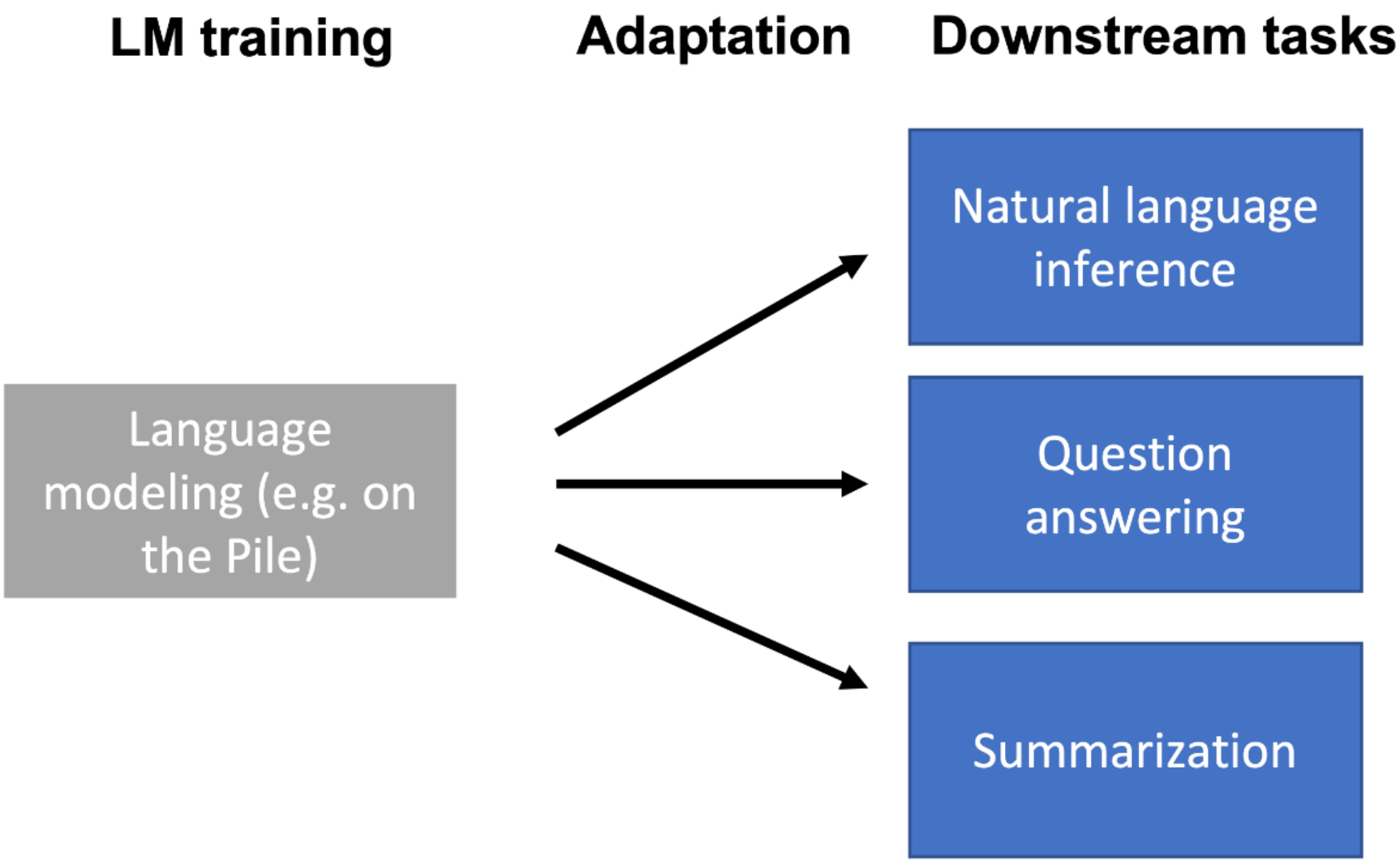
$$\hat{L}(N, D) \triangleq E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}.$$

Approach	Coeff. a where $N_{opt} \propto C^a$	Coeff. b where $D_{opt} \propto C^b$
1. Minimum over training curves	0.50 (0.488, 0.502)	0.50 (0.501, 0.512)
2. IsoFLOP profiles	0.49 (0.462, 0.534)	0.51 (0.483, 0.529)
3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan *et al.* (2020) [23]	0.73	0.27

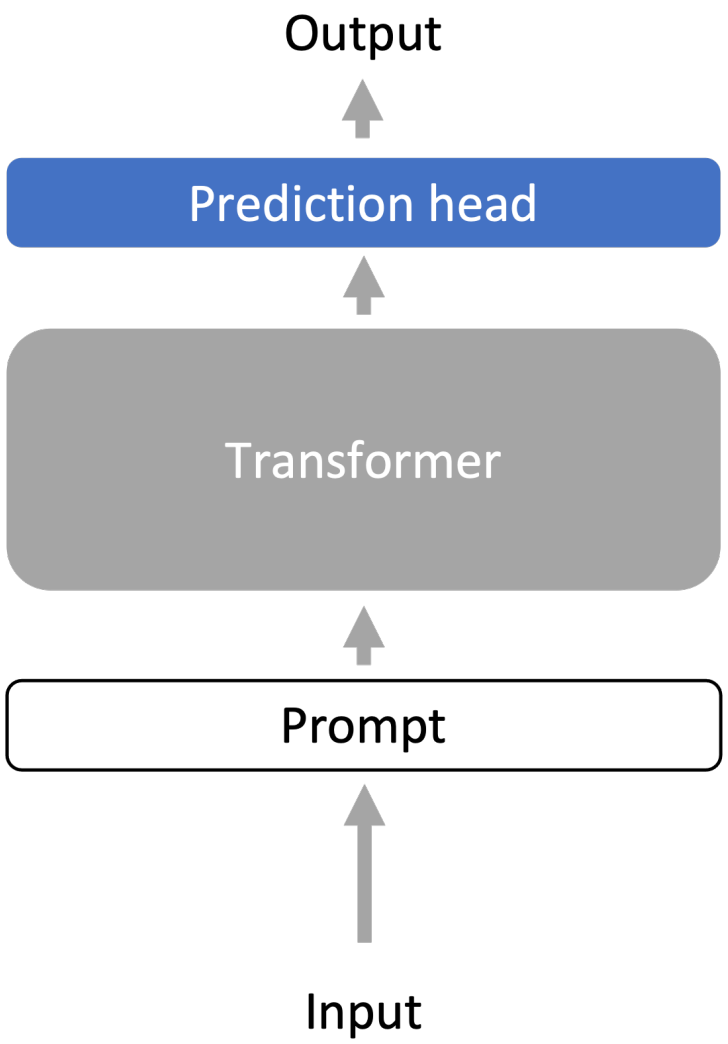
Hoffmann et al. Training compute-optimal large language models, 2022

Adaptation

Adaptation


LM training
Adaptation
Downstream tasks
Language modeling (e.g. on the Pile)
Natural language inference
Question answering
Summarization

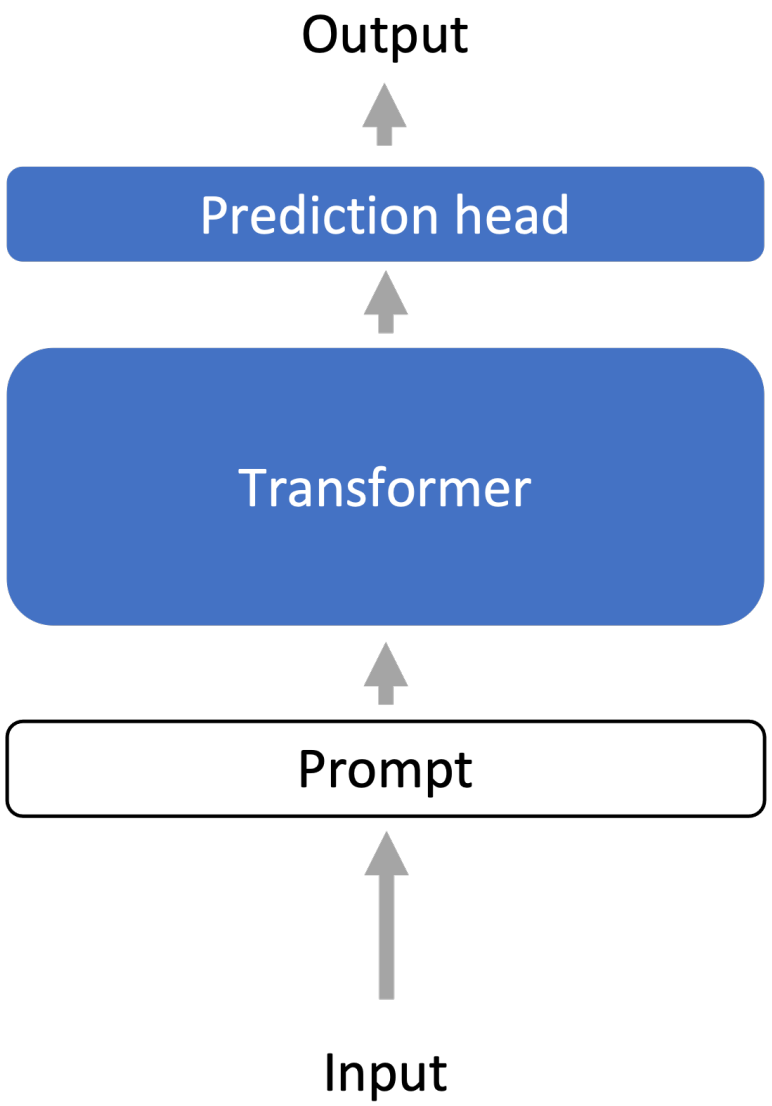
Adaptation: Probing



- **Freeze** (gray): language model representation encoder
- **Optimize** (blue, changes per task): probe (prediction head)
- **Models**: linear or shallow feedforward prediction head

https://stanford-cs324.github.io/winter2022/lectures/adaptation/

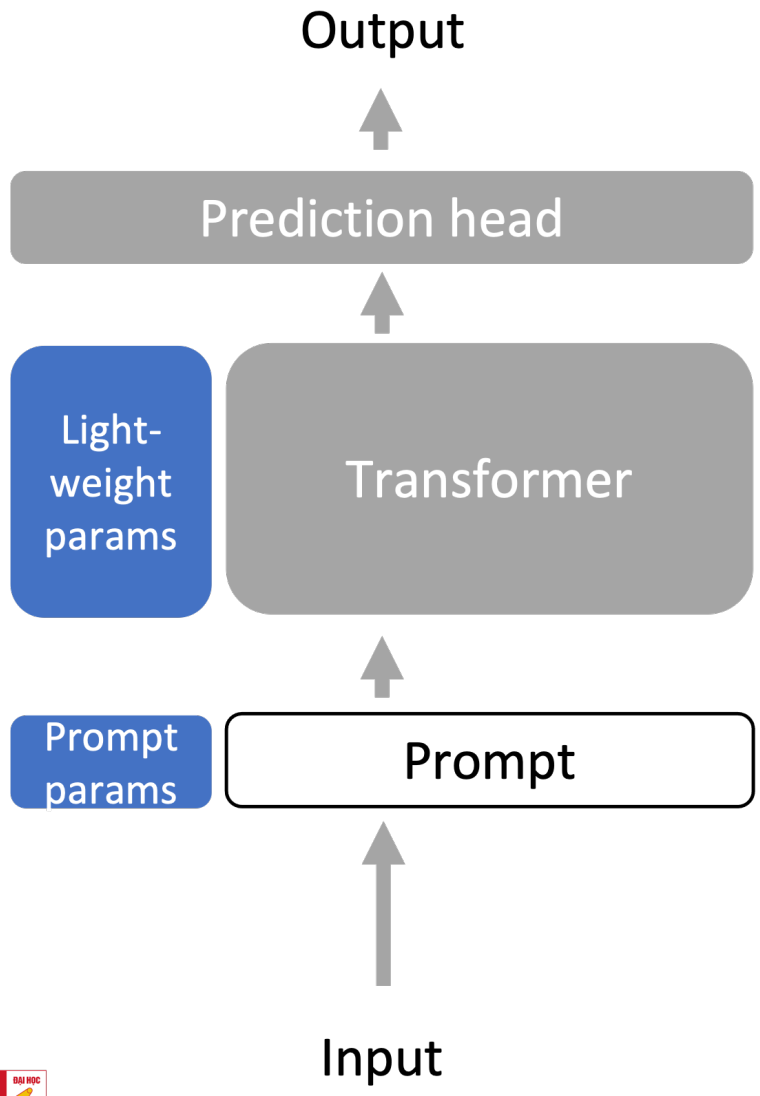
Adaptation: Fine-tuning



- **Freeze** (gray): nothing
- **Optimize** (blue, changes per task): all parameters of the language model, plus a new prediction head

https://stanford-cs324.github.io/winter2022/lectures/adaptation/

Adaptation: Lightweight Fine-tuning



- **Freeze** (gray): whole/most of language model
- **Optimize** (blue, changes per task): small number of additional parameters (<1% of the parameters)
- **Methods**: prompt tuning, prefix tuning, adapter tuning, and others (LoRA, BitFit, …)

https://stanford-cs324.github.io/winter2022/lectures/adaptation/

HUST

THANK YOU !

hust.edu.vn fb.com/dhbkhn