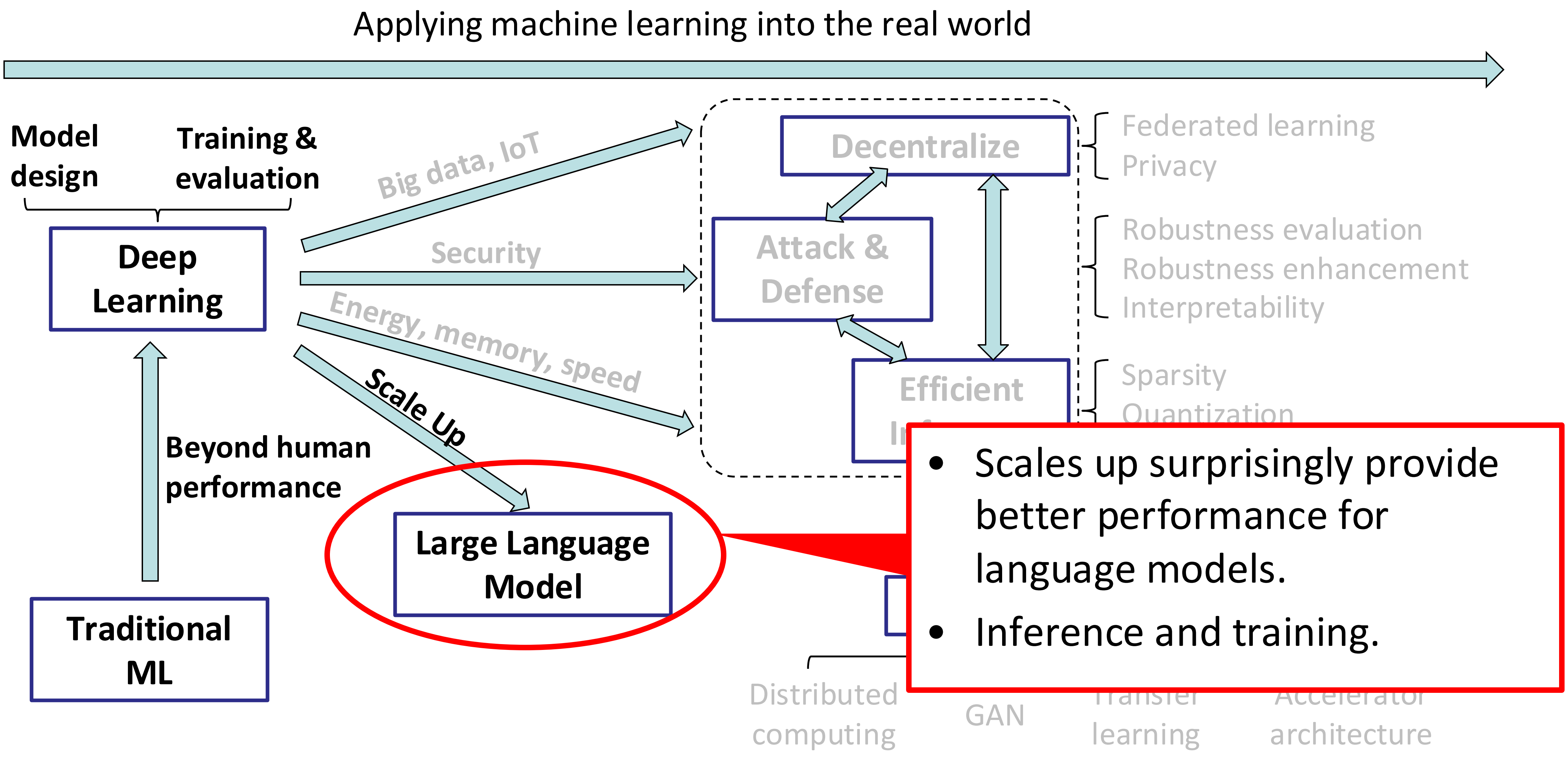

ECE 661 COMP ENG ML & DEEP NEURAL NETS

# 10. ATTENTION MODELS

HAI "HELEN" LI, SPRING 2025

# This lecture




- Scales up surprisingly provide better performance for language models.
- Inference and training.

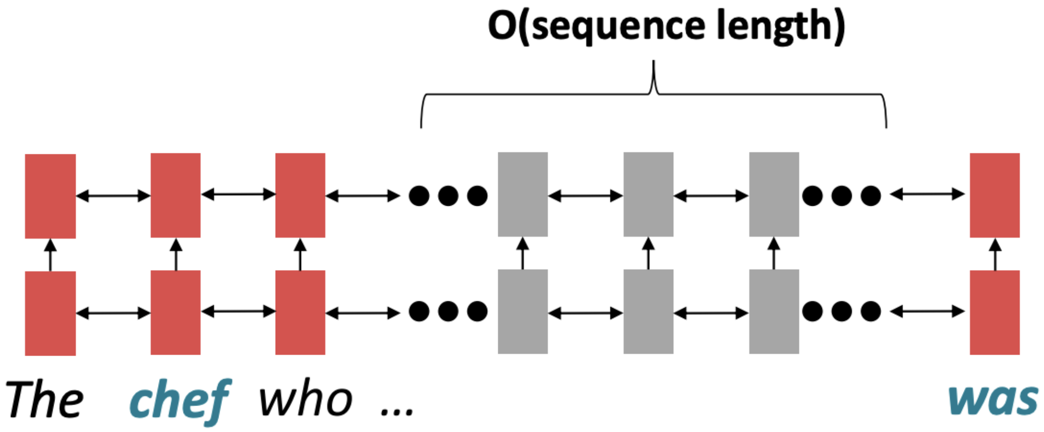
# Outline

**Lecture 10: Attention-based models and LLM Inference**

- **Self-attention**
- Transformer
- Vision Transformer
- Large Language Models
  - Architecture changes
  - Scaling Up

# Issue with recurrent models

- Recurrent models (e.g., LSTM, GRU) are unrolled from left to right
  - Word pairs will have linear interaction distance

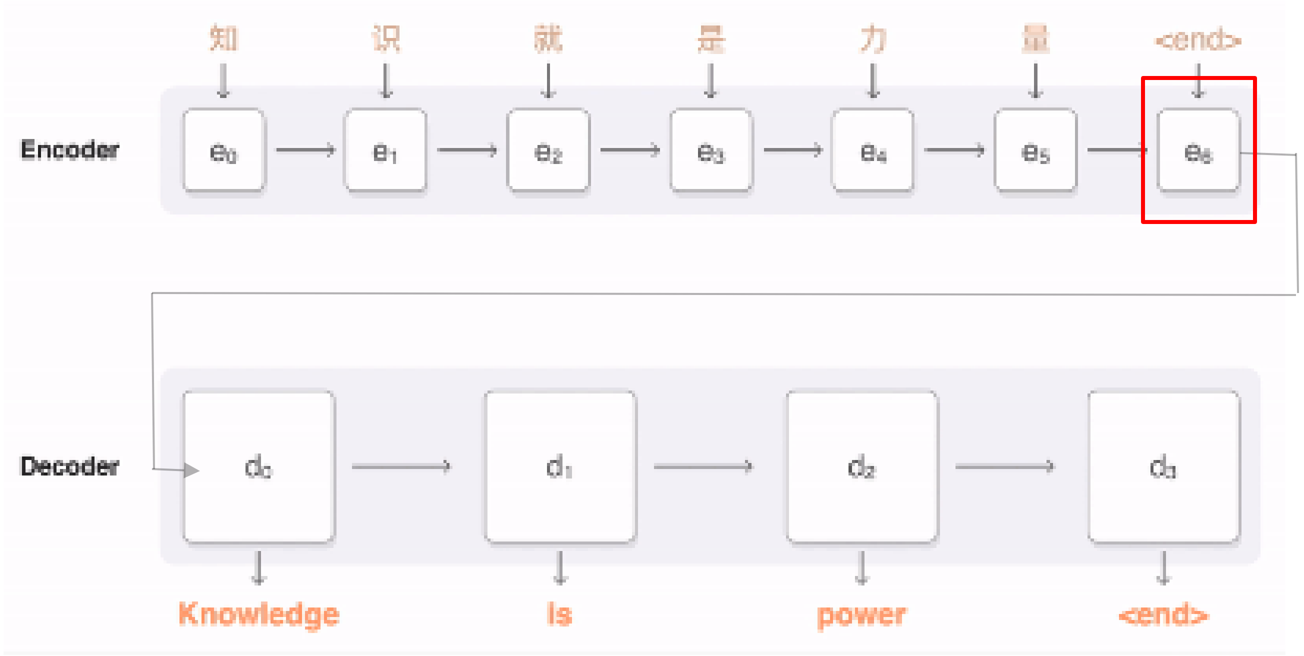



**Problems:**

- Hard to learn long-distance dependencies
  - Gradient vanishing issue
- Hard to parallelize
  - Forward and backward passes have *O(sequence length)* unparallelizable operations

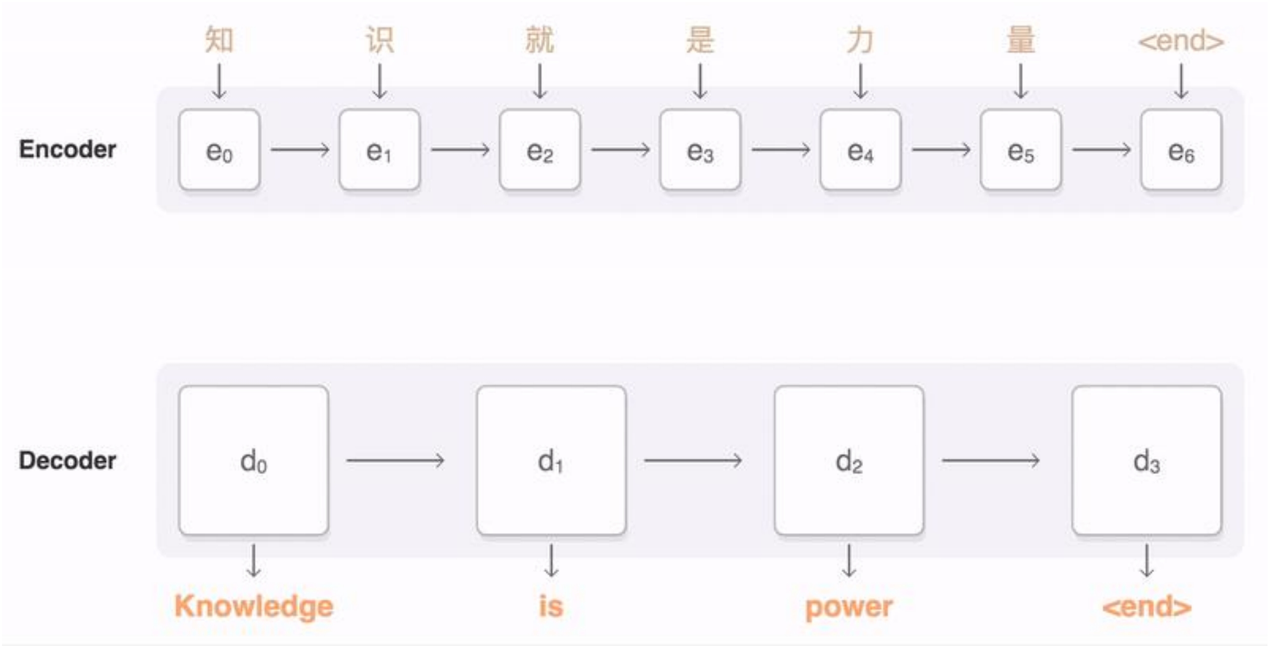
# Problems with classic Seq2Seq models

- Traditional encoder-decoder systems suffer from information bottleneck:
  - **Last hidden state** need to capture **all the information** about the source sentence



# Solution: attention mechanism

- Attention mechanism provides a solution to the problem
- Core idea: at each decoding step, **focus on different part** of the source sequence.




“A neural network for machine translation, at production scale” Google AI Blog

# How to compute attention?

- Suppose we have encoder hidden states $e_1, \ldots e_N \in \mathbb{R}^h$ ,
  step *t* decoder hidden state $d_t \in \mathbb{R}^h$
- At decoding step *t,*

1. Compute the attention score

$$s^t = [d_t^T e_1, \ldots d_t^T e_N] \in \mathbb{R}^N$$

2. Apply softmax to get the attention distribution over source tokens

$$w^t = softmax(s^t) \in \mathbb{R}^N$$

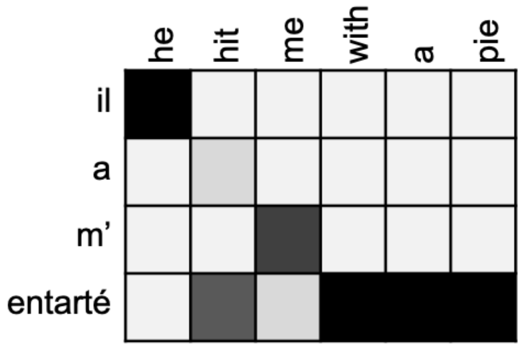
3. Compute weighted sum over the encoder hidden states

$$a_t = \sum_{i=1}^{N} w_i^t e_i \in \mathbb{R}^h$$

4. Concatenate $a_t$ with $d_t$ ,and feed $[a_t; d_t] \in \mathbb{R}^{2h}$to the decoder

# Why attention is so powerful?

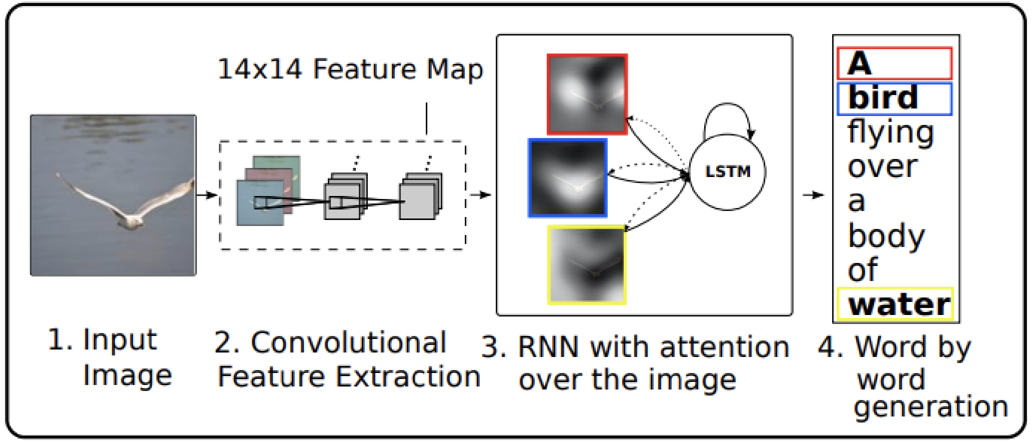
- Attention can significantly improve neural machine translation (NMT) performance
  - Allow decoder to focus on different parts of the source
  - Solves the information bottleneck problem
- Attention helps with the vanishing gradient issue
  - Provides shortcut to early source tokens
- Attention provides interpretability
  - Implicitly learn soft alignment between source and target sequence
  - Check the attention distribution for each output token

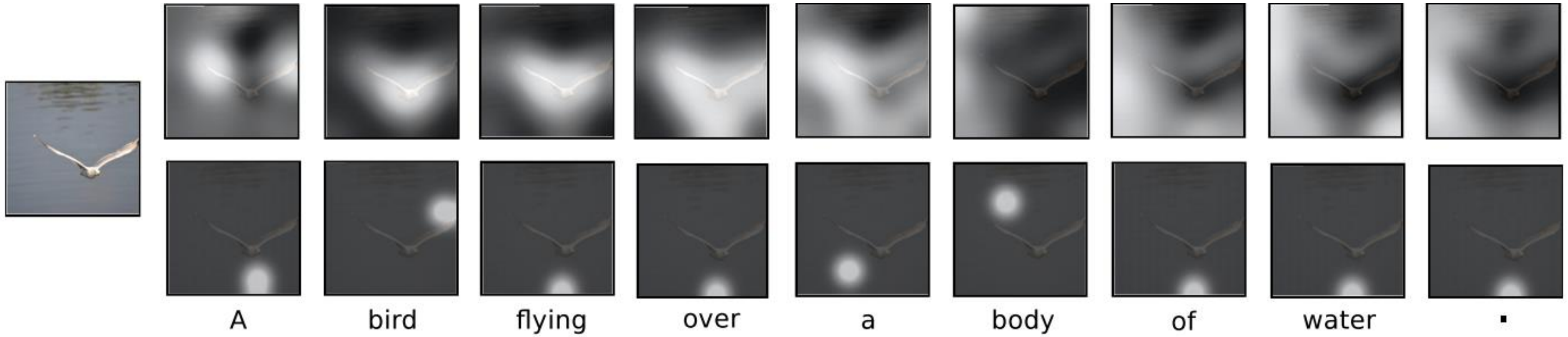



D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate." (2014)

# Attention as a general technique

- Attention is also used in computer vision:
  - Attend to different parts on input image when generating caption



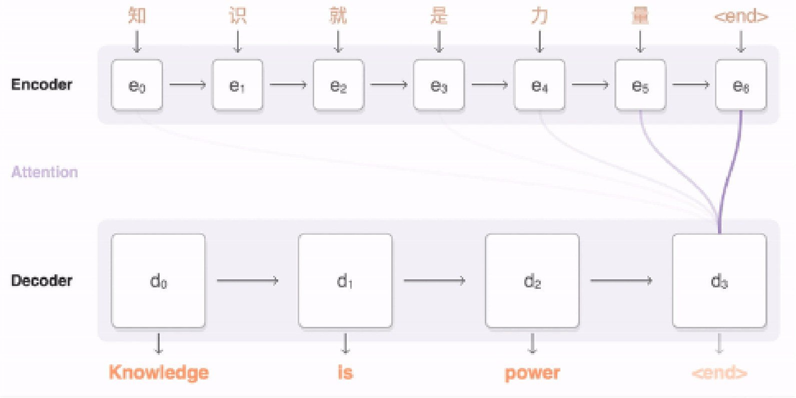




- Attention can also be a basic building block for sequence modeling
  - New sequence models: Transformers, BERT, GPT etc.

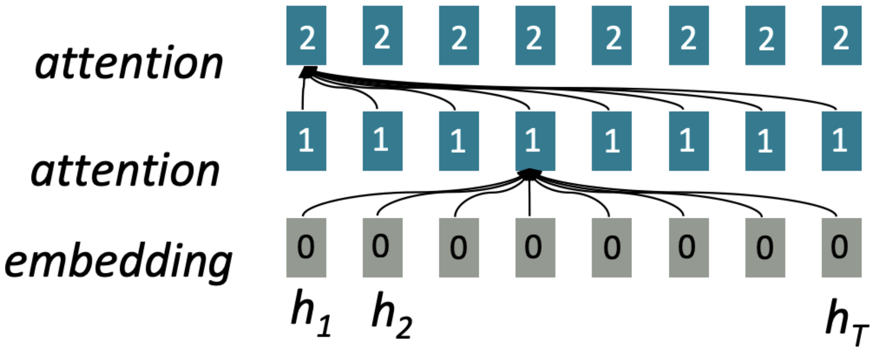
Xu, Kelvin, et al."Show, attend and tell: neural image caption generation with visual attention" ICML 2015

# Replace recurrent with self-attention

- Remember attention is introduced in Seq2Seq systems to attend different parts of source sentence




- Self-attention: apply attention within a single sentence
  - All words attend to all words in previous layer (most arrows are omitted)

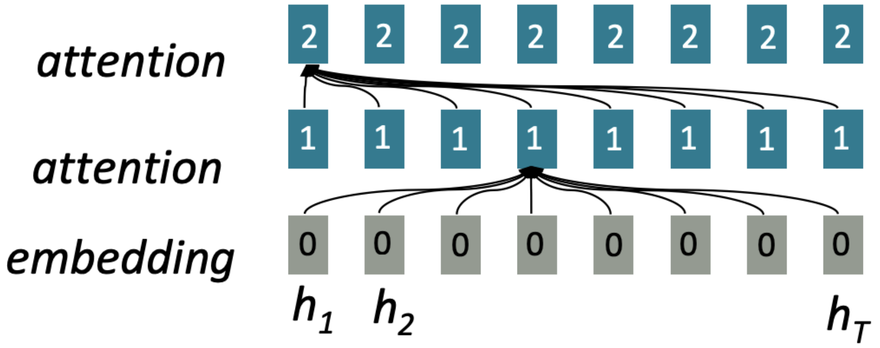



Credit: Stanford cs224n

# Self-attention computation

- To compute attention we need queries, keys, and values:
  - Queries: $q_1, q_2, \dots q_T$. Each $q_i \in \mathbb{R}^d$
  - Keys: $k_1, k_2, \dots k_T$. Each $k_i \in \mathbb{R}^d$
  - Values: $v_1, v_2, \dots v_T$. Each $v_i \in \mathbb{R}^d$
- In self-attention, the queries, keys and values come from the same source
  - $k_i = Kx_i,\ q_i = Qx_i,\ v_i = Vx_i$

    where $K, Q, V \in \mathbb{R}^{d\times d}$ are linear transformation used for all $x_i$
- Self-attention generate new representations as follows:
  - score: $s_{ij} = q_i^T k_j$, $\mathit{attention}: a_{ij} = \frac{\exp(s_{ij})}{\sum_{j'} \exp(s_{ij'})}$, $\mathit{output}_i = \sum_j a_{ij} v_j$



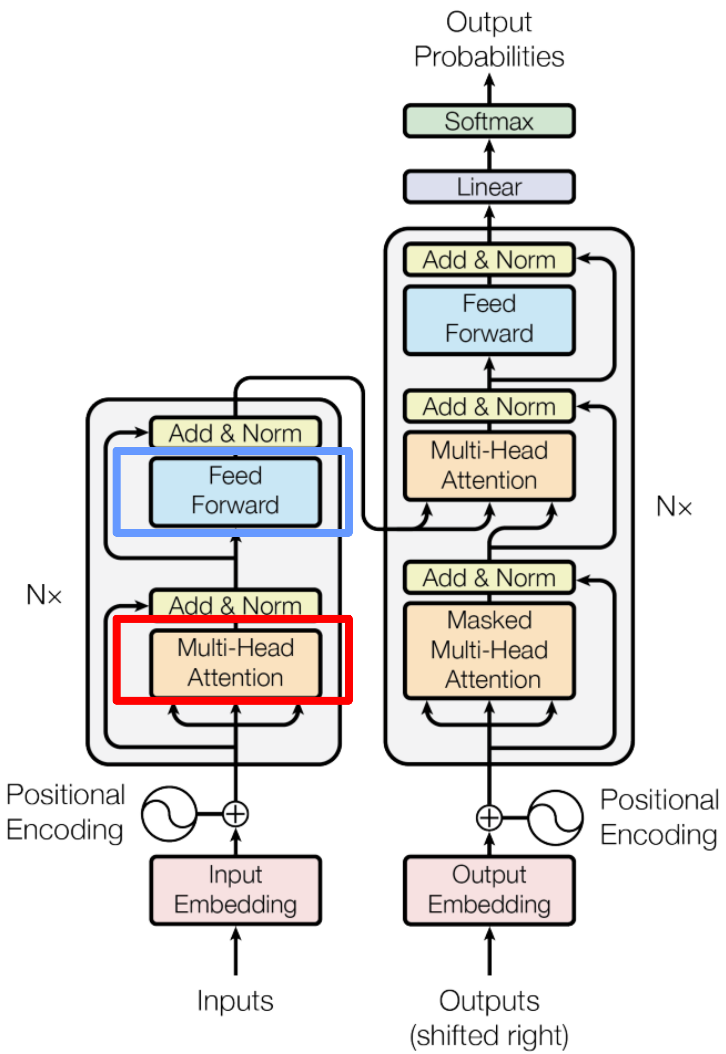
# Outline

**Lecture 10: Attention-based models and LLM Inference**

- Self-attention
- **Transformer**
- Vision Transformer
- Large Language Models
  - Architecture changes
  - Scaling Up

# Transformer

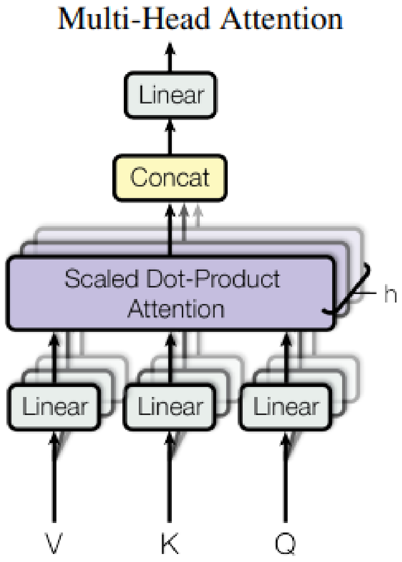
- Transformer structure:
  - Two parts: encoder & decoder (Seq2Seq model)
  - Basic block: self-attention + feed-forward
  - Stacked multiple blocks
  - Bunch of fixes/tricks




Vaswani, Ashish, et al. "Attention is all you need" Neurips 2017

# Multi-head self-attention

- Previously for each word $i$, we compute (**one**) attention over the
  - $k_i = Kx_i,\ q_i = Qx_i,\ v_i = Vx_i$ where $K, Q, V \in \mathbb{R}^{d\times d}$
  - score: $s_{ij} = q_i^T k_j$, attention: $a_{ij} = \frac{\exp(s_{ij})}{\sum_{j'} \exp(s_{ij'})}$, output$_i = \sum_j a_{ij} v_j$
- What if we want multiple attentions for each word?
  - We can define multiple attention "heads" by multiple $K, Q, V$ matrices
  - Each head will look at different things and combine values differently!

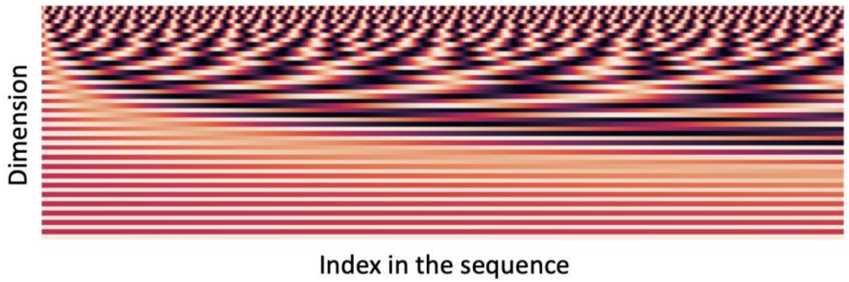



- Define $K^l, Q^l, V^l \in \mathbb{R}^{d\times\frac{d}{h}}$, where $h$ is the number of attention heads
  - For each head $l$: $k_i^l = K^l x_i,\ q_i^l = Q^l x_i,\ v_i^l = V^l x_i$
  - Use $k_i^l, q_i^l, v_i^l \in \mathbb{R}^{\frac{d}{h}}$ to compute score, attention and output$_i^l \in \mathbb{R}^{\frac{d}{h}}$
  - Combine all attention head outputs: output$_i = W_o[\text{output}_i^1; \ldots; \text{output}_i^h]$ where $W_o \in \mathbb{R}^{d\times d}$

# Encode sequence order

- Self-attention operation doesn't consider **the order information**
- Simple fix: we can represent the **sequence index** as a **vector**
  - Define positional embedding $p_i \in \mathbb{R}^d$, for $i \in \{1, 2, \ \dots \ , \ T\}$
- Suppose $e_i \in \mathbb{R}^d$, for $i \in \{1,2, \dots, T\}$ are the word embeddings, then we can add the positional embedding at layer 0: $x_i^0 = e_i + p_i$
- Options:
  - Sinusoidal position embedding:

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




  - Learned position embedding:

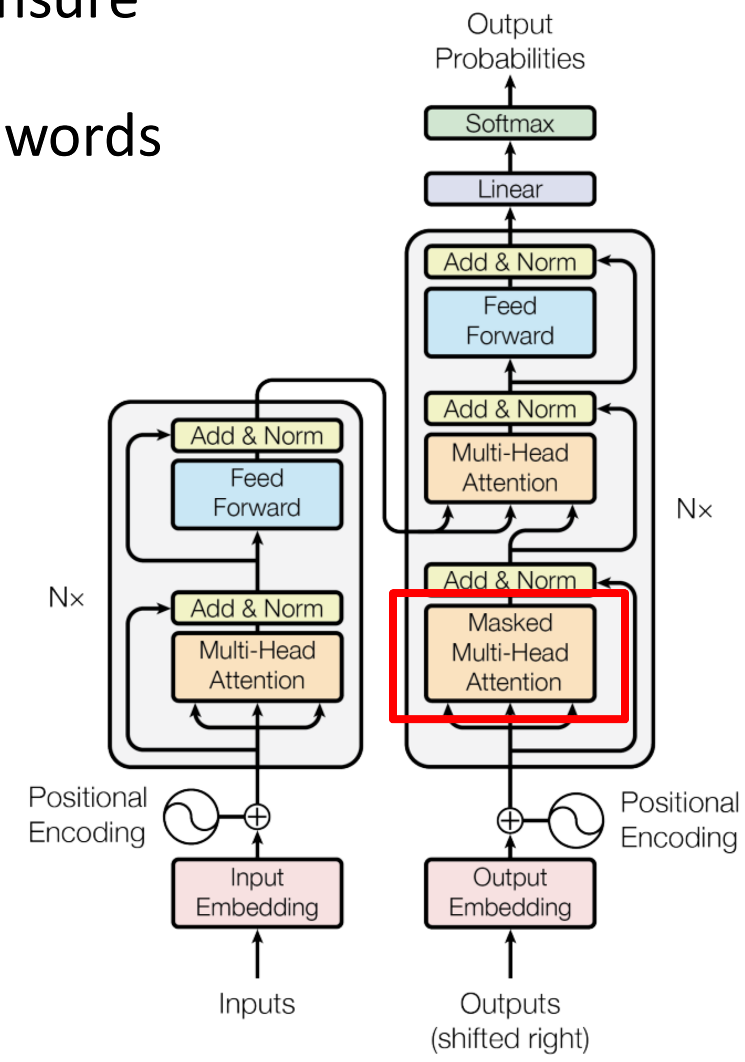
    Just make all $p_i$ as learnable parameters

# Transformer decoder: self-attention

- To use self-attention in decoders, we need to ensure the decoder **cannot peek the future**
- Simple fix: we can mask the attention to future words by setting attention score as $-\infty$:

$$s_{ij} = \begin{cases} q_i^T k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$

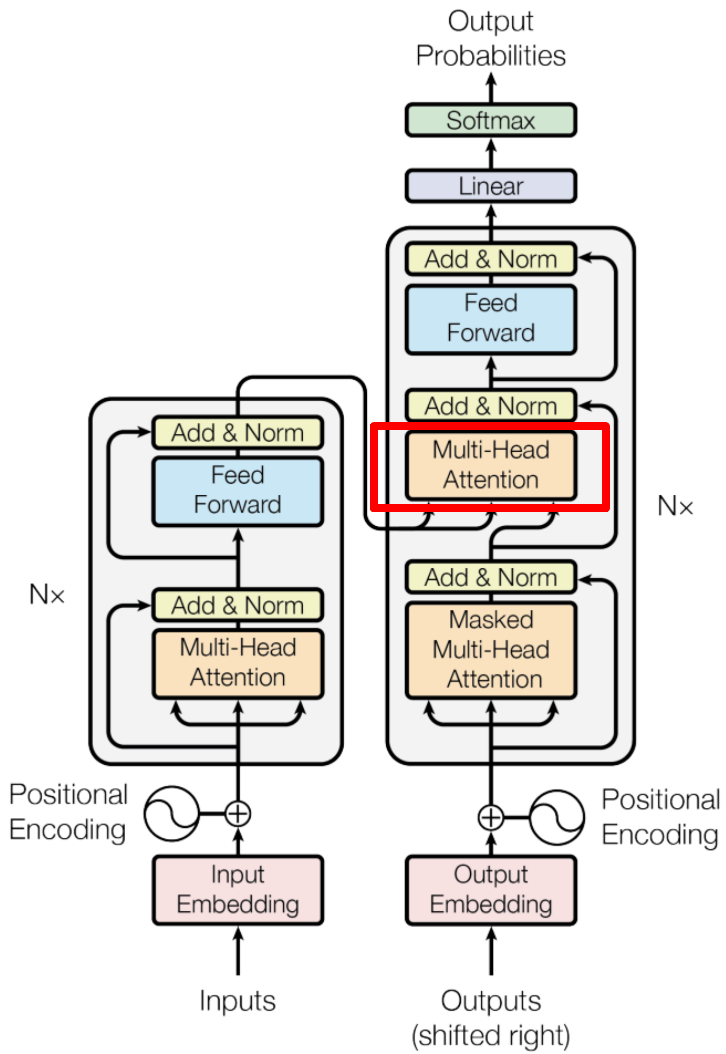
| | [start] | The | boy | want |
|---|---|---|---|---|
| [start] | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| The | | $-\infty$ | $-\infty$ | $-\infty$ |
| boy | | | $-\infty$ | $-\infty$ |
| want | | | | $-\infty$ |



# Transformer decoder: encoder-attention

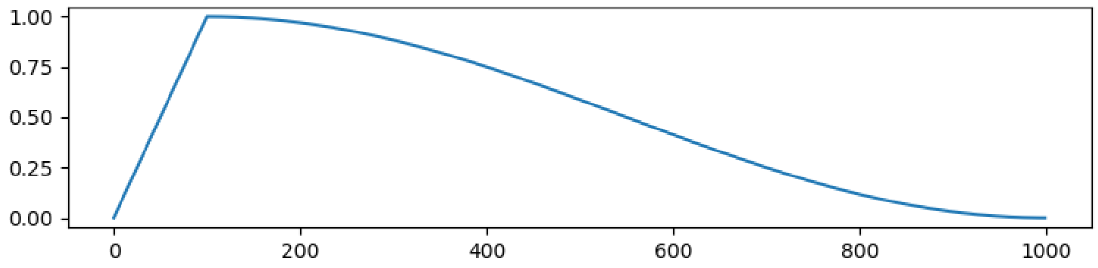
- In self-attention, keys, queries and values come from the same source
- However, on the decoder side, besides self-attention we also want to attend the states from encoder (Seq2Seq model)
- Simple fix: construct keys and values using encoder states
  - Define $x_1, \ldots x_T \in \mathbb{R}^d$ as the output vectors from the **encoder**
  - Define $h_1, \ldots h_N \in \mathbb{R}^d$ as the input vectors from the **decoder**
  - Compute key, value, query by:

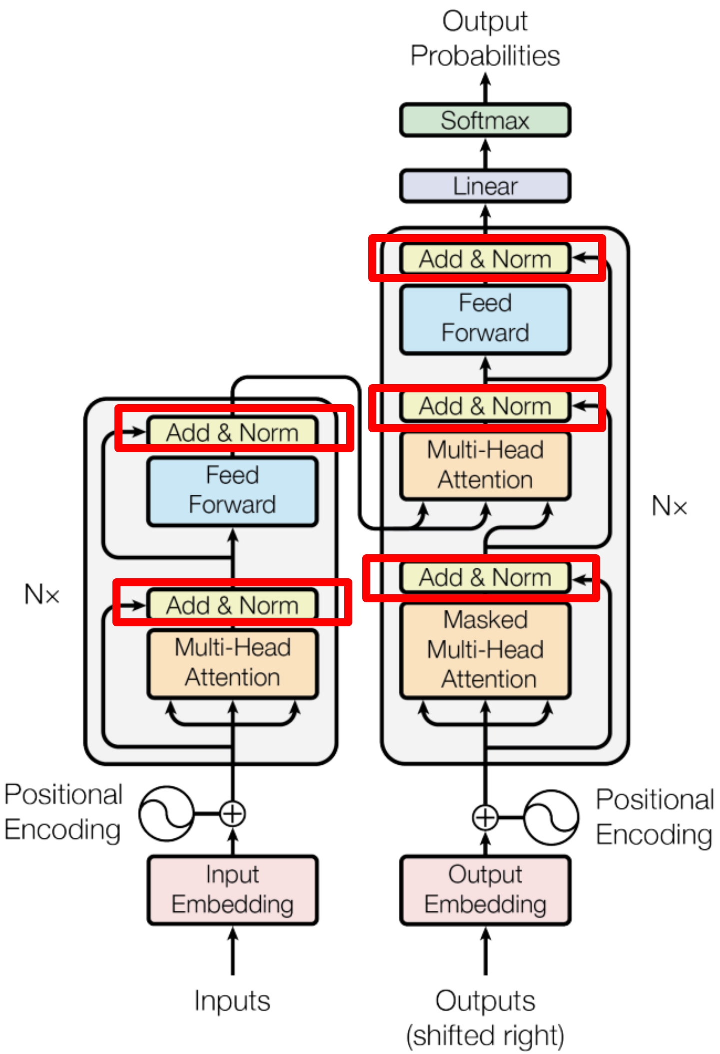
$$k_i = Kx_i, v_i = Vx_i, q_i = Qh_i$$



# Other tricks in Transformer

- Residual connection and layer normalization:
  - Add after multi-head attention and feed-forward modules
  - Help models train faster
- Learning rate schedule:
  - warm-up stage: learning rate first increase then decrease
  - Converge to better sub-optimal




Ba, Jimmy Lei, et al. "Layer normalization"

# Transformer result

| Model | BLEU | | Training Cost (FLOPs) | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | $\mathbf{3.3 \cdot 10^{18}}$ | |
| Transformer (big) | **28.4** | **41.8** | $2.3 \cdot 10^{19}$ | |

Vaswani, Ashish, et al. "Attention is all you need." (2017)

# Transformer's pros and cons

- Comparison with RNN
  - Decoder training efficiency
    - The training of RNN decoder is processed per token from left to right.
    - In the Transformer decoder training, all tokens are updated in parallel thanks to the attention mask.
  - Scalability
    - Compared to RNNs, Transformers are easier to be scaled to very deep layers (e.g., 12 or 24). Still transformers do not meet convergence plateau.
- Due to the global attention mechanism, extensive computation is required at each layer (e.g., $O(n^2)$ for a sequence of length n).
  - Sparse Transformer
    - https://arxiv.org/abs/1904.10509
  - Linformer
    - Self-Attention with Linear Complexity
    - https://arxiv.org/abs/2006.04768

# Transformer Summary

- Transformer becomes the de-facto structure in NLP domain (also getting popularized in vision area)
- Pre-trained models with transformer structure dominate the NLP benchmarks (will discuss soon)

| Rank | Name | Model | URL | Score | CoLA | SST-2 | MRPC | STS-B | QQP |
|---|---|---|---|---|---|---|---|---|---|
| 1 | T5 Team - Google | T5 | | 89.7 | 70.8 | 97.1 | 91.9/89.2 | 92.5/92.1 | 74.6/90.4 |
| 2 | ALBERT-Team Google Language | ALBERT (Ensemble) | | 89.4 | 69.1 | 97.1 | 93.4/91.2 | 92.5/92.0 | 74.2/90.5 |
| + 3 | 王玮 | ALICE v2 large ensemble (Alibaba DAMO NLP) | | 89.0 | 69.2 | 97.1 | 93.6/91.5 | 92.7/92.3 | 74.4/90.7 |
| 4 | Microsoft D365 AI & UMD | FreeLB-RoBERTa (ensemble) | | 88.8 | 68.0 | 96.8 | 93.1/90.8 | 92.4/92.2 | 74.8/90.3 |
| 5 | Facebook AI | RoBERTa | | 88.5 | 67.8 | 96.7 | 92.3/89.8 | 92.2/91.9 | 74.3/90.2 |
| 6 | XLNet Team | XLNet-Large (ensemble) | | 88.4 | 67.8 | 96.8 | 93.0/90.7 | 91.6/91.1 | 74.2/90.3 |
| + 7 | Microsoft D365 AI & MSR AI | MT-DNN-ensemble | | 87.6 | 68.4 | 96.5 | 92.7/90.3 | 91.1/90.7 | 73.7/89.9 |
| 8 | GLUE Human Baselines | GLUE Human Baselines | | 87.1 | 66.4 | 97.8 | 86.3/80.8 | 92.7/92.6 | 59.5/80.4 |
| 9 | Stanford Hazy Research | Snorkel MeTaL | | 83.2 | 63.8 | 96.2 | 91.5/88.5 | 90.1/89.7 | 73.1/89.9 |
| 10 | XLM Systems | XLM (English only) | | 83.1 | 62.9 | 95.6 | 90.7/87.1 | 88.8/88.2 | 73.2/89.8 |

https://gluebenchmark.com/leaderboard/

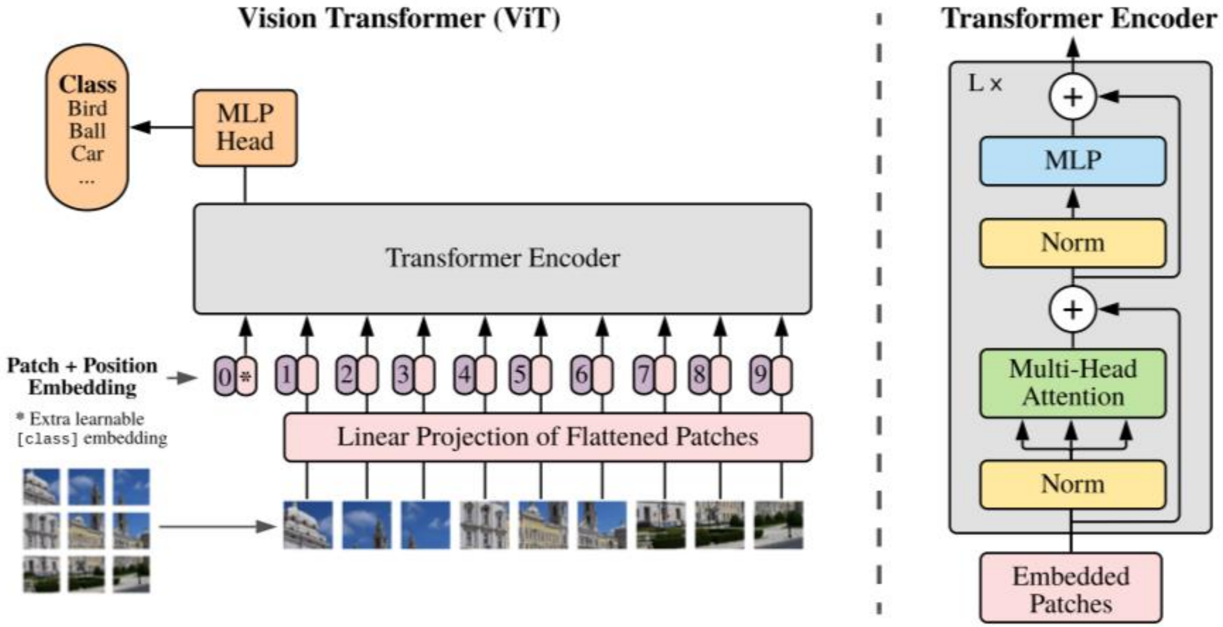
# Outline

**Lecture 10: Attention-based models and LLM Inference**

- Self-attention
- Transformer
- **Vision Transformer**
- Large Language Models
  - Architecture changes
  - Scaling Up

# Vision Transformer (ViT)

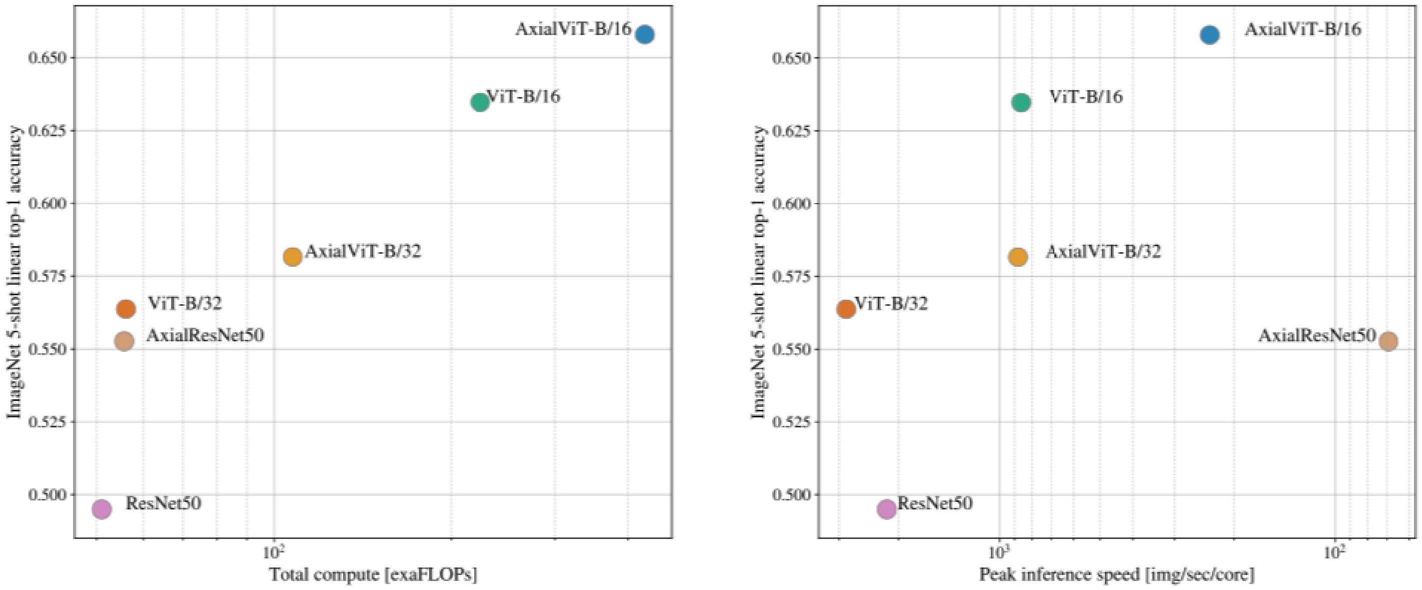
- Use Transformers for vision recognition
- We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder.
- In order to perform classification, an extra learnable “classification token” is added to the sequence.




Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale.” 2020.

# Vision Transformer (ViT)

- Performance and total computation (FLOPs) comparison.
- ViT-B/32 uses as similar computation (FLOPs) as ResNet50.
- FLOPs required by ViT-B/16 is four times of that by ViT-B/32 (smaller patch size leads to more patches).




Performance of Axial-Attention based models, in terms of top-1 accuracy on ImageNet 5-shot linear, versus their speed in terms of number of FLOPs (left) and inference time (left).

# Implement your model with Hugging Face

- HuggingFace is an AI community aiming to build the future of machine learning. Their platform holds many open-source models and datasets. Almost all models covered today can be found.
- You can easily access with Transformers, Dataset packages in Python.

```
>>> from transformers import pipeline
>>> unmasker = pipeline('fill-mask', model='bert-base-uncased')
>>> unmasker("Hello I'm a [MASK] model.")

[{'sequence': "[CLS] hello i'm a fashion model. [SEP]",
  'score': 0.1073106899857521,
  'token': 4827,
  'token_str': 'fashion'},
```

```
from transformers import GPT2Tokenizer, GPT2Model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2Model.from_pretrained('gpt2')
text = "Replace me by any text you'd like."
encoded_input = tokenizer(text, return_tensors='pt')
output = model(**encoded_input)
```

- Provide many APIs for loading/saving/**sharing** models, training, evaluating, etc.
- Integrate distributed computation libraries such as “Accelerate”, “Deepspeed”.
- Take a vital role in large language model development.
- Find more by yourself: [https://huggingface.co/](https://huggingface.co/)

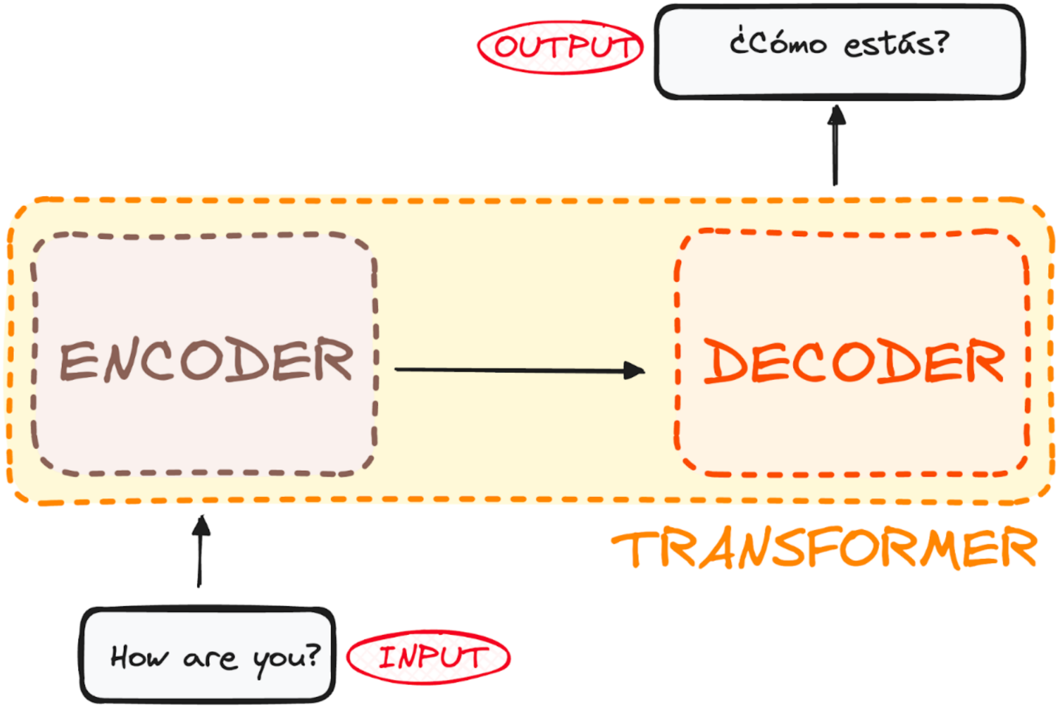
# Outline

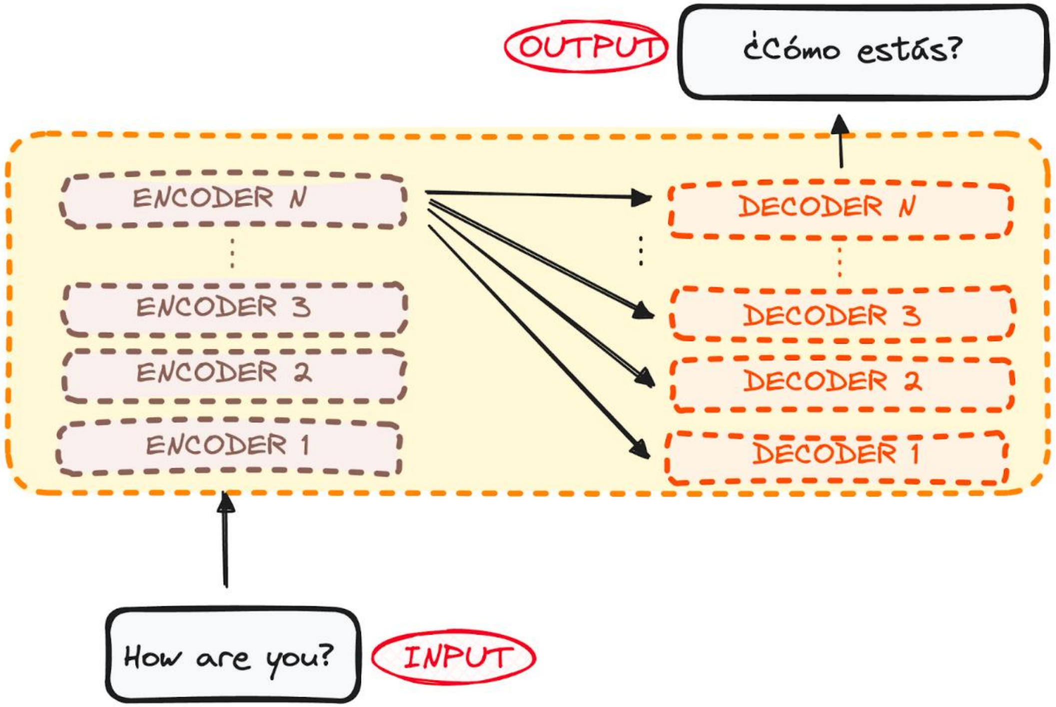
**Lecture 10: Attention-based models and LLM Inference**

- Self-attention
- Transformer
- Vision Transformer
- Large Language Models
  - **Architecture changes**
  - Scaling Up

# Encoder – Decoder Transformer Architecture

- Transformer is originally designed for language translation task
  - Encoder takes a sentence in language A
  - Decoder generates a sentence in language B

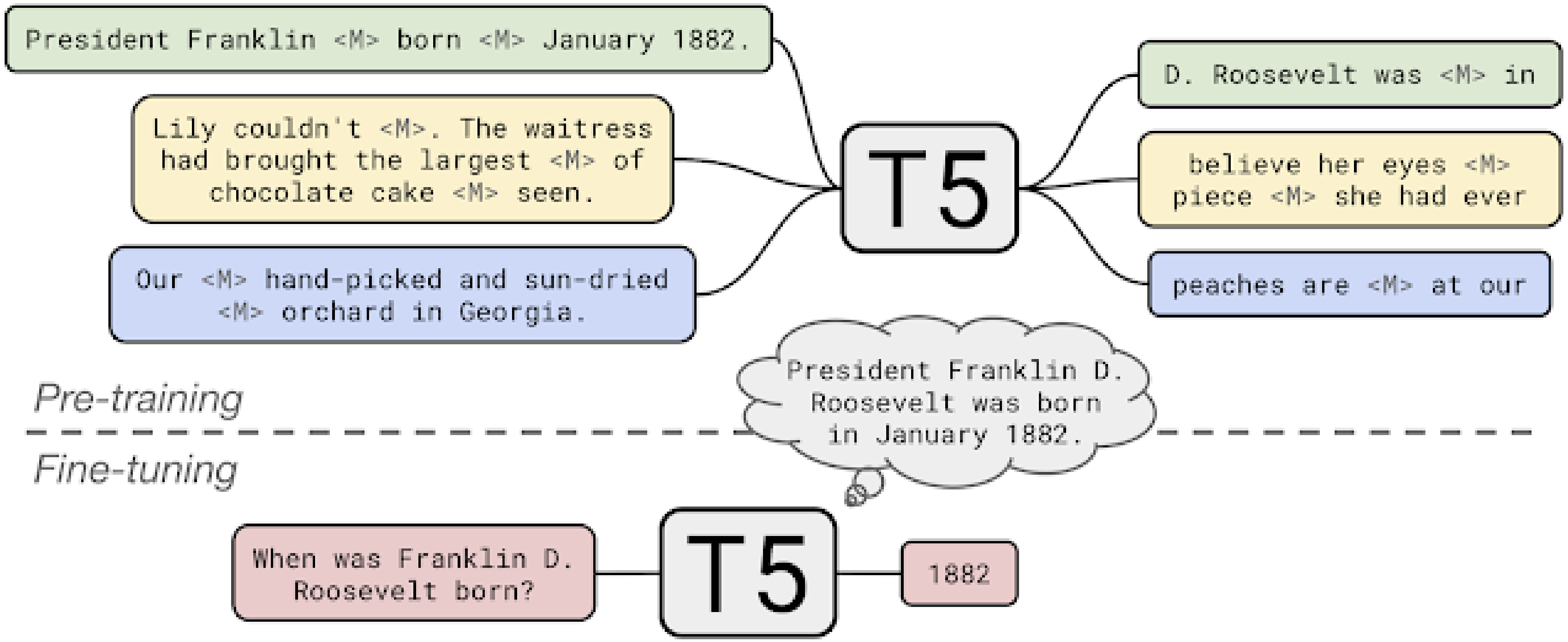



https://www.datacamp.com/tutorial/how-transformers-work

# Encoder - Decoder Transformer Model

- T5 (Text-to-Text Transfer Transformer)
  - Translate text between languages designed by Google in 2019
  - The T5 can be fine-tuned for a wide range of NLP tasks, including language translation, question answering, summarization, and more.




Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer [Raffel et al., 2019]

# Encoder Transformer Architecture

- Encoder-only Transformers are specifically designed for text classification tasks.
  - Classify a piece of text into one of several predefined categories.
  - Examples: Sentiment Analysis, Topic Classification, Spam Detection

- Encoding Process:
  - The encoder processes a sequence of tokens from the text.
  - It produces a fixed-size vector representation (embedding) of the entire sequence.
  - This vector encapsulates the meaning and context of the text.
  - The representation is then used for classification by downstream classifiers

# Encoder Transformer Model

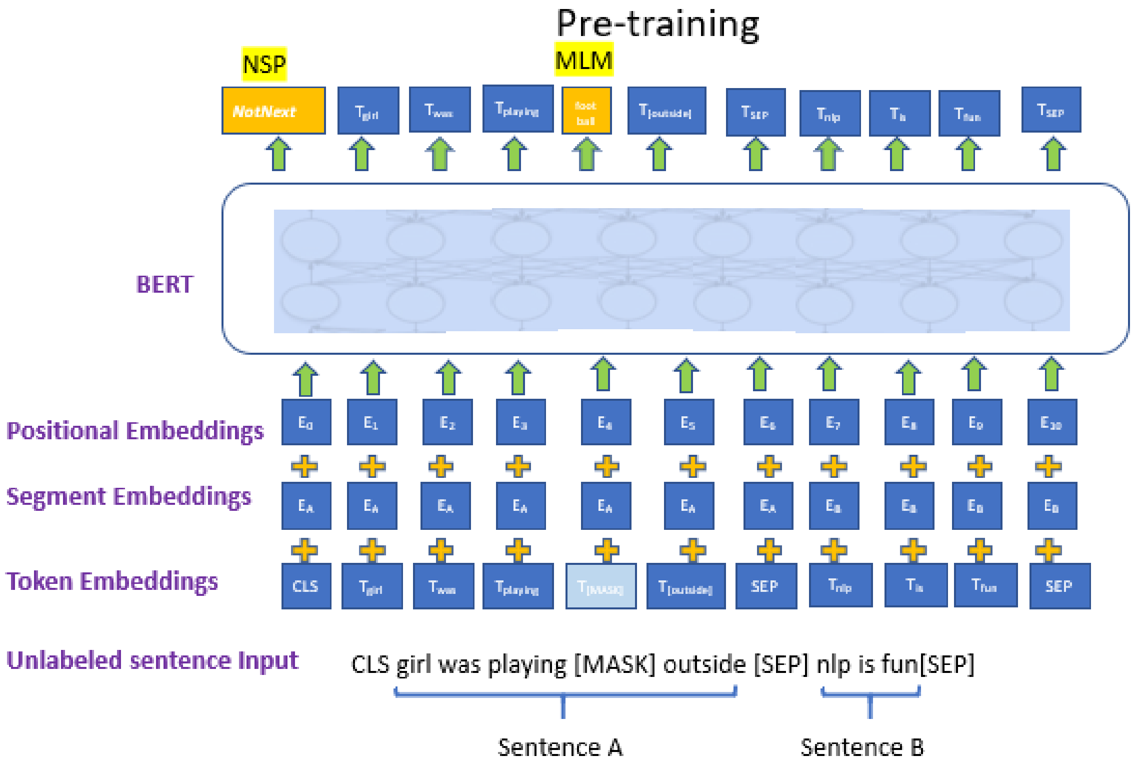
- BERT (Bidirectional Encoder Representations from Transformers)
  - bidirectionally trained language models can have a deeper sense of language context and flow than single-direction.

- Pre-training Tasks:
  - Masked LM (MLM)

    Predicts the original values of randomly masked tokens within a sequence

  - NSP (Next Sentence Predict)

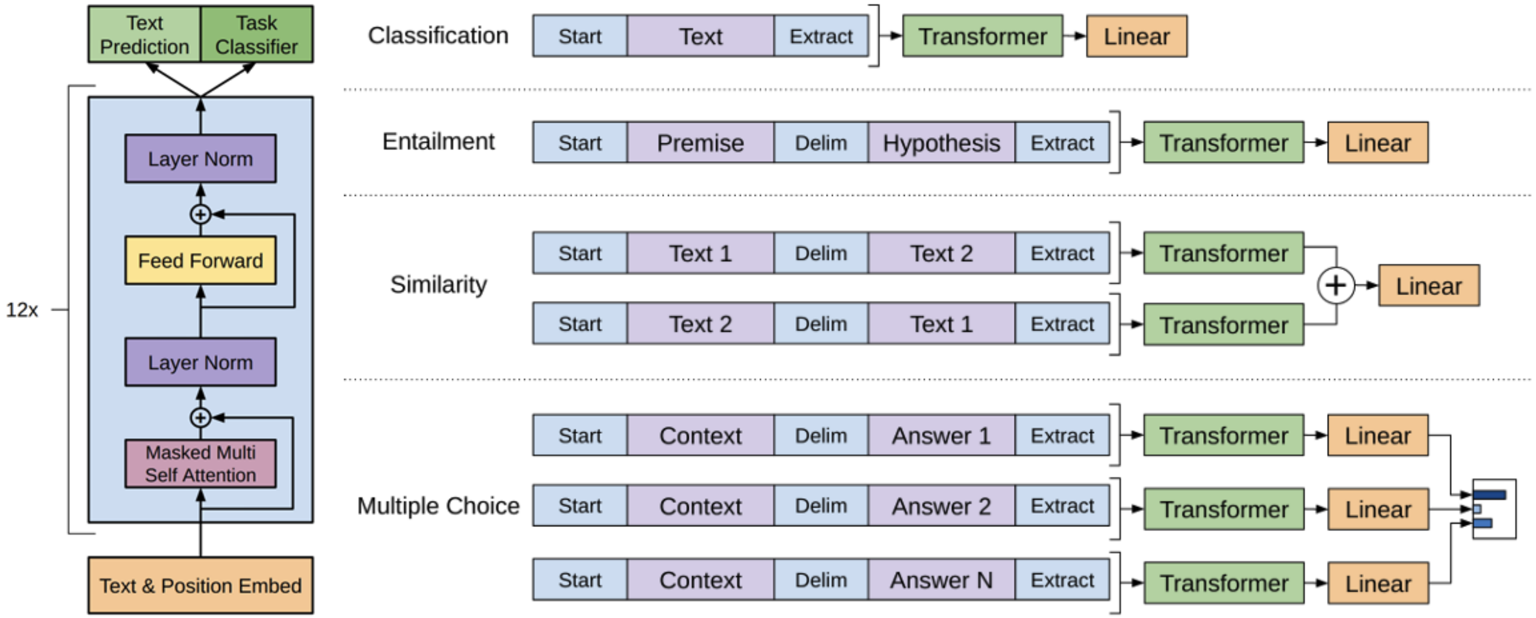
    Predicts if the second sentence in a pair is the subsequent sentence of the first one




https://www.analyticsvidhya.com/blog/2021/12/manual-for-the-first-time-users-google-bert-for-text-classification/

# Decoder Transformer Architecture

- Decoder-only Transformers are specifically designed for text generation tasks.
  - Takes a fixed-size vector representation of the context.
  - Generates a sequence of words one at a time.
  - Each word is conditioned on all previously generated words.
- Pre-trained model can be fine-tuned to downstream tasks

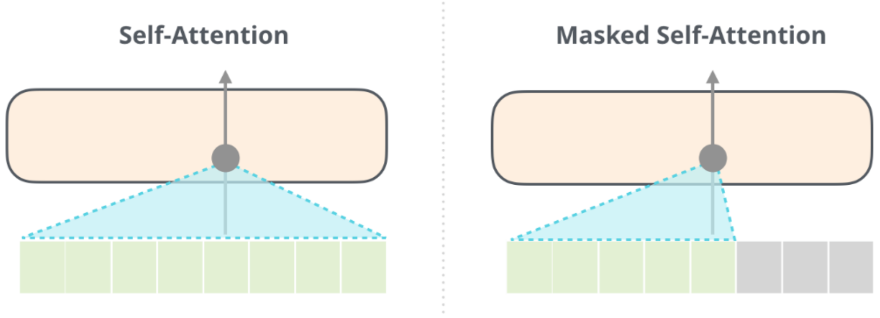


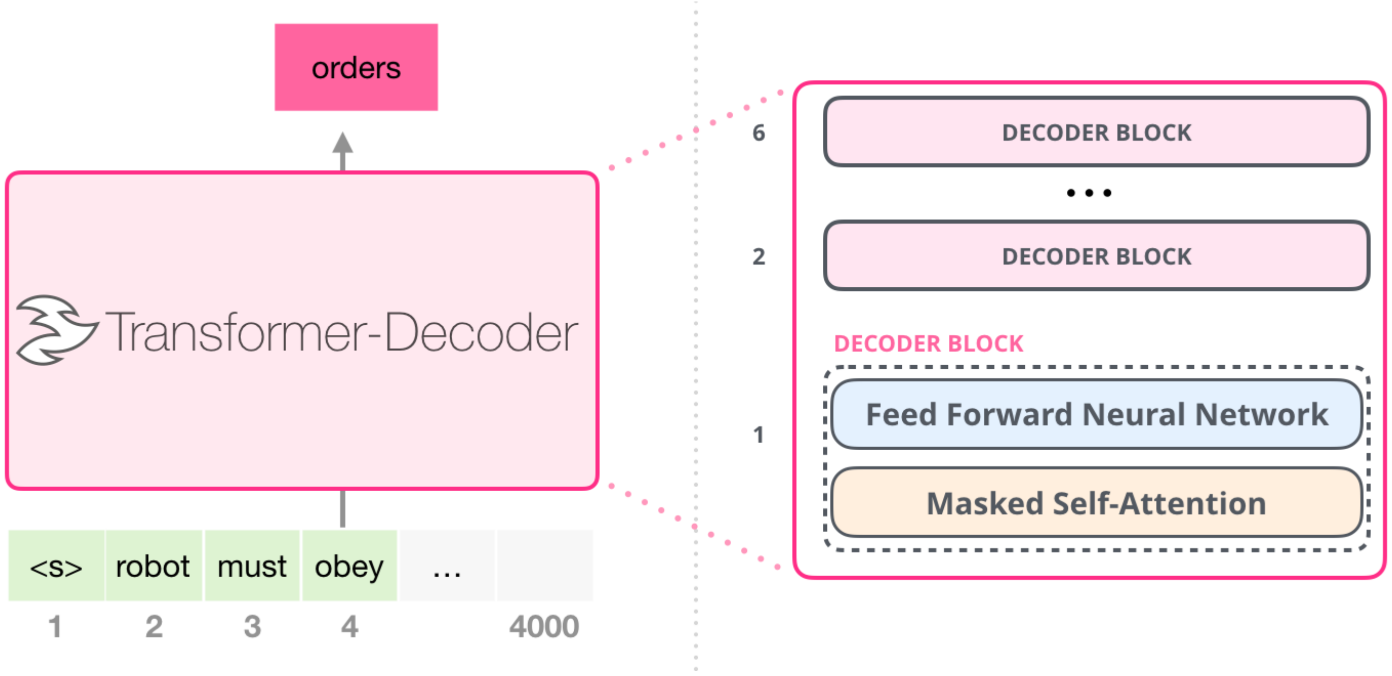

Improving Language Understanding by Generative Pre-Training [Radford et al., 2018]

# Decoder Transformer Model

- GPT (Generative Pre-trained Transformer)
  - Masked Attention

    blocking information from tokens that are to the right of the position being calculated.



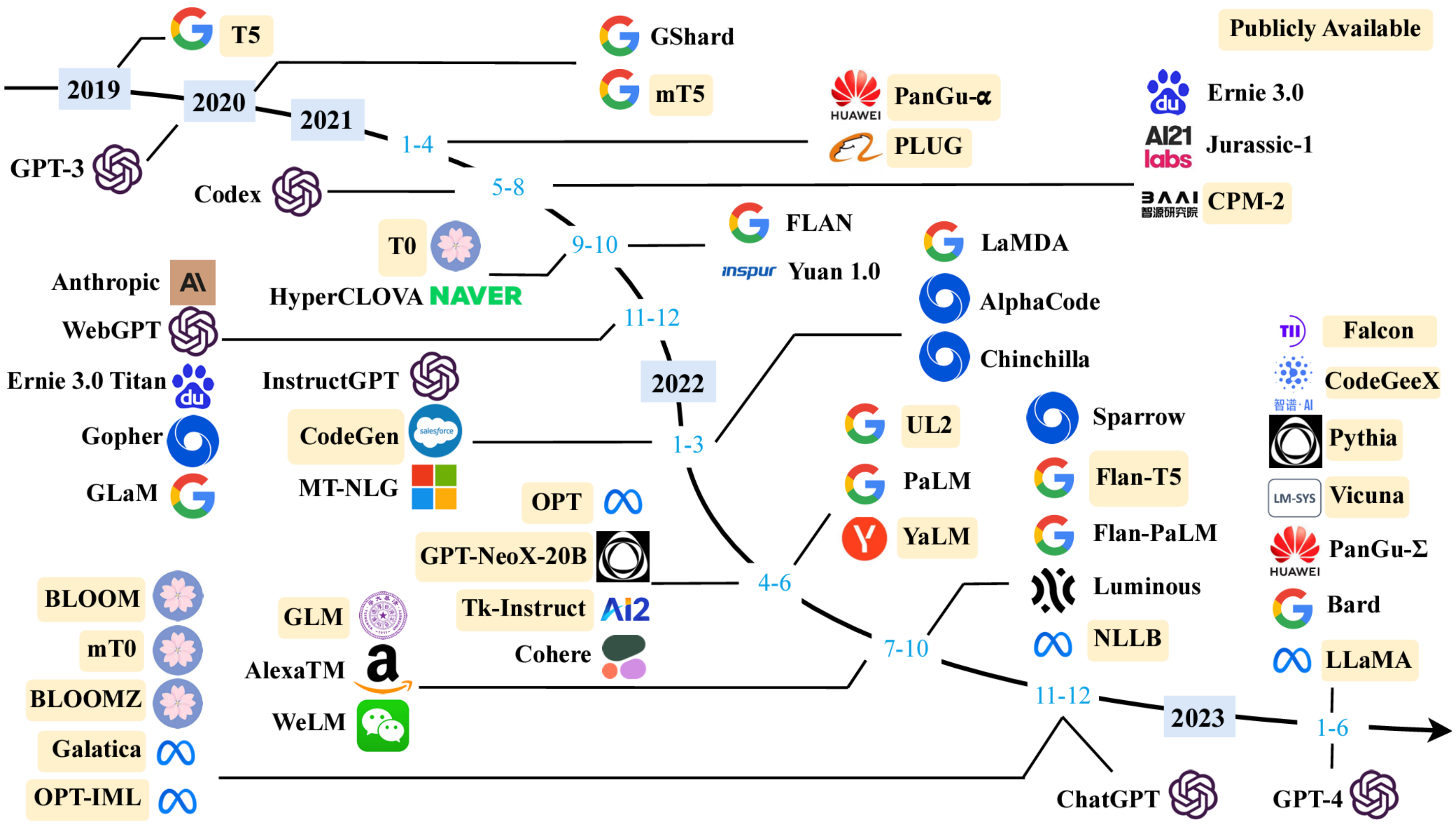

https://jalammar.github.io/illustrated-gpt2/

# Outline

**Lecture 10: Attention-based models and LLM Inference**

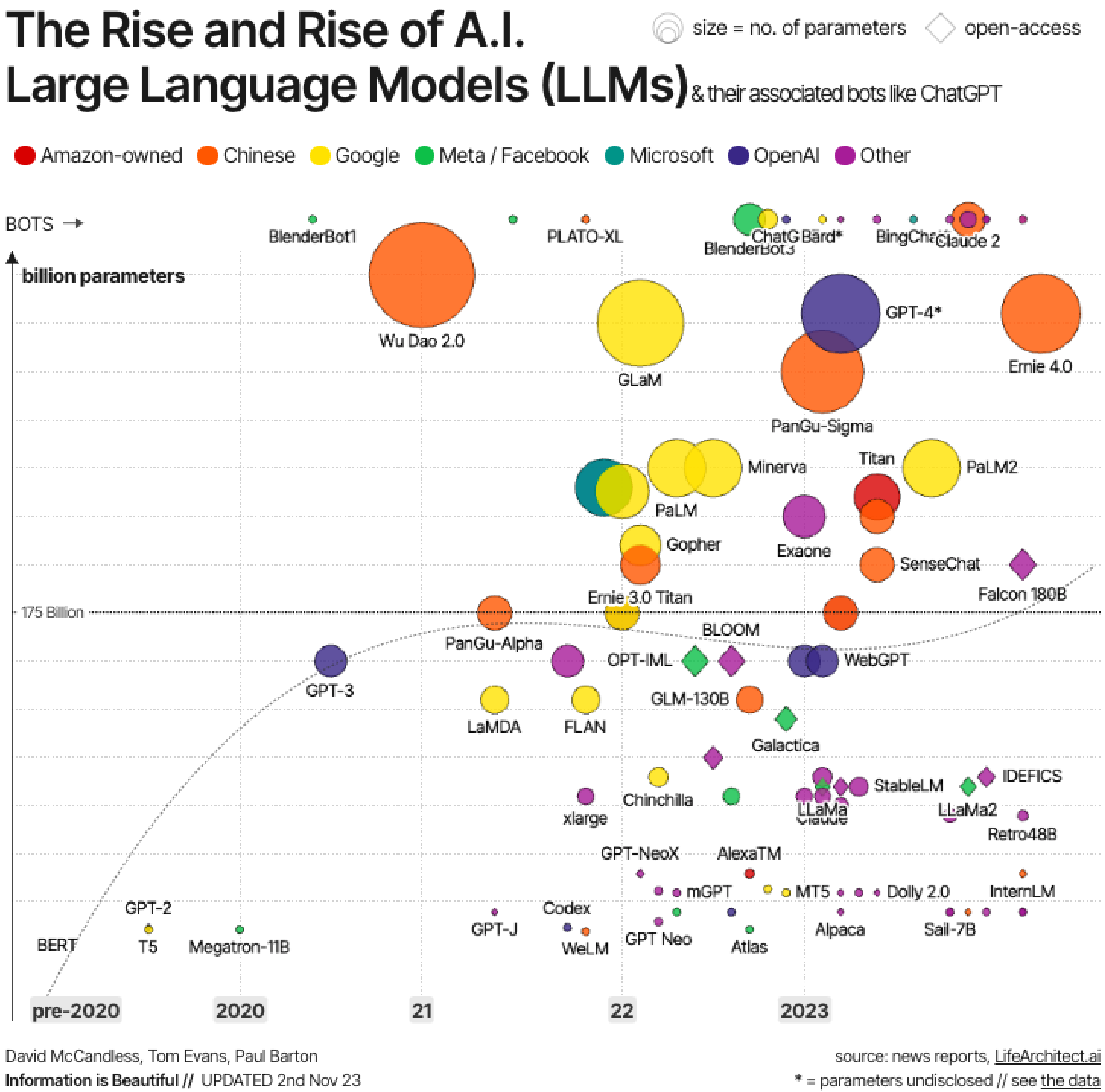
- Self-attention
- Transformer
- Vision Transformer
- Large Language Models
  - Architecture changes
  - **Scaling Up**

# Large Language Models (LLM)




A Survey of Large Language Models, [Zhao et al., 2023]

# Scaling up of LLM




https://www.datacamp.com/blog/what-is-an-llm-a-guide-on-large-language-models