

Deep Learning Transformer and Newer Architectures

Dareen Alharthi, Hao Chen

Spring 2025
Attendance:@964

Content

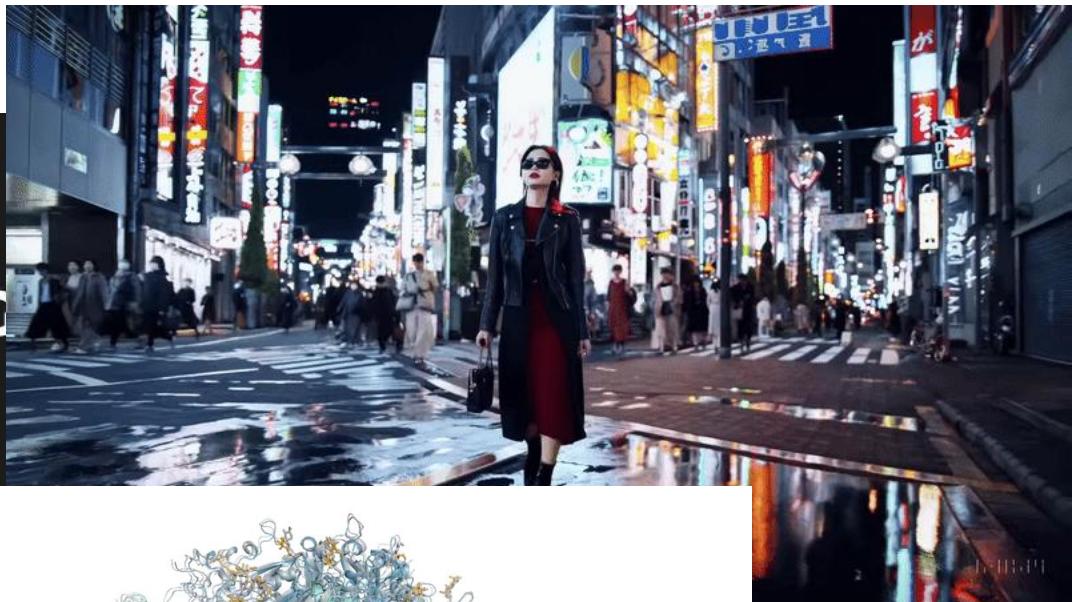
- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws

Content

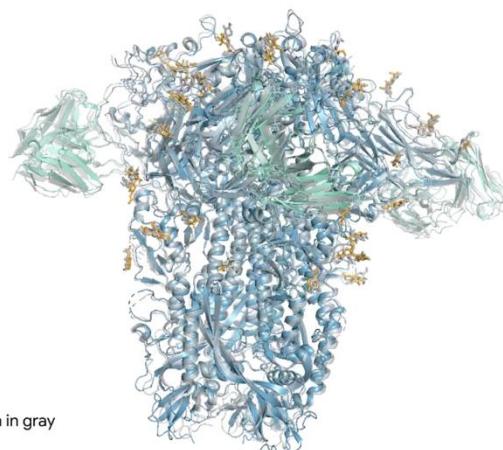
- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws

Why Transformer?

- Almost everything today in deep learning is Transformer



7PNM



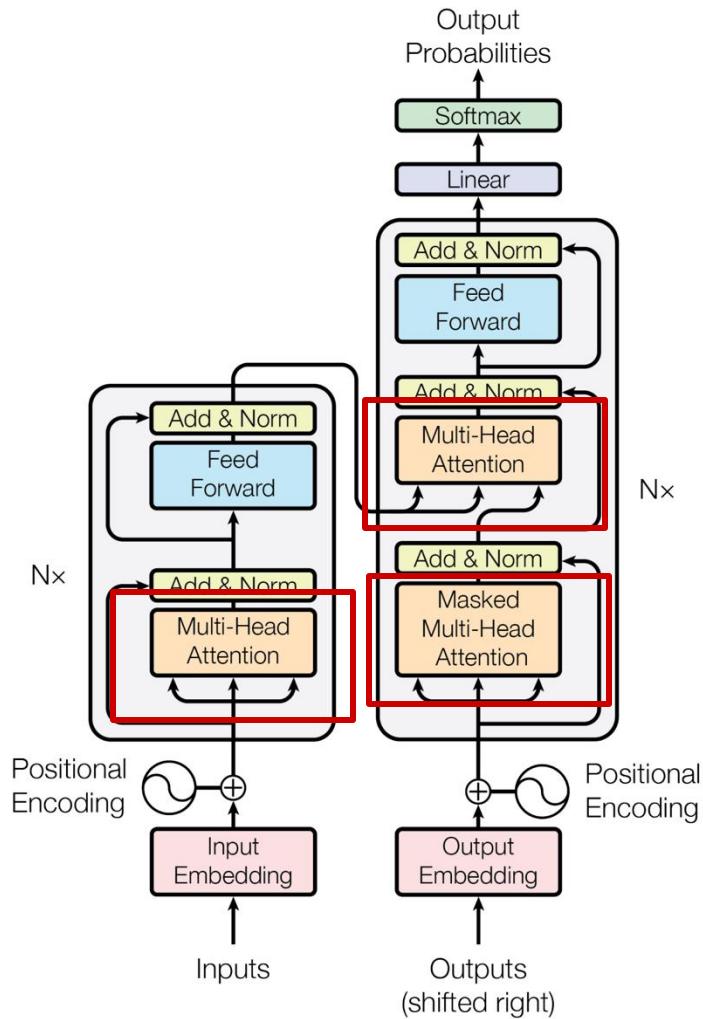
But...Why Transformer?

- Flexibility and universality of handling all modality
- Scaling with data and parameters
- “Emergent” capability and In-context Learning
- Parameter Efficient Tuning

Transformer Architecture

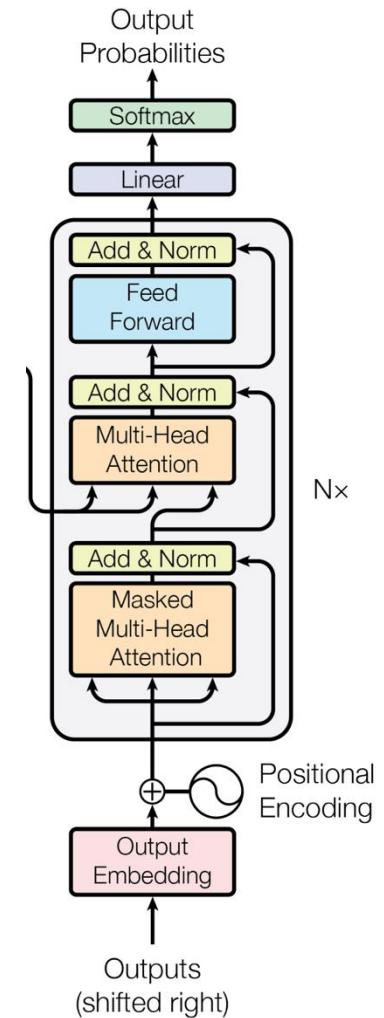
Transformer Architecture

- overview



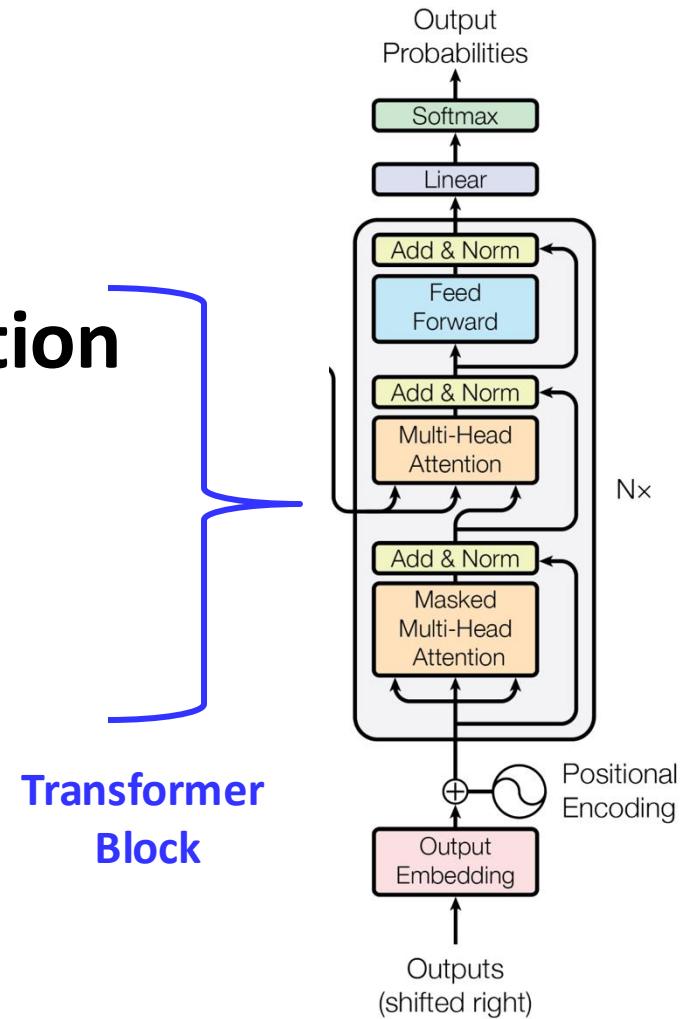
Transformer Architecture

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



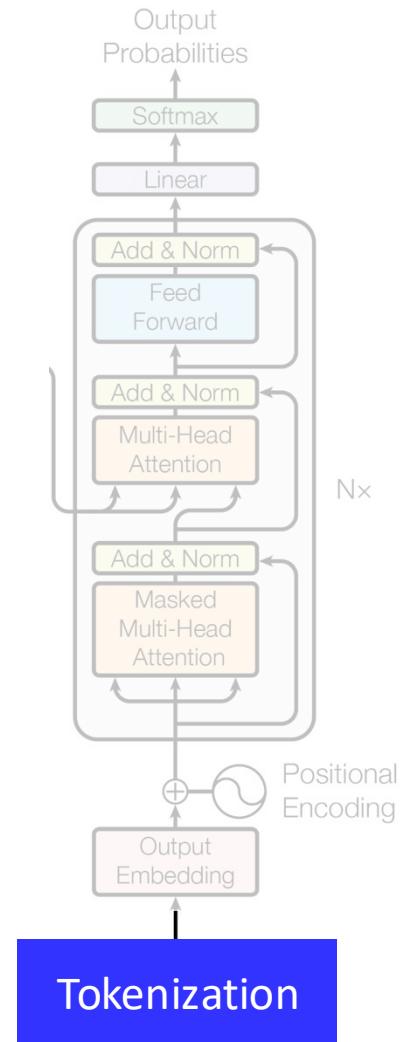
Transformer Architecture

- Word Tokenization
- Word Embedding
- **(Masked) Multi-Head Attention**
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Transformer Architecture

- **Word Tokenization**
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Tokenization

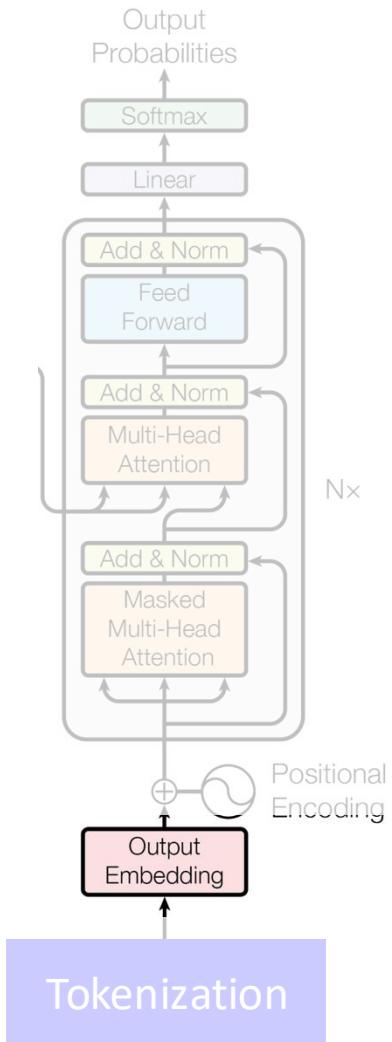
- Maps a word into one/multiple tokens
 - Each token represented as an index/class

Tokens	Characters
139	847
CMU's 11-785 Introduction to Deep Learning is a comprehensive course that offers students foundational knowledge and hands-on experience in deep learning. Designed to equip students with both theoretical concepts and practical skills, the course covers essential topics such as neural networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), generative models, and unsupervised learning techniques. It integrates mathematical foundations, optimization methods, and the latest advancements in model architectures, making it an ideal course for those interested in mastering deep learning applications across various domains. Students engage in coding assignments and projects that require implementing algorithms from scratch, giving them practical insight into real-world challenges and problem-solving with deep learning.	

Tokens	Characters
139	847
[14170, 52, 802, 220, 994, 12, 45085, 42915, 316, 28896, 25392, 382, 261, 16796, 4165, 484, 5297, 4501, 138200, 7124, 326, 8950, 13237, 3240, 306, 8103, 7524, 13, 53706, 316, 15160, 4501, 483, 2973, 47221, 23753, 326, 17377, 7870, 11, 290, 4165, 17804, 8731, 15083, 2238, 472, 58480, 20240, 11, 137447, 280, 58480, 20240, 350, 124144, 82, 936, 94157, 58480, 20240, 350, 49, 19022, 82, 936, 2217, 1799, 7015, 11, 326, 3975, 5813, 37861, 7524, 12905, 13, 1225, 91585, 58944, 64929, 11, 34658, 7933, 11, 326, 290, 6898, 102984, 306, 2359, 138910, 11, 4137, 480, 448, 9064, 4165, 395, 2617, 9445, 306, 133763, 8103, 7524, 9391, 5251, 5890, 45513, 13, 23372, 22338, 306, 22458, 41477, 326, 8554, 484, 1841, 36838, 41730, 591, 29133, 11, 9874, 1373, 17377, 24058, 1511, 1374, 52939, 13525, 326, 4792, 122400, 483, 8103, 7524, 13]	

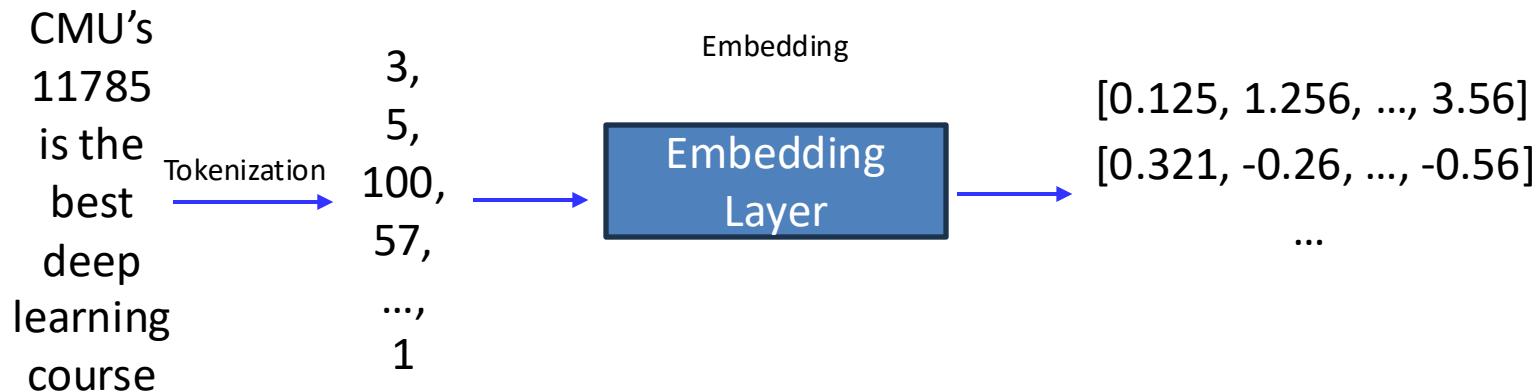
Transformer Architecture

- Word Tokenization
- **Word Embedding**
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Embedding

- Represents each discrete token index as continuous token embeddings



Embedding Layer

- The embedding layer is a look-up table that converts token index to continuous vectors

Token Index	Token Embedding
0	[0.235, -1.256, 3.513, ..., -0.187]
1	[1.291, -2.012, 0.624, ..., -1.291]
2	[0.536, 0.012, -0.024, ..., 2.345]
...	...
Vocab Size $ V $	[0.131, 2.102, 0.935, ..., -0.125]

- In Pytorch, it is $nn.Embedding$

Embedding Layer is a Linear Layer

- $nn.Embedding$ is essentially a linear layer $Y = XW$

One-Hot Vector
Token Index $X \in \mathbb{R}^{L \times |V|}$

Weight Matrix $W \in \mathbb{R}^{|V| \times D}$

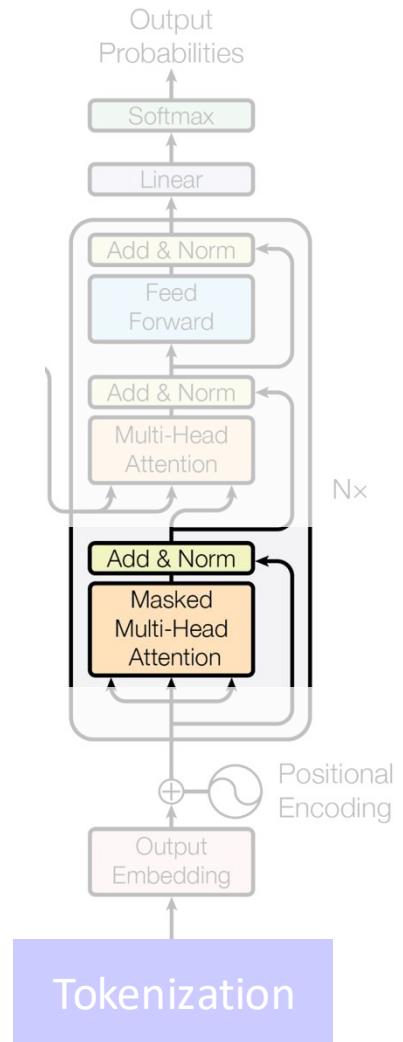
$$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} 0.235 & -1.256 & 3.513 & \dots & -0.187 \\ 1.291 & -2.012 & 0.624 & \dots & -1.291 \\ 0.535 & 0.012 & -0.024 & \dots & 2.345 \\ \dots & \dots & \dots & \dots & \dots \\ 0.131 & 2.102 & 0.935 & \dots & -0.125 \end{bmatrix}$$

$$\begin{bmatrix} 1.291 & -2.012 & 0.624 & \dots & -1.291 \\ 0.535 & 0.012 & -0.024 & \dots & 2.345 \\ 0.131 & 2.102 & 0.935 & \dots & -0.125 \end{bmatrix}$$

Token Embedding $Y \in \mathbb{R}^{L \times D}$

Transformer Architecture

- Word Tokenization
- Word Embedding
- **(Masked) Multi-Head Attention**
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Self-Attention

- Attention Operation

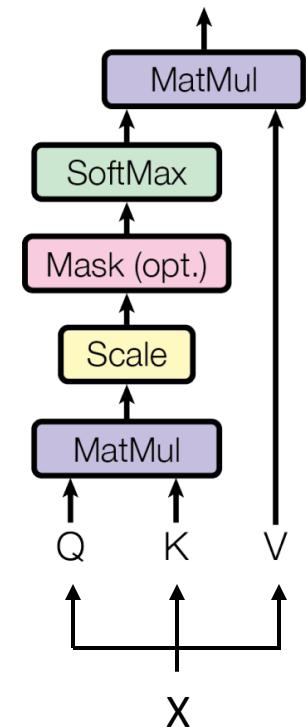
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

- Query-Key-Value

- Linear affine from input X itself

- Weighted-sum of V based on similarity/correlation between Q and K
 - Each token's weights sum to one



Self-Attention

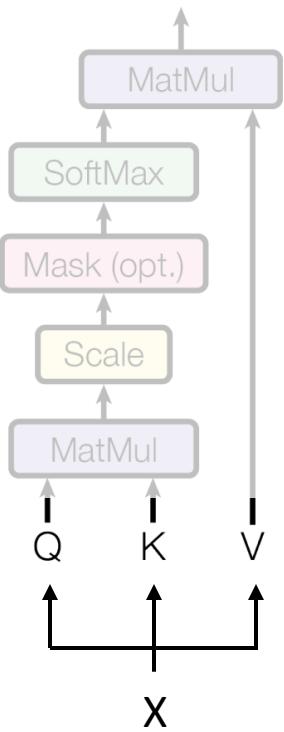
- Query-Key-Value from Three Linear Affine of X

$$\begin{array}{ccc} \mathbf{X} & & \mathbf{W}^Q \\ \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} & \times & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} \\ \mathbb{R}^{L \times D} & & \end{array} = \begin{array}{c} \mathbf{Q} \\ \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} \end{array}$$

$$\begin{array}{ccc} \mathbf{X} & & \mathbf{W}^K \\ \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} & \times & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} \\ \mathbb{R}^{L \times D} & & \end{array} = \begin{array}{c} \mathbf{K} \\ \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} \end{array}$$

$$\begin{array}{ccc} \mathbf{X} & & \mathbf{W}^V \\ \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} & \times & \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} \\ \mathbb{R}^{L \times D} & & \end{array} = \begin{array}{c} \mathbf{V} \\ \begin{matrix} \text{---} \\ \text{---} \\ \text{---} \end{matrix} \end{array}$$

Scaled Dot-Product Attention



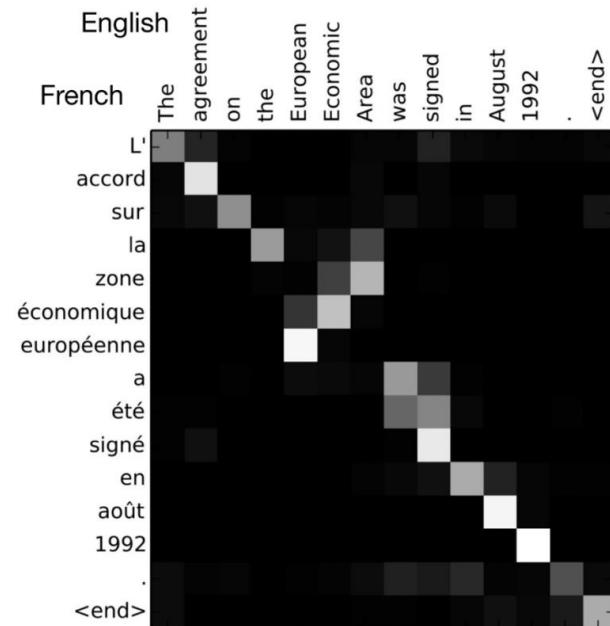
Self-Attention

- Attention weights

$$\text{softmax} \left(\frac{\begin{array}{c} \text{Q} \\ \times \\ \text{K}^T \end{array}}{\sqrt{d_k}} \right)$$

$\mathbb{R}^{L \times D}$

$\mathbb{R}^{L \times L}$



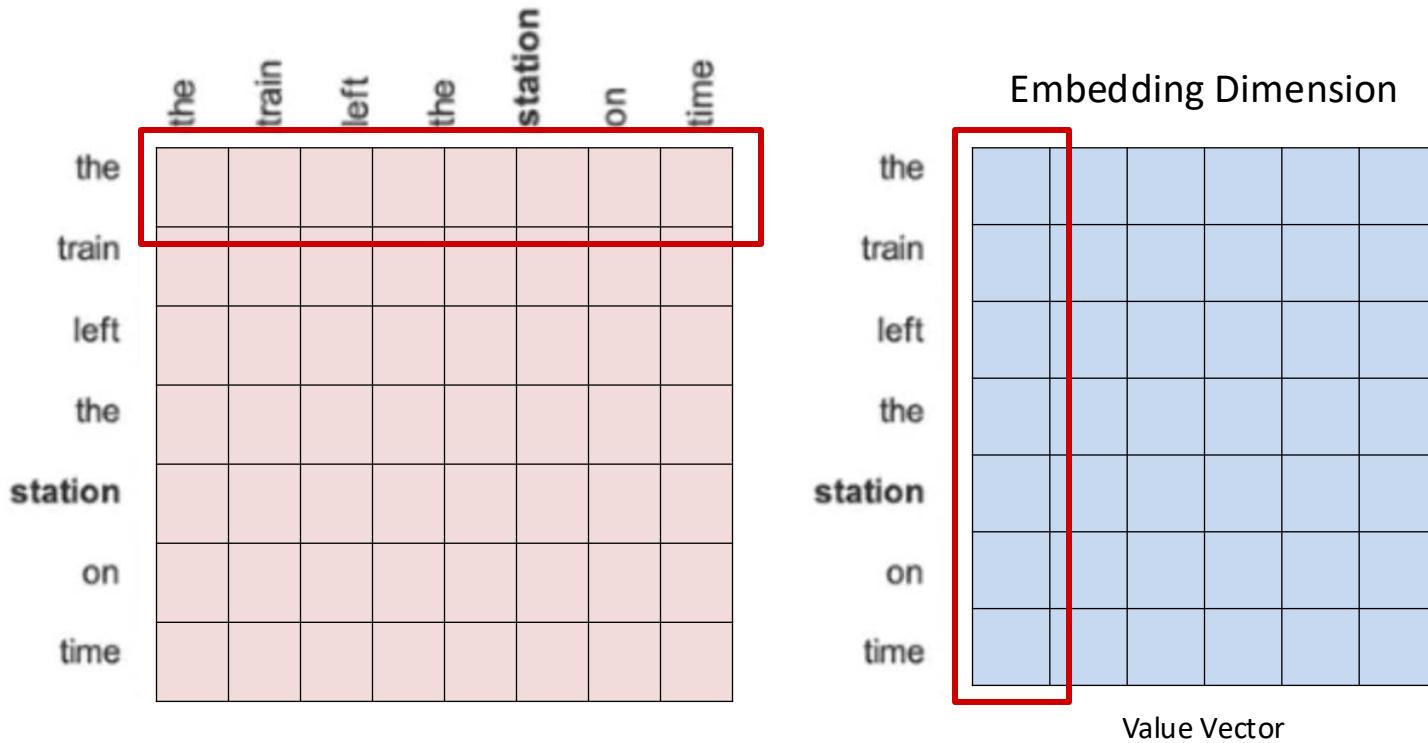
Self-Attention

- Output

$$\text{softmax} \left(\frac{\begin{matrix} Q \\ \times \\ K^T \end{matrix}}{\sqrt{d_k}} \right) V = Z$$

The diagram illustrates the computation of self-attention. It shows three input tensors: Q (purple, 3x3), K^T (orange, 3x3), and V (blue, 3x3). The Q and K^T tensors are multiplied together, and the result is divided by $\sqrt{d_k}$. This result is then multiplied by the V tensor to produce the output Z (pink, 3x3).

Self-Attention



Weighted-sum of V based on Attention Scores

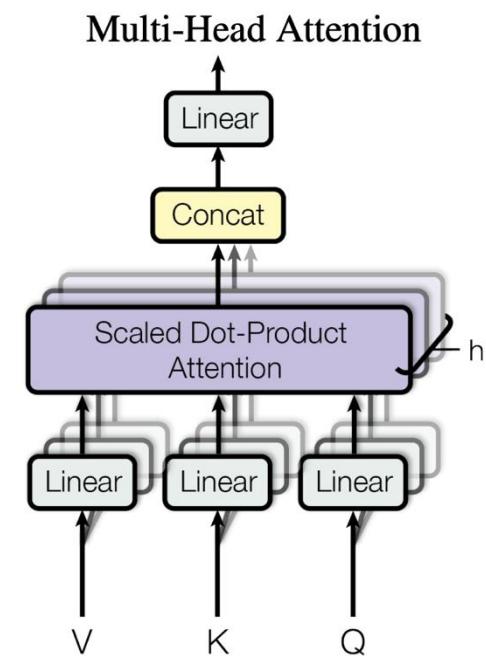
Multi-Head Self-Attention

- Multiple self-attention operations over the channel dimension

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^Q$$

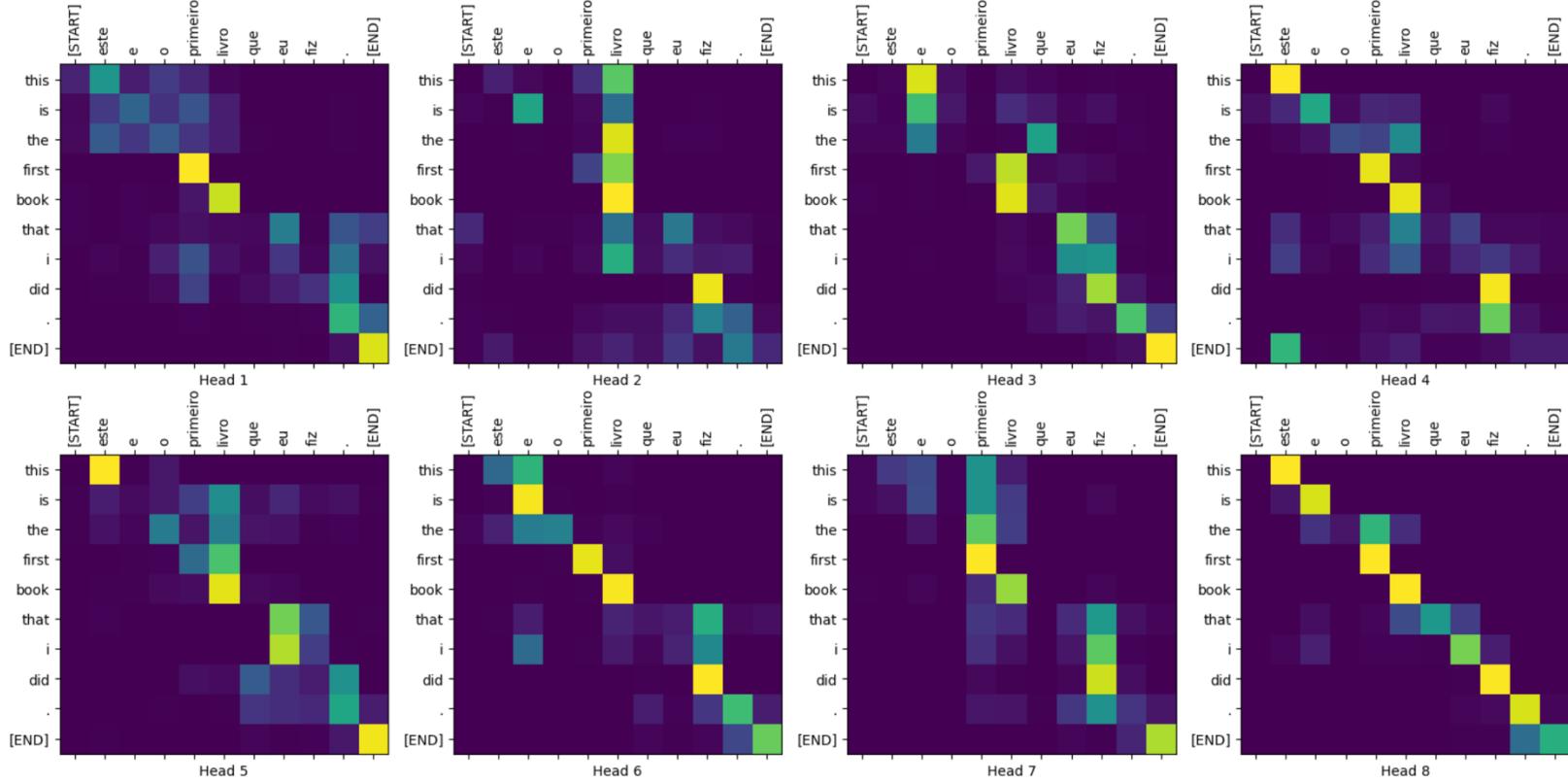
where $\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$

- Different attention maps capture different relationships

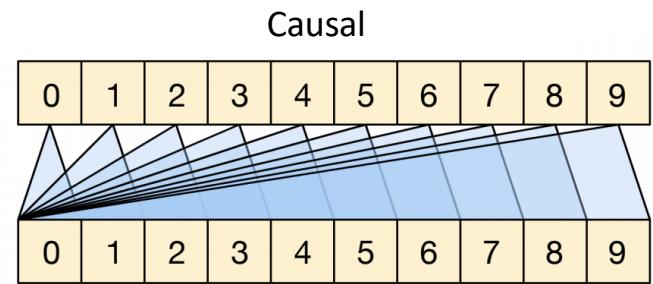
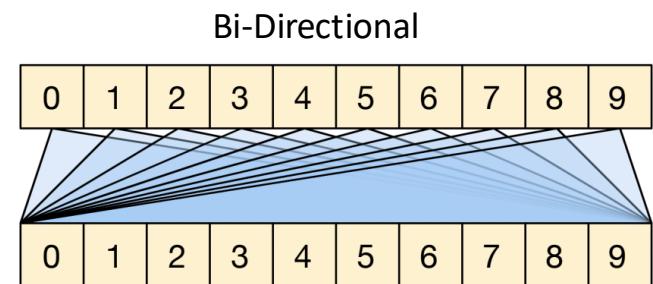
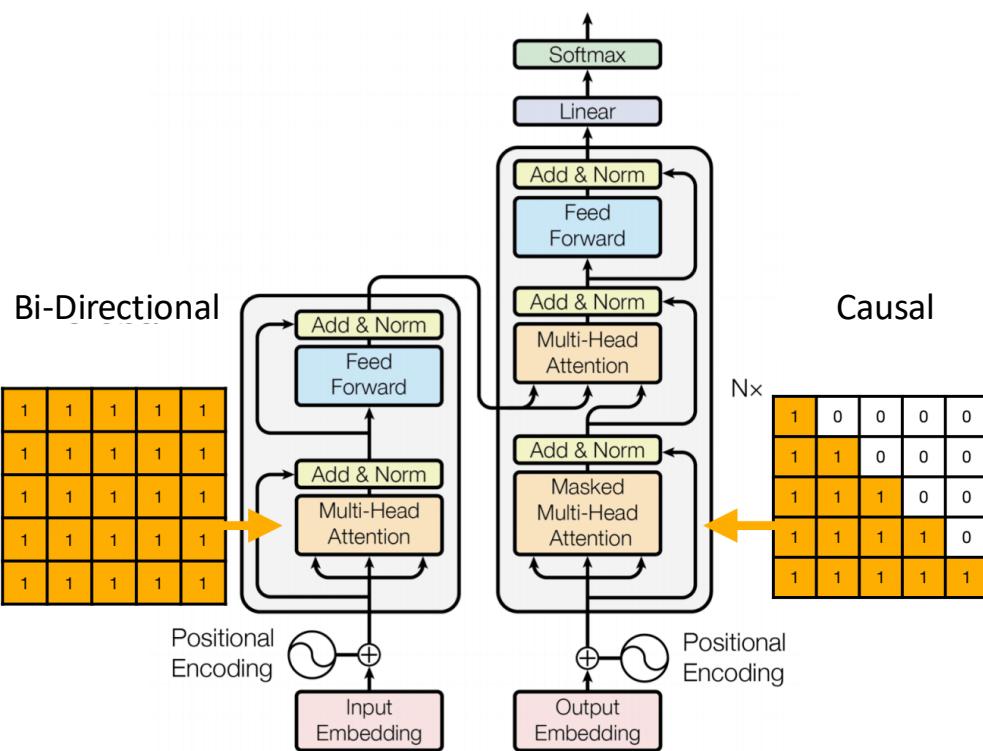


Multi-Head Attention

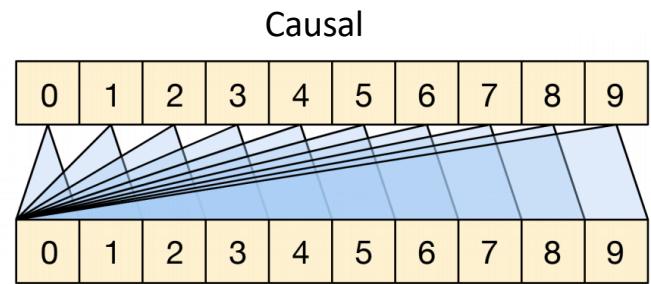
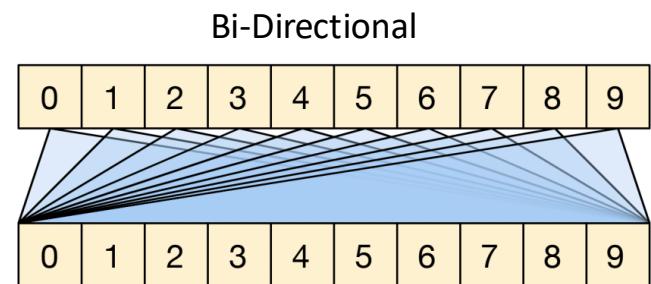
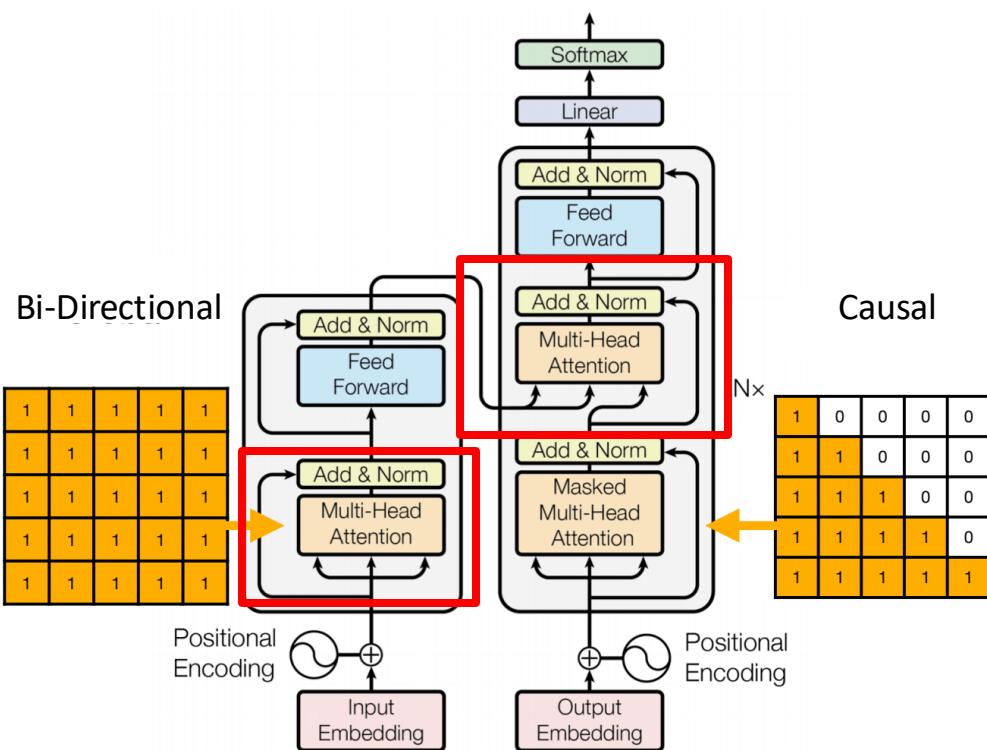
- Each head captures different semantics



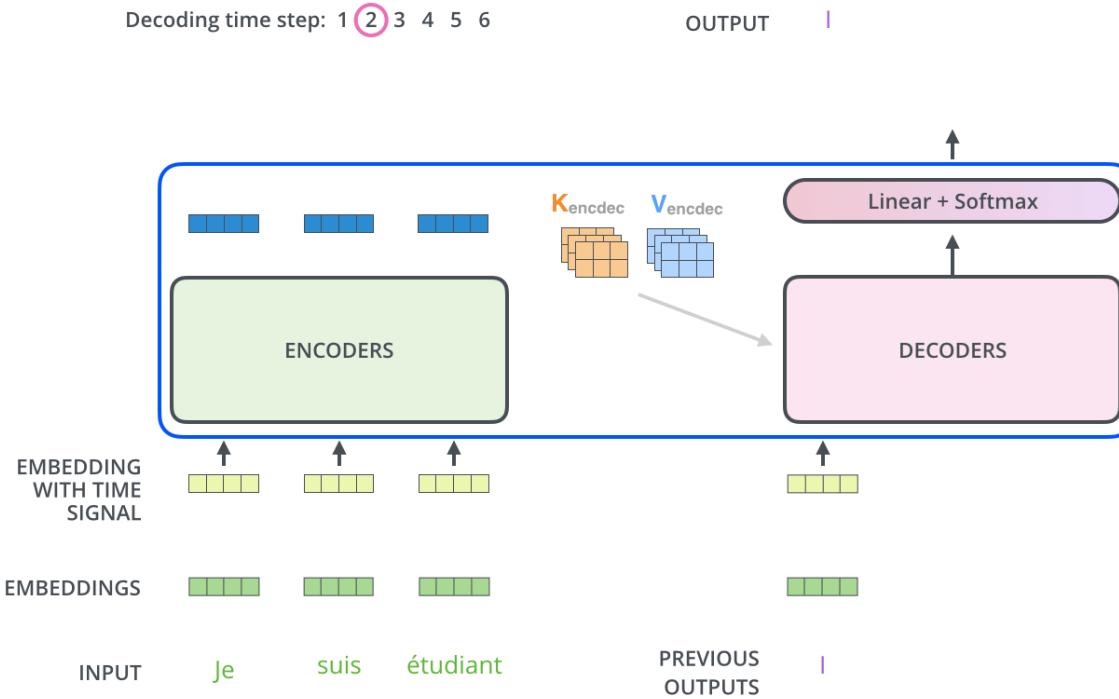
Attention Masking



Attention Masking

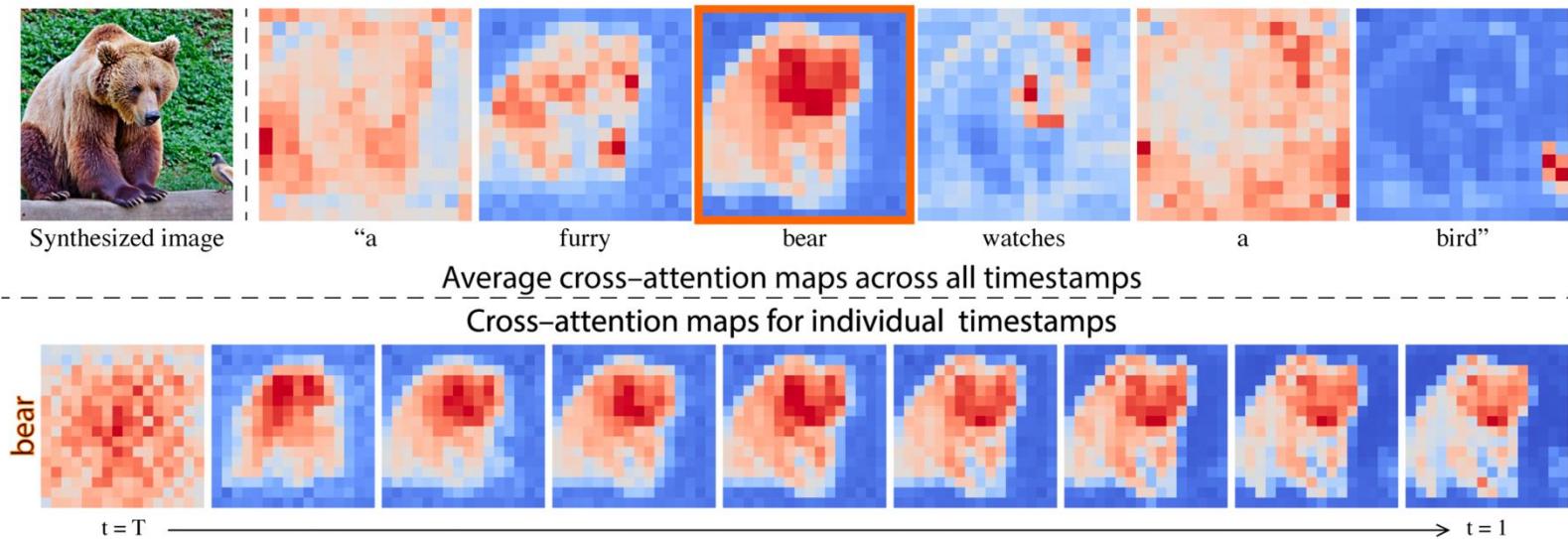


Cross-Attention



Cross-Attention

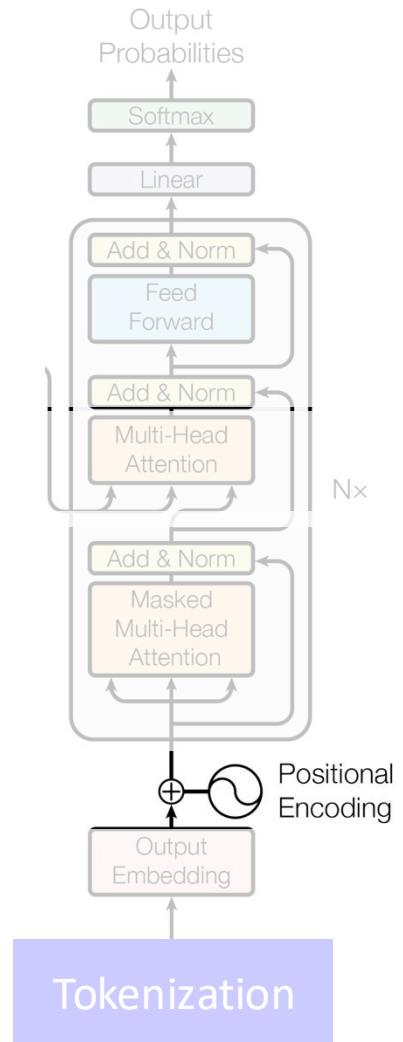
query, "a furry bear watches a bird."



The model iteratively denoise the noise vector based on the given text query to generate an Image

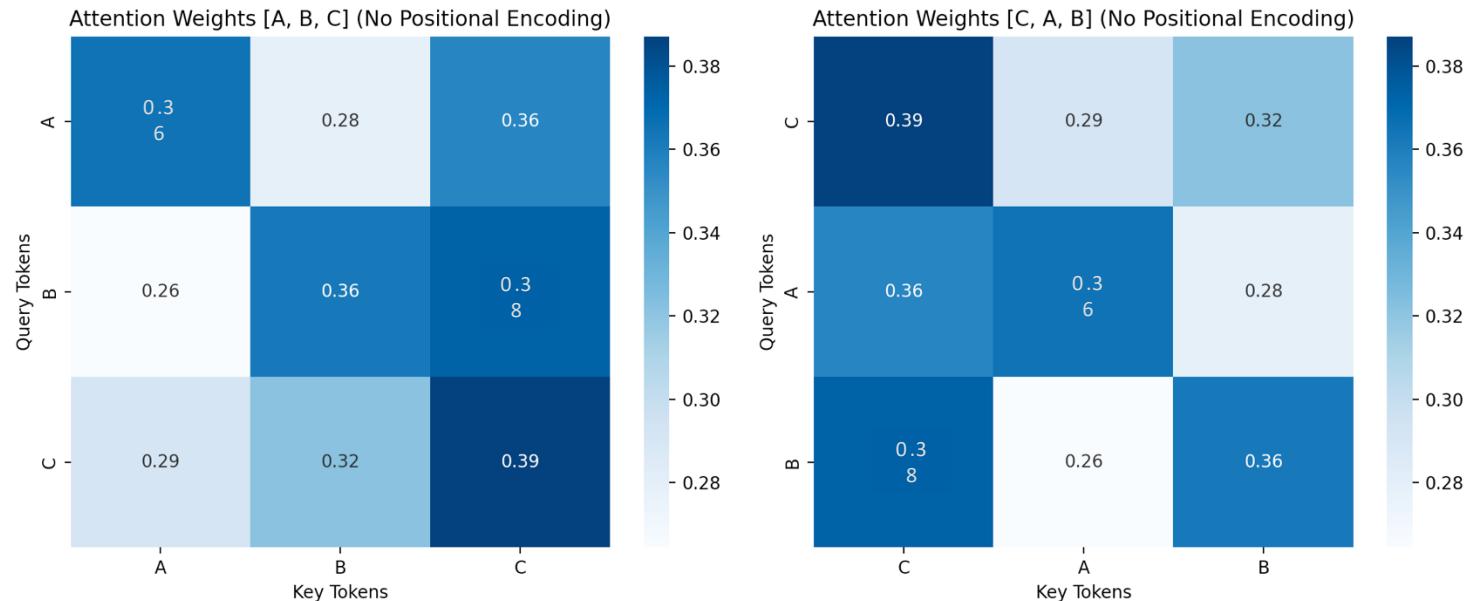
Transformer Architecture

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- **Position Encoding**
- Feed-Forward
- Add & Norm
- Output Projection Layer



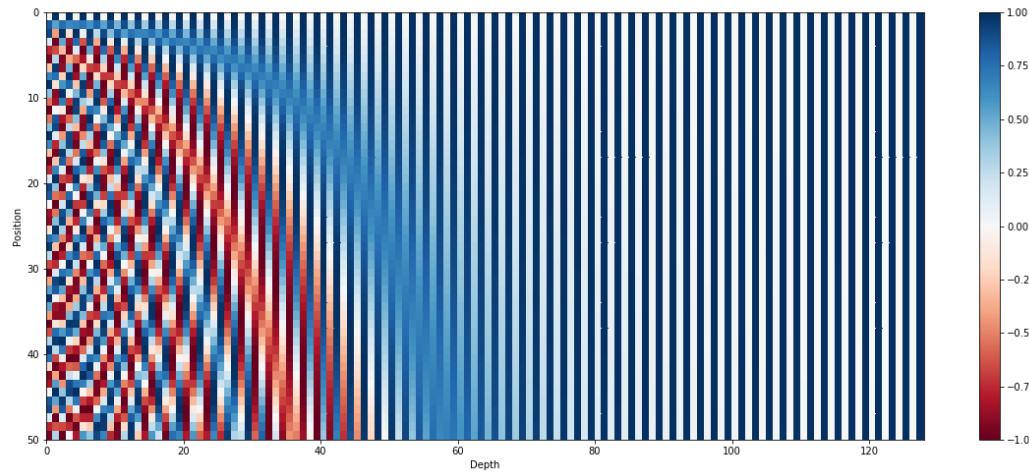
Position Encoding

- Why do we need them?
 - Self-attention is permutation-invariant!
- Considering a sequence of
 - [A, B, C] vs. [C, A, B]
- No position information!



Position Encoding

- Captures the abs./relative distance between tokens

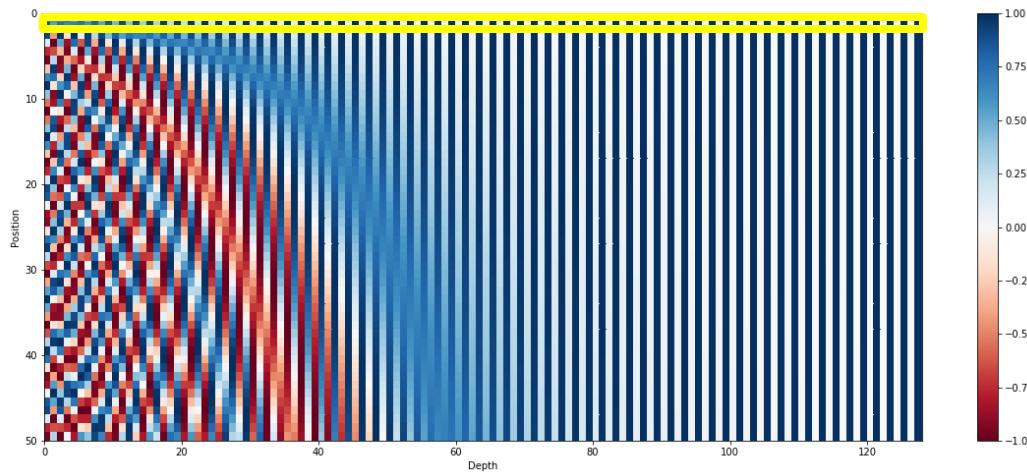


$$p_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

- A vector of sines and cosines of a harmonic series of frequencies
- Never Repeats

Position Encoding

- Captures the abs./relative distance between tokens

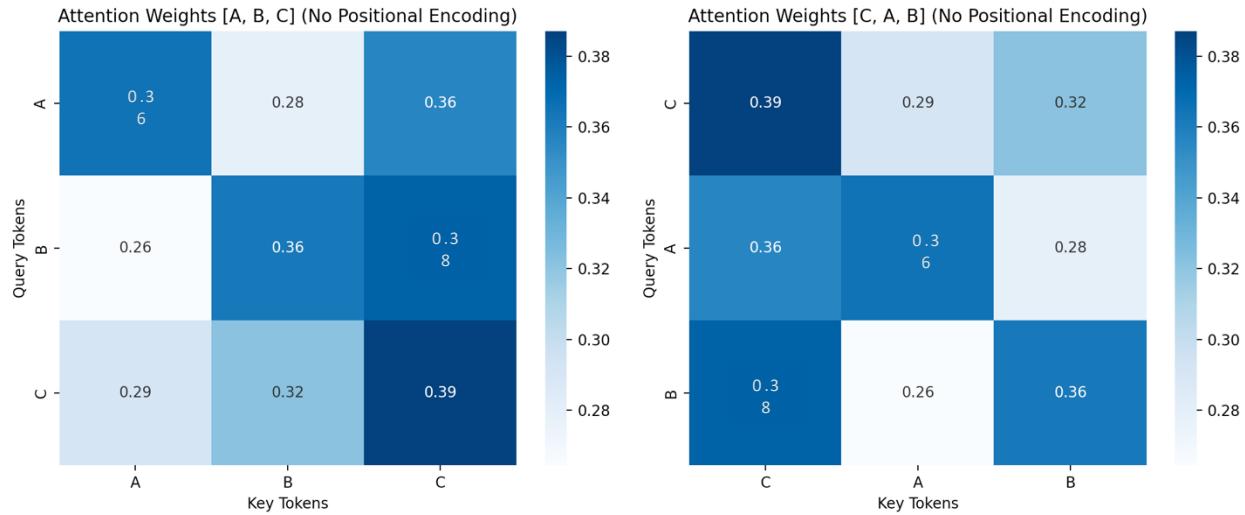


$$p_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

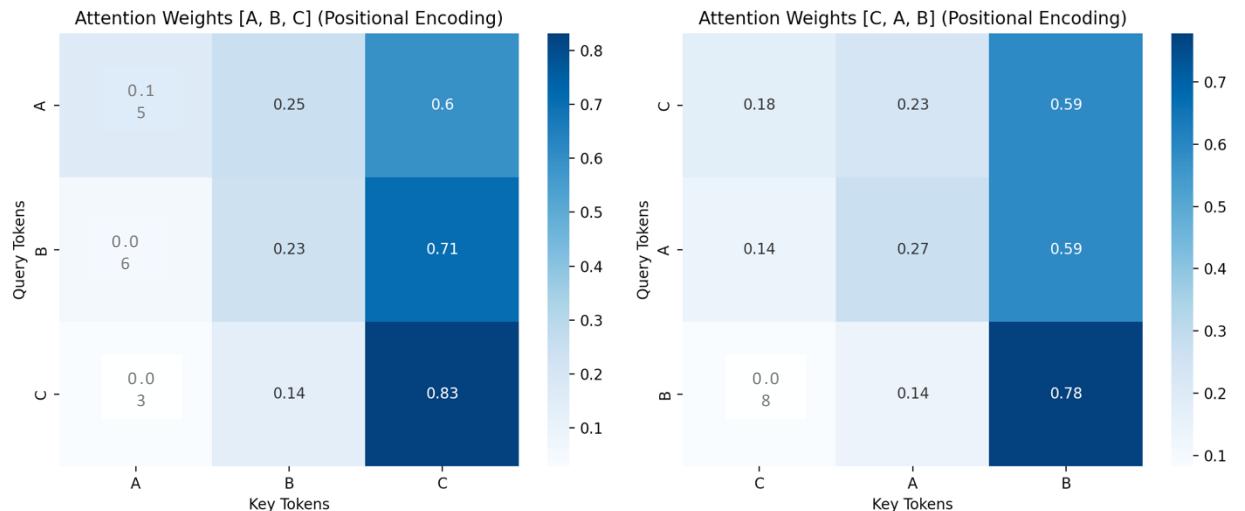
- A vector of sines and cosines of a harmonic series of frequencies
- Never Repeats

Position Encoding

No Position Info.

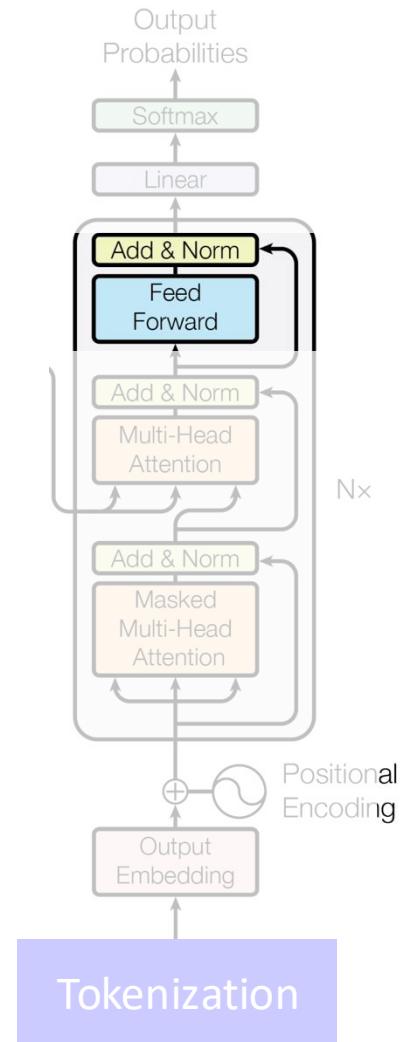


With Position Info.



Transformer Architecture

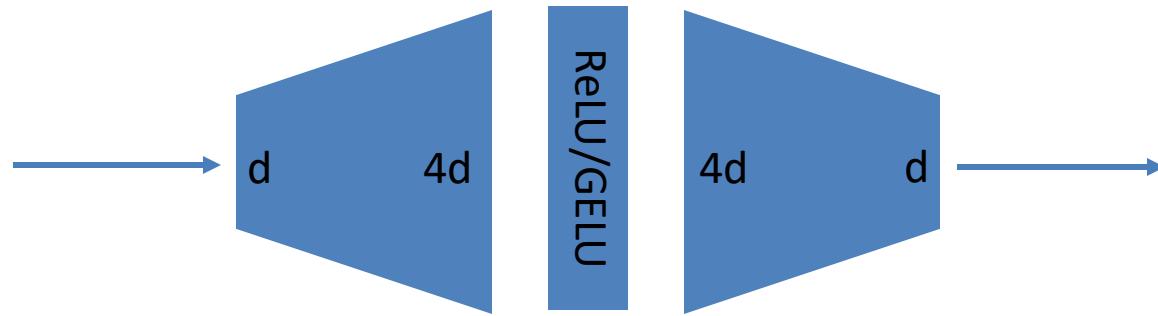
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- **Feed-Forward**
- Add & Norm
- Output Projection Layer



Feed-Forward Block

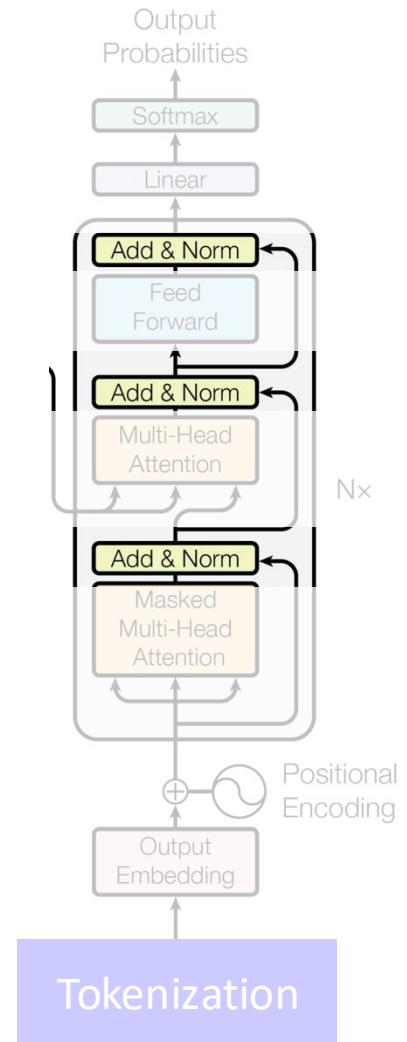
- Just a MLP!

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



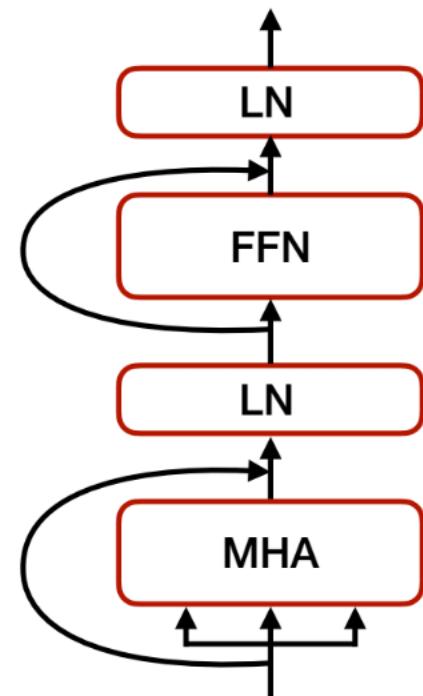
Transformer Architecture

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- **Add & Norm**
- Output Projection Layer



Residual and Normalization

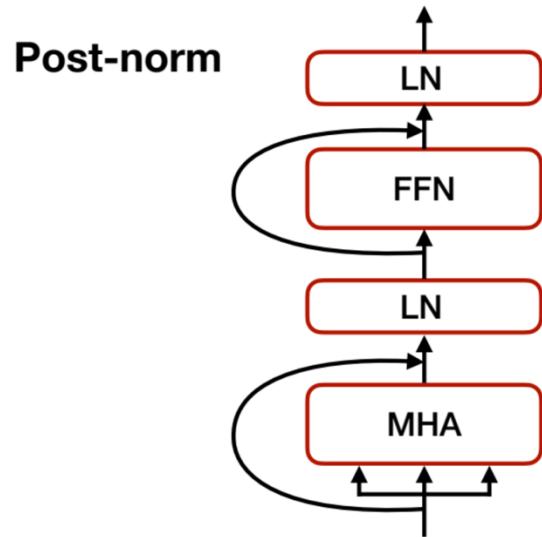
- Each layer in Transformer has:
 - A residual connection
 - A normalization layer
- Layer Norm. normalize each token by its embedding size dimension
 - For more stable training



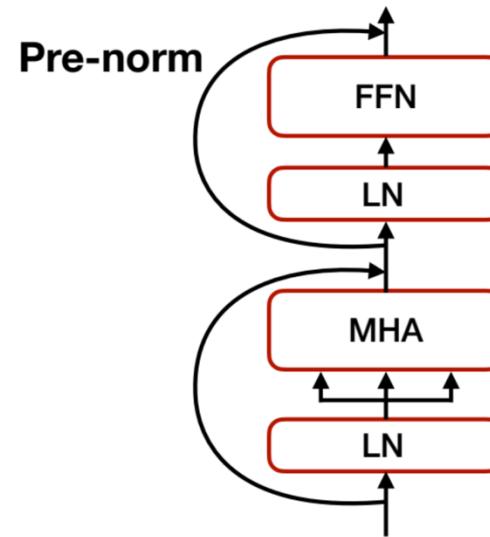
Position of Normalization

- Post-Norm vs Pre-Norm

$$\mathbf{x}_{t+1} = \text{Norm}(\mathbf{x}_t + F_t(\mathbf{x}_t))$$



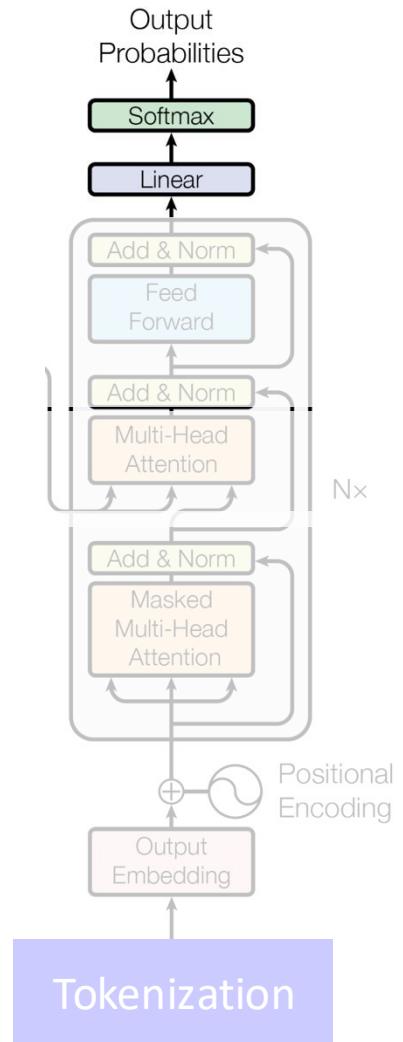
$$\mathbf{x}_{t+1} = \mathbf{x}_t + F_t(\text{Norm}(\mathbf{x}_t))$$



- Pre-Norm is easier and more stable to train
- Post-Norm tends to present better performance if properly trained

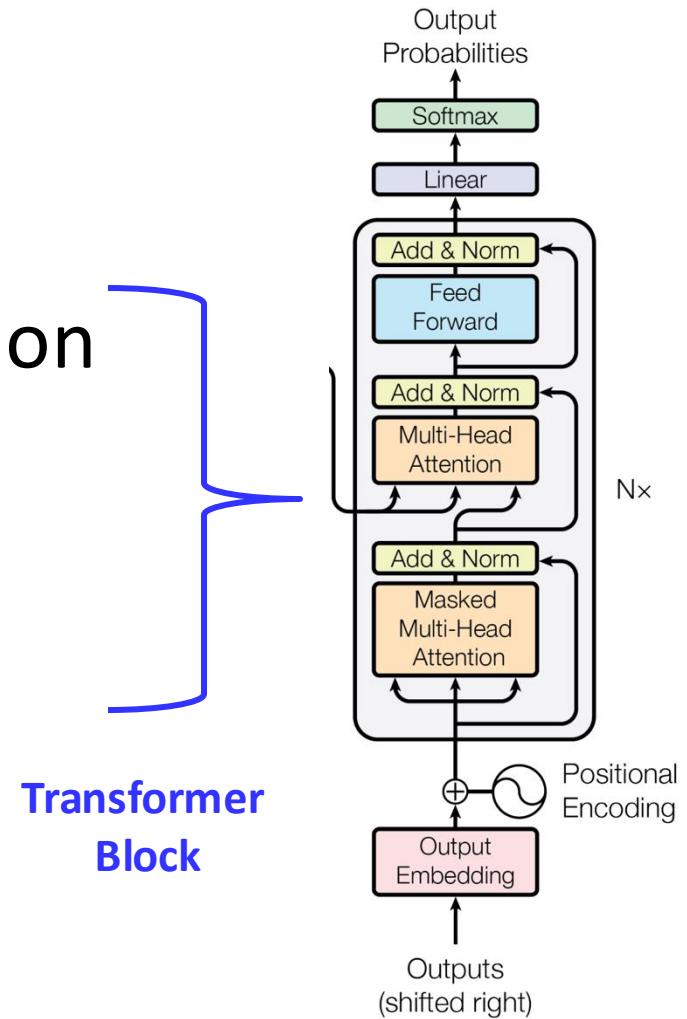
Transformer Architecture

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Feed-Forward
- Add & Norm
- Position Encoding
- **Output Projection Layer**
 - Just a linear layer



Putting Them Together - Transformer

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Transformer
Block

N_x

Positional
Encoding

Outputs
(shifted right)

Poll @967

Which of the following are true about self-attention?

- Self-attention is permutation invariant without position information
- The attention weights are scaled by the dimension d before computing softmax
- The attention weights are scaled by \sqrt{d} before computing softmax
- Q, K, V are copy of input X

Poll @967

Which of the following are true about self-attention?

- Self-attention is permutation invariant without position information
- The attention weights are scaled by the dimension d before computing softmax
- The attention weights are scaled by \sqrt{d} the dimension d before computing softmax
- Q, K, V are copy of input X

Content

- Transformer Architecture
- **Improvements on Transformers**
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws

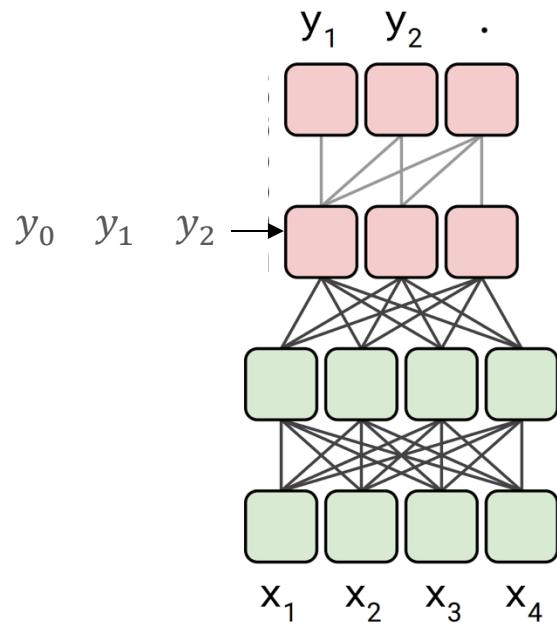
Overview

- Architecture
 - Encoder-Decoder
 - Encoder-Only
 - Decoder-Only
- Position Encoding
 - Relative Position Encoding
 - Rotary Position Encoding
- Efficient Attention Mechanism
 - Grouped Query Attention
 - Multi Query Attention
 - Flash Attention
 - Multi-head Latent Attention

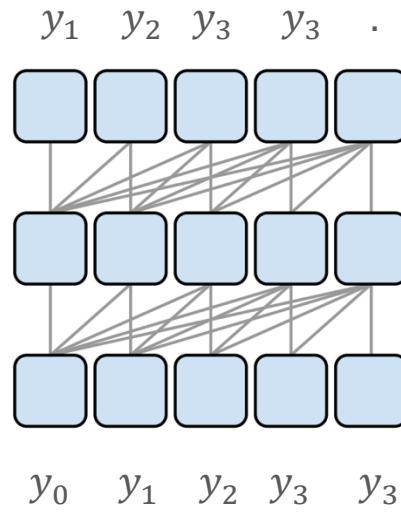
Overview

- Architecture
 - Encoder-Decoder
 - Encoder-Only
 - Decoder-Only
- Position Encoding
 - Relative Position Encoding
 - Rotary Position Encoding
- Efficient Attention Mechanism
 - Grouped Query Attention
 - Multi Query Attention
 - Flash Attention
 - Multi-head Latent Attention

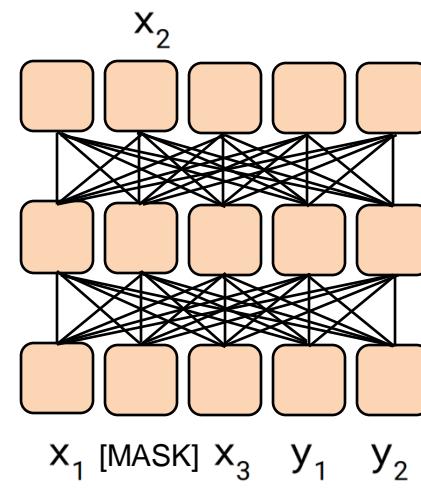
Encoder-Decoder



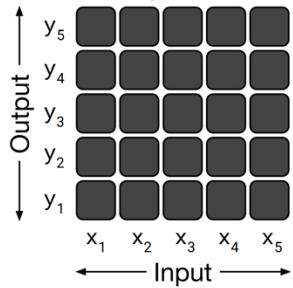
Decoder-Only



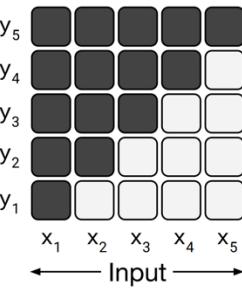
Encoder-Only



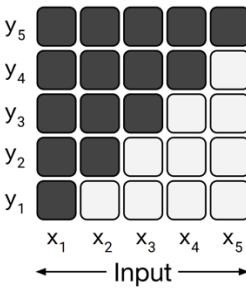
Fully-visible



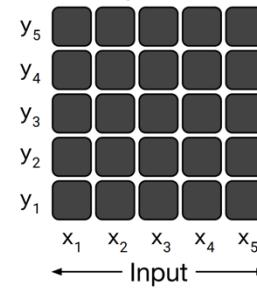
Causal



Causal

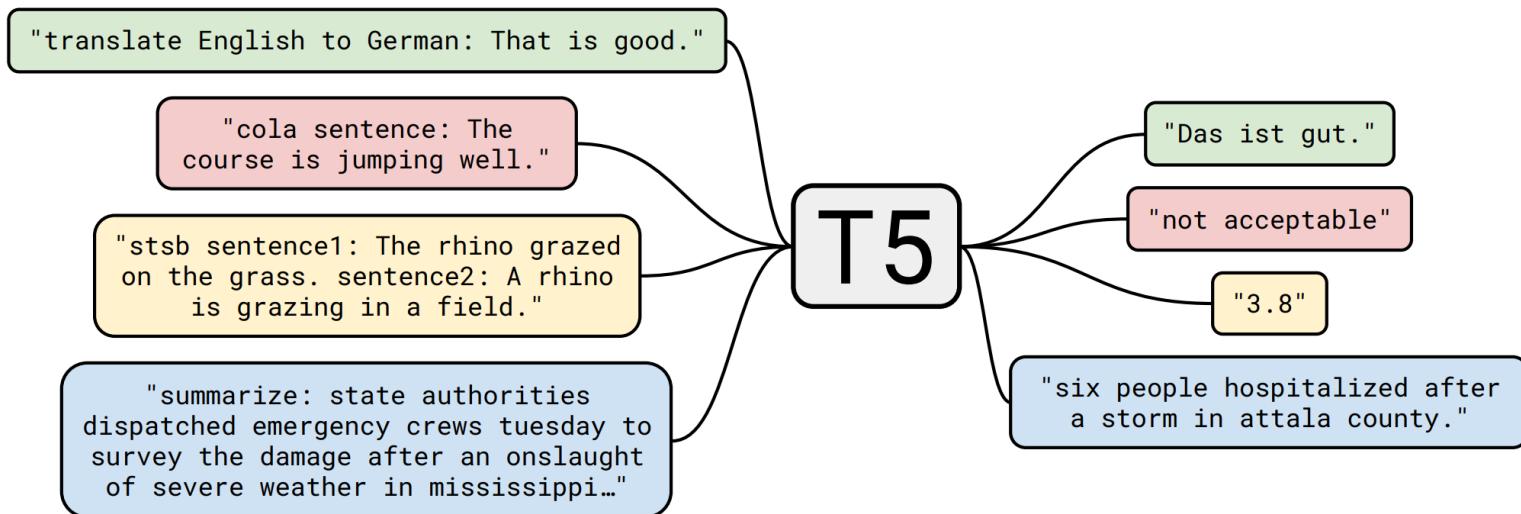


Fully-visible



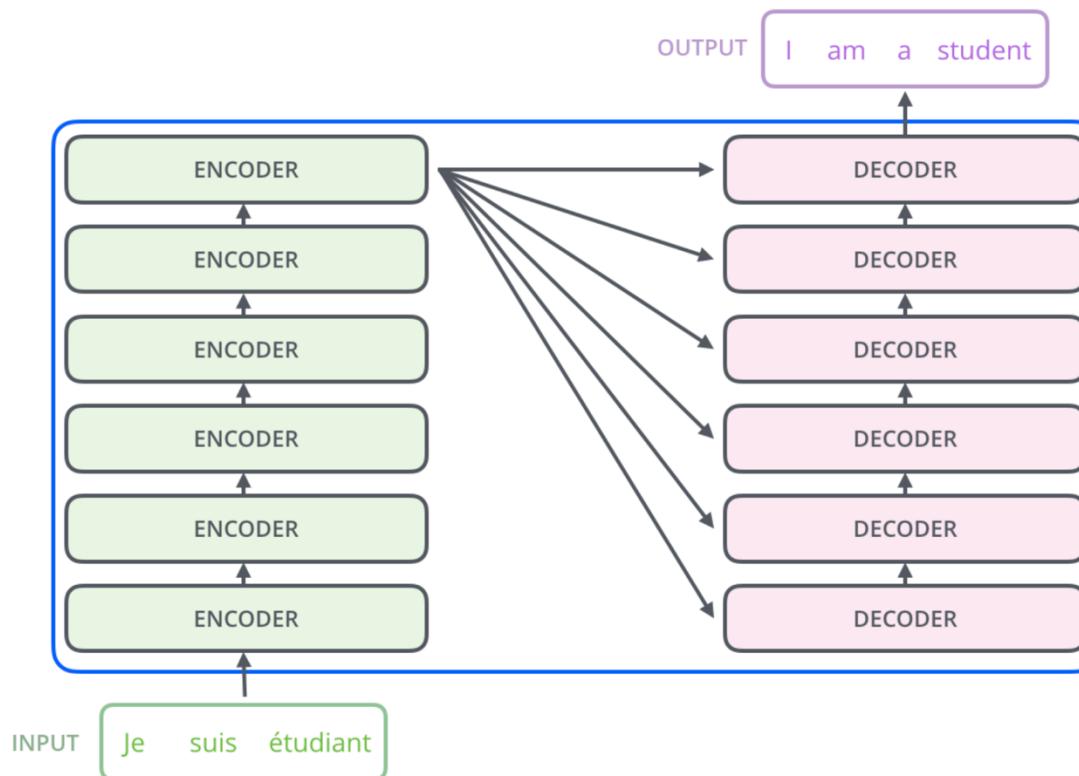
Encoder-Decoder - T5

- Encoder-Decoder architecture as in the original transformer paper
- A text-to-text model on various NLP tasks



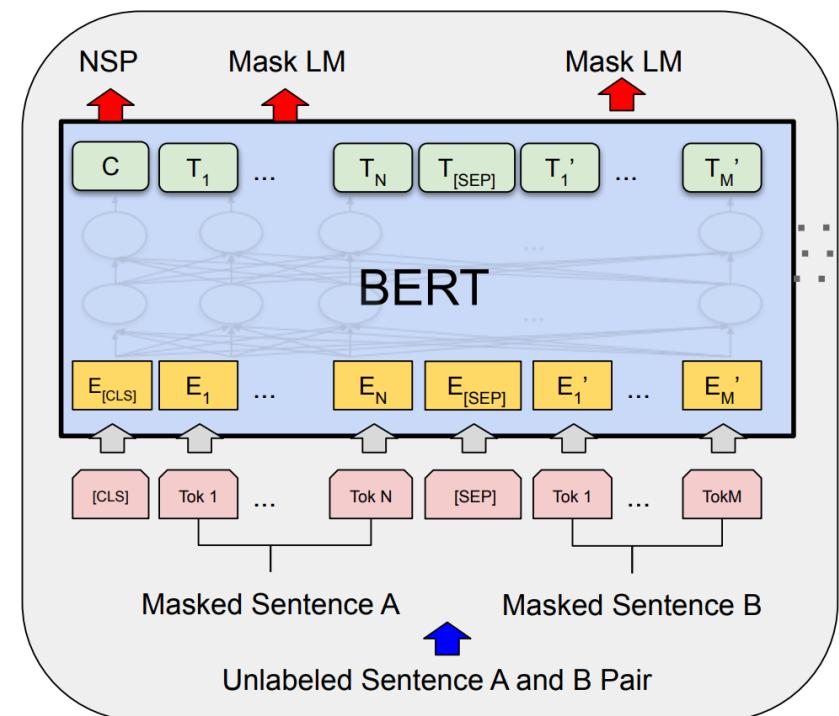
Encoder-Decoder - T5

- The prompt is fed into encoder, and the decoder generates answer



Encoder-Only - BERT

- Bidirectional Encoder Representations from Transformers (BERT)
 - Encoder-only arch.
- Trained with
 - Mask token prediction
 - Next sentence prediction



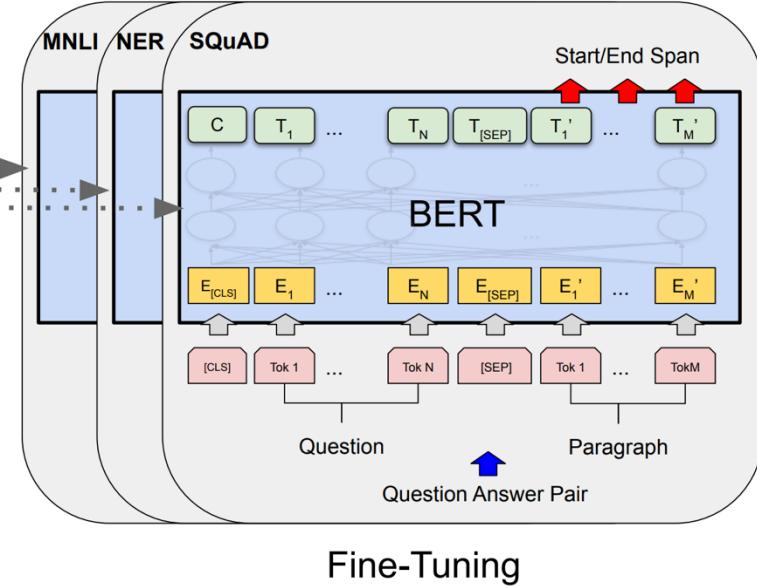
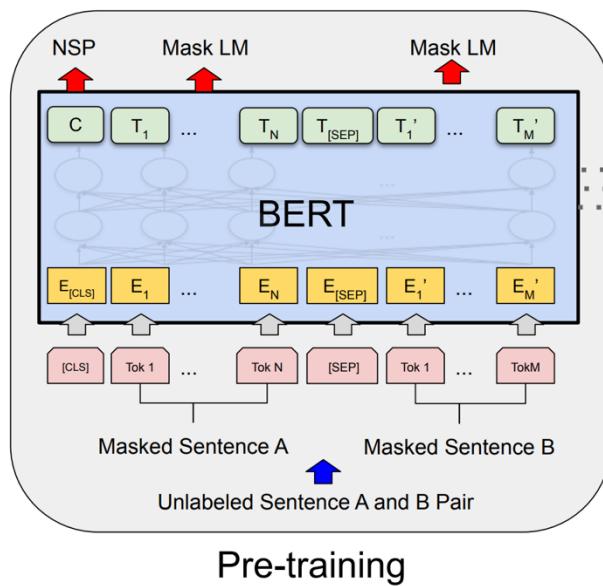
Pre-training and then Fine-Tuning

Pre-training on a proxy task

- Masked token prediction
- Next sentence prediction

Fine-tuning on specific downstream tasks

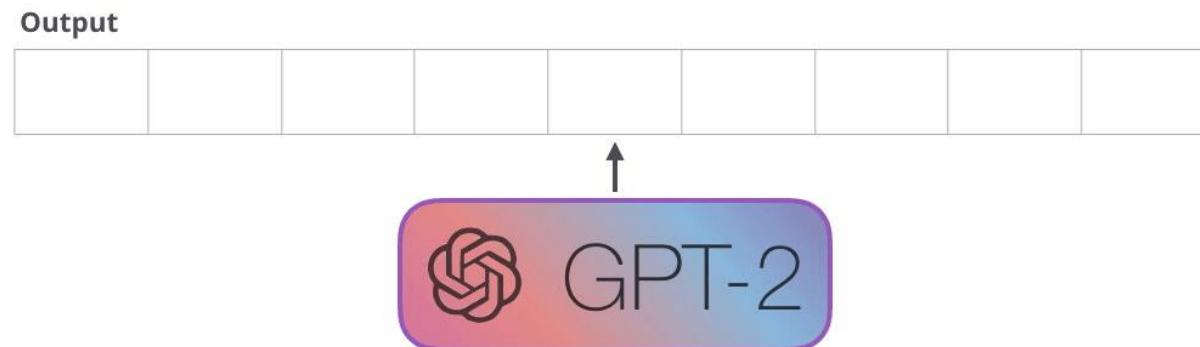
- Machine translation
- Question answering



Decoder-Only - GPT

- Generative Pre-training (GPT)
 - Decoder-only
- Trained with next token prediction
 - A language model!

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1}; \Theta)$$



Large Language Model

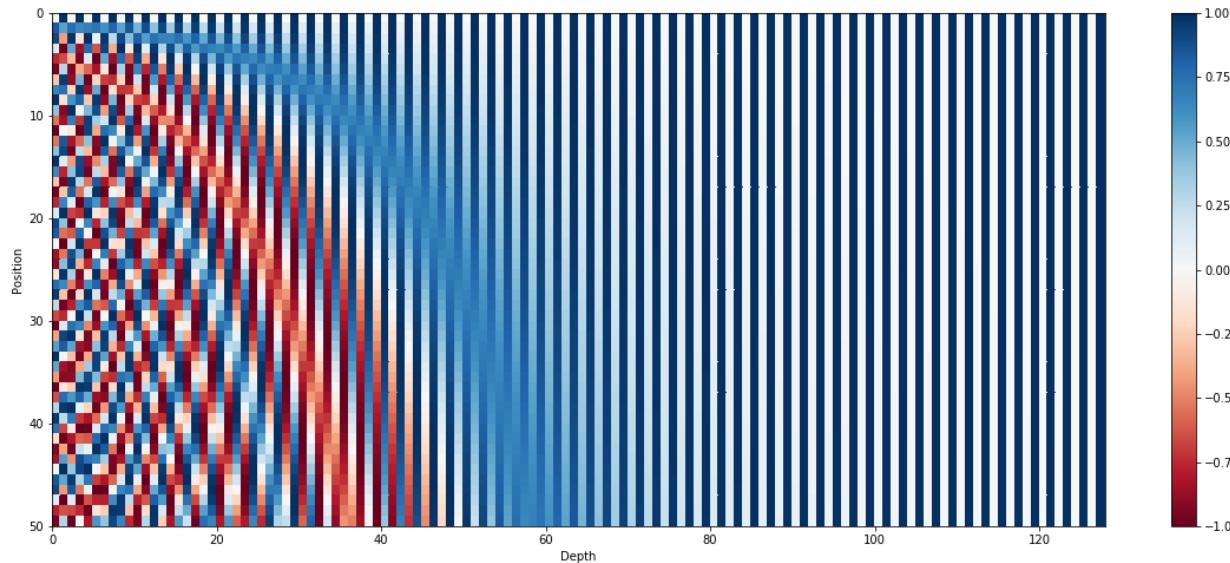
- GPT-2
 - Pre-training and fine-tuning on specific tasks
- GPT-3
 - zero-shot capability
 - in-context learning
 - Foundation for ChatGPT!
- GPT-4

Overview

- Architecture
 - Encoder-Decoder
 - Encoder-Only
 - Decoder-Only
- Position Encoding
 - Relative Position Encoding
 - Rotary Position Encoding
- Efficient Attention Mechanism
 - Grouped Query Attention
 - Multi Query Attention
 - Flash Attention
 - Multi-head Latent Attention

Absolute Position Encoding

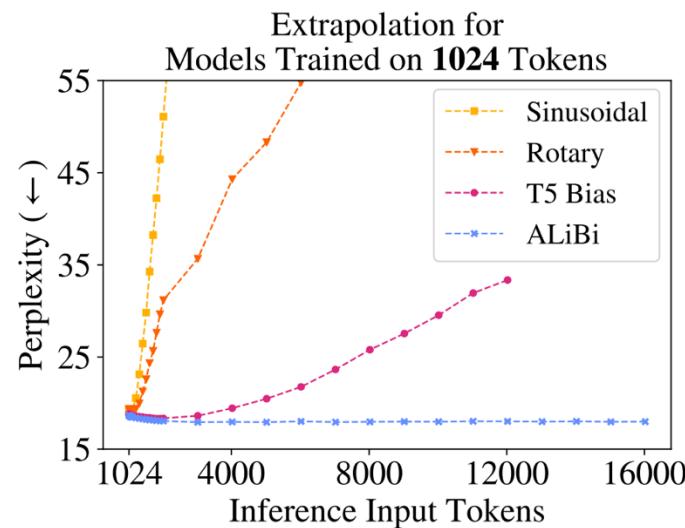
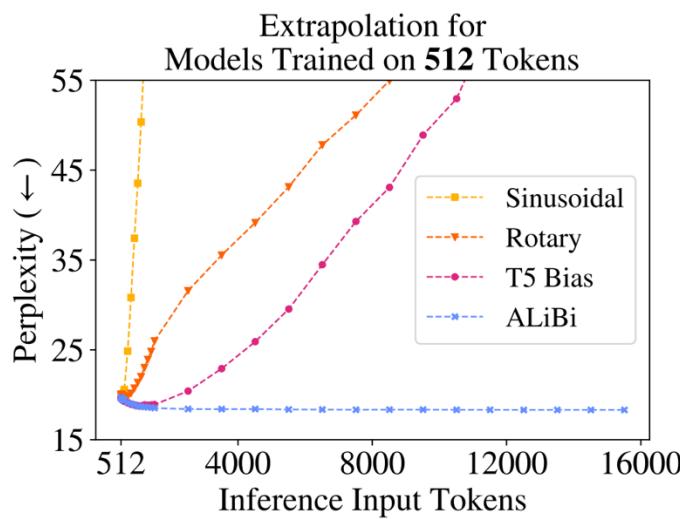
- Absolute position embedding fuses the position information into input embeddings



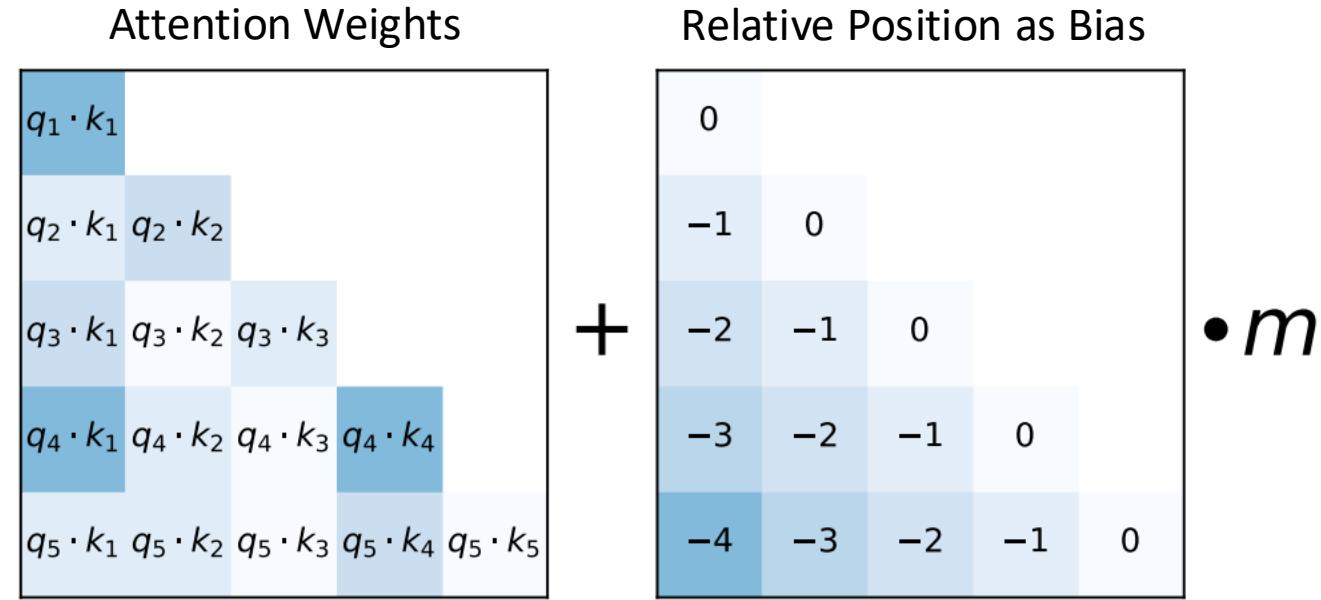
- Fixed length! Not generalize to longer input sequence

Relative Position Encoding

- Relative position embedding fuses position information into attention matrices
- Attention with linear bias
 - Input length extrapolation!



Relative Position Encoding



- Relative distance as offset added to attention matrix
- Absolute position embedding not needed

Rotary Position Encoding

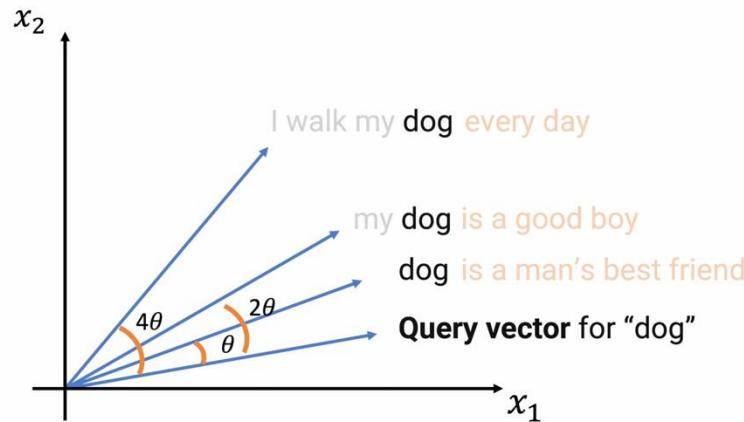
- Used in Large Language Models such as LLAMA
- Rotate the embedding in 2D space

$$\langle f_q(x_m, m), f_k(x_n, n) \rangle = g(x_m, x_n, m - n)$$

$$f_q(x_m, m) = (W_q x_m) e^{im\theta}$$

$$f_k(x_n, n) = (W_k x_n) e^{in\theta}$$

$$g(x_m, x_n, m - n) = \operatorname{Re} \left[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta} \right]$$



Rotary Position Encoding

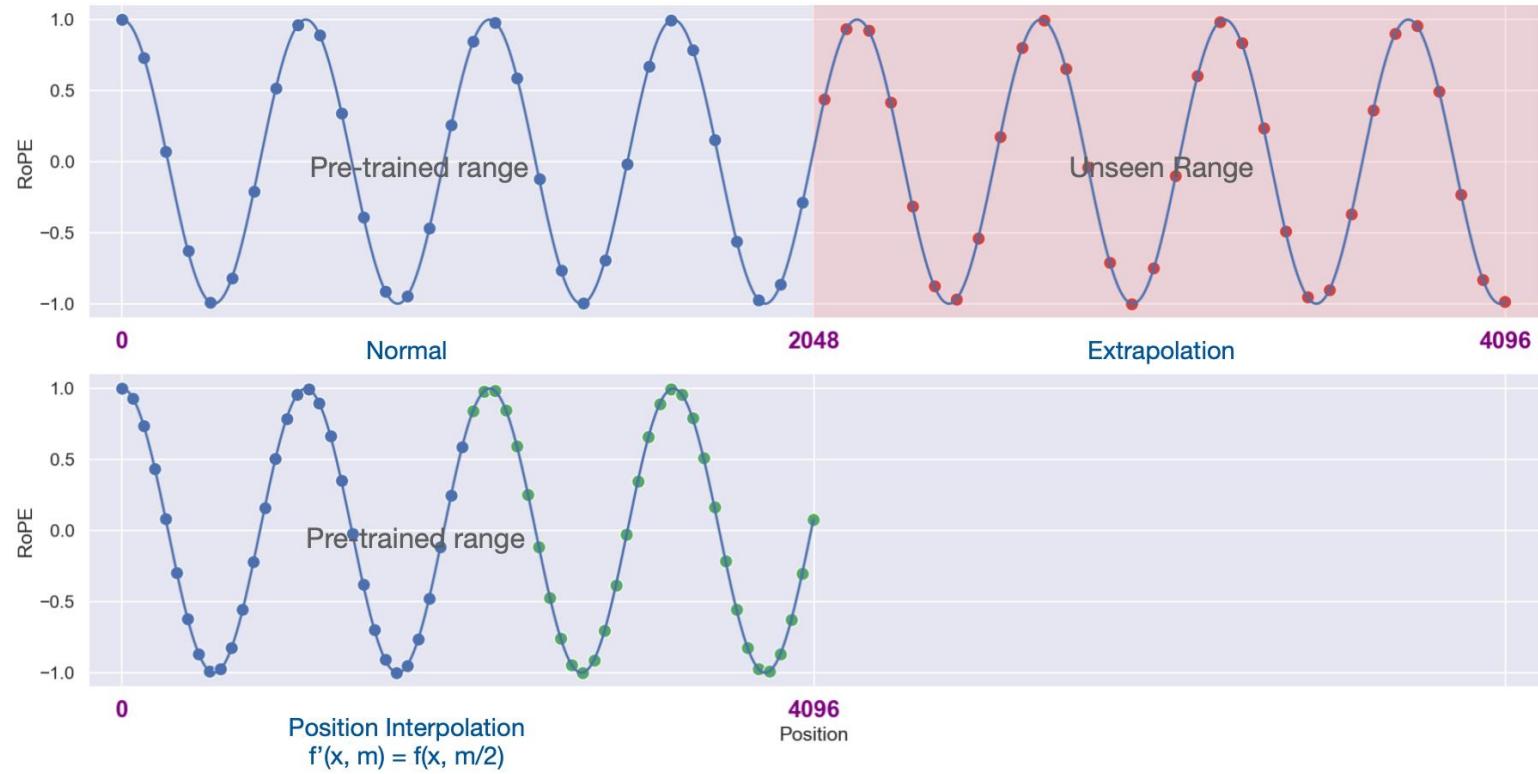
- General form

$$\begin{aligned}f_q(x_m, m) &= (W_q x_m) e^{im\theta} \\f_k(x_n, n) &= (W_k x_n) e^{in\theta} \\g(x_m, x_n, m - n) &= \operatorname{Re} \left[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta} \right]\end{aligned}$$

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}_{\Theta, m}^d \mathbf{W}_{\{q,k\}} \mathbf{x}_m$$
$$\mathbf{R}_{\Theta, m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Rotary Position Encoding

- Allows extension of the context window



Overview

- Architecture
 - Encoder-Decoder
 - Encoder-Only
 - Decoder-Only
- Position Encoding
 - Relative Position Encoding
 - Rotary Position Encoding
- Efficient Attention Mechanism
 - Linear Attention
 - Flash Attention
 - Grouped Query Attention
 - Multi Query Attention
 - Multi-head Latent Attention

Quadratic Complexity

- Self-attention has quadratic complexity to input length

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- $O(L^2d)$ FLOPS

- Many attempts for reducing the quadratic complexity to linear
 - Linear Attention
 - Flash Attention
 - Grouped Query Attention
 - Multi Query Attention
 - Multi-head Latent Attention

Linear Attention

- Modification on Softmax

$$\text{Softmax}(QK^T)V = \frac{\exp(QK^T)}{\sum_{i=1}^L \exp(QK_i^T)}V \longrightarrow \frac{\text{sim}(Q, K)}{\sum_{i=1}^L \text{sim}(Q, K_i)}V$$

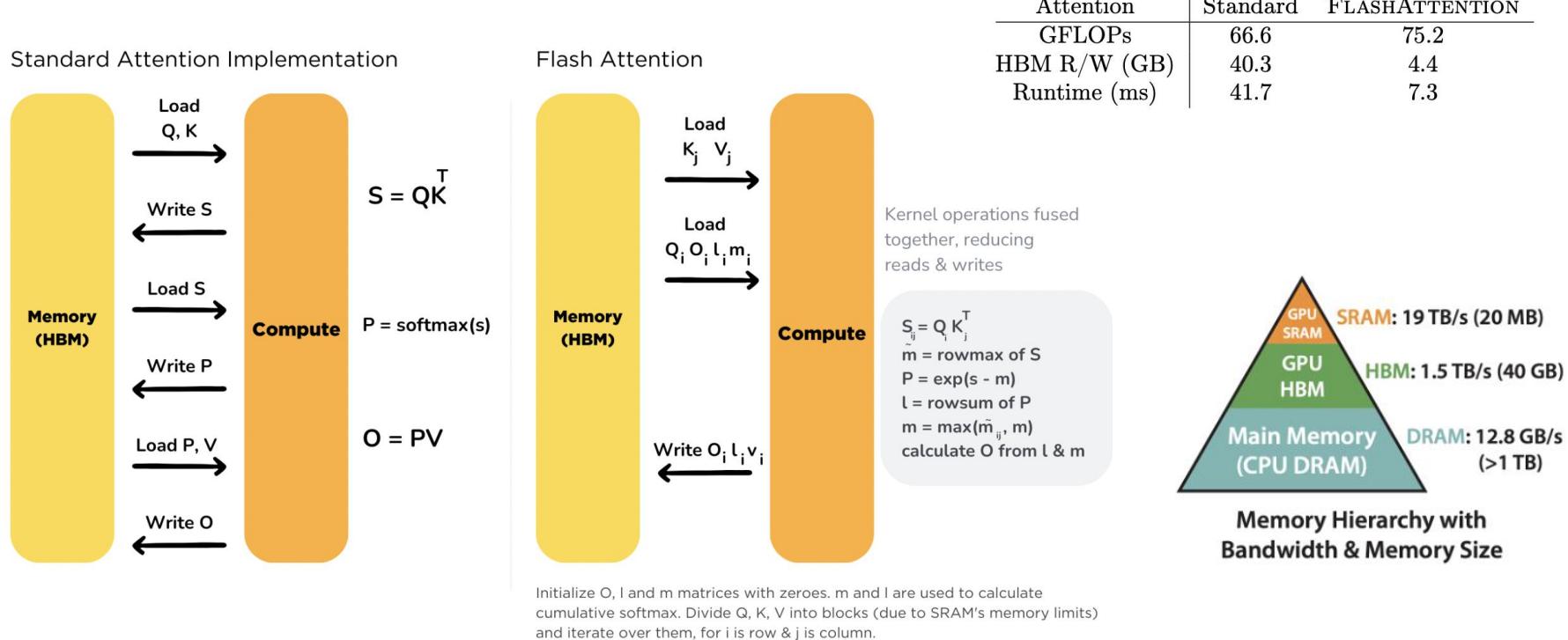
- Kernel function

$$\text{sim}(Q, K) = \phi(Q) \cdot \phi(K) = \phi(Q)\phi(K)^T$$

- Linear form of attention

$$O(L^2) \xrightarrow{\frac{\phi(Q)\phi(K)^T}{\sum_{i=1}^L \phi(Q)\phi(K_i)^T}V} \frac{\phi(Q)(\phi(K)^T V)}{\phi(Q)\sum_{i=1}^L \phi(K_i)^T} \xrightarrow{} O(d'd)$$

Flash Attention



Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." Advances in Neural Information Processing Systems 35 (2022): 16344-16359.

Flash Attention

Without Tiling 32 access

With Tiling 16 access

$$\begin{array}{c|cc}
 \text{A} & \begin{matrix} 1 & 2 \\ 4 & 8 \end{matrix} & \begin{matrix} 0 & 5 \\ 2 & -1 \end{matrix} \\
 \hline
 \begin{matrix} 3 & 1 \\ -7 & 5 \end{matrix} & \begin{matrix} 6 & 3 \\ 0 & 8 \end{matrix} \\
 \hline
 \begin{matrix} 3 & 1 \\ -7 & 5 \end{matrix} & \begin{matrix} 6 & 3 \\ 0 & 8 \end{matrix}
 \end{array}
 \times
 \begin{array}{c|cc}
 \text{B} & \begin{matrix} 3 & 1 \\ -7 & 5 \end{matrix} & \begin{matrix} 6 & 3 \\ 0 & 8 \end{matrix} \\
 \hline
 \begin{matrix} 1 & 2 \\ 0 & 3 \end{matrix} & \begin{matrix} 0 & 5 \\ 5 & 1 \end{matrix} \\
 \hline
 \begin{matrix} 1 & 2 \\ 0 & 3 \end{matrix} & \begin{matrix} 0 & 5 \\ 5 & 1 \end{matrix}
 \end{array}
 =
 \begin{array}{c|cc}
 \text{C} & \begin{matrix} \text{C1,1} \\ \text{C2,1} \end{matrix} & \begin{matrix} \text{C1,2} \\ \text{C2,2} \end{matrix} \\
 \hline
 \begin{matrix} \text{C1,1} \\ \text{C2,1} \end{matrix} & & \\
 \hline
 \begin{matrix} \text{C1,2} \\ \text{C2,2} \end{matrix} & &
 \end{array}$$

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$
 $C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$
 $C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$
 $C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$

Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." Advances in Neural Information Processing Systems 35 (2022): 16344-16359.

Flash Attention

Softmax

Computation requires two loops: one to calculate the normalizing factor (the sum of exponentials) and another to compute the attention weights by dividing each exponentiated value by this factor.

$$\frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$$

Safe Softmax

Requires three loops: one to find the maximum value (for numerical stability), one to compute the normalizing factor, and one to obtain the attention weights.

```
for i ← 1, N do
     $m_i \leftarrow \max(m_{i-1}, x_i)$ 
for i ← 1, N do
     $d_i \leftarrow d_{i-1} + e^{x_i - m_N}$ 
for i ← 1, N do
     $a_i \leftarrow \frac{e^{x_i - m_N}}{d_N}$ 
```

Online Softmax

Requires two loops: one to find the maximum value (and to compute the normalizing factor, and one to obtain the attention weights.

```
for i ← 1, N do
     $m_i \leftarrow \max(m_{i-1}, x_i)$ 
     $d'_i \leftarrow d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i}$ 
for i ← 1, N do
     $a_i \leftarrow \frac{e^{x_i - m_N}}{d'_N}$ 
```

Flash Attention

Fused computation to one loop!

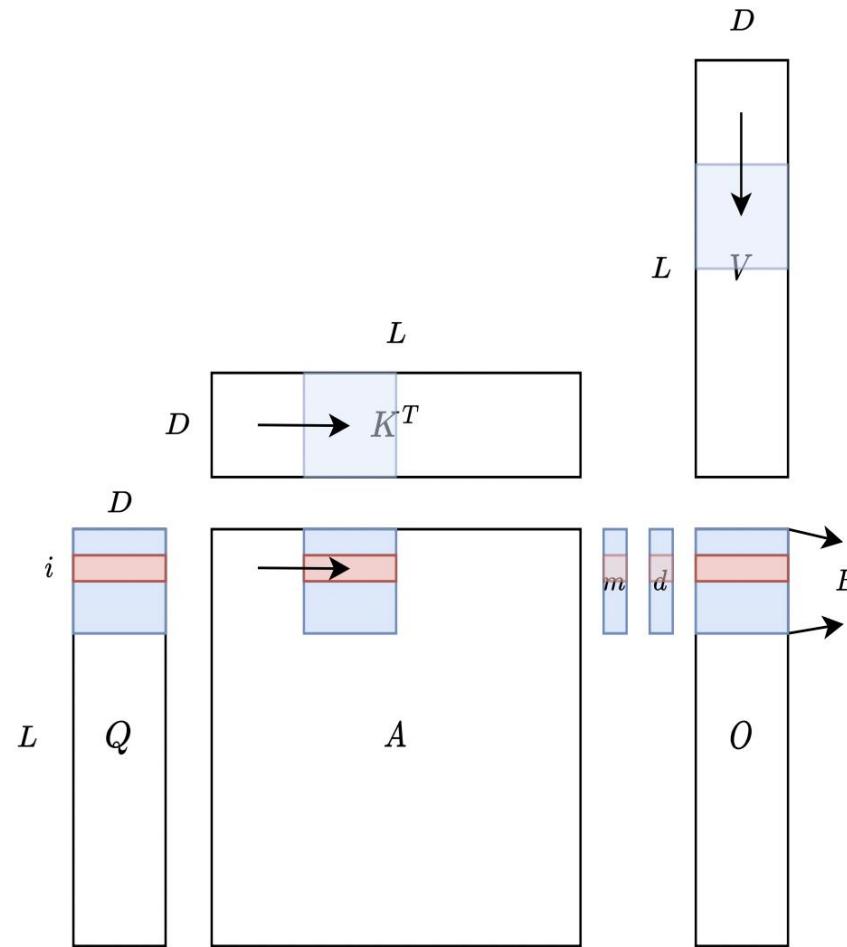
for $i \leftarrow 1, N$ **do**

$$\begin{aligned}x_i &\leftarrow Q[k, :] K^T[:, i] \\m_i &\leftarrow \max(m_{i-1}, x_i) \\d'_i &\leftarrow d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i} \\o'_i &\leftarrow o'_{i-1} \frac{d'_{i-1} e^{m_{i-1} - m_i}}{d'_i} + \frac{e^{x_i - m_i}}{d'_i} V[i, :]\end{aligned}$$

end

$$O[k, :] \leftarrow o'_N$$

Flash Attention



Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." Advances in Neural Information Processing Systems 35 (2022): 16344-16359.

Flash Attention

Algorithm 1 FLASHATTENTION

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M .

- 1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil, B_r = \min(\lceil \frac{M}{4d} \rceil, d)$.
 - 2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
 - 3: Divide \mathbf{Q} into $T_r = \lceil \frac{N}{B_r} \rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide \mathbf{K}, \mathbf{V} in to $T_c = \lceil \frac{N}{B_c} \rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
 - 4: Divide \mathbf{O} into T_r blocks $\mathbf{O}_1, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_1, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks m_1, \dots, m_{T_r} of size B_r each.
 - 5: **for** $1 \leq j \leq T_c$ **do**
 - 6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
 - 7: **for** $1 \leq i \leq T_r$ **do**
 - 8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
 - 9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
 - 10: On chip, compute $\tilde{m}_{ij} = \text{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}, \tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
 - 11: On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}, \ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
 - 12: Write $\mathbf{O}_i \leftarrow \text{diag}(\ell_i^{\text{new}})^{-1} (\text{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
 - 13: Write $\ell_i \leftarrow \ell_i^{\text{new}}, m_i \leftarrow m_i^{\text{new}}$ to HBM.
 - 14: **end for**
 - 15: **end for**
 - 16: Return \mathbf{O} .
-

Flash Attention

Models	ListOps	Text	Retrieval	Image	Pathfinder	Avg	Speedup
Transformer	36.0	63.6	81.6	42.3	72.7	59.3	-
FLASHATTENTION	37.6	63.9	81.4	43.5	72.7	59.8	2.4×
Block-sparse FLASHATTENTION	37.0	63.0	81.3	43.6	73.3	59.6	2.8×
Linformer [84]	35.6	55.9	77.7	37.8	67.6	54.9	2.5×
Linear Attention [50]	38.8	63.2	80.7	42.6	72.5	59.6	2.3×
Performer [12]	36.8	63.6	82.2	42.1	69.9	58.9	1.8×
Local Attention [80]	36.1	60.2	76.7	40.6	66.6	56.0	1.7×
Reformer [51]	36.5	63.8	78.5	39.6	69.4	57.6	1.3×
Smyrf [19]	36.1	64.1	79.0	39.6	70.5	57.9	1.7×

IO complexity:

Flash Attention: $\mathbf{O}\left(\frac{N^2 d^2}{M}\right)$

Standard Attention: $\Omega(Nd + N^2)$

Where **N** is sequence length, **d** head dimensions and **M** the size of SRAM.

KV- Caching

Step 1

Without Cache

$$\begin{array}{ccc}
 Q & K^T & QK^T \\
 \text{Query Token 1} & \times \text{Key Token 1} & = \quad \boxed{Q_i, K_i} \\
 (1, \text{emb_size}) & (\text{emb_size}, 1) & (1, 1)
 \end{array}
 \quad
 \begin{array}{ccc}
 V & \text{Attention} \\
 \text{Value Token 1} & \times \quad \text{Token 1} \\
 (1, \text{emb_size}) & = & (1, \text{emb_size})
 \end{array}$$

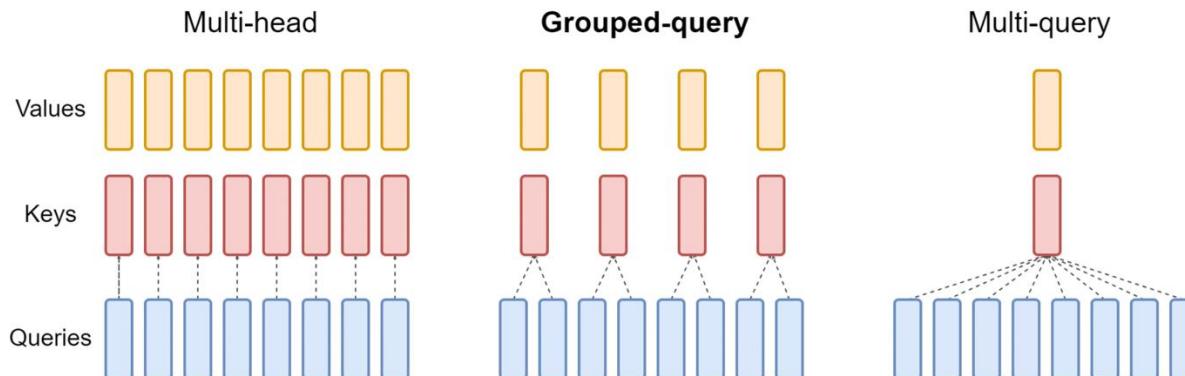
With cache

$$\begin{array}{ccc}
 Q & K^T & QK^T \\
 \text{Query Token 1} & \times \text{Key Token 1} & = \quad \boxed{Q_i, K_i} \\
 (1, \text{emb_size}) & (\text{emb_size}, 1) & (1, 1)
 \end{array}
 \quad
 \begin{array}{ccc}
 V & \text{Attention} \\
 \text{Value Token 1} & \times \quad \text{Token 1} \\
 (1, \text{emb_size}) & = & (1, \text{emb_size})
 \end{array}$$

□ Values that will be masked ■ Values that will be taken from cache

Multi and Grouped Query Attention

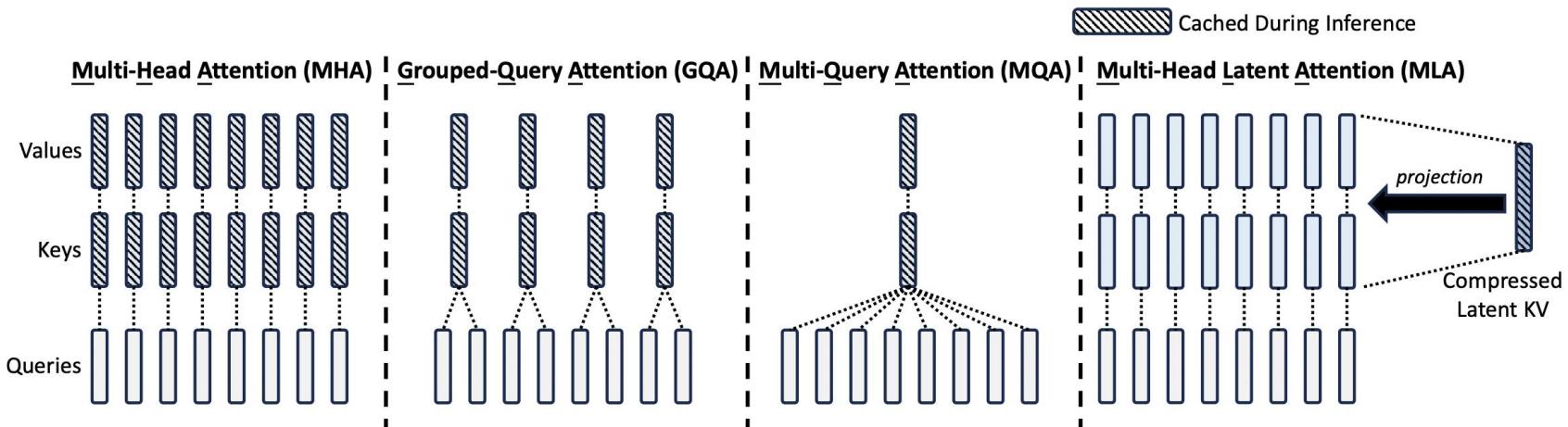
- Multi-head attention has H query, key, and value heads.
- Multi-query attention shares single key and value heads across all query heads.
- Grouped-query attention instead shares single key and value heads for each group of query heads.



Benchmark (Metric)	# Shots	Dense 7B w/ MQA	Dense 7B w/ GQA (8 Groups)	Dense 7B w/ MHA
# Params	-	7.1B	6.9B	6.9B
BBH (EM)	3-shot	33.2	35.6	37.0
MMLU (Acc.)	5-shot	37.9	41.2	45.2
C-Eval (Acc.)	5-shot	30.0	37.7	42.9
CMMLU (Acc.)	5-shot	34.6	38.4	43.5

Multihead Latent Attention

- Low-rank key-value joint compression
- Caching compressed latent KV pairs during inference



Attention Mechanism	KV Cache per Token (# Element)	Capability
Multi-Head Attention (MHA)	$2n_h d_h l$	Strong
Grouped-Query Attention (GQA)	$2n_g d_h l$	Moderate
Multi-Query Attention (MQA)	$2d_h l$	Weak
MLA (Ours)	$(d_c + d_h^R)l \approx \frac{9}{2}d_h l$	Stronger

Benchmark (Metric)	# Shots	Small MoE w/ MHA	Small MoE w/ MLA	Large MoE w/ MHA	Large MoE w/ MLA
# Activated Params	-	2.5B	2.4B	25.0B	21.5B
# Total Params	-	15.8B	15.7B	250.8B	247.4B
KV Cache per Token (# Element)	-	110.6K	15.6K	860.2K	34.6K
BBH (EM)	3-shot	37.9	39.0	46.6	50.7
MMLU (Acc.)	5-shot	48.7	50.0	57.5	59.0
C-Eval (Acc.)	5-shot	51.6	50.9	57.9	59.2
CMMLU (Acc.)	5-shot	52.3	53.4	60.7	62.5

Poll @968

Which of the following statements is true?

- FlashAttention is particularly effective for long sequences, as it stores the full attention matrix in memory, which would otherwise grow quadratically with sequence length due to a higher number of memory accesses.
- FlashAttention improves efficiency by splitting computations into blocks that fit in fast SRAM, reducing memory access overhead while maintaining mathematical equivalence to standard attention.
- FlashAttention performs worse than standard attention implementations because the block-wise computation approach introduces additional computational overhead that outweighs any memory benefits.
- FlashAttention is primarily designed for CPU optimization and shows minimal performance improvements when implemented on GPU hardware.

Poll @968

Which of the following statements is true?

- FlashAttention is particularly effective for long sequences, as it stores the full attention matrix in memory, which would otherwise grow quadratically with sequence length due to a higher number of memory accesses.
- FlashAttention improves efficiency by splitting computations into blocks that fit in fast SRAM, reducing memory access overhead while maintaining mathematical equivalence to standard attention.
- FlashAttention performs worse than standard attention implementations because the block-wise computation approach introduces additional computational overhead that outweighs any memory benefits.
- FlashAttention is primarily designed for CPU optimization and shows minimal performance improvements when implemented on GPU hardware.

Content

- Transformer Architecture
- Improvements on Transformers
- **Transformer for different modalities**
- Parameter Efficient Tuning
- Scaling Laws

Transformer in Vision and Audio

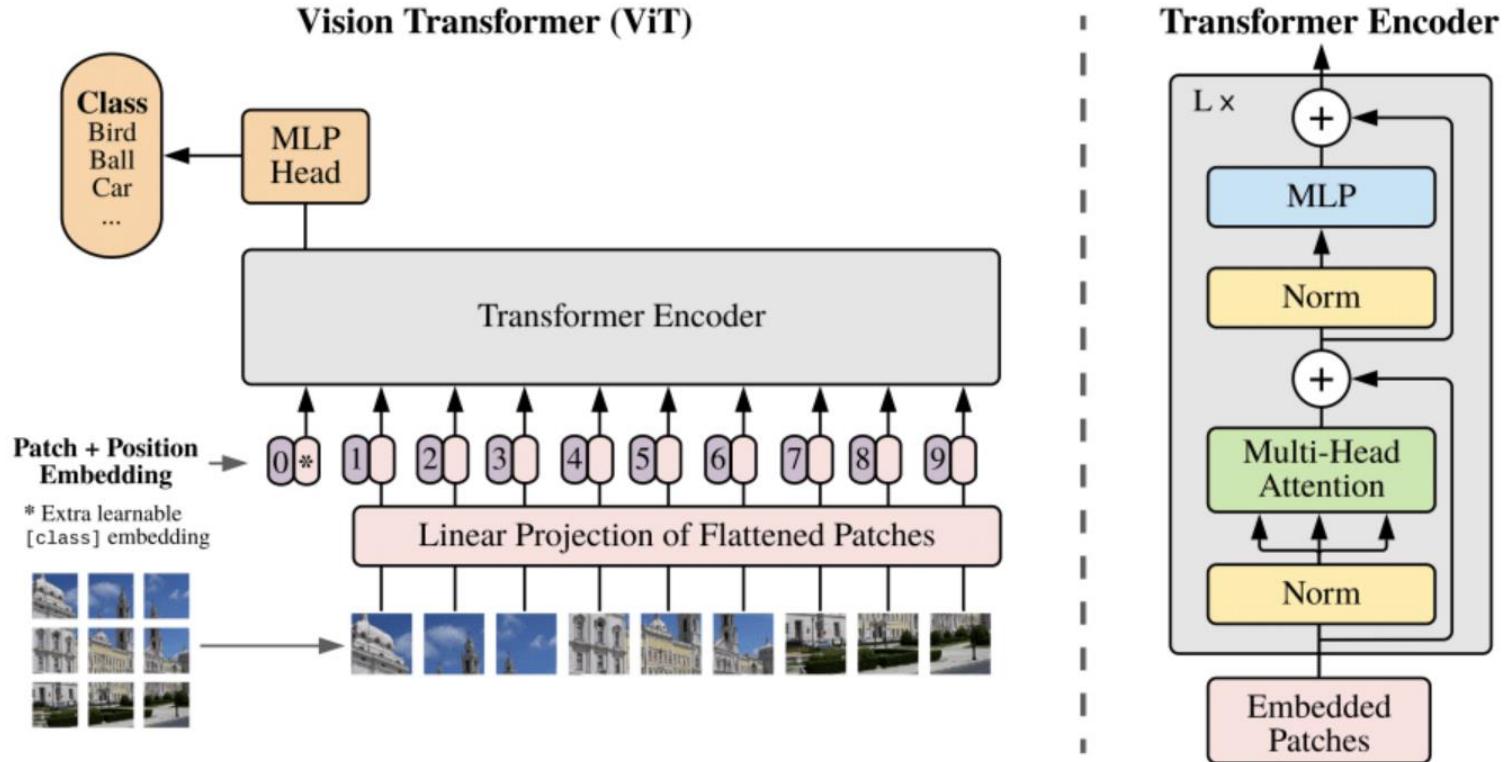
Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizer

Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizers

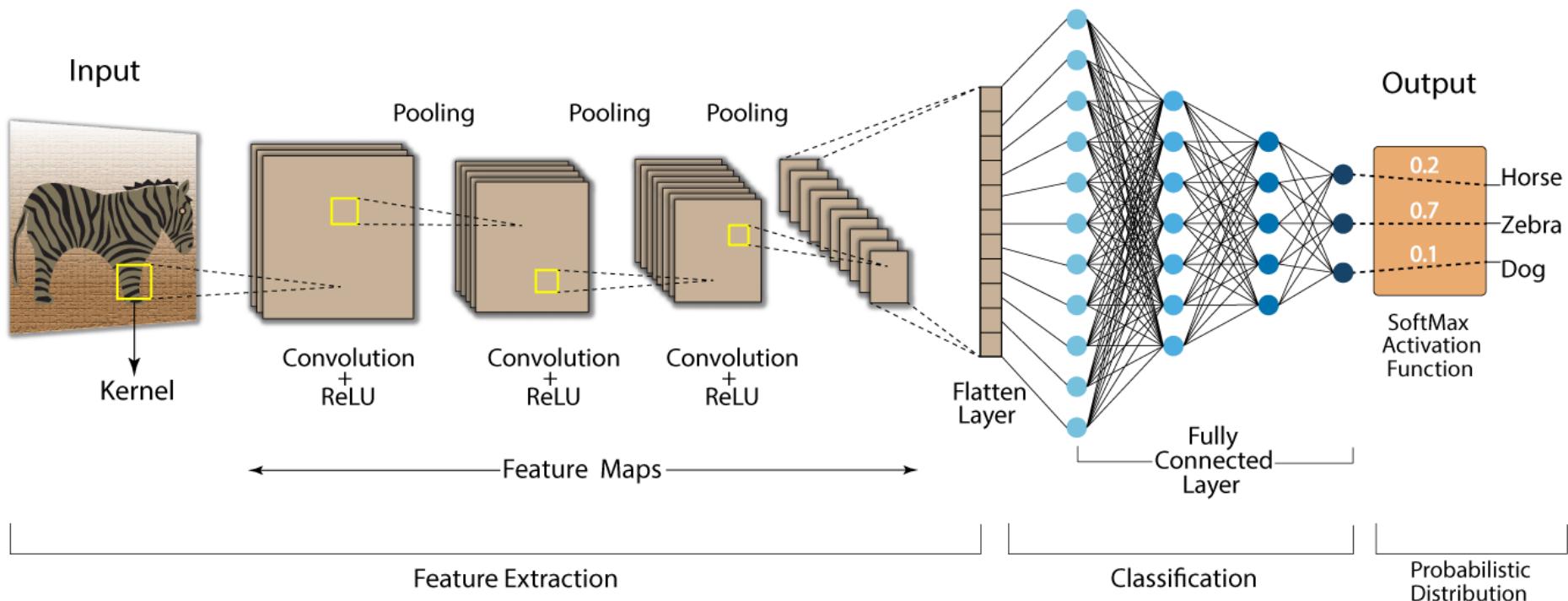
Vision Transformer (ViT)



- Transformer architecture can also be used for images
- How do we process an image into tokens?

CNN

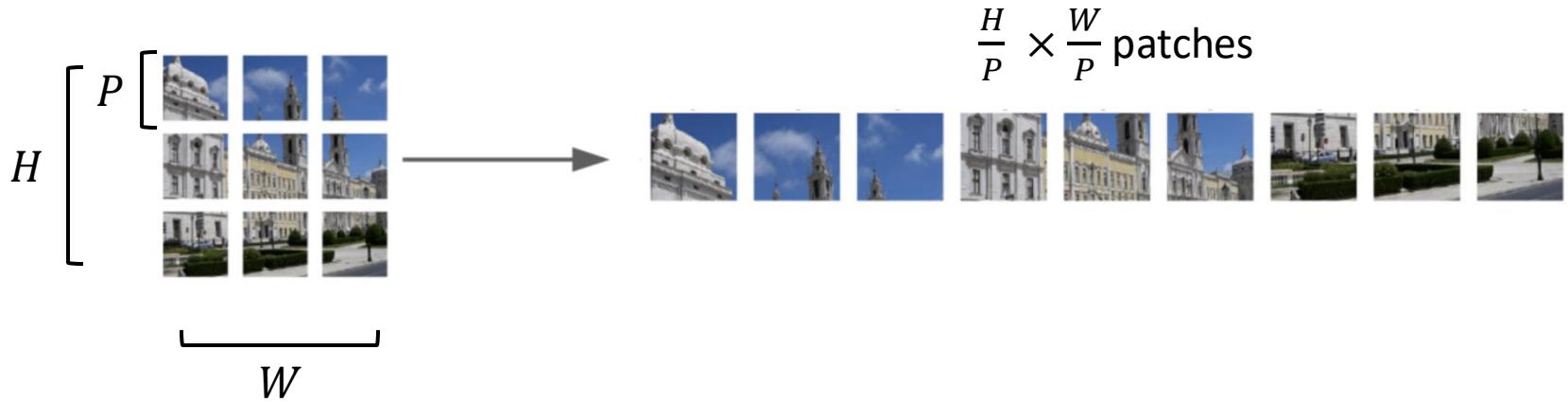
Convolution Neural Network (CNN)



- Naturally fits to 2D images

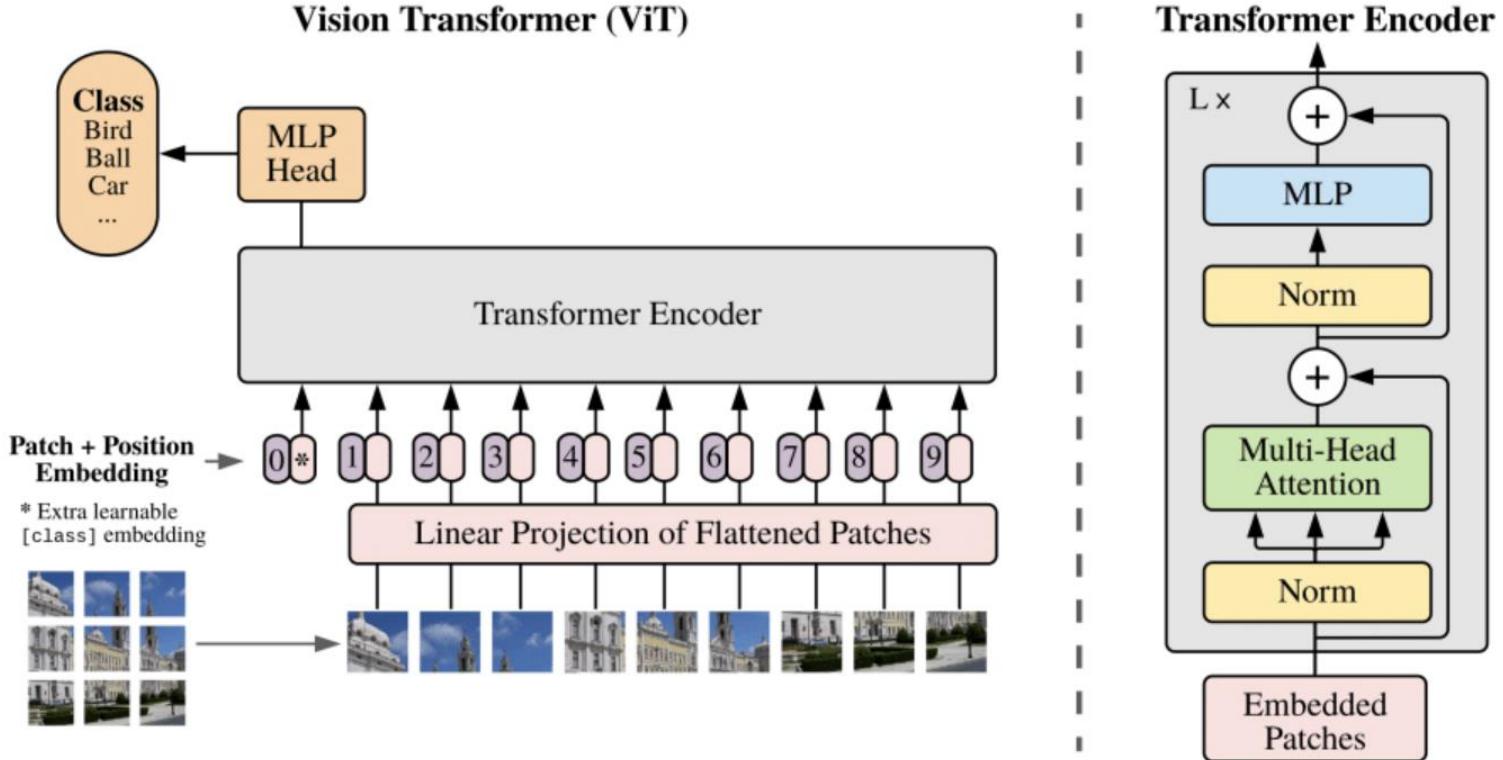
ViT

- Split images into a sequence of **patches**



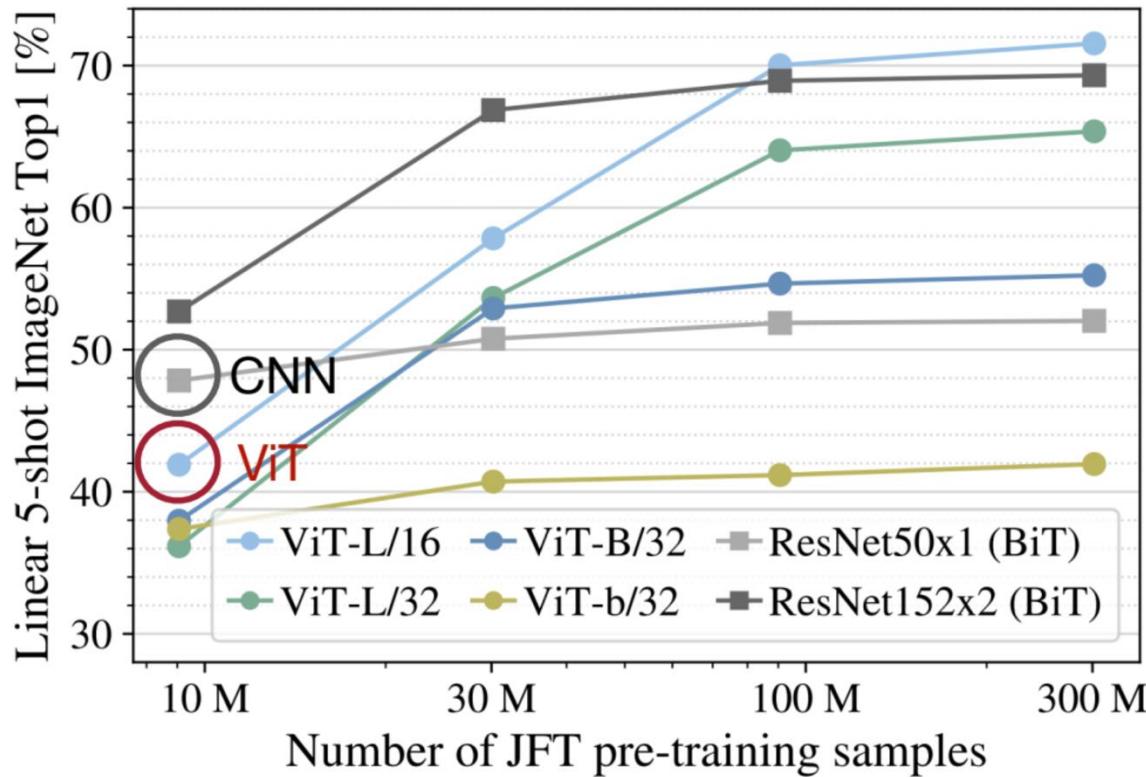
- Each patch is treated as one token as input to ViT
 - A convolution layer with kernel P and stride P!
 - Or a linear layer on the flatten pixels

ViT



- The remaining is same as Transformer
 - As an encoder-only model

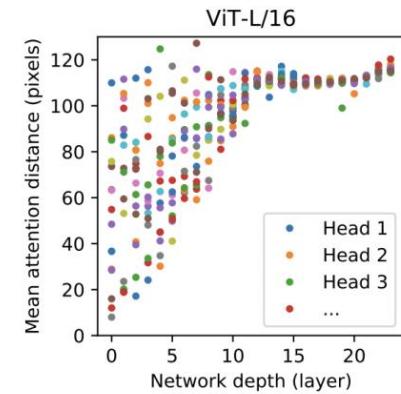
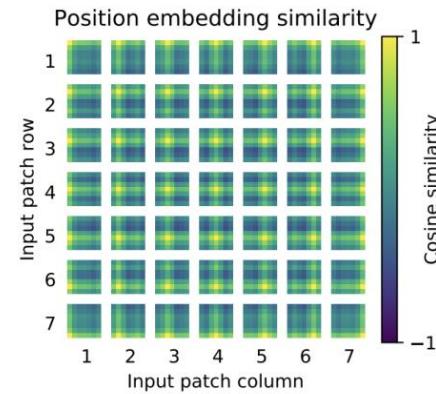
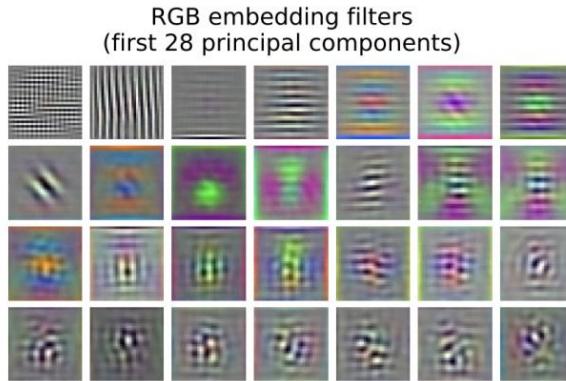
Image Classification



- Inferior performance compared to CNN when dataset size is limited – Why?

Inductive Bias

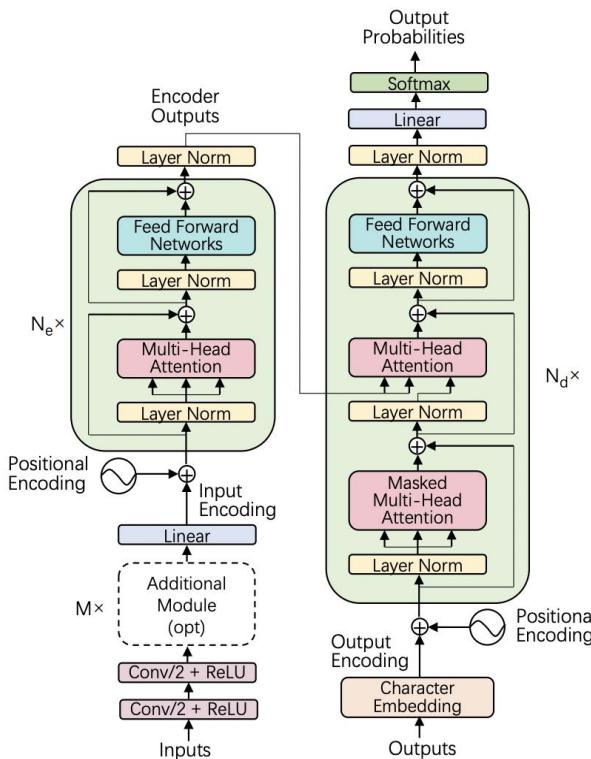
- Convolutional Neural Networks
 - Locality
 - Sharing weights
- Vision Transformer
 - None!
 - Has to learn locality and dependency from data!
 - A lot lot lot lot lot lot lot lot of data!



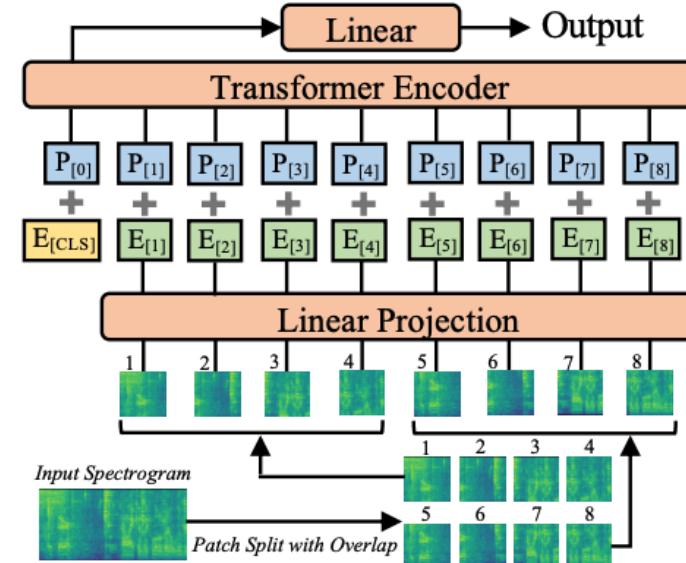
Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizer

Transformer in Audio



Speech Transformer for ASR

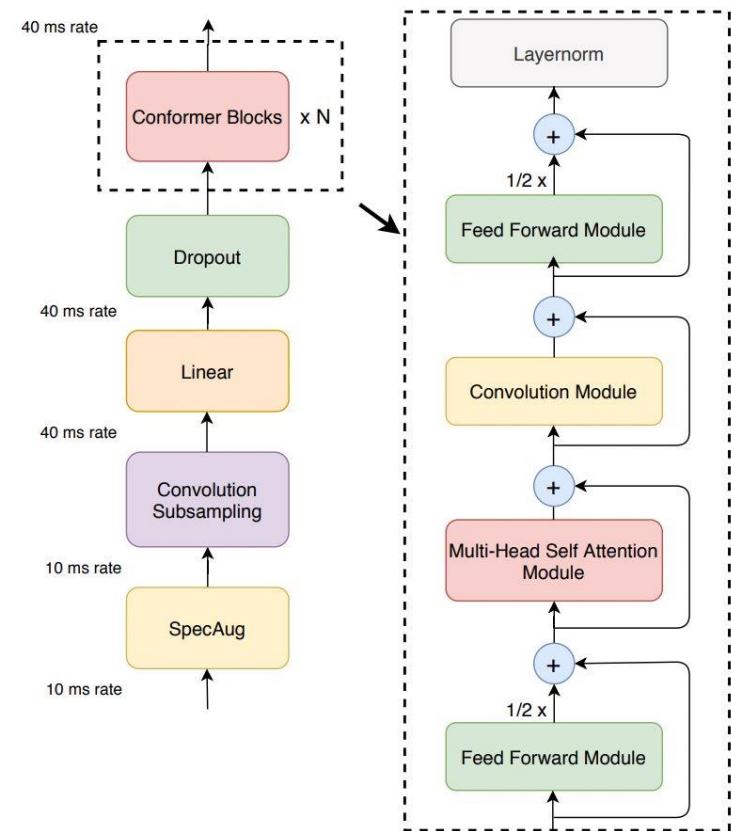


Audio Spectrogram Transformer

- [1] Dong, Linhao, Shuang Xu, and Bo Xu. "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition." *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018.
- [2] Gong, Yuan, Yu-An Chung, and James Glass. "Ast: Audio spectrogram transformer." *arXiv preprint arXiv:2104.01778* (2021).

Conformer

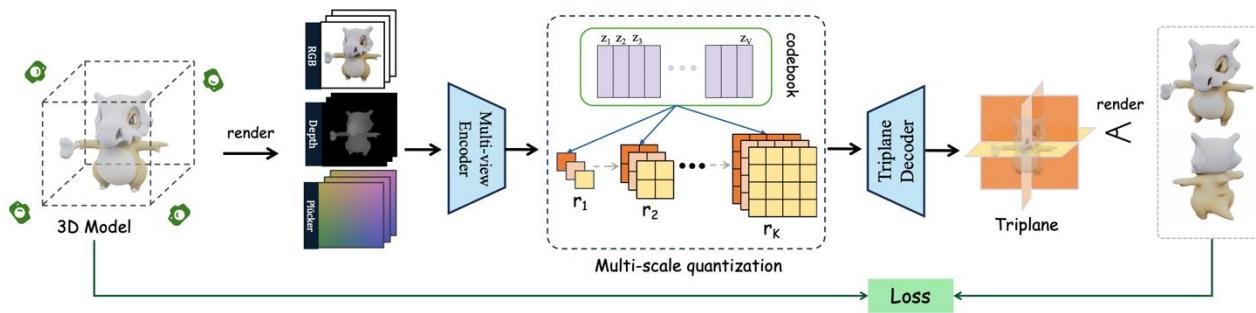
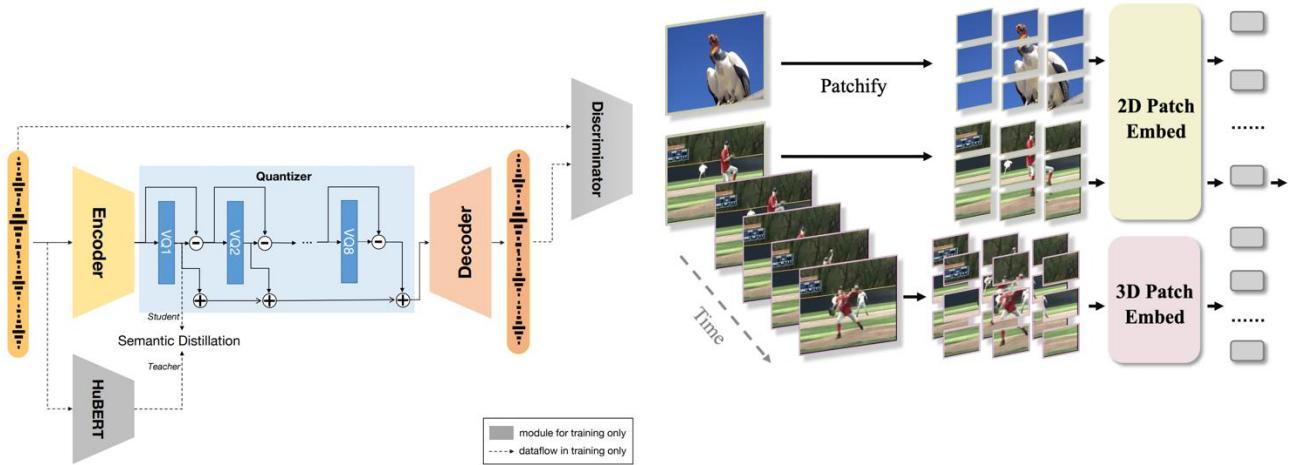
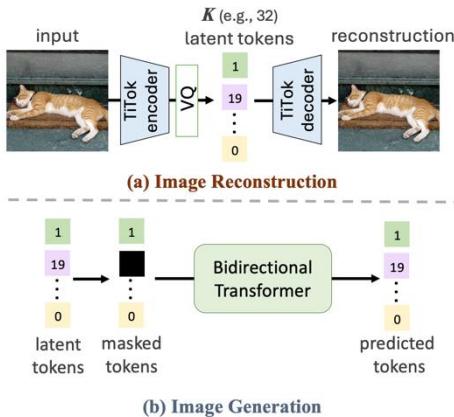
- The Conformer architecture augments a transformer by embedding convolution layers within the transformer blocks.
- Transformers capture global dependencies, CNNs capture local features efficiently.



Overview

- Vision Transformer Architecture
- Transformer in Audio: Conformer
- Tokenizer

Tokenizers



Zhang, Xin, et al. "Speechtokenizer: Unified speech tokenizer for speech language models." The Twelfth International Conference on Learning Representations. 2024.

Chen, Yongwei, et al. "SAR3D: Autoregressive 3D object generation and understanding via multi-scale 3D VQVAE." arXiv preprint arXiv:2411.16856 (2024).

Yu, Qihang, et al. "An Image is Worth 32 Tokens for Reconstruction and Generation." arXiv preprint arXiv:2406.07550 (2024).

Wang, Junke, et al. "OmniTokenizer: A Joint Image-Video Tokenizer for Visual Generation." arXiv preprint arXiv:2406.09399 (2024).

Poll @965 and @966

Which ones of the following are properties of ViT, compared to CNN?

- Weight sharing
- Dynamic weights from data
- Locality
- Global dependency from data

Which of the following statements about the Conformer architecture is correct?

- The Conformer uses convolution layers to replace self-attention entirely
- Conformer blocks have convolutional modules placed after the self-attention module
- The Conformer architecture eliminates the need for Feed Forward modules
- Conformer was primarily designed for computer vision tasks rather than speech recognition

Poll @965 and @966

Which ones of the following are properties of ViT, compared to CNN?

- Weight sharing
- Dynamic weights from data
- Locality
- Global dependency from data

Which of the following statements about the Conformer architecture is correct?

- The Conformer uses convolution layers to replace self-attention entirely
- Conformer blocks have convolutional modules placed after the self-attention module
- The Conformer architecture eliminates the need for Feed Forward modules
- Conformer was primarily designed for computer vision tasks rather than speech recognition

Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- **Parameter Efficient Tuning**
- Scaling Laws

Parameter Efficient Tuning

Overview

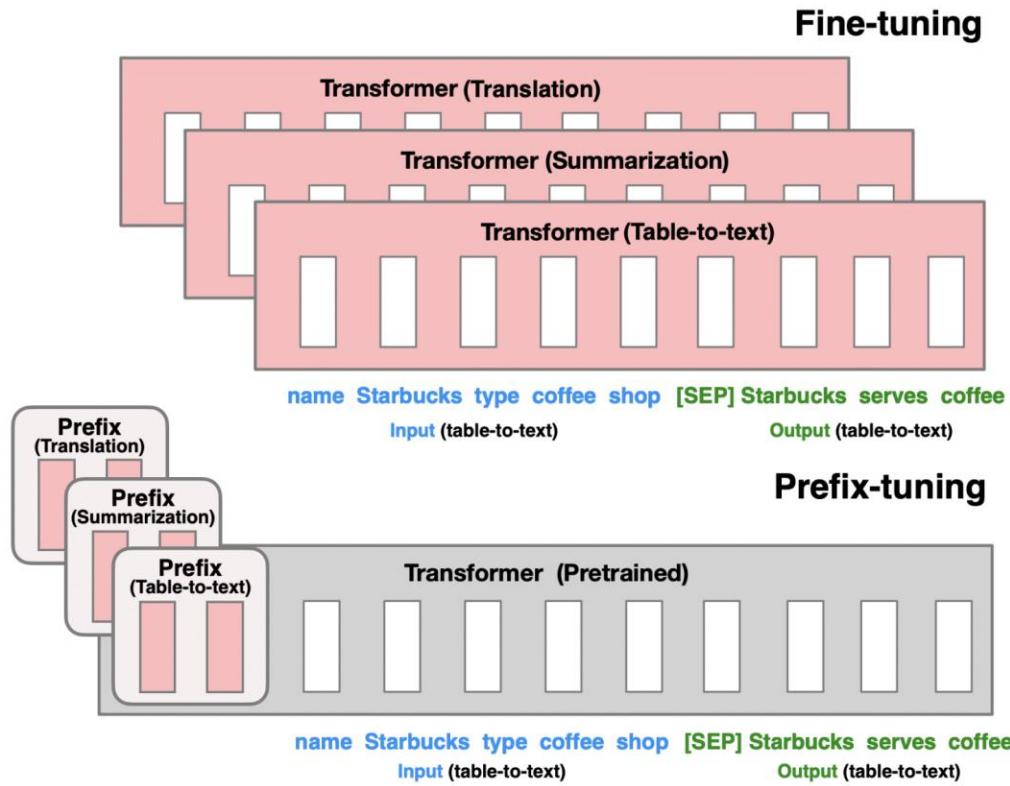
- Parameter Efficient Tuning Methods
 - Prefix Tuning
 - Prompt Tuning
 - Adapter
 - LoRA
- Interpretation

Parameter Efficient Tuning

- Traditionally, you need to fine-tune entire network on specific downstream tasks
- Parameter Efficient Tuning – Only tune a small proportion of parameters of the pre-trained transformer
 - Prefix Tuning
 - Prompt tuning
 - Adapter
 - LoRA

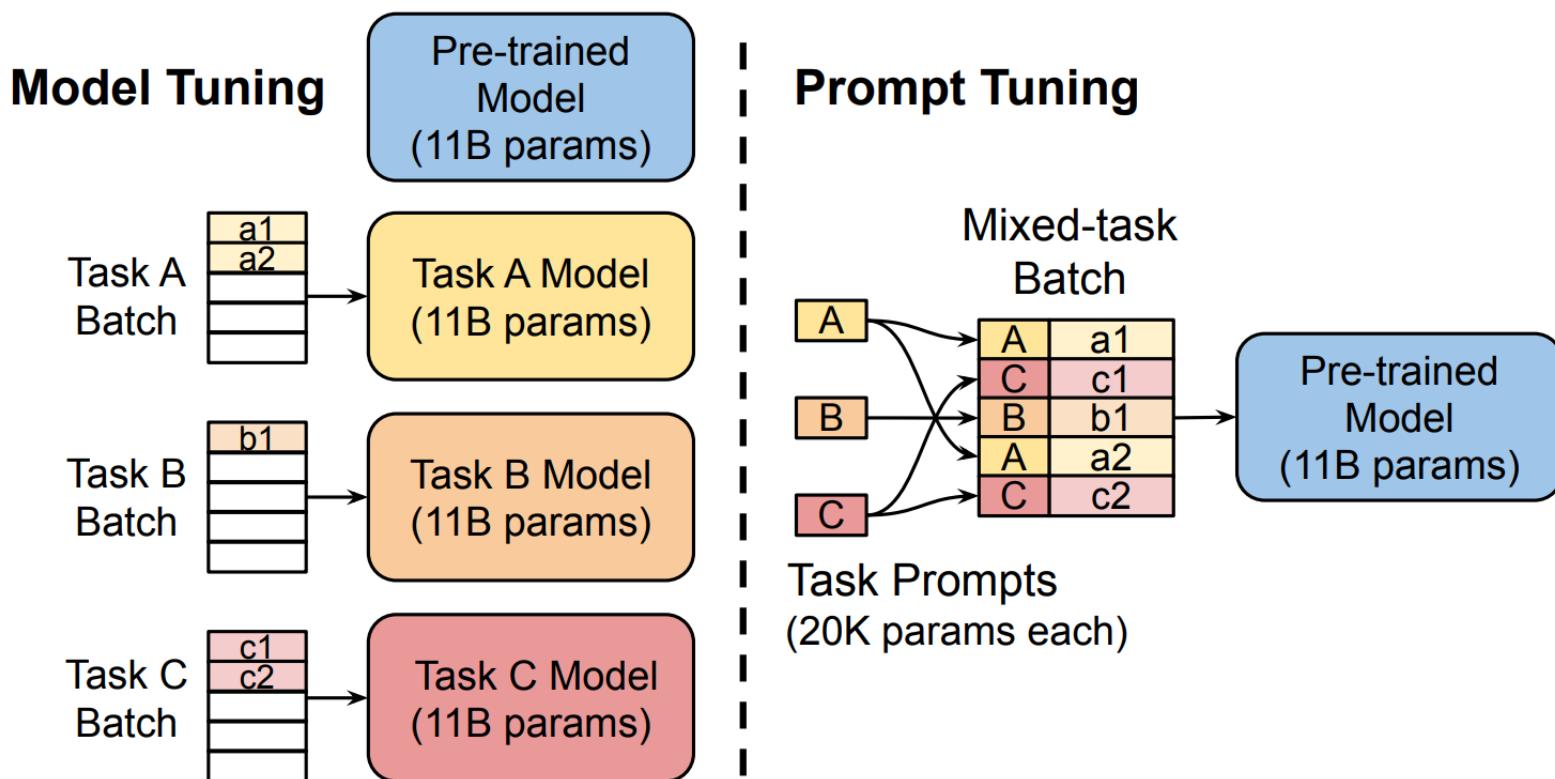
Prefix Tuning

- Only learns a set continuous prefixes)added to the input and transformer layers for each task.



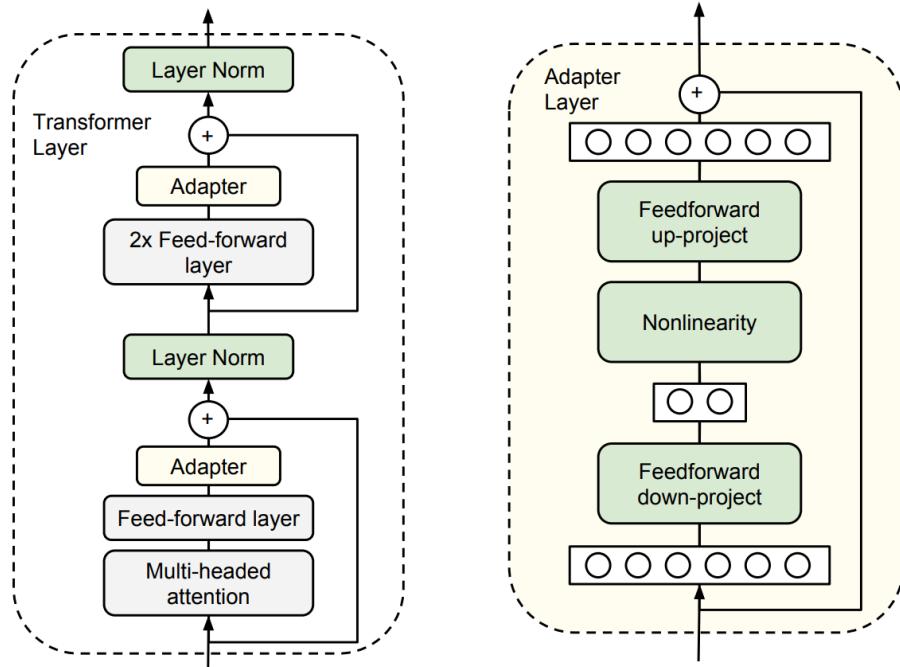
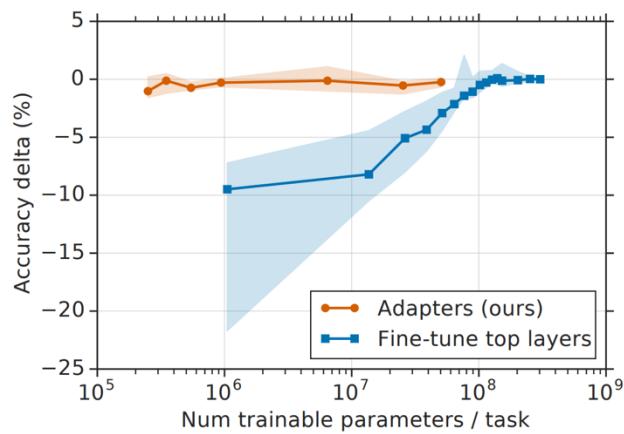
Prompt Tuning

- Only learns a set of ‘prompt’ or ‘token’ for each task



Adapter

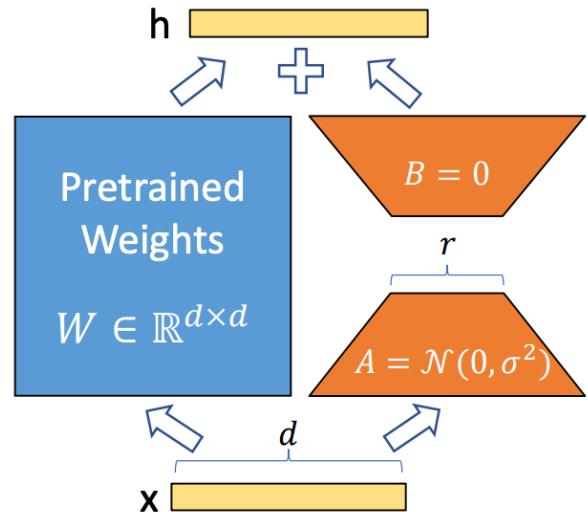
- Insert MLP at Feed-forward layers



LoRA

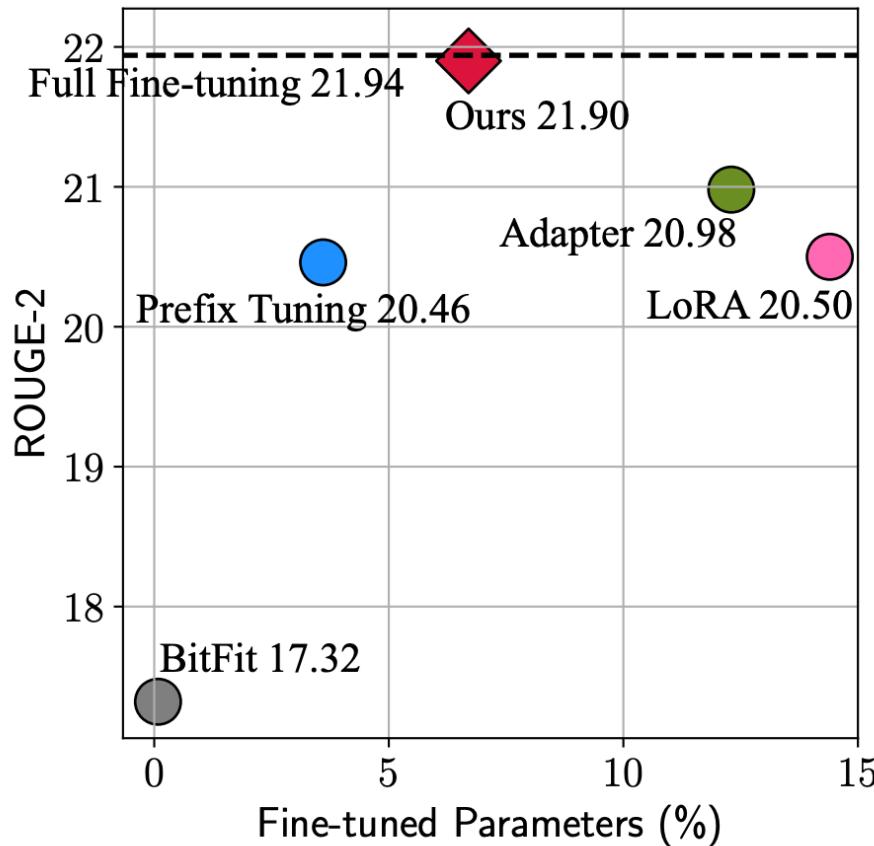
- Low-rank Adaptation (LoRA)
- No activation in-between
- A and B can be fused into W

$$h = W_0x + \Delta Wx = W_0x + BAx$$



Parameter-Efficient Tuning

- Performance close to full fine-tuning while just train less than 15% of original parameters

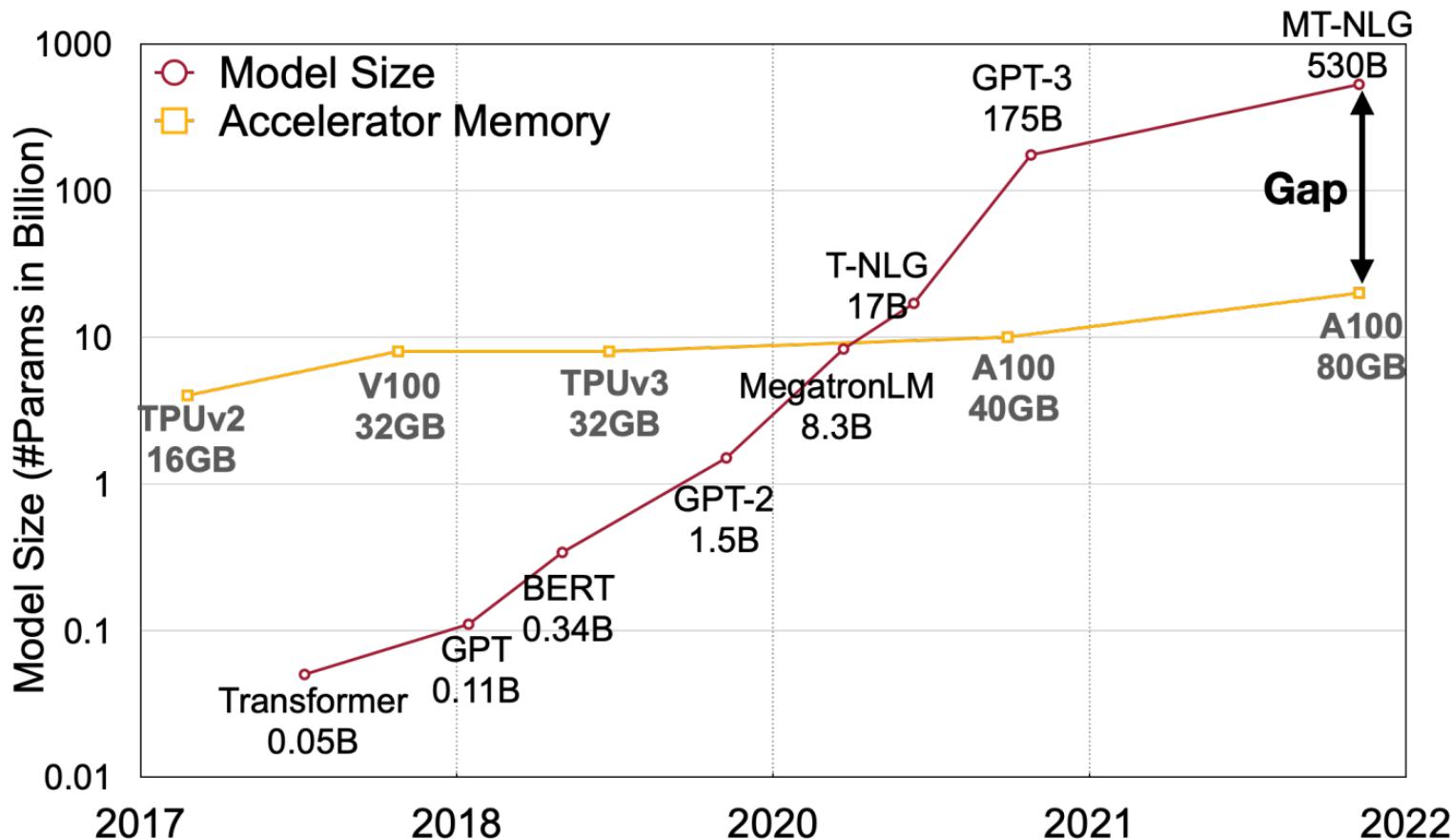


Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- **Scaling Laws**

Scaling Laws

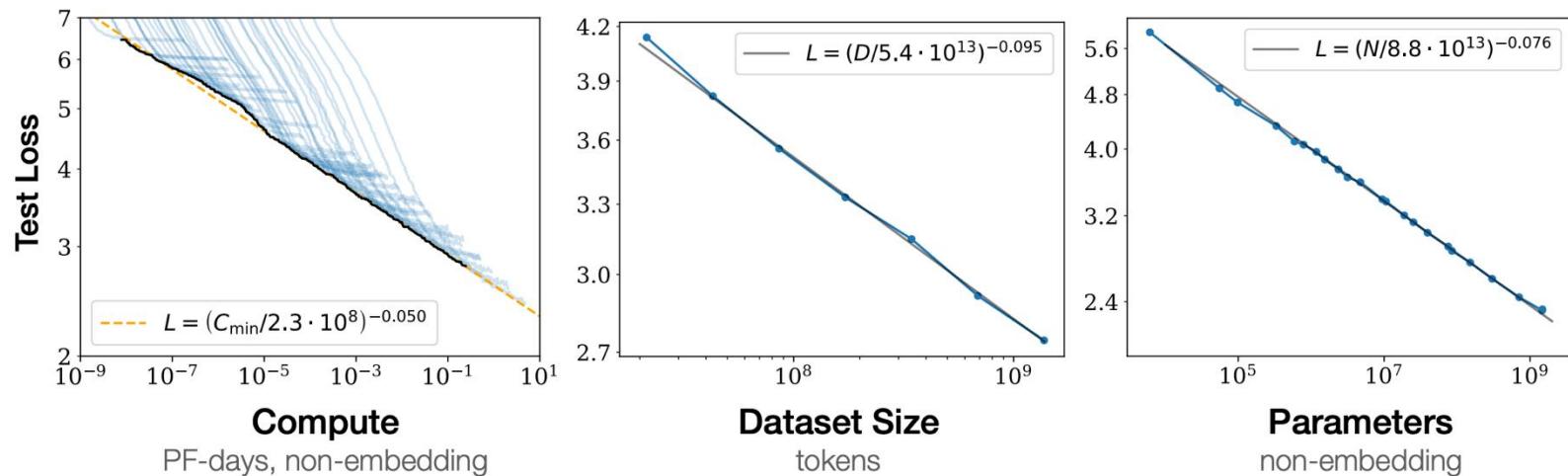
“Magic” of Transformer - Scaling



- Performance gets better as transformer scales up

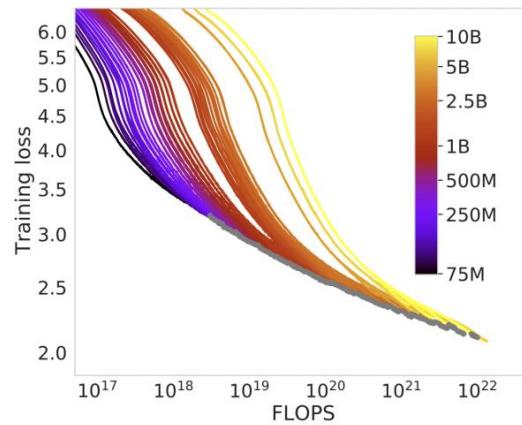
Scaling Law

- For decoder-only models, the final performance is only related to **Compute**, **Data Size**, and **Parameter Size**
 - power law relationship for each factor
 - w/o constraints by the others

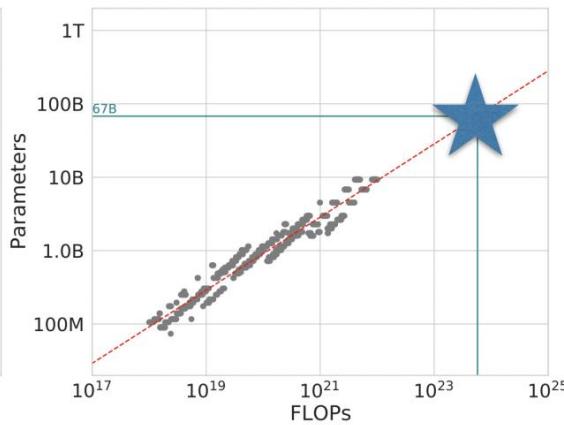


$$\text{PF-day} = 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19} \text{ floating point operations.}$$

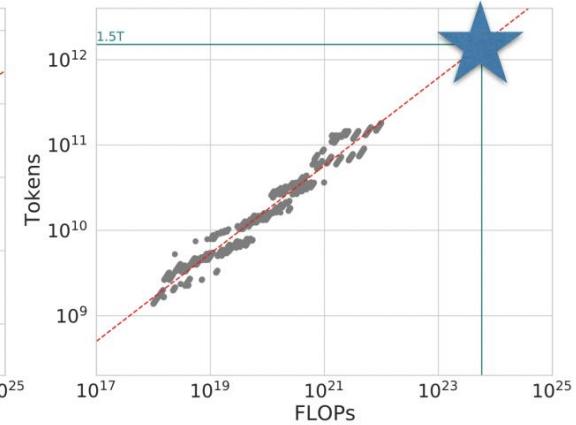
Scaling Law



Run experiments

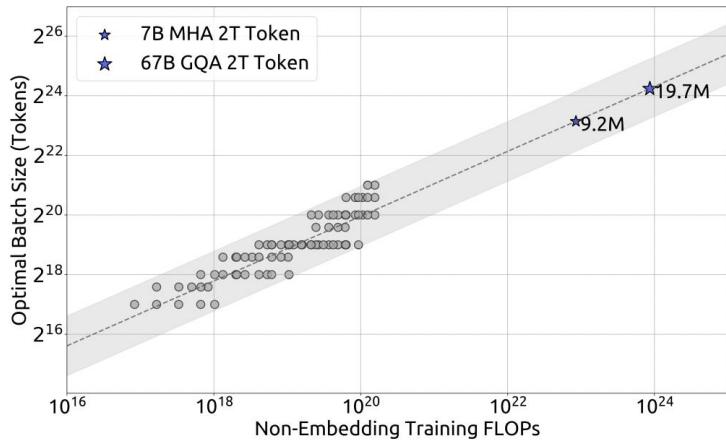


Fit a line and
predict optimal
model size

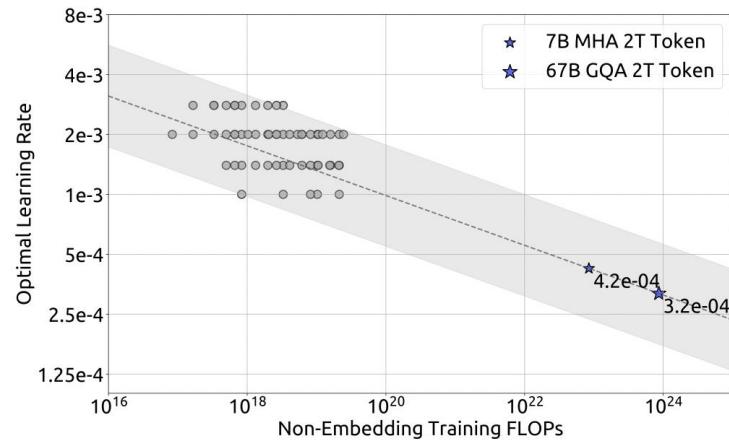


Fit a line and
predict optimal
of tokens

Scaling Law

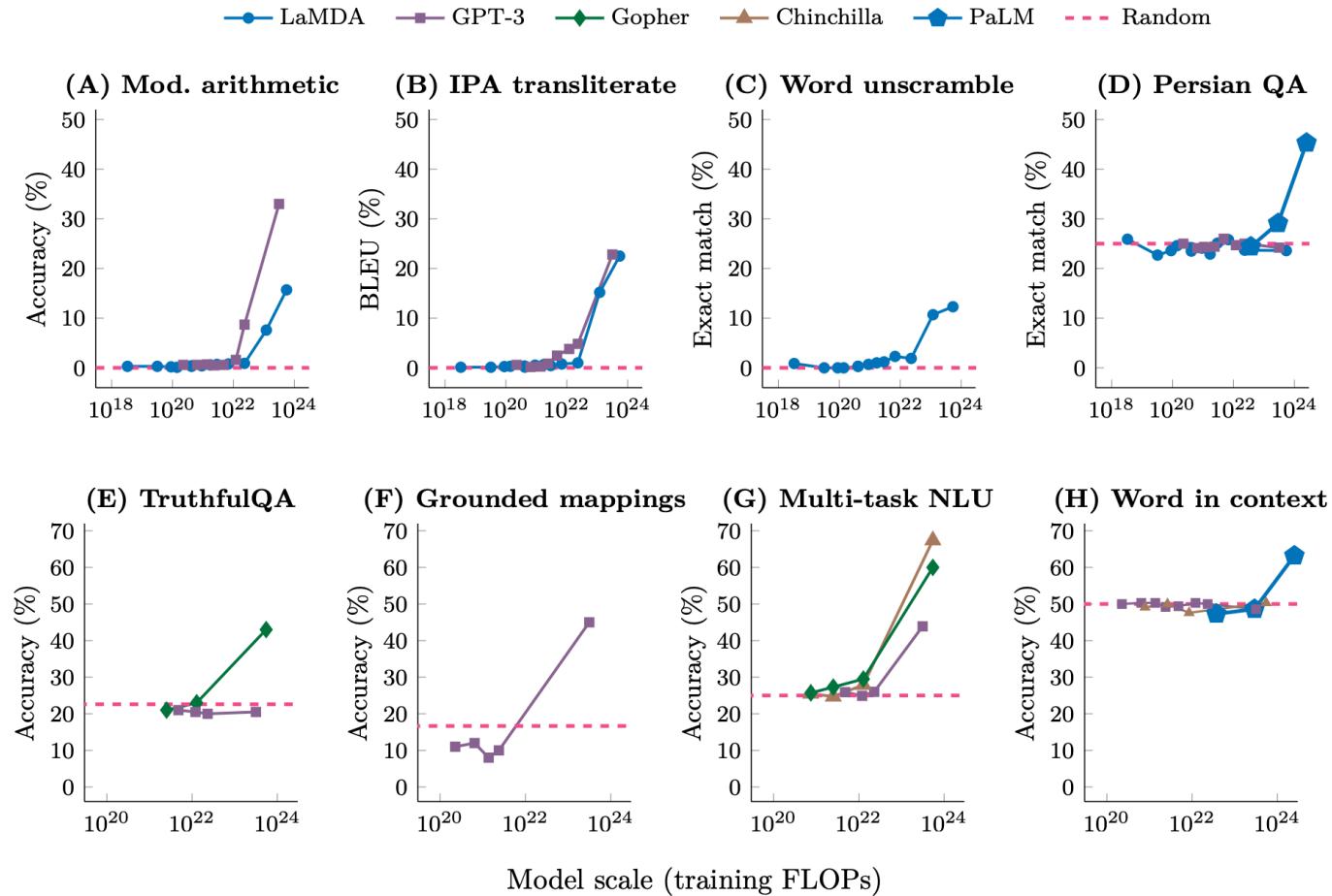


(a) Batch size scaling curve



(b) Learning rate scaling curve

“Emergent” Capability



In-Context Learning

- Scaled models can generalize to new tasks without fine-tuning!
 - Zero-shot
 - Few-shot

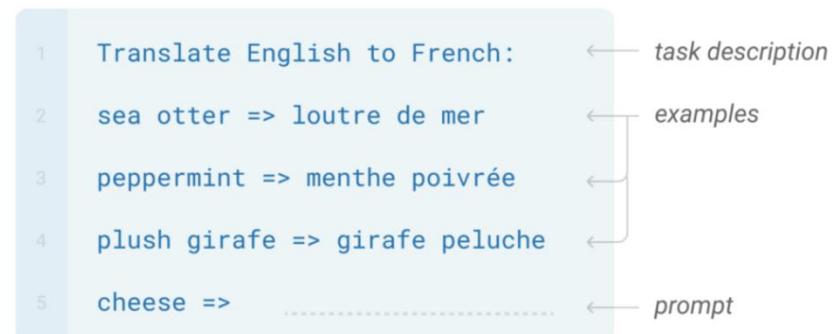
Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



We learned...

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Tokenizers
- Parameter Efficient Tuning
- Scaling Laws