Deep Learning
Transformer and Newer Architectures

Dareen Alharthi, Hao Chen

Spring 2025
Attendance:@964

Content

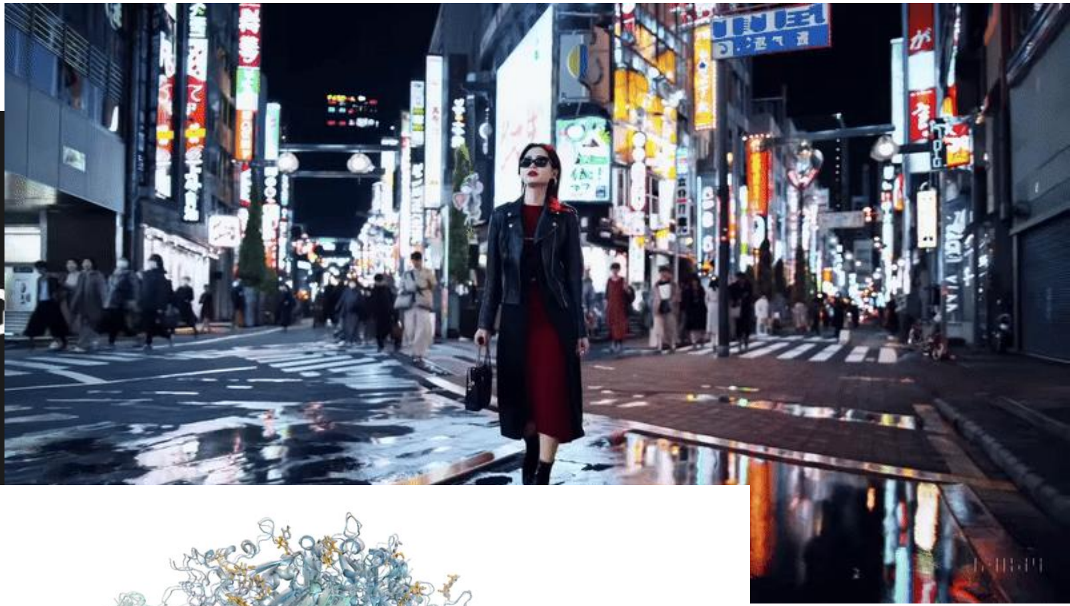
- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws

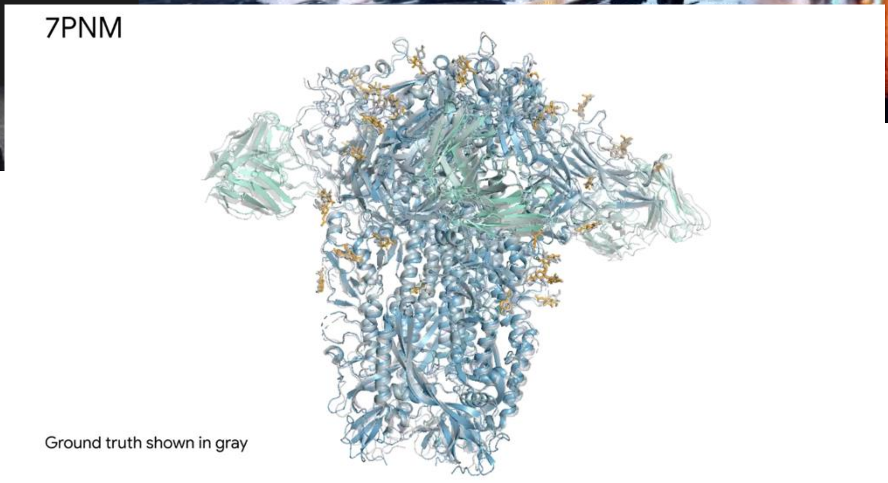
Content

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

Why Transformer?

- Almost everything today in deep learning is **Transformer**





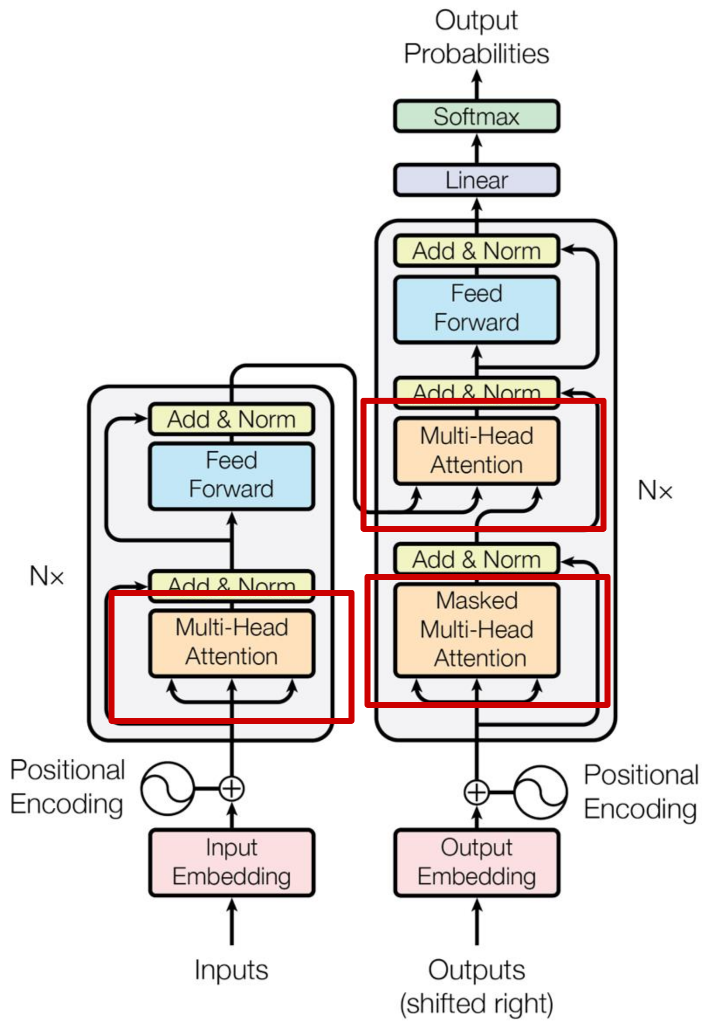
But…Why Transformer?

- 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

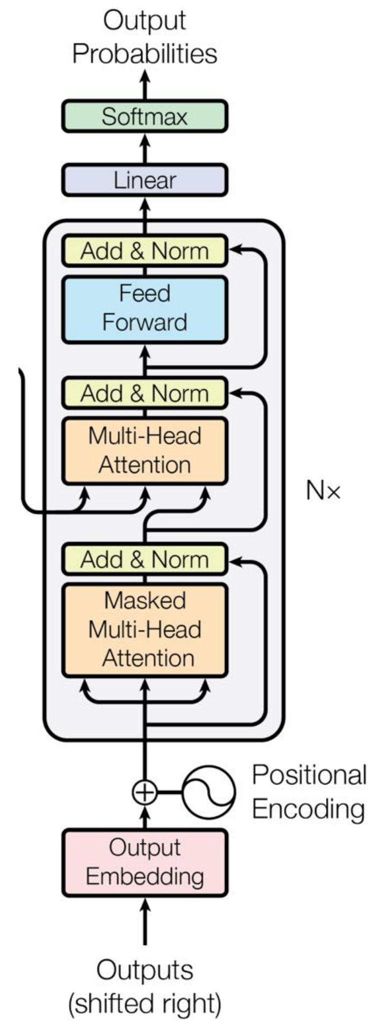
- overview



Vaswani, A. "Attention is all you need." *Advances in Neural Information Processing Systems* (2017).

Transformer Architecture

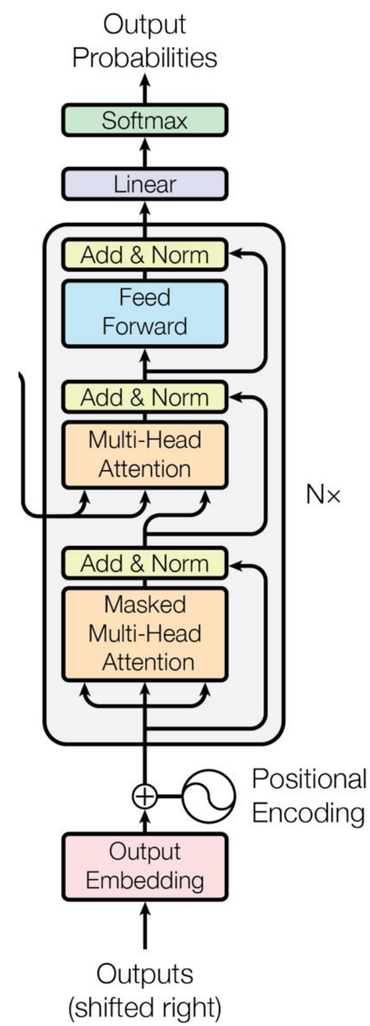
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Transformer Architecture

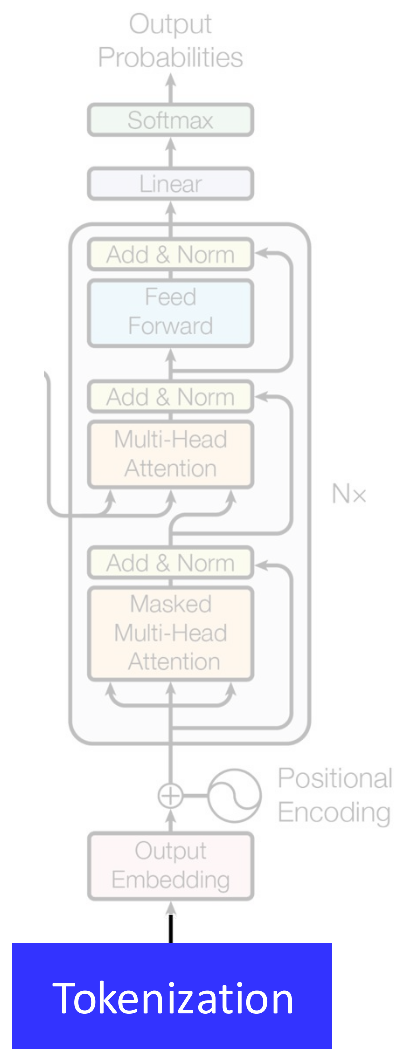
- Word Tokenization
- Word Embedding
- **(Masked) Multi-Head Attention**
- **Position Encoding**
- **Feed-Forward**
- **Add & Norm**
- Output Projection Layer

Transformer Block



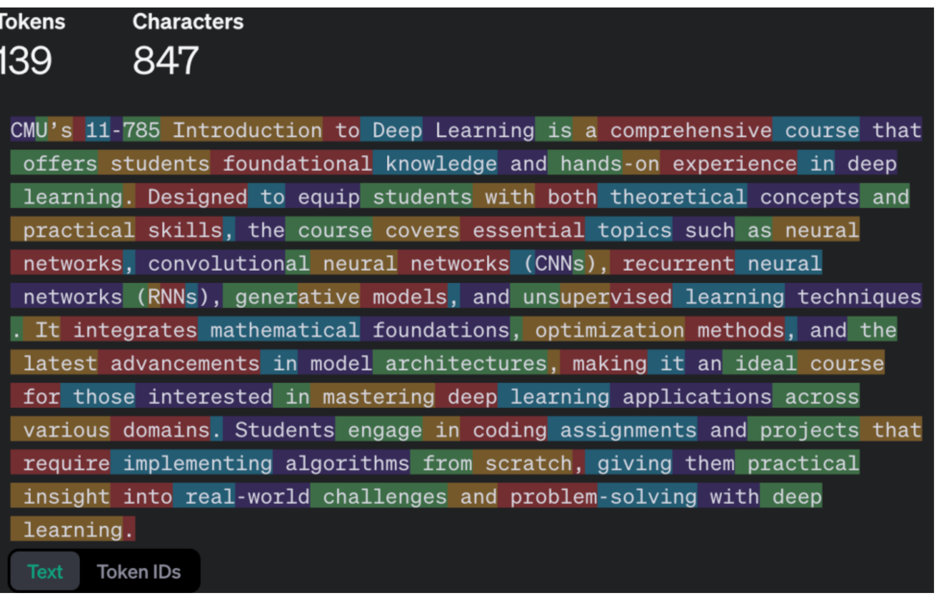
Transformer Architecture

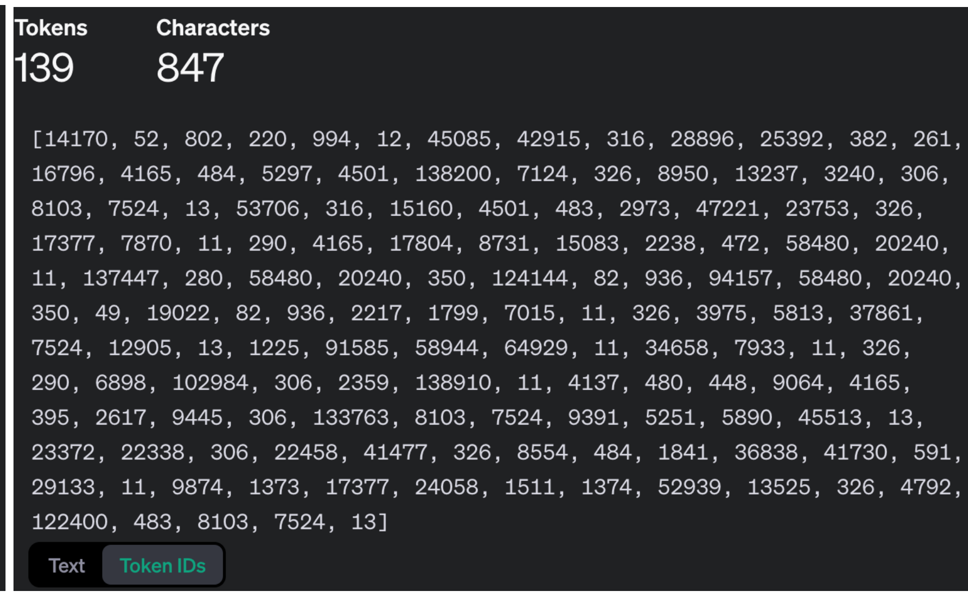
- **Word Tokenization**
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Tokenization

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

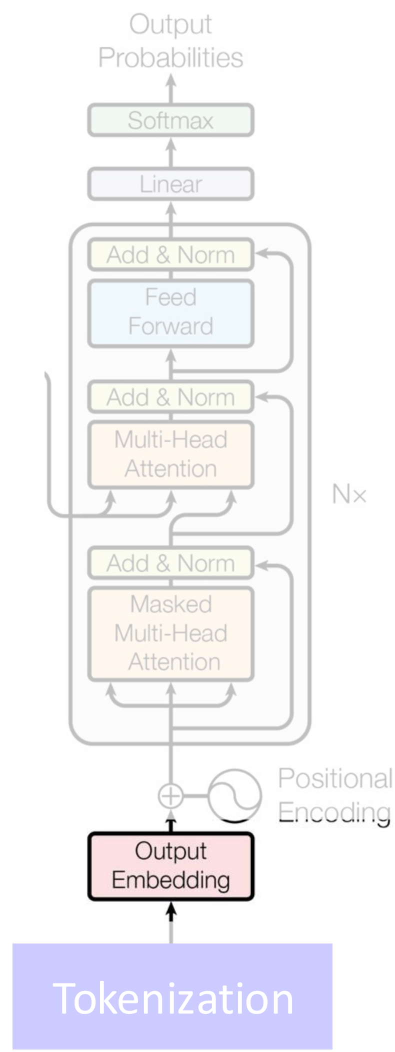




https://platform.openai.com/tokenizer

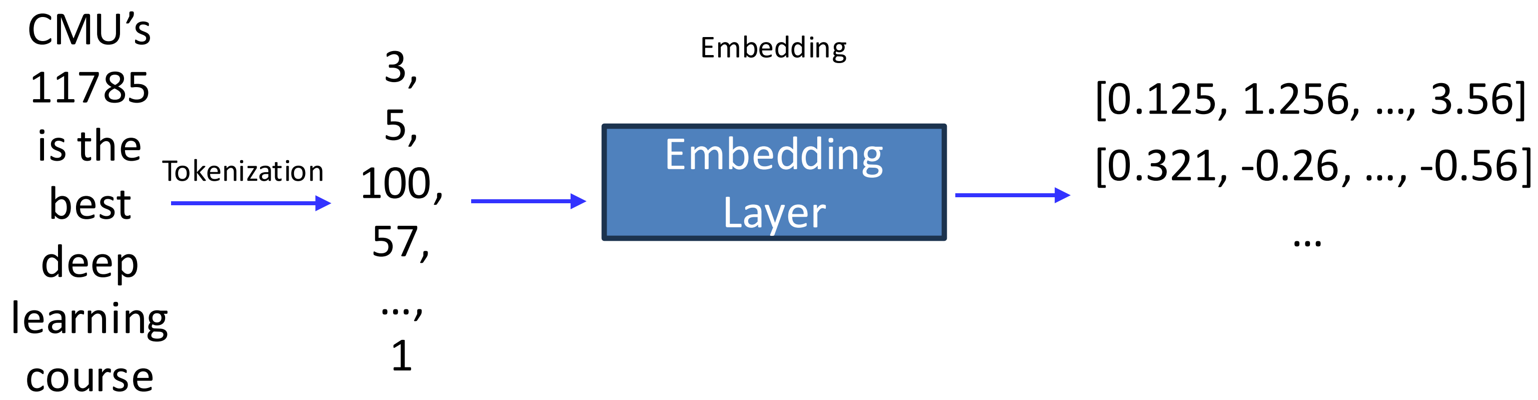
Transformer Architecture

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

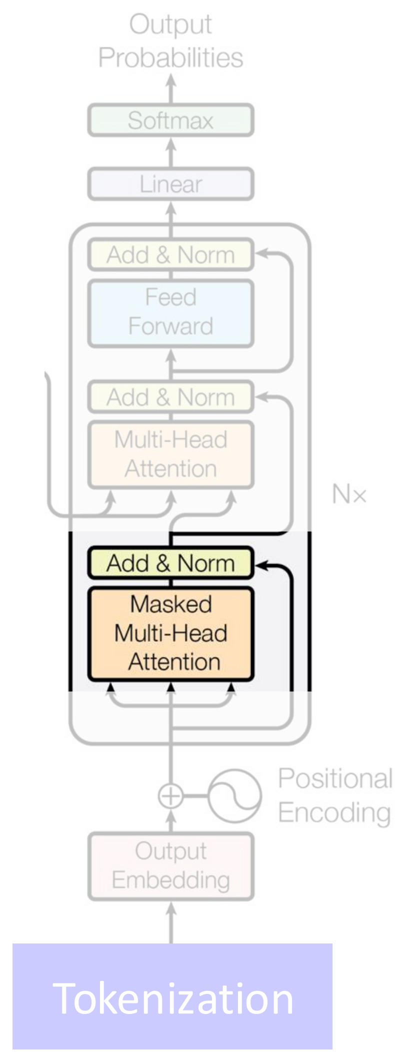
$$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} 0.235 & -1.256 & 3.513 & \dots & -0.187 \\ 1.291 & -2.012 & 0.624 & \dots & -1.291 \\ 0.535 & 0.012 & -0.024 & \dots & 2.345 \\ \dots & \dots & \dots & \dots & \dots \\ 0.131 & 2.102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

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

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



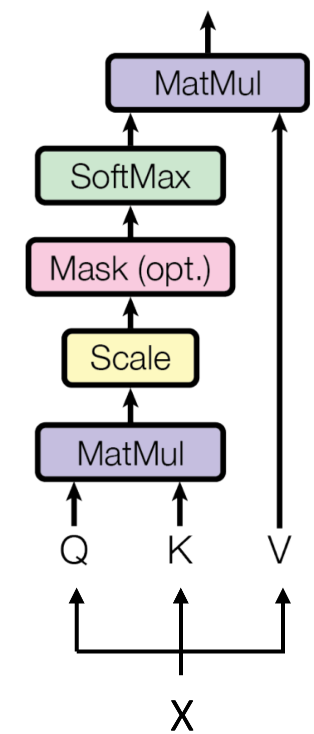
Self-Attention

- Attention Operation

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

- 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

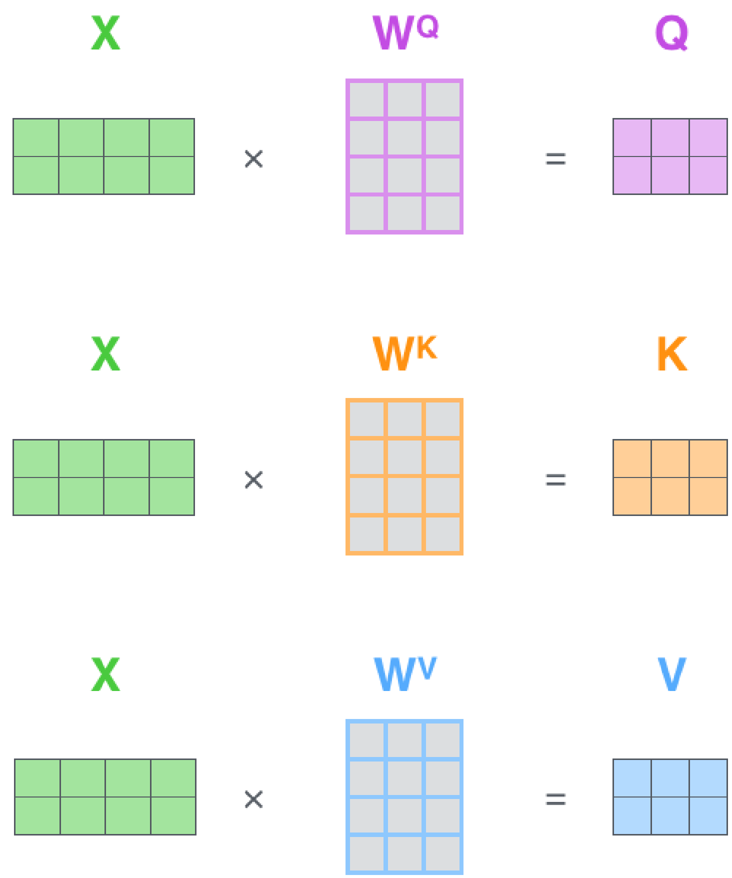
Scaled Dot-Product Attention

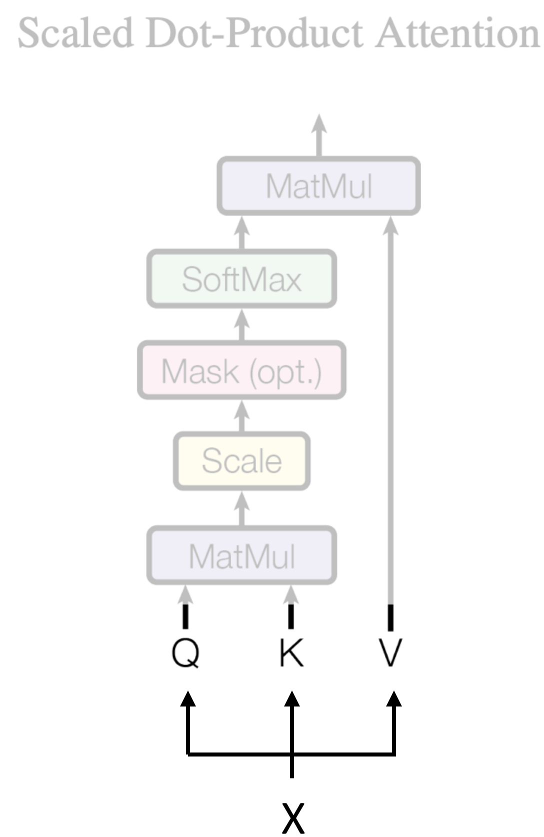


Self-Attention

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

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



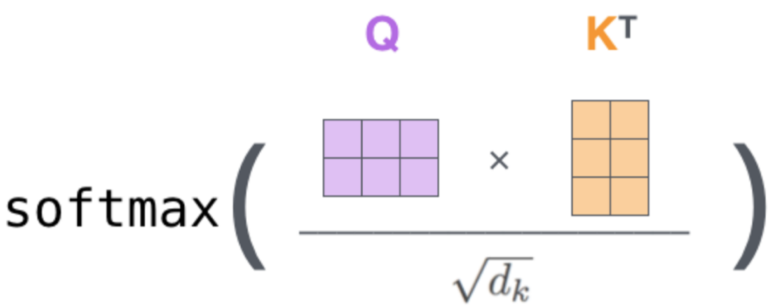


The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

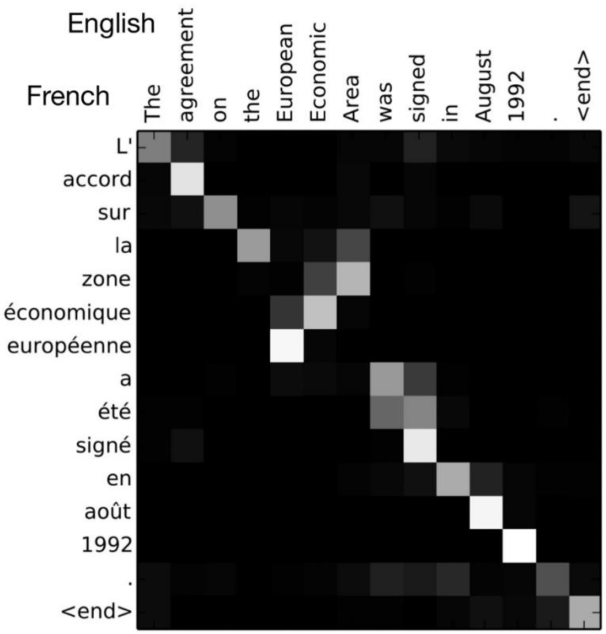
Self-Attention

- Attention weights

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

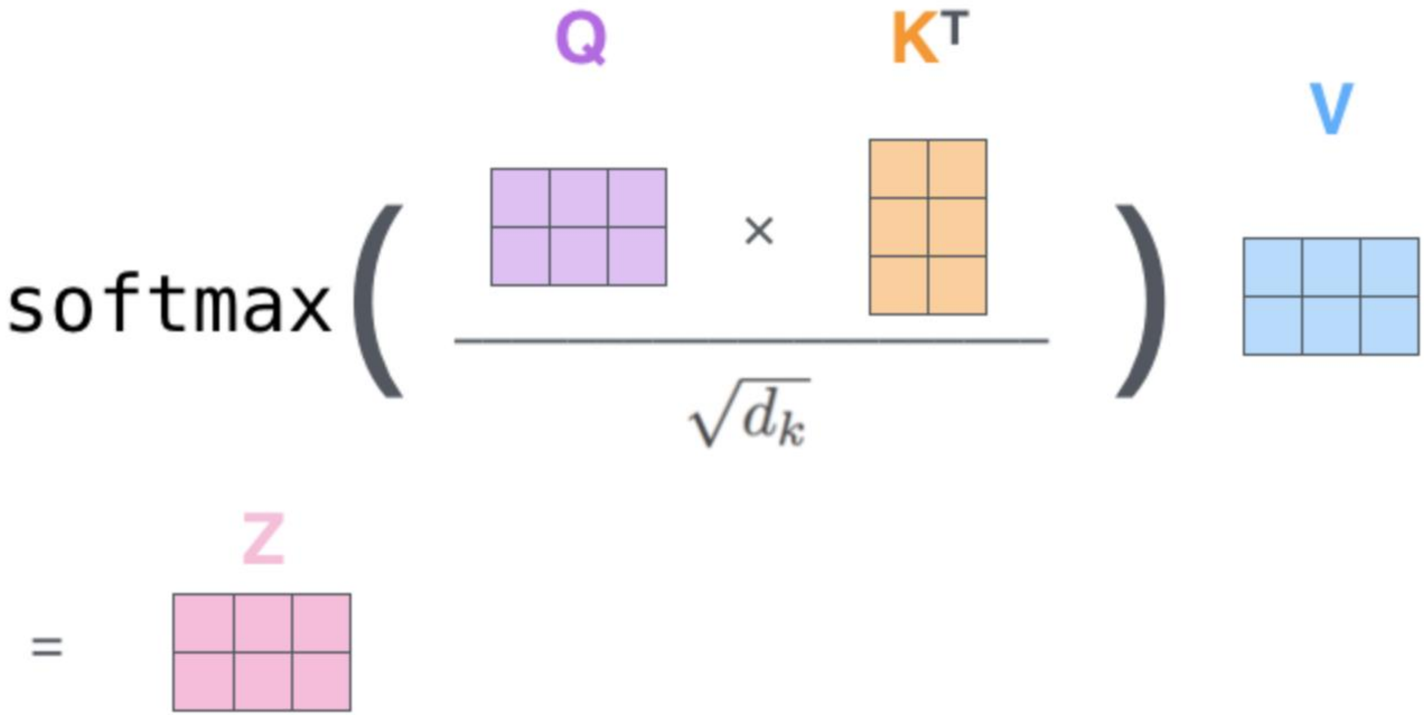


$\mathbb{R}^{L \times L}$

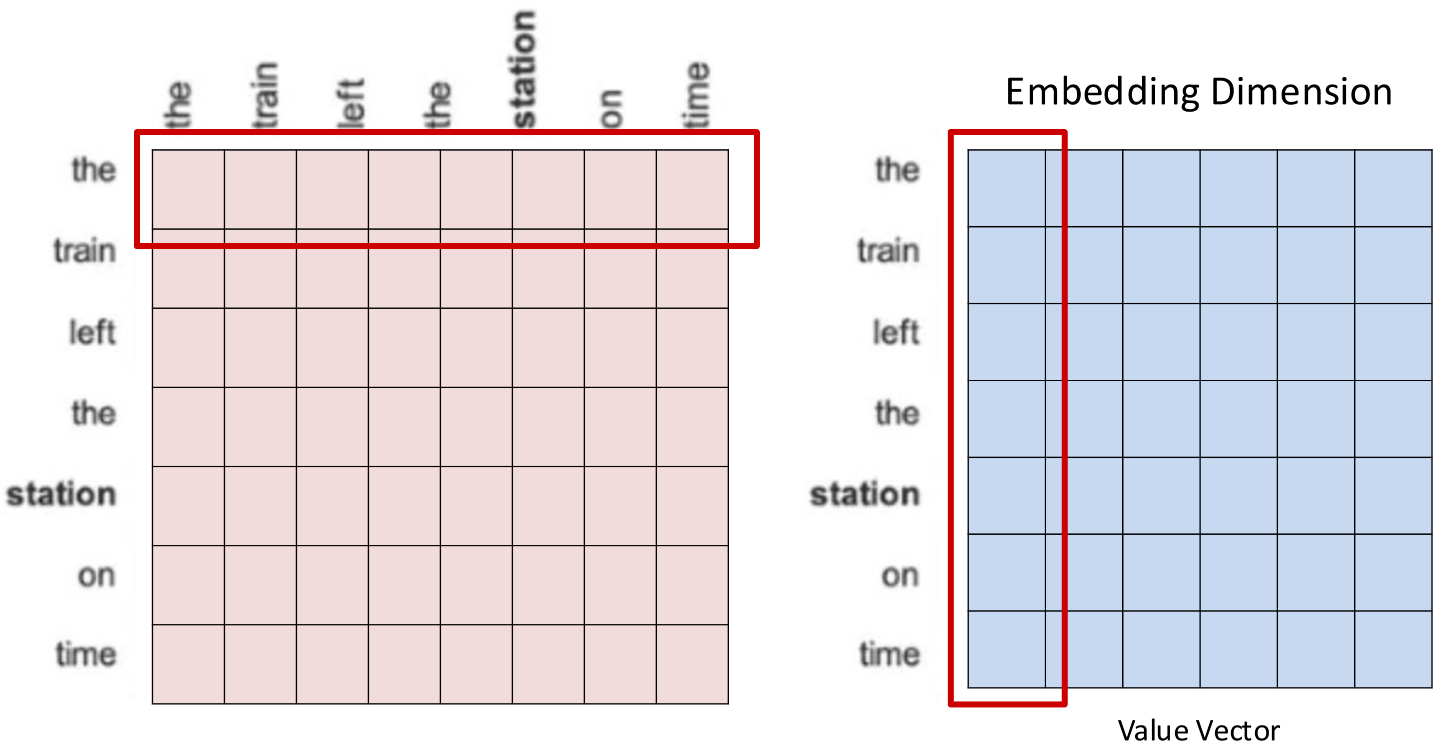


Self-Attention

- Output



Self-Attention



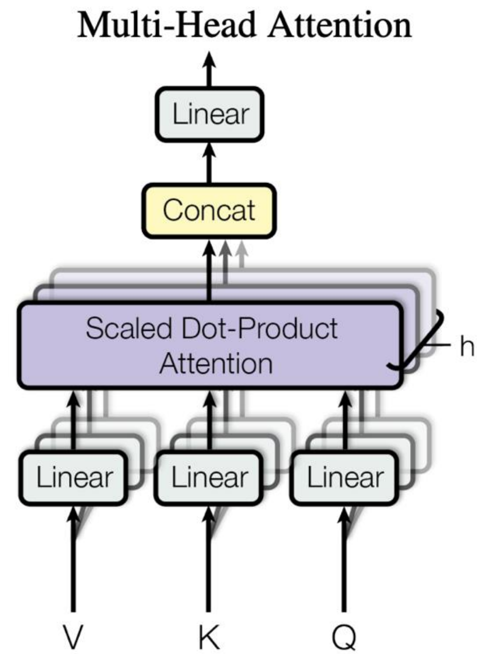
Weighted-sum of V based on Attention Scores

Multi-Head Self-Attention

- Multiple self-attention operations over the channel dimension

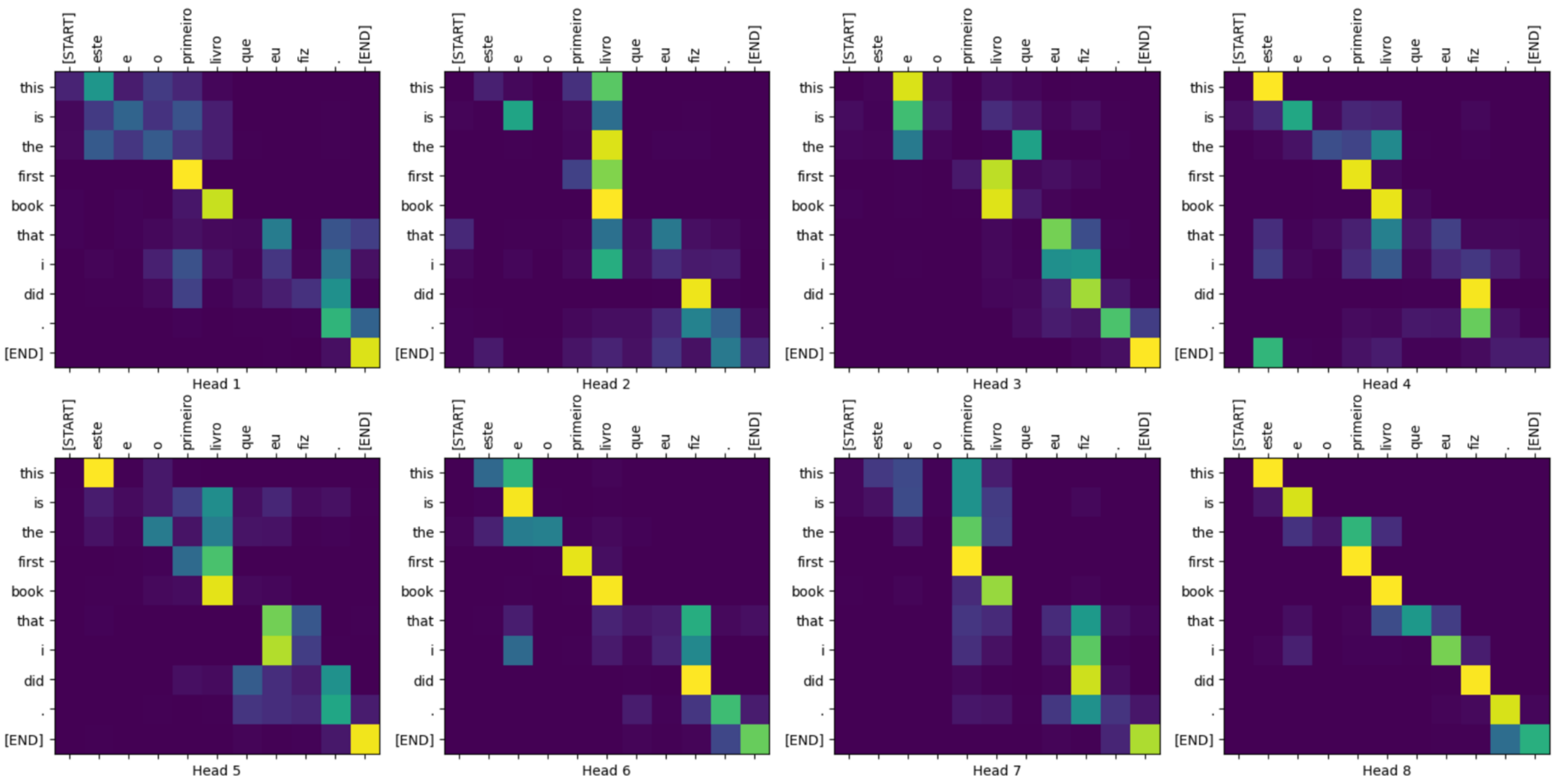
$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{ head }_{\text{h}}\right)W^Q$$
$$\text{where head } = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

- Different attention maps capture different relationships



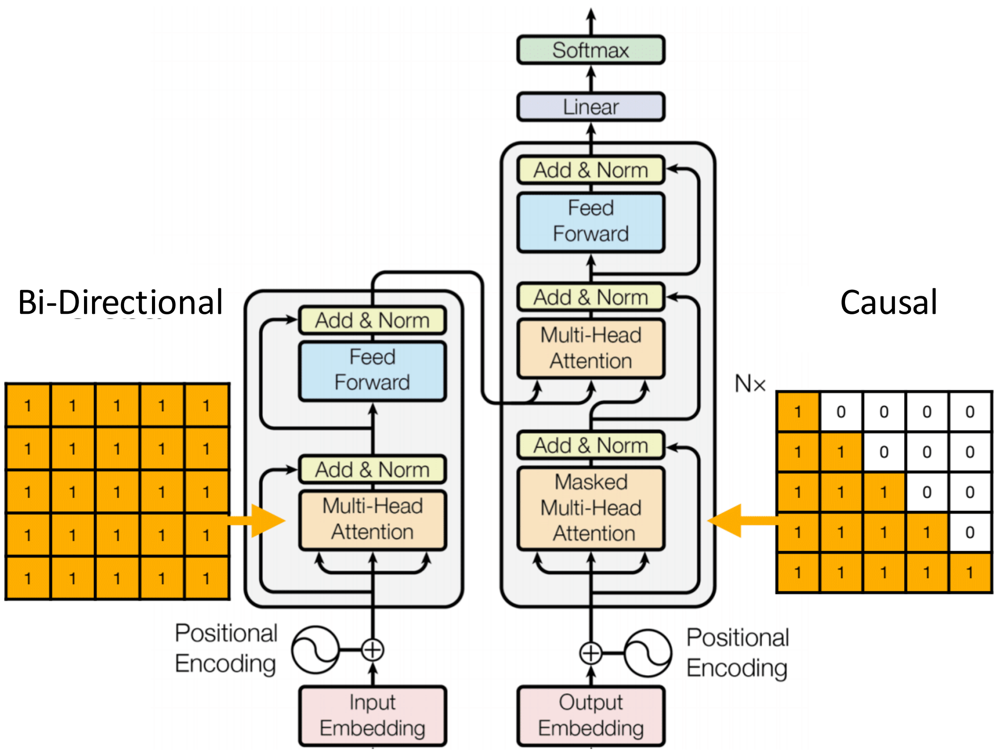
Multi-Head Attention

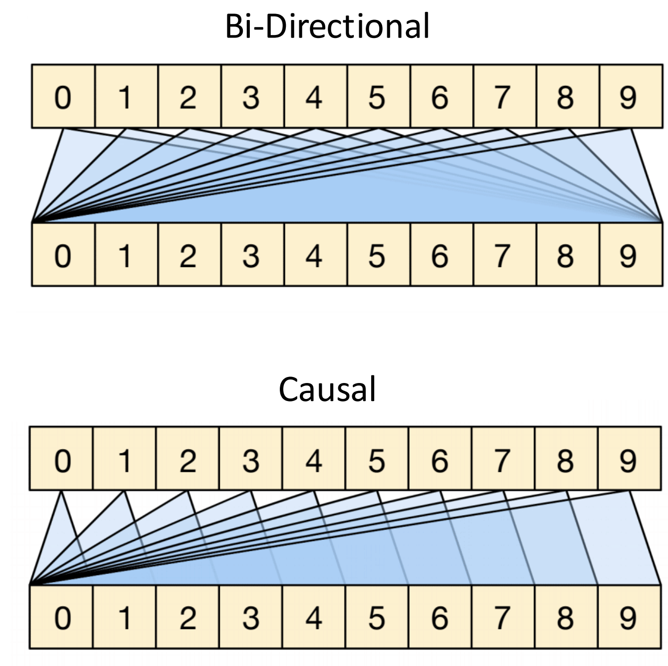
- Each head captures different semantics



https://www.tensorflow.org/text/tutorials/transformer

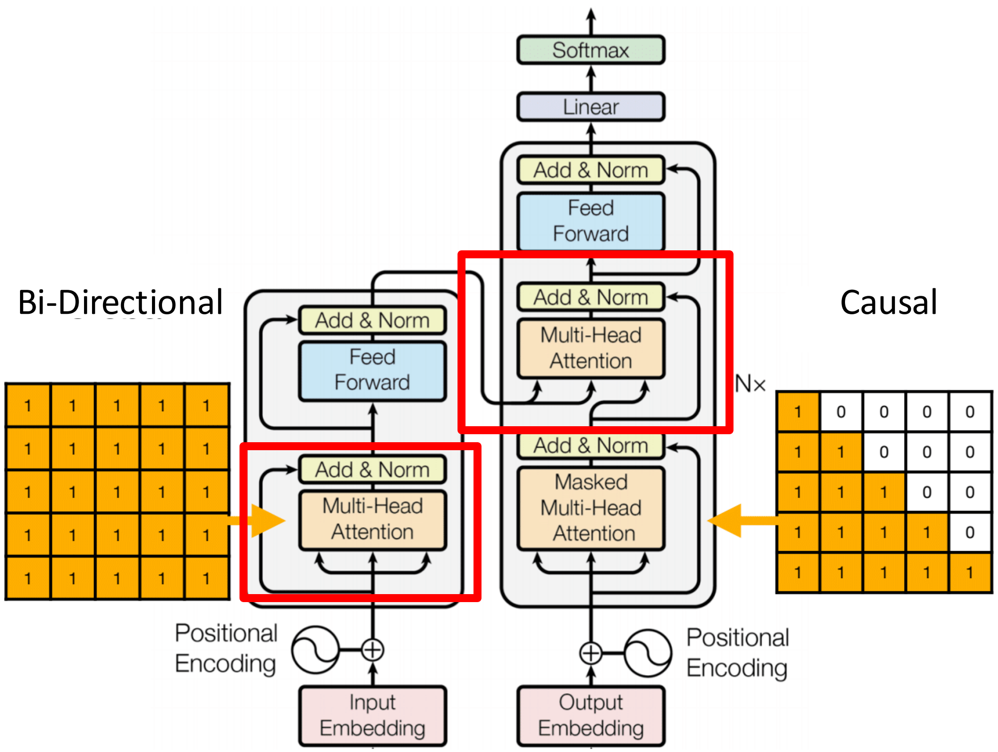
Attention Masking

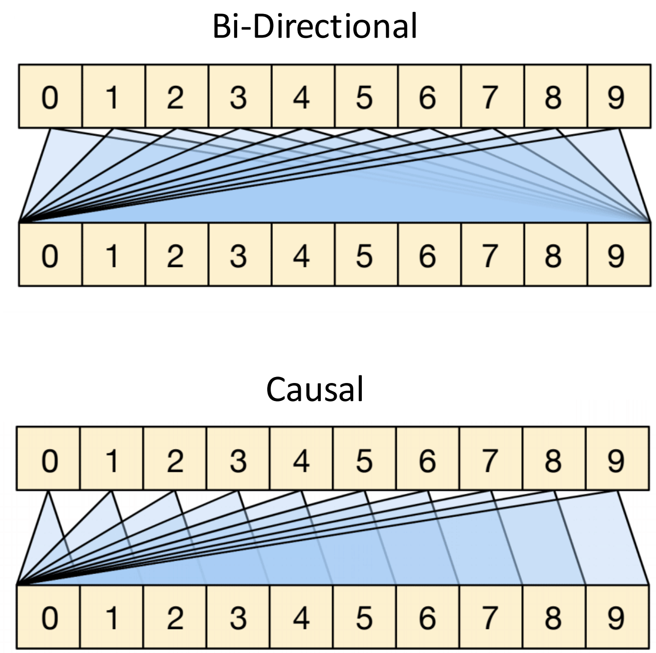




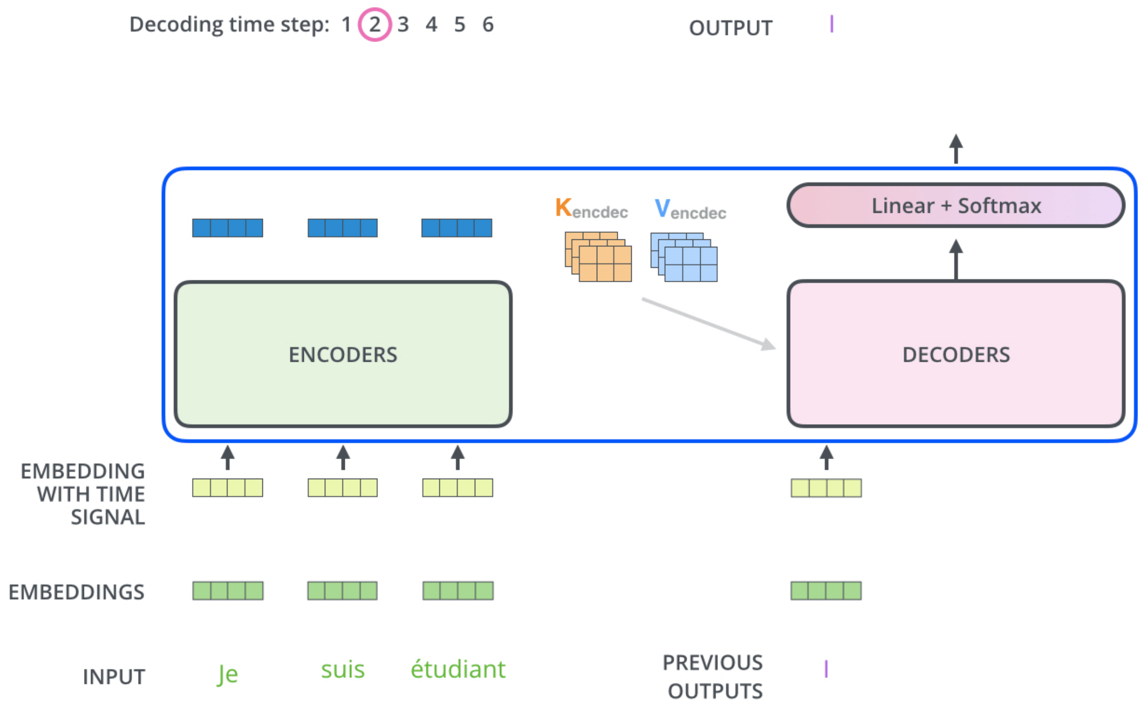
Credit to: Han Song. TinyML. Lecture 12 Transformers.

Attention Masking





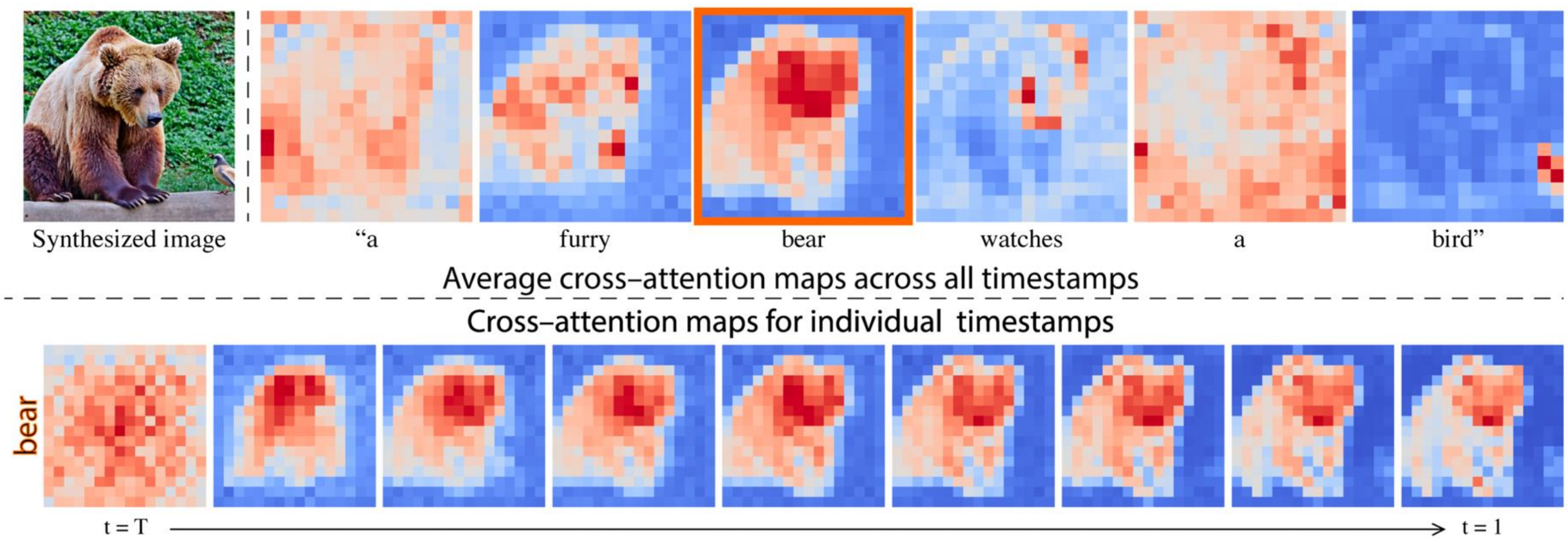
Cross-Attention



The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Cross-Attention

query, "a furry bear watches a bird."

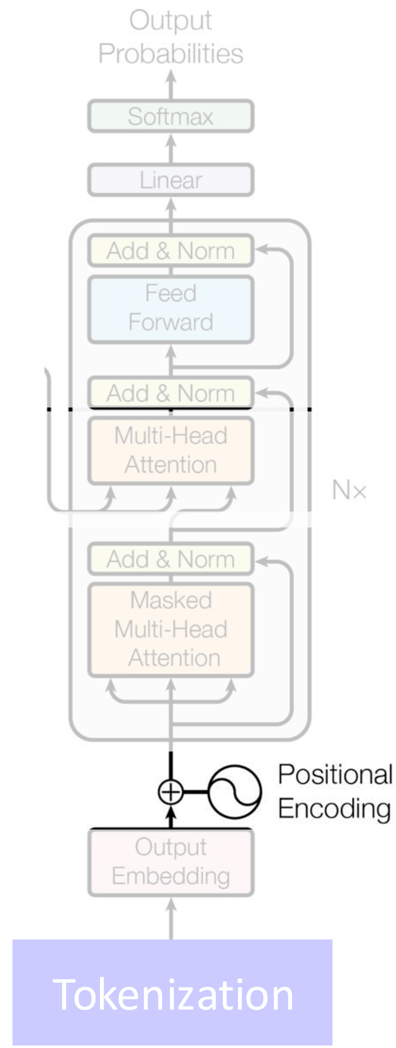


The model iteratively denoise the noise vector based on the given text query to generate an Image

Wonsik Shin, Jessica Ruan, Aradhya Talan, and Brandon Dong. "3D Gaussian Splatting Editing with Diffusion Personalization." IDL Project - Carnegie Mellon University.

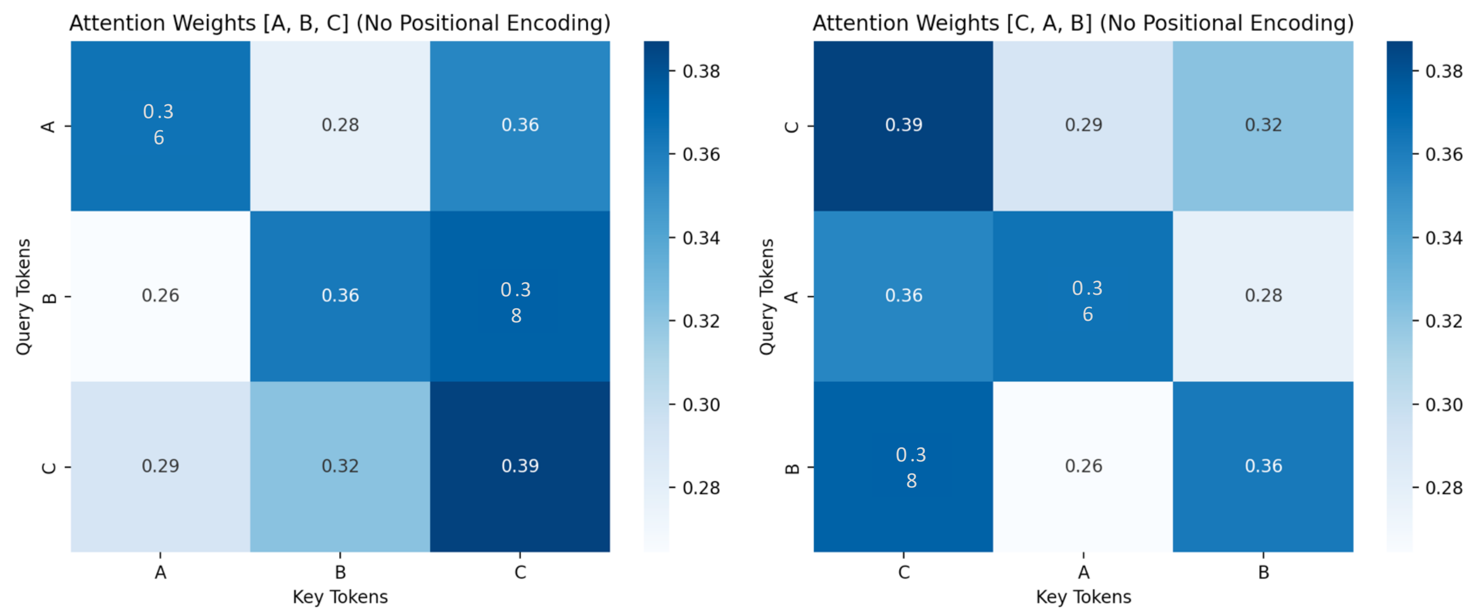
Transformer Architecture

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



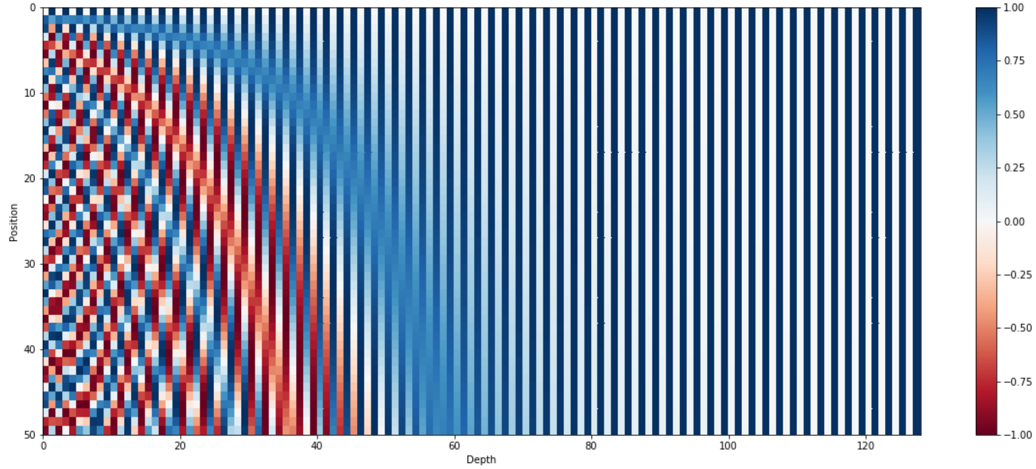
Position Encoding

- 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

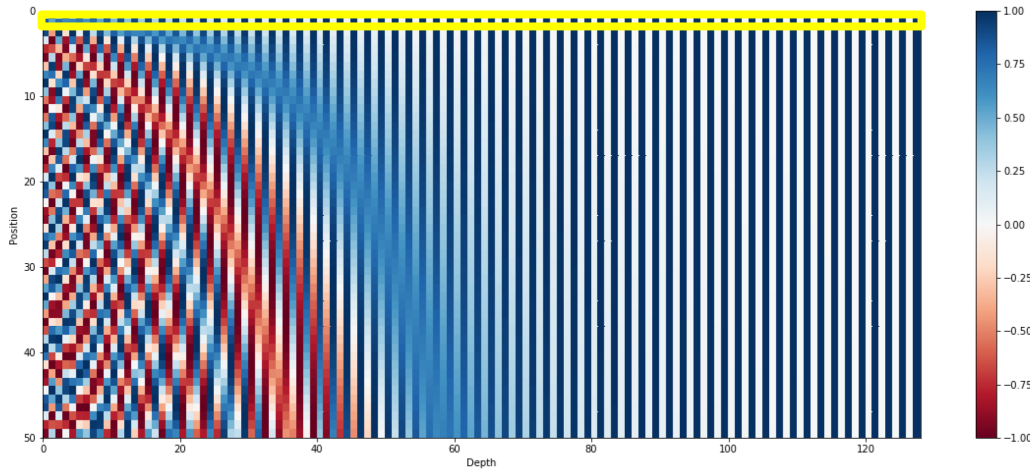


$$p_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

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

Position Encoding

- Captures the abs./relative distance between tokens



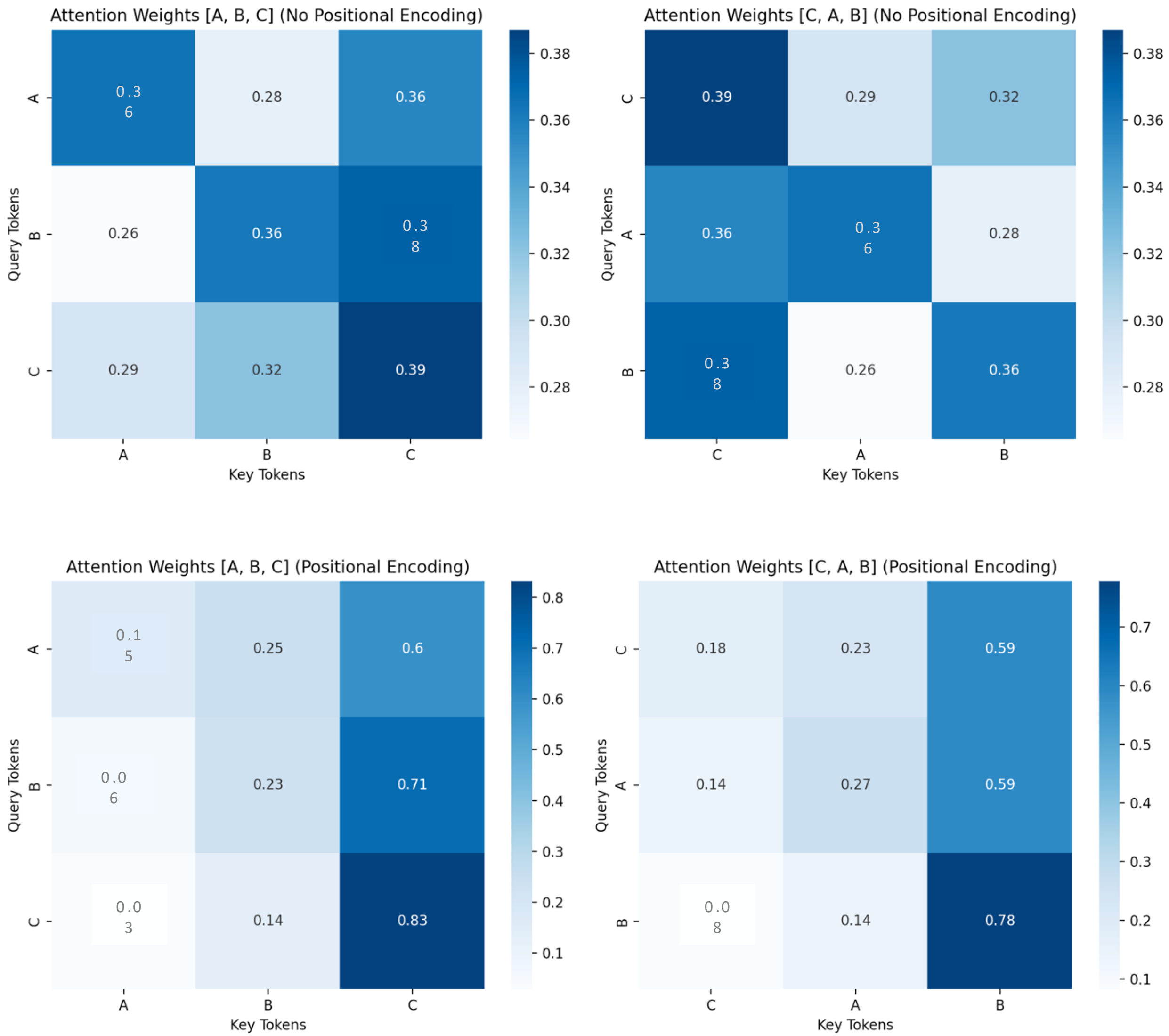
$$p_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

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

Position Encoding

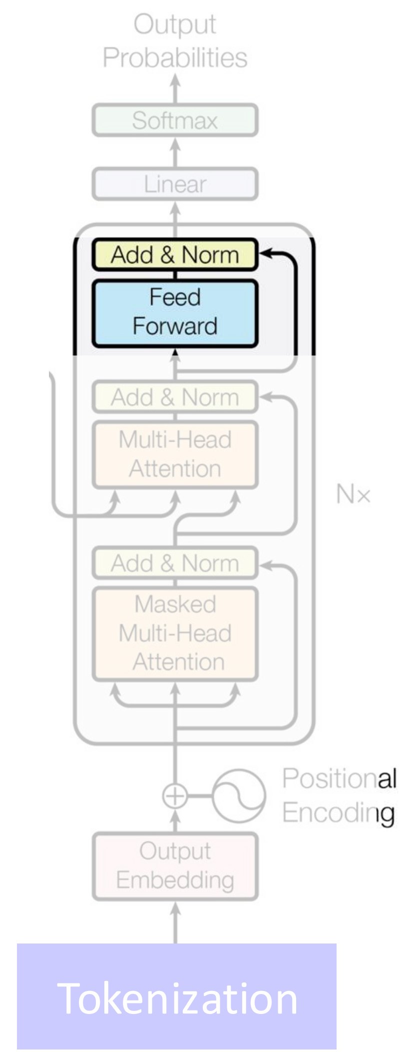
No Position Info.

With Position Info.



Transformer Architecture

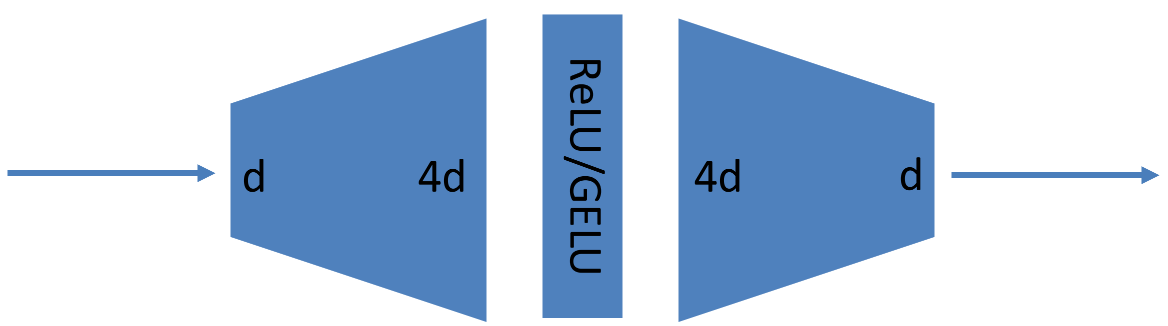
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- **Feed-Forward**
- Add & Norm
- Output Projection Layer



Feed-Forward Block

- Just a MLP!

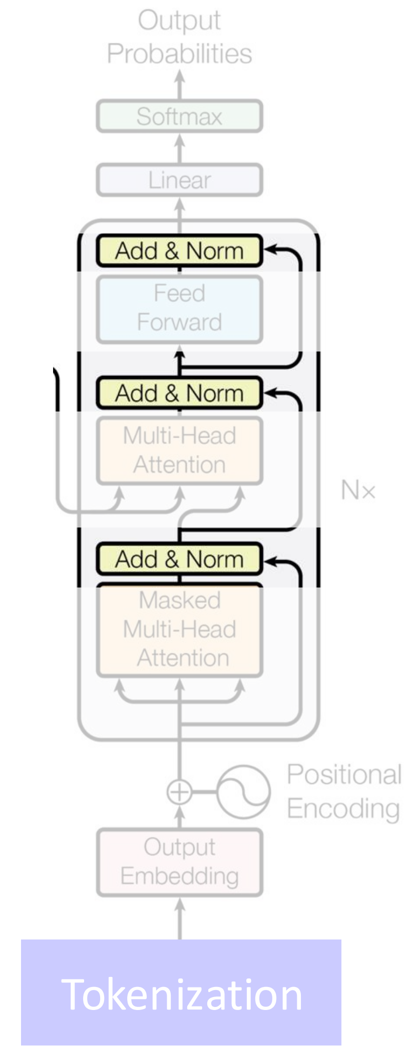
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



https://www.youtube.com/watch?v=KJtZARuO3JY&t=694s

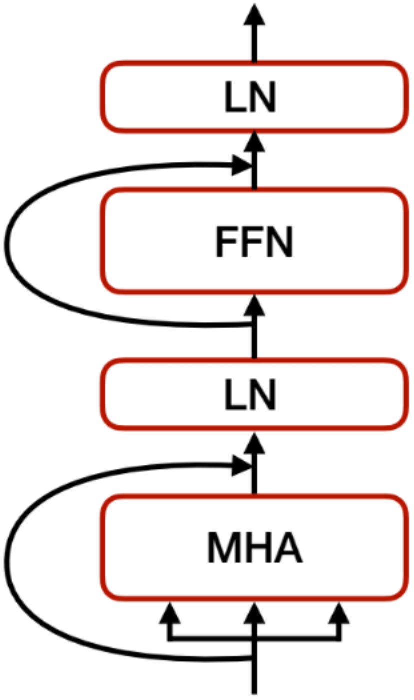
Transformer Architecture

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



Residual and Normalization

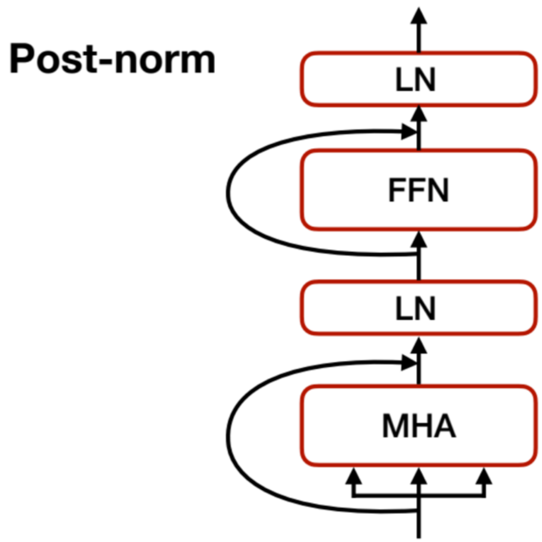
- Each layer in Transformer has:
  - A residual connection
  - A normalization layer
- Layer Norm. normalize each token by its embedding size dimension
  - For more stable training



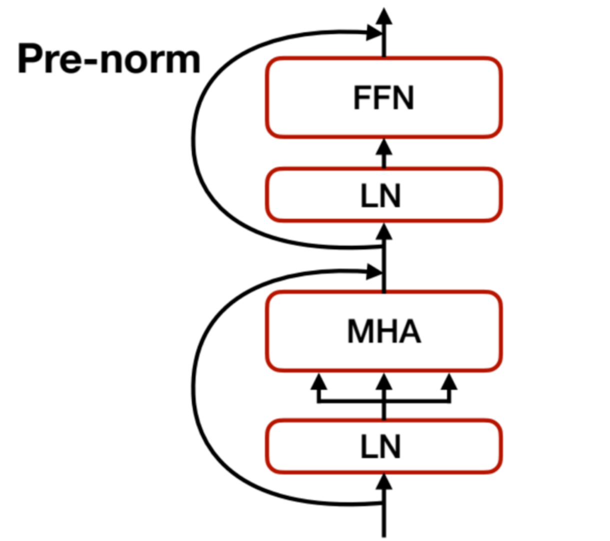
Position of Normalization

- Post-Norm vs Pre-Norm

$$\boldsymbol{x}_{t+1} = \mathrm{Norm}\left(\boldsymbol{x}_t + F_t\left(\boldsymbol{x}_t\right)\right)$$



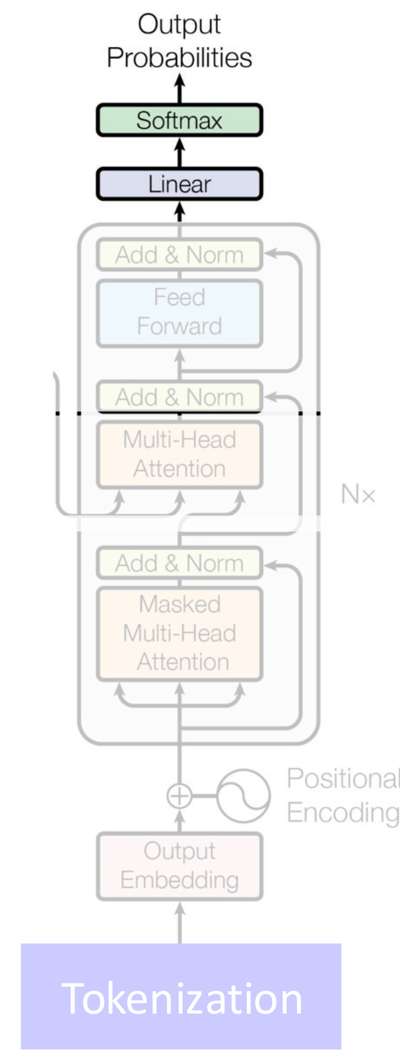
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + F_t\left(\mathrm{Norm}\left(\boldsymbol{x}_t\right)\right)$$



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

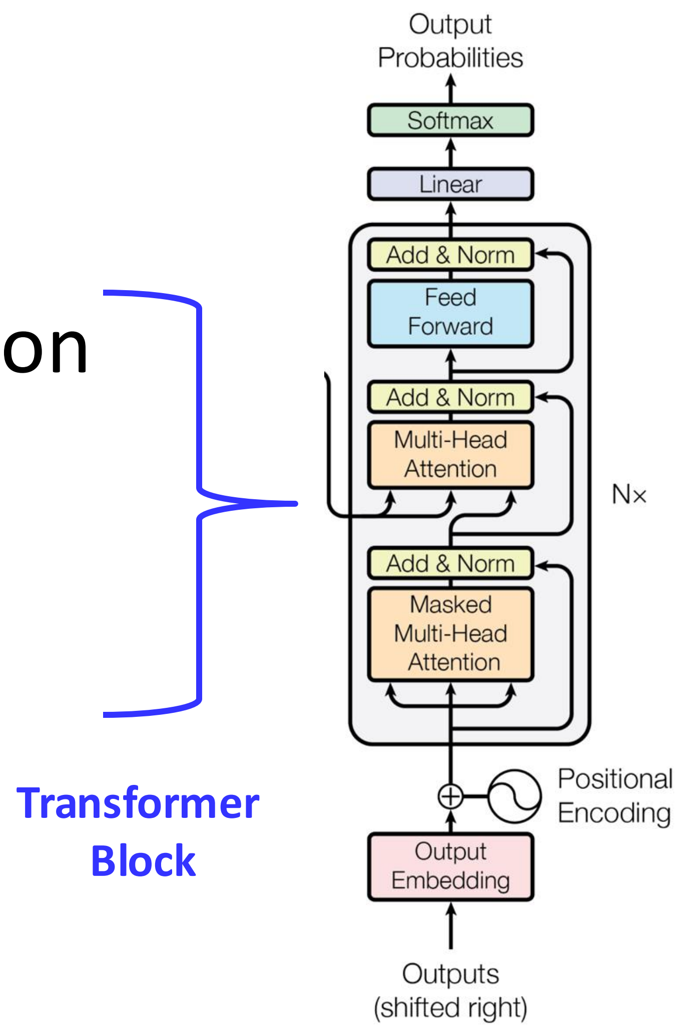
Transformer Architecture

- 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



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
- In self-attention 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 the dimension d before computing softmax
- In self-attention Q, K, V are copy of input X

Content

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

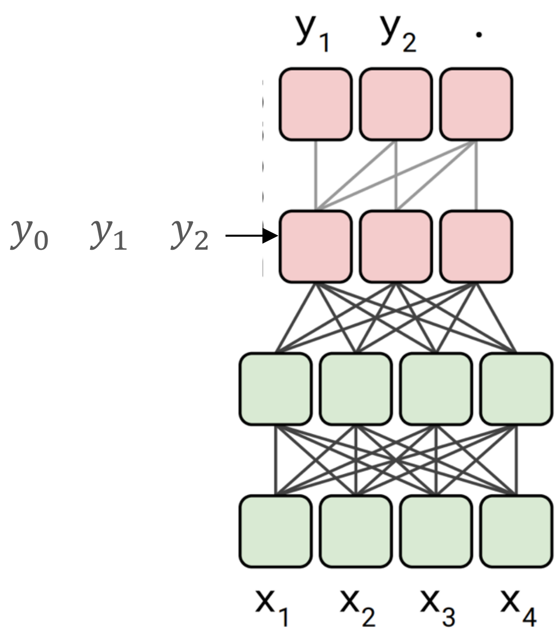
Overview

- 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

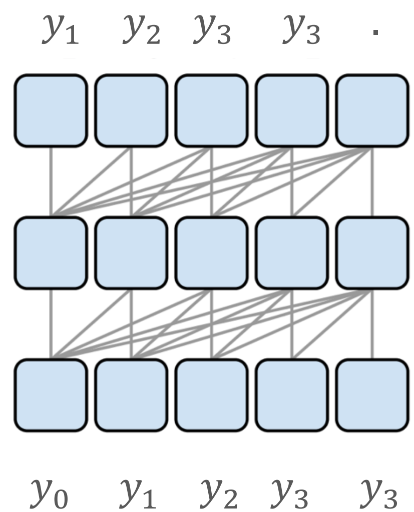
Overview

- 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

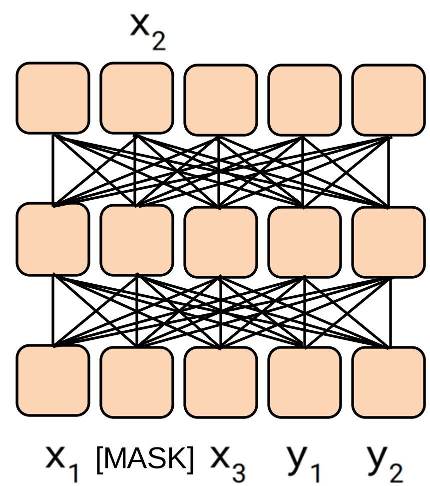
Encoder-Decoder

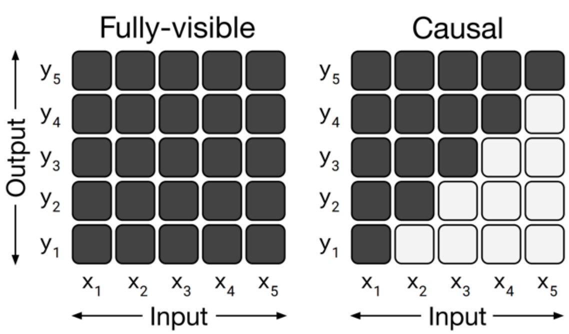


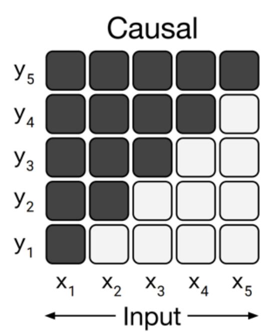
Decoder-Only

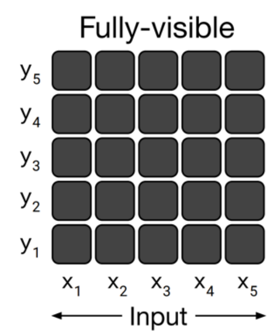


Encoder-Only



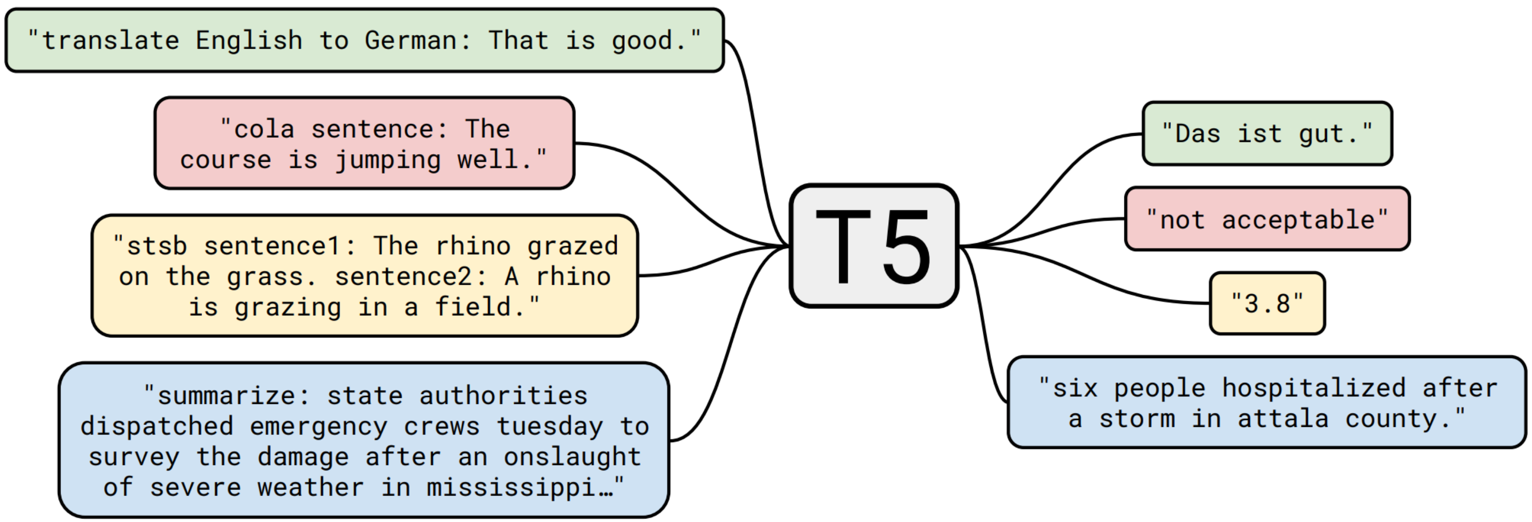






Encoder-Decoder - T5

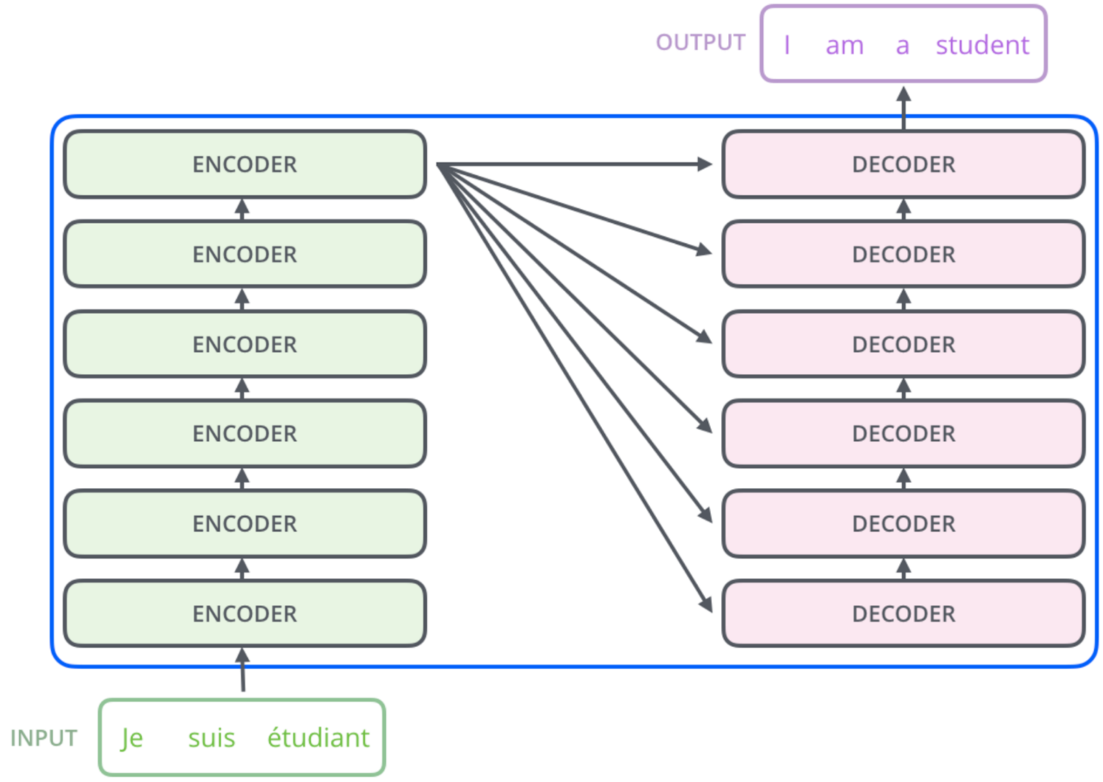
- Encoder-Decoder architecture as in the original transformer paper
- A text-to-text model on various NLP tasks



Raffel et. al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. 2019

Encoder-Decoder - T5

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

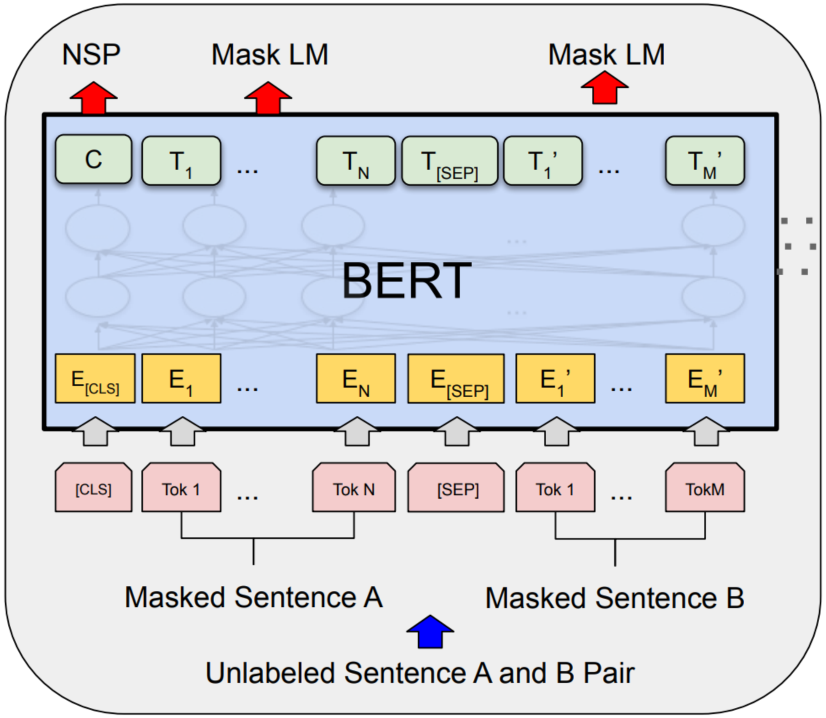


Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Encoder-Only - BERT

- Bidirectional Encoder Representations from Transformers (BERT)
  – Encoder-only arch.

- Trained with
  – Mask token prediction
  – Next sentence prediction

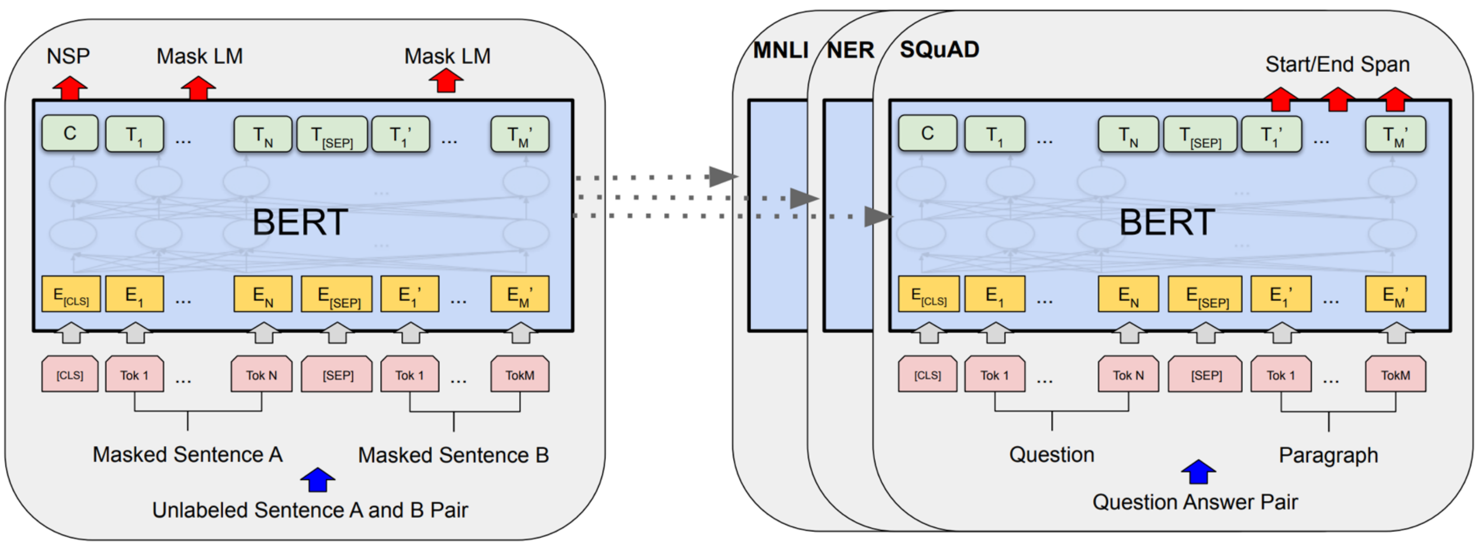


Jacob et. al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019

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



Pre-training

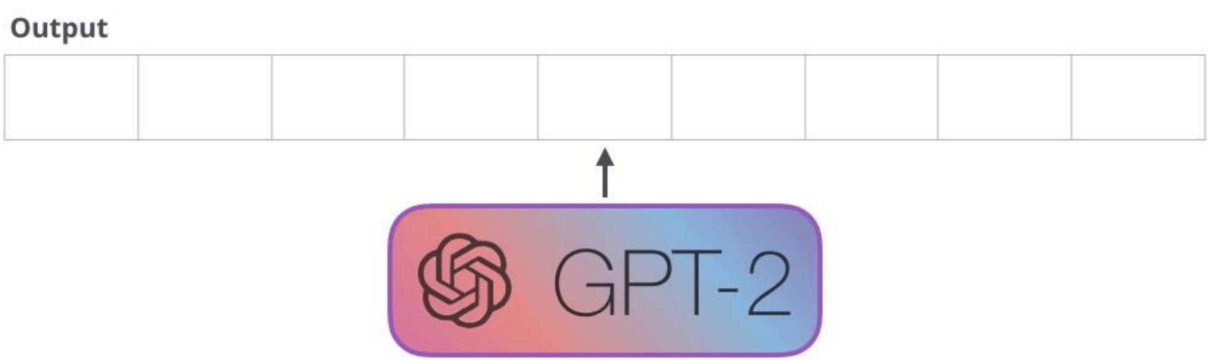
Fine-Tuning

Jacob et. al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019

Decoder-Only - GPT

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

$$L_1(\mathcal{U}) = \sum_i \log P\left(u_i \mid u_{i-k}, \ldots, u_{i-1}; \Theta\right)$$



Radford et. al. Improving Language Understanding by Generative Pre-Training.
Illustrated GPT-2.

Large Language Model

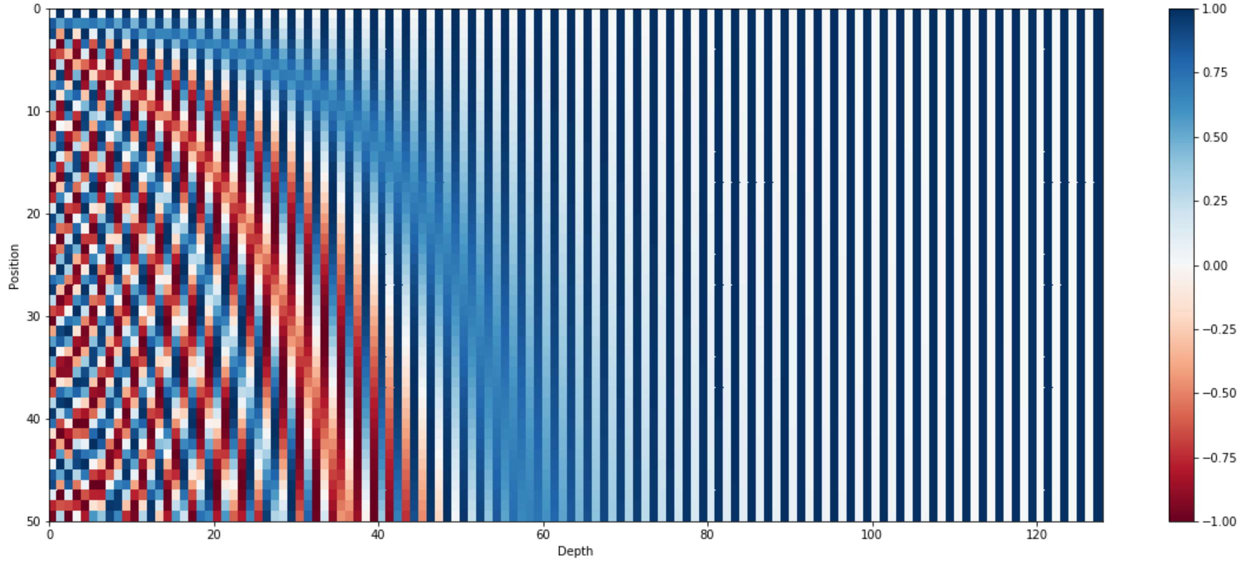
- GPT-2
  - Pre-training and fine-tuning on specific tasks
- GPT-3
  - zero-shot capability
  - in-context learning
  - Foundation for ChatGPT!
- GPT-4

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

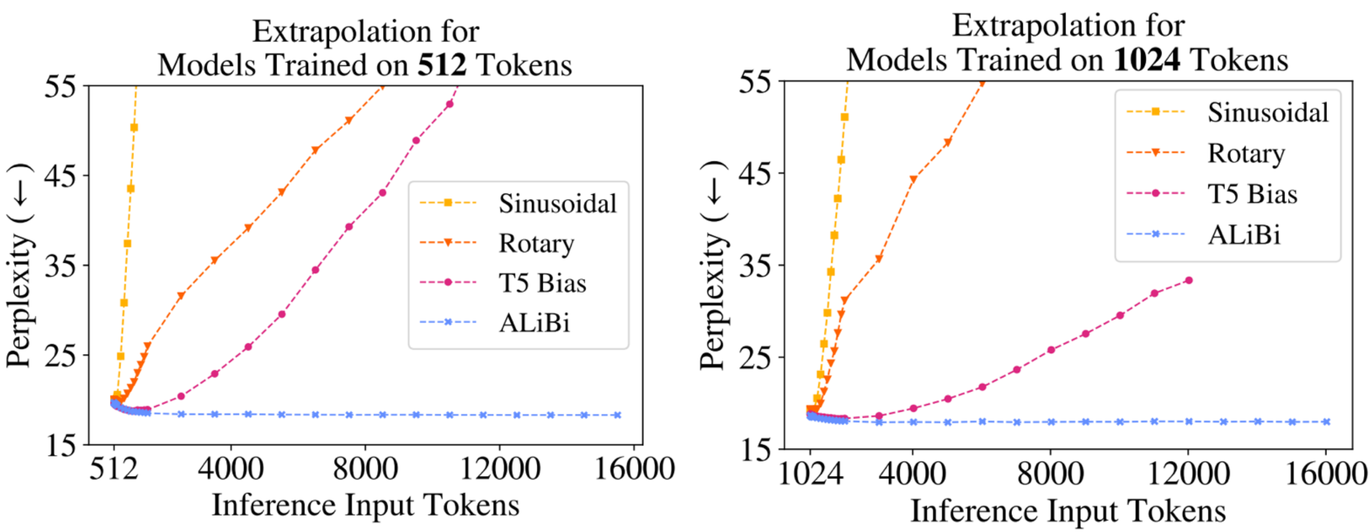
- Absolute position embedding fuses the position information into input embeddings



- Fixed length! Not generalize to longer input sequence

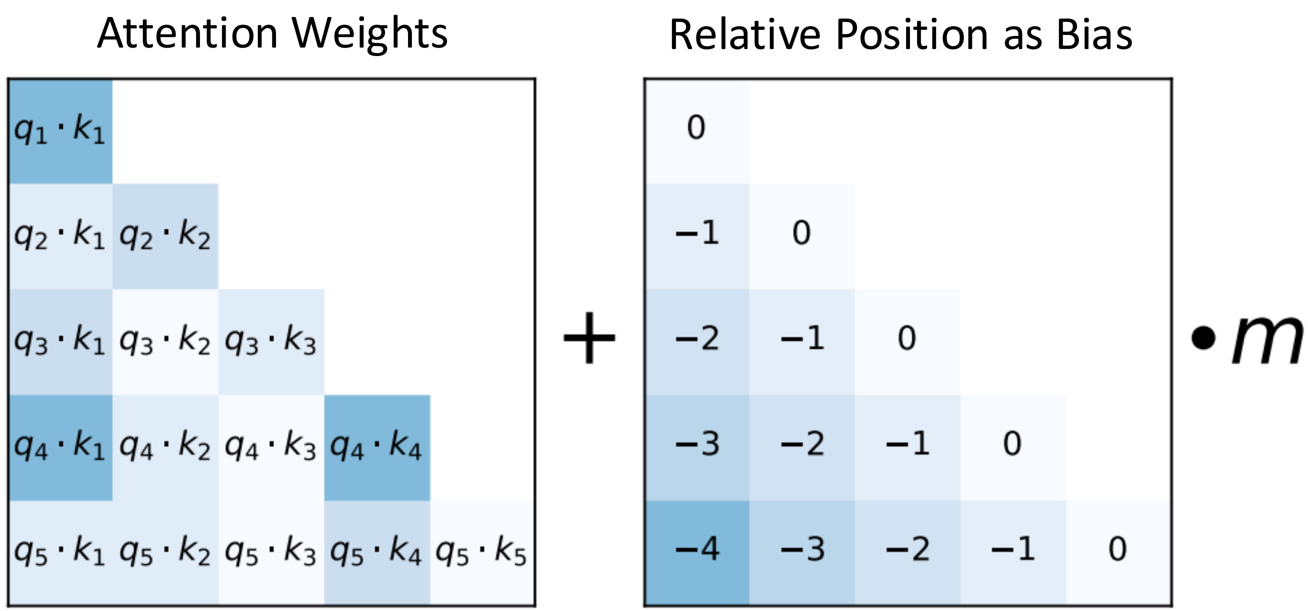
Relative Position Encoding

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



Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. 2021.

Relative Position Encoding



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

Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. 2021.

Rotary Position Encoding

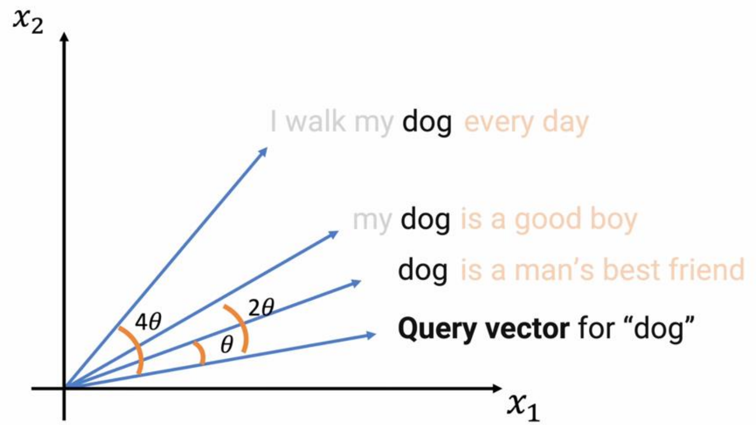
- Used in Large Language Models such as LLAMA
- Rotate the embedding in 2D space

$$< f_q(x_m, m), f_k(x_n, n) >= g(x_m, x_n, m-n)$$

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

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

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



Su et al. RoFormer: Enhanced Transformer with Rotary Position Embedding. 2021

How Rotary Position Embedding Supercharges Modern LLMs: https://www.youtube.com/watch?v=SMBkImDWOyQ

Rotary Position Encoding

- General form

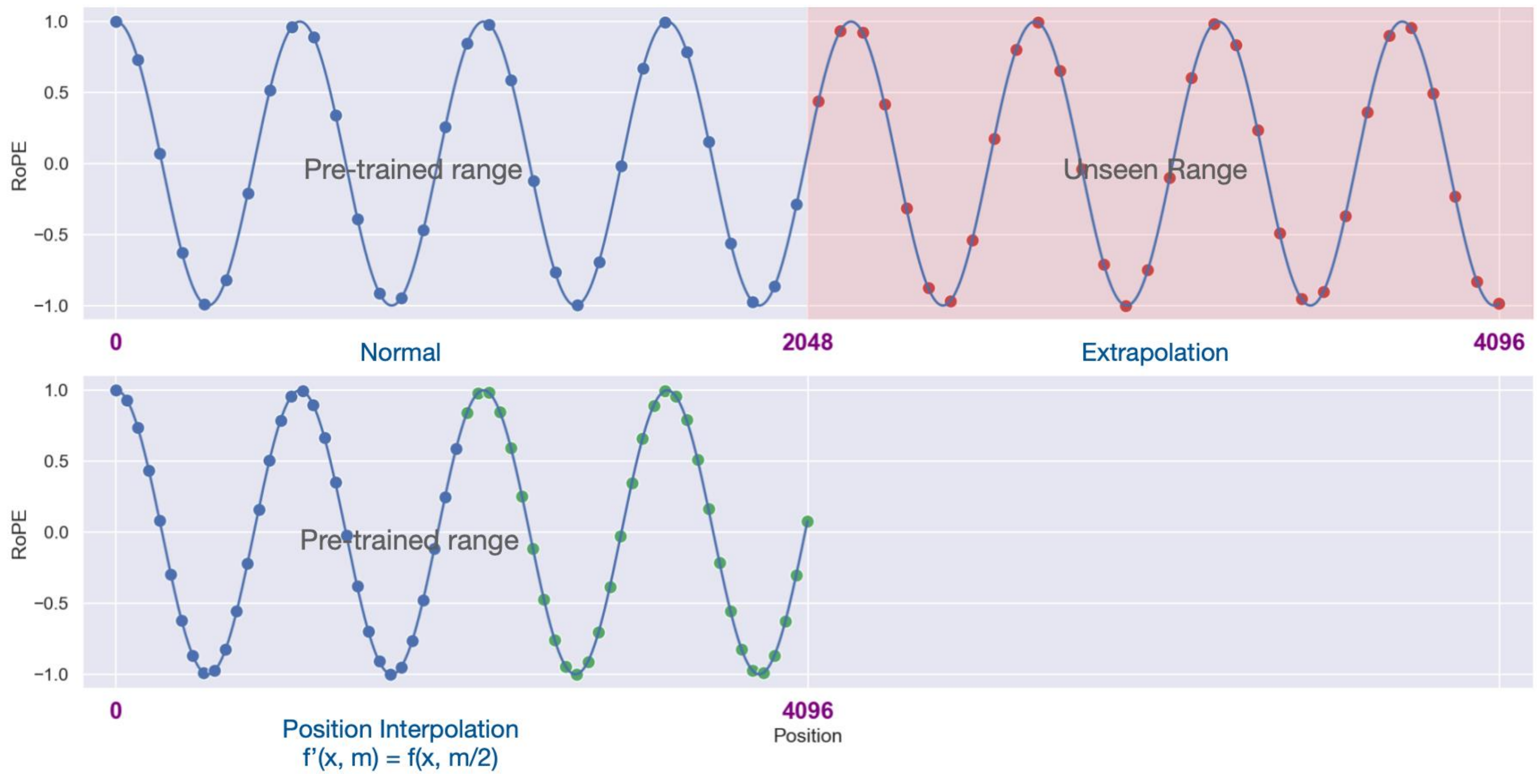
$$
\begin{aligned}
f_q(x_m, m) &= (W_q x_m) e^{im\theta} \\
f_k(x_n, n) &= (W_k x_n) e^{in\theta} \\
g(x_m, x_n, m-n) &= \mathrm{Re}\left[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}\right]
\end{aligned}
$$

$$
f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m
$$

$$
\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix}
\cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\
\sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\
0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\
0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{\boxed{d/2}} \\
0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2}
\end{pmatrix}
$$

Rotary Position Encoding

- Allows extension of the context window



Chen et al. Extending Context Window of Large Language Models via Positional Interpolation. 2023.

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

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

  – $O\left(L^2 d\right)$ 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

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

- Kernel function

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

- Linear form of attention

$$\frac{\boxed{\phi(Q)\phi(K)^T}}{\sum_{i=1}^{L}\phi(Q)\phi(K_i)^T}V = \frac{\phi(Q)\boxed{\left(\phi(K)^T V\right)}}{\phi(Q)\sum_{i=1}^{L}\phi(K_i)^T}$$

$O\left(L^2\right)$

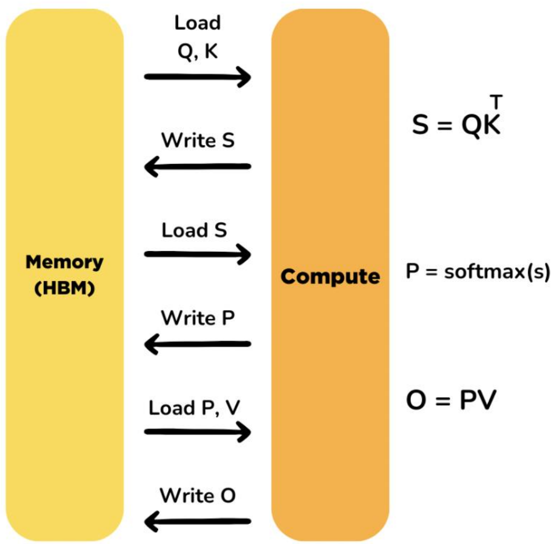
$O\left(d'd\right)$

Katharopoulos et al. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. 2020.

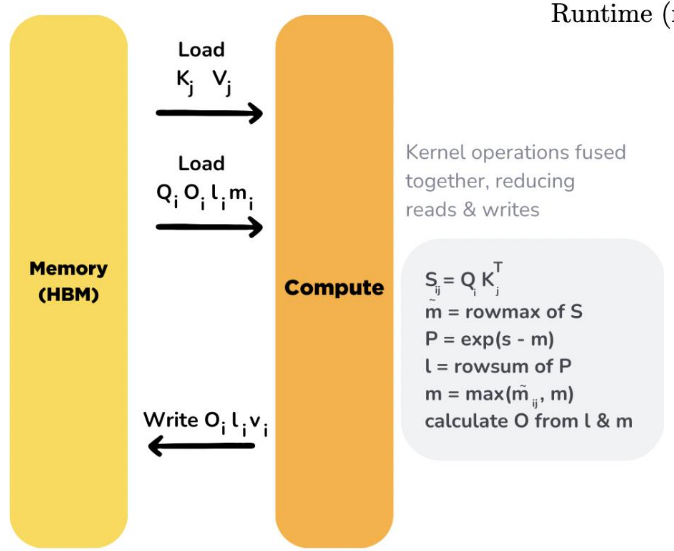
Flash Attention

Attention	Standard	FlashAttention
GFLOPs	66.6	75.2
HBM R/W (GB)	40.3	4.4
Runtime (ms)	41.7	7.3

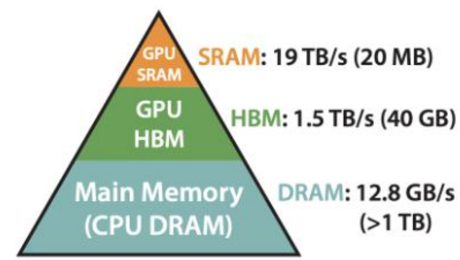
Standard Attention Implementation



Flash Attention



Initialize O, l and m matrices with zeroes. m and l are used to calculate cumulative softmax. Divide Q, K, V into blocks (due to SRAM's memory limits) and iterate over them, for i is row & j is column.



Memory Hierarchy with Bandwidth & Memory Size

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

https://huggingface.co/docs/text-generation-inference/en/conceptual/flash_attention

Flash Attention

Without Tiling 32 access

$$
\overset{A}{\begin{bmatrix} 1 & 2 & 0 & 5 \\ 4 & 8 & 2 & -1 \\ 3 & 1 & 6 & 3 \\ -7 & 5 & 0 & 8 \end{bmatrix}}
\times
\overset{B}{\begin{bmatrix} 3 & 1 & 6 & 3 \\ -7 & 5 & 0 & 8 \\ 1 & 2 & 0 & 5 \\ 0 & 3 & 5 & 1 \end{bmatrix}}
=
\overset{C}{\begin{bmatrix} C1,1 & C1,2 & C1,3 & C1,4 \\ C2,1 & C2,2 & C2,3 & C2,4 \\ C3,1 & C3,2 & C3,3 & C3,4 \\ C4,1 & C4,2 & C4,3 & C4,4 \end{bmatrix}}
$$

$C_{1,1} = 1 \times 1 + 2 \times 5 + 3 \times 9 + 4 \times 13$

$C_{1,2} = 1 \times 2 + 2 \times 6 + 3 \times 10 + 4 \times 14$

$C_{2,1} = 5 \times 1 + 6 \times 5 + 7 \times 9 + 8 \times 13$

$C_{2,2} = 5 \times 2 + 6 \times 6 + 7 \times 10 + 8 \times 14$

……………………………………

……………………………………

With Tiling 16 access

$$
\overset{A}{\left[\begin{array}{cc|cc} 1 & 2 & 0 & 5 \\ 4 & 8 & 2 & -1 \\ \hline 3 & 1 & 6 & 3 \\ -7 & 5 & 0 & 8 \end{array}\right]}
\times
\overset{B}{\left[\begin{array}{cc|cc} 3 & 1 & 6 & 3 \\ -7 & 5 & 0 & 8 \\ \hline 1 & 2 & 0 & 5 \\ 0 & 3 & 5 & 1 \end{array}\right]}
=
\overset{C}{\begin{bmatrix} C1,1 & C1,2 \\ C2,1 & C2,2 \end{bmatrix}}
$$

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

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

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

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

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

How FlashAttention Accelerates Generative AI Revolution: https://www.youtube.com/watch?v=gBMO1JZav44

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

$$m_i \leftarrow \max(m_{i-1}, x_i)$$

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

$$d_i \leftarrow d_{i-1} + e^{x_i - m_N}$$

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

$$a_i \leftarrow \frac{e^{x_i - m_N}}{d'_N}$$

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

https://courses.cs.washington.edu/courses/cse599m/23sp/notes/flashattn.pdf

Flash Attention

Fused computation to one loop!

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

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

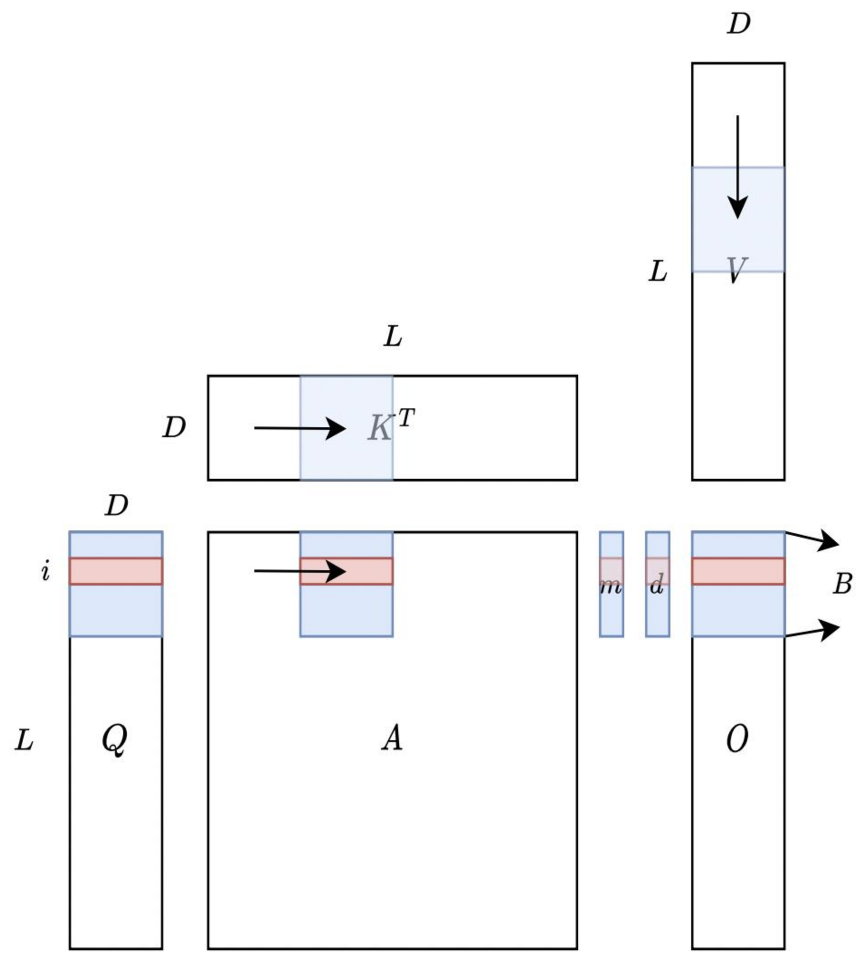
end

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

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

https://courses.cs.washington.edu/courses/cse599m/23sp/notes/flashattn.pdf

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.

https://courses.cs.washington.edu/courses/cse599m/23sp/notes/flashattn.pdf

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

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

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:

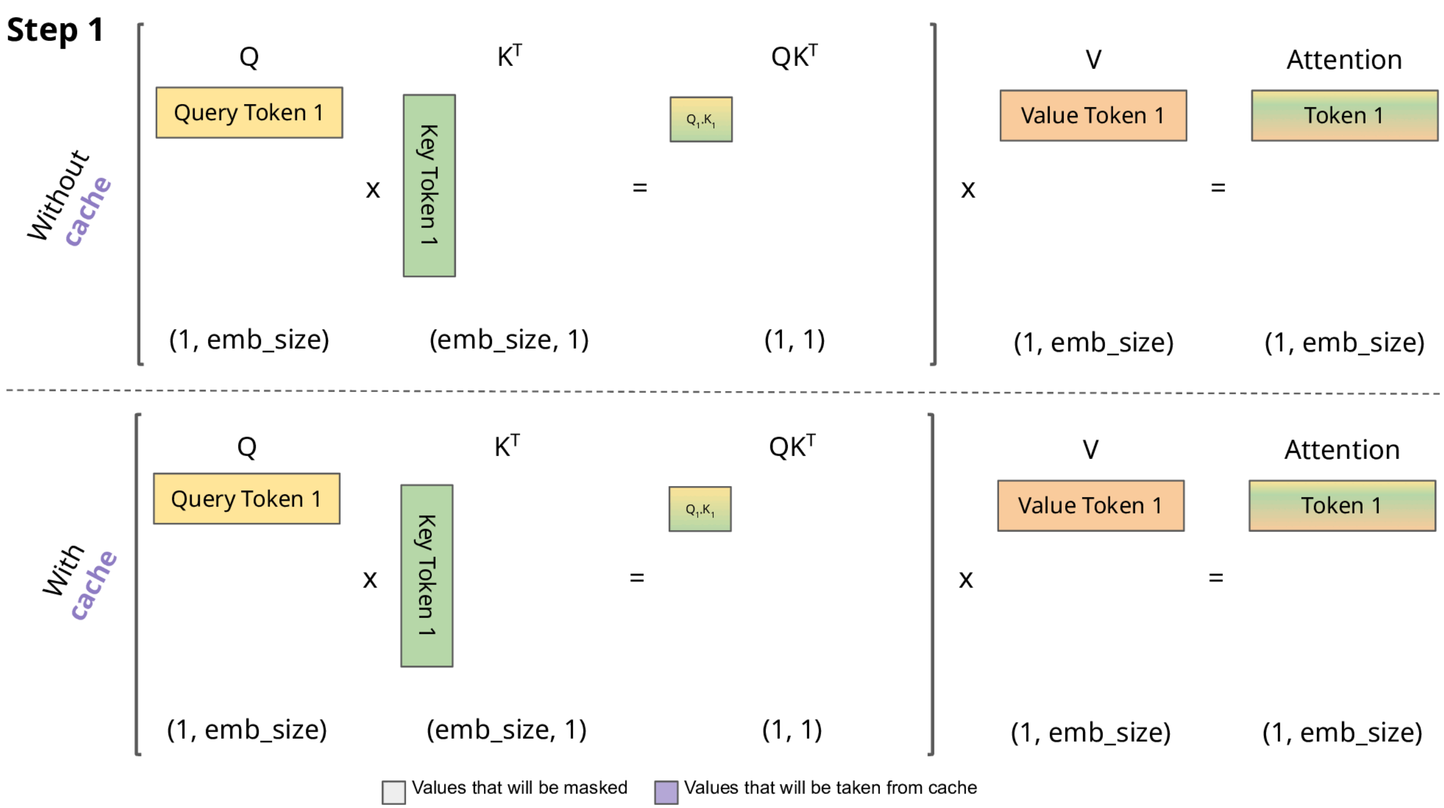
Flash Attention: $\mathbf{O}\left(\frac{\boldsymbol{N^2 d^2}}{\boldsymbol{M}}\right)$

Standard Attention: $\boldsymbol{\Omega}\left(\boldsymbol{Nd} + \boldsymbol{N^2}\right)$

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

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

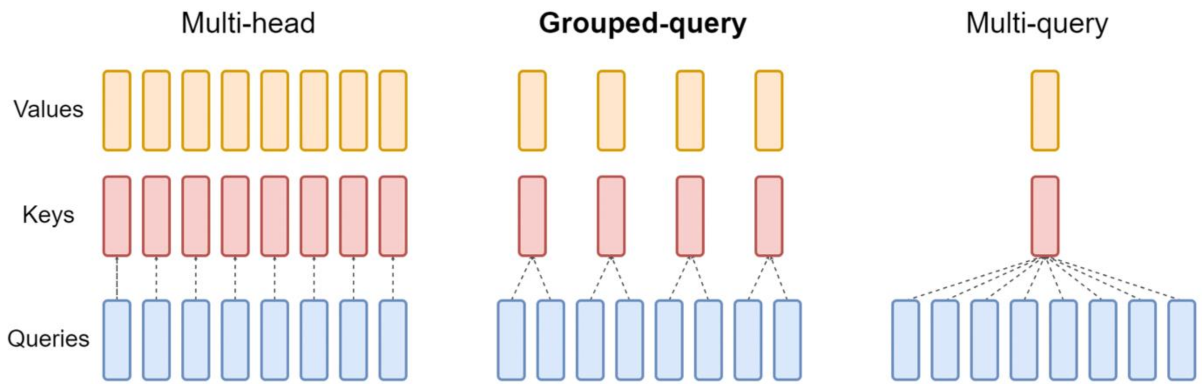
KV- Caching



Step 1
Without cache
Q
Query Token 1
(1, emb_size)
x
K^T
Key Token 1
(emb_size, 1)
=
QK^T
$Q_1.K_1$
(1, 1)
x
V
Value Token 1
(1, emb_size)
=
Attention
Token 1
(1, emb_size)
With cache
Q
Query Token 1
(1, emb_size)
x
K^T
Key Token 1
(emb_size, 1)
=
QK^T
$Q_1.K_1$
(1, 1)
x
V
Value Token 1
(1, emb_size)
=
Attention
Token 1
(1, emb_size)
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.

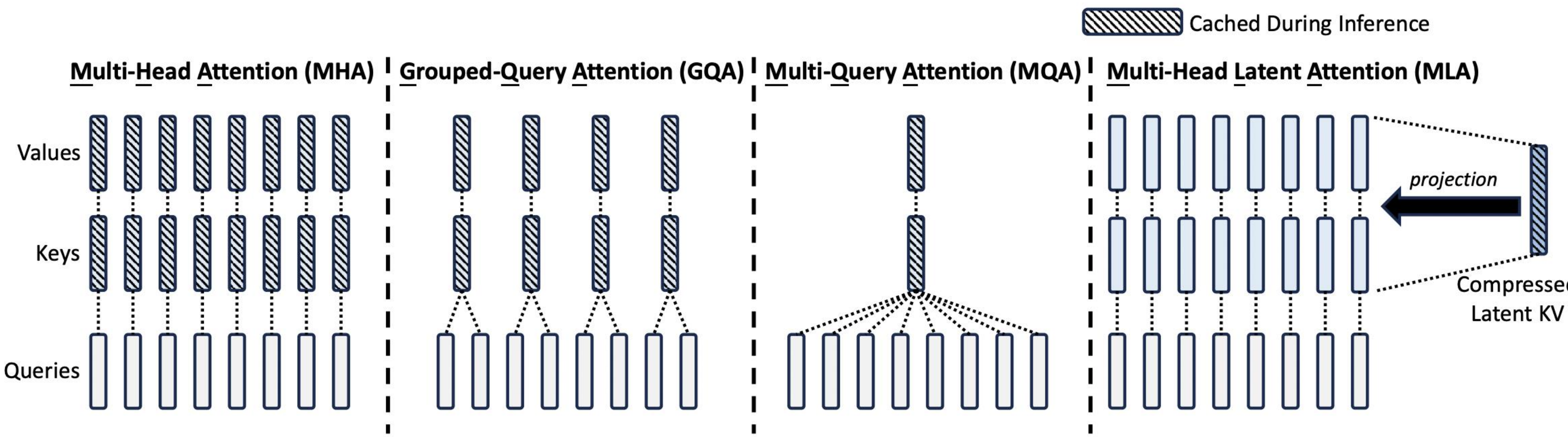


Benchmark (Metric)	# Shots	Dense 7B w/ MQA	Dense 7B w/ GQA (8 Groups)	Dense 7B w/ MHA
# Params	-	7.1B	6.9B	6.9B
BBH (EM)	3-shot	33.2	35.6	**37.0**
MMLU (Acc.)	5-shot	37.9	41.2	**45.2**
C-Eval (Acc.)	5-shot	30.0	37.7	**42.9**
CMMLU (Acc.)	5-shot	34.6	38.4	**43.5**

Ainslie, Joshua, et al. "Gqa: Training generalized multi-query transformer models from multi-head checkpoints." arXiv preprint arXiv:2305.13245 (2023).

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**

Liu, Aixin, et al. "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model." arXiv preprint arXiv:2405.04434 (2024)

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

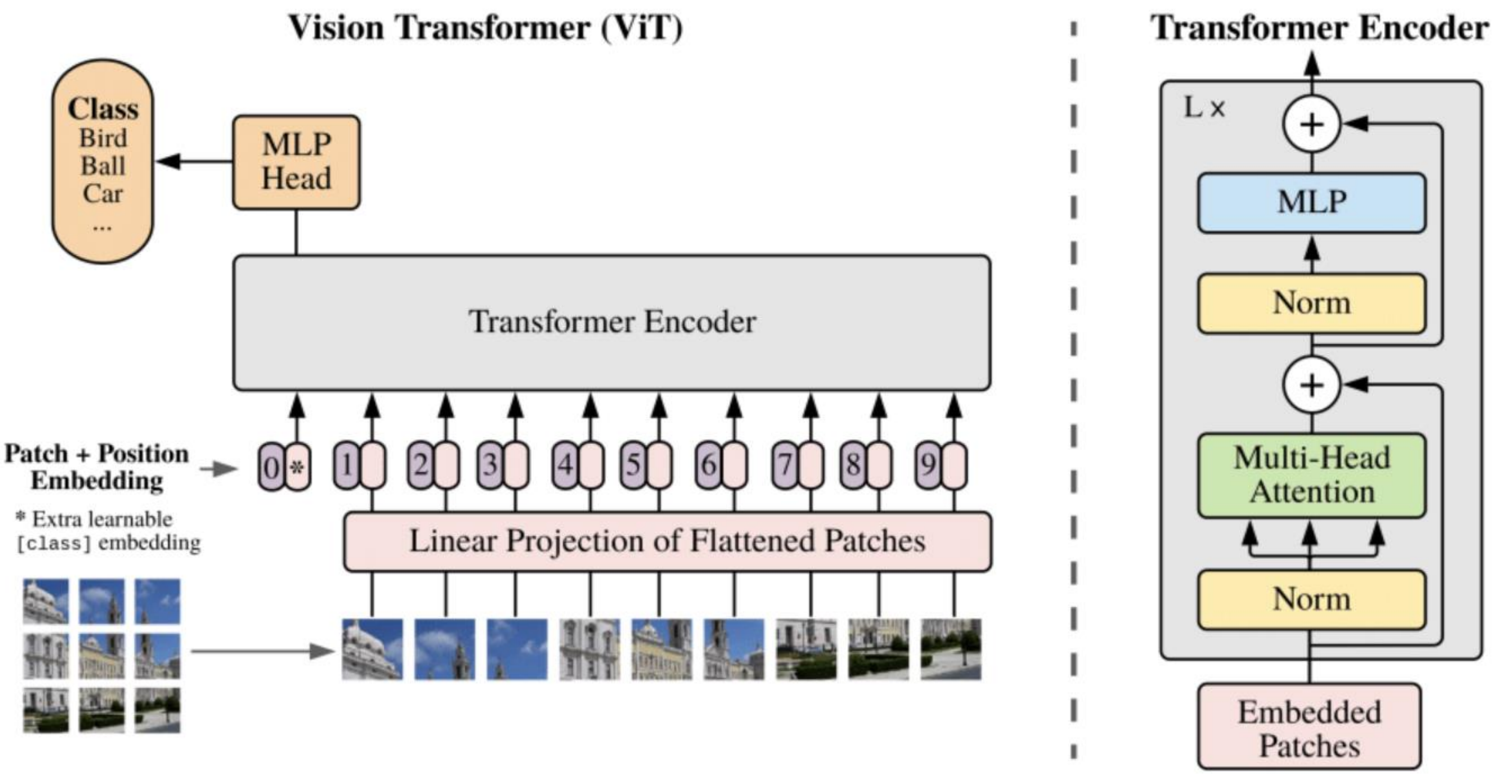
Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizer

Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizers

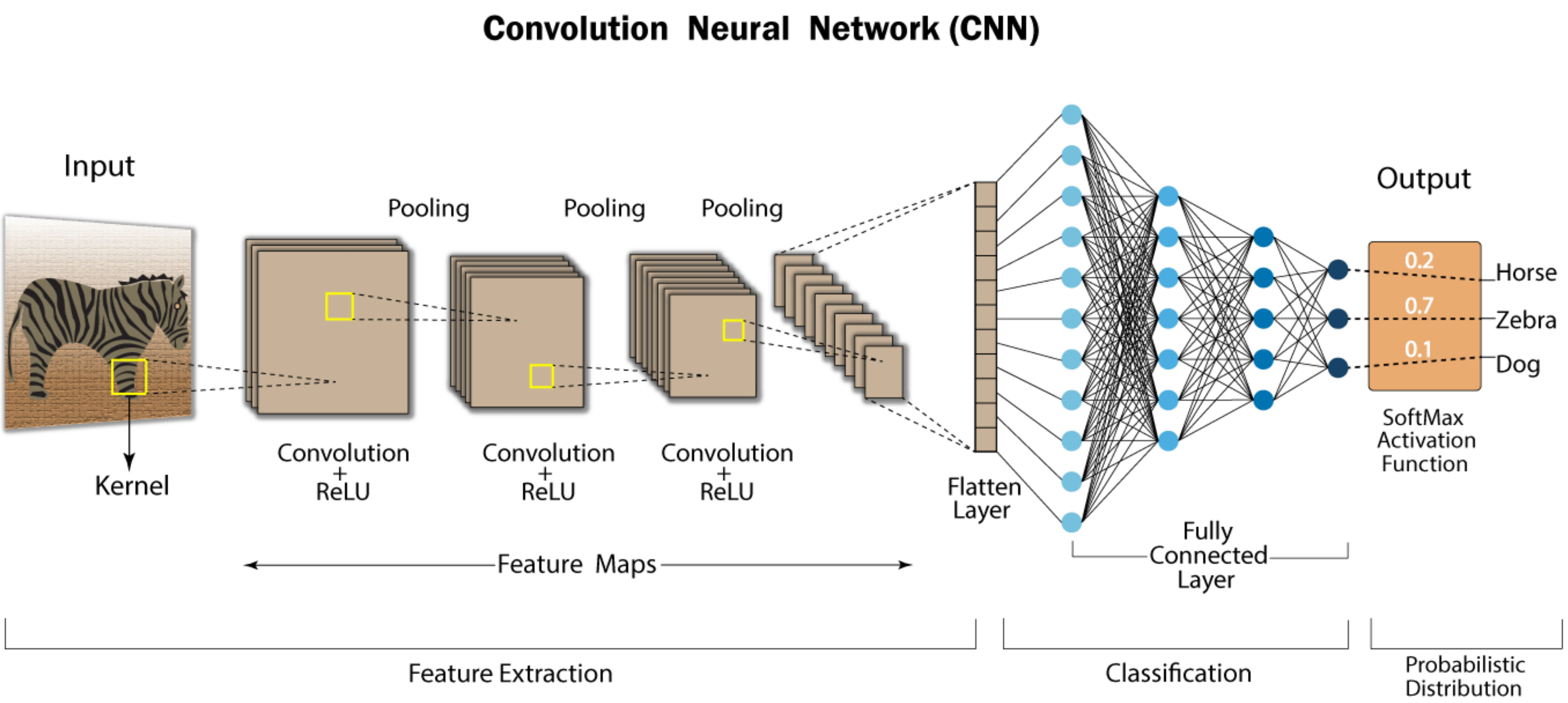
Vision Transformer (ViT)



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

Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

CNN



- Naturally fits to 2D images

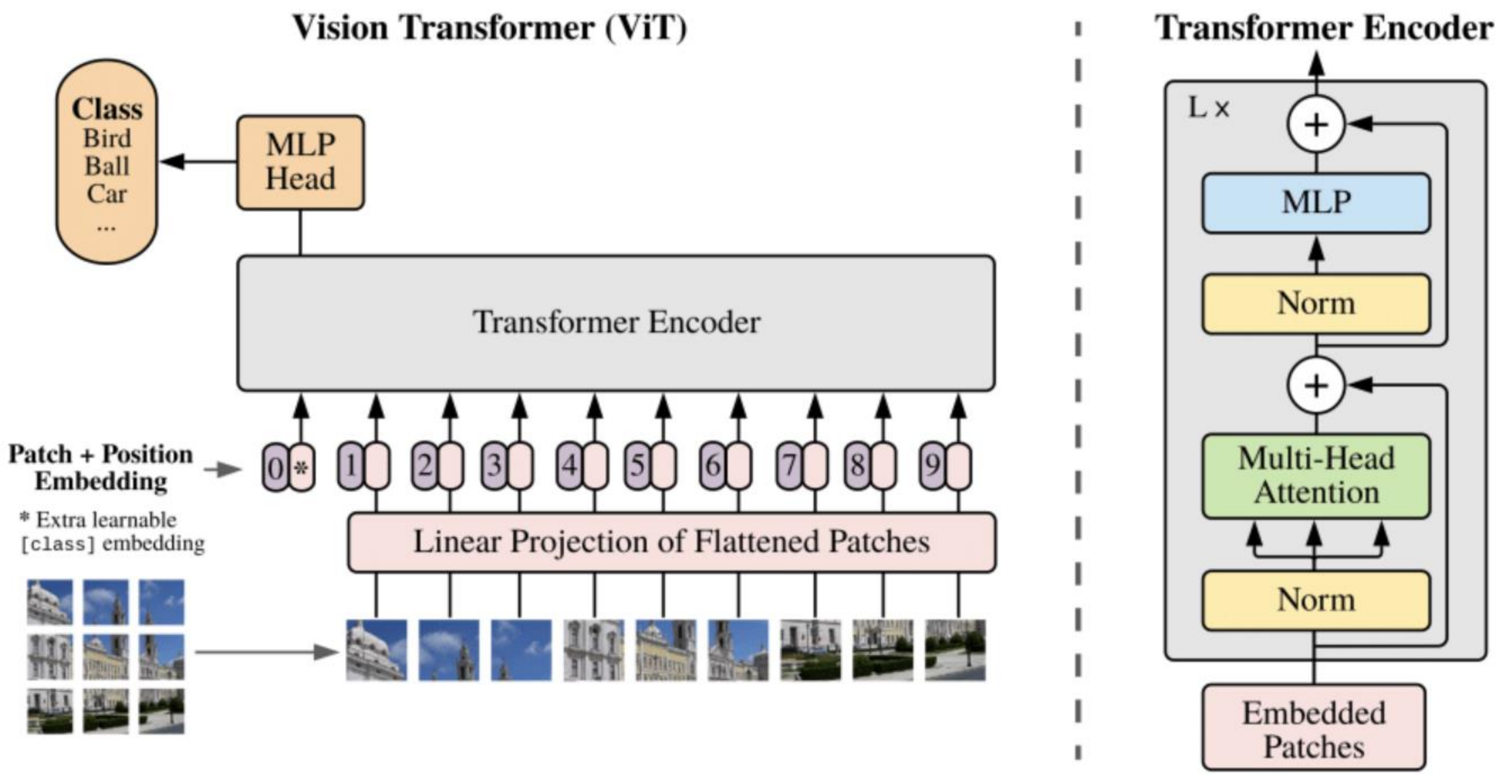
ViT

- 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

Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

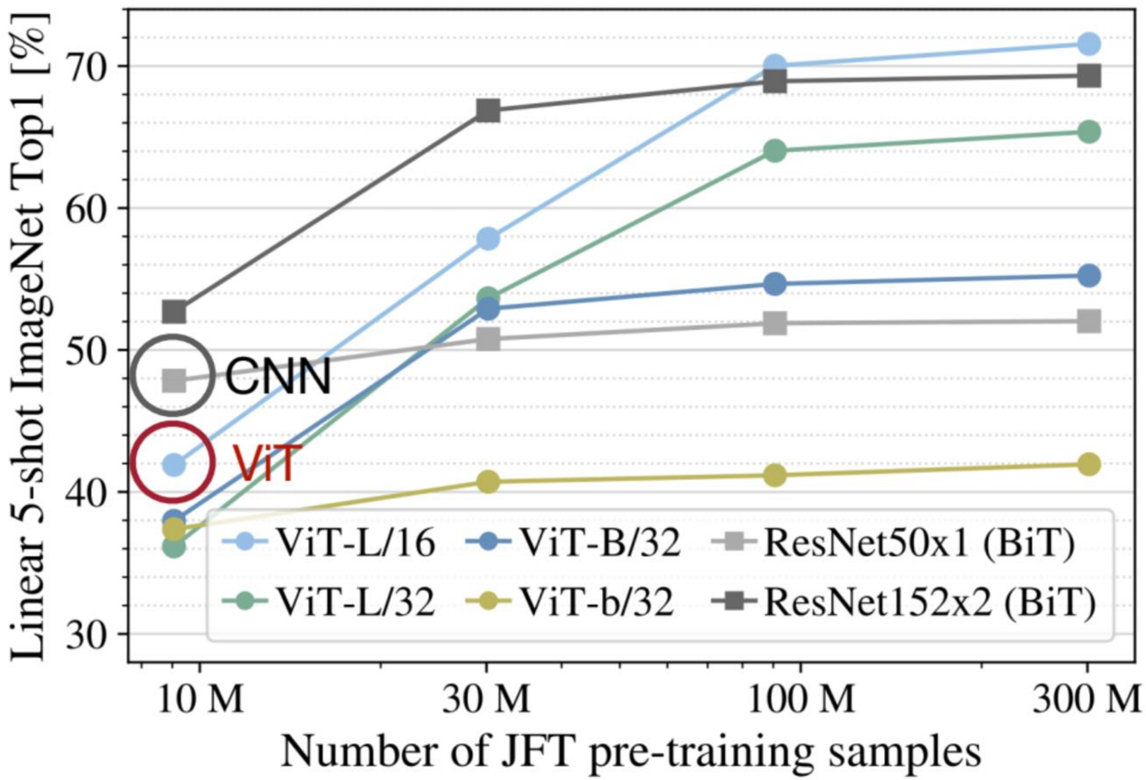
ViT



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

Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

Image Classification

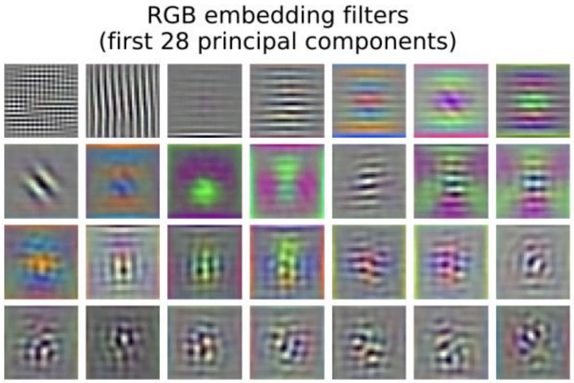


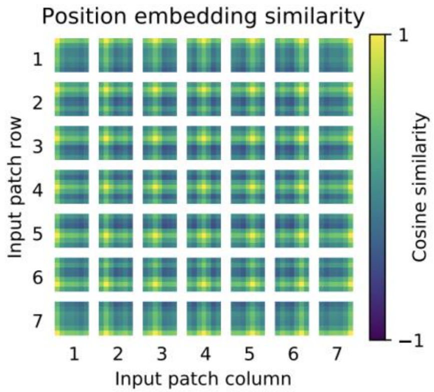
- Inferior performance compared to CNN when dataset size is limited – Why?

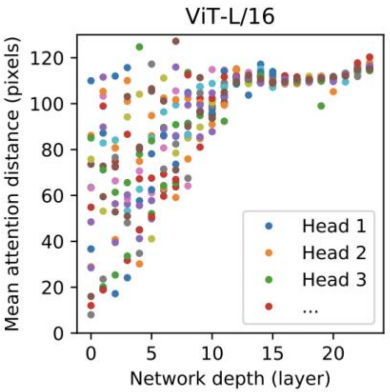
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

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!





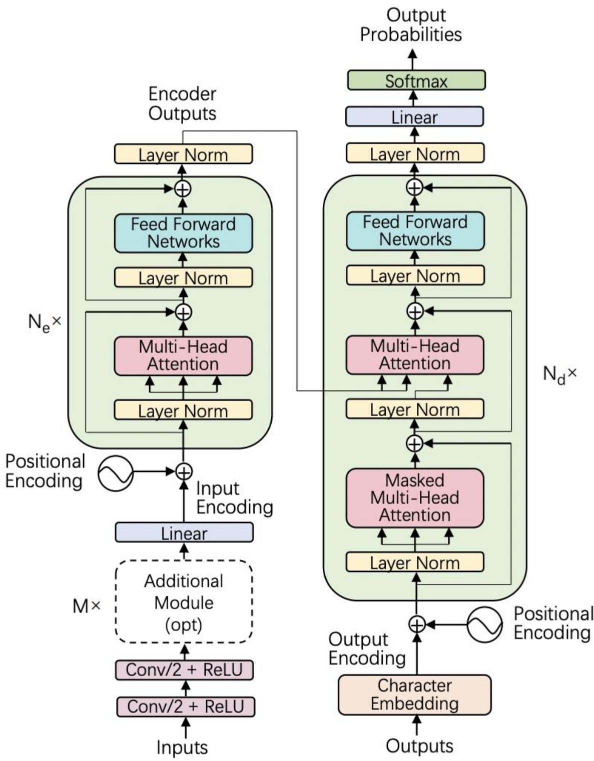


Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

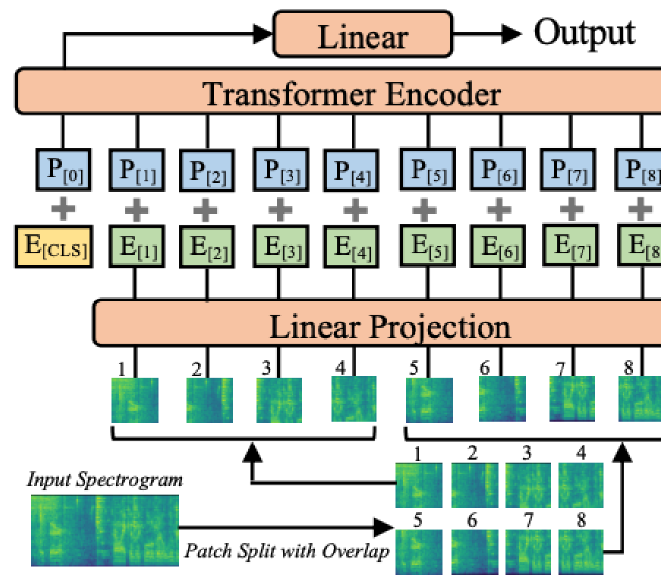
Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizer

Transformer in Audio



Speech Transformer for ASR



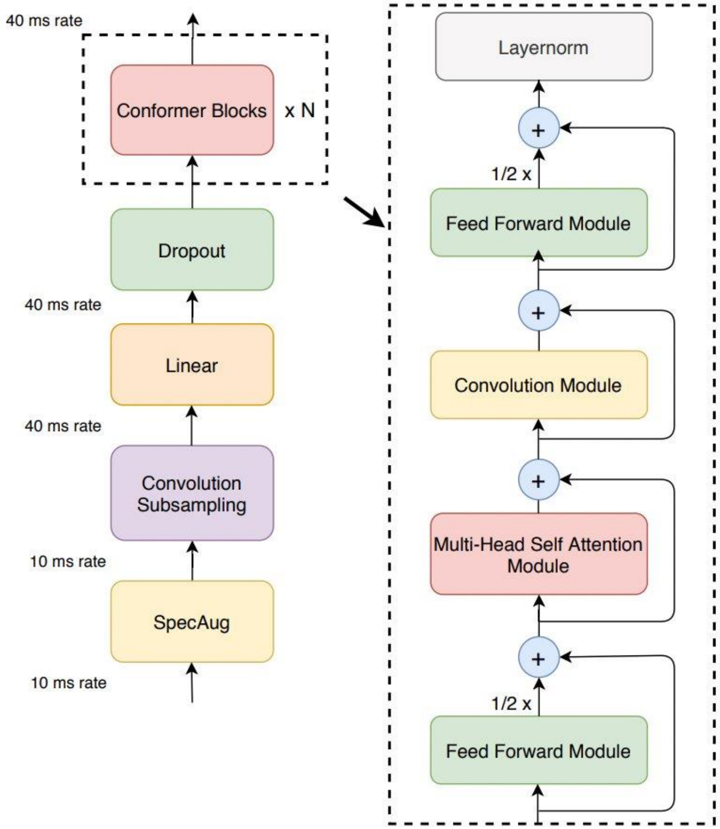
Audio Spectrogram Transformer

[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.

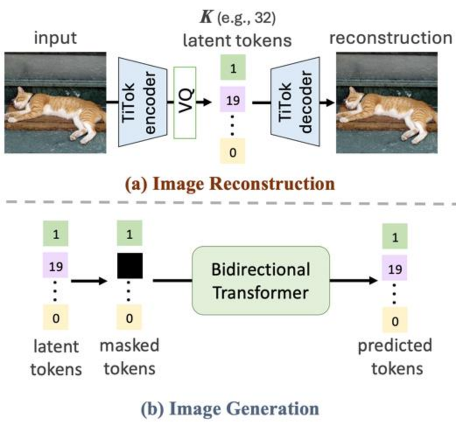


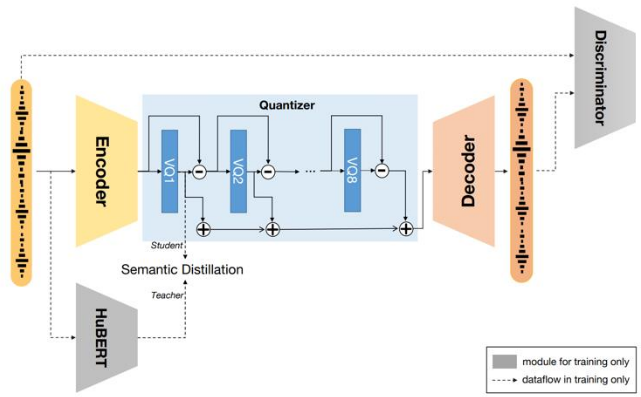
Gulati, Anmol, et al. "Conformer: Convolution-augmented transformer for speech recognition." *arXiv preprint arXiv:2005.08100* (2020).

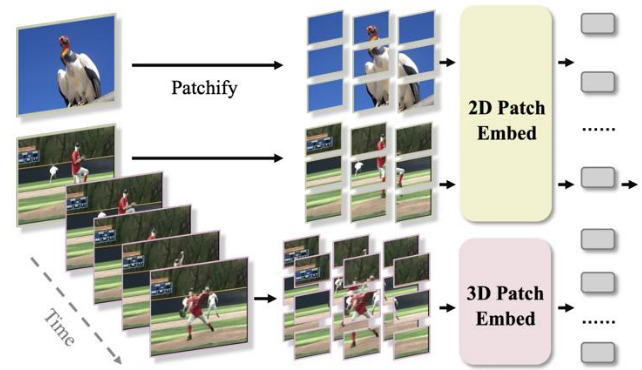
Overview

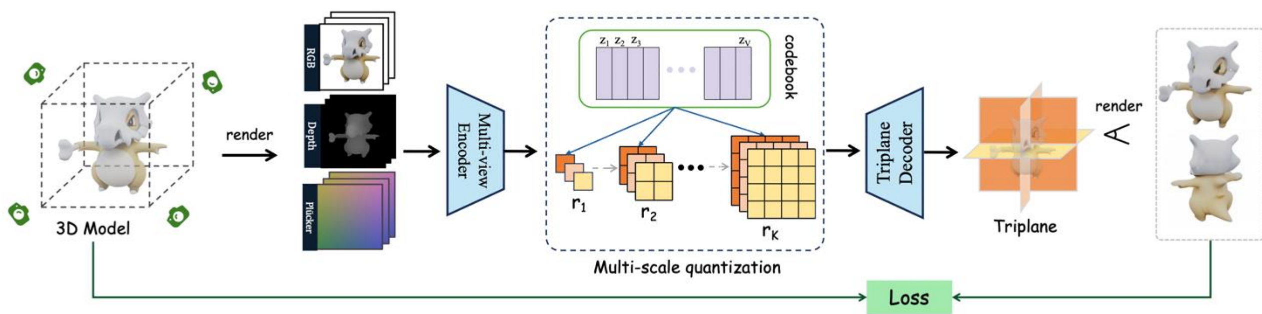
- Vision Transformer Architecture
- Transformer in Audio: Conformer
- Tokenizer

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

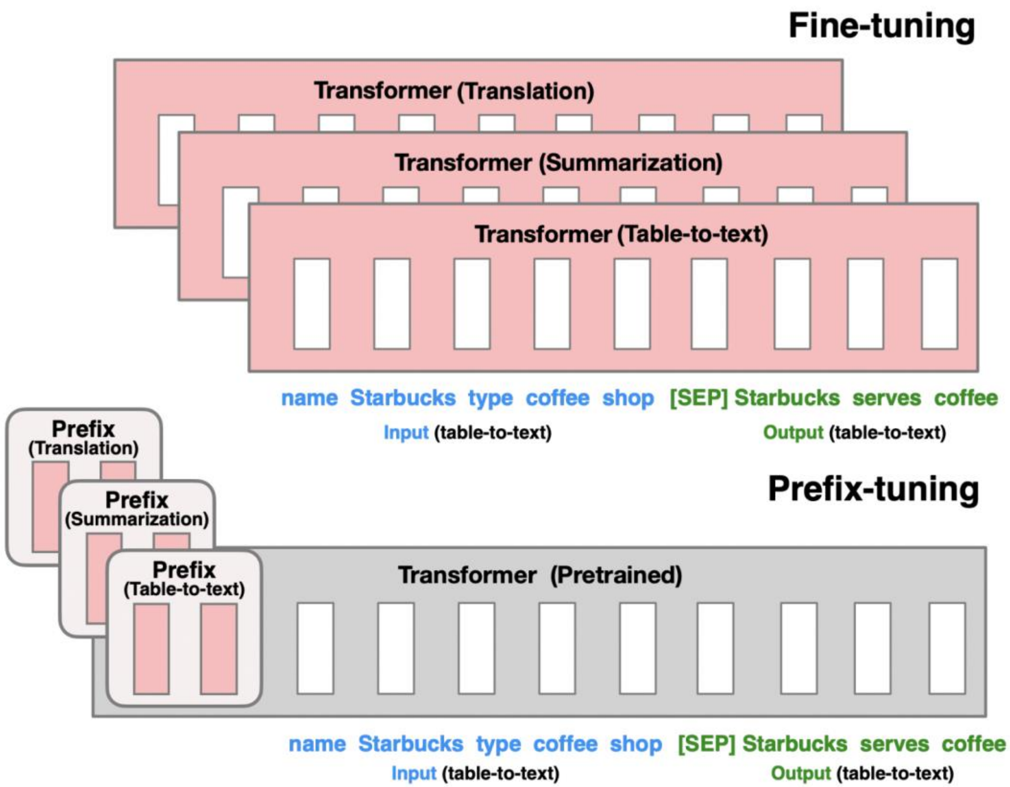
Parameter Efficient Tuning

- 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

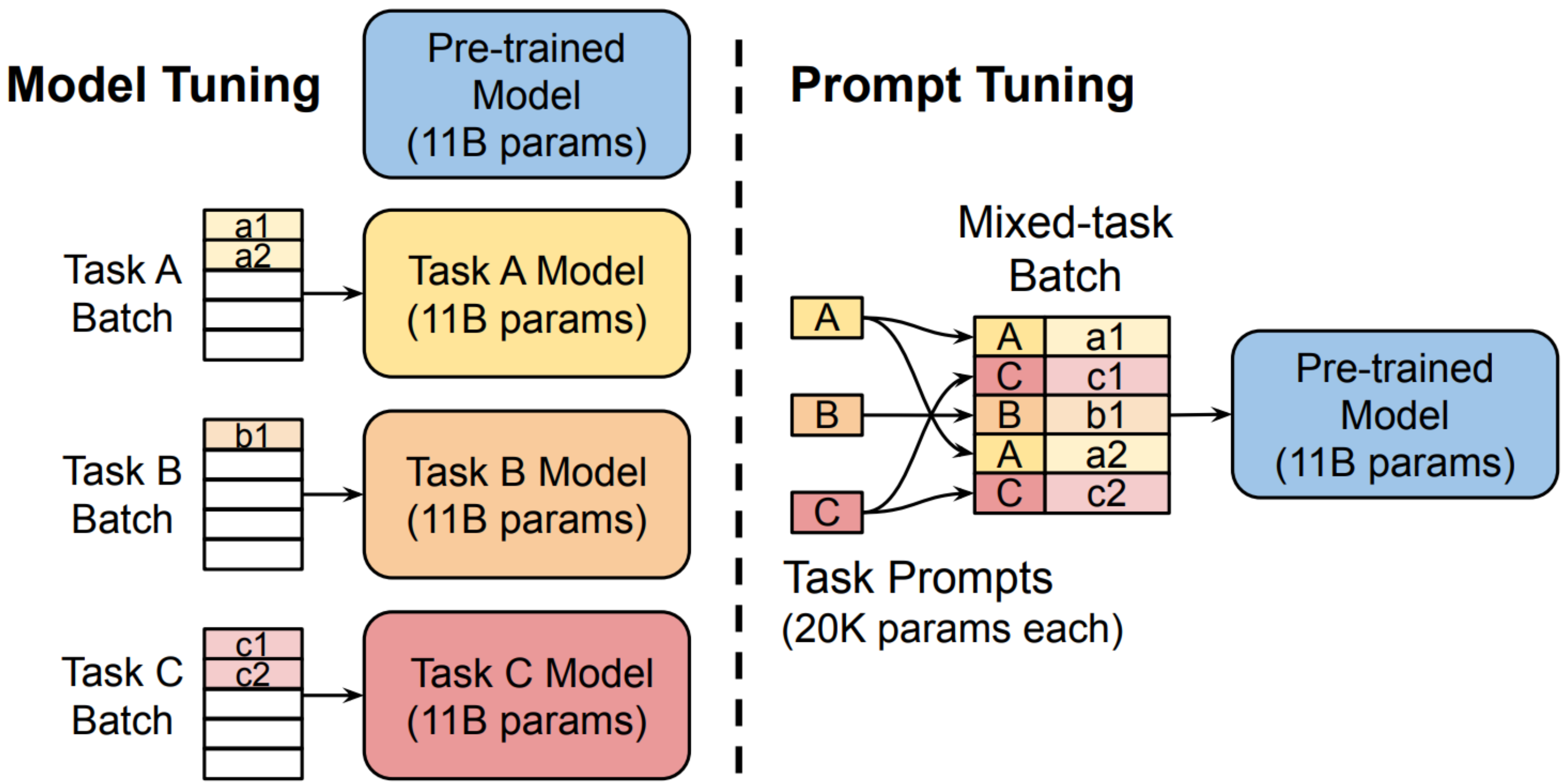
- Only learns a set continuous prefixes)added to the input and transformer layers for each task.



Li, Xiang Lisa, and Percy Liang. "Prefix-tuning: Optimizing continuous prompts for generation." arXiv preprint arXiv:2101.00190 (2021).

Prompt Tuning

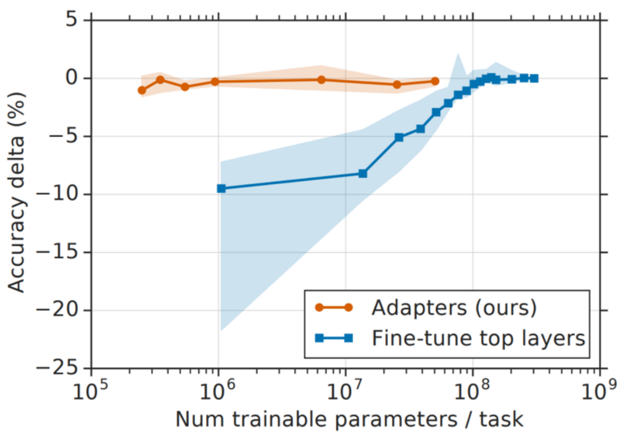
- Only learns a set of ‘prompt’ or ‘token’ for each task

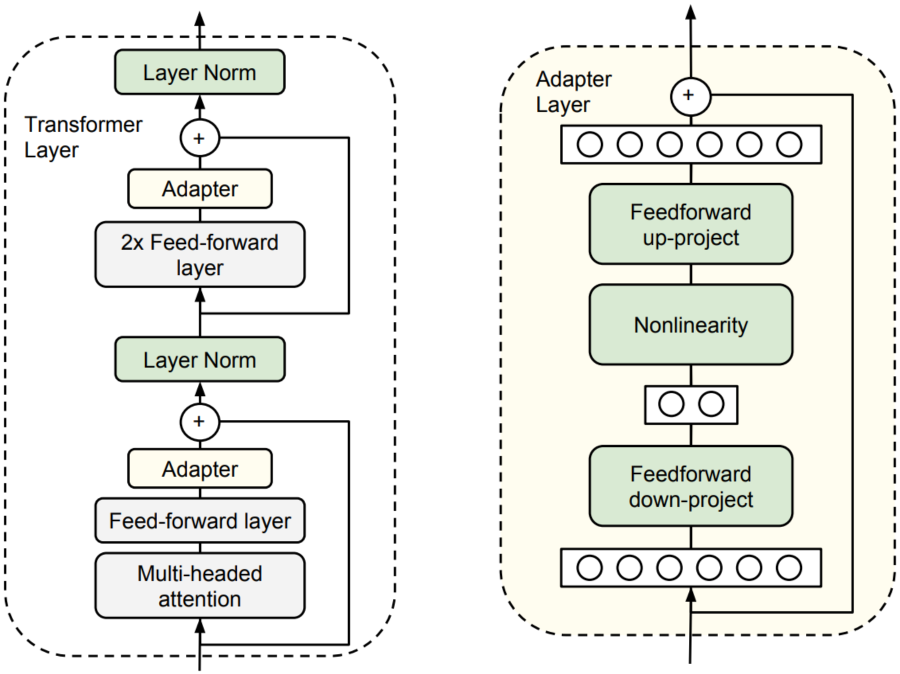


Lester et al. The Power of Scale for Parameter-Efficient Prompt Tuning. 2021.

Adapter

- Insert MLP at Feed-forward layers



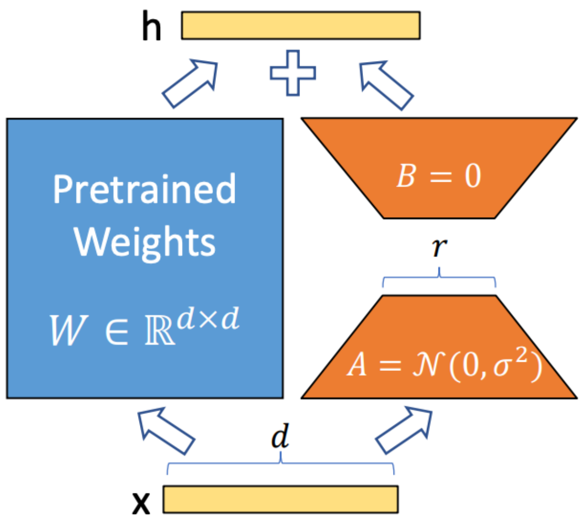


Houlsby et al. Parameter-Efficient Transfer Learning for NLP. 2022.

LoRA

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

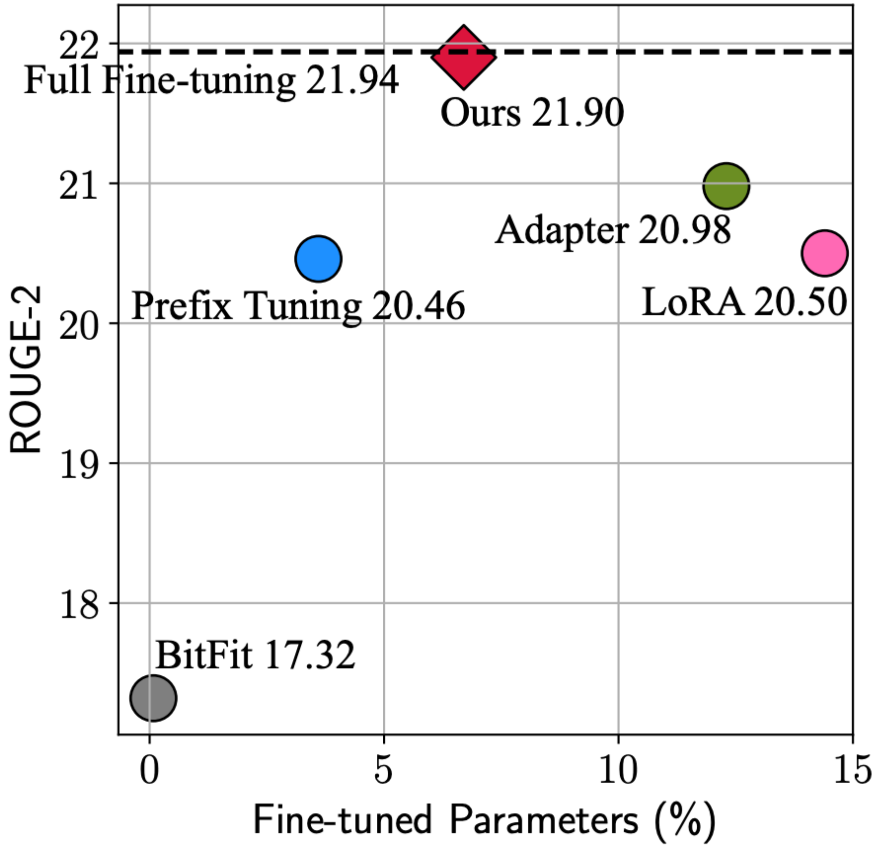
$$h = W_0 x + \Delta W x = W_0 x + BAx$$



Hu et al. LoRA: Low-Rank Adaptation of Large Language Models. 2022.

Parameter-Efficient Tuning

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



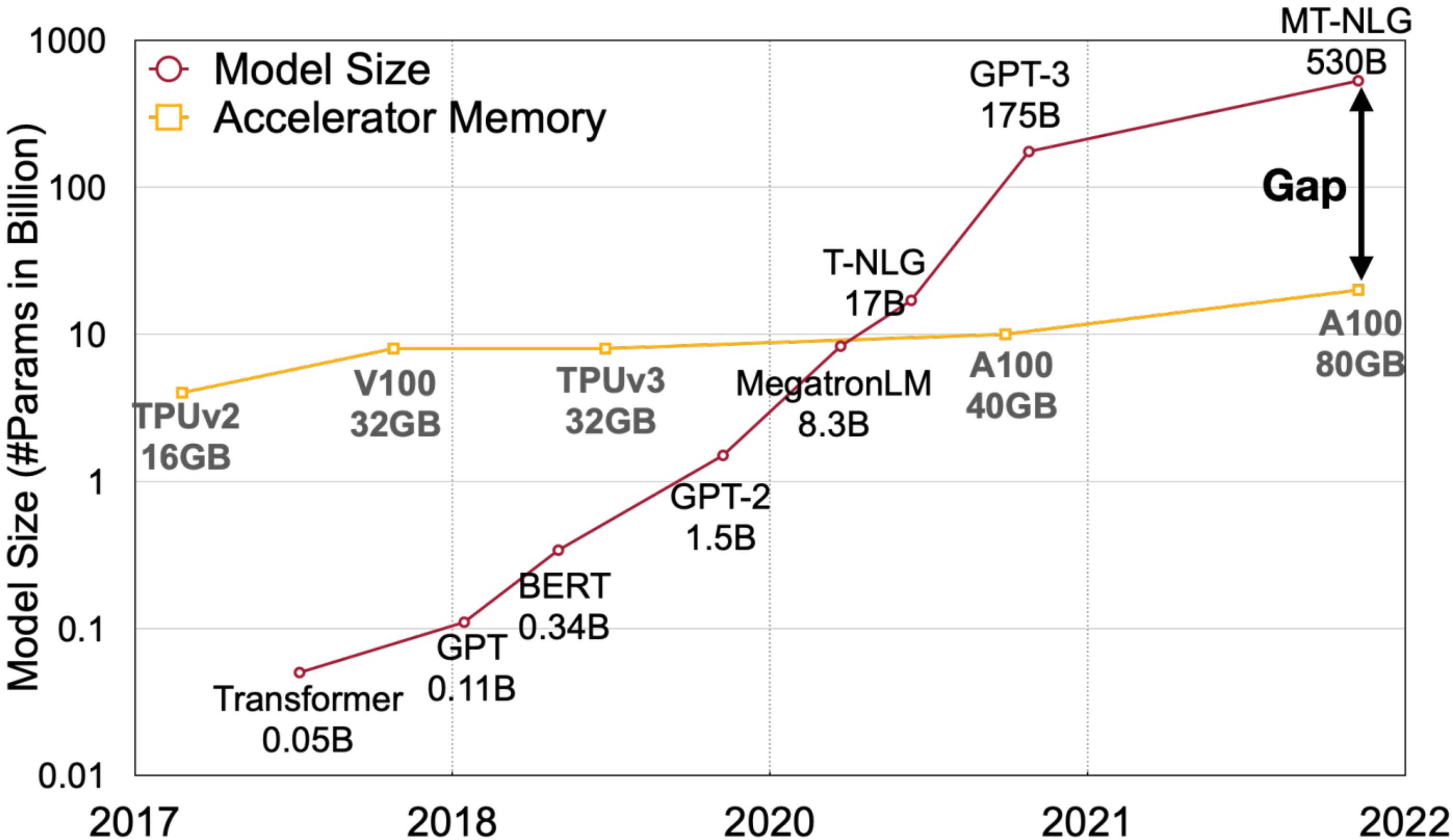
He, Junxian, et al. "Towards a unified view of parameter-efficient transfer learning." arXiv preprint arXiv:2110.04366 (2021).

Content

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

Scaling Laws

"Magic" of Transformer - Scaling

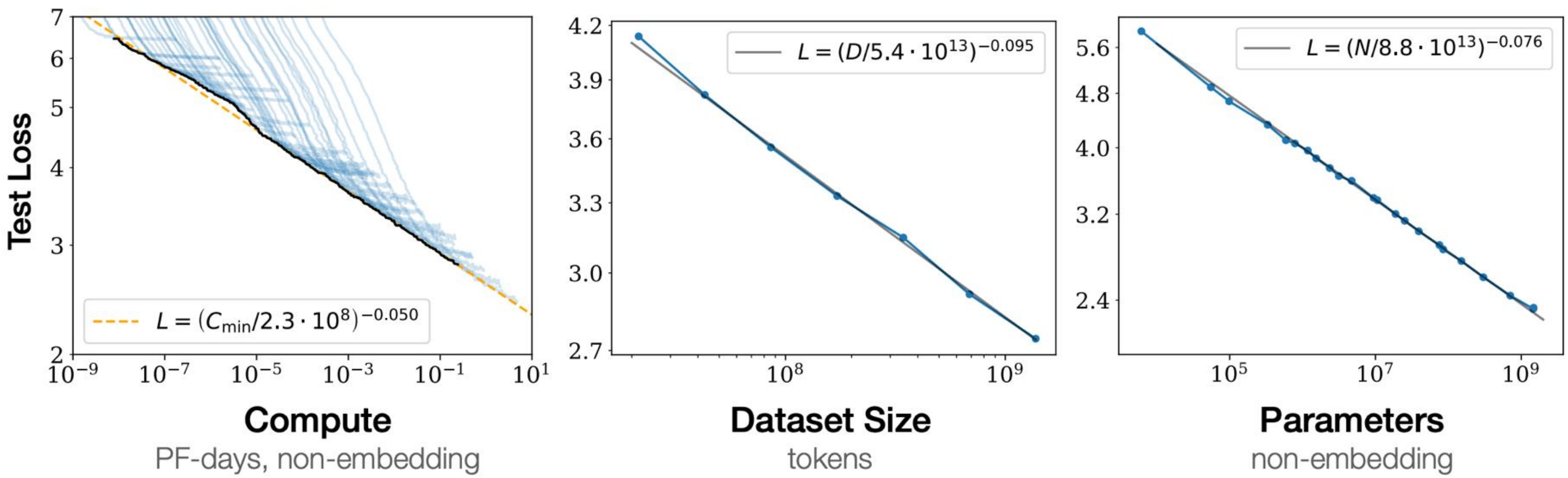


- Performance gets better as transformer scales up

Xiao et al. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. 2024.

Scaling Law

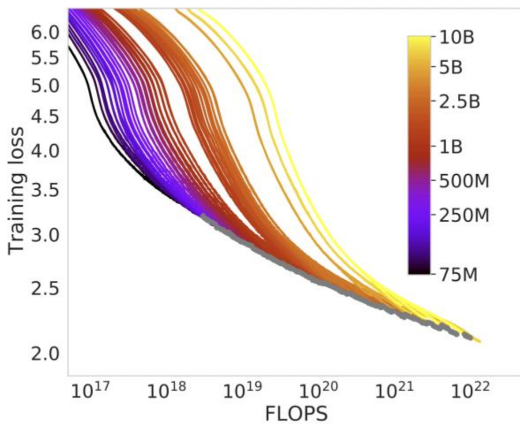
- For decoder-only models, the final performance is only related to **Compute**, **Data Size**, and **Parameter Size**
  - power law relationship for each factor
  - w/o constraints by the others



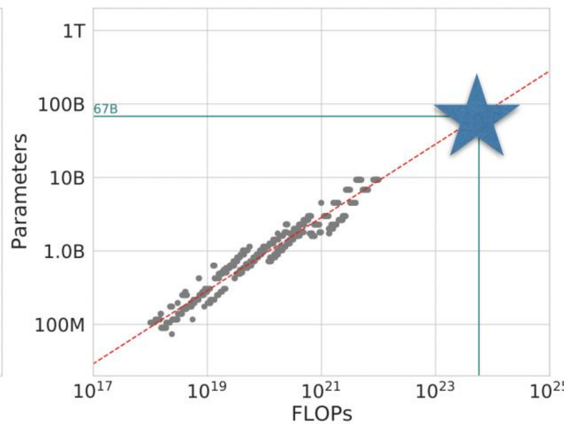
$$\text{PF-day} = 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19} \text{ floating point operations.}$$

Kaplan et al. Scaling Laws for Neural Language Models. 2020.

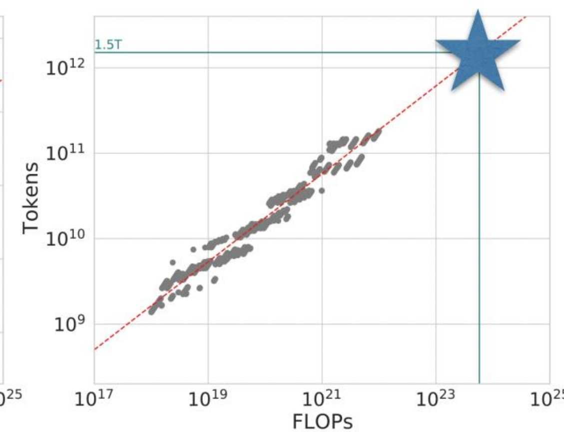
Scaling Law



Run experiments



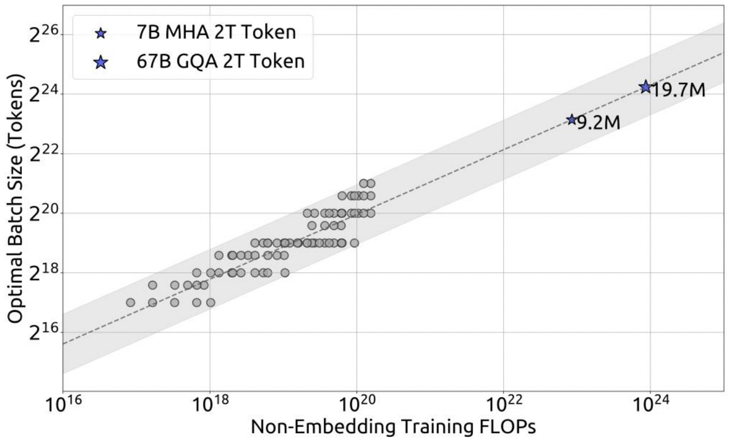
Fit a line and predict optimal model size



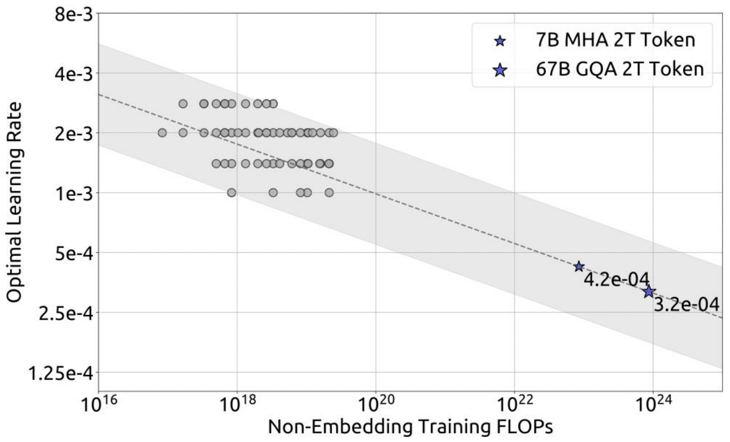
Fit a line and predict optimal # of tokens

Hoffmann, Jordan, et al. "Training compute-optimal large language models. arXiv 2022." arXiv preprint arXiv:2203.15556 10 (2022).

Scaling Law



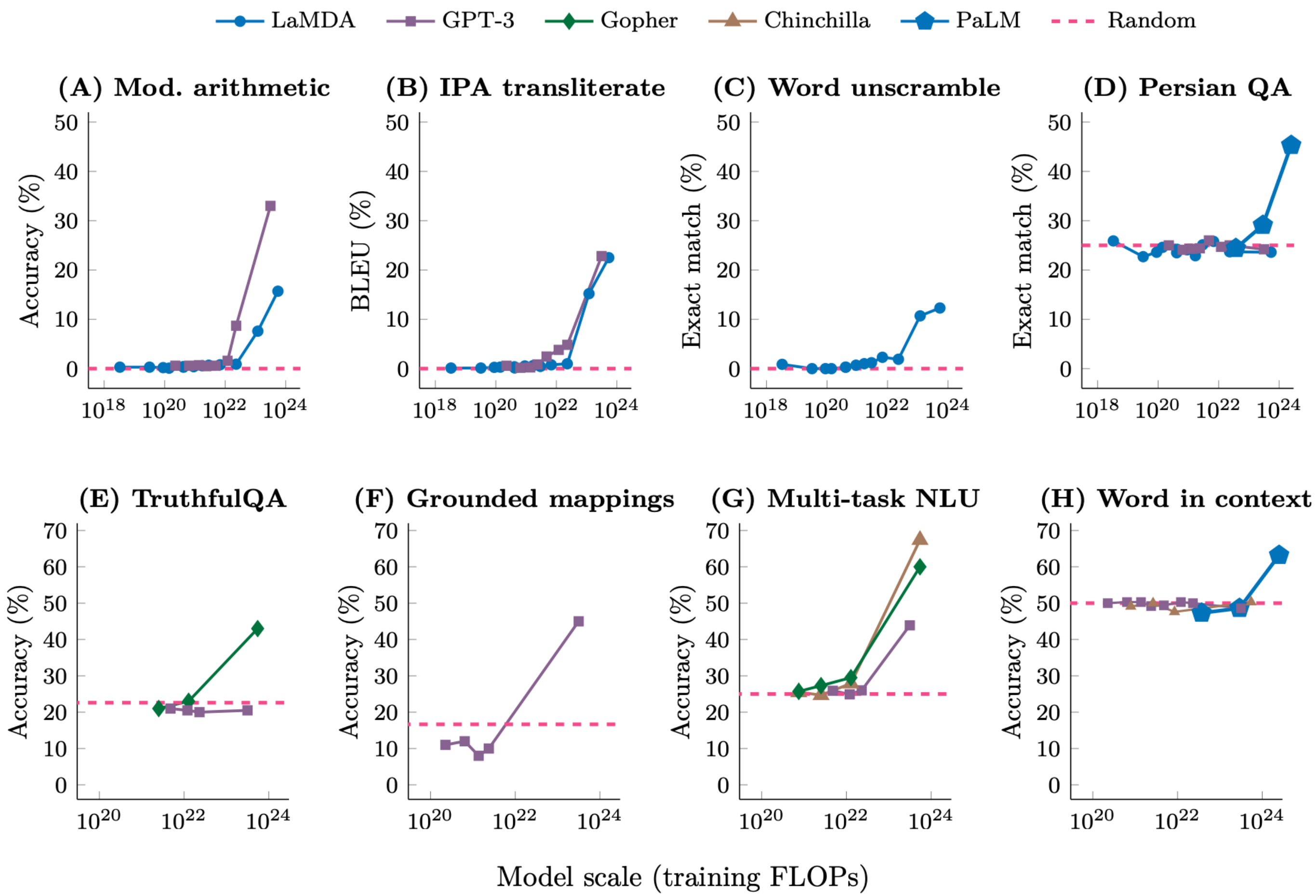
(a) Batch size scaling curve



(b) Learning rate scaling curve

Bi, Xiao, et al. "Deepseek llm: Scaling open-source language models with longtermism." arXiv preprint arXiv:2401.02954 (2024).

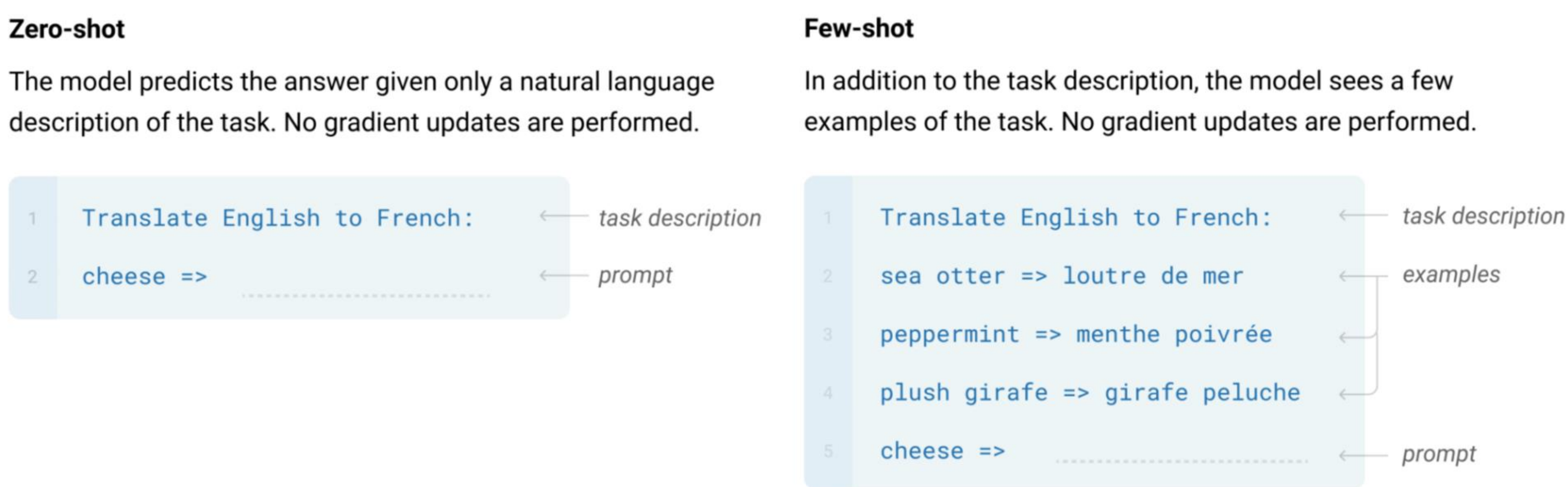
“Emergent” Capability



Wei et al. Emergent Abilities of Large Language Models. 2020.

In-Context Learning

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



Language Models are Few-shot Learners.

We learned…

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