Deep Learning

19: Transformer and Newer Architectures

Puru Samal
Special credits: Shrey Jain, Dareen Alharthi, Hao Chen

Spring 2026

Attendance:@TBD

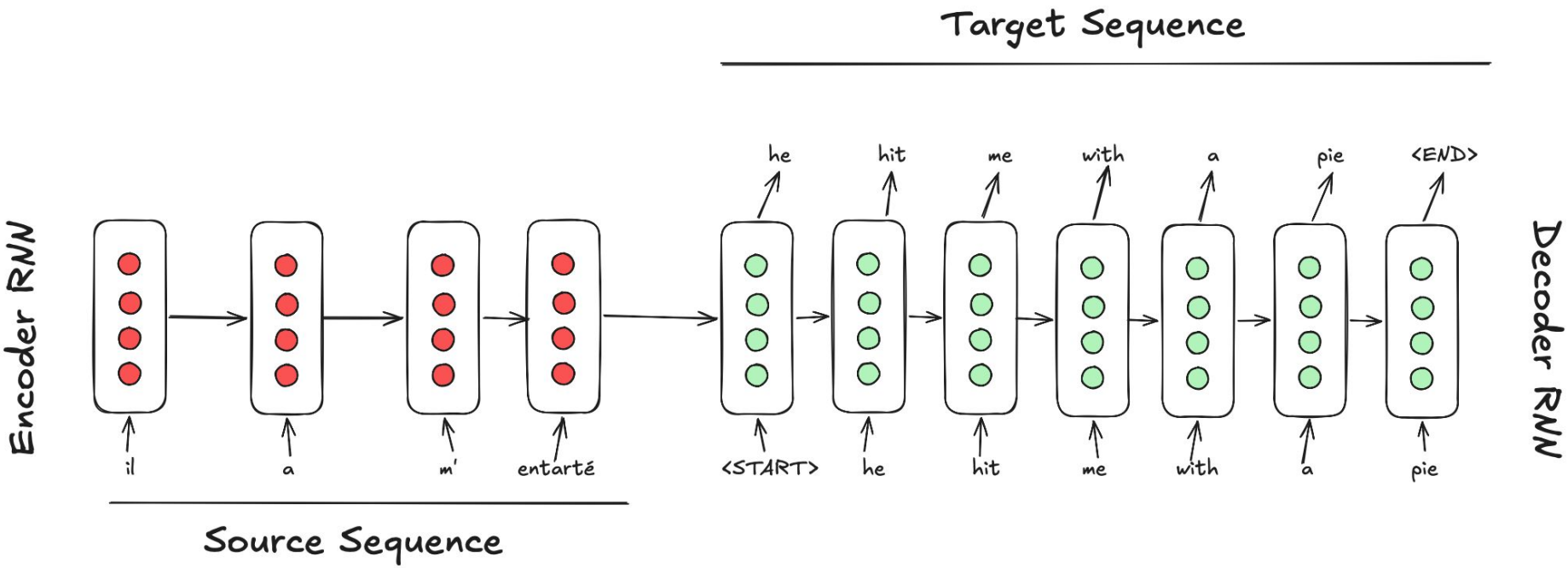
Content

- Transformer Architecture
- Improvements on Transformers
- Transformers across modalities
- Parameter Efficient Fine-Tuning
- Quantizing Transformers

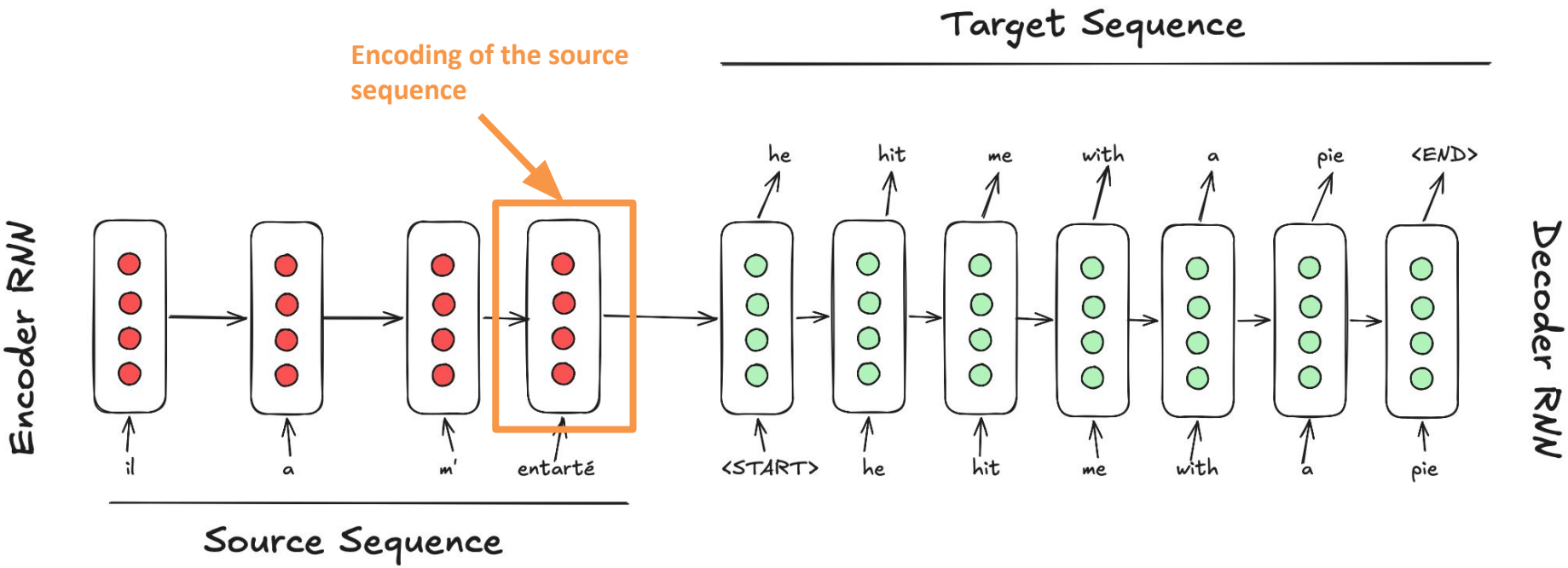
Content

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

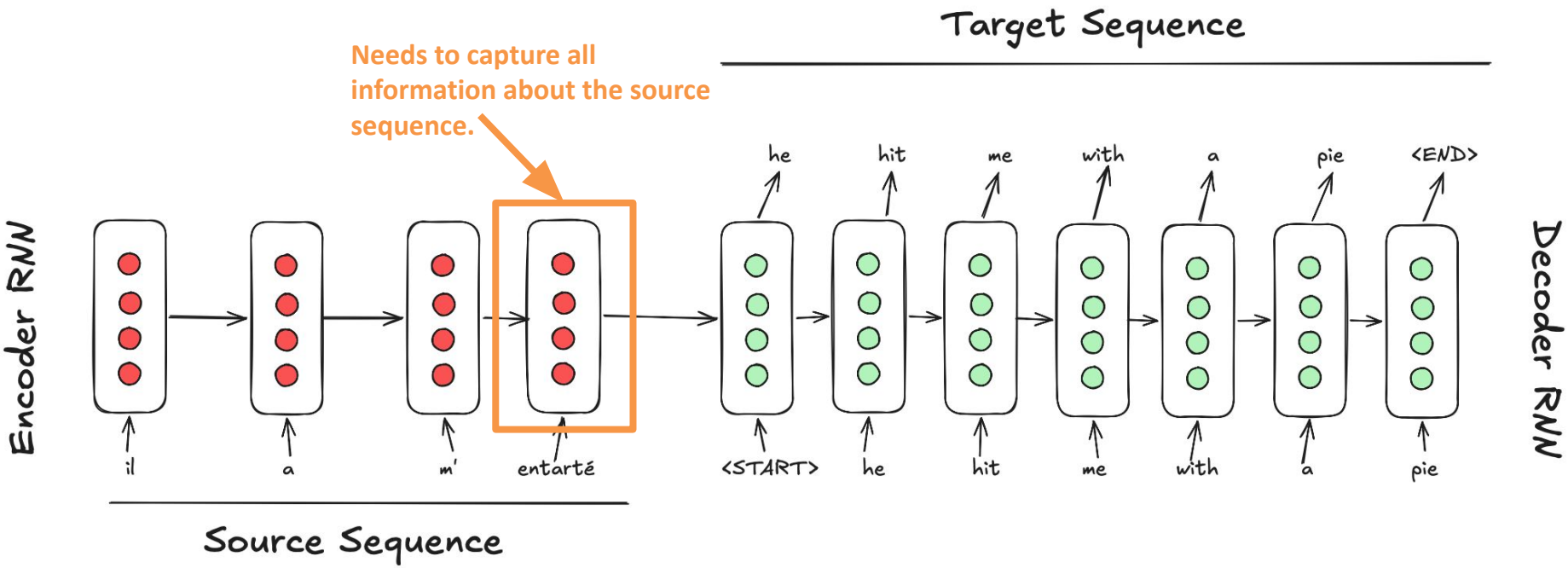
Issues with Recurrent Models



Issues with Recurrent Models

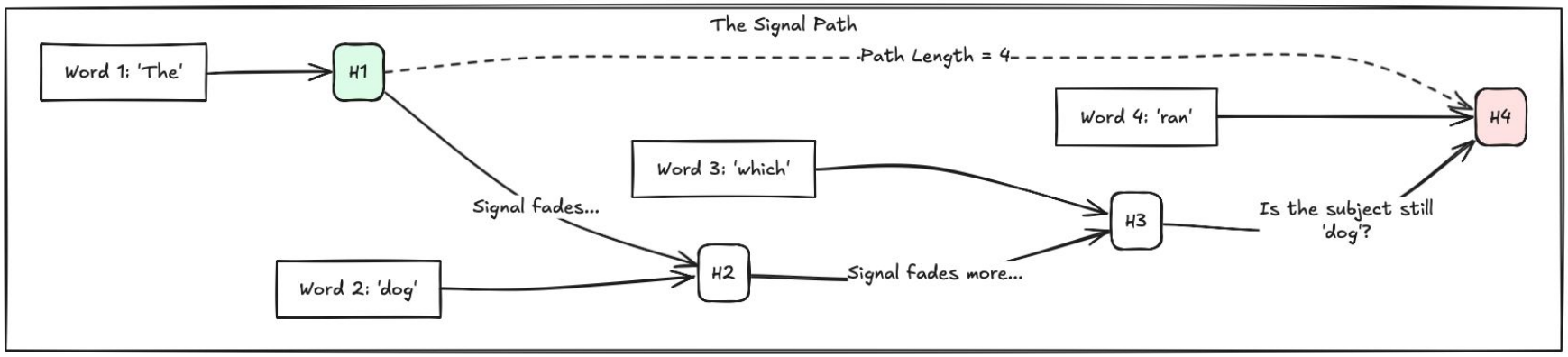


Issues with Recurrent Models



Information bottleneck!

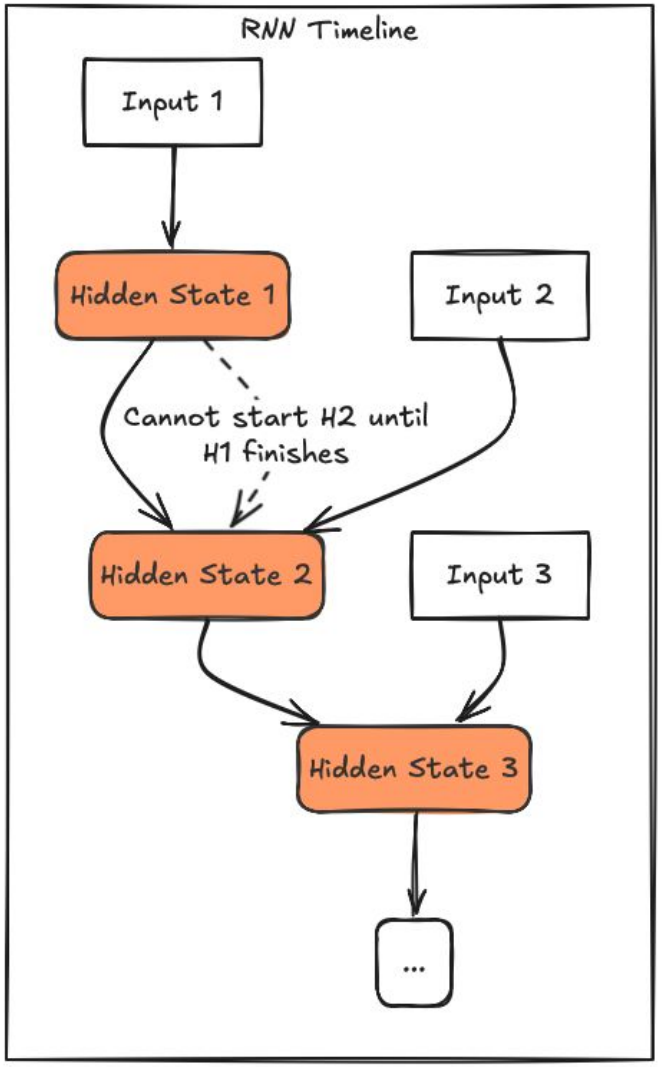
Issues 1: Linear Interaction Distance



The "Information Decay" Problem

- **O(n) Path Length:** Distant words require n steps to "talk" to each other.
- **Vanishing Gradients:** Signal weakens or distorts over long sequences, making long-distance dependencies hard to learn.
- **Baked-in Linearity:** Forces a strict sequential chain, ignoring the natural hierarchical structure of language.

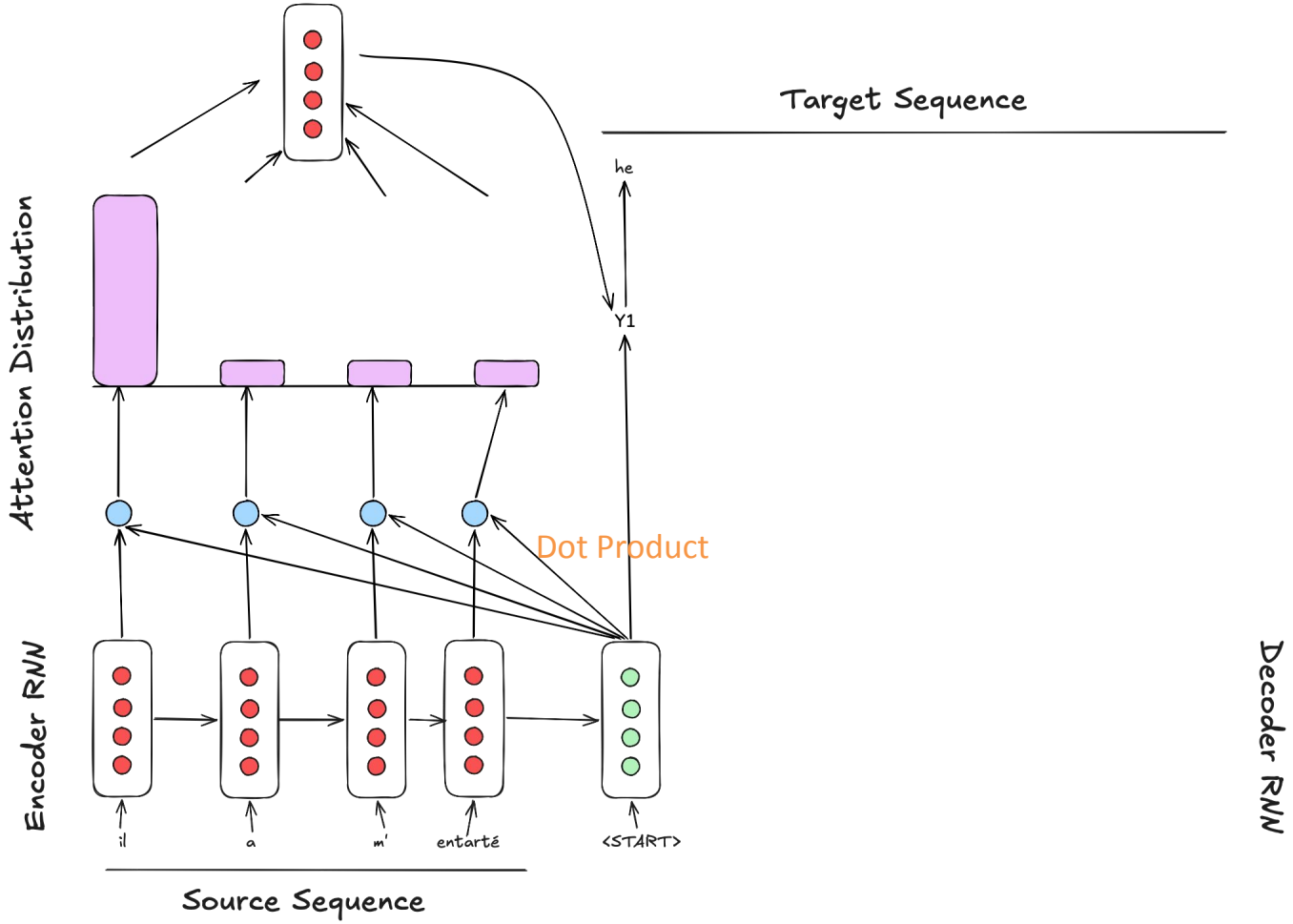
Issues 2: Lack of Parallelizability



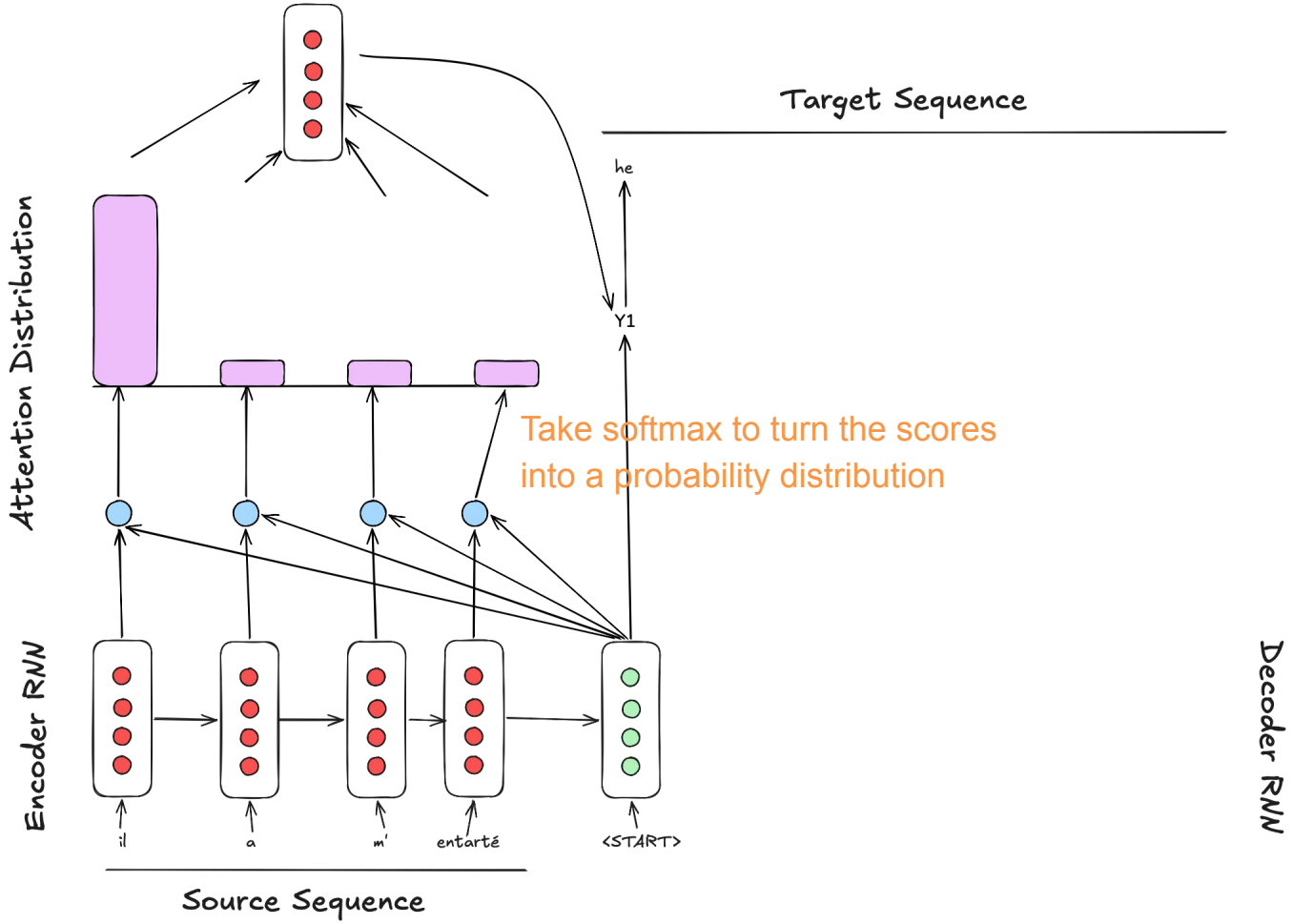
The "Hardware Bottleneck" Problem

- **Sequential Dependency:** Step t cannot begin until step t−1 is complete.
- **GPU Underutilization:** Modern hardware is built for parallel "grid" work, but RNNs force "single-lane" processing.
- **The Scaling Wall:** Training on massive datasets (Web-scale) becomes computationally impossible due to time constraints.

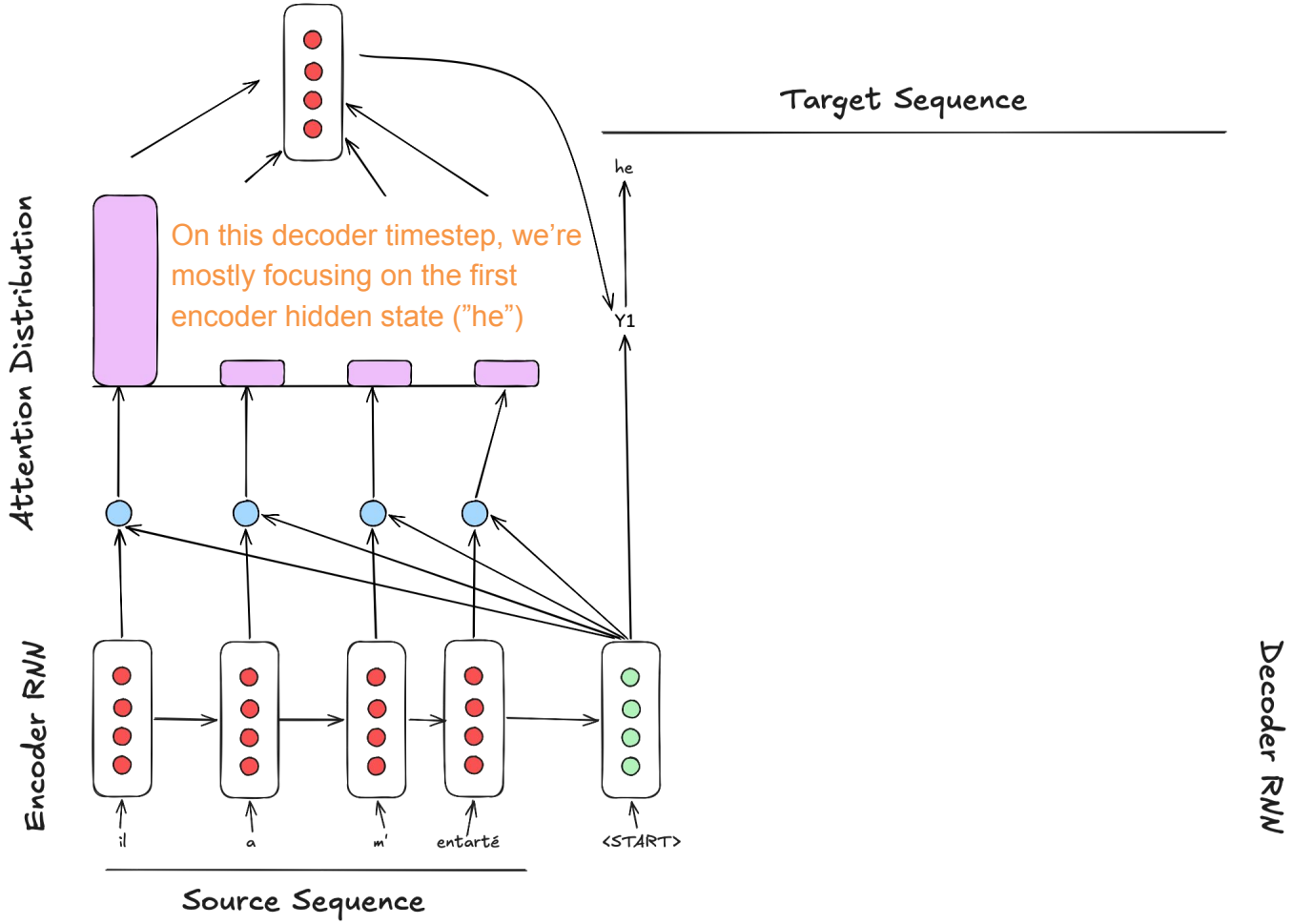
Solution?


Target Sequence
he
Y1
Attention Distribution
Dot Product
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
Source Sequence

Solution?

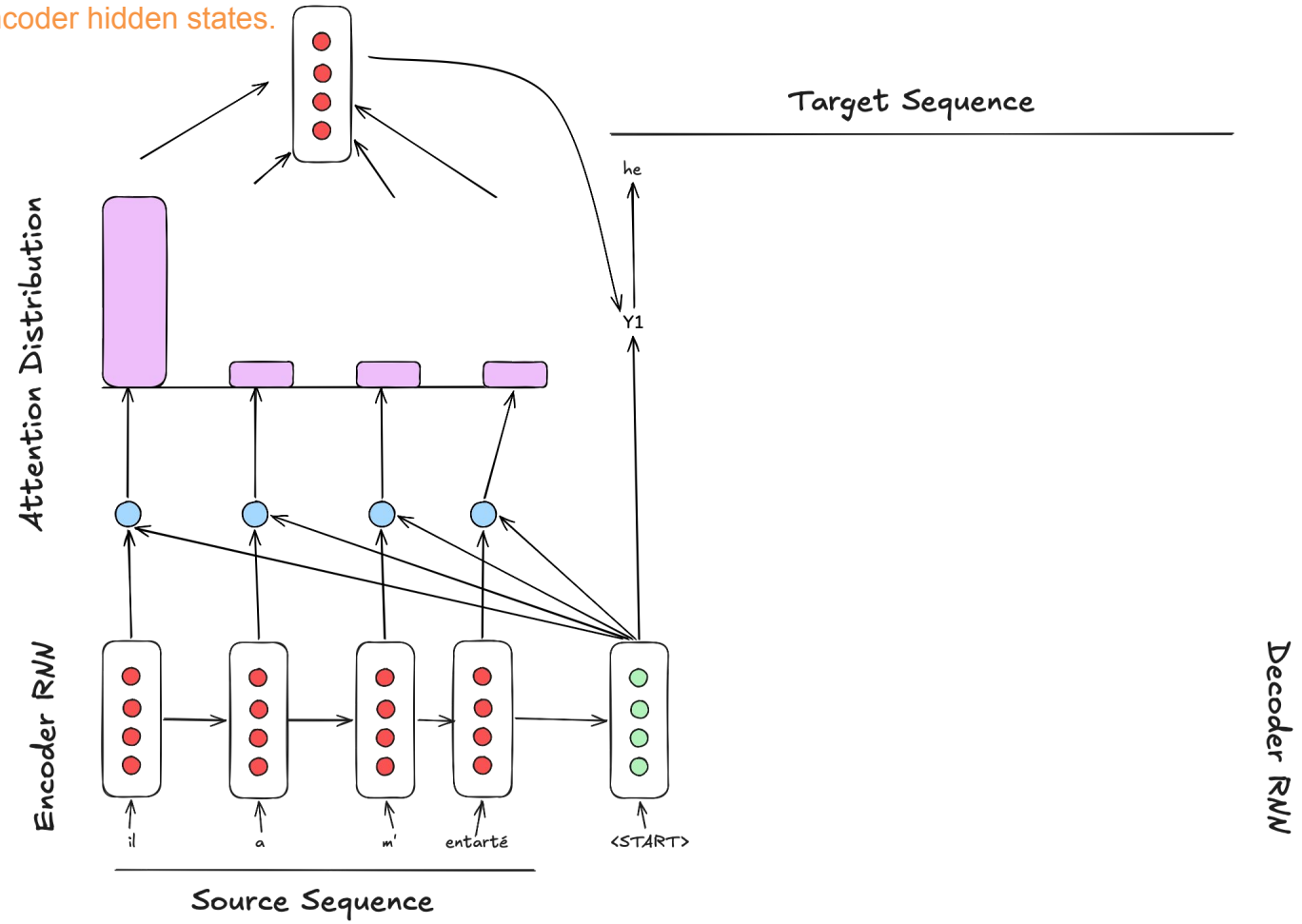

Target Sequence
he
Y1
Attention Distribution
Take softmax to turn the scores into a probability distribution
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
Source Sequence

Solution?

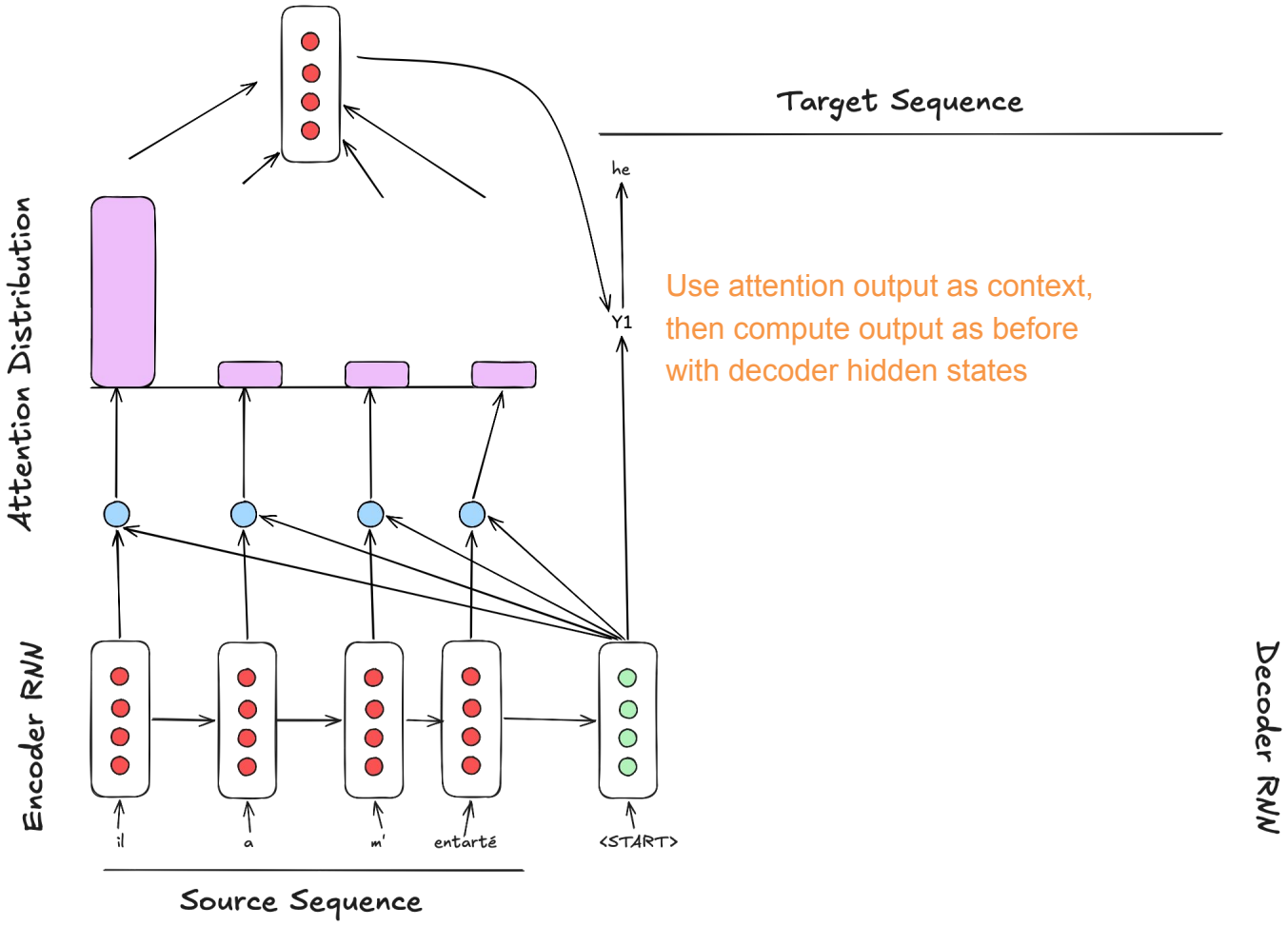

Target Sequence
he
Y1
On this decoder timestep, we're mostly focusing on the first encoder hidden state ("he")
Attention Distribution
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
Source Sequence

Solution?

Use the attention distribution to take a weighted sum of the encoder hidden states.

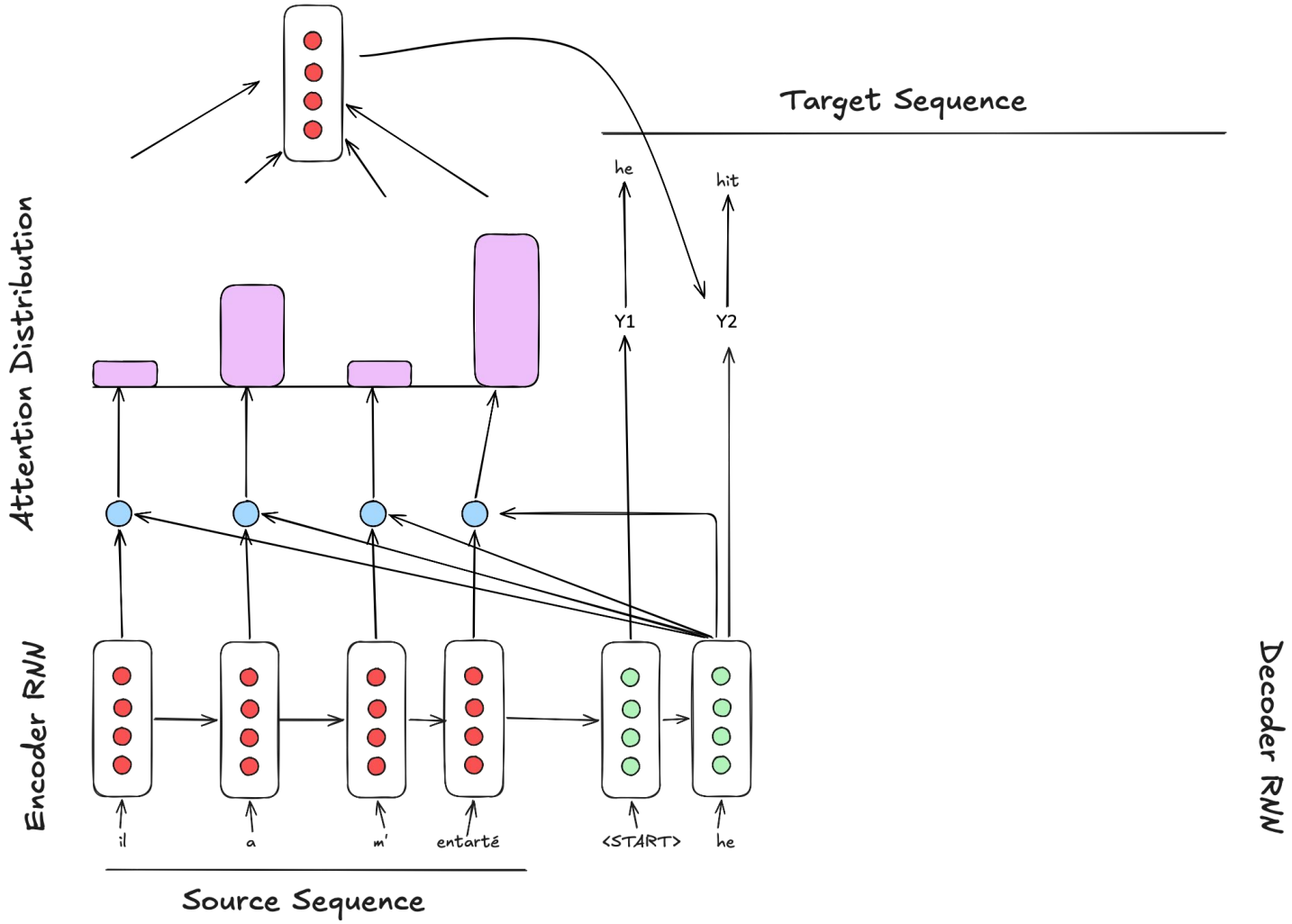


Solution?

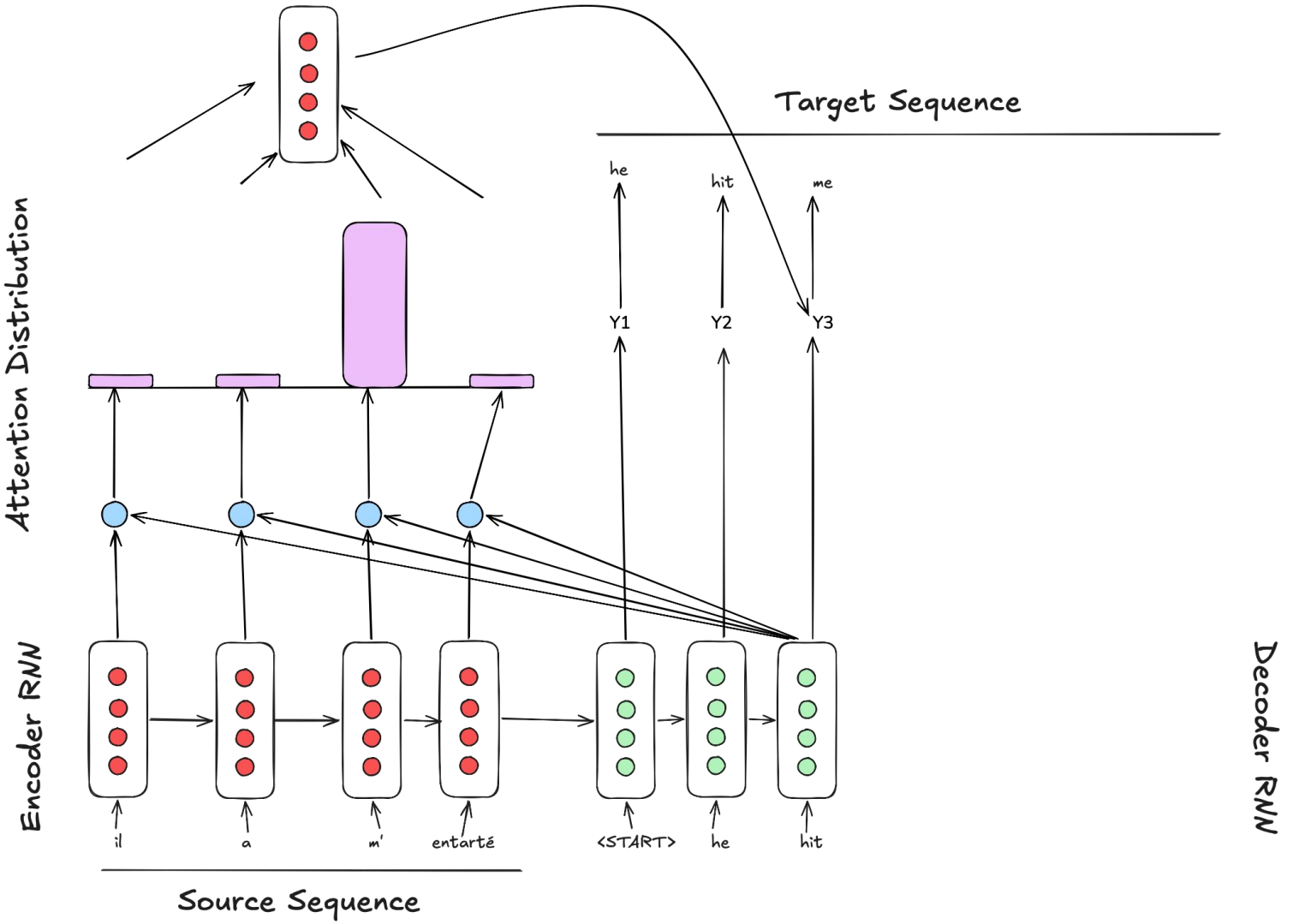


Use attention output as context, then compute output as before with decoder hidden states

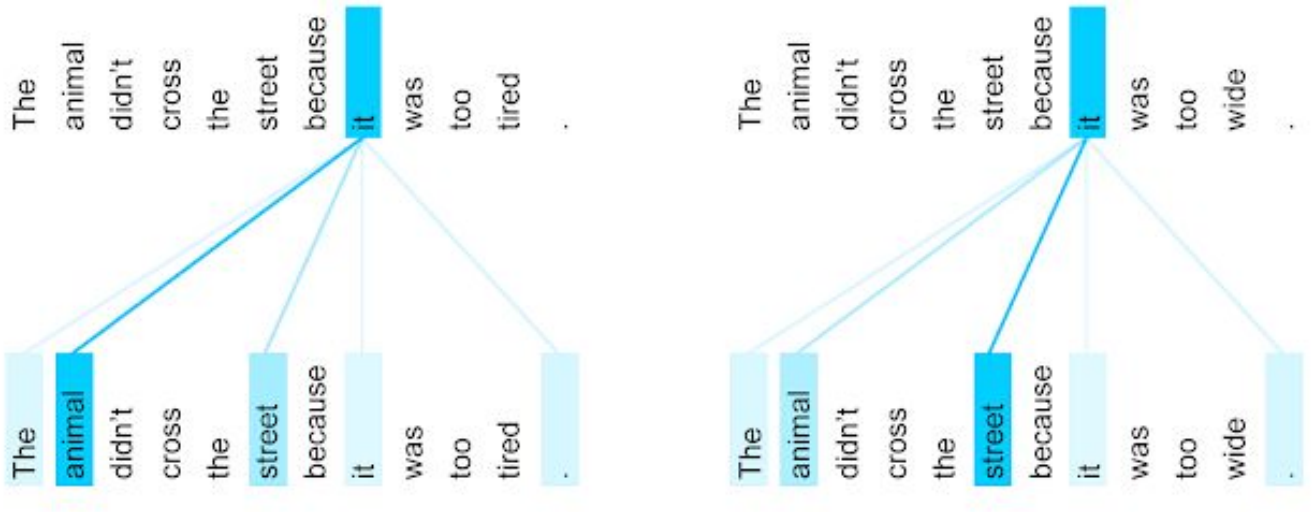
Solution?


Target Sequence
he
hit
Y1
Y2
Attention Distribution
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
Source Sequence

Solution?


Target Sequence
he
hit
me
Y1
Y2
Y3
Attention Distribution
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
hit
Source Sequence

From Recurrence to Attention

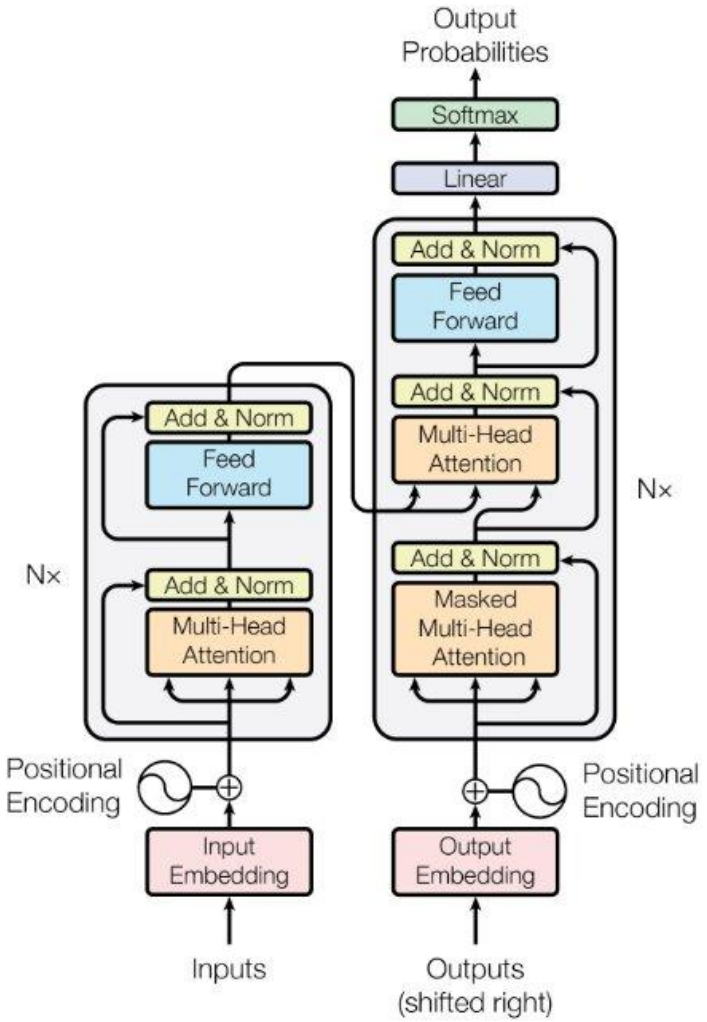


- **O(1) Path Length:** Every token "talks" directly to every other token simultaneously, eliminating the information decay found in RNNs.
- **Sequential Bottleneck Removed:** Computation is no longer restricted by sequence length N, allowing for massive parallelization on modern GPUs.
- **Dynamic Attention Routing:** Replaces rigid "fixed-size" memory vectors with a flexible system that queries the entire sequence for relevant context.

Do we need recurrence at all?

- Both Attention and Recurrent Neural Networks (RNNs) serve the exact same abstract purpose: passing sequential information into a neural network.
- Attention provides a highly flexible, selective summary of information by computing a weighted sum of values based on a specific query.
- Since attention handles memory manipulation and information routing so effectively, we can potentially discard RNNs entirely in favor of pure attention mechanisms.

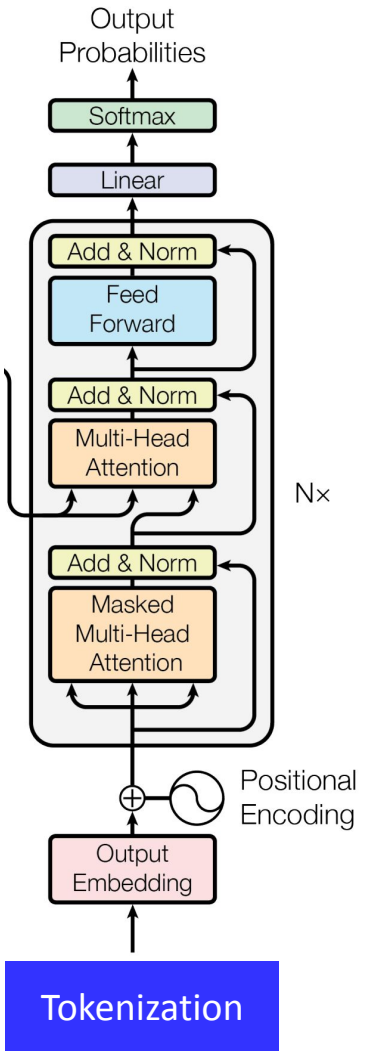
Transformer Architecture



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

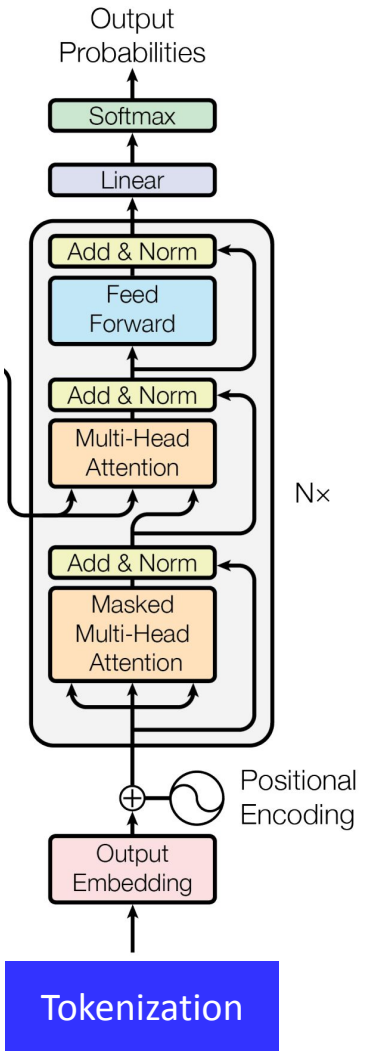
Transformer Architecture

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

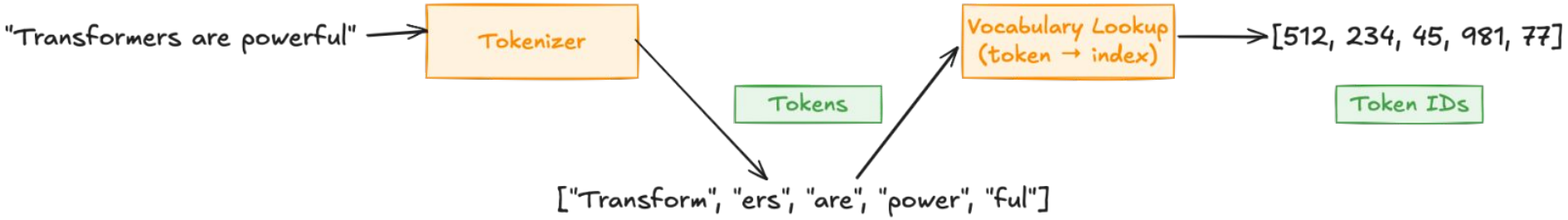


Transformer Architecture

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

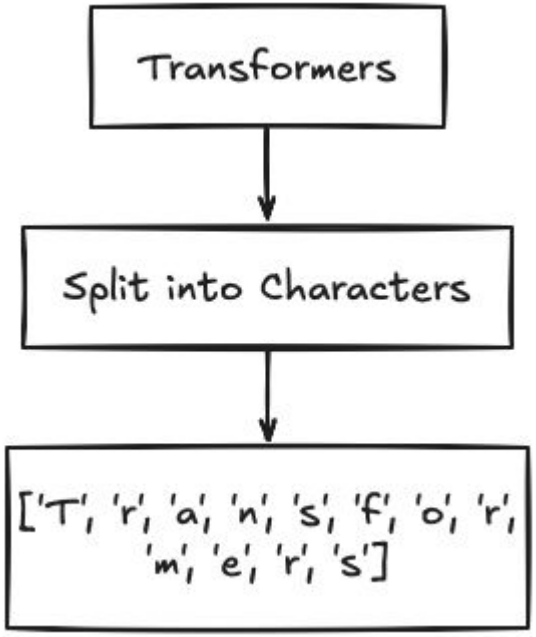


Tokenization



- Splits text into manageable units (words, subwords, or characters) for AI models to process.
- Each token is mapped to a **unique integer ID**
- Output: sequence of integers representing the sentence
- Trade Offs:
  - Vocabulary Size vs. Sequence Length
  - Meaning vs. Generalization

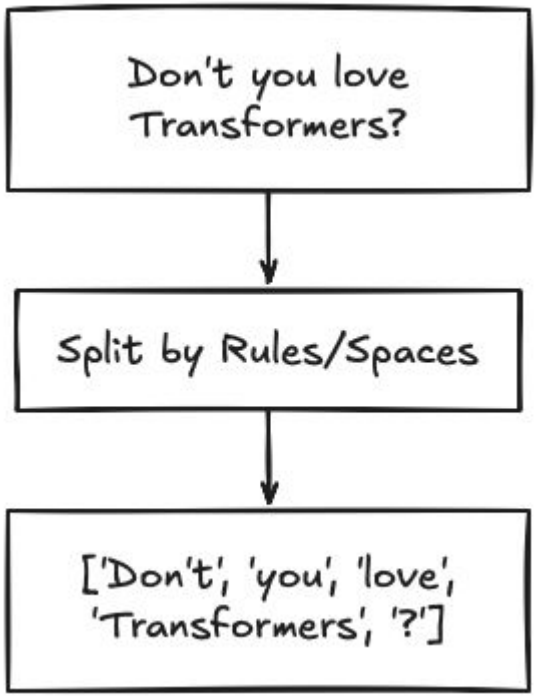
Character-Level Tokenization



- Breaks down all text into individual, single characters.
- Extremely small and compact, guaranteeing every word can be represented without <unk> tokens.
- Creates massive sequence lengths that are computationally expensive to process.
- Individual characters (like "l" or "o") carry very little inherent meaning compared to full words or subwords, hurting performance.

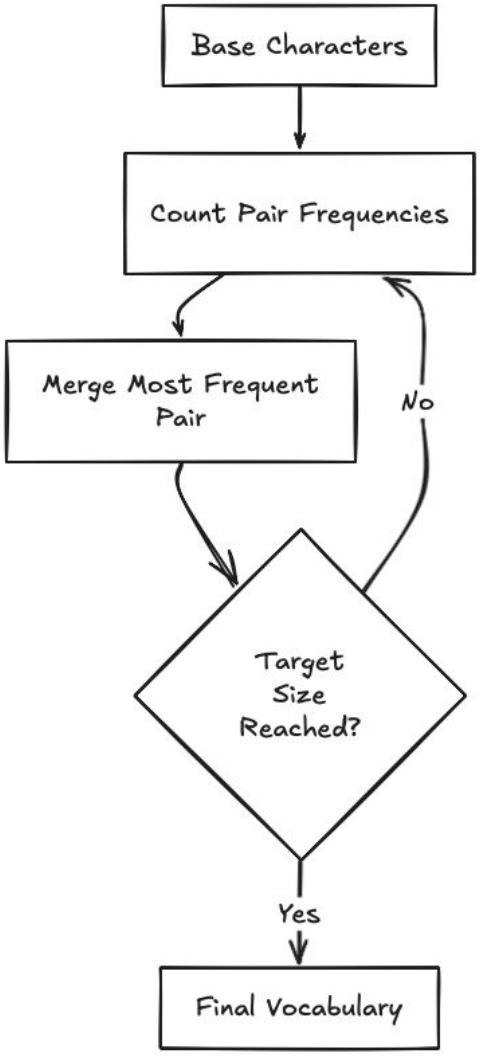
Word-Level Tokenization

- Splits text strictly by spaces, punctuation, or language rules.
- Creates enormous embedding matrices because every variation (e.g., "love", "loving", "loved") requires a unique token.
- Fails gracefully; any unseen word defaults to an uninformative <unk> token.
- The massive vocabulary increases both memory usage and compute time.



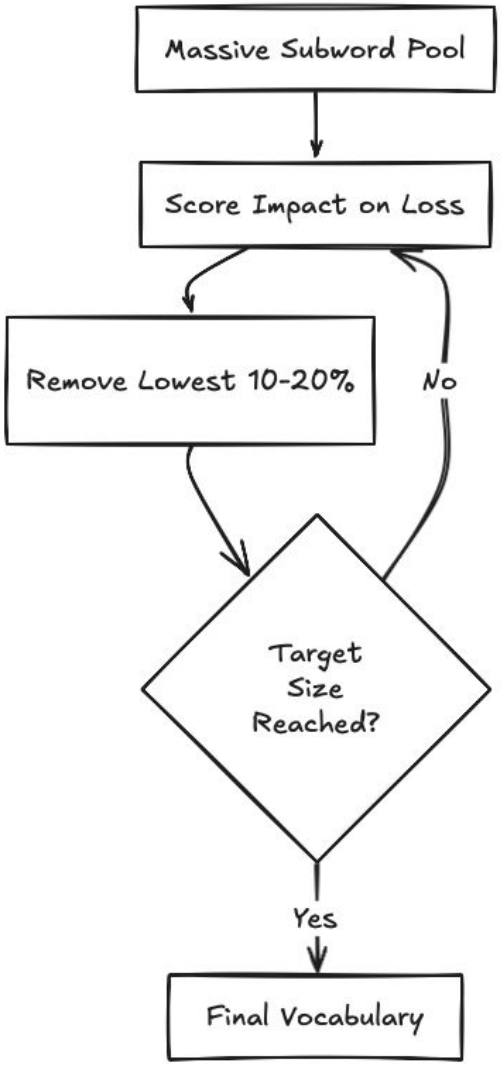
Byte Pair Encoding (BPE)

- The most popular algorithm, used by major models like Llama, Gemma, and Qwen2.
- Builds a vocabulary from the bottom up by aggressively merging the most frequent adjacent character pairs.
- Uses 256 byte values as a base (instead of all Unicode characters) to prevent base vocabulary bloat while guaranteeing no <unk> tokens.



"Implementing A Byte Pair Encoding (BPE) Tokenizer From Scratch"

Unigram Tokenization

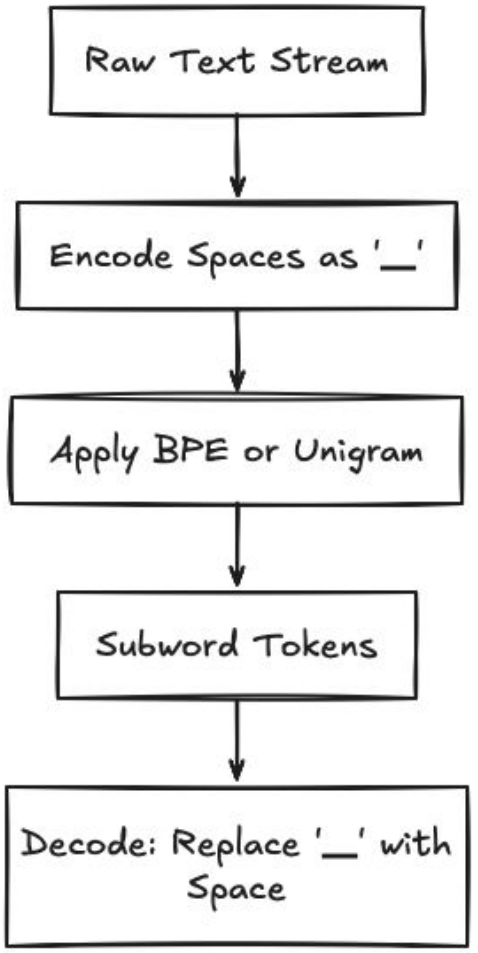


- Starts with a massive pool of subwords and ruthlessly prunes the bottom 10-20% least useful tokens.
- Keeps tokens that minimize overall model loss, rather than just merging what appears most often.
- Can sample different tokenizations for the exact same word during training, adding a regularization effect.
- Favored by T5 and Pegasus for its nuanced, context-aware subword selection.

"Unigram Tokenization"

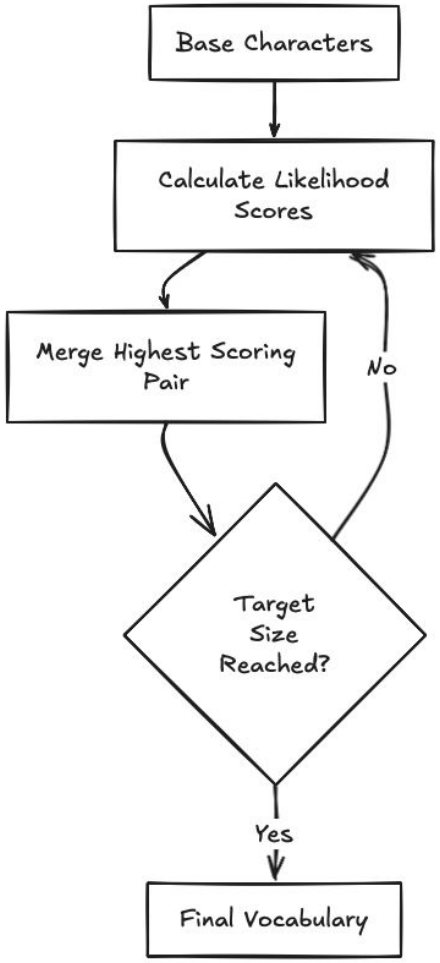
SentencePiece

- Standard tokenizers assume spaces separate words, which fails for languages like Chinese or Japanese.
- Treats the entire input text as a raw stream of bytes or characters.
- Injects spaces directly into the vocabulary, representing them with a special character ("_").
- Once the raw stream is prepped, it applies either BPE or Unigram under the hood.



SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing

WordPiece



- The standard tokenization algorithm for BERT-family models (DistilBERT, Electra).
- Similar bottom-up merging style to BPE, but uses a different selection criteria.
- Merges pairs that maximize the likelihood of the training data, rather than raw frequency.
- Prioritizes merging tokens that appear together much more often than chance would predict.

JAPANESE AND KOREAN VOICE SEARCH - Google Research

Poll @ TBD

Which of the following is a potential downside of character-level tokenization compared to subword tokenization?

- It increases the computational cost of self-attention.
- It makes the model unable to learn spelling..
- It cannot handle out-of-vocabulary words.
- It leads to a very large vocabulary size.

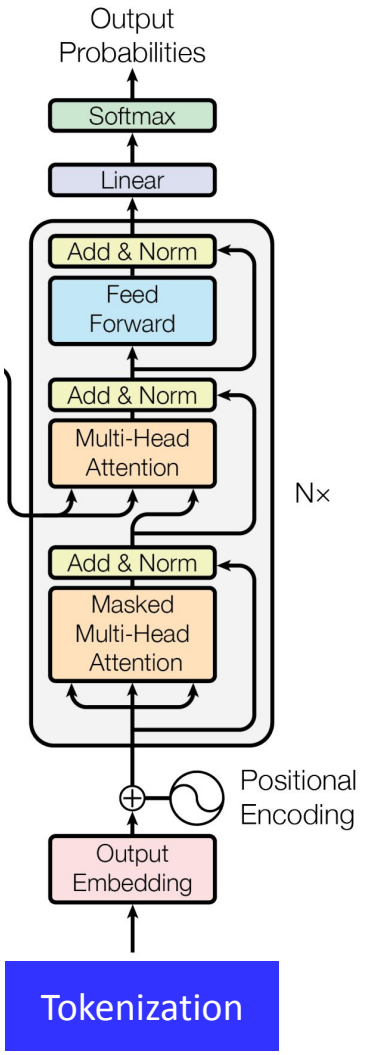
Poll @ TBD

Which of the following is a potential downside of character-level tokenization compared to subword tokenization?

- It increases the computational cost of self-attention.
- It makes the model unable to learn spelling..
- It cannot handle out-of-vocabulary words.
- It leads to a very large vocabulary size.

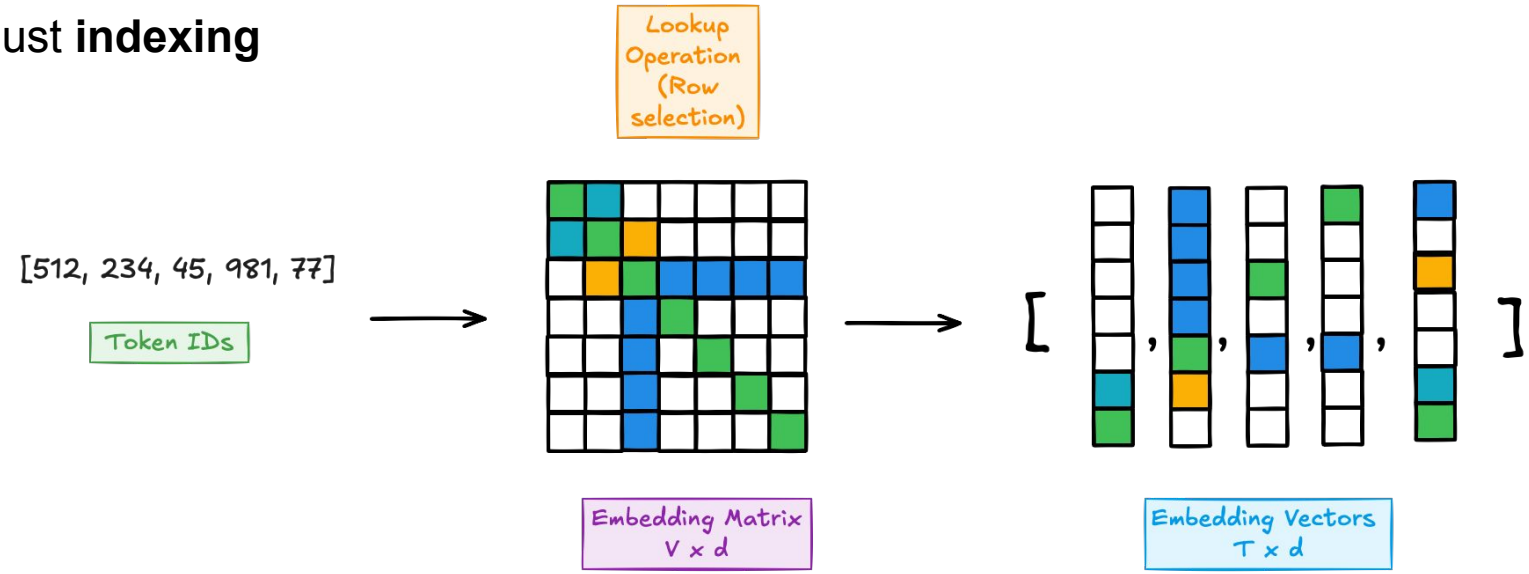
Transformer Architecture

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



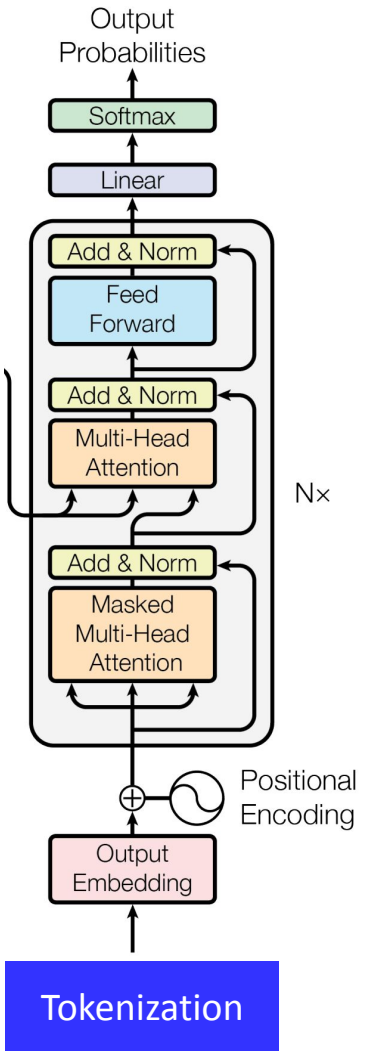
Embedding

- `nn.Embedding` is a **learned lookup table**
- Matrix size: **(vocab_size × embedding_dim)**
- Each token ID selects **one row (vector)**
- Just **indexing**



Transformer Architecture

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



Position Encoding

- Unlike RNNs, Transformers process all words in a sequence simultaneously.
  - "The dog bit the man" and "The man bit the dog" look identical.
- Without order, the model loses the structural meaning of syntax and grammar, treating a sentence as a "bag of words"
- Need to "inject" a sense of time or place into the word embeddings before they hit the first Attention layer.

The Core Requirements

- **Unambiguous:** Every position (1, 2, 3...) must have a unique signature. If two positions look the same, the model gets confused.
- **Deterministic:** The encoding for "position 3" should be the same every single time, regardless of the sentence content.
- **Distance Estimation:** The distance between any two tokens should be consistent. The model needs to easily learn that "word 2" is closer to "word 3" than it is to "word 10."
- **Scalability:** The method should ideally generalize to sequences longer than those seen during training (extrapolation).

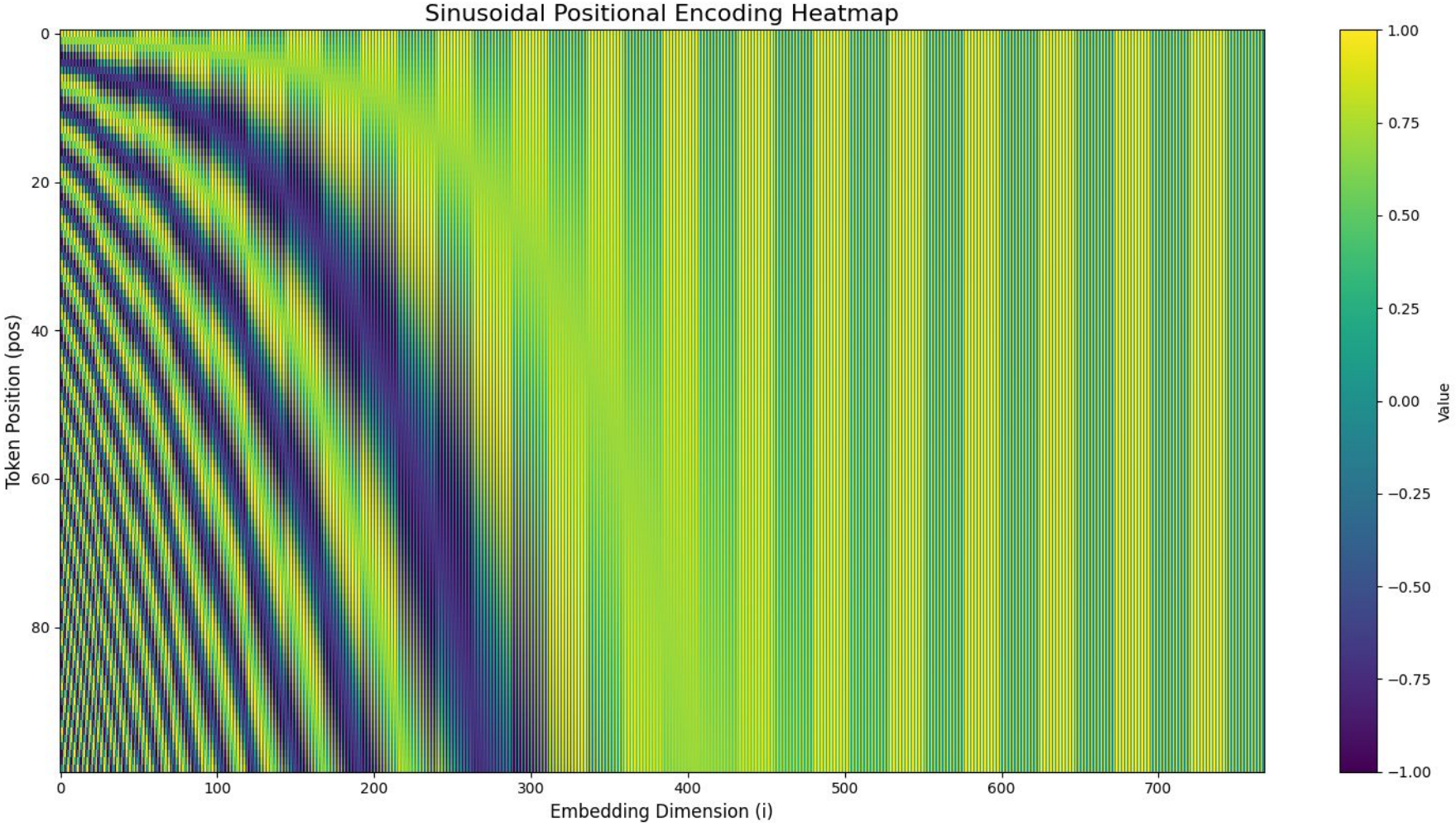
Sinusoidal Encoding

The original Transformer paper used sine and cosine functions of different frequencies to create a unique "fingerprint" for every position.

- **The Math:** It uses a geometric progression of frequencies:

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

- **Why Sinusoids?** This allows the model to learn **relative** positions easily, as for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.
- These vectors are simply **added** to the word embeddings. Since they have the same dimension (dmodel), they don't increase the data size.
- **The Flaws:** While theoretically scalable, it can struggle with very long sequences it wasn't tuned for.


Sinusoidal Positional Encoding Heatmap
Token Position (pos)
0
20
40
60
80
0
100
200
300
400
500
600
700
Embedding Dimension (i)
Value
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


sin (even dims)
cos (odd dims)
dim 0
(high freq)
dim 8
(med-high)
dim 32
(med-low)
dim 48
(low freq)
position
position
pos 5 (reference)
pos 6 (close)
pos 80 (far)

Poll @ TBD

In a positional encoding vector, why are different dimensions assigned different wavelengths (frequencies)?

- To reduce the computational complexity of the dot-product attention mechanism.
- To allow the model to capture patterns at different scales.
- To ensure the positional encoding is orthogonal to the word embedding space
- To prevent the self-attention weights from concentrating only on the current token.

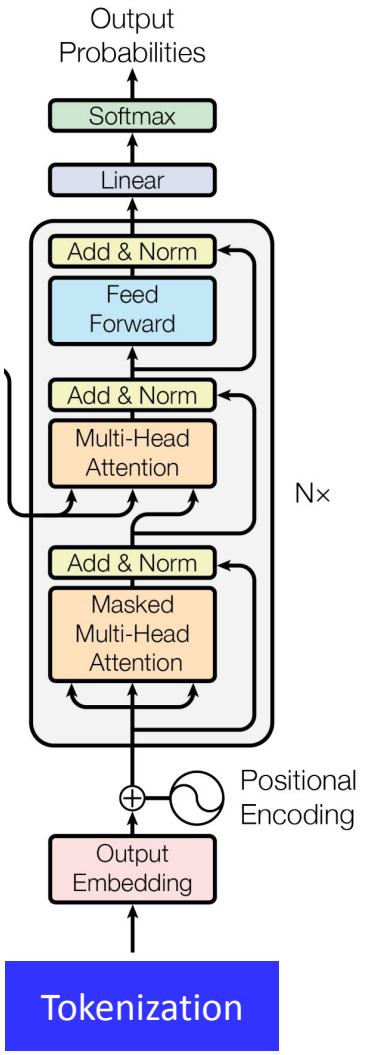
Poll @ TBD

In a positional encoding vector, why are different dimensions assigned different wavelengths (frequencies)?

- To reduce the computational complexity of the dot-product attention mechanism.
- To allow the model to capture patterns at different scales.
- To ensure the positional encoding is orthogonal to the word embedding space
- To prevent the self-attention weights from concentrating only on the current token.

Transformer Architecture

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



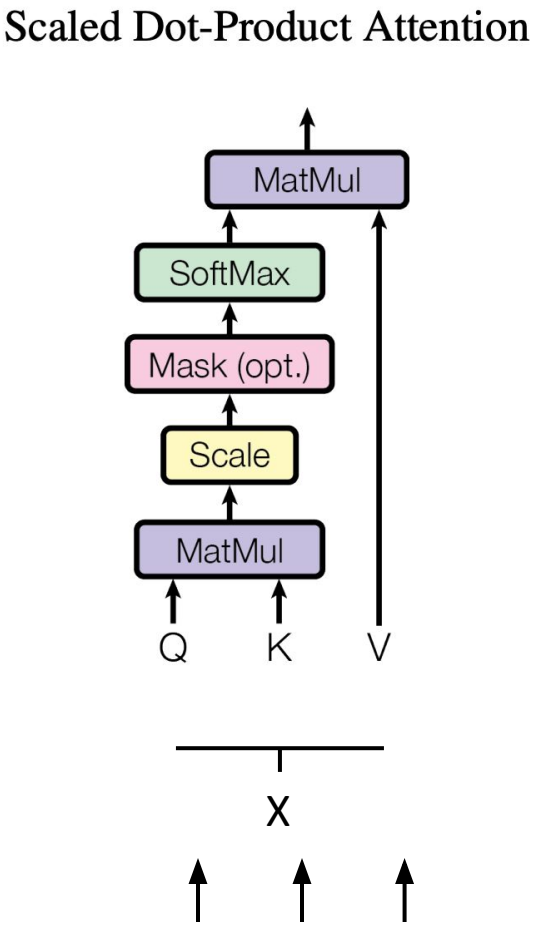
Scaled Dot-Product Attention

- Attention Operation

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

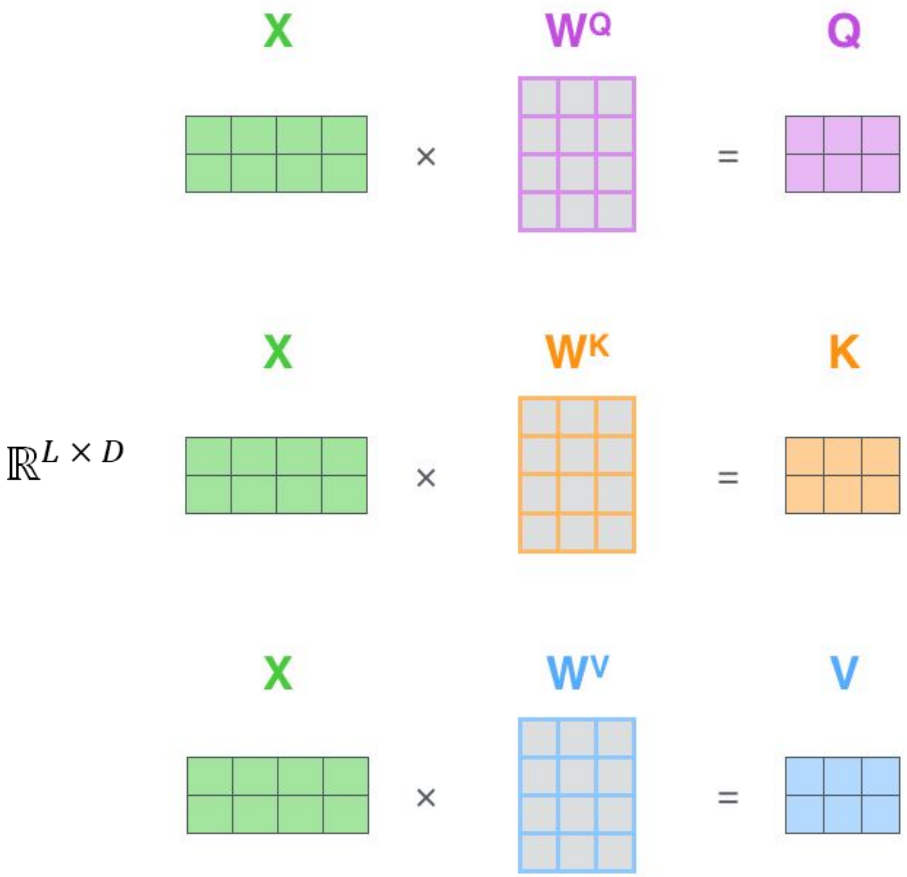
- Query-Key-Value
  - Linear affine from input X itself

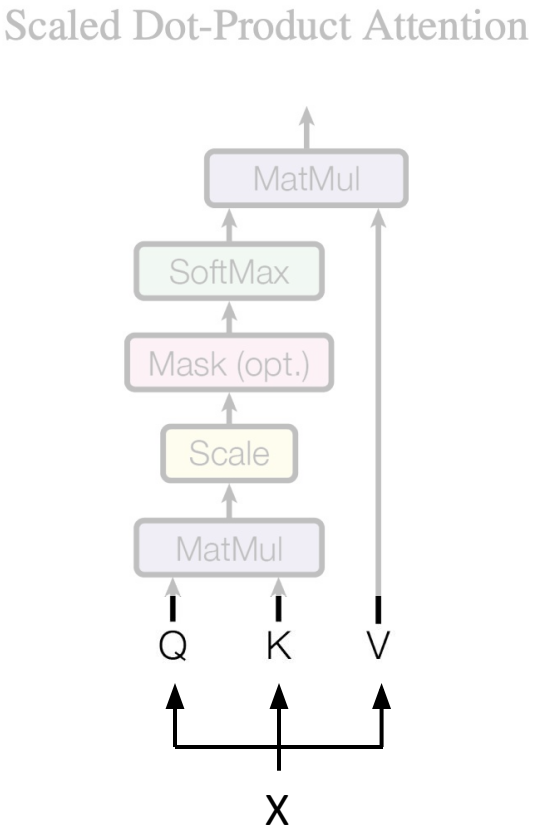
- Weighted-sum of V based on similarity/correlation between Q and K
  - Each token's weights sum to one



Scaled Dot-Product Attention

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





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

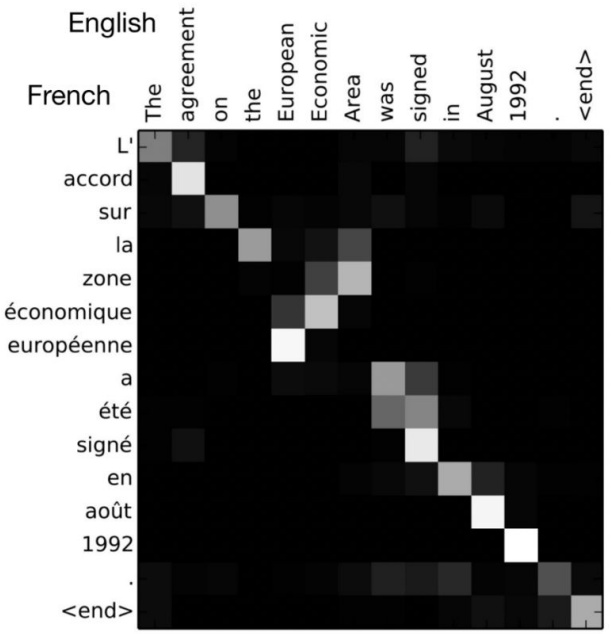
Scaled Dot-Product Attention

- Attention weights

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

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

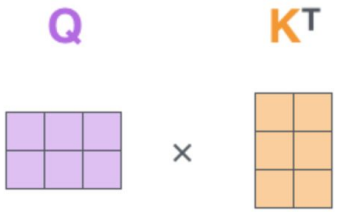
$$\mathbb{R}^{L \times L}$$



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

Scaled Dot-Product Attention

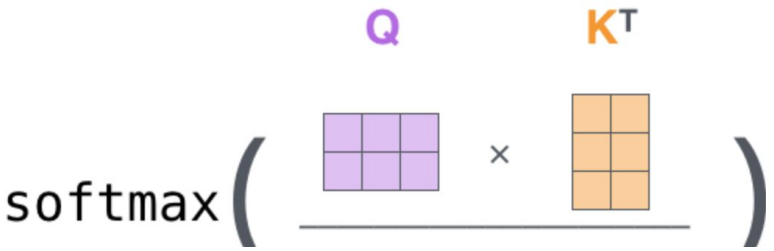
- Why is the dot product not enough?



- Dot product could be arbitrary numbers

Scaled Dot-Product Attention

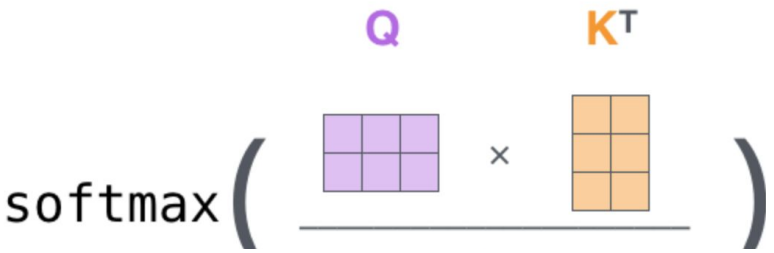
- Why is the dot product not enough?



- Dot product could be arbitrary numbers
- We want a weighted average of all attention scores to get the appropriate V => Softmax

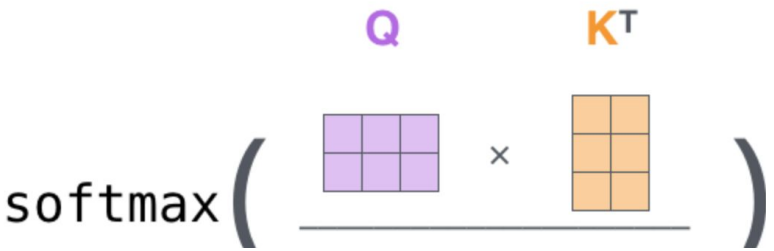
Scaled Dot-Product Attention

- That seems good enough, why do we need the sqrt(d_k)



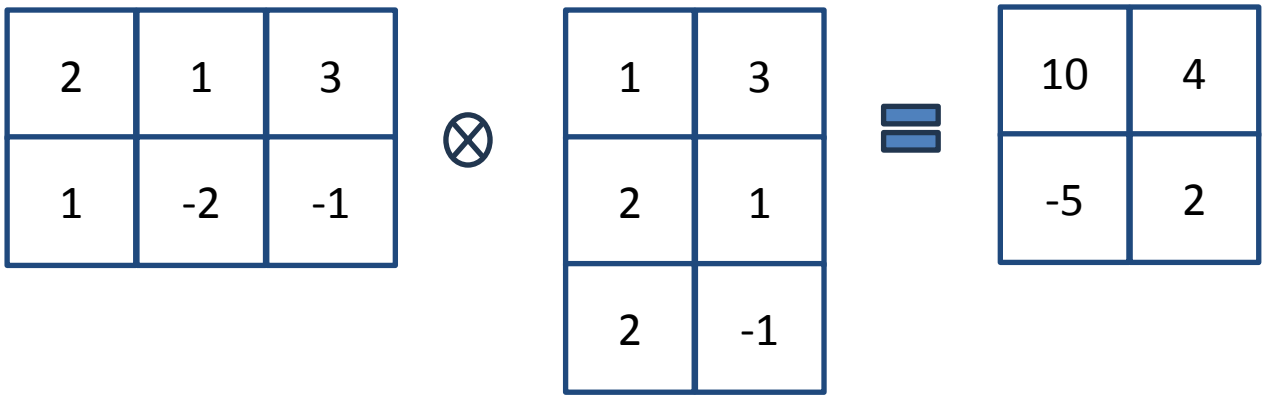
Scaled Dot-Product Attention

- That seems good enough, why do we need the sqrt(d_k)



- Softmax has a vanishing/exploding problem!

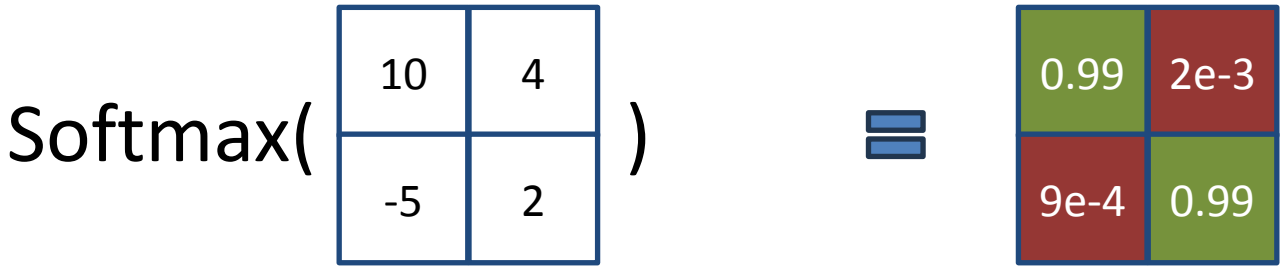
why d_k



why d_k

Softmax(| 10 | 4 | | -5 | 2 |) = | 0.99 | 2e-3 | | 9e-4 | 0.99 |

why d_k

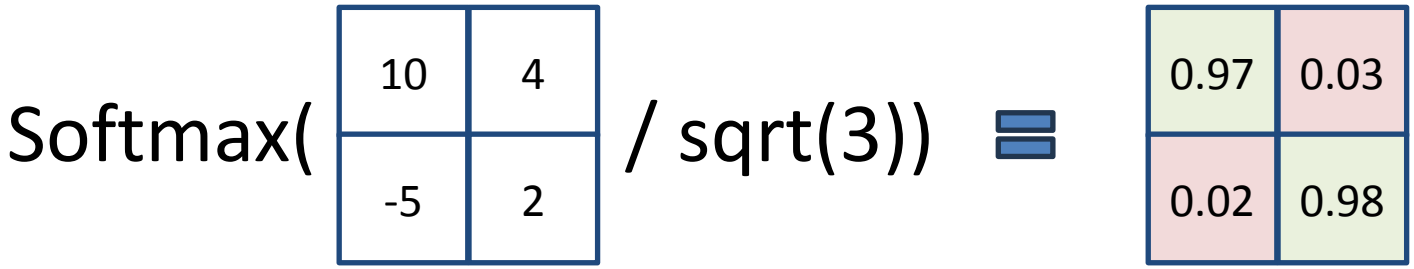


Softmax pushes disparity to extremes
=> vanishing gradients

why d_k

Softmax(| 10 | 4 | / sqrt(3)) = | 0.97 | 0.03 |
| --- | --- | --- | --- | --- |
| -5 | 2 | | 0.02 | 0.98 |

why d_k

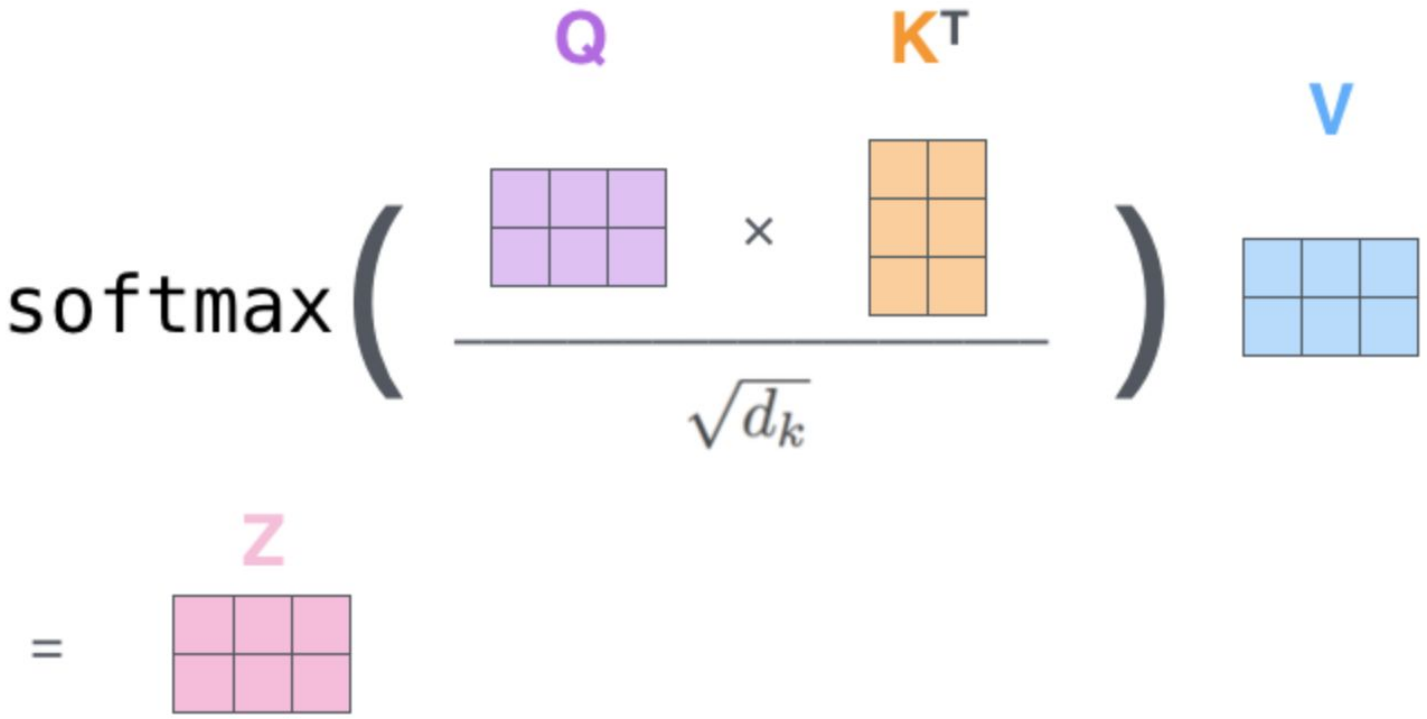


Sqrt(d_k) is the std. dev. of the dot product

Scaling by that reduces the vanishing gradient!

Scaled Dot-Product Attention

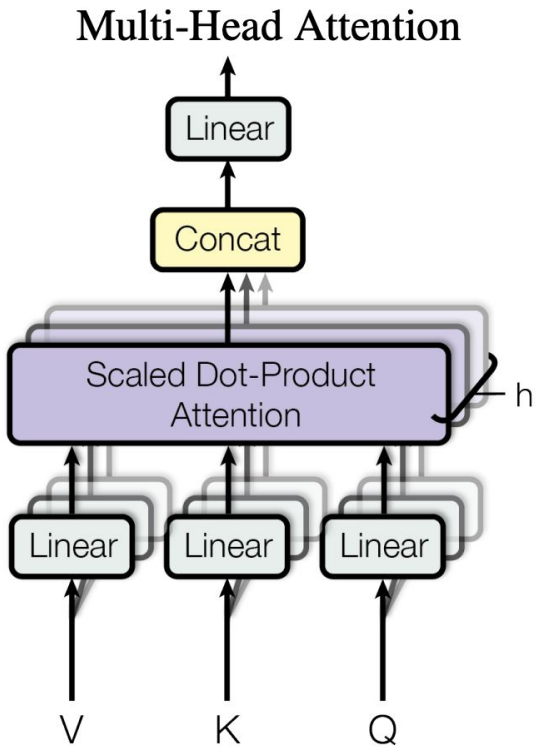
- Output



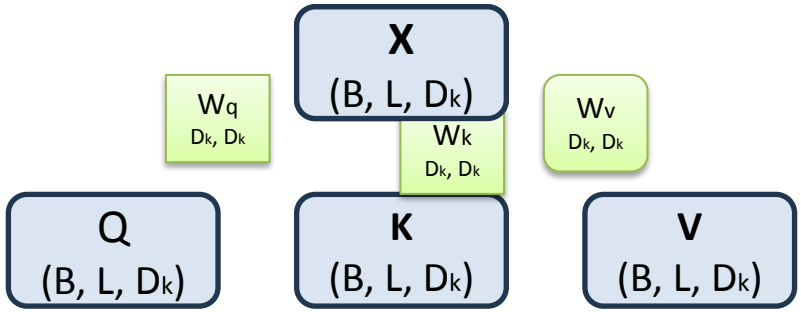
Multi-Head Attention

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

- Instead of one large attention pass, we split the focus into h independent "heads" to capture different types of relationships (e.g., grammar vs. context).
- Each head uses unique projections
- Each head computes its own output independently.
- All head outputs are concatenated and passed through a final weight matrix Y∈R^(d×d) to produce the final result.



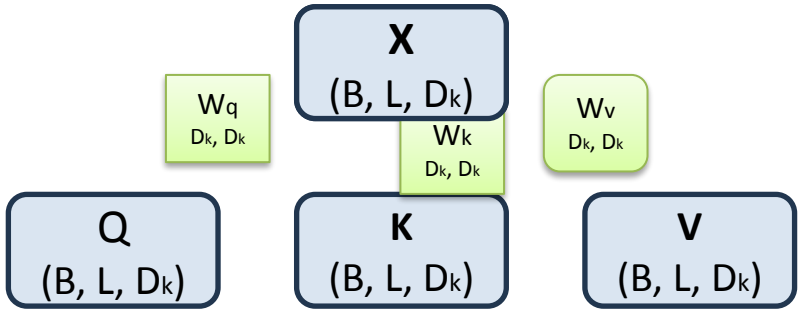
(F.scaled_dot_product_attention) # SHA vs MHA (torch.nn.MultiheadAttention)



S(Q@K.T)
(B, L, L)

A@V = O
(B, L, Dk)

(F.scaled_dot_product_attention) # SHA vs MHA (torch.nn.MultiheadAttention)



$D_k = h * D_h$

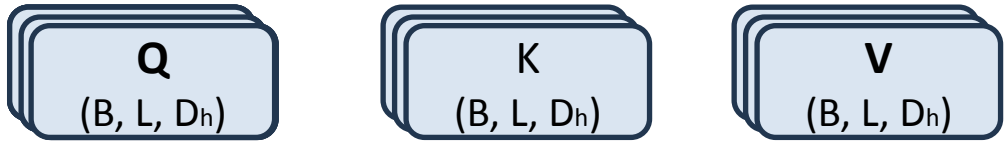
Reshape to (B, L, h, D_h)

Transpose (-3, -2) -> (B, h, L, D_h)

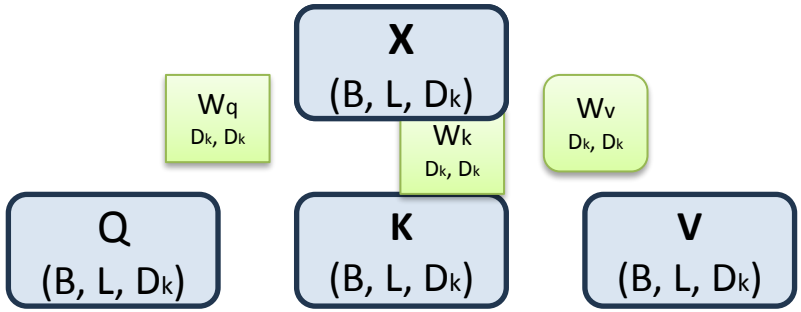
S(Q@K.T)
(B, L, L)

A@V = O
(B, L, D_k)

h



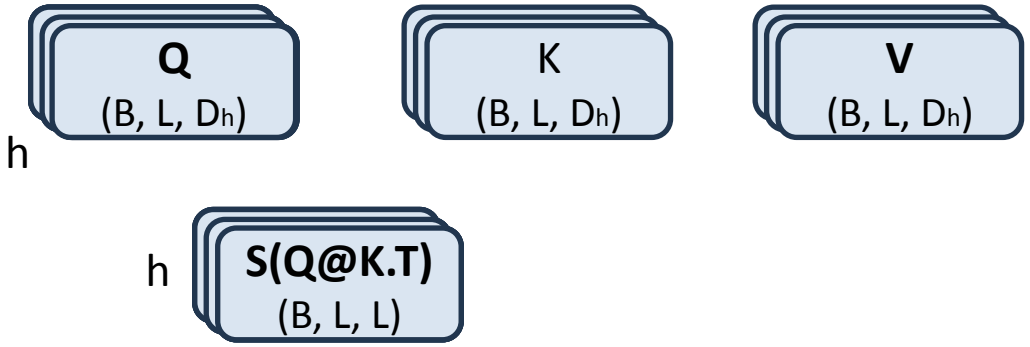
(F.scaled_dot_product_attention) # SHA vs MHA (torch.nn.MultiheadAttention)



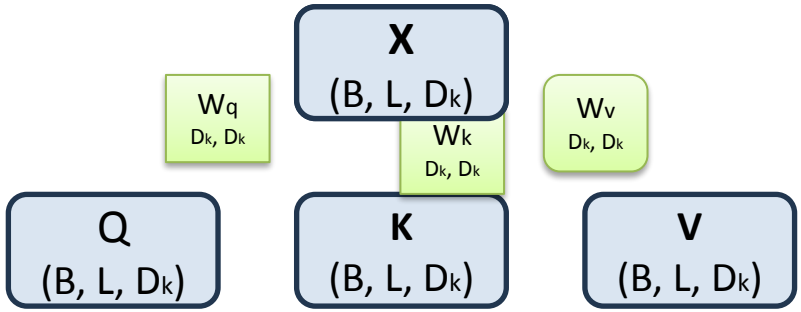
$D_k = h * D_h$
Reshape to (B, L, h, D_h)
Transpose (-3, -2) -> (B, h, L, D_h)

S(Q@K.T)
(B, L, L)

A@V = O
(B, L, D_k)



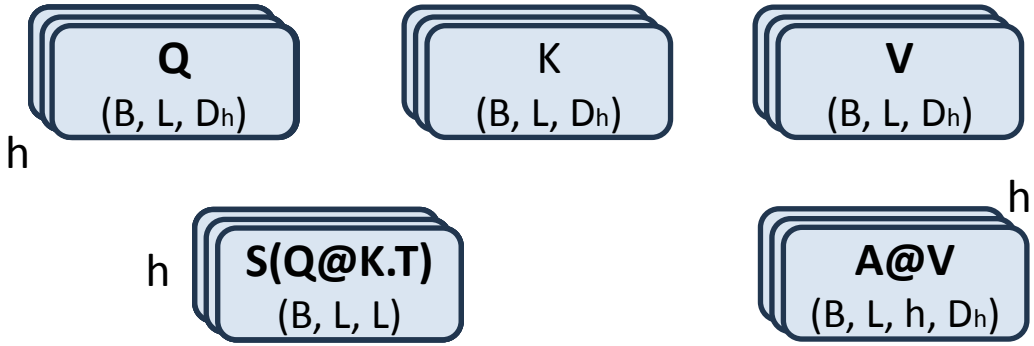
(F.scaled_dot_product_attention) # SHA vs MHA (torch.nn.MultiheadAttention)



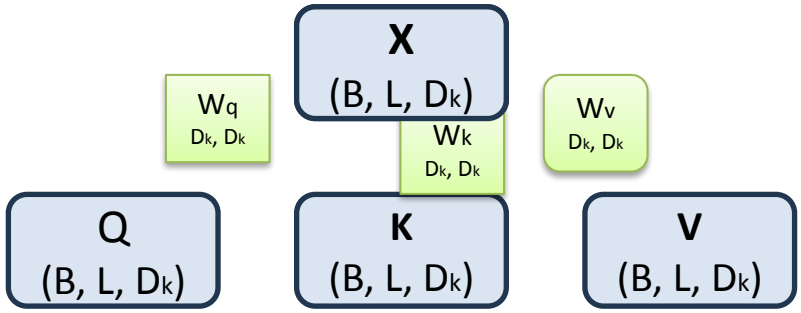
$D_k = h * D_h$
Reshape to (B, L, h, D_h)
Transpose (-3, -2) -> (B, h, L, D_h)

S(Q@K.T)
(B, L, L)

A@V = O
(B, L, D_k)



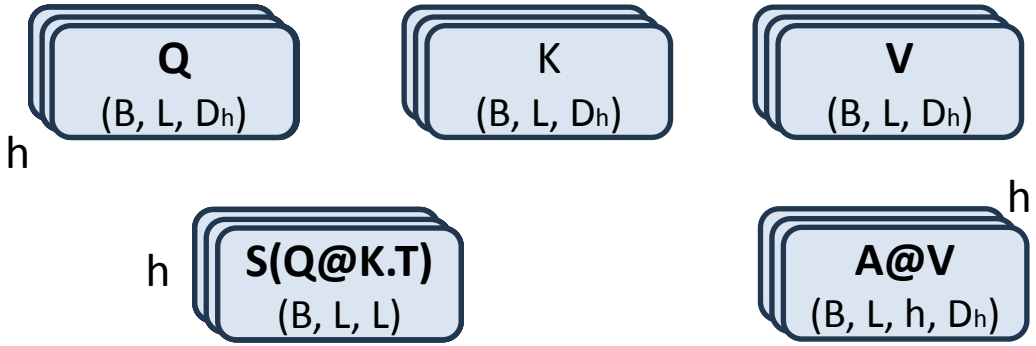
(F.scaled_dot_product_attention) SHA vs MHA (torch.nn.MultiheadAttention)



$D_k = h * D_h$
Reshape to (B, L, h, D_h)
Transpose (-3, -2) -> (B, h, L, D_h)

S(Q@K.T)
(B, L, L)

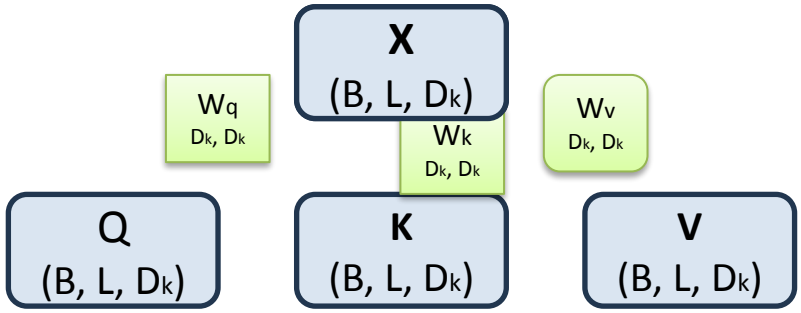
A@V = O
(B, L, D_k)



Transpose (B, h, L, D_h) -> (B, L, h, D_h)
Reshape to (B, L, h*D_h)

Z
(B, L, D_k)

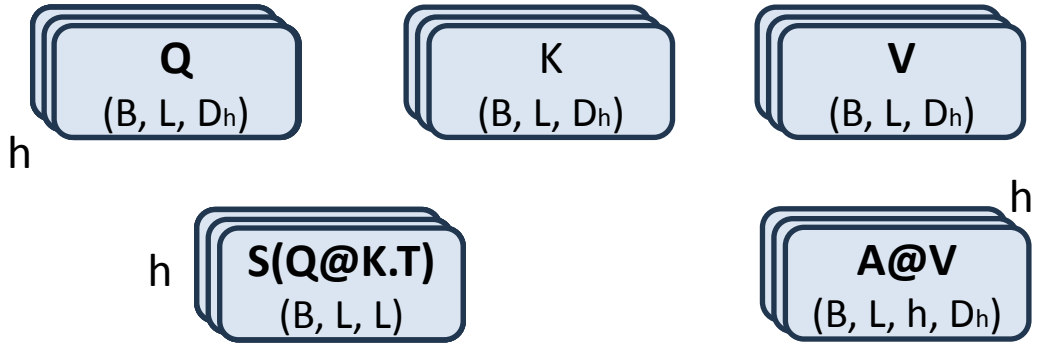
(F.scaled_dot_product_attention) SHA vs MHA (torch.nn.MultiheadAttention)



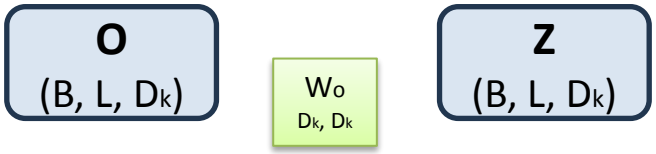
$D_k = h * D_h$
Reshape to (B, L, h, D_h)
Transpose (-3, -2) -> (B, h, L, D_h)

S(Q@K.T)
(B, L, L)

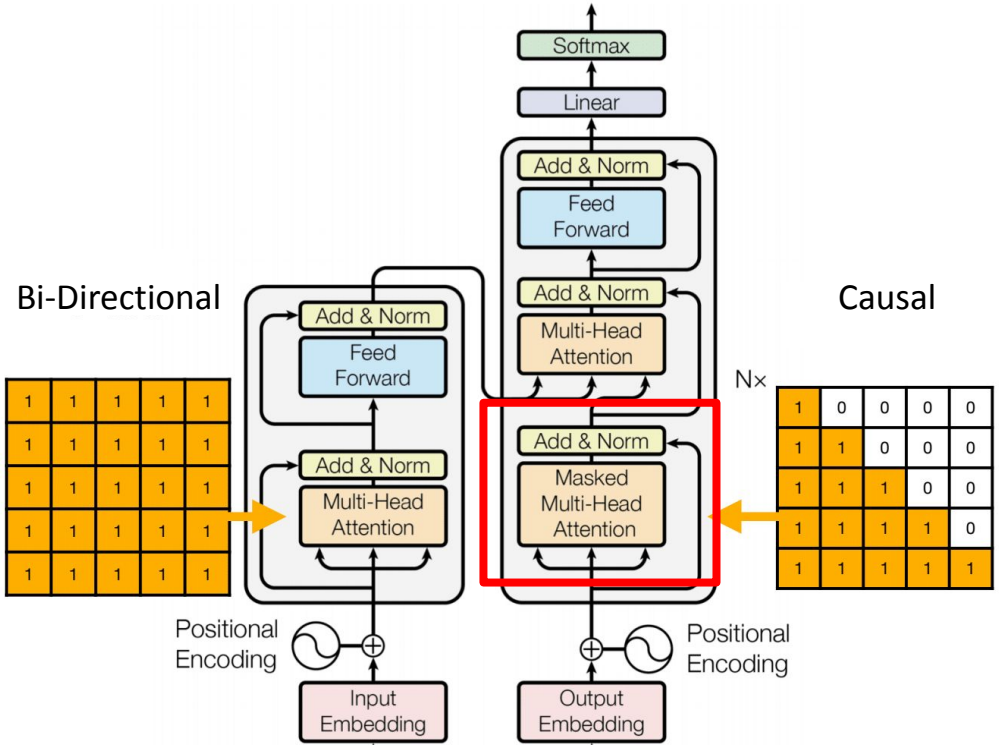
A@V = O
(B, L, D_k)

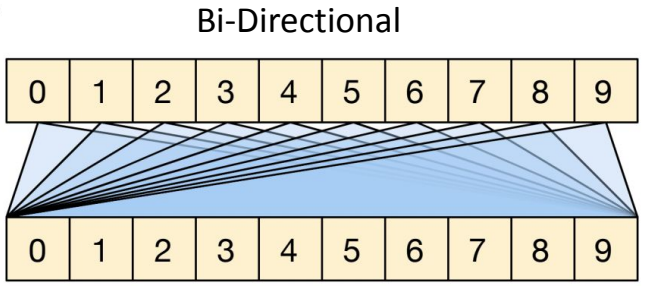


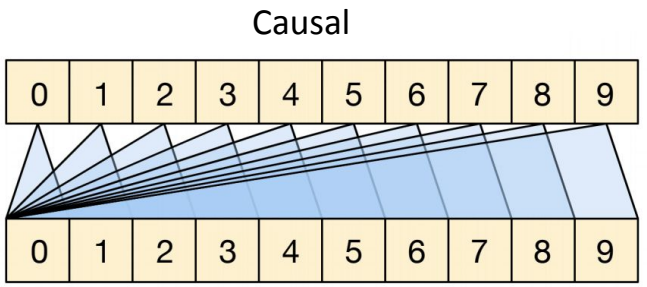
Transpose (B, h, L, D_h) -> (B, L, h, D_h)
Reshape to (B, L, h*D_h)



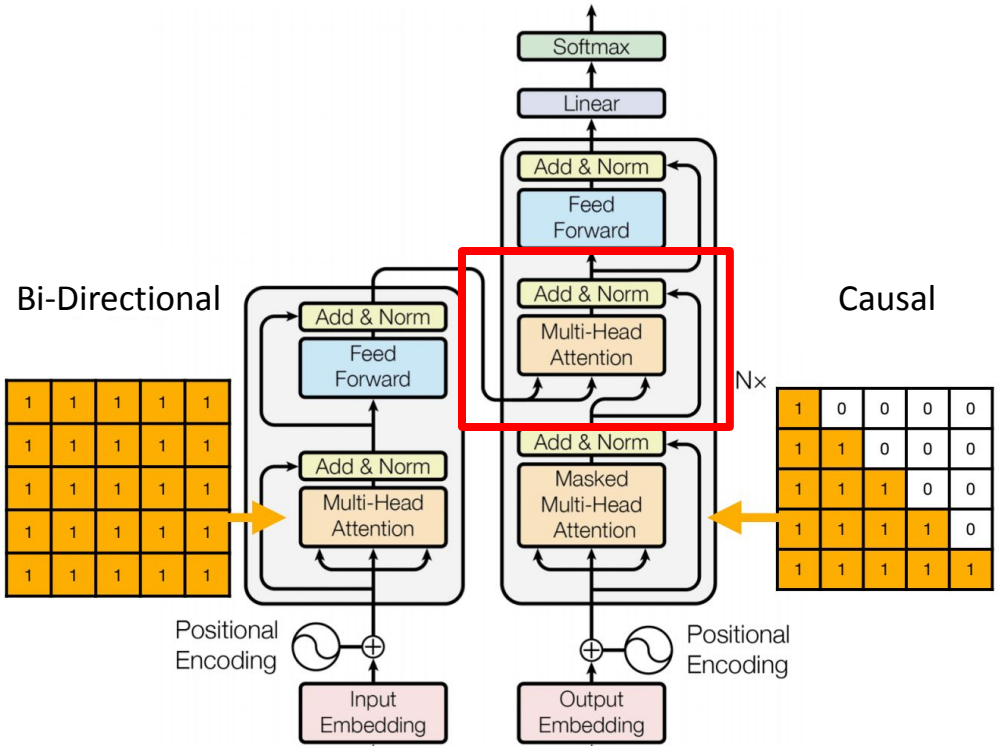
Attention Masking

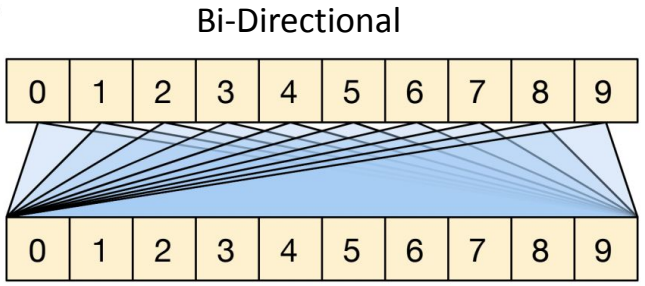


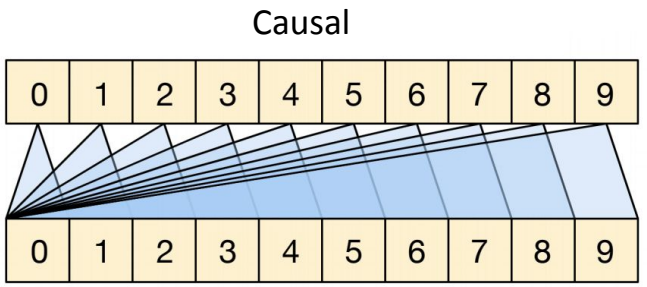




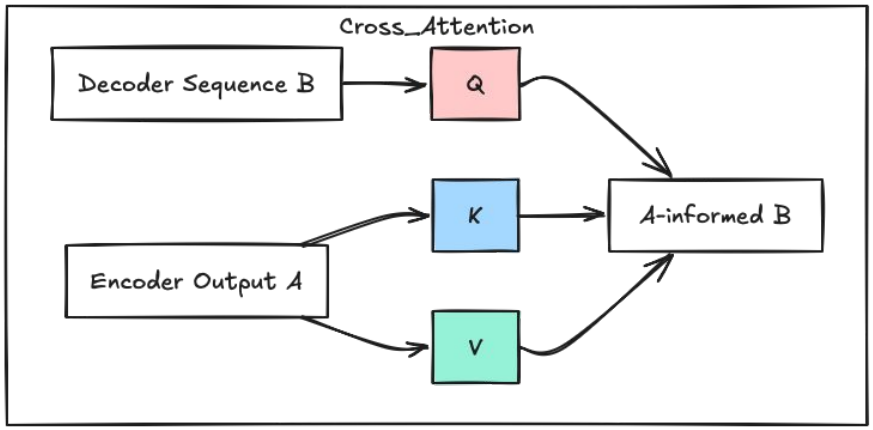
Attention Masking

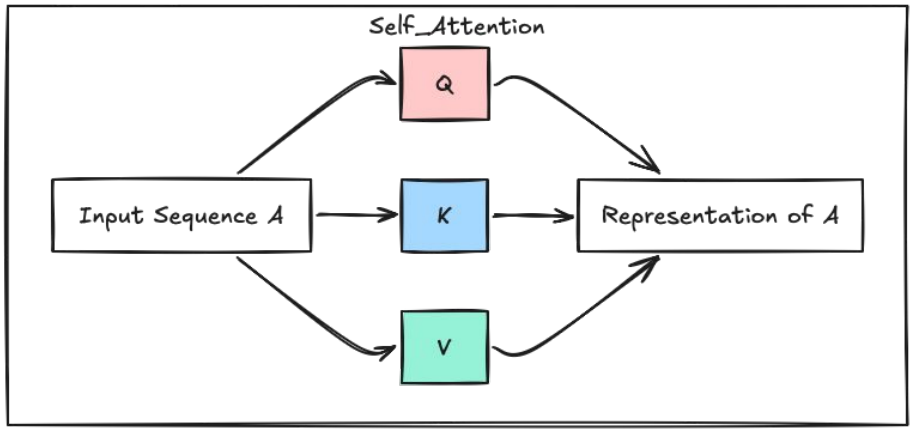






Self-Attention vs Cross-Attention

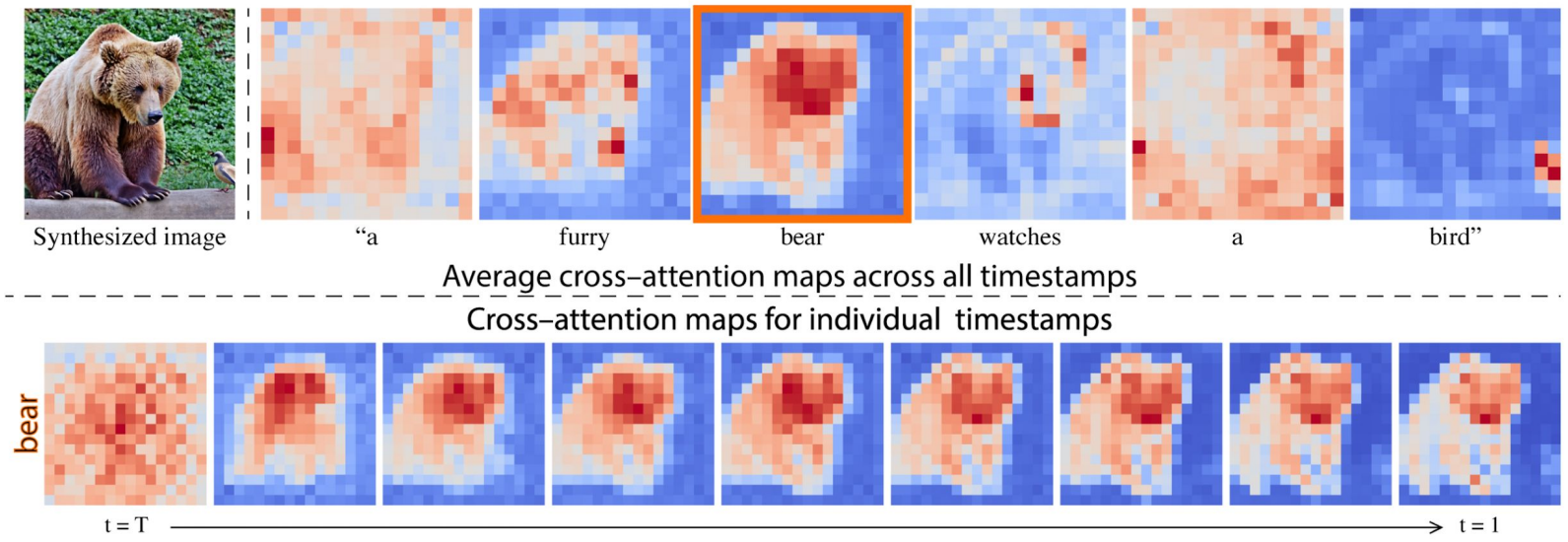




- In **Self-Attention**, the Queries (Q), Keys (K), and Values (V) all originate from the *same* input sequence (e.g., the sentence currently being processed).
- In **Cross-Attention**, the Query (Q) comes from the decoder (the sequence being generated), while the Keys (K) and Values (V) come from the encoder (the original source sequence).
- **The "Goal":** Self-attention helps a word understand its own context within its sentence; Cross-attention helps the model "look back" at the original input to decide what to generate next.
- Self-attention layers are found in both the Encoder and Decoder blocks, whereas Cross-attention specifically lives between them.

Cross-Attention

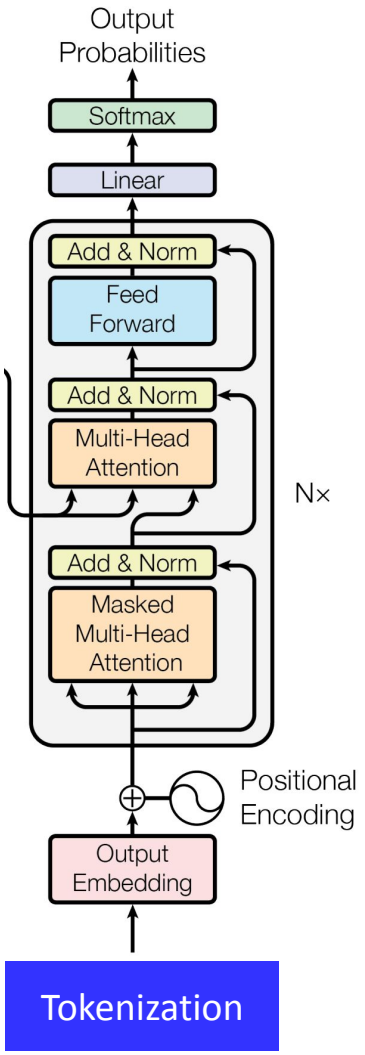
query, "a furry bear watches a bird."



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

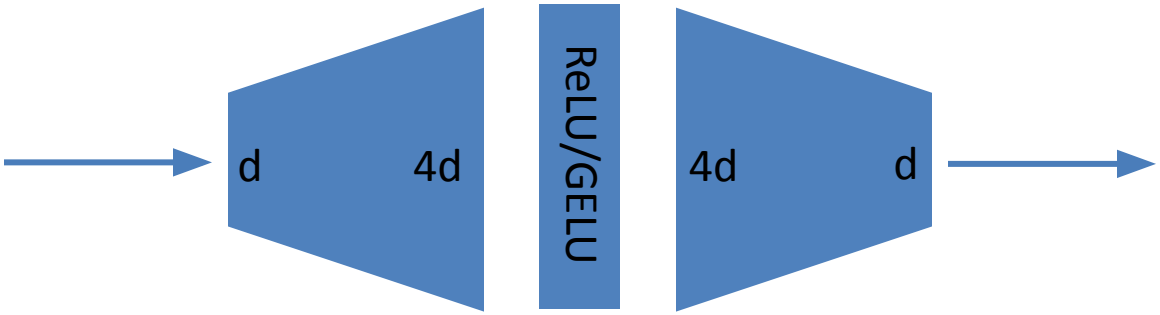
Transformer Architecture

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



Feed-Forward Block

- Just a MLP!

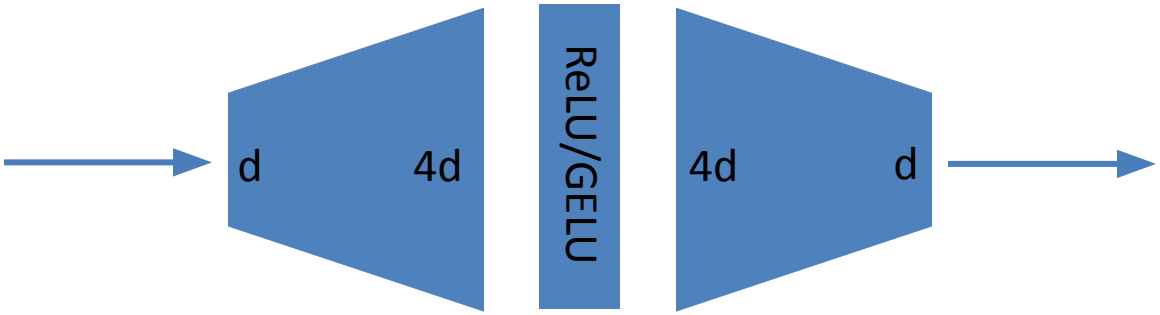


- This is where the main memorization happens!

https://www.youtube.com/watch?v=KJtZARuO3JY&t=694s;
Transformer Feed-Forward Layers Are Key-Value Memories" (Geva et al., 2020)

Feed-Forward Block

- Just a MLP!

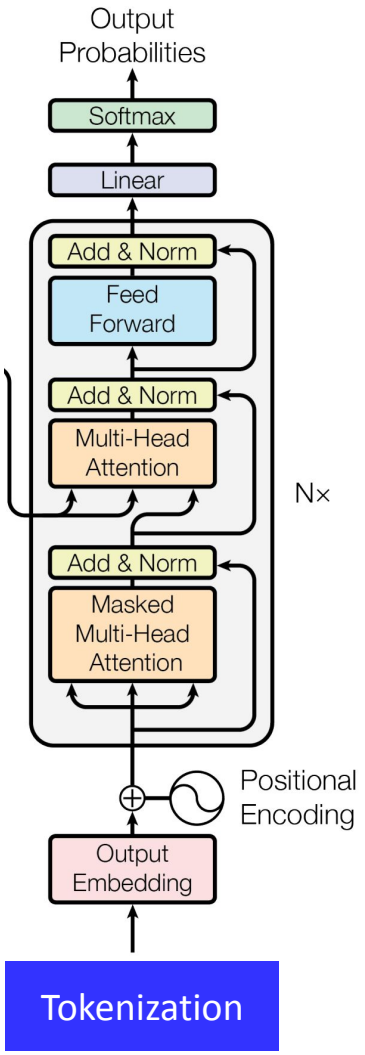


- Why expand: to let it understand context better ->4x more non-linearities
- More connections created -> more 'memory'

https://www.youtube.com/watch?v=KJtZARuO3JY&t=694s;
Transformer Feed-Forward Layers Are Key-Value Memories" (Geva et al., 2020)

Transformer Architecture

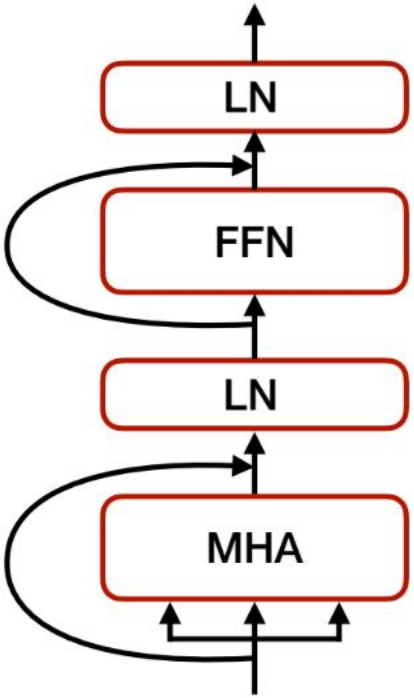
- Tokenization
- Embedding
- Positional Encoding
- (Masked) Multi-Head Attention
- Feed-Forward
- Add & Norm
- Output Projection Layer



Residual and Normalization

- Each layer in Transformer has:
  - A residual connection
  - A normalization layer

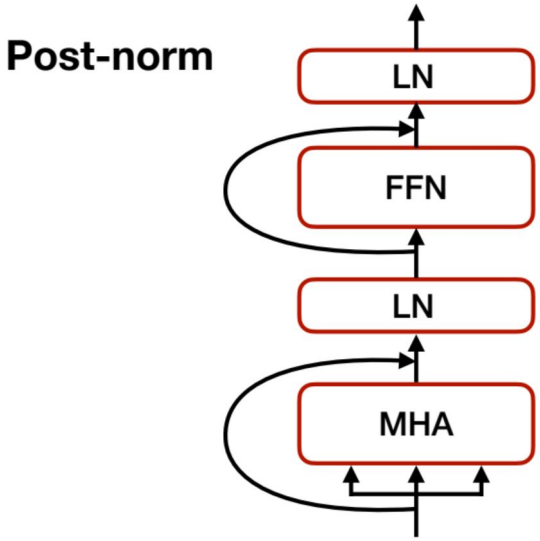
- Layer Norm. normalize each token by its embedding size dimension
  - For more stable training
  - Mean-Var norm along embedding dim



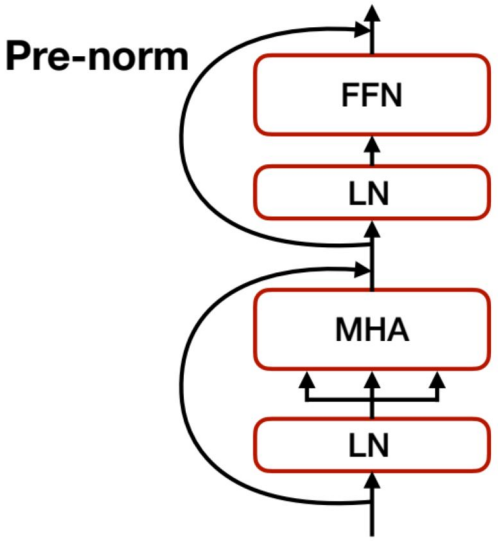
Position of Normalization

- Post-Norm vs Pre-Norm

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



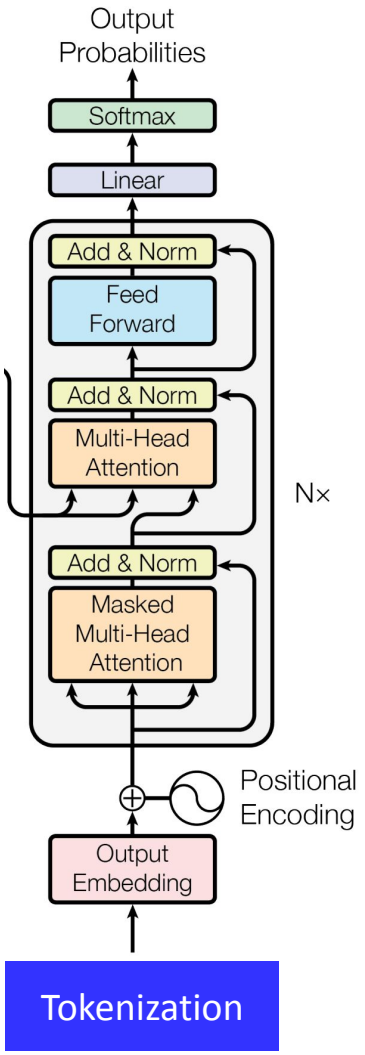
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + F_t\left(\mathrm{Norm}\left(\boldsymbol{x}_t\right)\right)$$



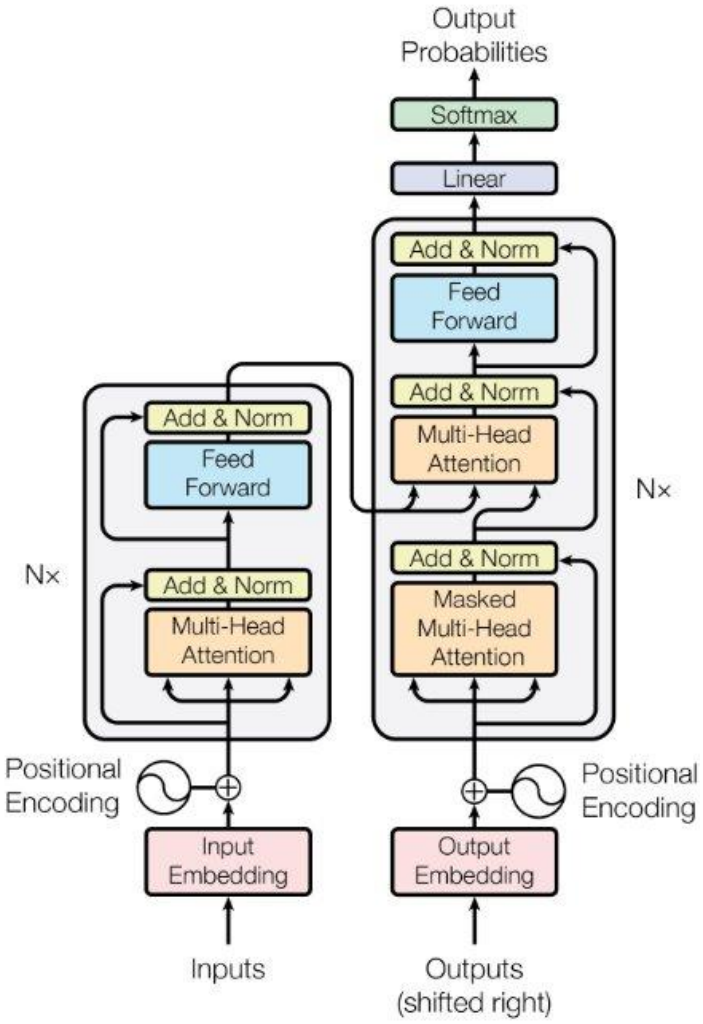
- Pre-Norm is easier and more stable to train
- Post-Norm tends to present better performance if properly trained

Transformer Architecture

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



Putting Them Together - Transformer



Poll @ TBD

Which of the following are true about self-attention?

- 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 affine transformations of input X

Poll @ TBD

Which of the following are true about self-attention?

- 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
- Q, K, V are affine transformations of input X

Content

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

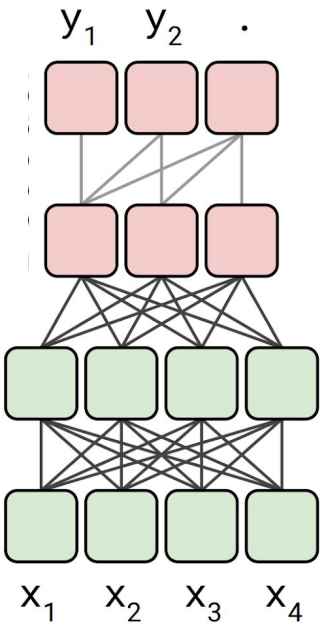
Overview

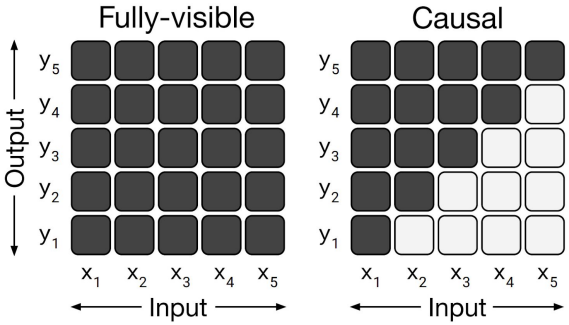
- Architecture
  - Encoder-Decoder
  - Encoder-Only
  - Decoder-Only
- Position Encoding
  - Relative Position Encoding
  - Rotary Position Encoding
- Efficient Attention Mechanism
  - Grouped Query Attention
  - Multi Query Attention
  - Flash Attention
  - Multi-head Latent Attention

Overview

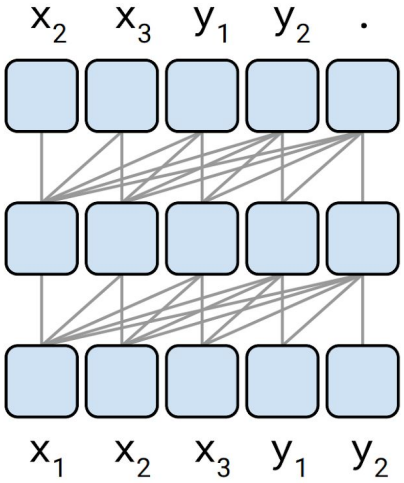
- Architecture
  - Encoder-Decoder
  - Encoder-Only
  - Decoder-Only
- Position Encoding
  - Relative Position Encoding
  - Rotary Position Encoding
- Efficient Attention Mechanism
  - Grouped Query Attention
  - Multi Query Attention
  - Flash Attention
  - Multi-head Latent Attention

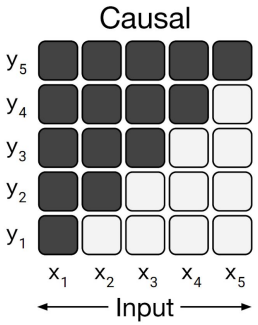
Encoder-Decoder



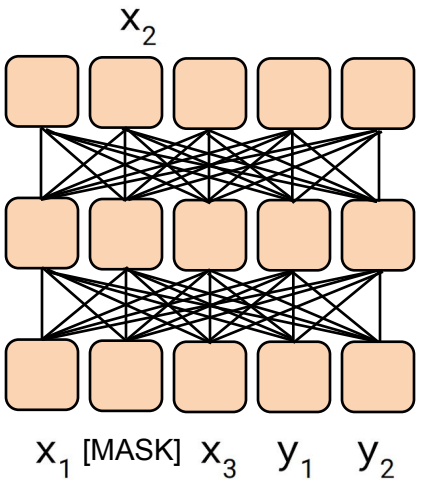


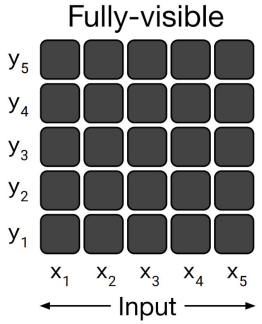
Decoder-Only





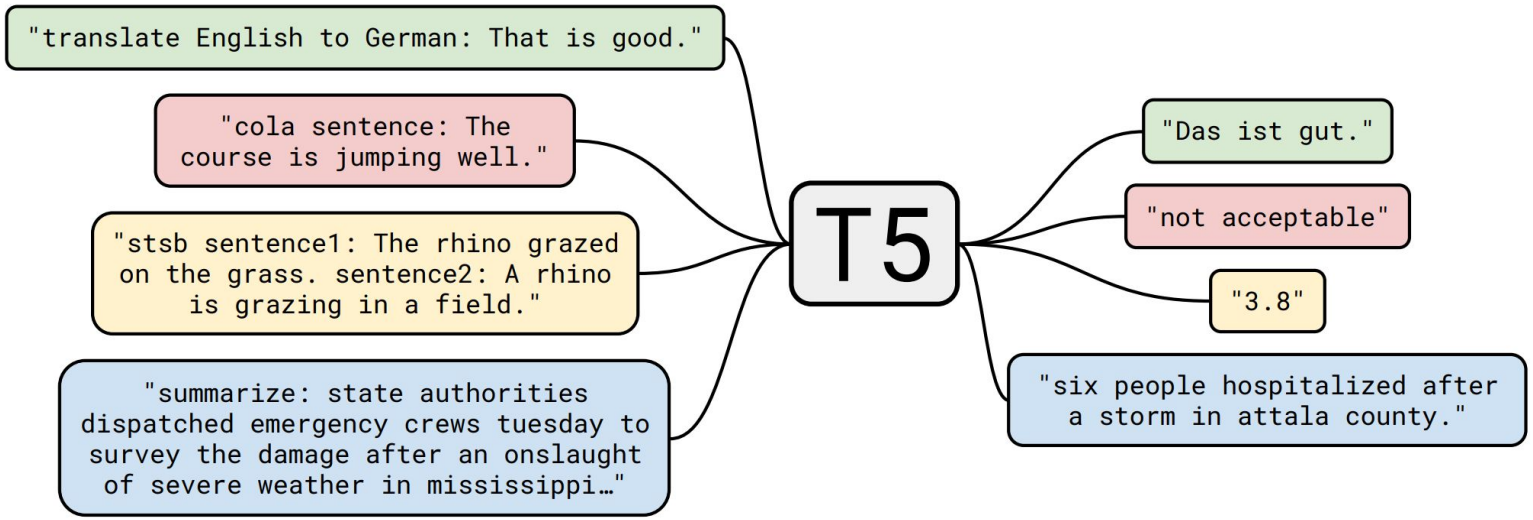
Encoder-Only





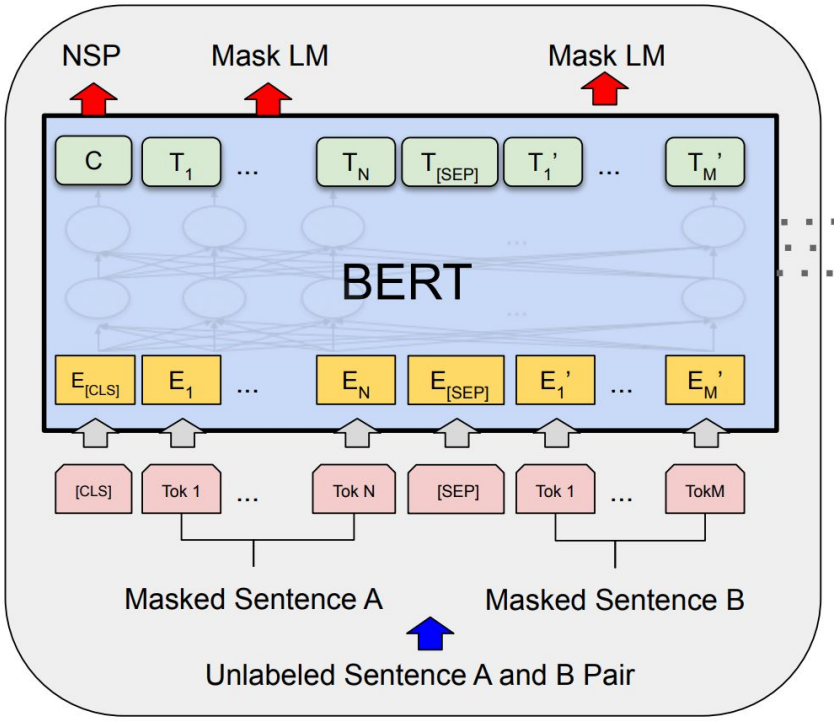
Encoder-Decoder - T5

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



Encoder-Only - BERT

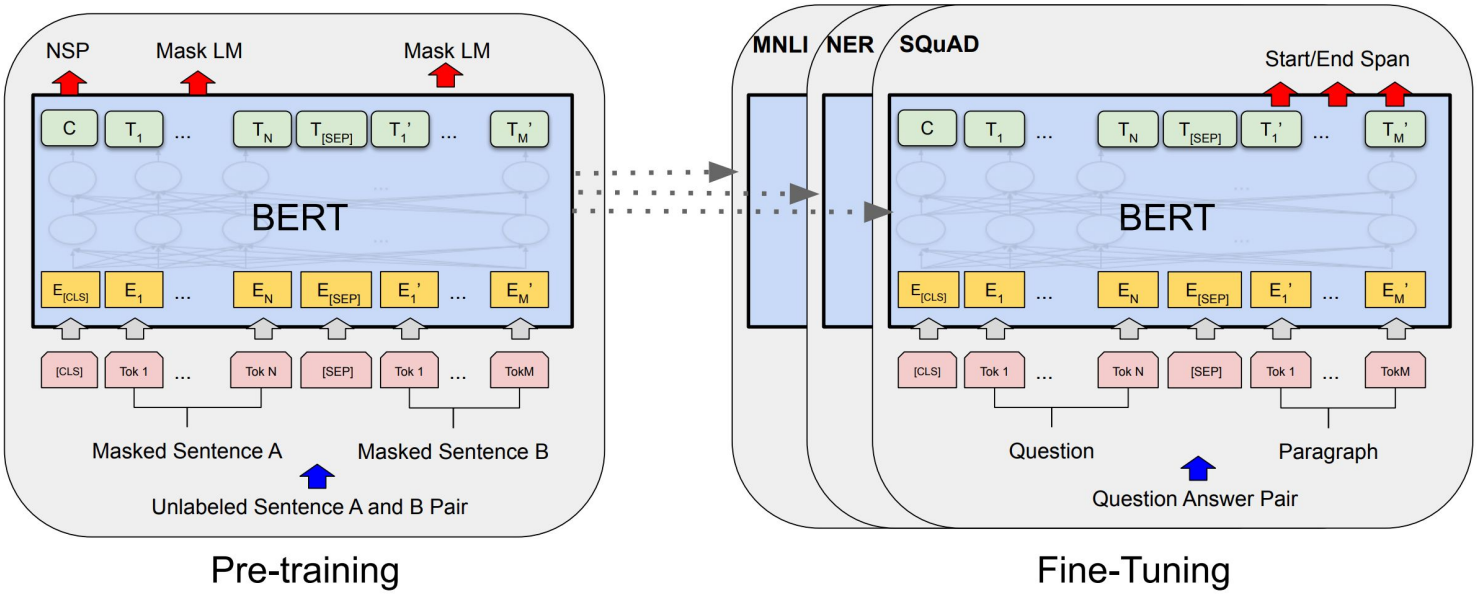
- Bidirectional Encoder Representations from Transformers (BERT)
  - Encoder-only arch.
- Trained with
  - Mask token prediction
  - Next sentence prediction



Pre-training and then Fine-Tuning

Pre-training on a proxy task
- Masked token prediction
- Next sentence prediction

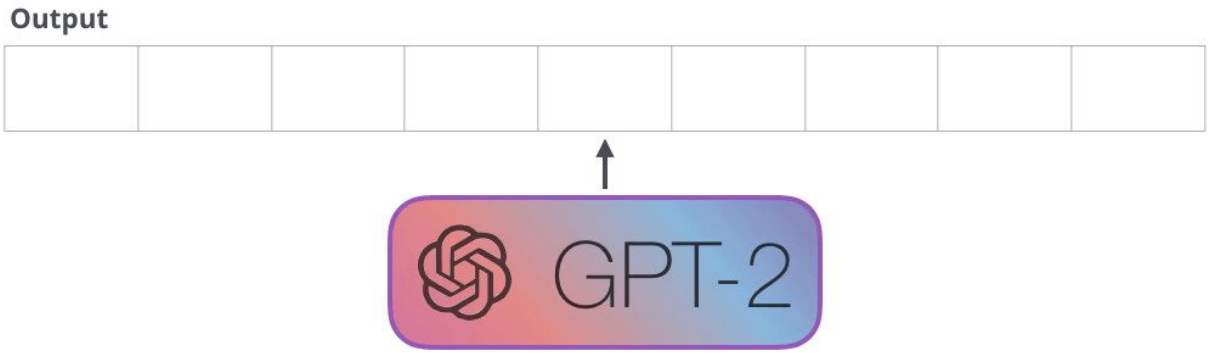
Fine-tuning on specific downstream tasks
- Machine translation
- Question answering



Decoder-Only - GPT

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

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



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
  - ChatGPT!
- GPT-4
  - Strong reasoning and multimodal capabilities (text + image)
  - More reliable and aligned than GPT-3
- GPT-5
  - Improved reasoning, planning, and tool use
  - Deeper multimodal integration (text, image, audio, video)
  - Stronger long-context understanding

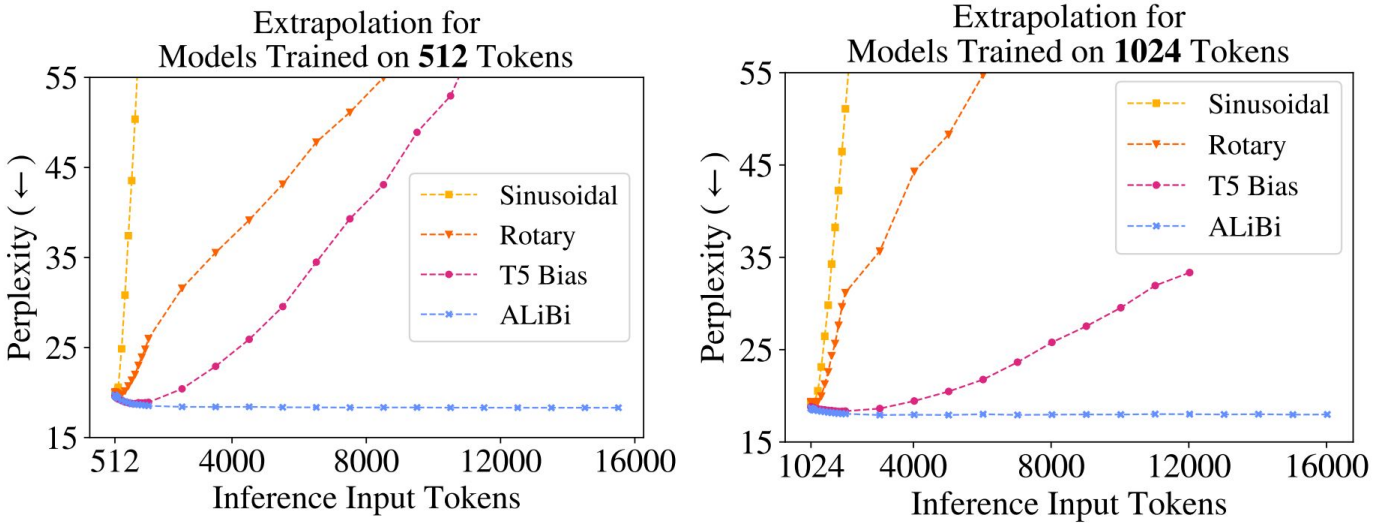
Overview

- 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

Relative Position Encoding

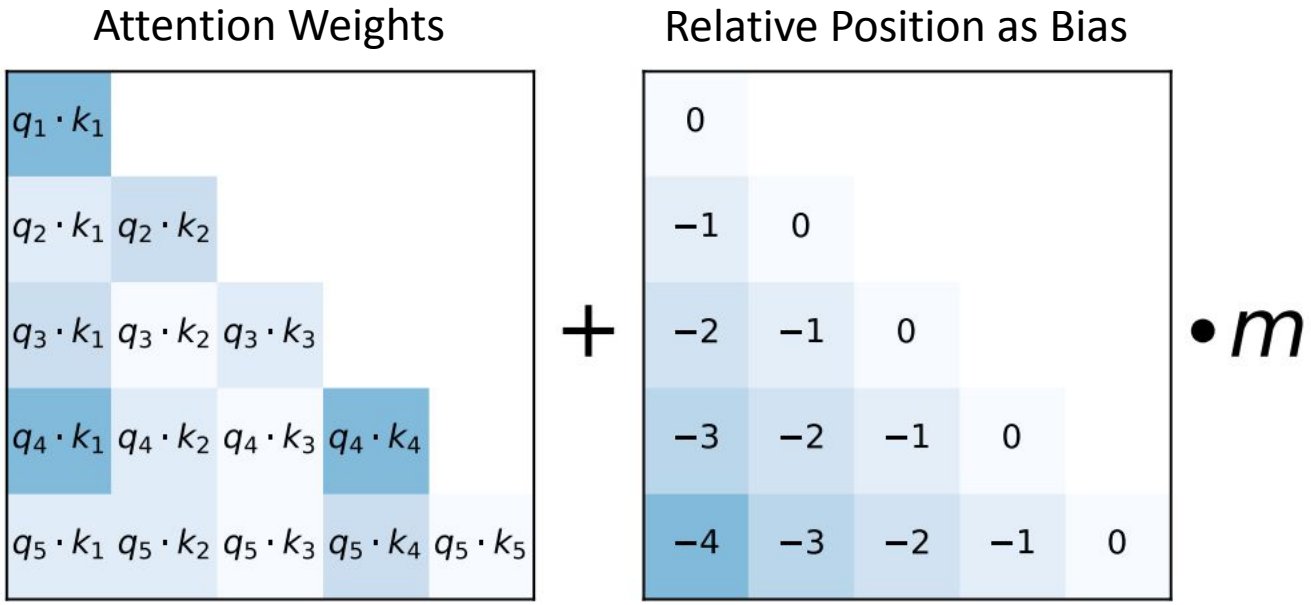
- Relative position embedding fuses position information into attention matrices

- 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

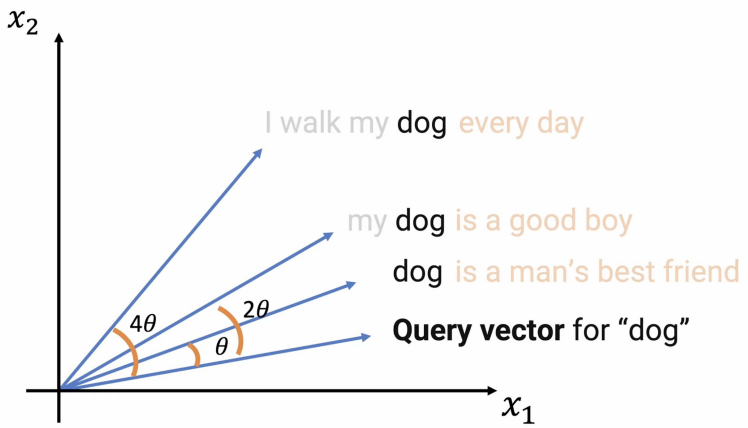
- Used in Large Language Models such as LLAMA
- Rotate the embedding in 2D space

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

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

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

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



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

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

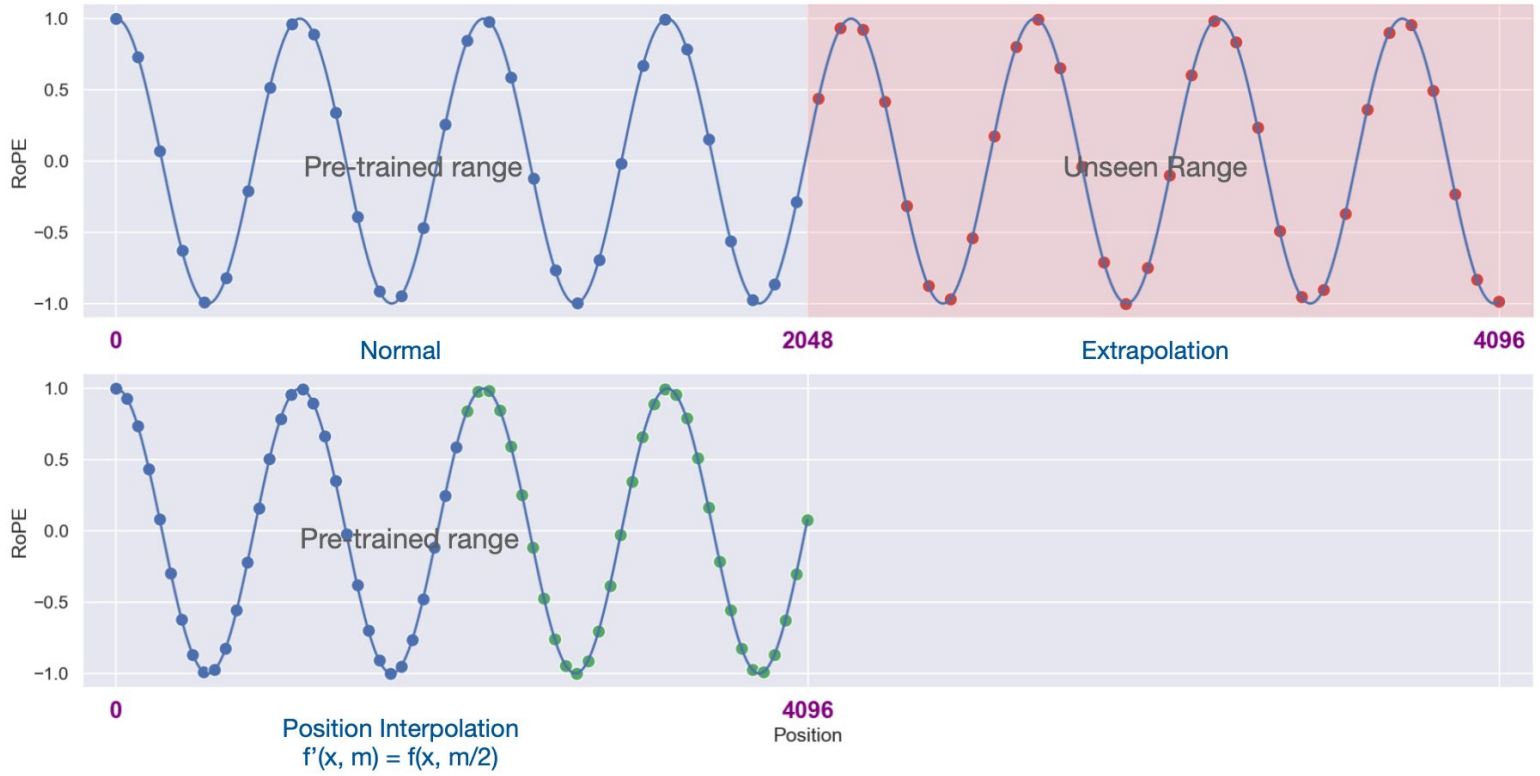
$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

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

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

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

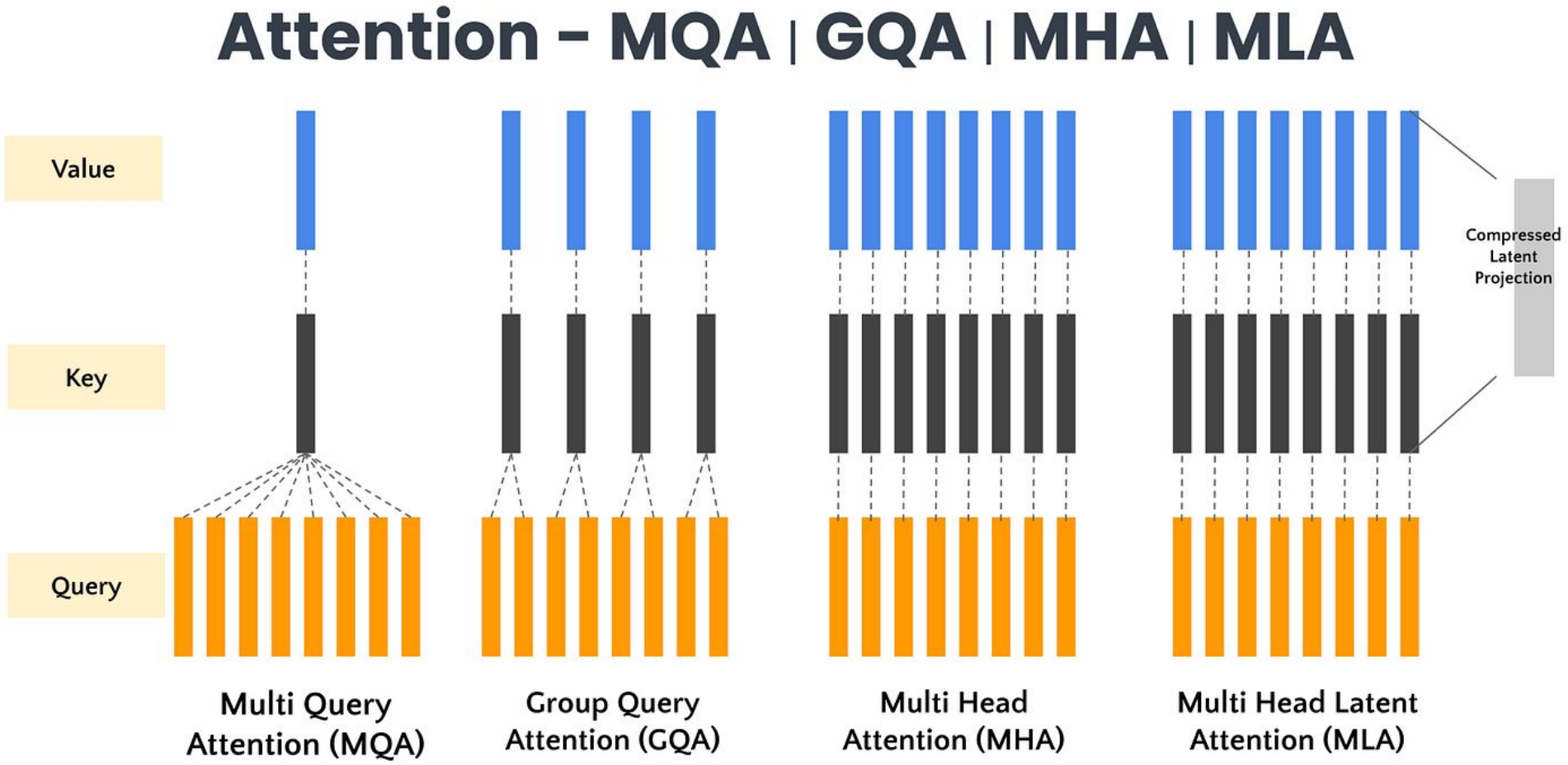
Quadratic Complexity

- Self-attention has quadratic complexity to input length
  - $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$ FLOPS

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

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

Attention Types



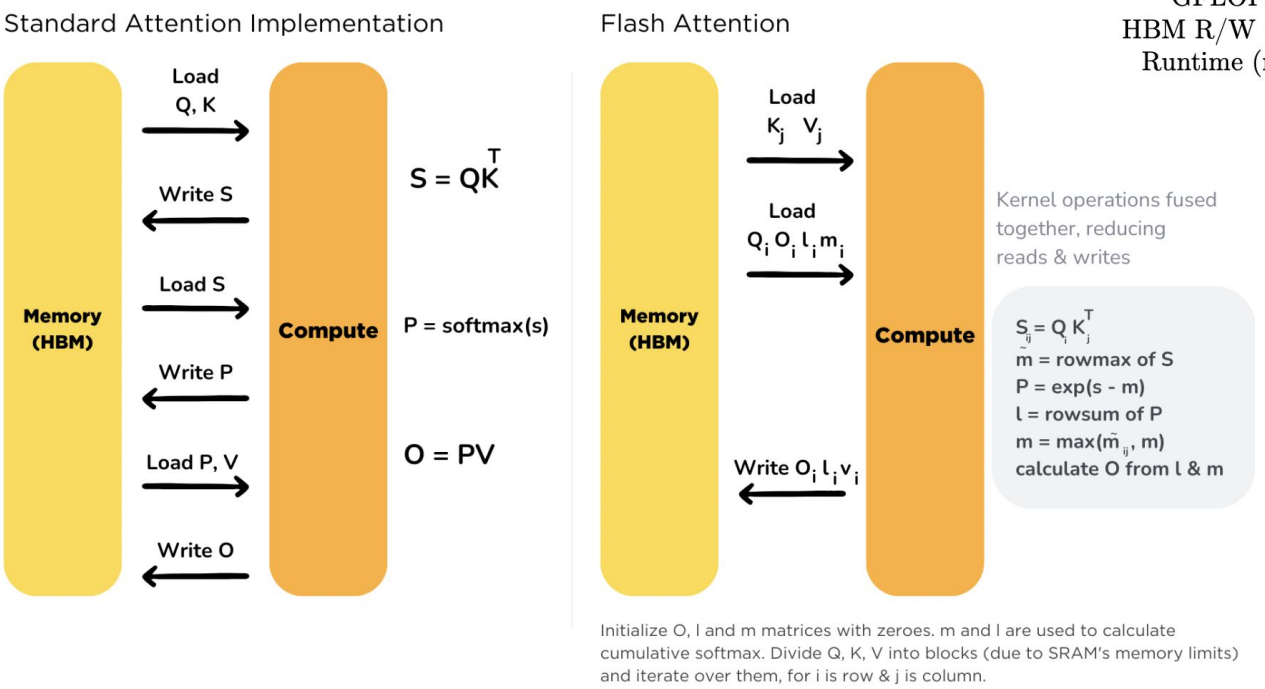
Liu, Aixin, et al. "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model." arXiv preprint arXiv:2405.04434 (2024)

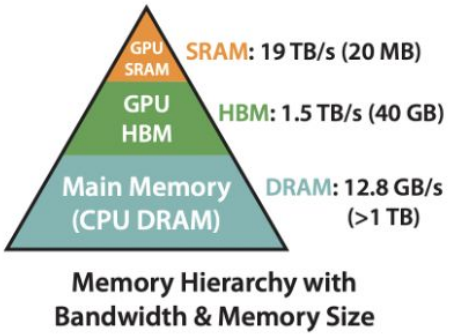
https://verticalserve.medium.com/group-query-attention-58283b337c65

Flash Attention

- Reduce GPU <-> CPU memory trips

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





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

- Classic GPU trick – tiled multiplication

A

1	2	0	5
4	8	2	-1
3	1	6	3
-7	5	0	8

x

B

3	1	6	3
-7	5	0	8
1	2	0	5
0	3	5	1

=

C

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

$C_{1,1} = 1 \times 1 + 2 \times 5 + 3 \times 9 + 4 \times 13$
$C_{1,2} = 1 \times 2 + 2 \times 6 + 3 \times 10 + 4 \times 14$
$C_{2,1} = 5 \times 1 + 6 \times 5 + 7 \times 9 + 8 \times 13$
$C_{2,2} = 5 \times 2 + 6 \times 6 + 7 \times 10 + 8 \times 14$
.....
.....

With Tiling 16 access

A

1	2	0	5
4	8	2	-1
3	1	6	3
-7	5	0	8

x

B

3	1	6	3
-7	5	0	8
1	2	0	5
0	3	5	1

=

C

C1,1	C1,2
C2,1	C2,2

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

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

- online softmax - revitalizes safe softmax

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**

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

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**

$$\begin{aligned} m_i &\leftarrow \max\left(m_{i-1}, x_i\right) \\ d'_i &\leftarrow d'_{i-1} e^{m_{i-1} - m_i} + e^{x_i - m_i} \end{aligned}$$

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

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:

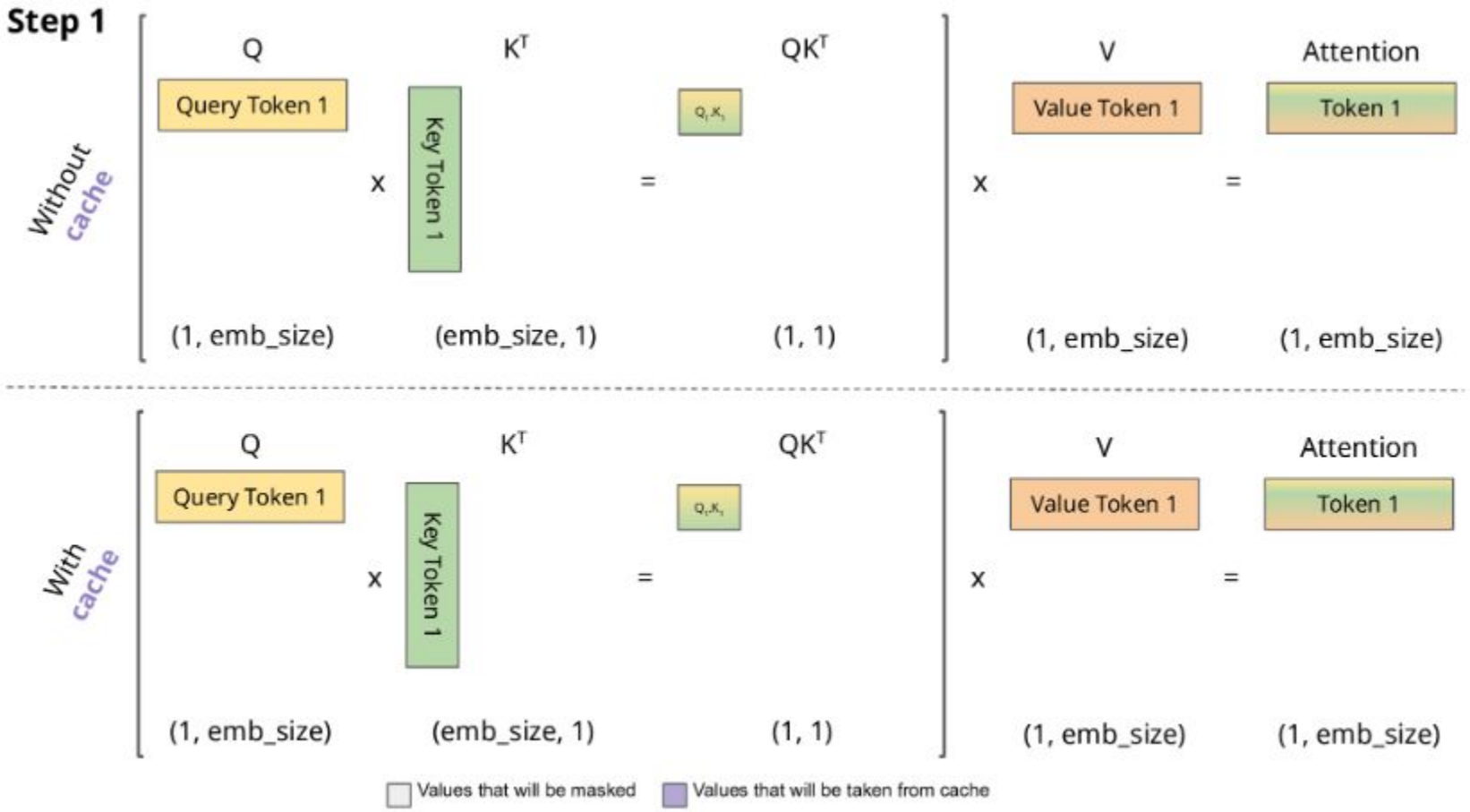
Flash Attention: $\mathbf{O}\left(\frac{N^2 d^2}{M}\right)$

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

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

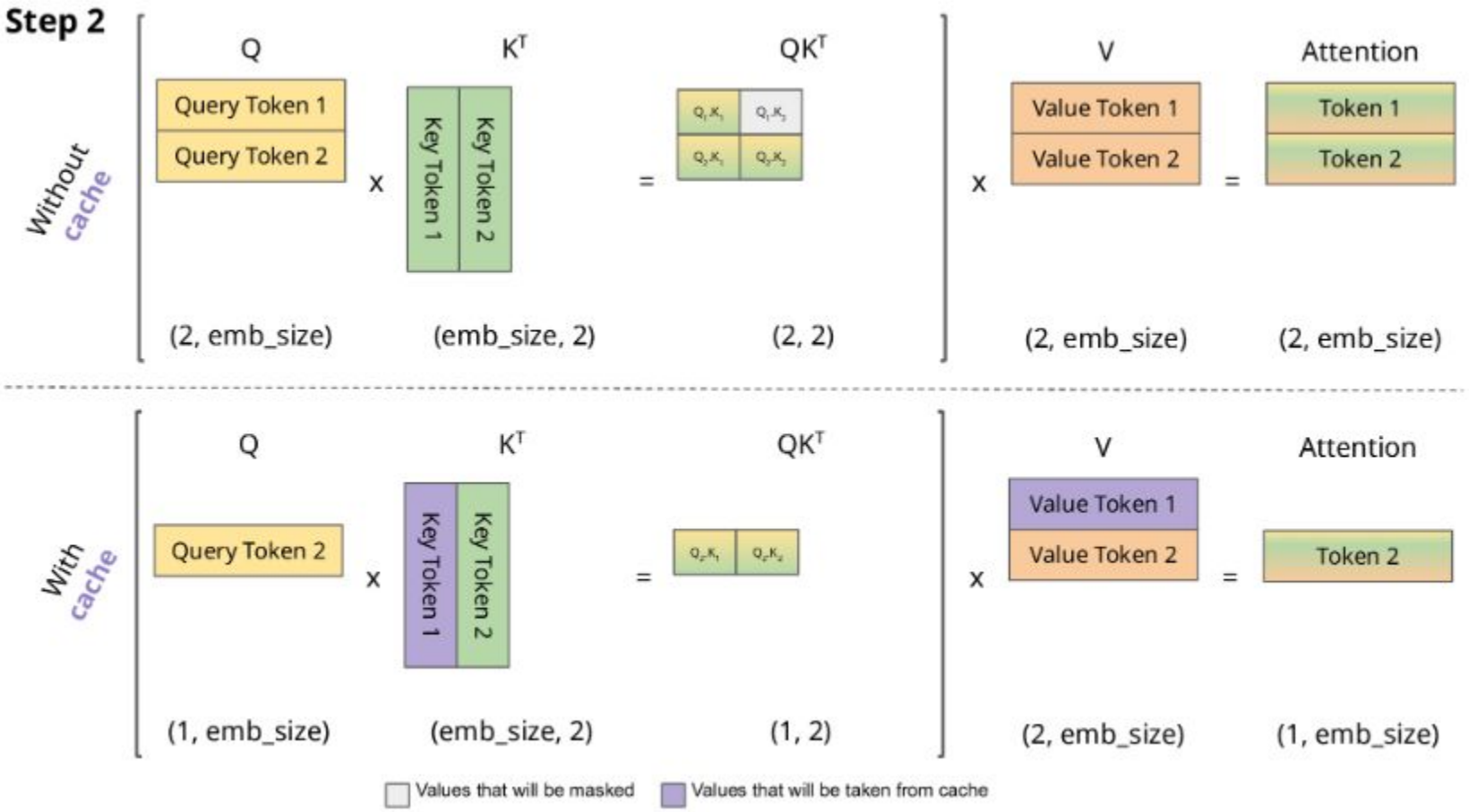
Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." Advances in Neural Information Processing Systems 35 (2022): 16344-16359.

KV- Caching



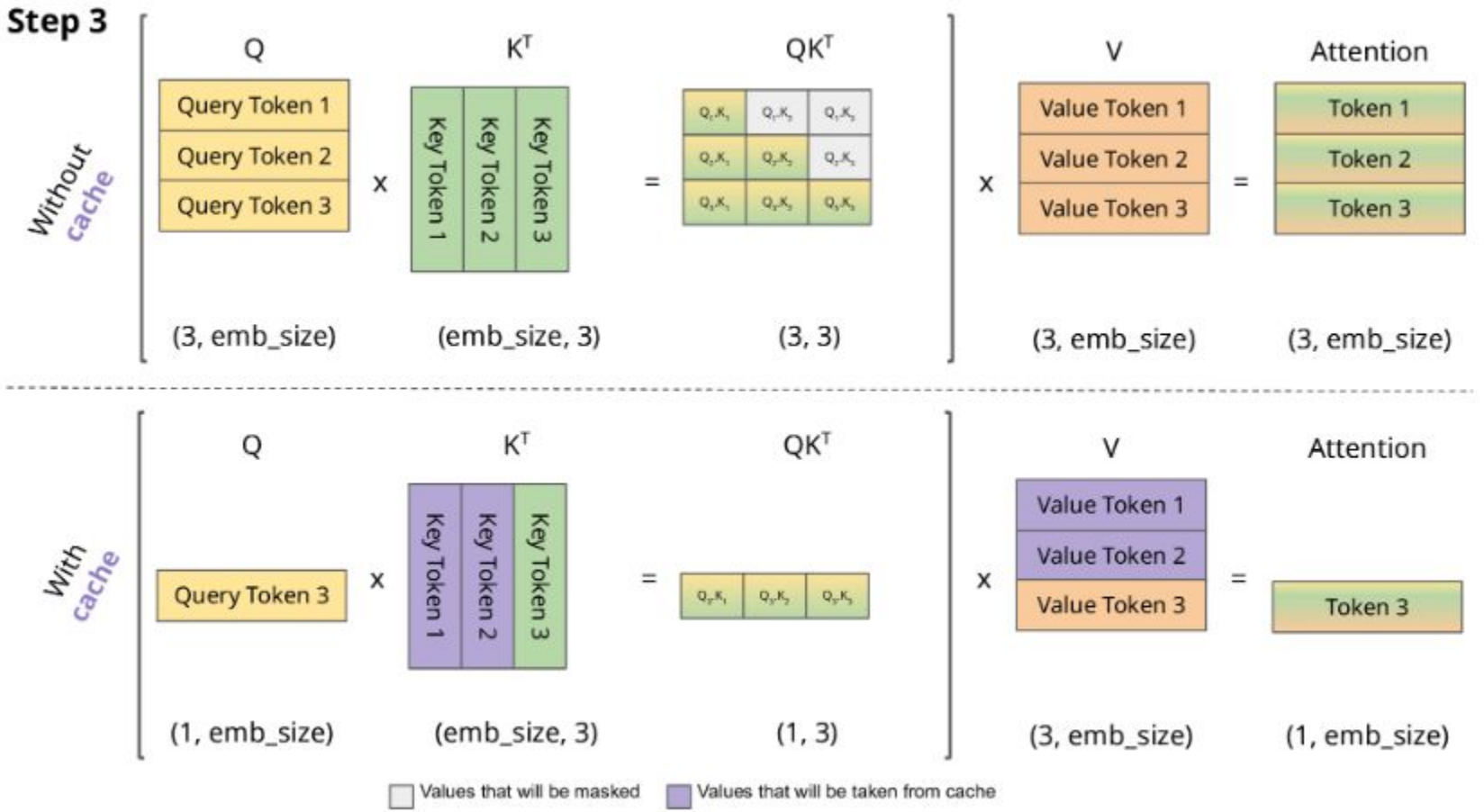
Transformers KV Caching Explained: https://medium.com/@joaolages/kv-caching-explained-276520203249

KV- Caching



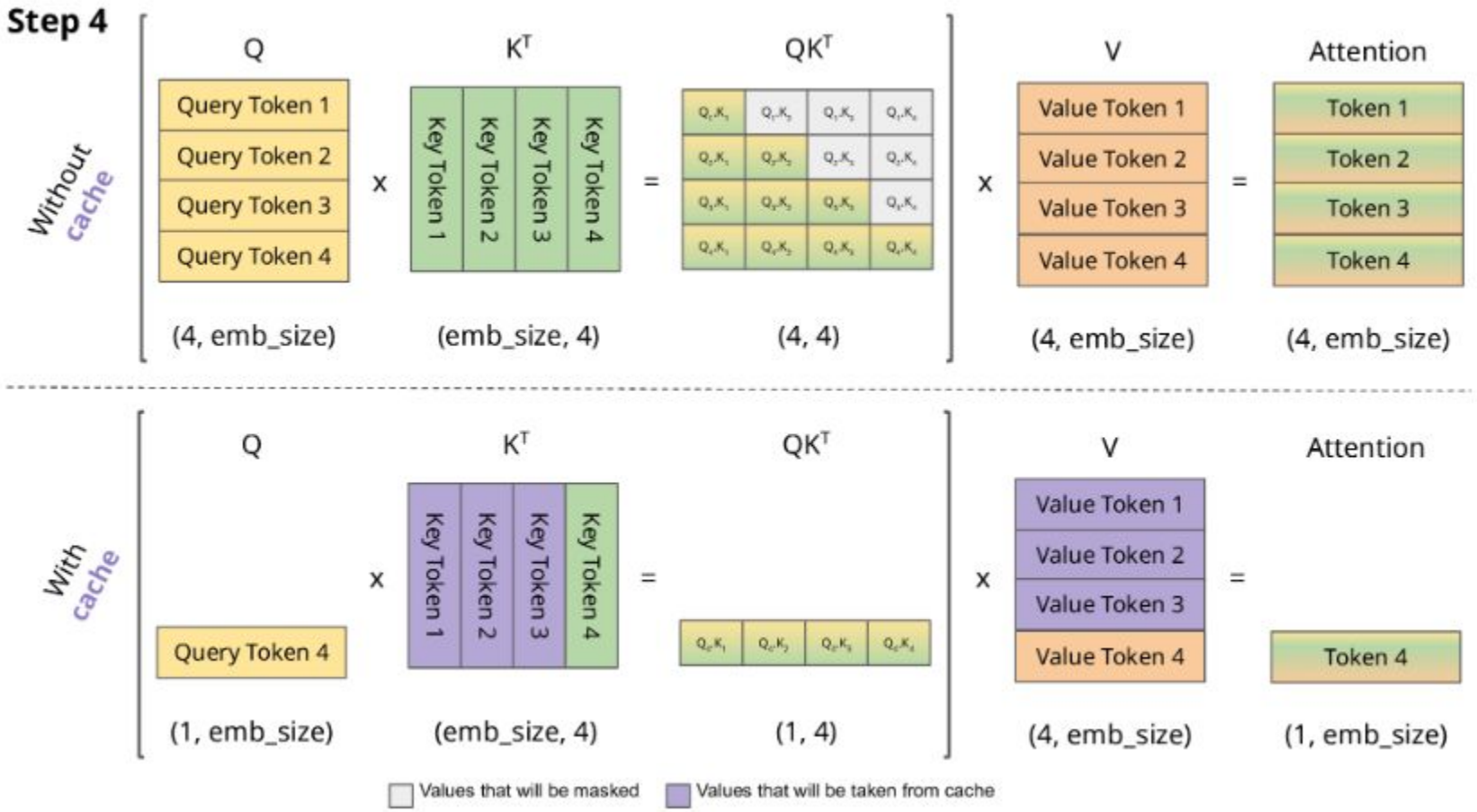
Transformers KV Caching Explained: https://medium.com/@joaolages/kv-caching-explained-276520203249

KV- Caching



Transformers KV Caching Explained: https://medium.com/@joaolages/kv-caching-explained-276520203249

KV- Caching



Transformers KV Caching Explained: https://medium.com/@joaolages/kv-caching-explained-276520203249

Poll @ TBD

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 @ TBD

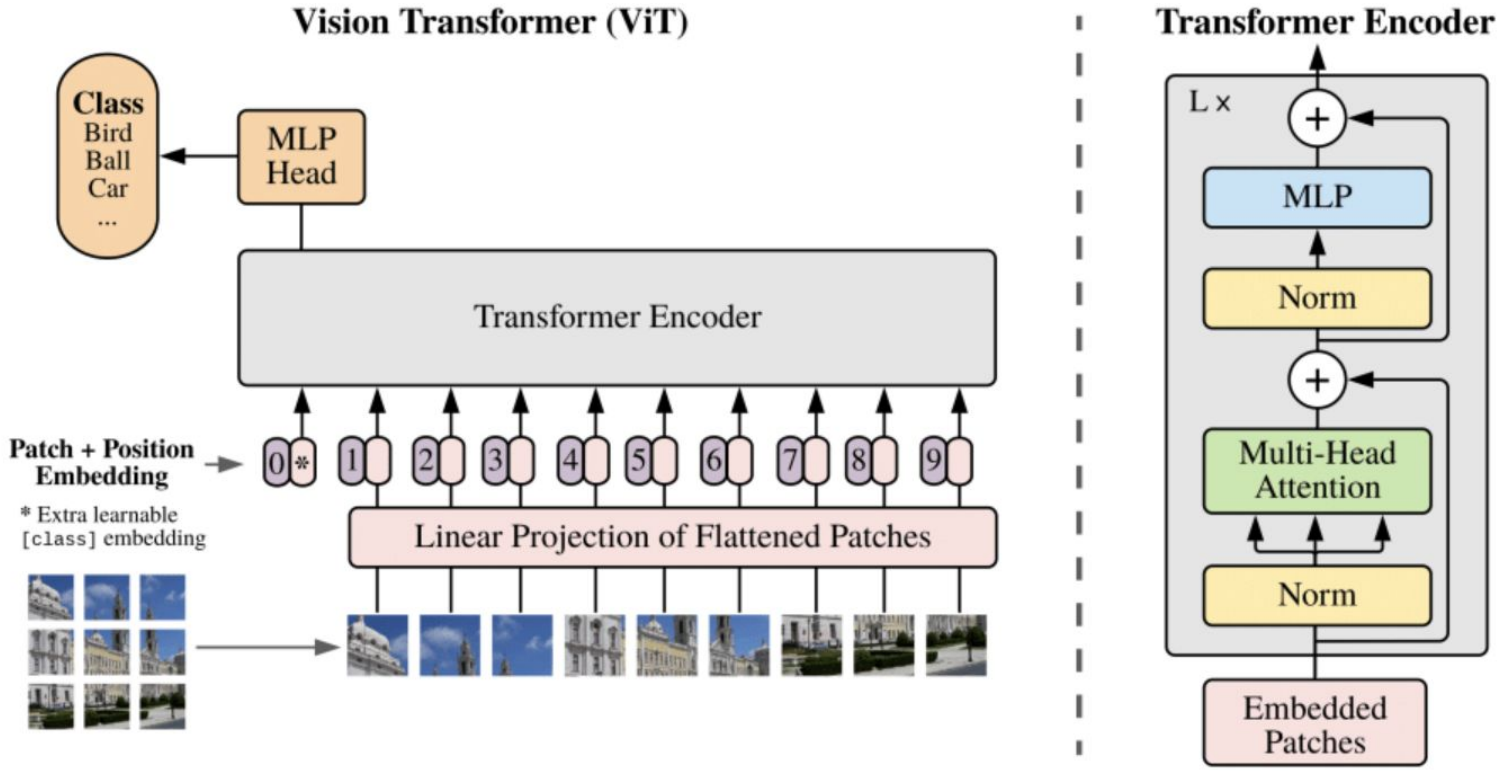
Which of the following statements is true?

- 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
- Quantizing Transformers

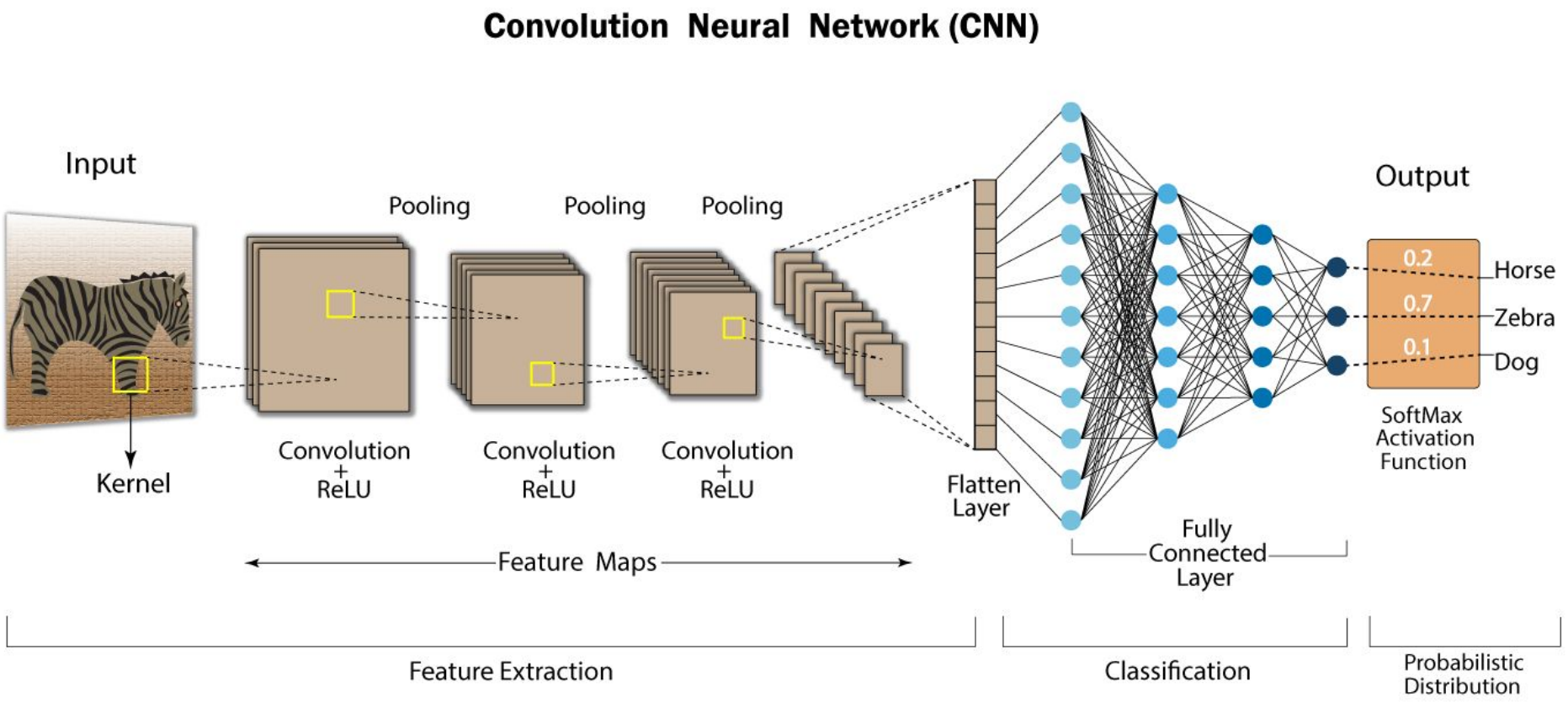
Vision Transformer (ViT)



- 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

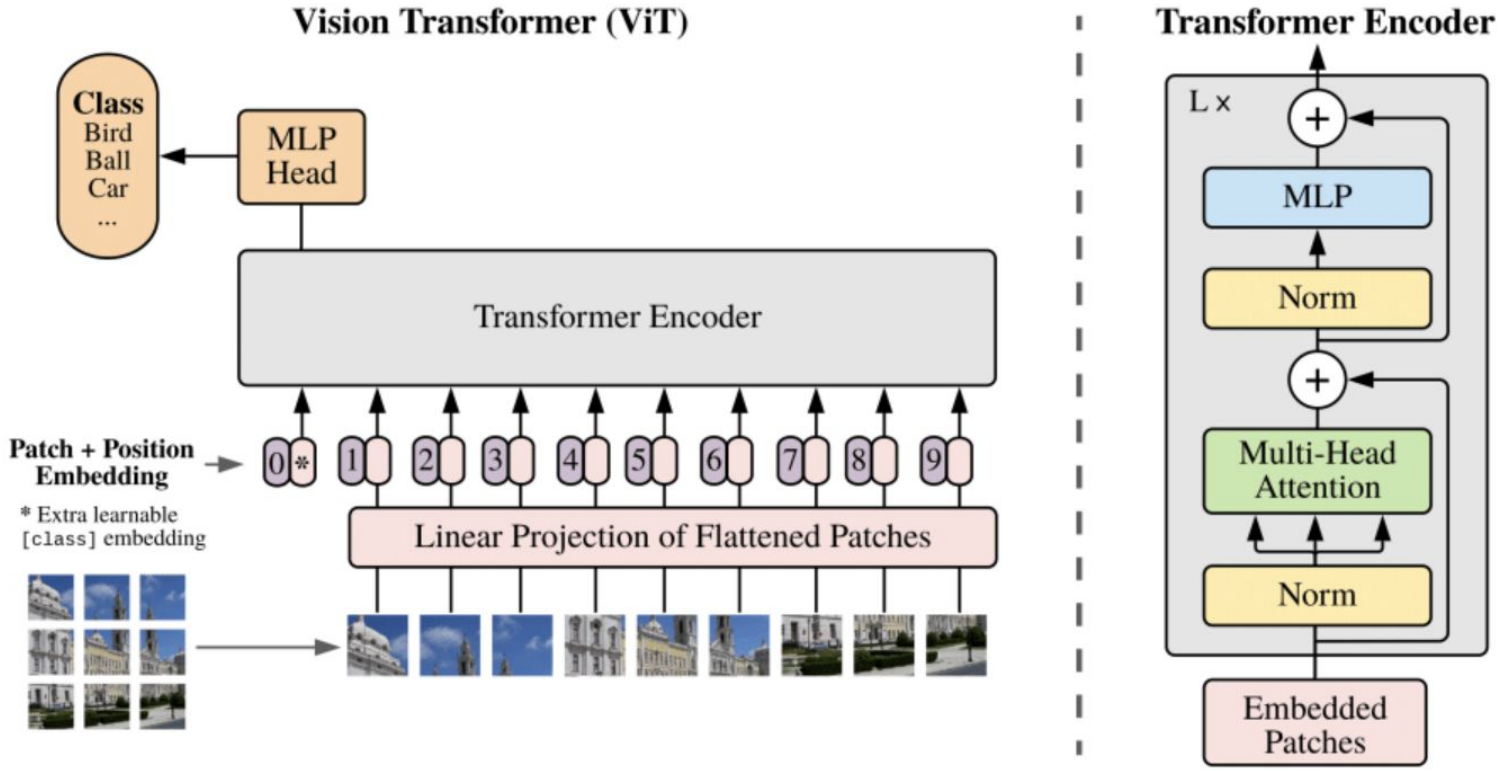
ViT

- 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

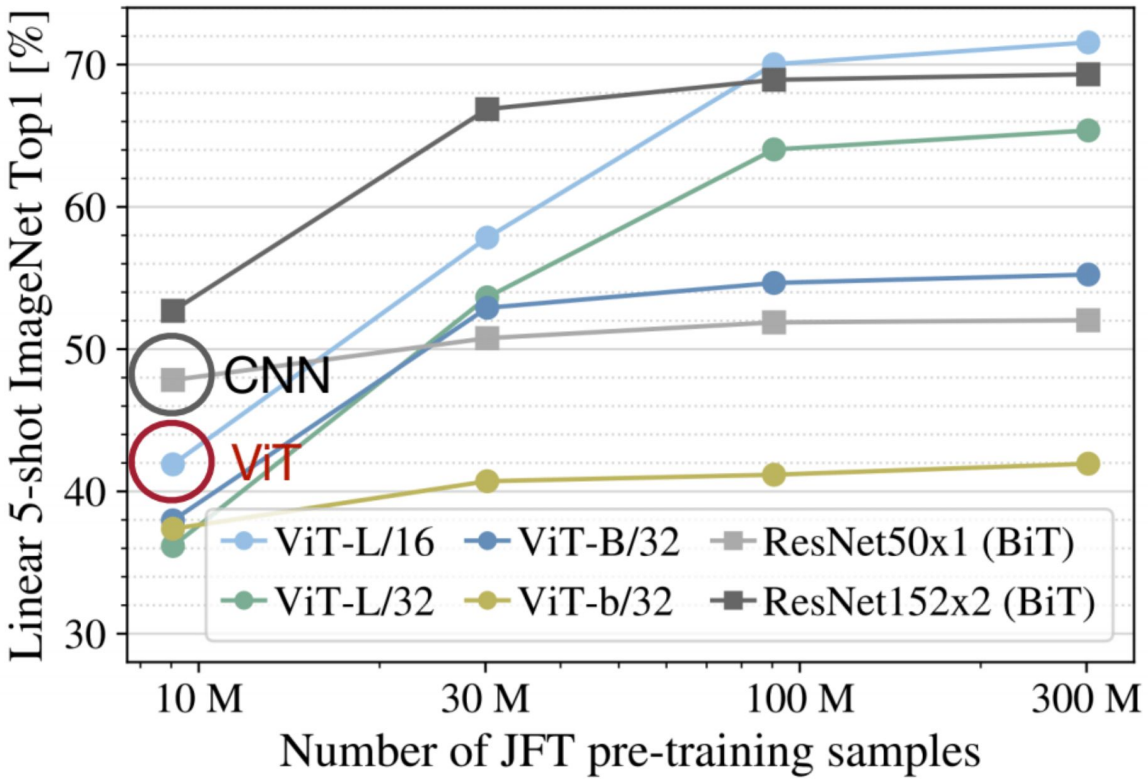
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

ViT



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

Image Classification

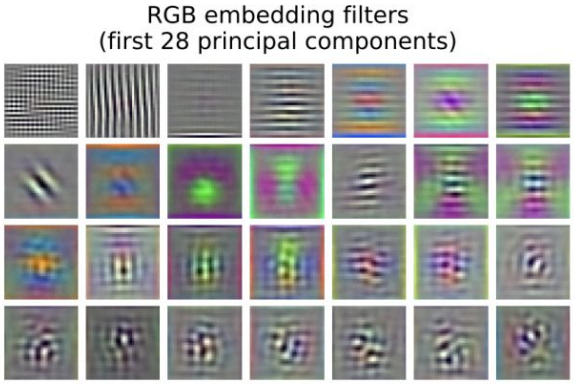


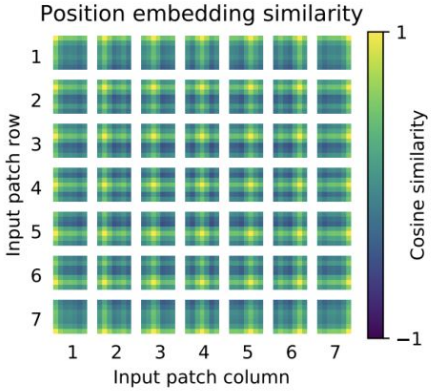
- Inferior performance compared to CNN when dataset size is limited – Why?

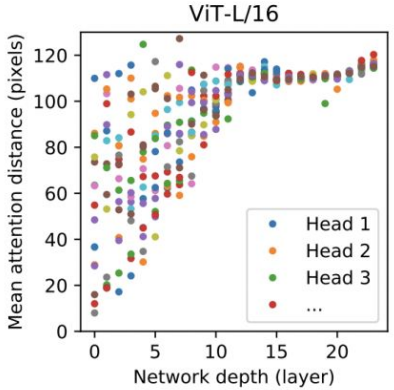
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

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!

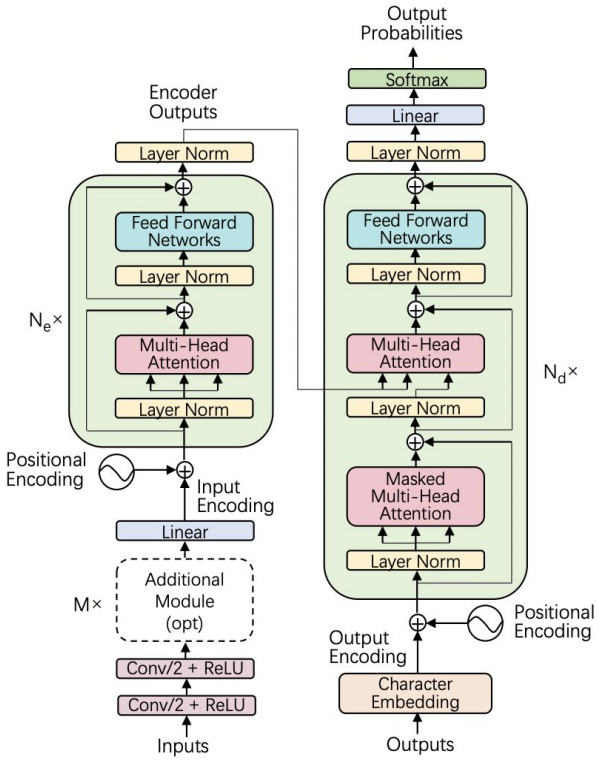




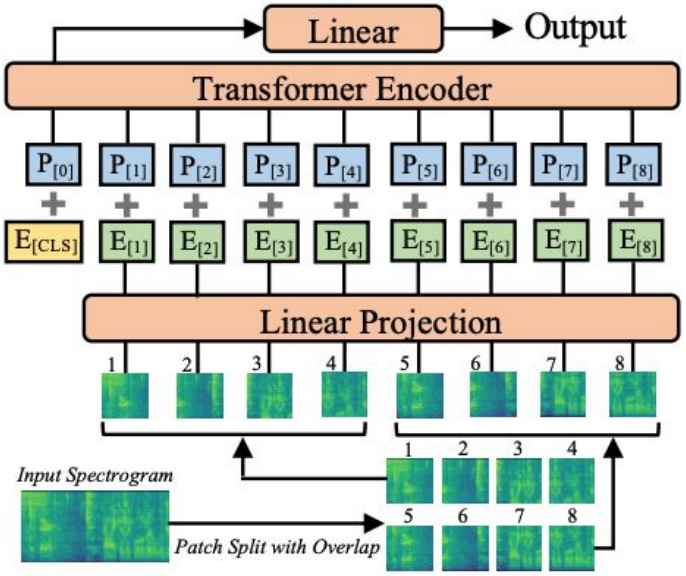


Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

Transformer in Audio



Speech Transformer for ASR



Audio Spectrogram Transformer

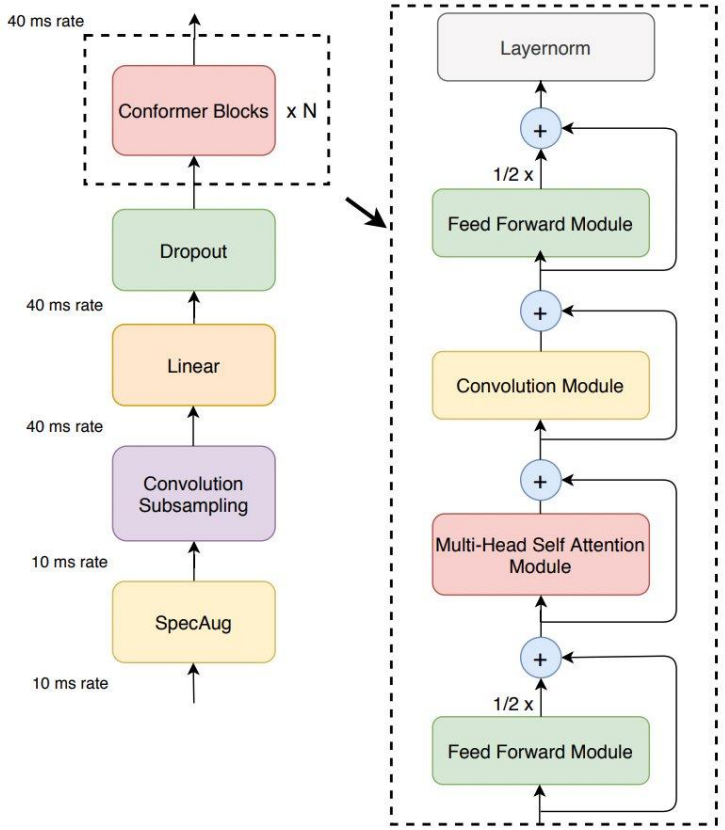
[1] Dong, Linhao, Shuang Xu, and Bo Xu. "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition." *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018.

[2] Gong, Yuan, Yu-An Chung, and James Glass. "Ast: Audio spectrogram transformer." *arXiv preprint arXiv:2104.01778* (2021).

Conformer

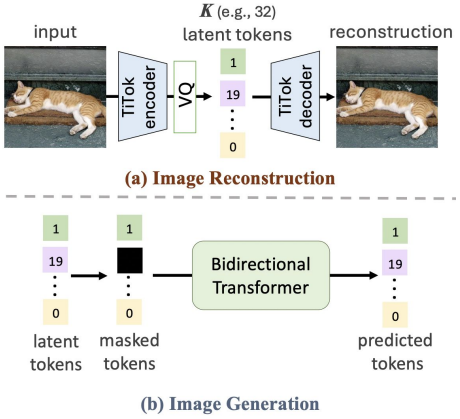
- The Conformer architecture augments a transformer by embedding convolution layers within the transformer blocks.

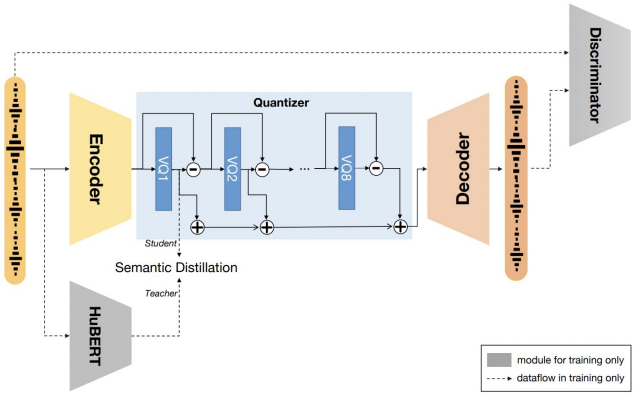
- Transformers capture global dependencies, CNNs capture local features efficiently.

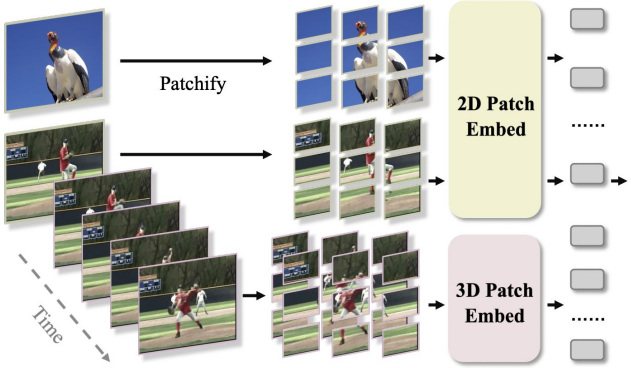


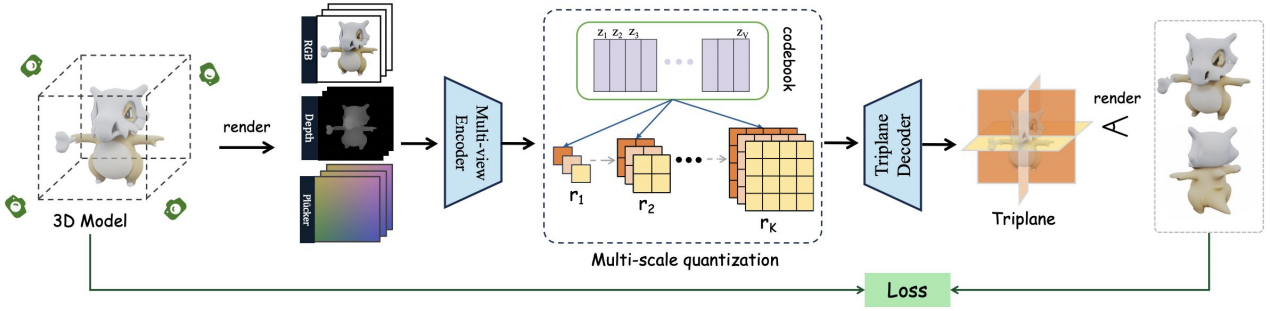
Gulati, Anmol, et al. "Conformer: Convolution-augmented transformer for speech recognition." *arXiv preprint arXiv:2005.08100* (2020).

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 @ TBD

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 @ TBD

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
- Quantizing Transformers
- Interpreting Transformer – attention, logitlens

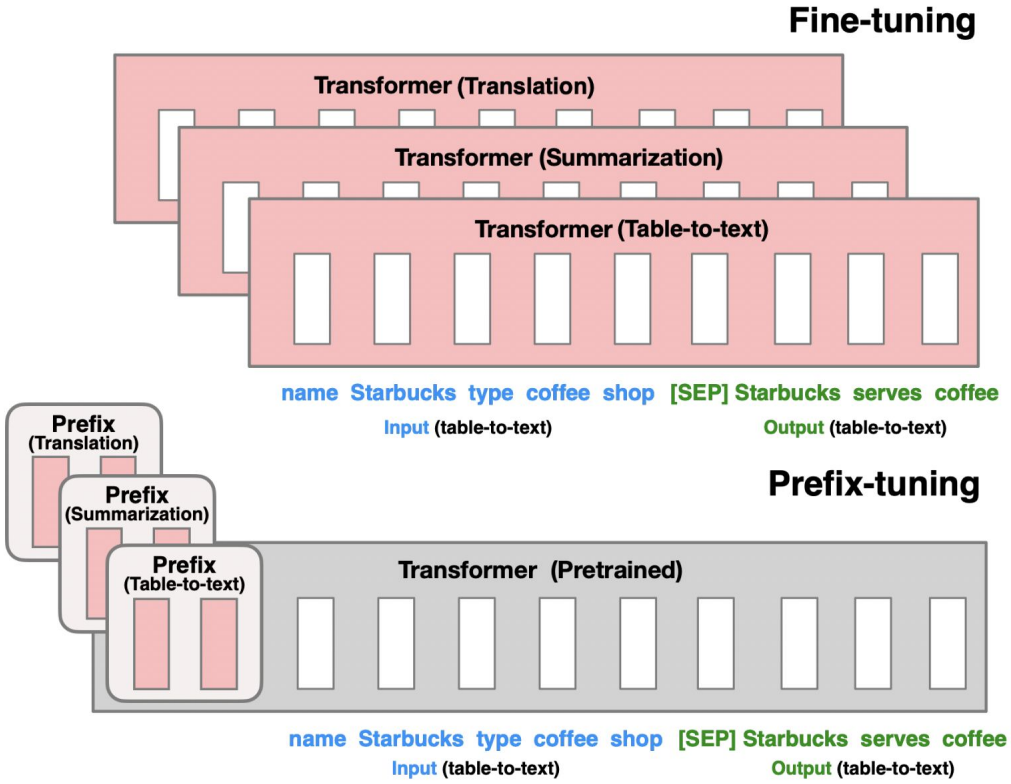
Parameter Efficient Tuning

- 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

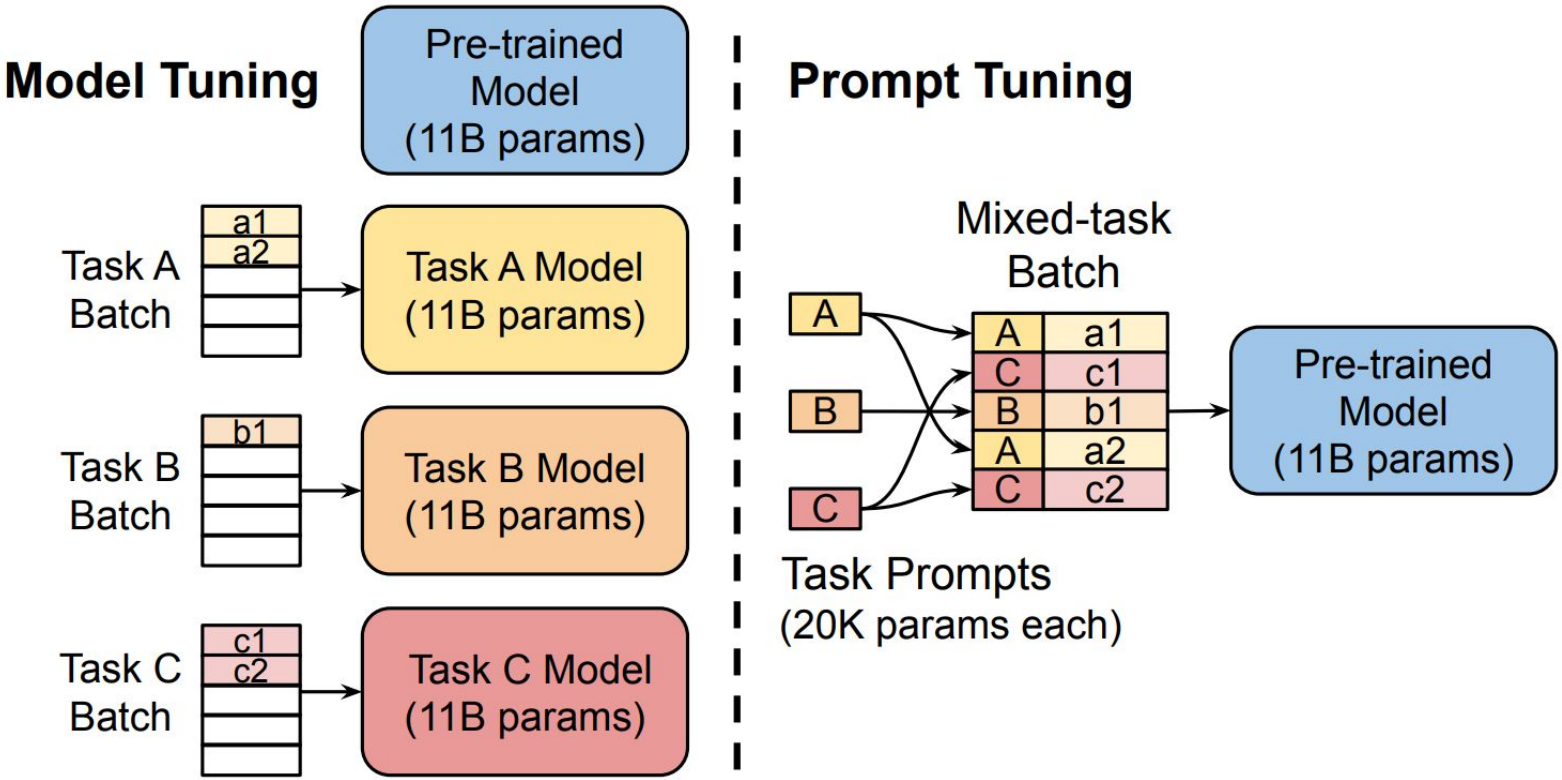
- Only learns a set continuous prefixes)added to the input and transformer layers for each task.



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

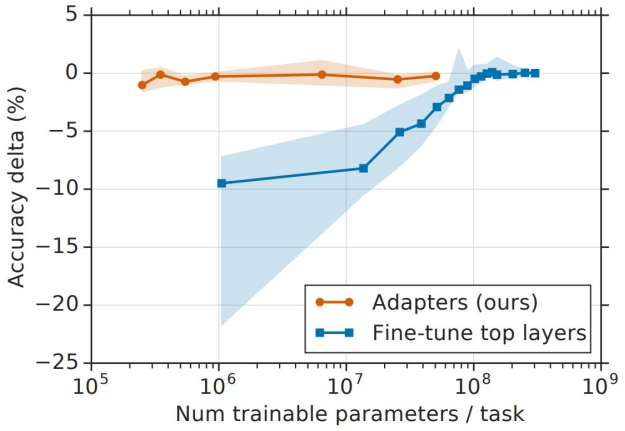
Prompt Tuning

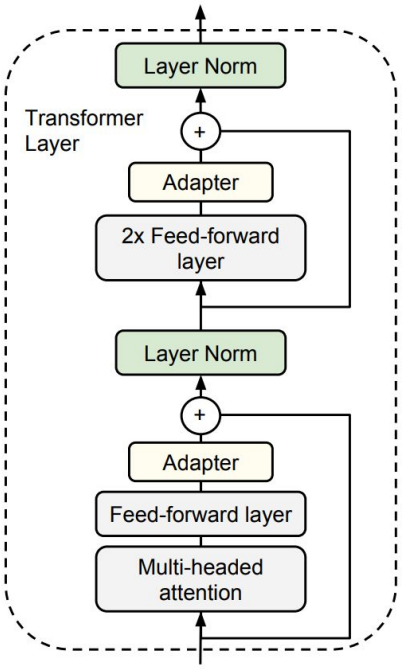
- Only learns a set of ‘prompt’ or ‘token’ for each task

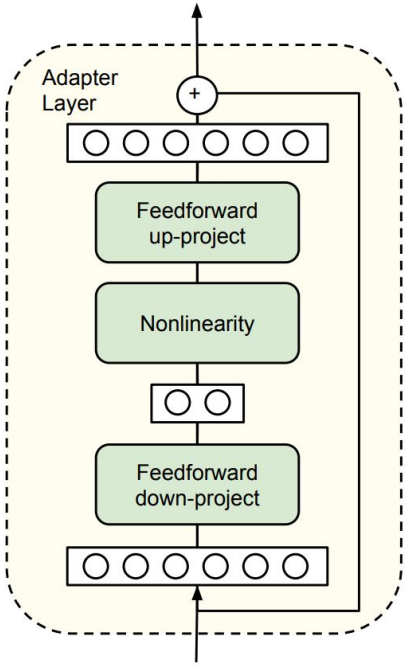


Adapter

- Insert MLP at Feed-forward layers



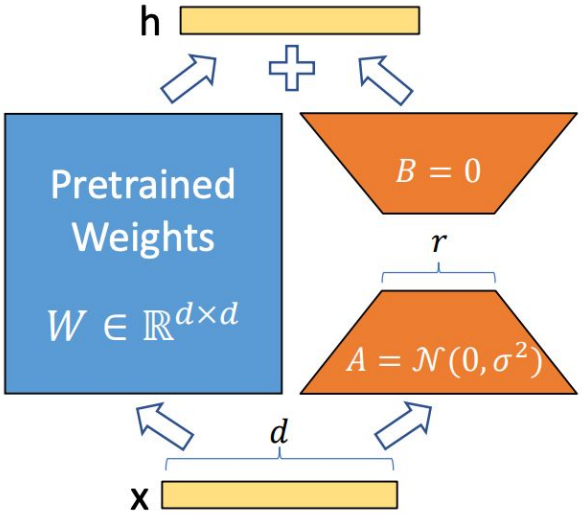




LoRA

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

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

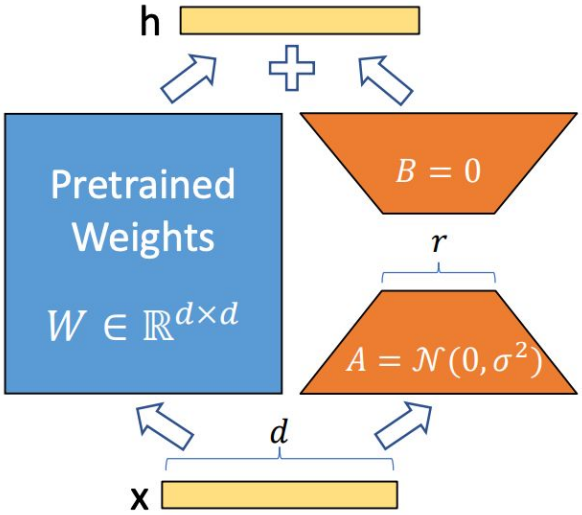


- By default, only W_Q and W_V have the AB pairs
  - Just changing which part to focus on works!

LoRA

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

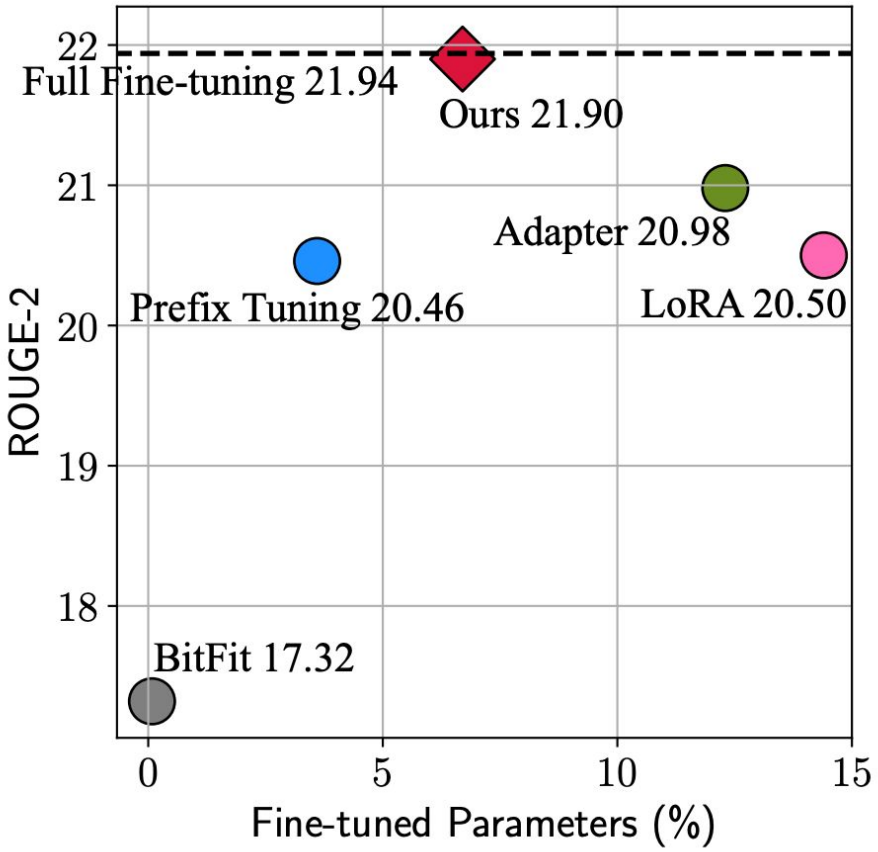
$$h = W_0 x + \Delta W x = W_0 x + BAx$$



- By default, only W_Q and W_V have the AB pairs
  - Just changing which part to focus on works!
- For MLP when we want to add more knowledge

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
- Quantizing Transformers

Quantization

- Human neurons can represent roughly 24 states
- Transformers are in 32 bits, only needs ~4.5
- Basic idea: Weights are floats in a range, represent range with fewer bits
- GPTQ – cuts down to <=4 bits weight, <5% perplexity difference for 1B+ models

We learned…

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