Deep Learning
Transformer and Newer Architectures

Shrey Jain, Dareen Alharthi, Hao Chen

Fall 2025
Attendance:@1205

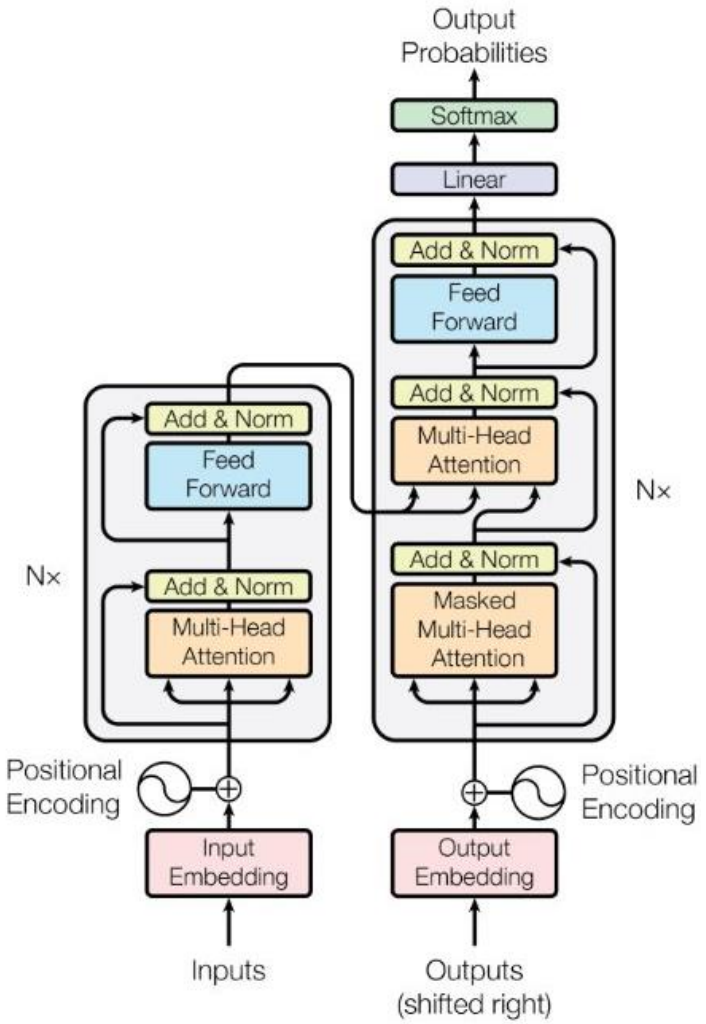
Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Scaling Laws
- Parameter Efficient Tuning
- Quantizing Transformers
- Interpreting Transformer – attention, logit lens

Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Scaling Laws
- Parameter Efficient Tuning
- Quantizing Transformers
- Interpreting Transformer – attention, logit lens

Transformer Architecture

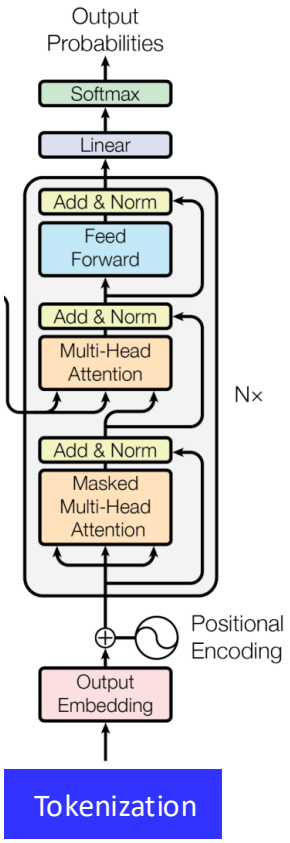


Vaswani, A. "Attention is all you need." *Advances in Neural Information Processing Systems* (2017).

Transformer Architecture

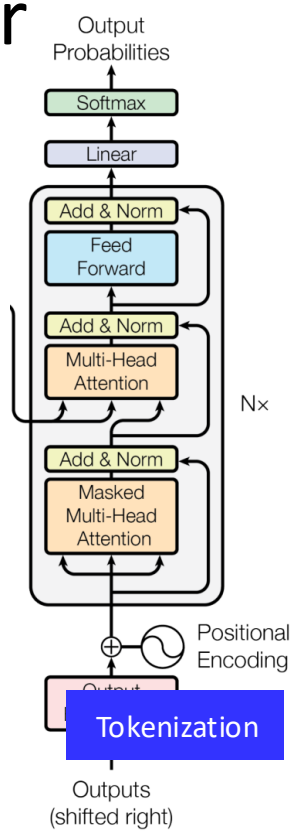
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer

Transformer Block



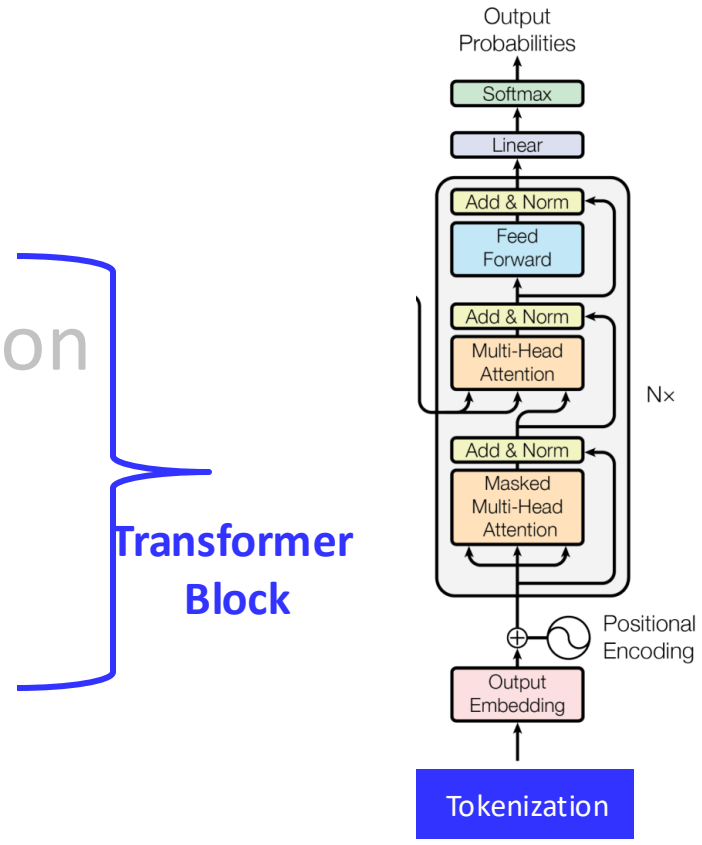
Transformer Architecture

- Transformer is a giant number cruncher
- But it takes in text, outputs text
- How do you go from text to numbers and back?
  - Old solution: one-hot vectors
  - But one-hot of what?



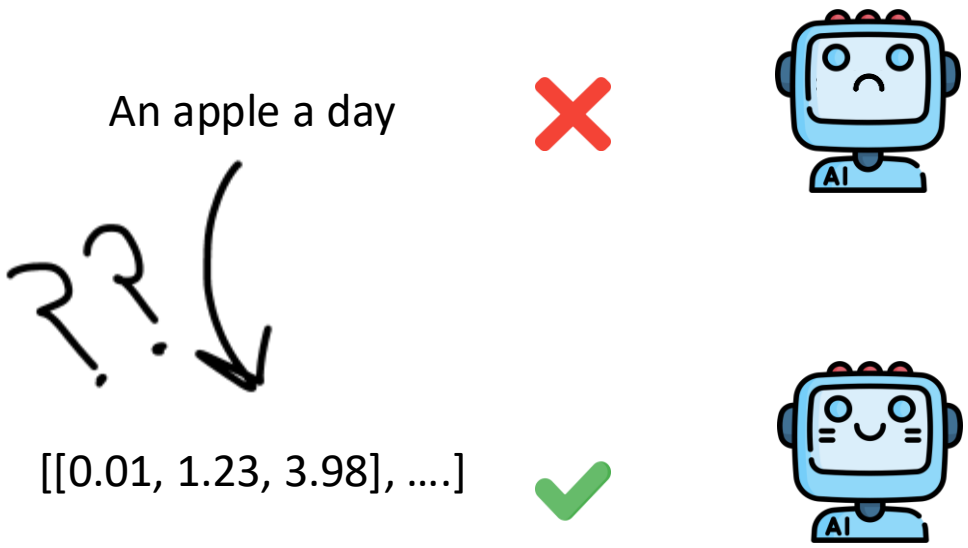
Transformer Architecture

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



Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?



Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Words are basic semantic units, so, one-hot of a dictionary of all words?

[[0,0,…,1,0…,0],
[0,0,…,0,1…,0],
[0,0,…,1,0…,0],
[0,…,1,0,0…,0]]

An apple a day → An apple a day

Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Words are basic semantic units, so, one-hot of a dictionary of all words?
  - What about new words? <UNK> helps no one!
  - 'Hello', 'hello' and 'HeLlO' almost same yet not same, aword for each permutation?

Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Words are basic semantic units, so, one-hot of a dictionary of all words?
  - Advantage would be never producing an illegible word ✔
  - But it's too inflexible for input:
    - New words can't be handled ❌
    - Typos can't be handled either ❌
    - Permutation nightmare ❌

Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Index for each character? Representing Input would be simple!
  - Represent any input unambiguously
  - Tiny vocabulary!

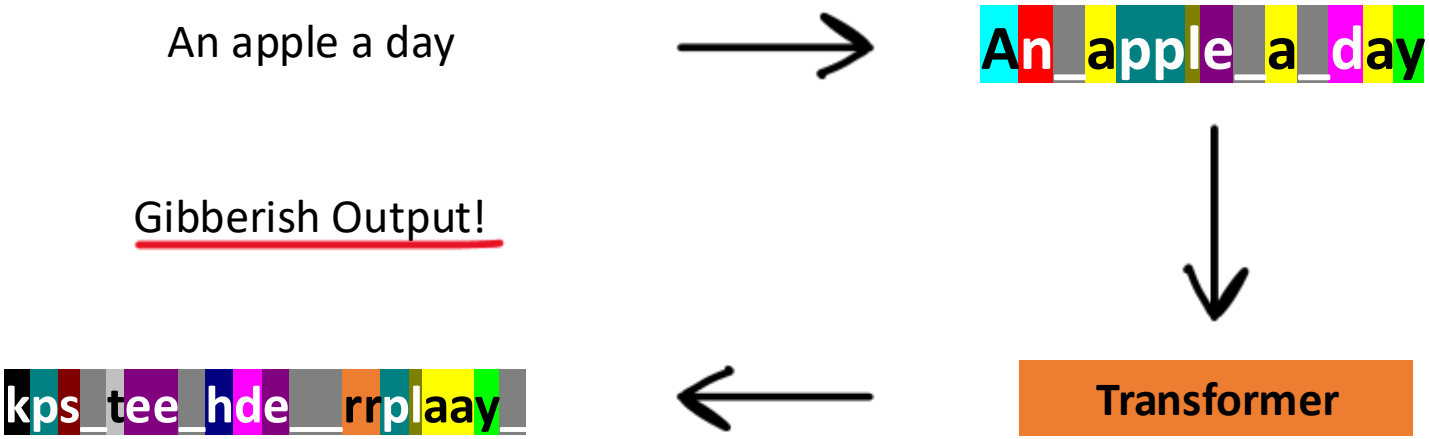
Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Index for each character? Representing Input would be simple!

An apple a day  An_apple_a_day

Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Index for each character? Representing Input would be simple!



Tokenization

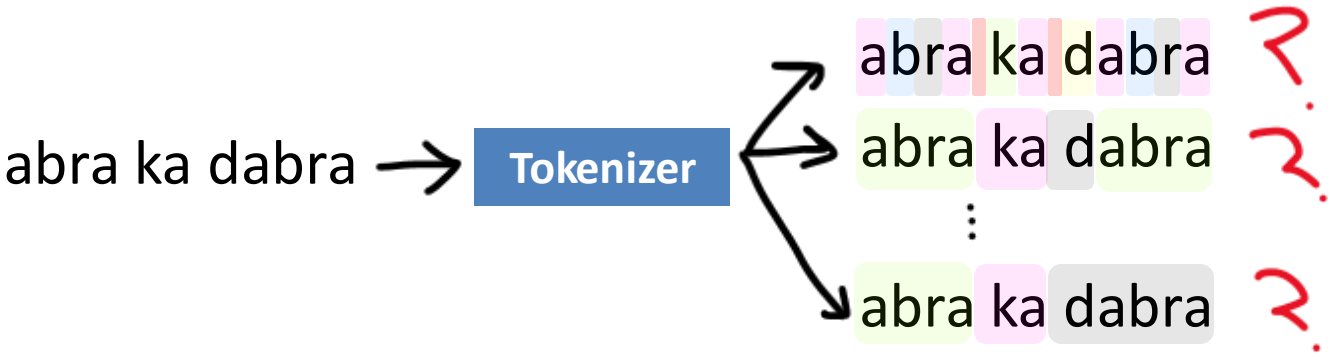
- Computers don't understand text, they only get numbers
- How to represent text as numbers?
  - Words are semantically perfect but inflexible
  - Characters too flexible and output is junk

Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Enter: Sub-word tokenization, a middle-ground

Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Enter: Sub-word tokenization, a middle-ground
  - Get meaningfulness of a word for the outputs
  - Get flexibility of smaller units for inputs
  - Units can even include punctuations!



Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Enter: Sub-word tokenization, a middle-ground
  - Get meaningfulness of a word for the outputs
  - Get flexibility of smaller units for inputs
  - Units can even include punctuations!
- Tokenizer, an external module, is learned to do this deterministically

Tokenization

- Computers don't understand text, they only get numbers
- How to represent text as numbers?
- Enter: Sub-word tokenization, a middle-ground
  - Get meaningfulness of a word for the outputs
  - Get flexibility of smaller units for inputs
  - Units can even include punctuations!
- Tokenizer, an external module, is learned to do this deterministically
  - Methods: **Byte-pair encoding**, WordPiece

Tokenization - BPE

- Iteratively, replace most common token pair with a single token

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

Tokenize: aaabdaaabac

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

Tokenize: aaabdaaabac

Counts: {'aa': 4, 'ab': 2, 'bd': 1, 'da': 1, 'ba': 1, 'ac': 1}

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

Tokenize: aaabdaaabac

Counts: {'aa': 4, 'ab': 2, 'bd': 1, 'da': 1, 'ba': 1, 'ac': 1}

Replace 'aa' with Z: aaabdaaabac -> ZabdZabac

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

Tokenize: ZabdZabac

Counts: {'ab': 2, 'Za': 2, 'bd': 1, 'dZ': 1, 'ba': 1, 'ac': 1}

Replace 'ab' with Y: ZabdZabac -> ZYdZYac

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

So we'd get:

aaabdaaabac -> ZabdZabac -> ZYdZYac -> XdXac

Where Z = aa, Y = ab, X = ZY = aaab

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

So we'd get:

aaabdaaabac -> ZabdZabac -> ZYdZYac -> XdXac

Where Z = aa, Y = ab, X = ZY = aaab

This compressed text to < ½ the original length

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

If we started with just lowercase vocab ('a': 1, 'b': 2...):

aaabdaaabac = [1, 1, 1, 2, 4, 1, 1, 1, 2, 1, 3]

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

If we started with just lowercase vocab ('a': 1, 'b': 2...):

aaabdaaabac = [1, 1, 1, 2, 4, 1, 1, 1, 2, 1, 3]

We connect (1, 1) or ('a', 'a') -> 27 or 'aa'

Then (1, 2) -> 28 and finally (27, 28)-> 29

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

If we started with just lowercase vocab ('a': 1, 'b': 2...):

aaabdaaabac = [1, 1, 1, 2, 4, 1, 1, 1, 2, 1, 3]

We get XdXac = [29, 4, 29, 1, 3]

Much shorter and simpler!

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Let's take an example:

If we started with just lowercase vocab ('a': 1, 'b': 2...):

aaabdaaabac = [1, 1, 1, 2, 4, 1, 1, 1, 2, 1, 3]

We get XdXac = [29, 4, 29, 1, 3]

Much shorter and simpler!

https://en.wikipedia.org/wiki/Byte-pair_encoding

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Worked well for the small example,
  how to handle for our multilingual complex world?

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Worked well for the small example,
  how to handle for our multilingual complex world?
- Every known digital character part of Unicode (160K)
- Represent each char as a combination of bytes
  - Byte is 8-bits or 256 total possible values
  - e.g. G = [71], ü = [195, 188]

https://en.wikipedia.org/wiki/Unicode

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Worked well for the small example,
  how to handle for our multilingual complex world?
- Every known digital character part of Unicode (160K)
- Can represent each as combination of bytes
  - Byte is 8-bits or 256 total possible values
- Represent all data as bytes and then do BPE!

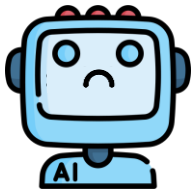
https://en.wikipedia.org/wiki/Unicode

Only BPE isn't enough - Decoding

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz	0-127		
U+0080	U+07FF	110xxxyy	10yyzzzz	194-223, 128-191	
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	224-239, 128-191, 128-191
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz
		240-247,	128-191,	128-191,	128-191

Decoding '\x80' or '128' fails



Only BPE isn't enough - Decoding

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz	0-127		
U+0080	U+07FF	110xxxyy	10yyzzzz	194-223, 128-191	
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	224-239, 128-191, 128-191
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz
		240-247,	128-191,	128-191,	128-191

Decoding '\x80' or '128' fails

- handle gracefully: *errors='replace' to use* '  '

Only BPE isn't enough - Decoding

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz	0-127		
U+0080	U+07FF	110xxxyy	10yyzzzz	194-223, 128-191	
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	224-239, 128-191, 128-191
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz
		240-247,	128-191,	128-191,	128-191

Decoding '\x80' or '128' fails

- handle gracefully: *errors='replace' to use '*'
- learn correct usage: model learns correct usage

Only BPE isn't enough

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0yyyzzzz	0-127		
U+0080	U+07FF	110xxxyy	10yyzzzz	194-223, 128-191	
U+0800	U+FFFF	1110wwww	10xxxxyy	10yyzzzz	224-239, 128-191, 128-191
U+010000	U+10FFFF	11110uvv	10vvwwww	10xxxxyy	10yyzzzz
		240-247,	128-191,	128-191,	128-191

Decoding '\x80' or '128' fails

- handle gracefully: *errors='replace' to use '*'
- learn correct usage: model learns correct usage
- Constraints during decoding:

Only BPE isn't enough

Decoding '\x80' or '128' fails

- handle gracefully: *errors='replace' to use '*'
- learn correct usage: model learns correct usage
- Constraints during decoding: Through HuggingFace's transformers.LogitsProcessor classes

Only BPE isn't enough

Decoding '\x80' or '128' fails

- handle gracefully: *errors='replace' to use '*'
- learn correct usage: model learns correct usage
- Constraints during decoding: Through HuggingFace's transformers.LogitsProcessor classes

e.g. Decoding a json... output so far: ```{"age": ```
Next tokens: '{': 2.4 | '5': 2.91 | '}': 3.04 | ',': 4.1 | ...

Only BPE isn't enough

Decoding '\x80' or '128' fails

- handle gracefully: *errors='replace' to use '*'
- learn correct usage: model learns correct usage
- Constraints during decoding: Through HuggingFace's transformers.LogitsProcessor classes

e.g. Decoding a json... output so far: ```{"age": ```

Next tokens: '{': 2.4 | '5': 2.91 | '}': 3.04 | ',': 4.1 | ...

Invalid format -> set logits to –inf before softmax

Only BPE isn't enough

Decoding '\x80' or '128' fails

- handle gracefully: *errors='replace' to use '*'
- learn correct usage: model learns correct usage
- Constraints during decoding: Through HuggingFace's transformers.LogitsProcessor classes

SGLang LLM infer. framework leverages constrained decoding to get better format adherence while decoding e.g. JSON

Tokenization at a Large Scale

- How to avoid unintelligible strings from being decoded?
- Deliberate design choices pre-tokenization
  - Directly BPE on all docs?
    'Hello world', 'Hello world.', 'Hello world!' … other variations would be common
  - Do we learn all variants of world? ' world', ' world.', ' world!'
  - Unnecessarily redundant tokens

Tokenization at a Large Scale

- Case Study: GPT-2 [src]
- Regex Pattern:

```
's|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+| ?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+
```

```
"Hello world I've learned to loveee!     tokenization, I'M NOW THE #1Fan "
```

Tokenization at a Large Scale

- Case Study: GPT-2 [src]
- Regex Pattern:

```
's|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+| ?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+
```

```
"Hello world I've learned to loveee!  tokenization, I'M NOW THE #1Fan "
```

```
['Hello', '_world', '_I', "'ve", '_learned', '_to', '_loveee',
'!', '_', '_tokeni', 'zation', ',', '_I', "'", 'M', '_NOW',
'_THE', '_#', '1', 'Fan', '_']
```

Tokenization at a Large Scale

- Case Study: GPT-2 [src]
- Regex Pattern:

```
's|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+| ?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+
```

```
"Hello world I've learned to loveee!  tokenization, I'M NOW THE #1Fan "
```

```
['Hello', '_world', '_I', "'ve", '_learned', '_to', '_loveee',
'!', '_', '_tokeni', 'zation', ',', '_I', "'", 'M', '_NOW',
'_THE', '_#', '1', 'Fan', '_']
```

Didn't set regex to ignore case
A bug in tokenization that was too costly to fix since they had already pre-trained the model!

Tokenization at a Large Scale

- Case Study: GPT-2 [src]
- Regex Pattern:

```
's|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+| ?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+
```

```
"Hello world I've learned to loveee!  tokenization, I'M NOW THE #1Fan "
```

```
['Hello', '_world', '_I', "'ve", '_learned', '_to', '_loveee',
'!', '_', '_tokeni', 'zation', ',', '_I', "'", 'M', '_NOW',
'_THE', '_#', '1', 'Fan', '_']
```

- Convert all these chunks to bytes -> Do BPE till vocab size achieved

Tokenization at a Large Scale

- Case Study: GPT-2 [src]
- Regex Pattern:

```
's|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+| ?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+
```

```
"Hello world I've learned to loveee!  tokenization, I'M NOW THE #1Fan "
```

```
['Hello', '_world', '_I', "'ve", '_learned', '_to', '_loveee',
'!', '_', '_tokeni', 'zation', ',', '_I', "'", 'M', '_NOW',
'_THE', '_#', '1', 'Fan', '_']
```

- Wouldn't we just learn words this way? No, it'll still be sub-words:
  world is common, 1 token; tokenization is uncommon so 'token' + 'ization'

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Worked well for the small example, but we live in a multilingual complex world -> how to handle for that?
- Represent all data as bytes and then do BPE!
- Tokenization can have a huge impact!

Tokenization – BPE Issues

- GPT-2 was great at natural English language, not at other languages or even code – why?

Tokenization – BPE Issues

- GPT-2 was great at natural English language, not at other languages or even code but GPT 4 is – why?
- Each space was one token, overwhelming context window

Token count
32

GPT 2

```
for i in range(10):
    for j in range (10):
        print("Hello!")
```

Token count
20

GPT 4

```
for i in range(10):
    for j in range(10):
        print("Hello!")
```

https://tiktokenizer.vercel.app/?model=gpt2

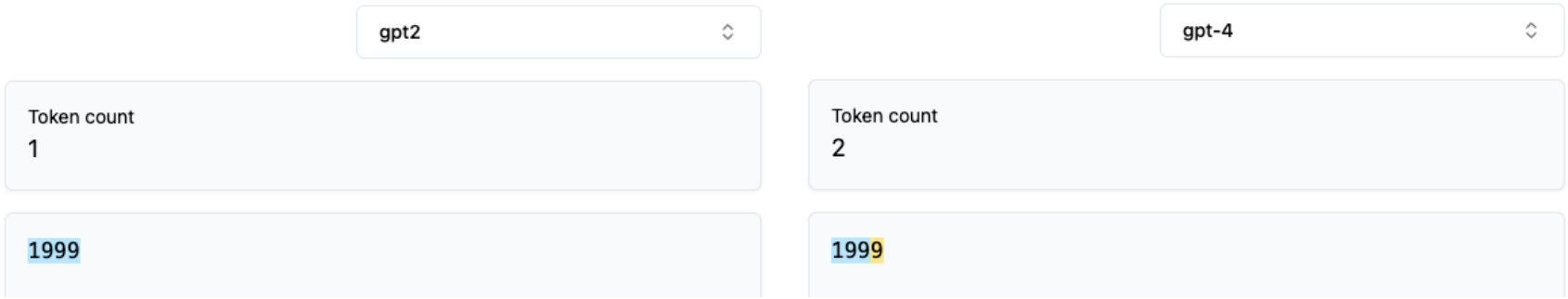
Tokenization – BPE Issues

- Mandarin had >2x tokens; recall issue with char-level tokenization?

https://tiktokenizer.vercel.app/?model=gpt2

Tokenization – BPE Issues

- GPT 2 was bad at math - it didn't limit length on numbers, GPT 4 limited to 3 digits
- Struggled to learn multiplication mechanics this way



https://tiktokenizer.vercel.app/?model=gpt2

Tokenization – BPE Issues

- What if a token appears a lot in tokenization data but not in training data?
  - SolidGoldMagikarp -> LLM would go berzerk!

- LLM can't spell words
  - The infamous 'how many r's in strawberry'

https://tiktokenizer.vercel.app/?model=gpt2

Tokenization - BPE

- Iteratively, replace most common token pair with a single token
- Worked well for the small example, but we live in a multilingual complex world -> how to handle for that?
- Represent all data as bytes and then do BPE!
  - All data = all text on the internet!
- GPT-4 handled a lot of edge cases, starting with 258 tokens (256+2 special) and did 100K merges to have a 100,258 vocab - fixed for problems with GPT-2

Inspired by Andrej Karpathy's talk: https://www.youtube.com/watch?v=zduSFxRajkE ; watch this to dive deeper!

Poll @1206

What is the primary harm of allowing a BPE algorithm to merge tokens across pre-tokenization boundaries (e.g., merging the 'o' from 'Hello' with the ' w' from ' world')?

- It would violate UTF-8 encoding rules and cause the model to crash.
- It would fill the vocabulary with semantically meaningless but frequent 'junk' tokens (like 'o w' or 'd t') and lead to inefficient, broken tokenization.
- It is not harmful and is a standard part of all modern tokenizers.
- It would make the tokenizer too slow to run during inference.

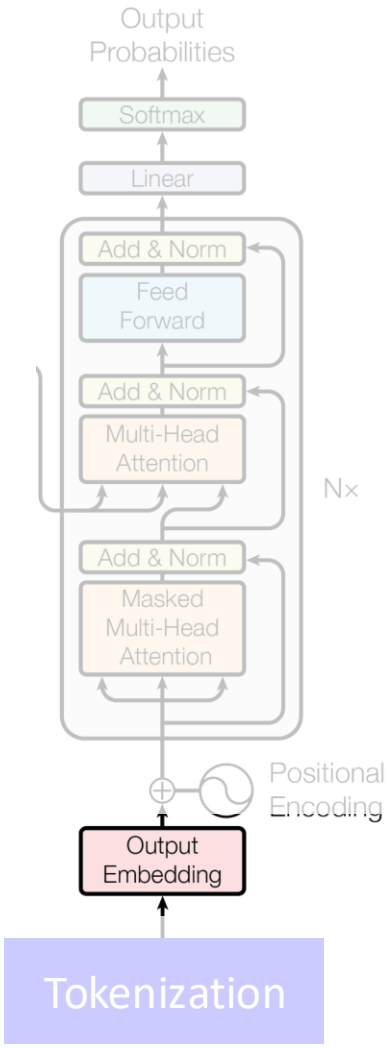
Poll @1206

What is the primary harm of allowing a BPE algorithm to merge tokens across pre-tokenization boundaries (e.g., merging the 'o' from 'Hello' with the ' w' from ' world')?

- It would violate UTF-8 encoding rules and cause the model to crash.
- It would fill the vocabulary with semantically meaningless but frequent 'junk' tokens (like 'o w' or 'd t') and lead to inefficient, broken tokenization.
- It is not harmful and is a standard part of all modern tokenizers.
- It would make the tokenizer too slow to run during inference.

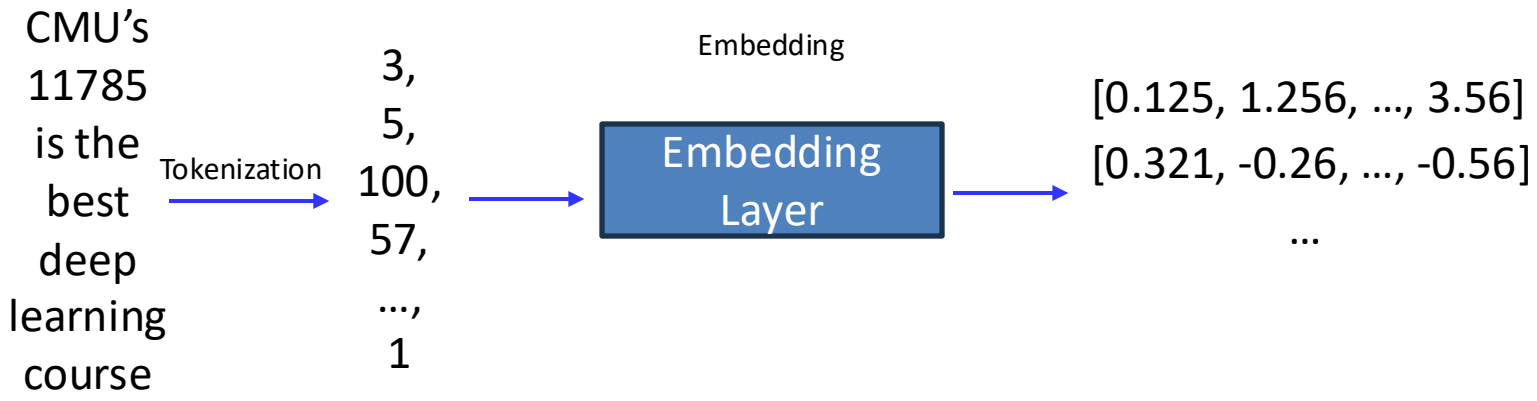
Transformer Architecture

- 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 is a Linear Layer

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

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

$$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

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

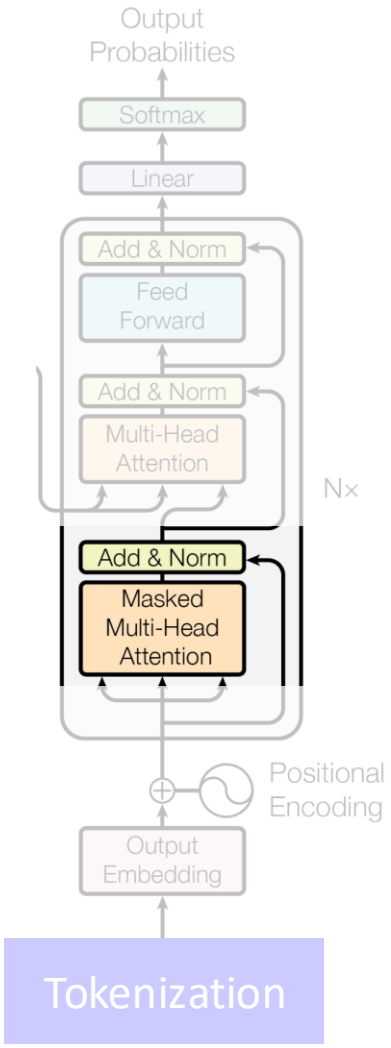
$$\begin{bmatrix} 0.235 & -1.256 & 3.513 & \dots & -0.187 \\ 1.291 & -2.012 & 0.624 & \dots & -1.291 \\ 0.535 & 0.012 & -0.024 & \dots & 2.345 \\ \dots & \dots & \dots & \dots & \dots \\ 0.131 & 2.102 & 0.935 & \dots & -0.125 \end{bmatrix}$$

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

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

Transformer Architecture

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

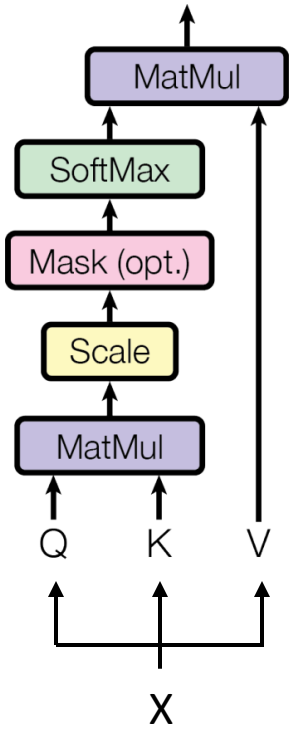


Self-Attention

- Attention Operation

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

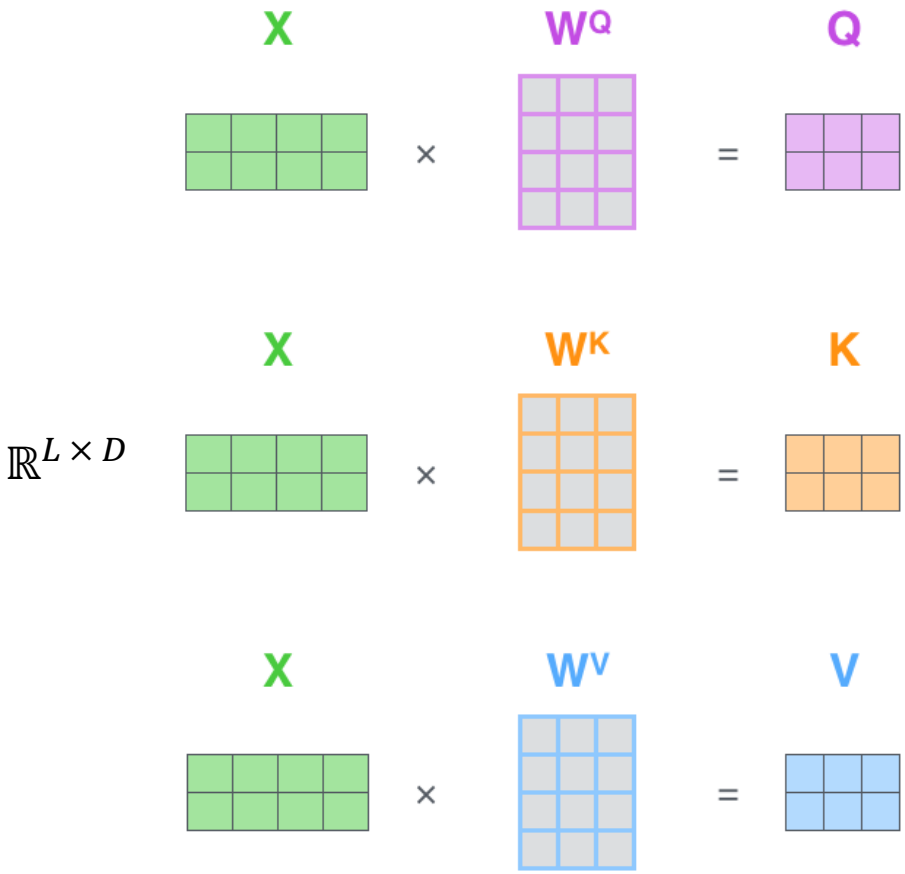
Scaled Dot-Product Attention

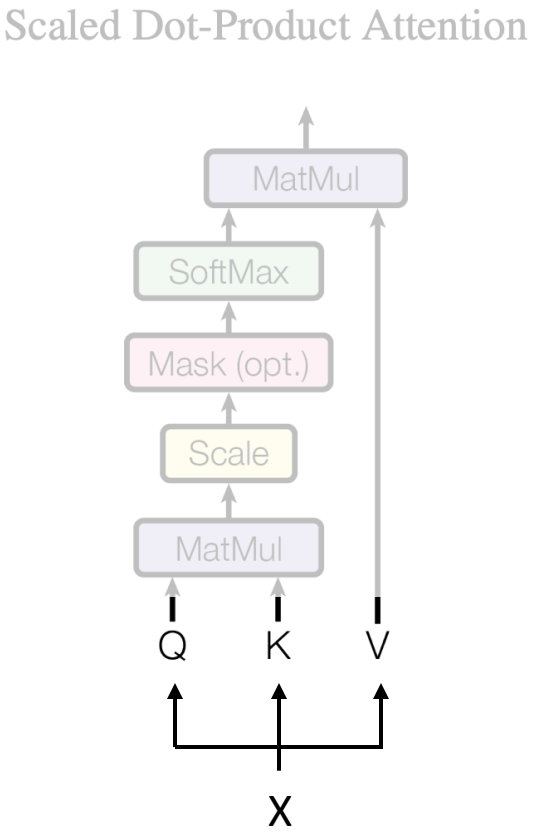


- Query-Key-Value
  - Linear affine from input X itself
- Weighted-sum of V based on similarity/correlation between Q and K
  - Each token's weights sum to one

Self-Attention

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





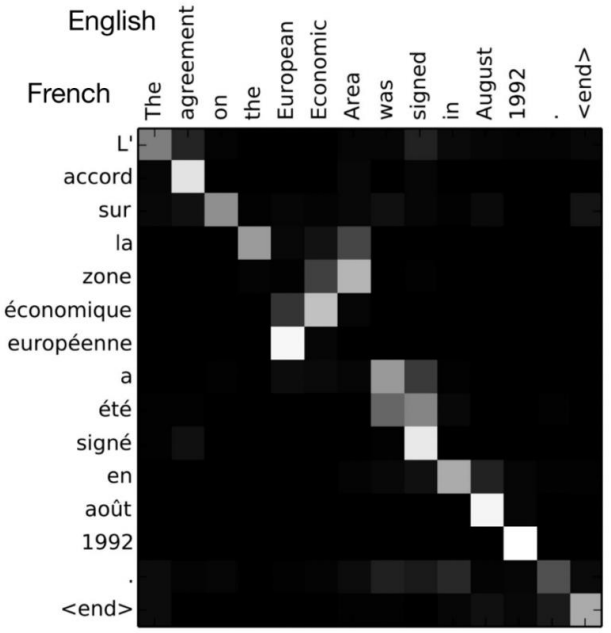
Self-Attention

- Attention weights

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

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

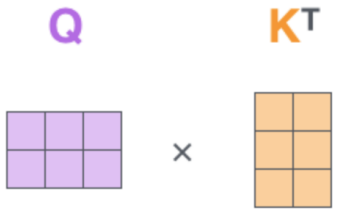
$$\mathbb{R}^{L \times L}$$



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

Self-Attention

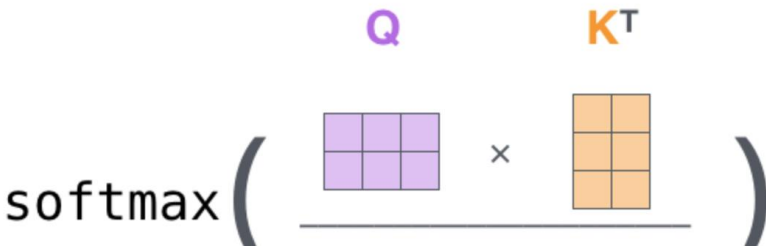
- Why is the dot product not enough?



- Dot product could be arbitrary numbers
-

Self-Attention

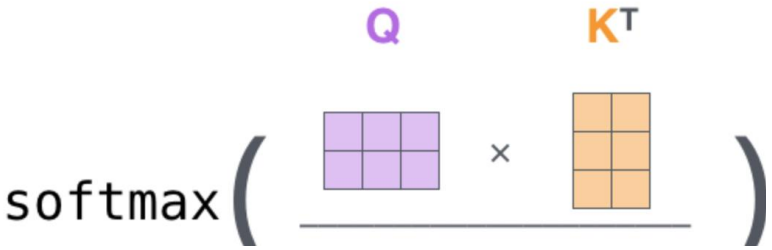
- Why is the dot product not enough?



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

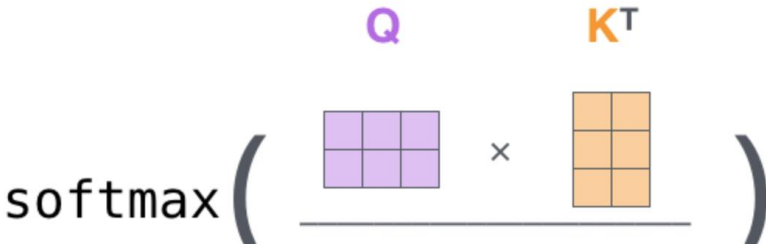
Self-Attention

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



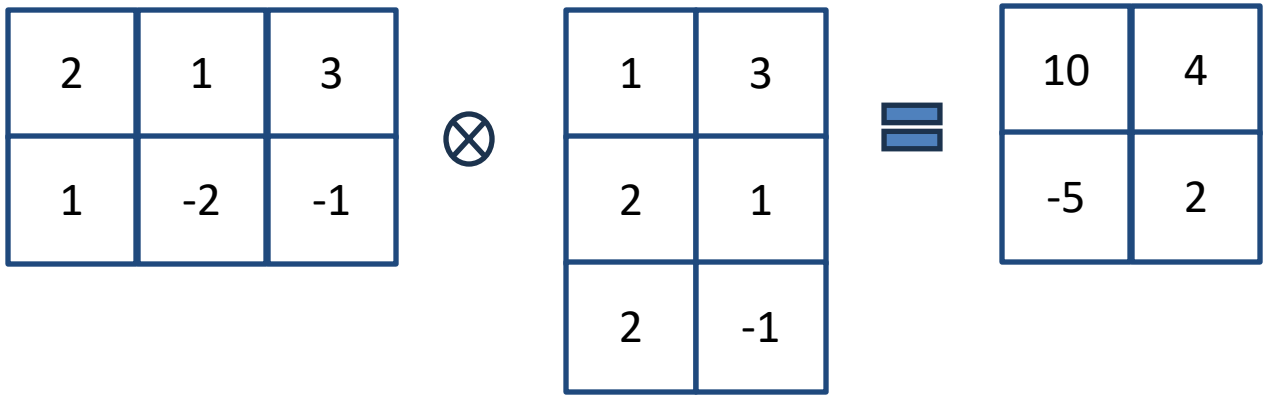
Self-Attention

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

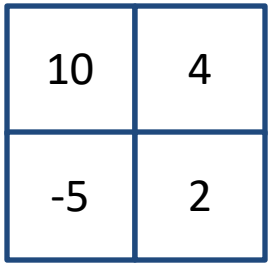


- Softmax has a vanishing/exploding problem!

Self-Attention – why d_k



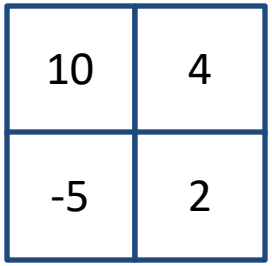
Self-Attention – why d_k

Softmax()

=

0.99	2e-3
9e-4	0.99

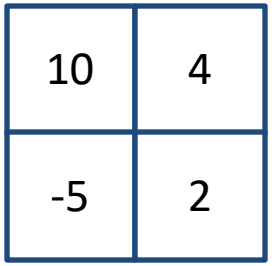
Self-Attention – why d_k

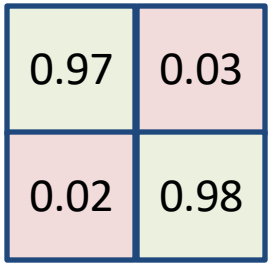
Softmax(

) = 

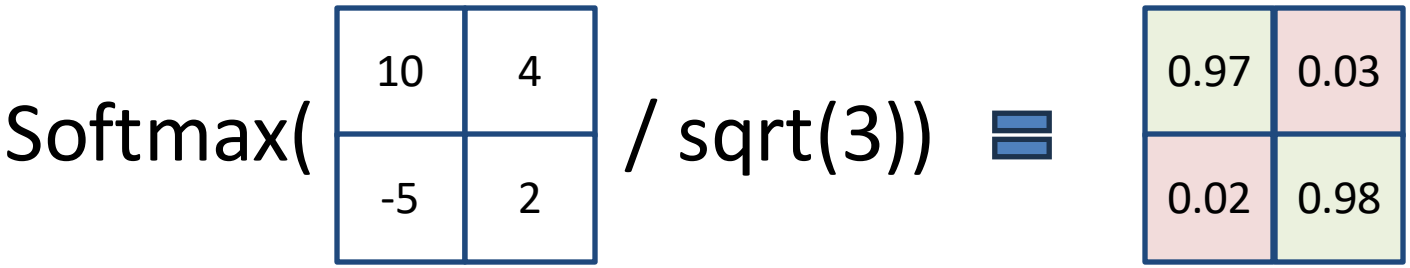
Softmax pushes disparity to extremes
=> vanishing gradients

Self-Attention – why d_k

Softmax(

/ sqrt(3)) = 

Self-Attention – why d_k

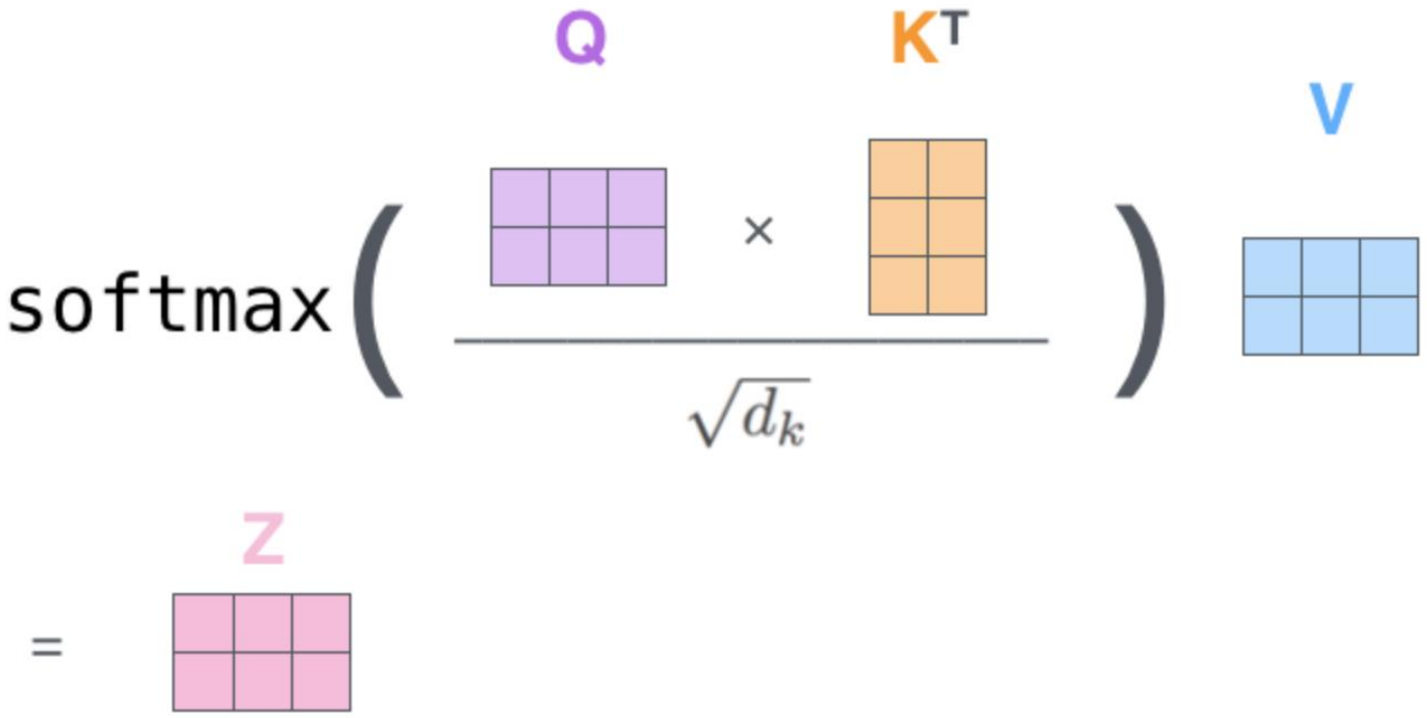


Sqrt(d_k) is the std. dev. of the dot product

Scaling by that reduces the vanishing gradient!

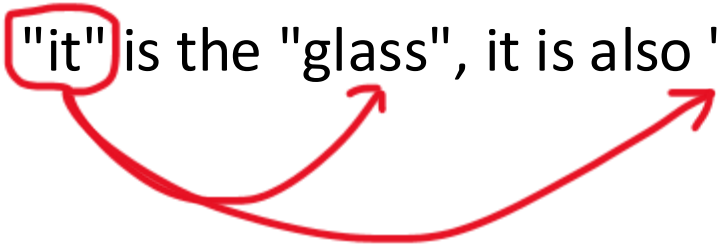
Self-Attention

- Output



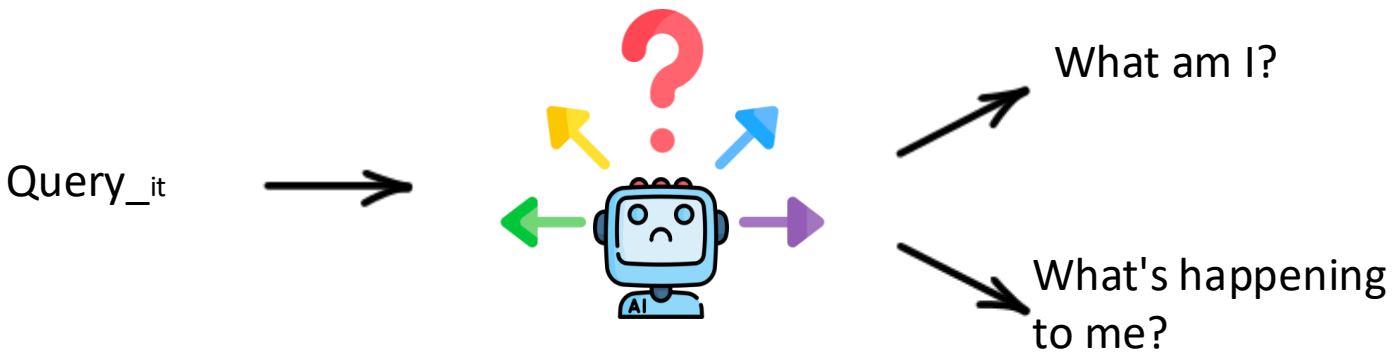
Multi-Head Self-Attention

- Why do we need multiple self-attention operations? Take the sentence:
- "I put the glass on the table, but **it** broke."
- "it" is the "glass", it is also "broke"



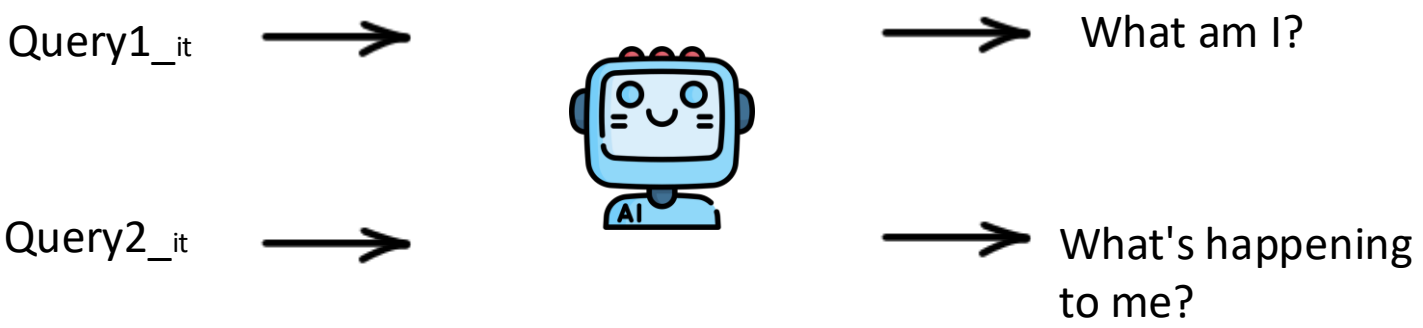
Multi-Head Self-Attention

- Why do we need multiple self-attention operations? Take the sentence:
- "I put the glass on the table, but **it** broke."
- "it" is the "glass", it is also "broke"



Multi-Head Self-Attention

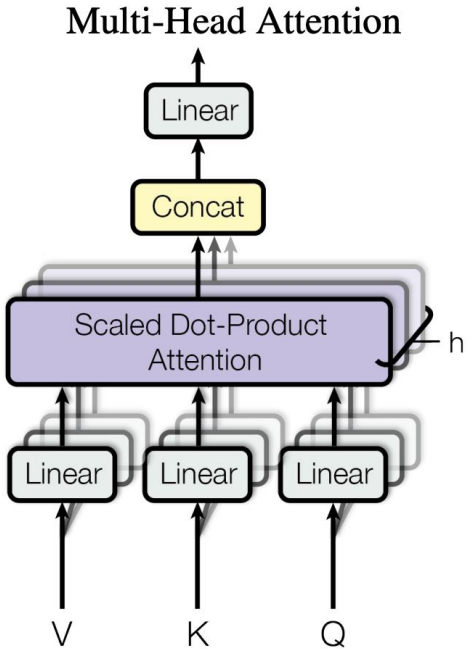
- Why do we need multiple self-attention operations? Take the sentence:
- "I put the glass on the table, but **it** broke."
- "it" is the "glass", it is also "broke"



Multi-Head Self-Attention

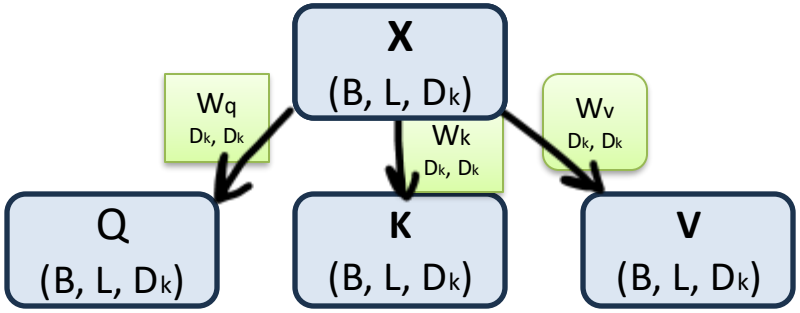
- Multiple self-attention operations over the channel dimension

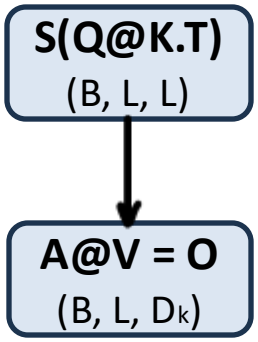
$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_\text{h}\right) W^Q$$
$$\text{where head} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$



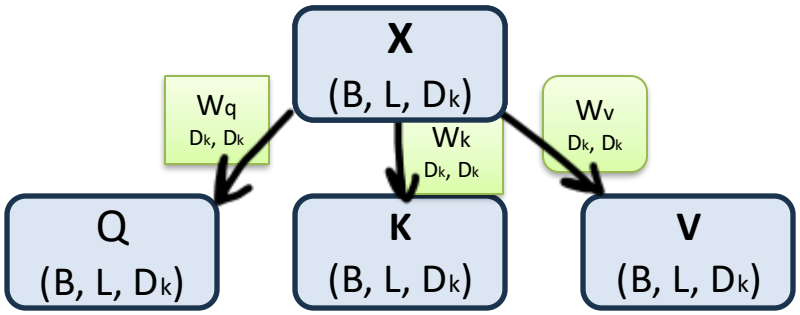
- Advantages:
  - Different attention maps capture different relationships
  - Parallel processing

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





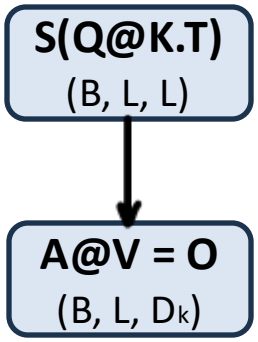
(F.scaled_dot_product_attention) SHA vs MHA (torch.nn.MultiheadAttention)

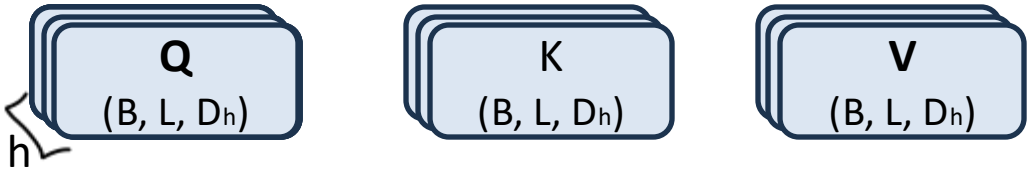


$D_k = h * D_h$

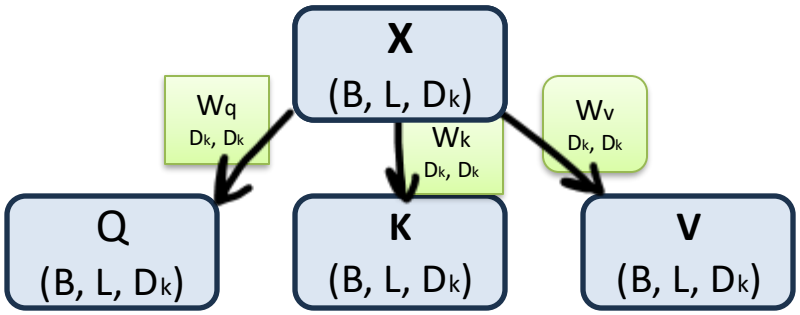
Reshape to (B, L, h, D_h)

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





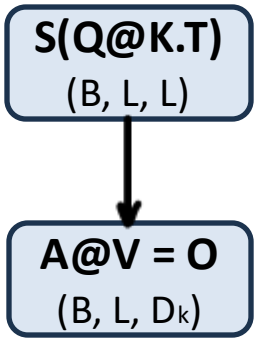
(F.scaled_dot_product_attention) SHA vs MHA (torch.nn.MultiheadAttention)

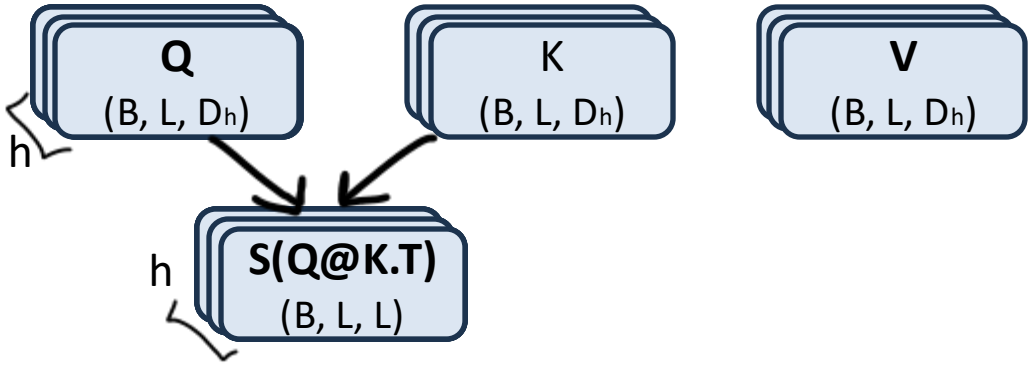


$D_k = h * D_h$

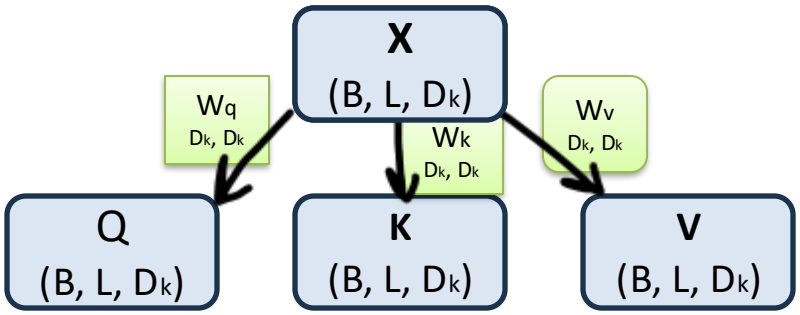
Reshape to (B, L, h, D_h)

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





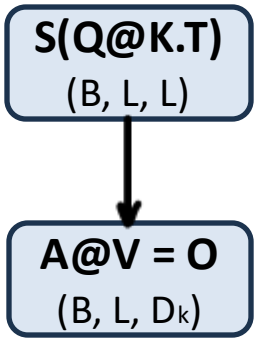
(F.scaled_dot_product_attention) SHA vs MHA (torch.nn.MultiheadAttention)

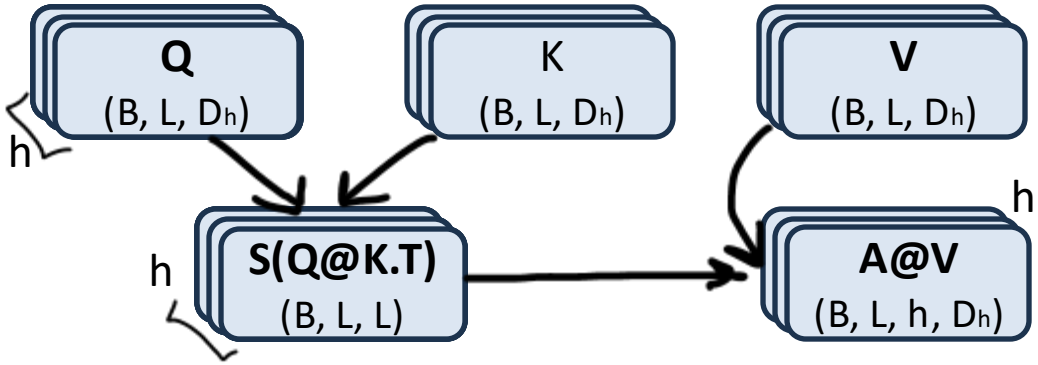


$D_k = h * D_h$

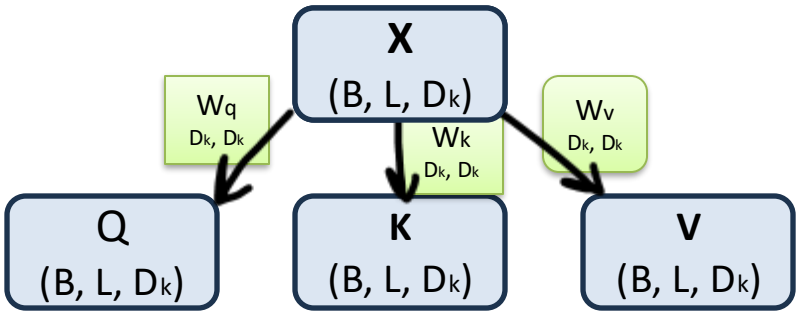
Reshape to (B, L, h, D_h)

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

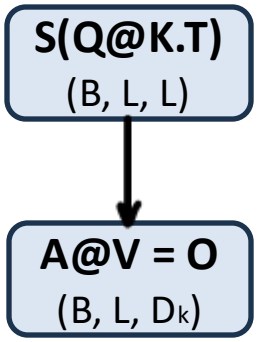


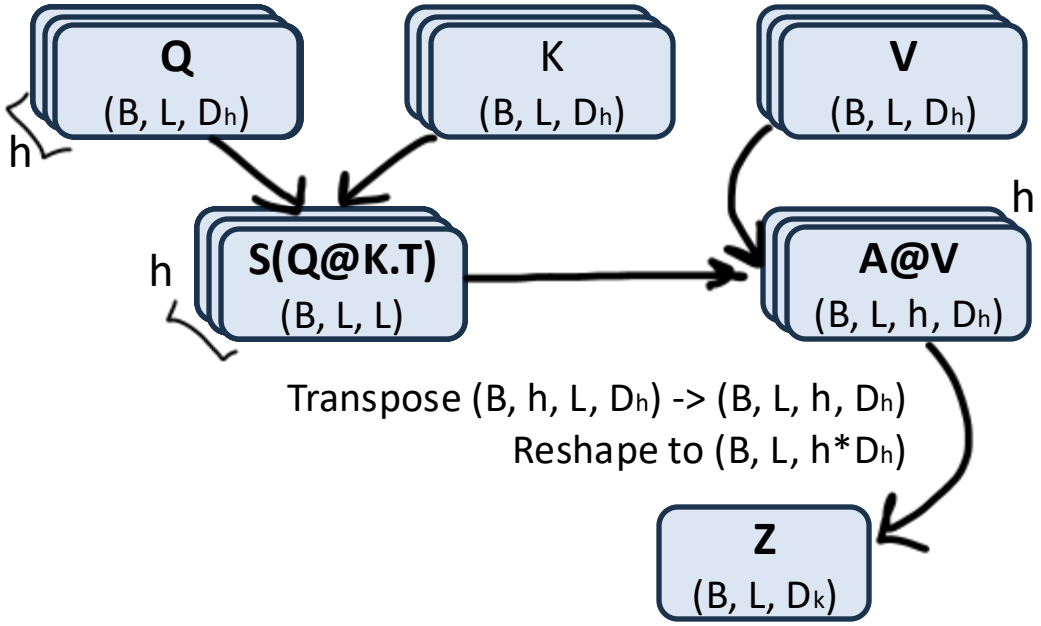


(F.scaled_dot_product_attention) SHA vs MHA (torch.nn.MultiheadAttention)

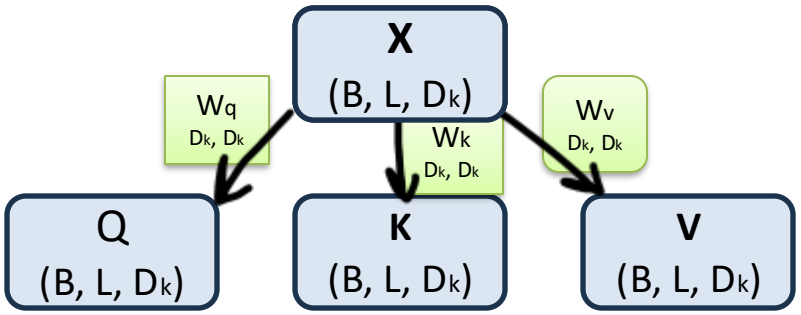


$D_k = h * D_h$
Reshape to (B, L, h, D_h)
Transpose (-3, -2) -> (B, h, L, D_h)

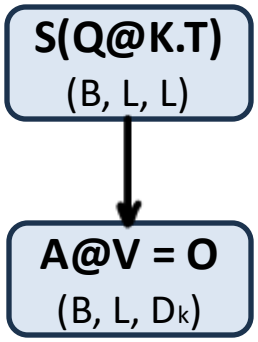


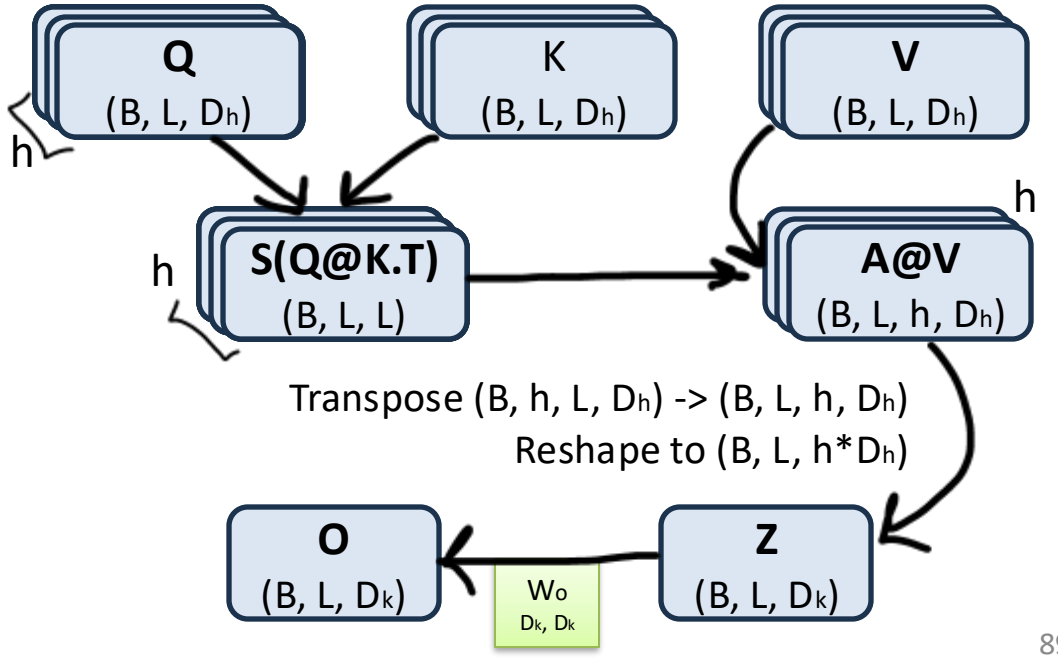


(F.scaled_dot_product_attention) SHA vs MHA (torch.nn.MultiheadAttention)

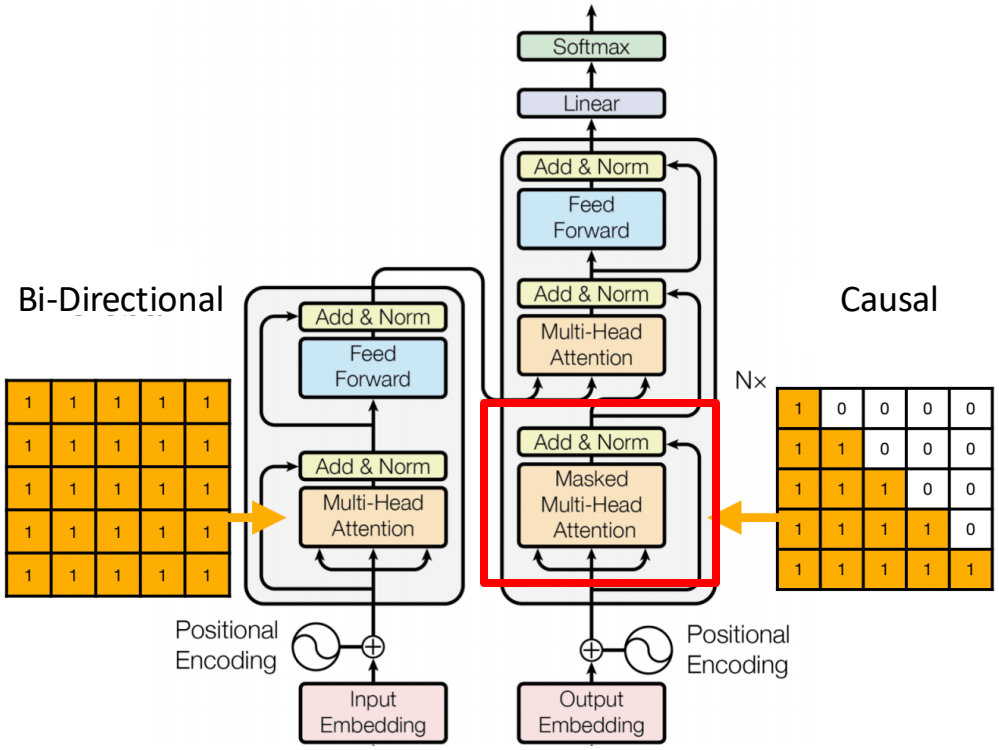


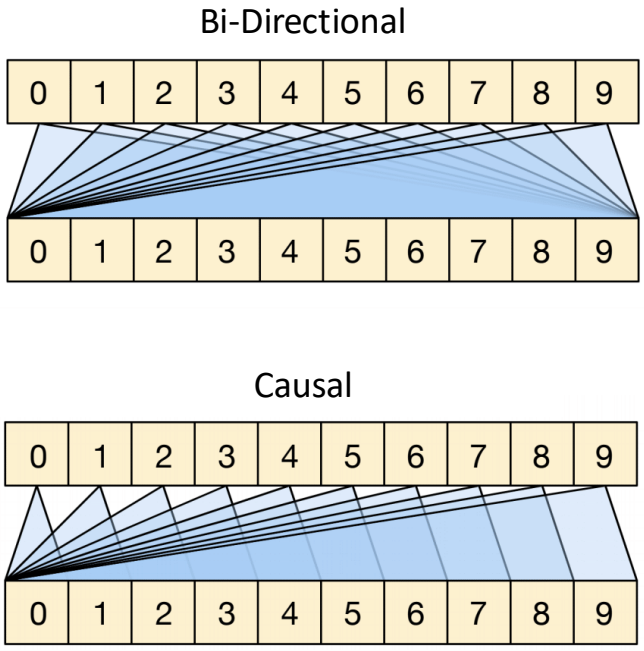
$D_k = h * D_h$
Reshape to (B, L, h, D_h)
Transpose (-3, -2) -> (B, h, L, D_h)



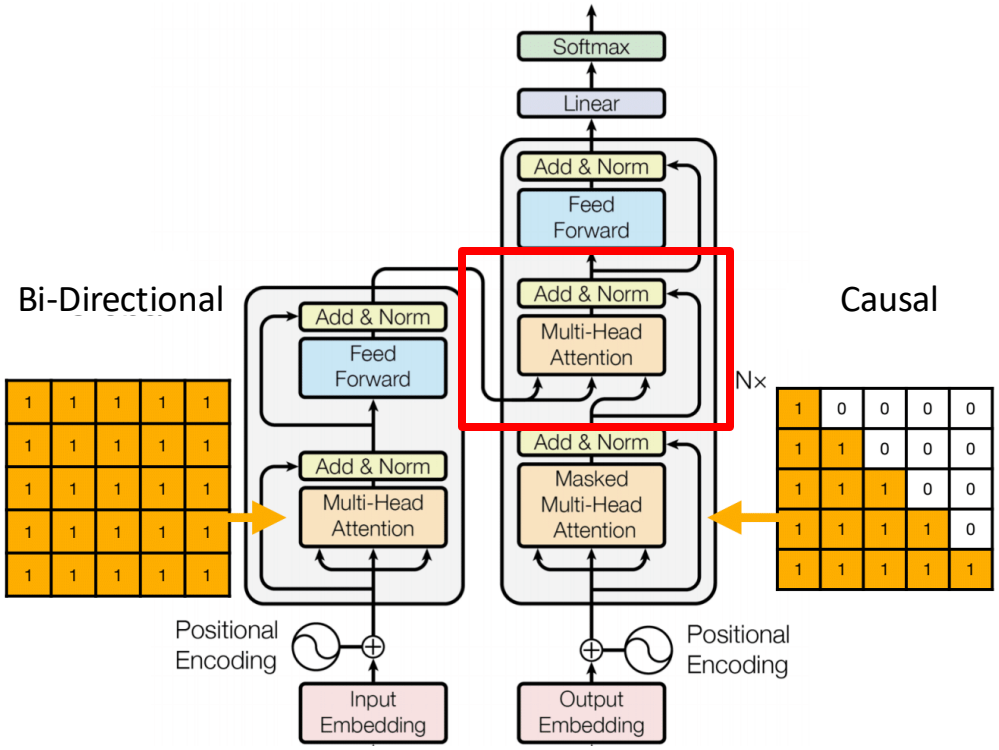


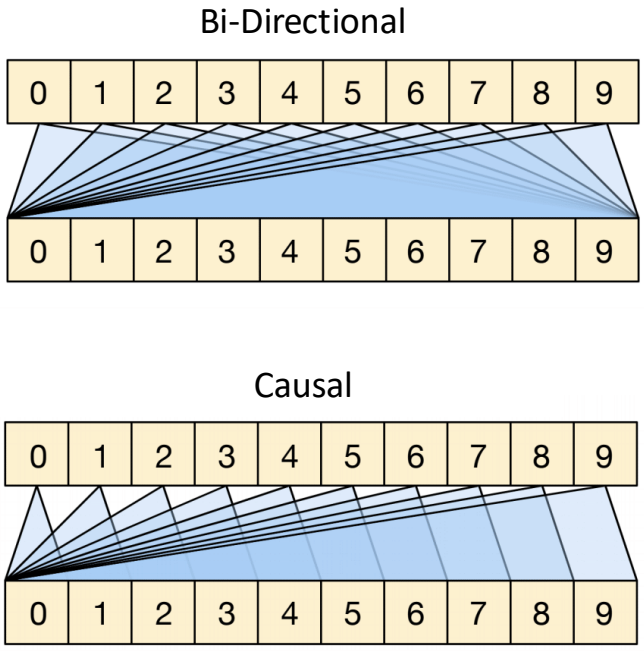
Attention Masking





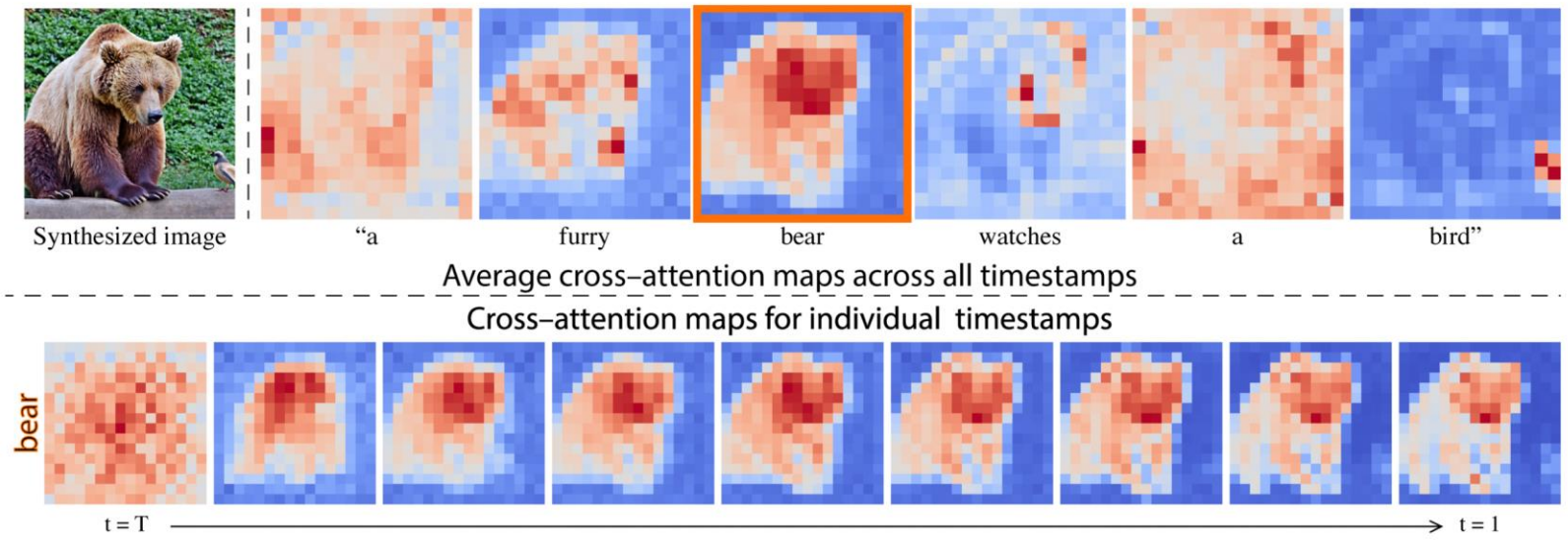
Attention Masking





Cross-Attention

query, "a furry bear watches a bird."

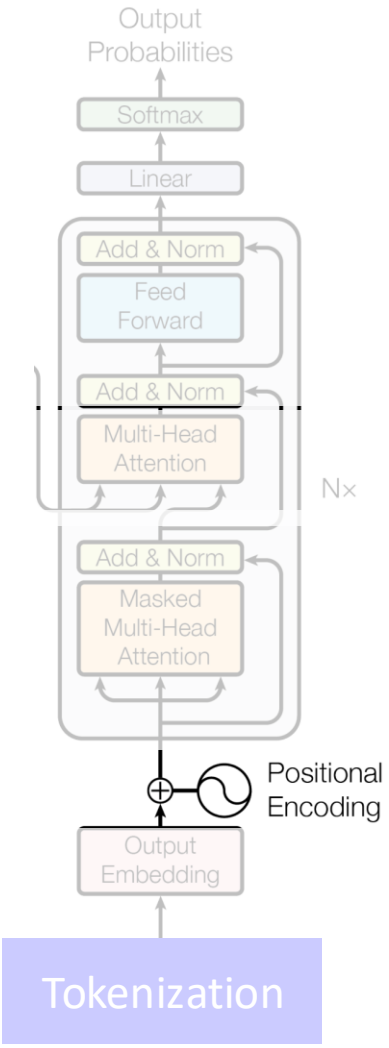


The model iteratively denoise the noise vector based on the given text query to generate an Image

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

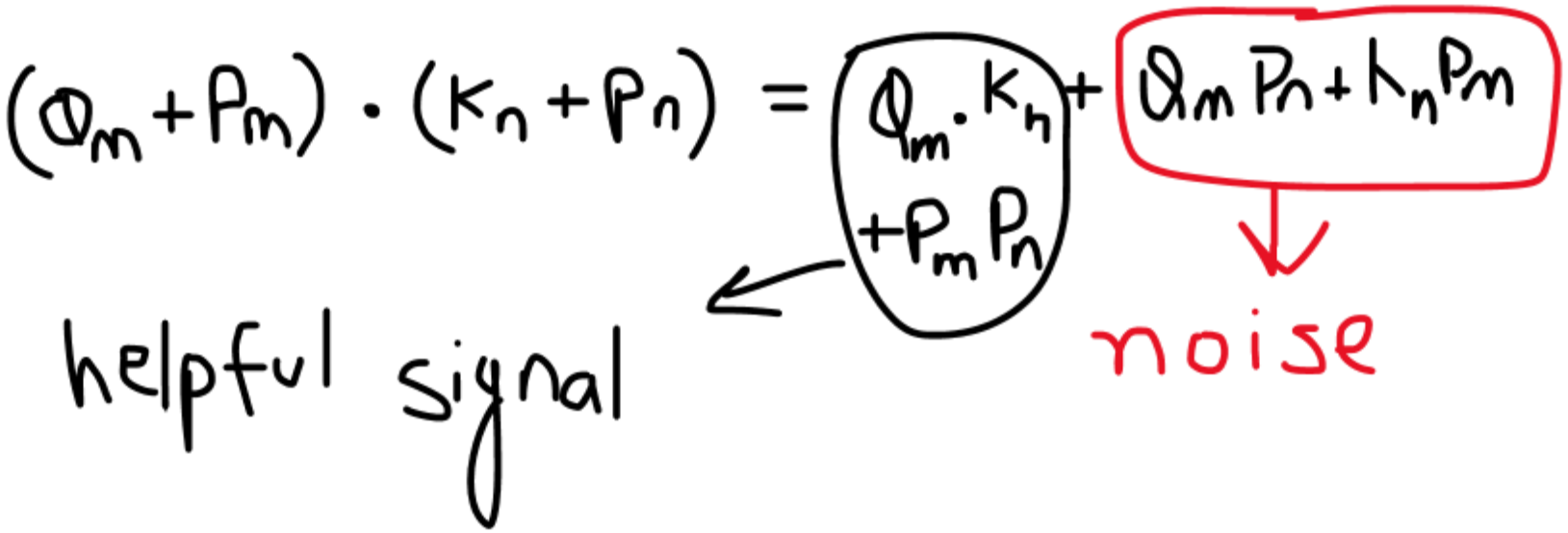


Position Encoding

- Why do we need them?
  - Self-attention is permutation-invariant!
  - "James Bond signed a Bond with the bank"
  - Both 'Bond' will have same embeddings and then equal attention weight -> that can't be right, no?
  - We need something that gives idea of where the token in positioned in the context

Position Encoding

- Position encoding = P_t
  - Vaswani et al. 2017 decided to add it to X
  - So, Q_m_pe = Q_m + P_m; same for K_m_pe

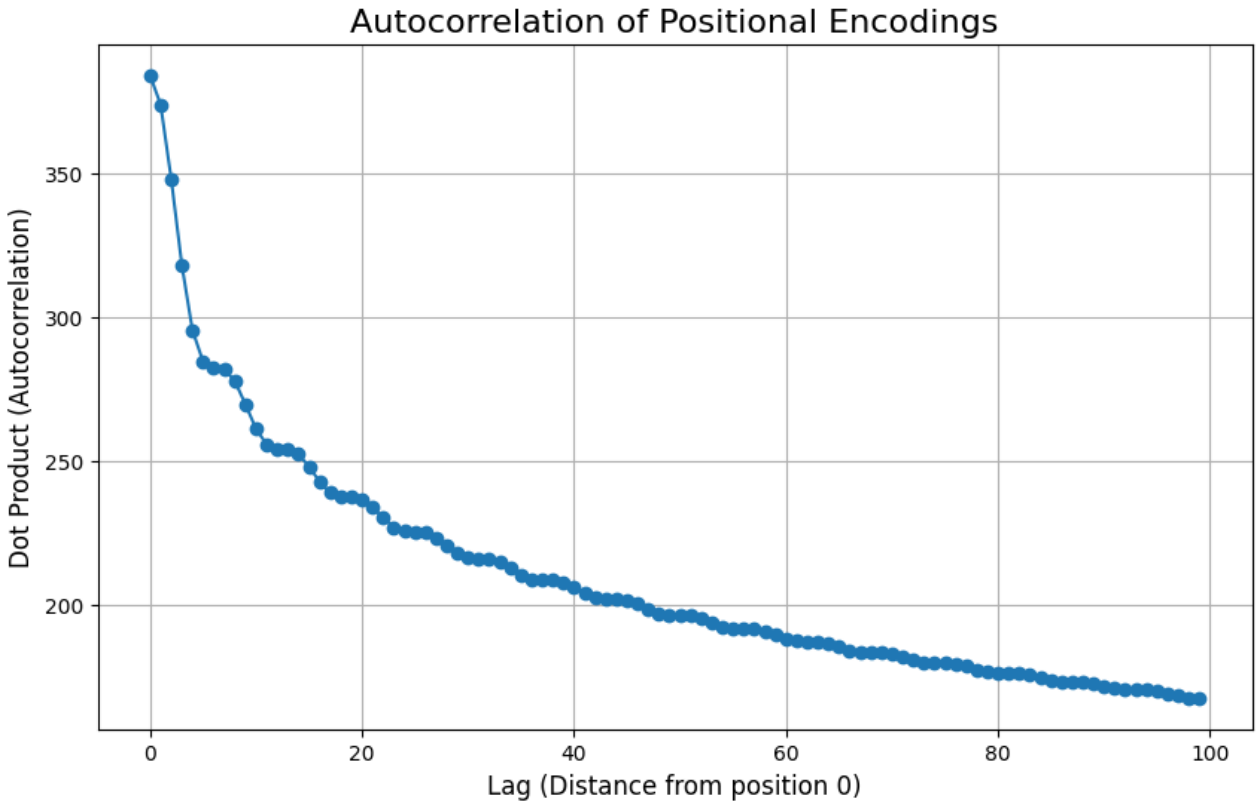


Position Encoding

- Position encoding= P_t
  - The P_m @ P_n term is used to give positional insight

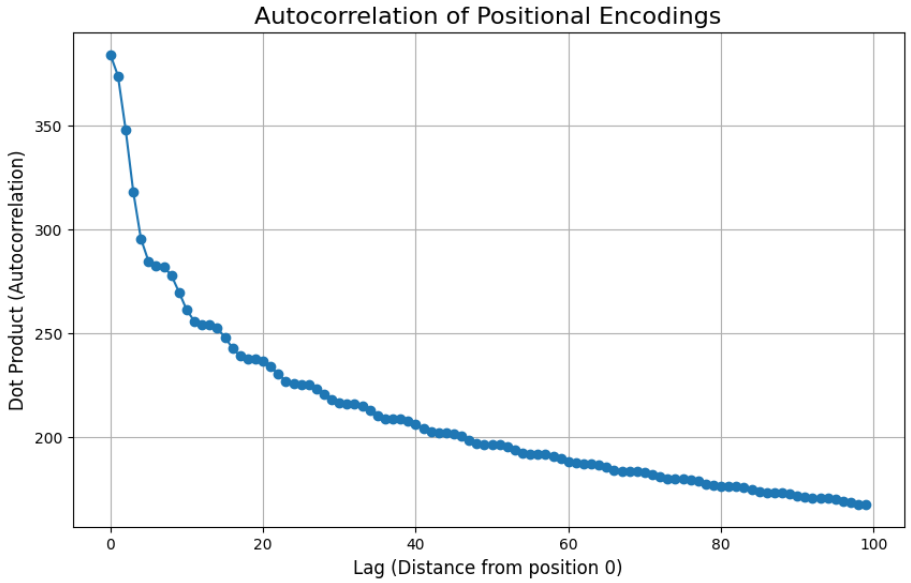
Position Encoding

- Position embedding = P_t
  - Actual autocorrelation curve for Vaswani et al 2017.



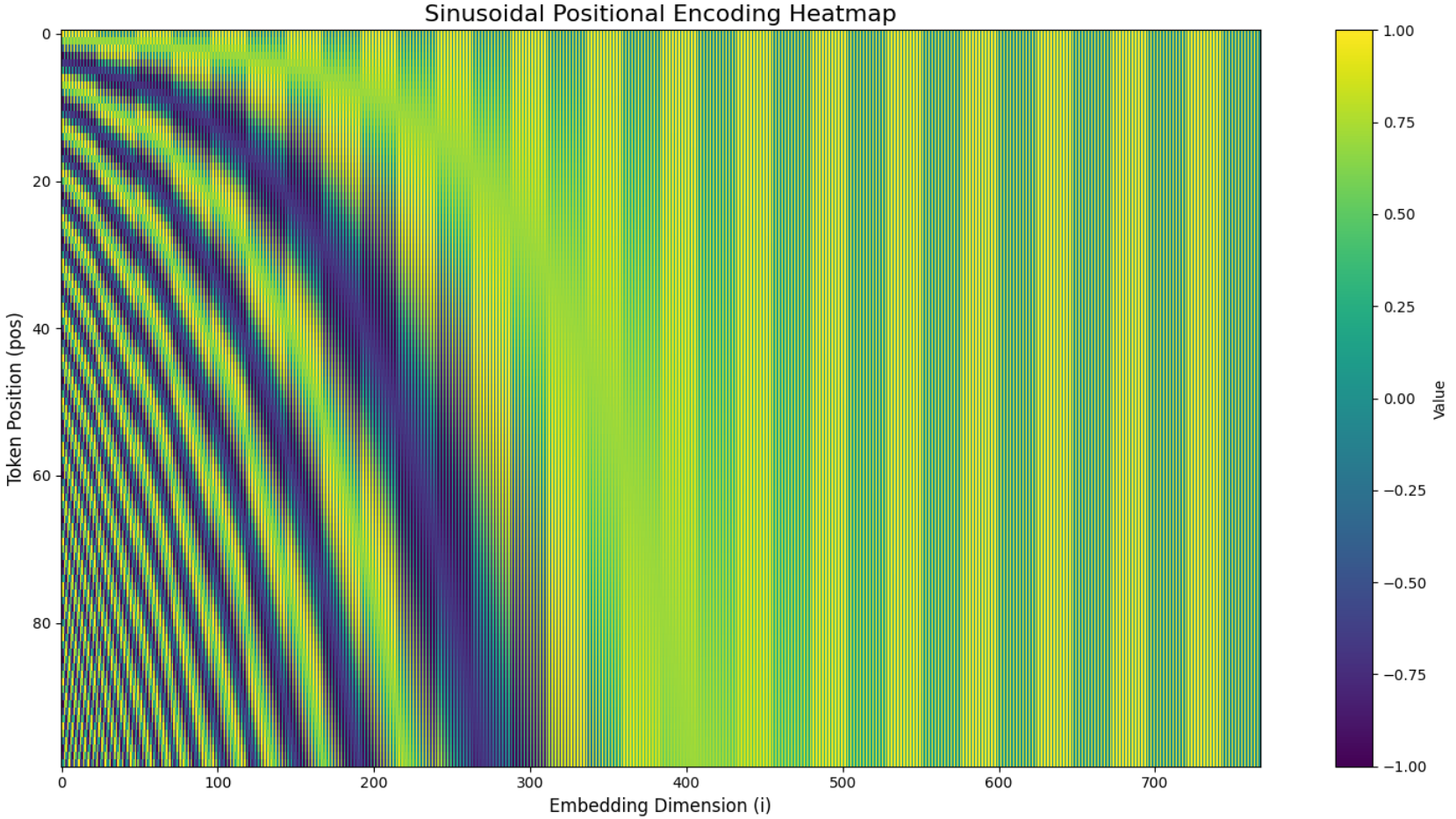
Position Encoding

- Position encoding= P_t
  - Attention gets relative positional insight from P_m @ P_n
  - M and N terms will be some distance apart, this dot product essentially an autocorrelation term.
  - If it falls smoothly with distance, we embed bias that closer tokens are more important and farther ones aren't



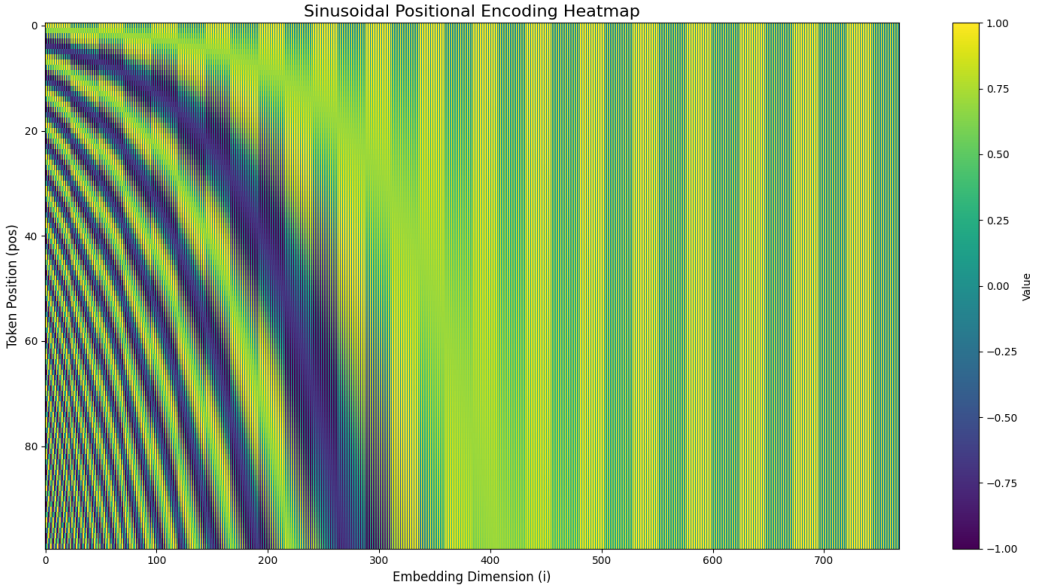
Positional Encoding

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



Positional Encoding

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



- The vector is made of sines and cosines of a harmonic series of frequencies
- All positions get unique values, it never repeats!

Poll @1207

A set of positional encodings (let's call them P_t for position t) must effectively encode both absolute and relative position. Which of the following statements incorrectly describes a necessary property or mechanism?

- The dot product $P_t \cdot P_{t'}$ must be a mathematical component of the final $Q_t \cdot K_{t'}$ attention score, allowing the model to factor in position when calculating relevance.
- The embeddings must be designed so that the relationship between any two positions (e.g., $P_t \cdot P_{t+k}$) depends only on the offset k, not on the absolute position t.
- To ensure all positions are unique, the embeddings for any two different positions (P_t and $P_{t'}$) must be perfectly orthogonal (i.e., $P_t \cdot P_{t'} = 0$).

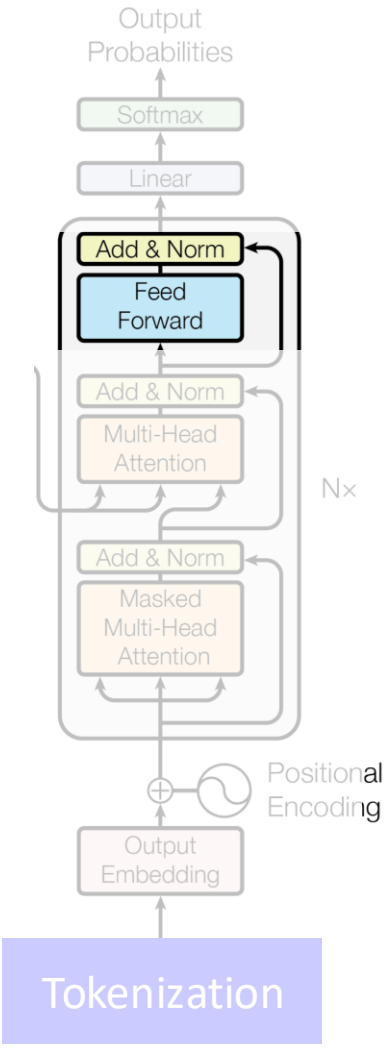
Poll @1207

A set of positional encodings (let's call them P_t for position t) must effectively encode both absolute and relative position. Which of the following statements correctly describes a necessary property or mechanism?

- The dot product $P_t \cdot P_{t'}$ must be a mathematical component of the final $Q_t \cdot K_{t'}$ attention score, allowing the model to factor in position when calculating relevance.
- The embeddings must be designed so that the relationship between any two positions (e.g., $P_t \cdot P_{t+k}$) depends only on the offset k, not on the absolute position t.
- To ensure all positions are unique, the embeddings for any two different positions (P_t and $P_{t'}$) must be perfectly orthogonal (i.e., $P_t \cdot P_{t'} = 0$).

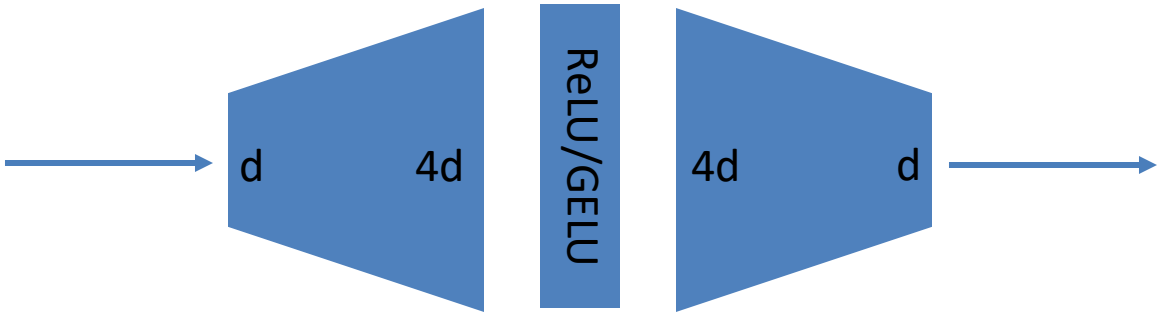
Transformer Architecture

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



Feed-Forward Block

- Just a MLP!

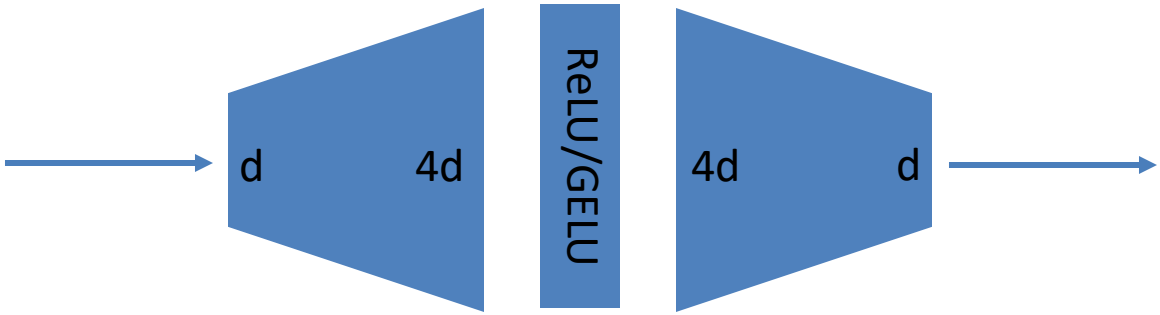


- This is where the main memorization happens!

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!

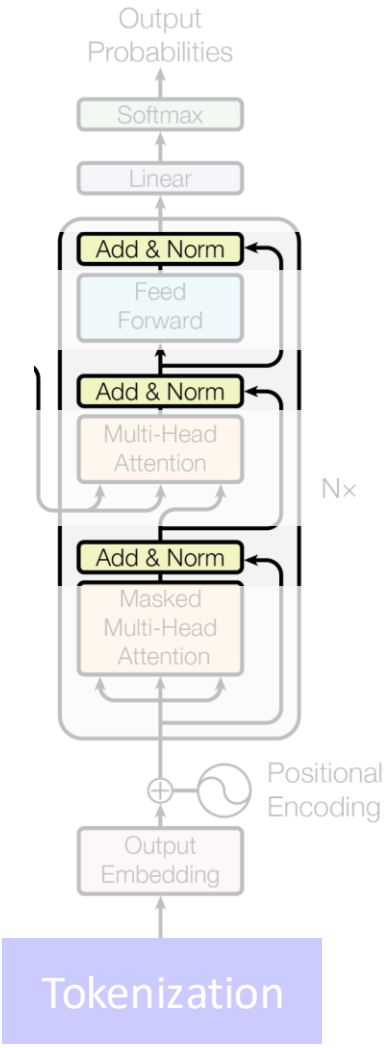


- Why expand: to let it understand context better ->4x more non-linearities
- More connections created -> more 'memory'

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

Transformer Architecture

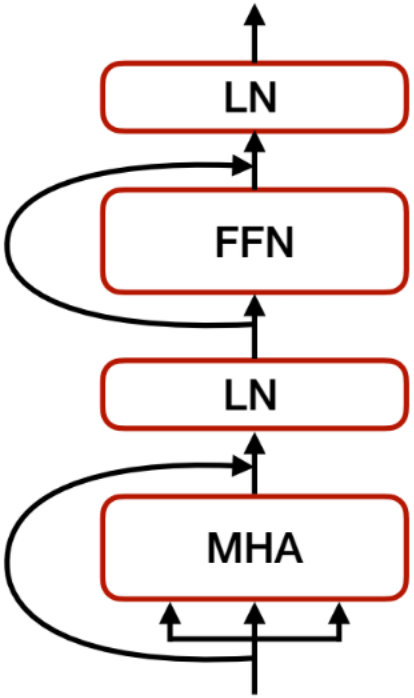
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- **Add & Norm**
- Output Projection Layer



Residual and Normalization

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

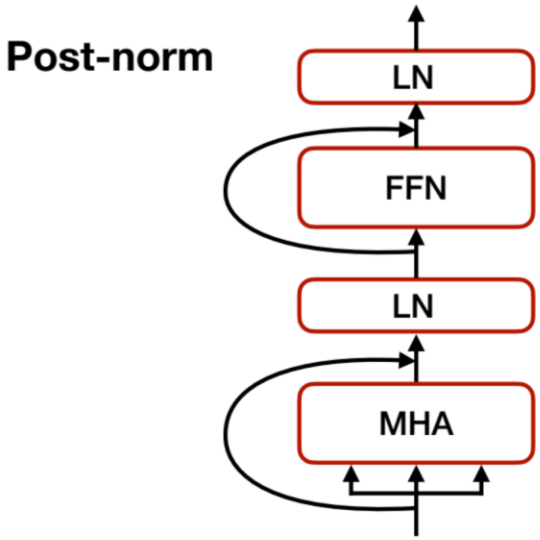
- Layer Norm. normalize each token by its embedding size dimension
  - For more stable training



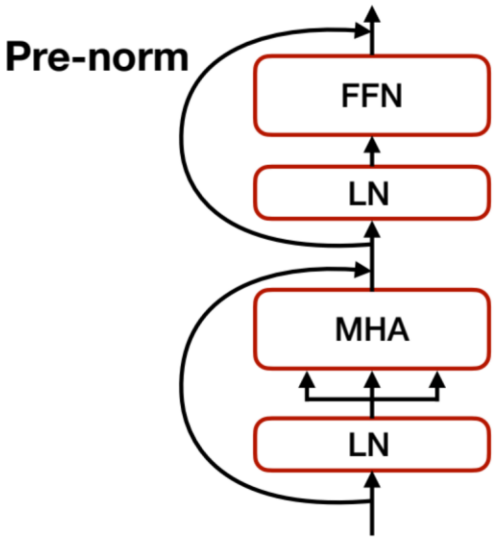
Position of Normalization

- Post-Norm vs Pre-Norm

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



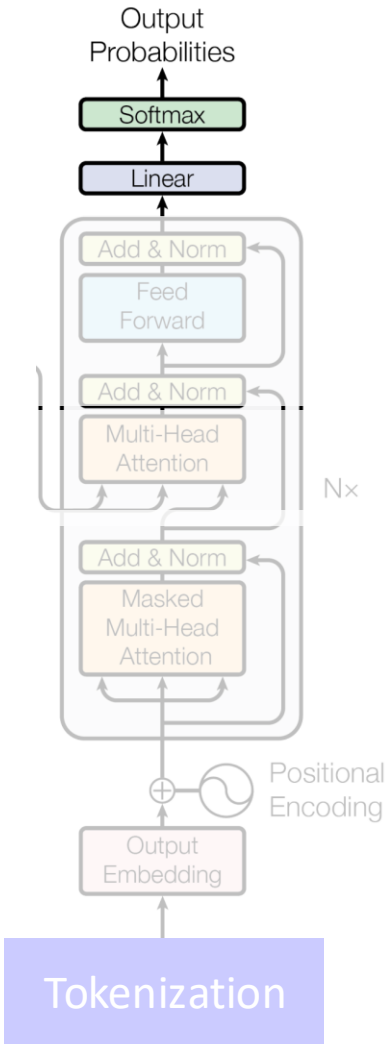
$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + F_t\left(\mathrm{Norm}\left(\boldsymbol{x}_t\right)\right)$$



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

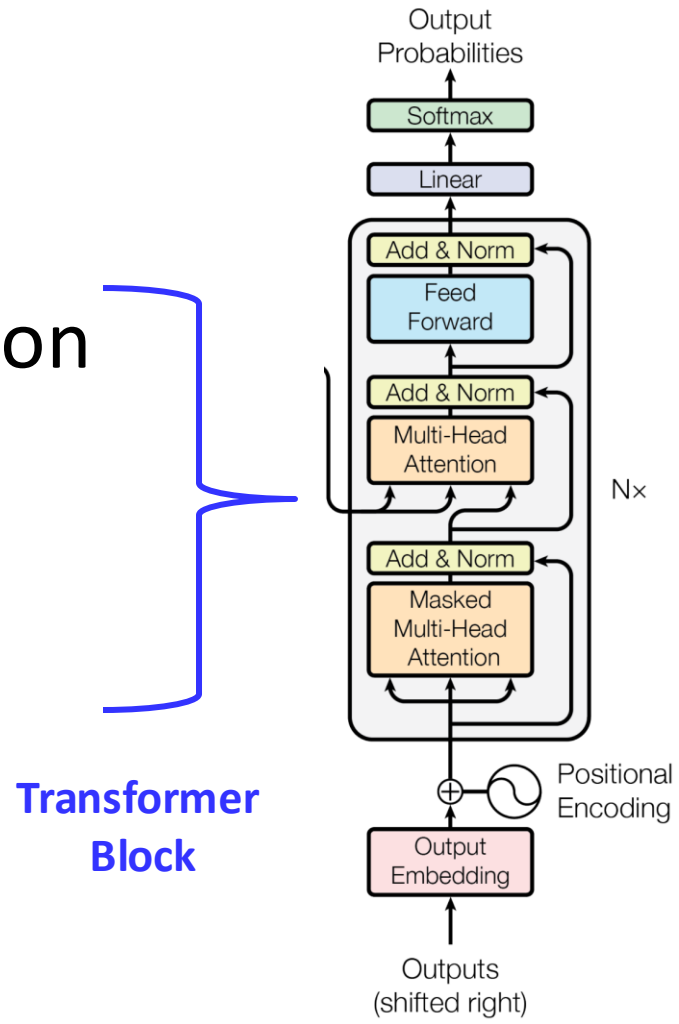
Transformer Architecture

- 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 @1208

Which of the following are true about self-attention?

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

Poll @1208

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

Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws
- Quantizing Transformers
- Interpreting Transformer – attention, logit lens

Overview

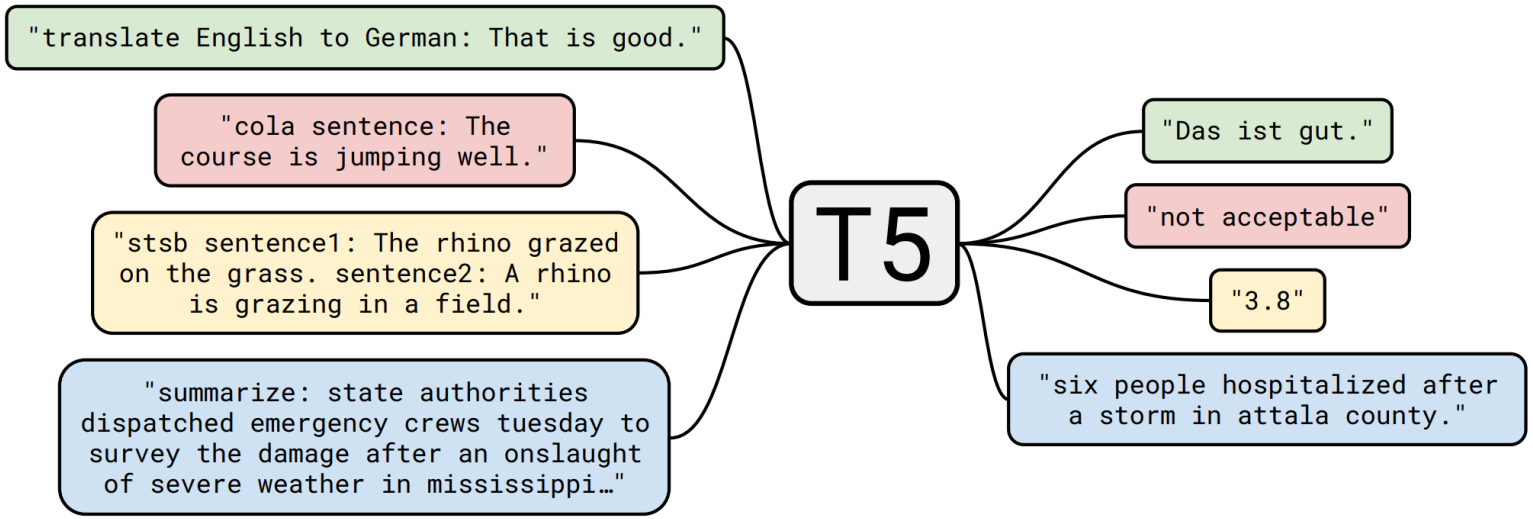
- Architecture
  - Encoder-Decoder
  - Encoder-Only
  - Decoder-Only
- Position Encoding
  - Relative Position Encoding
  - Rotary Position Encoding
- Efficient Attention Mechanism
  - Grouped Query Attention
  - Multi Query Attention
  - Flash Attention
  - Multi-head Latent Attention

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 - T5

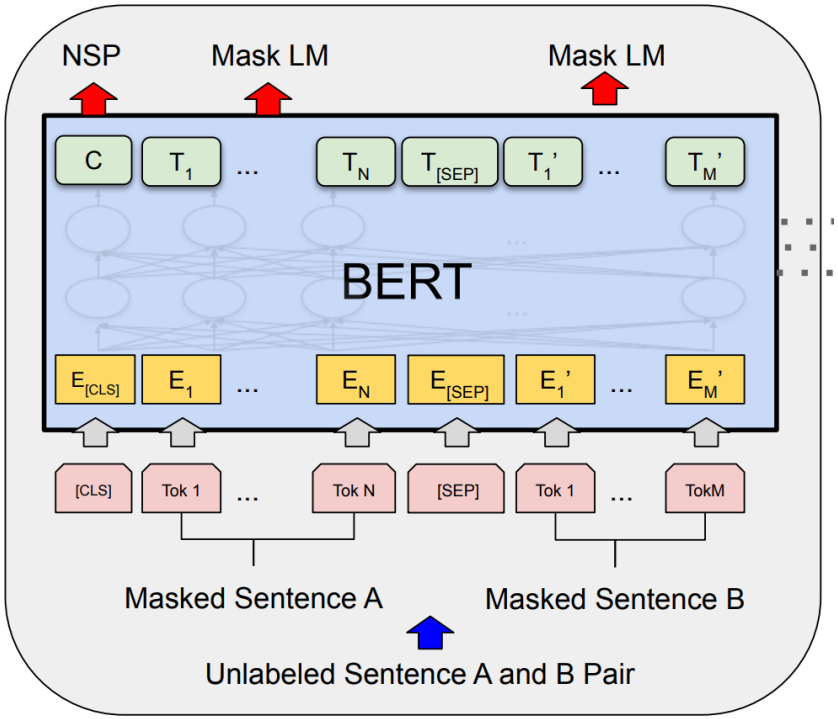
- Encoder-Decoder architecture as in the original transformer paper
- A text-to-text model on various NLP tasks



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

Encoder-Only - BERT

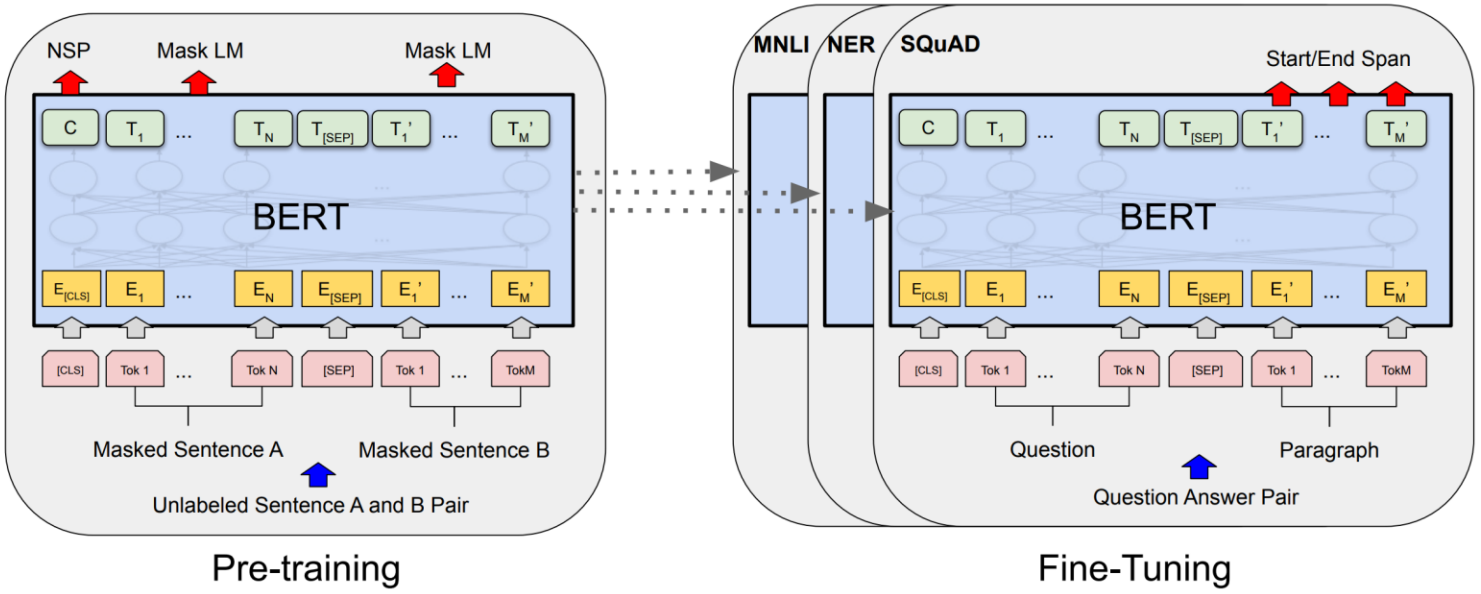
- Bidirectional Encoder Representations from Transformers (BERT)
  - Encoder-only arch.
- Trained with
  - Mask token prediction
  - Next sentence prediction



Pre-training and then Fine-Tuning

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

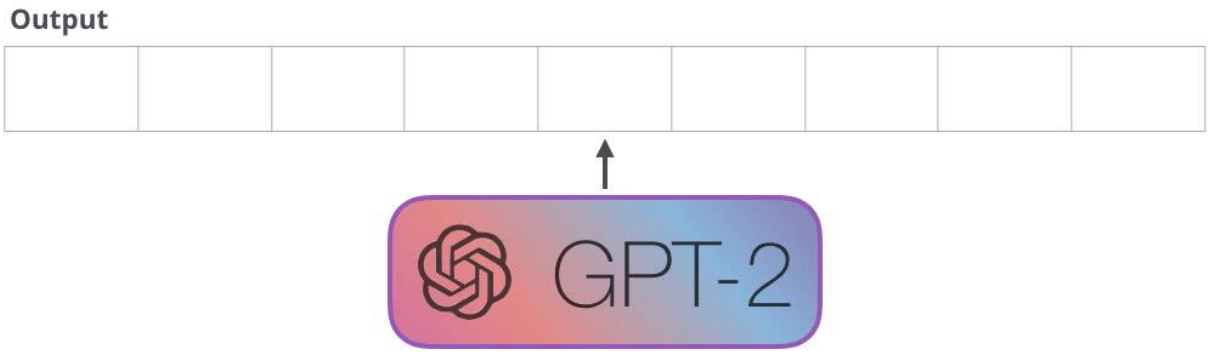
Fine-tuning on specific downstream tasks
- Machine translation
- Question answering



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

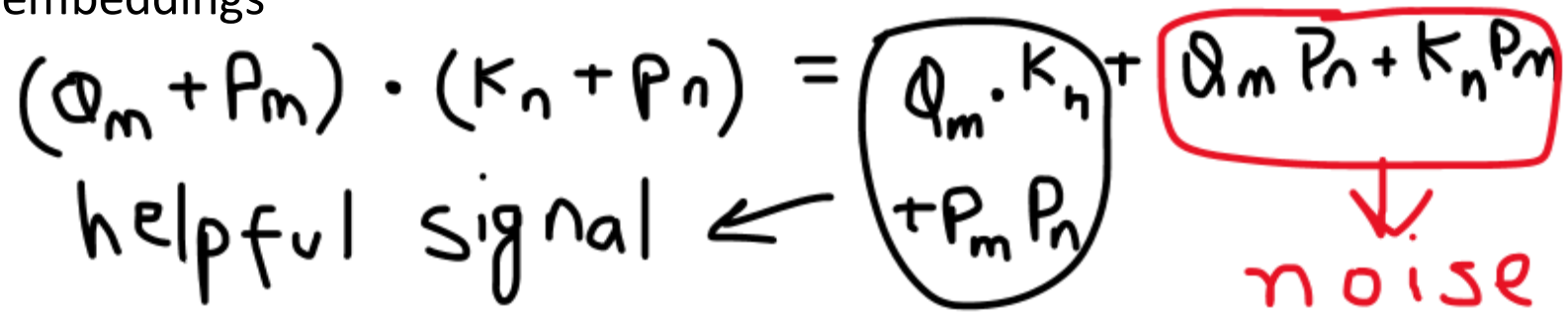
- GPT-2
  - Pre-training and fine-tuning on specific tasks
- GPT-3
  - zero-shot capability
  - in-context learning
  - ChatGPT!
- GPT-4

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



- But, it's for a fixed length! Unable to generalize to longer input sequence and then there are noise terms as well.

Absolute Position Encoding

- Absolute position embedding fuses the position information into input embeddings

$$(q_m + p_m) \cdot (k_n + p_n) = q_m \cdot k_n + q_m p_n + k_n p_m + p_m p_n$$

helpful signal ← $q_m \cdot k_n + p_m p_n$; $q_m p_n + k_n p_m$ → noise

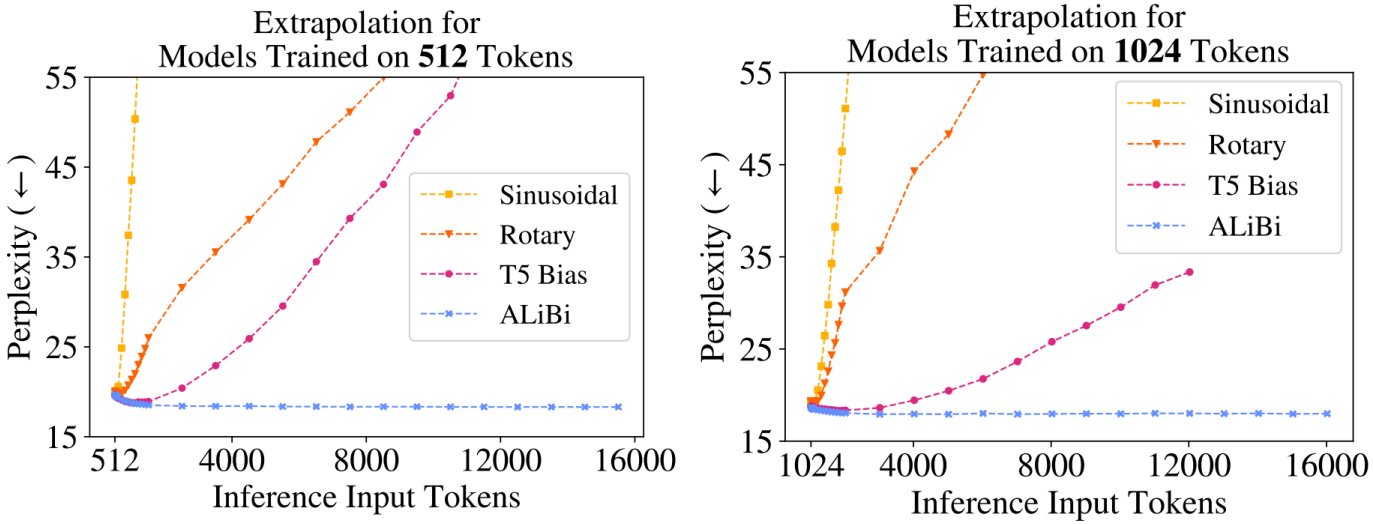
- But, it's for a fixed length! Unable to generalize to longer input sequence and then there's noise.
- We only need the relative position insight in attention, why not add just that?

$$Att_n = q_m \cdot k_n + r[m-n] \mid p_m \cdot p_n$$

Relative Position Encoding

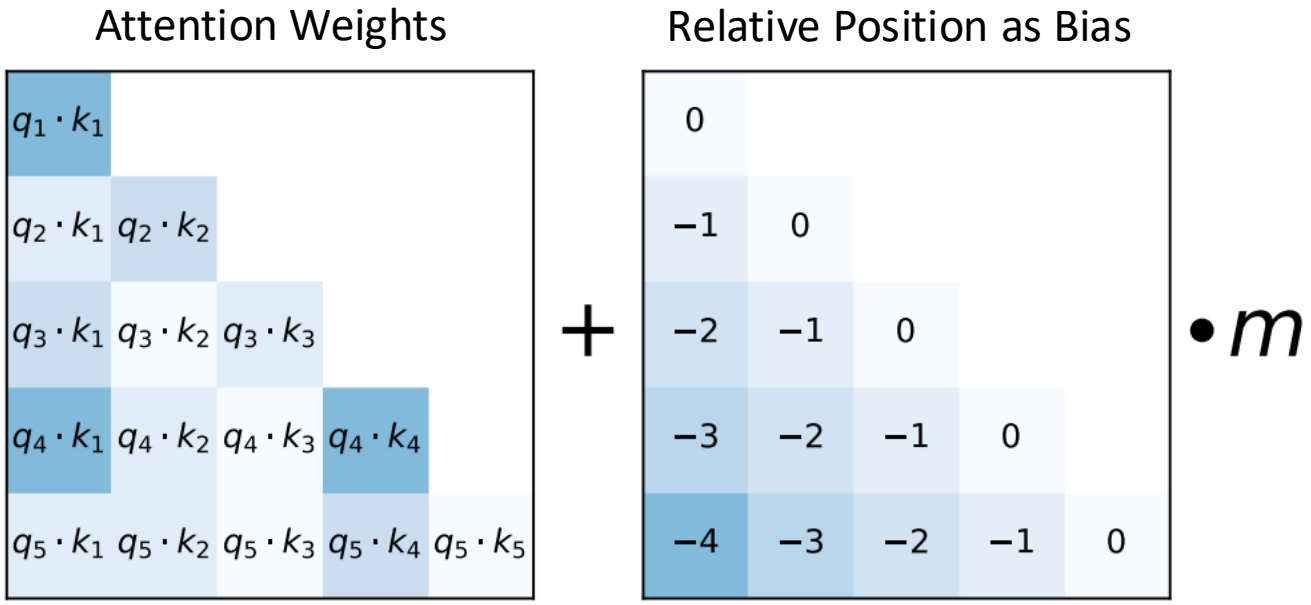
- Relative position embedding fuses position information into attention matrices

- 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

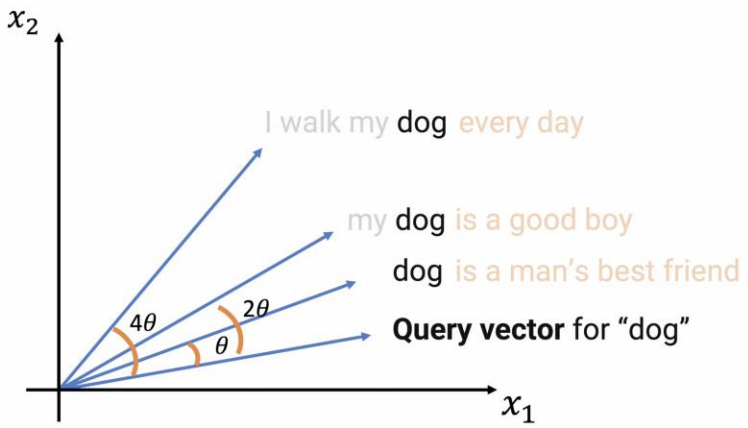
- Used in Large Language Models such as LLAMA
- Rotate the embedding in 2D space

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

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

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

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



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

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

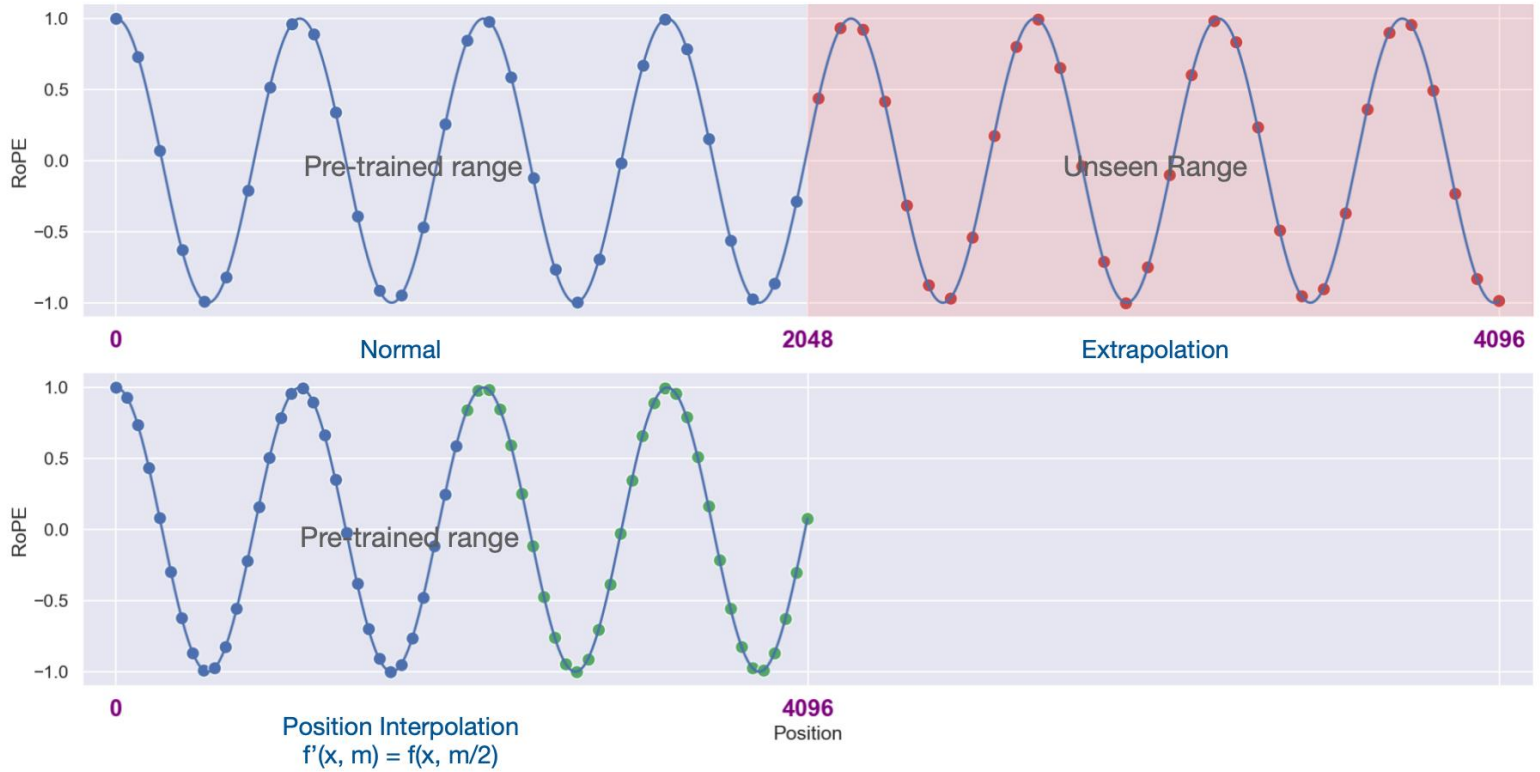
$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

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

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

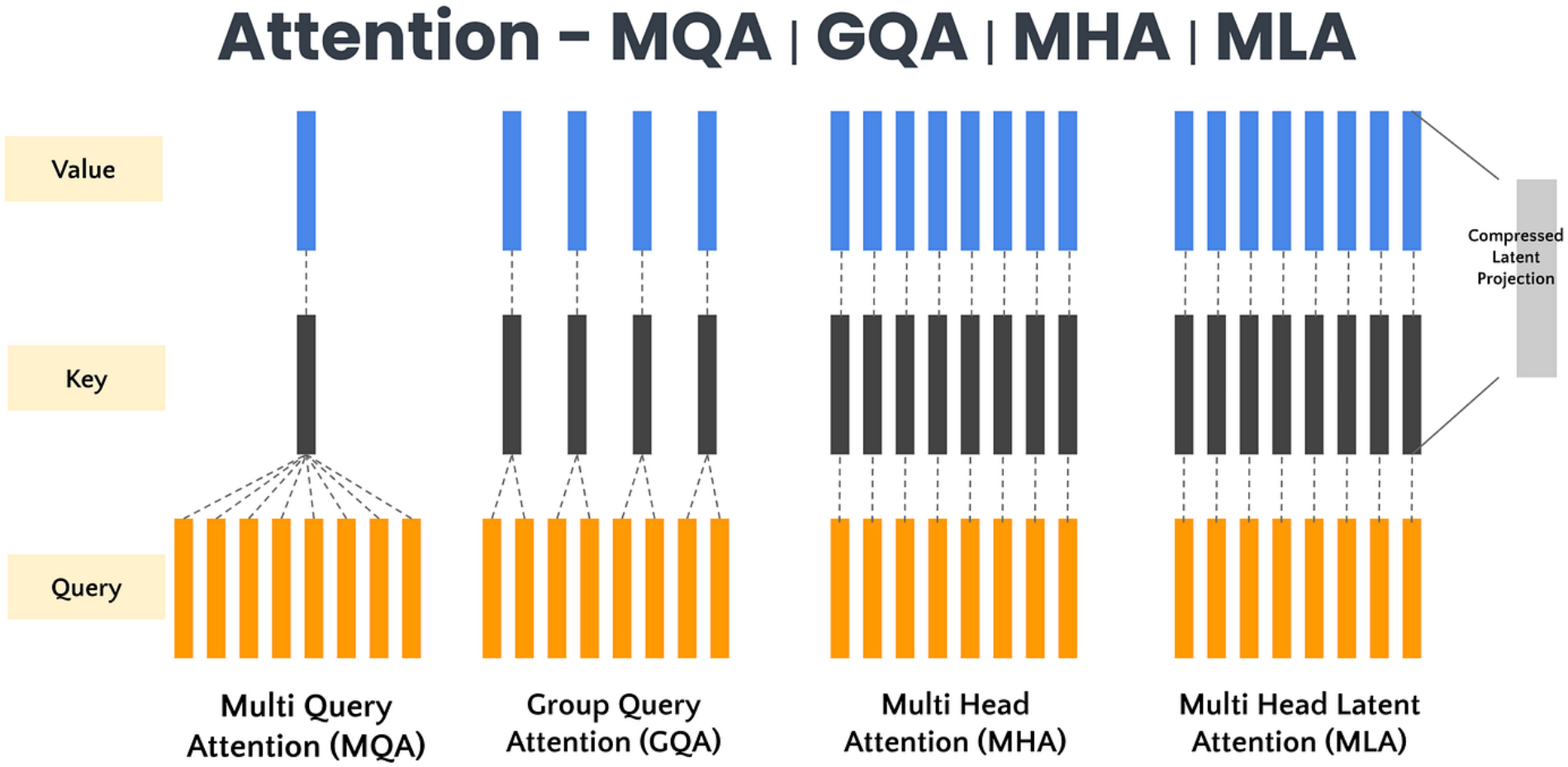
- Self-attention has quadratic complexity to input length

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

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

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

Attention Types



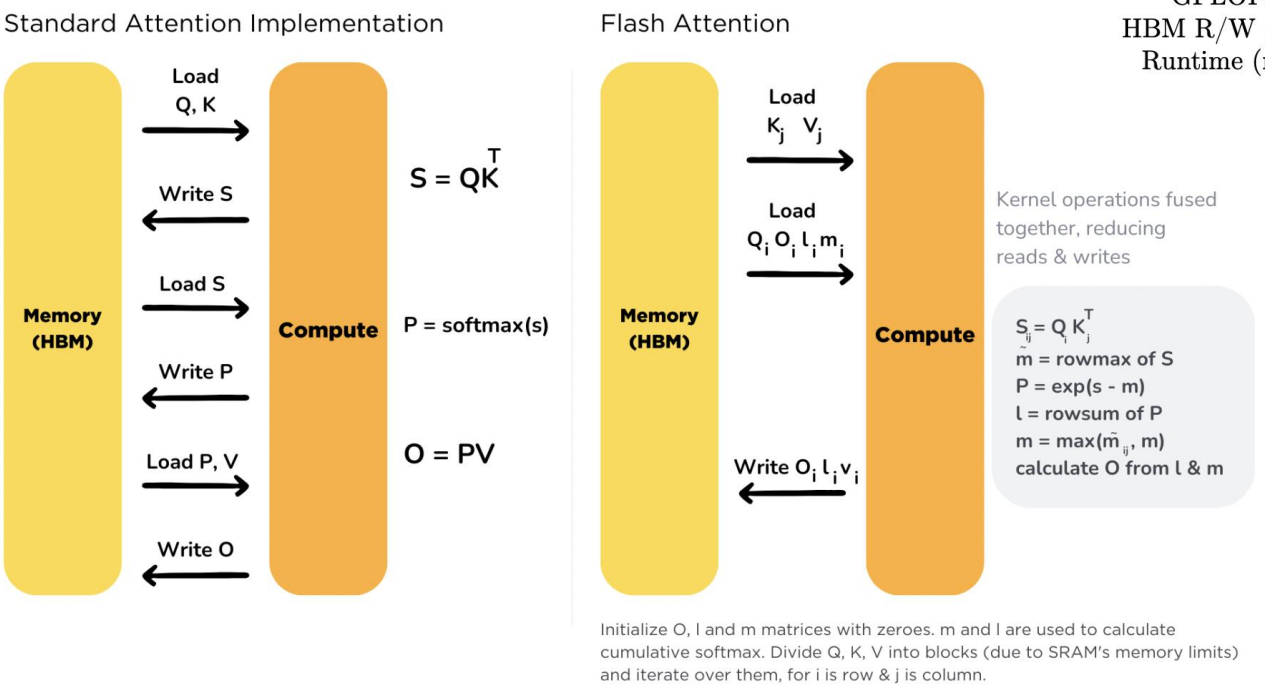
Liu, Aixin, et al. "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model." arXiv preprint arXiv:2405.04434 (2024)

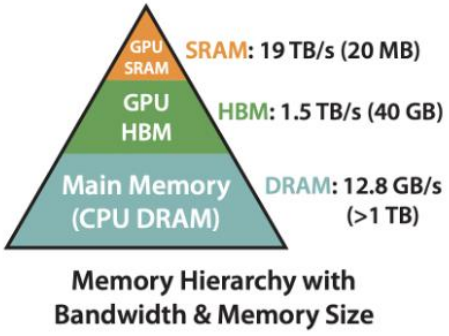
https://verticalserve.medium.com/group-query-attention-58283b337c65

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:

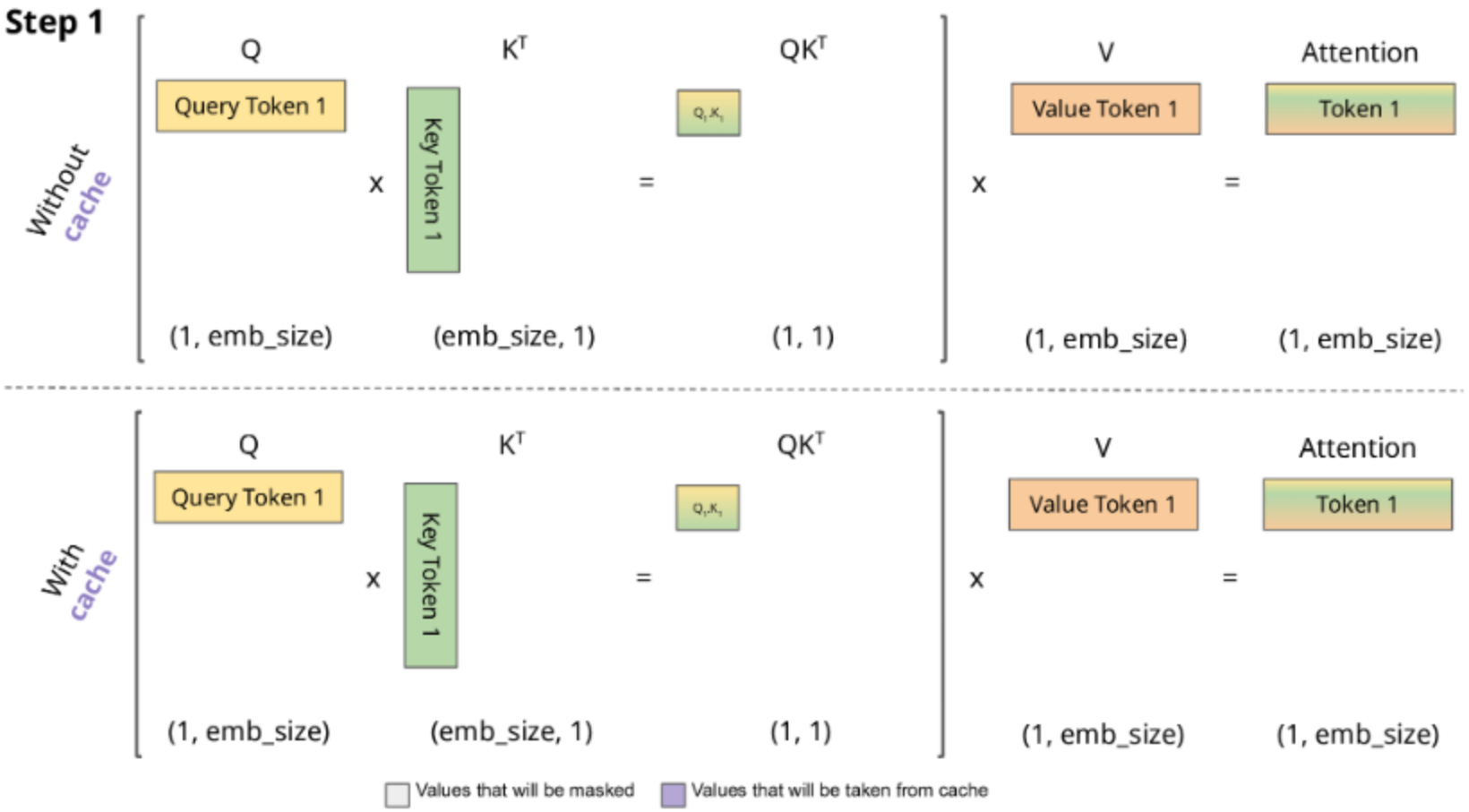
Flash Attention: $\mathbf{O}\left(\frac{N^2 d^2}{M}\right)$

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

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

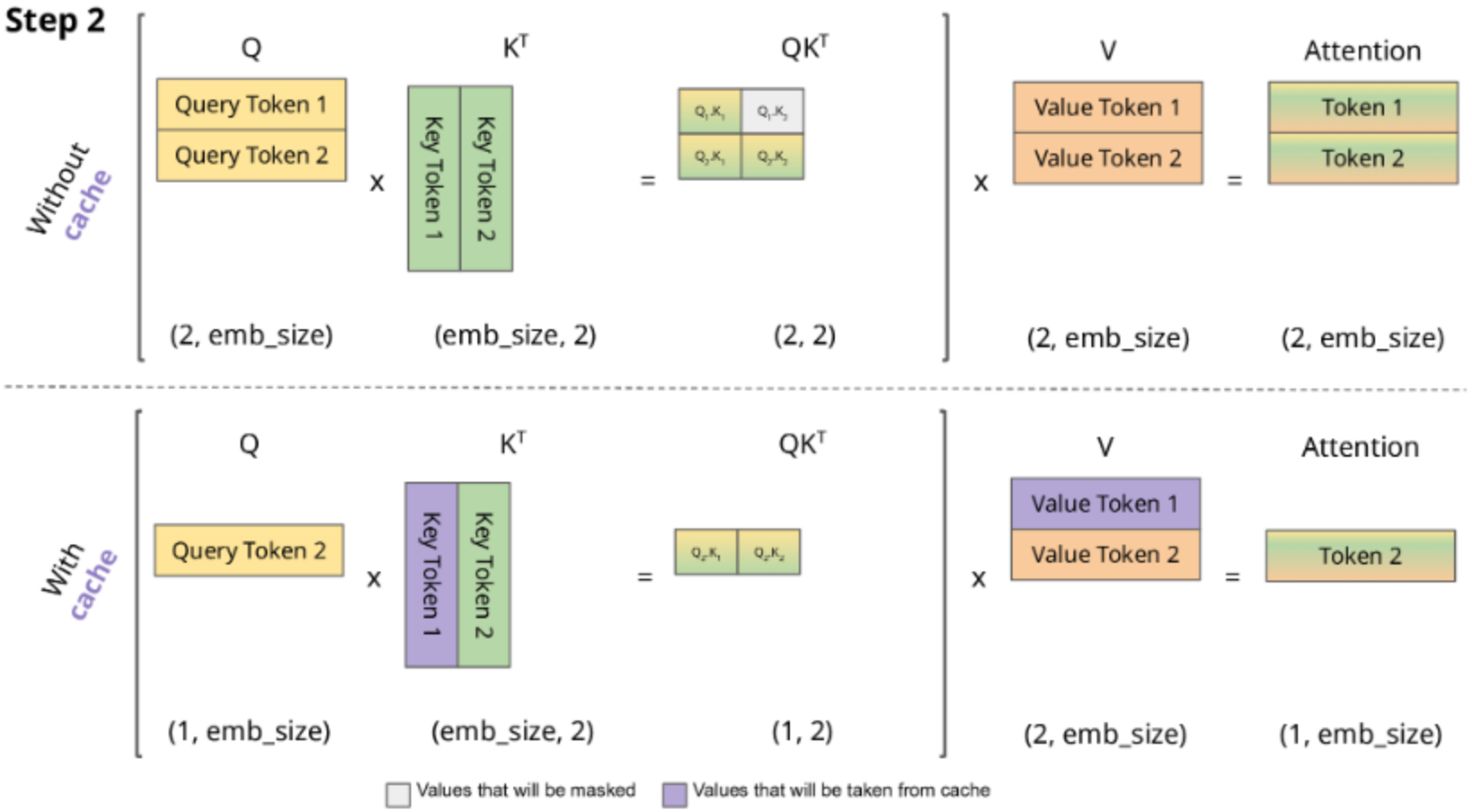
Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." Advances in Neural Information Processing Systems 35 (2022): 16344-16359.

KV- Caching



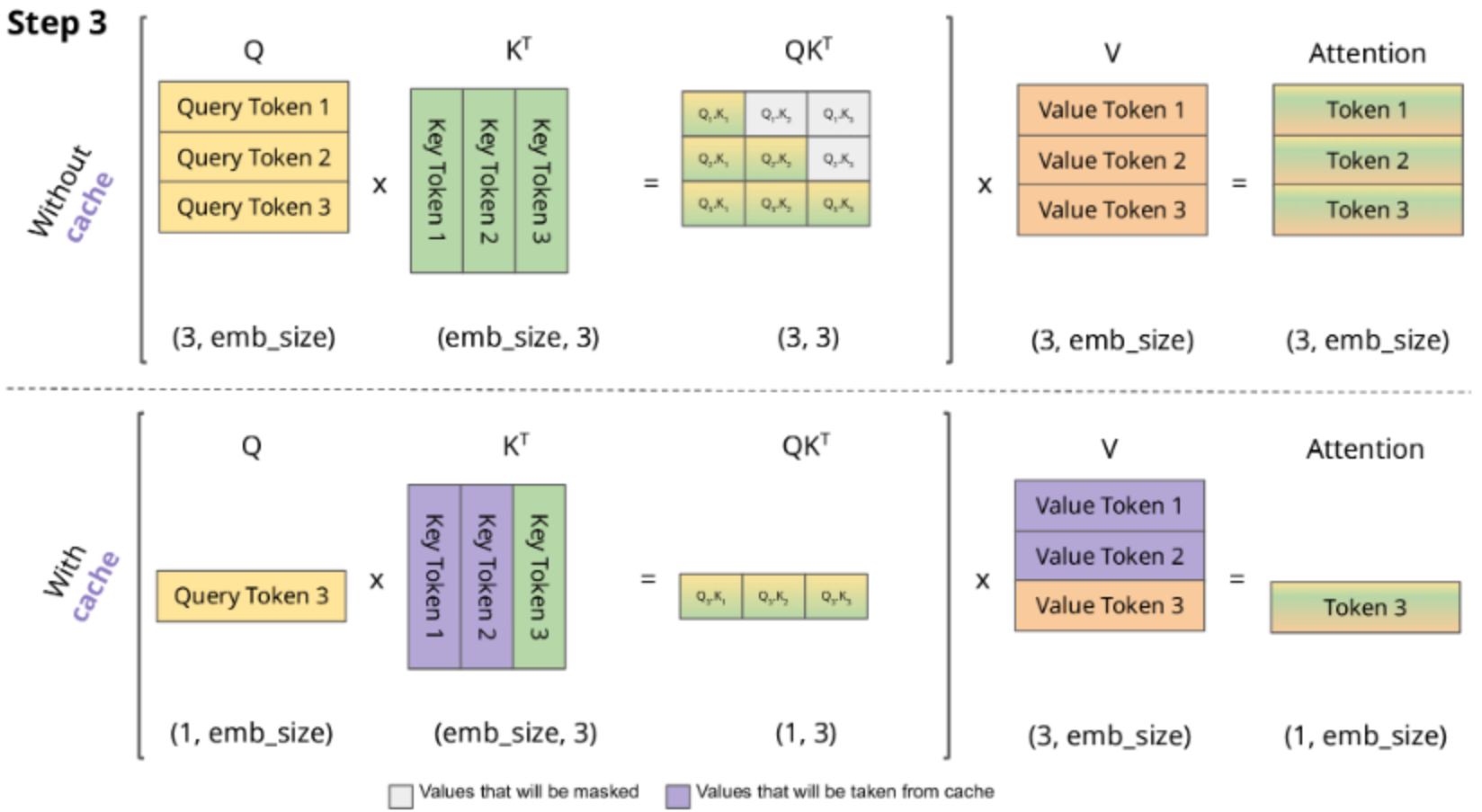
Transformers KV Caching Explained: https://medium.com/@joaolages/kv-caching-explained-276520203249

KV- Caching



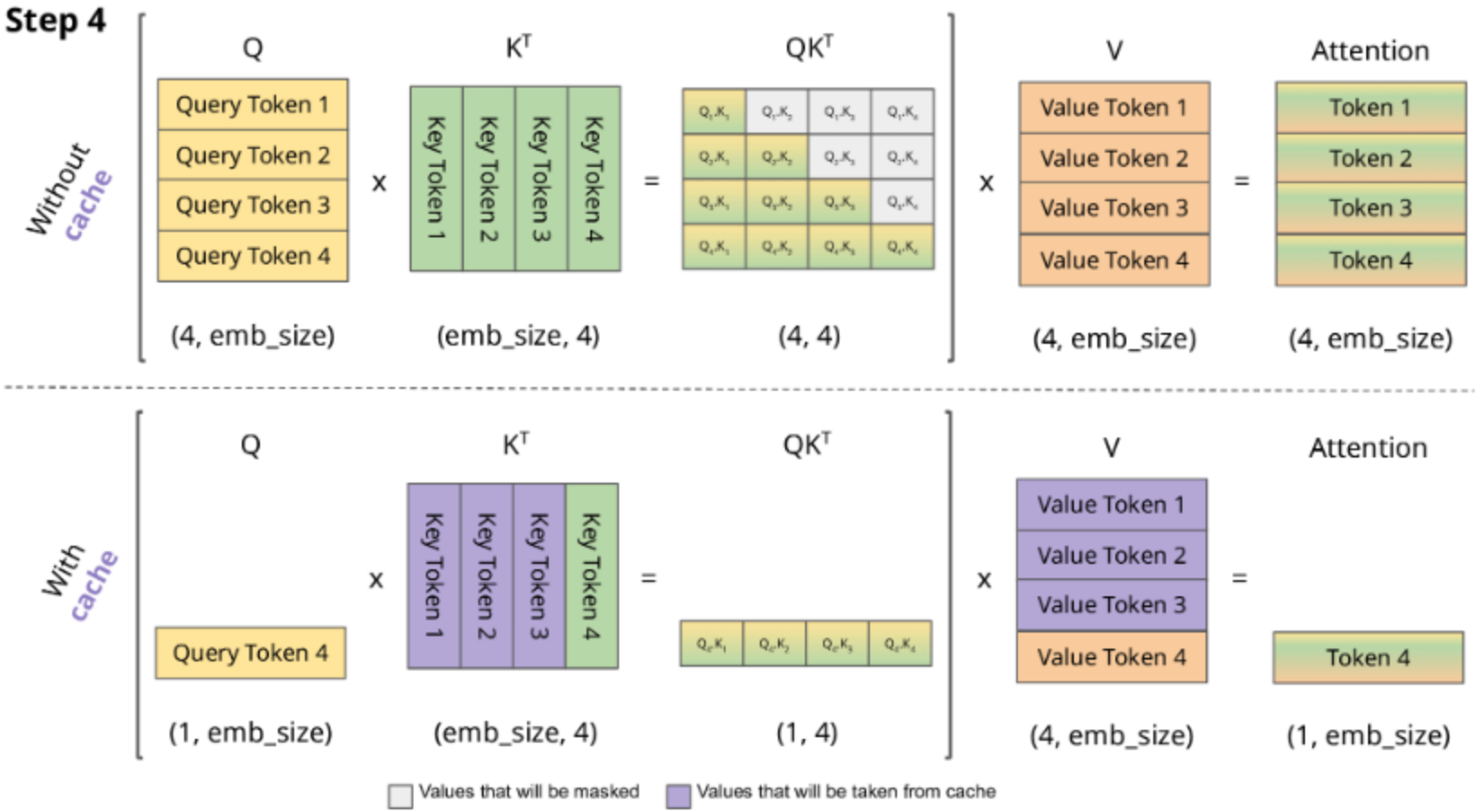
Transformers KV Caching Explained: https://medium.com/@joaolages/kv-caching-explained-276520203249

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 @1209

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 @1209

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
- Interpreting Transformer – attention, logitlens

Transformer in Vision and Audio

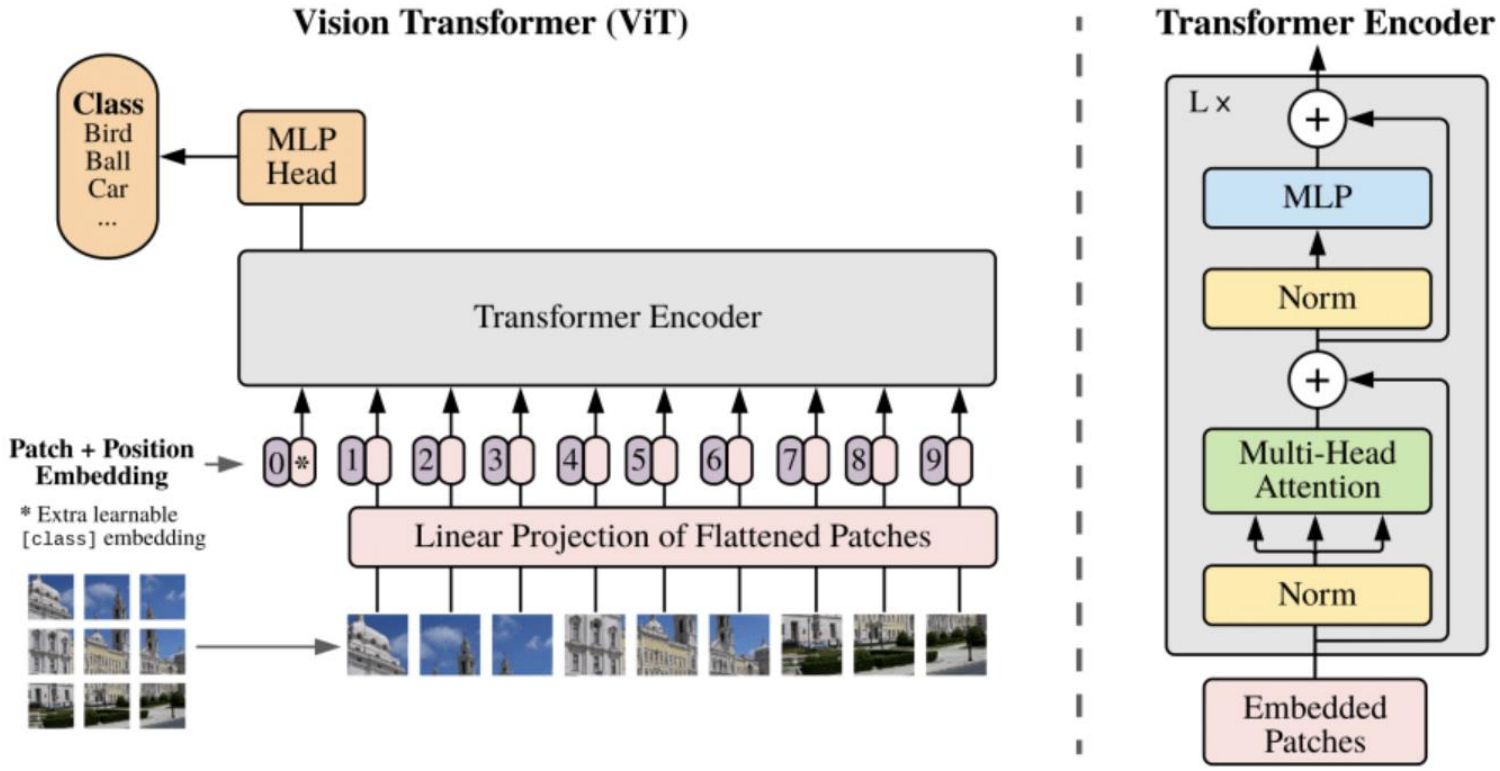
Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizer

Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizers

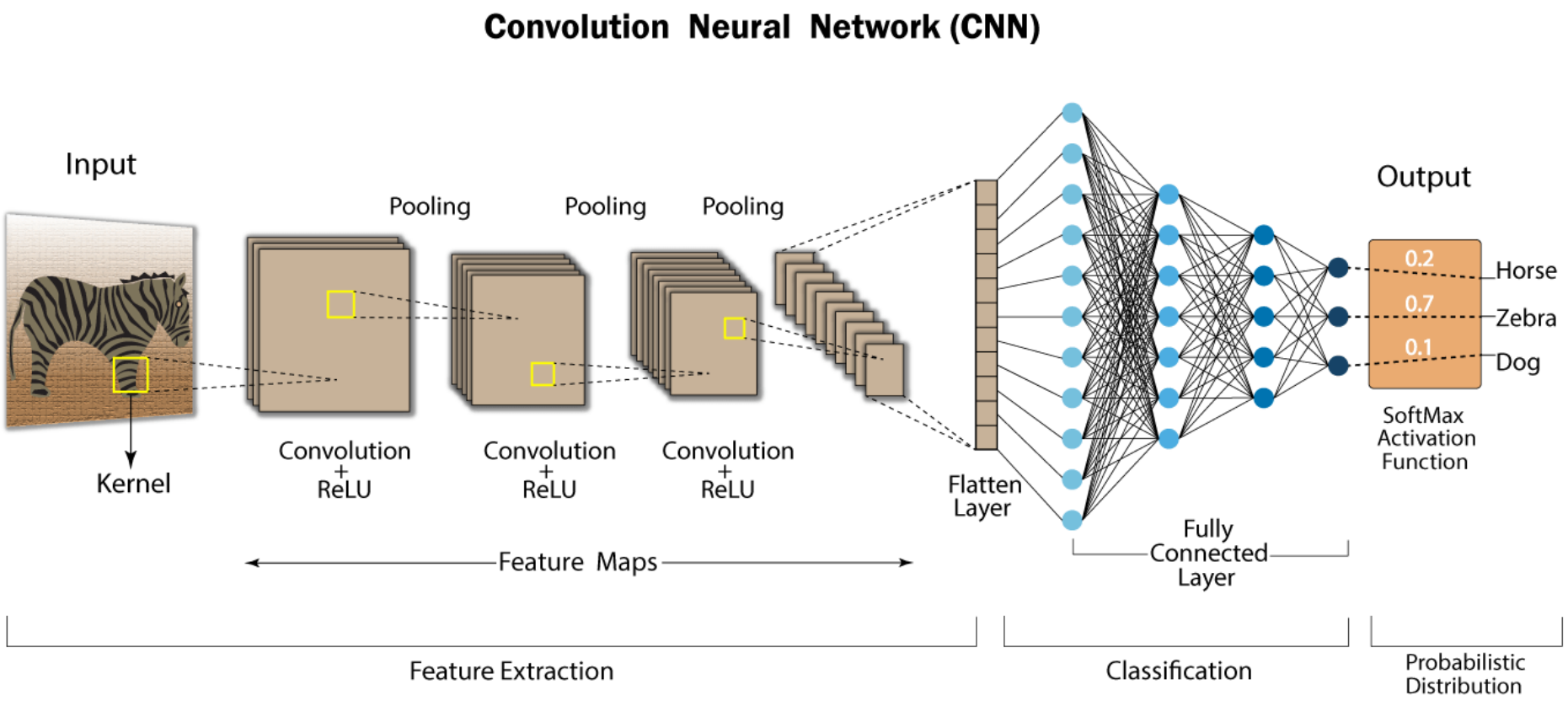
Vision Transformer (ViT)



- 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

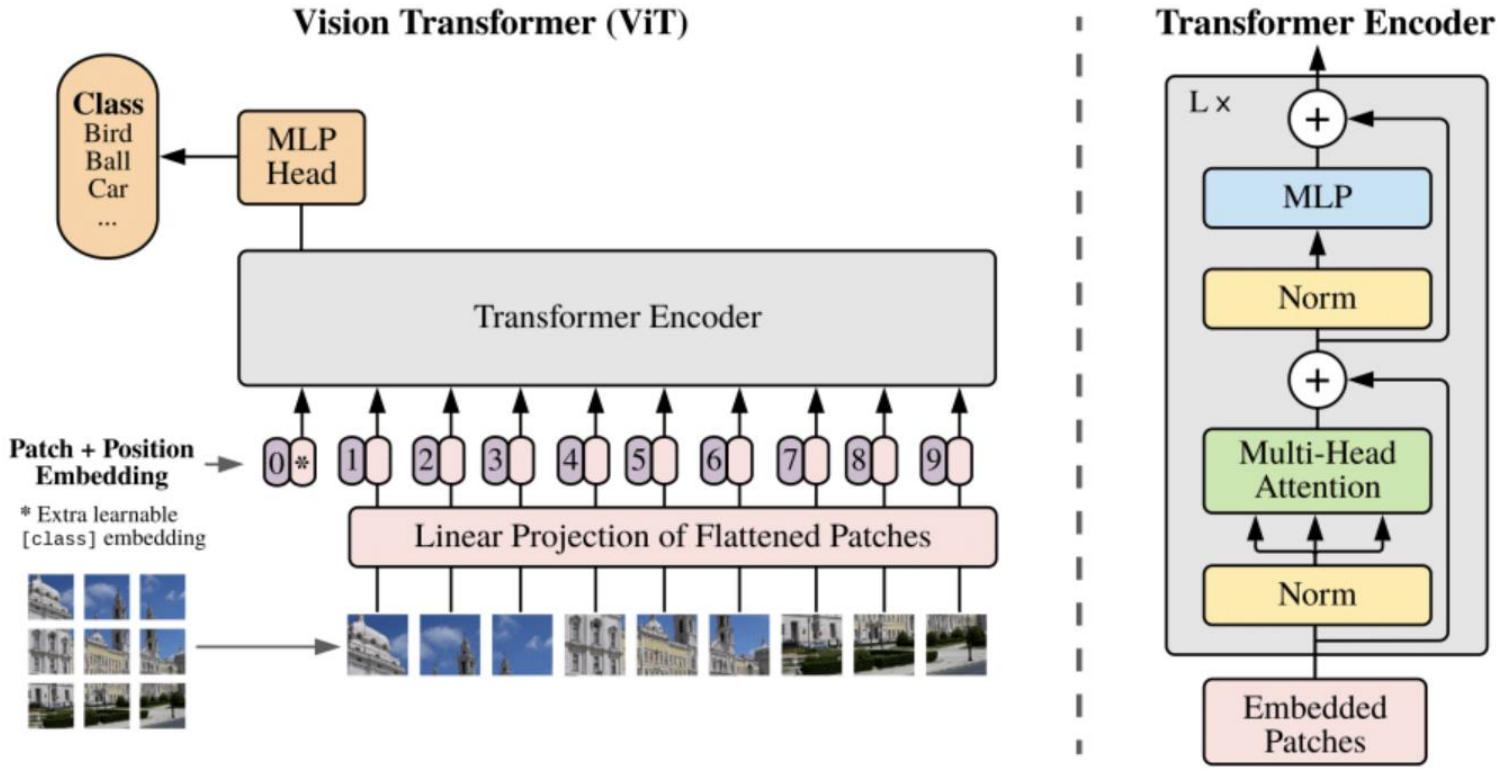
ViT

- 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.

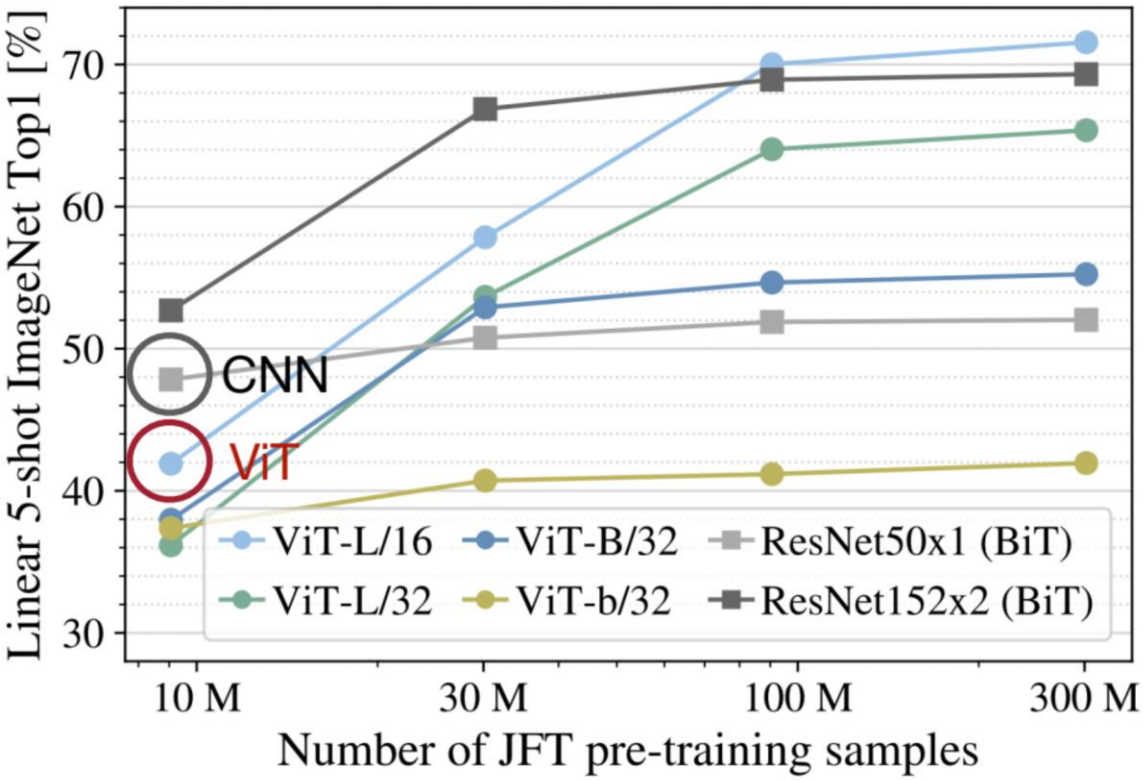
ViT



- 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

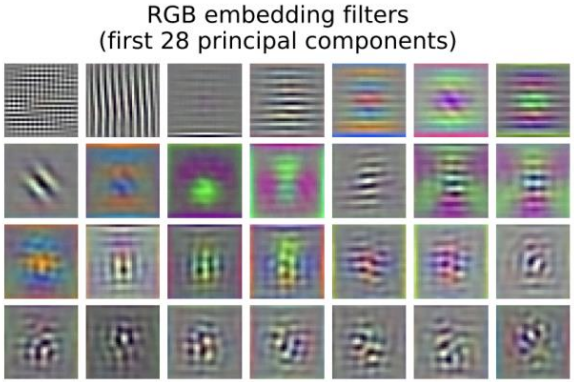


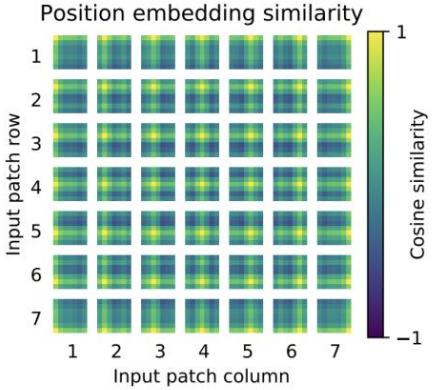
- Inferior performance compared to CNN when dataset size is limited – Why?

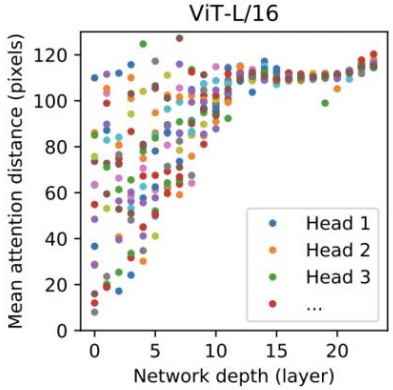
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

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!





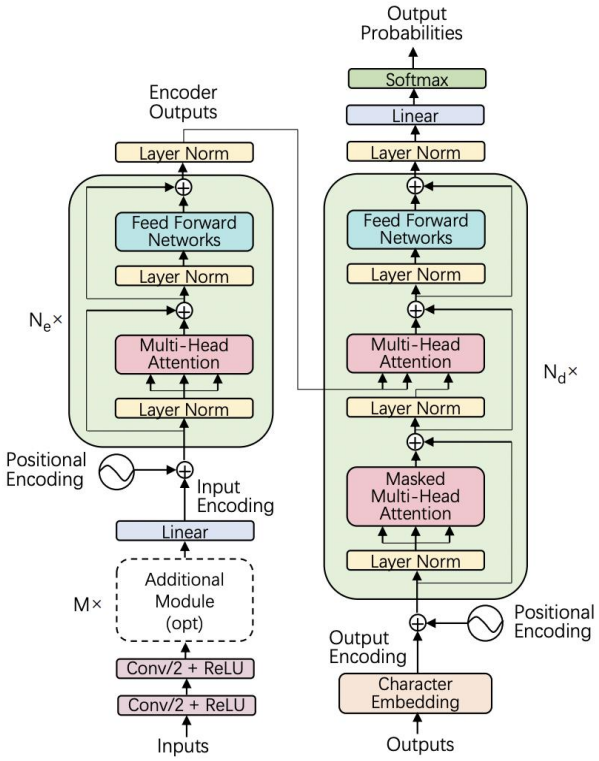


Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

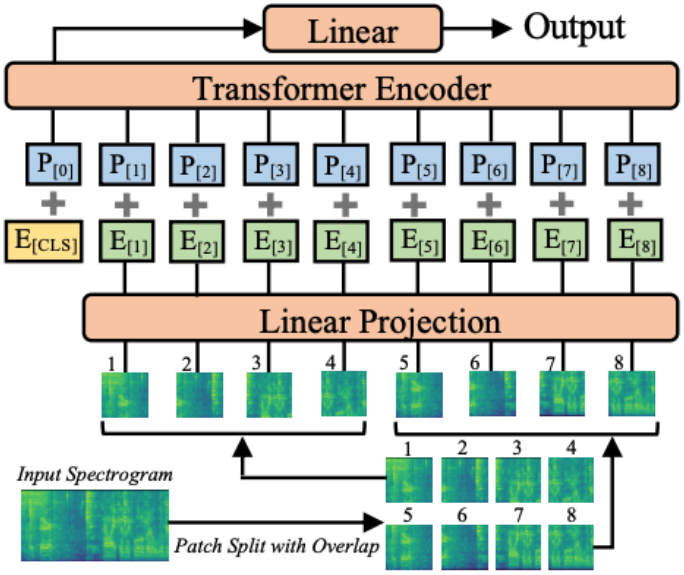
Overview

- Vision Transformer Architecture
- Transformer in Audio
- Tokenizer

Transformer in Audio



Speech Transformer for ASR



Audio Spectrogram Transformer

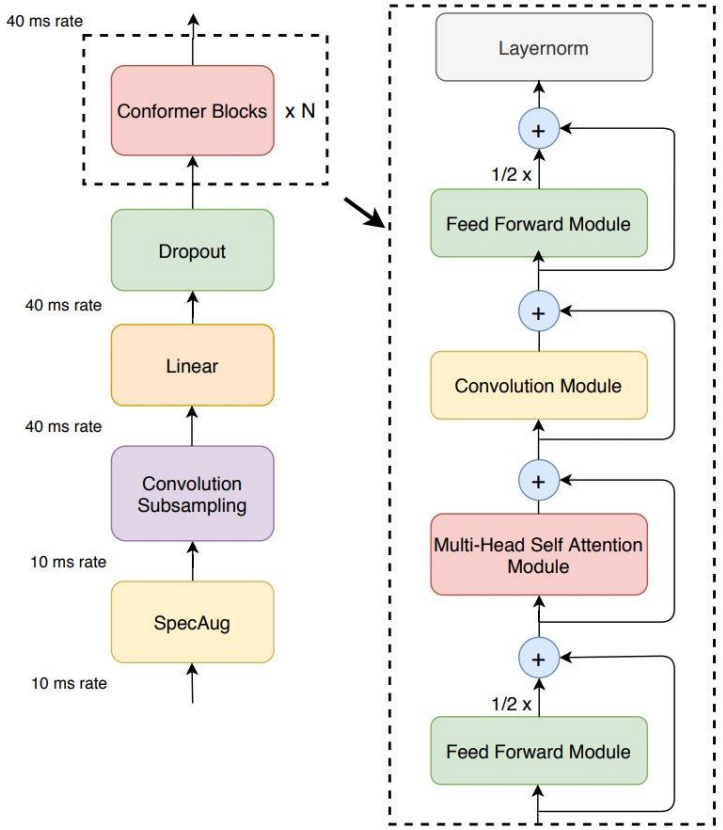
[1] Dong, Linhao, Shuang Xu, and Bo Xu. "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition." *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018.

[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.

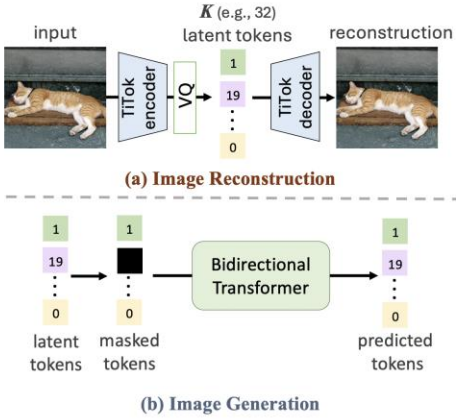


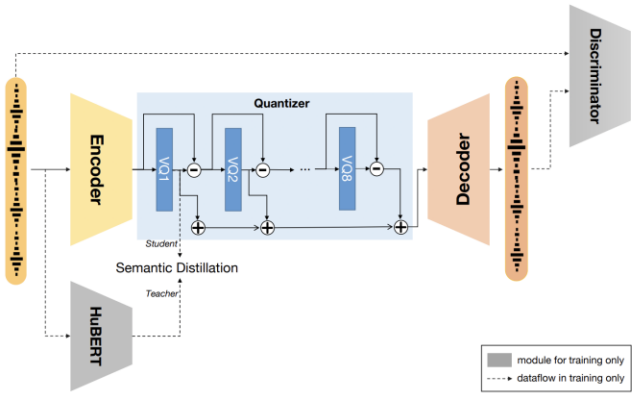
Gulati, Anmol, et al. "Conformer: Convolution-augmented transformer for speech recognition." *arXiv preprint arXiv:2005.08100* (2020).

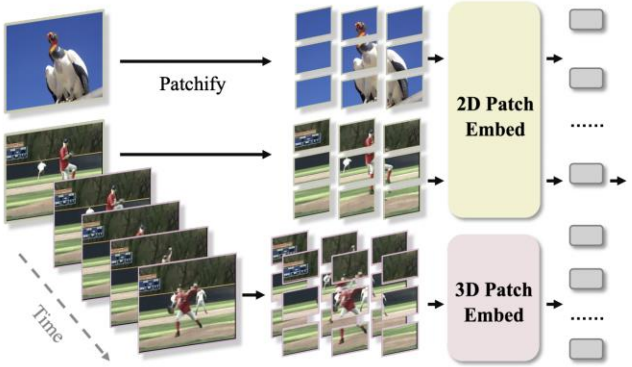
Overview

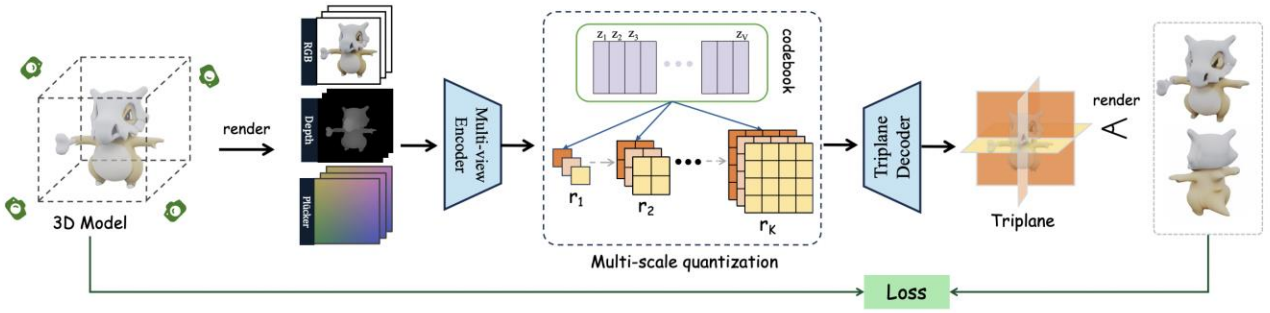
- Vision Transformer Architecture
- Transformer in Audio: Conformer
- Tokenizer

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 @ 1210,1211

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 @ 1210,1211

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

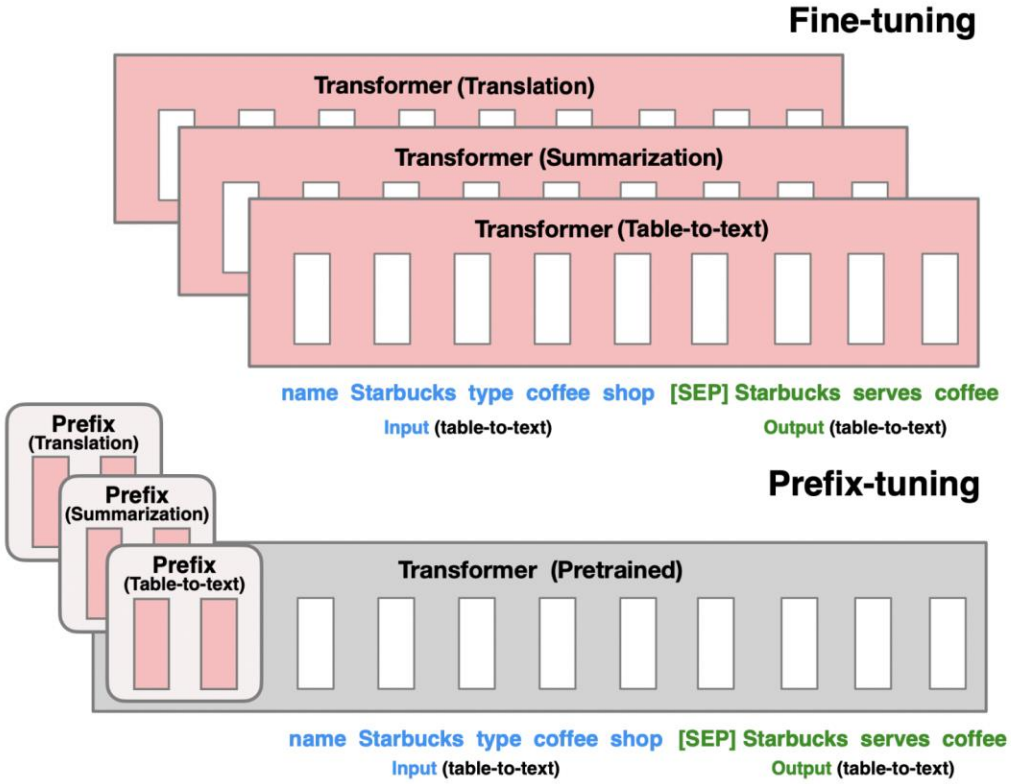
Parameter Efficient Tuning

- 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

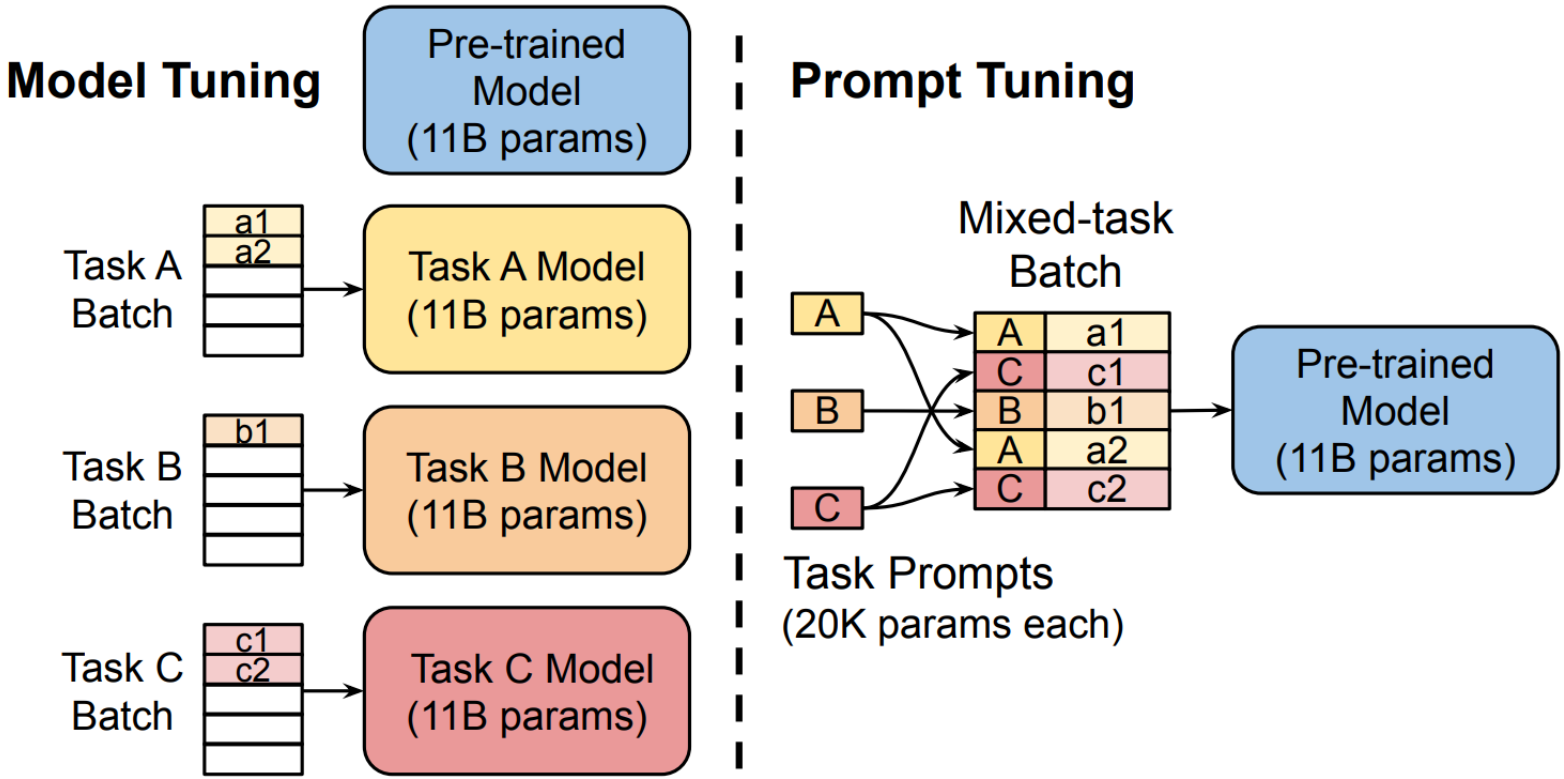
- Only learns a set continuous prefixes)added to the input and transformer layers for each task.



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

Prompt Tuning

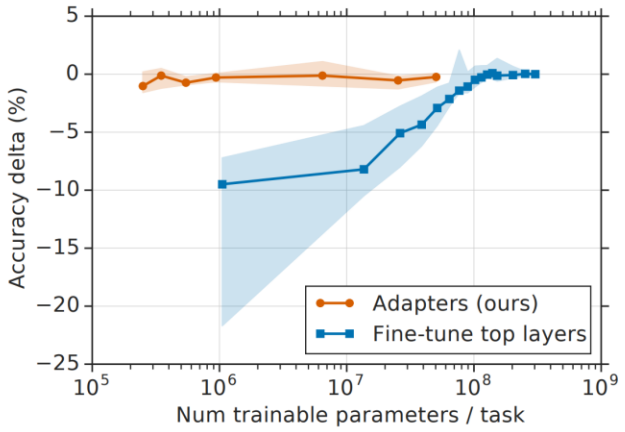
- Only learns a set of ‘prompt’ or ‘token’ for each task

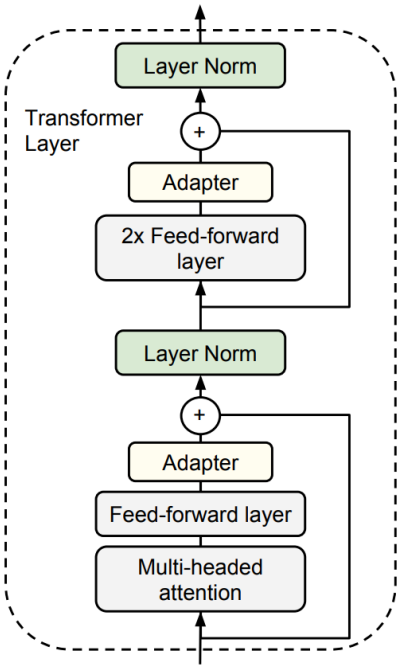


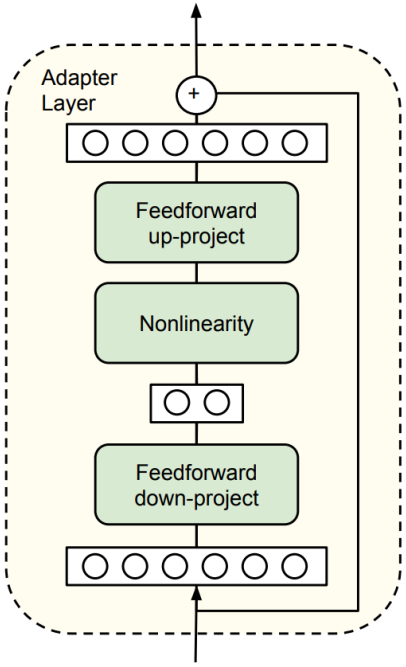
Lester et al. The Power of Scale for Parameter-Efficient Prompt Tuning. 2021.

Adapter

- Insert MLP at Feed-forward layers





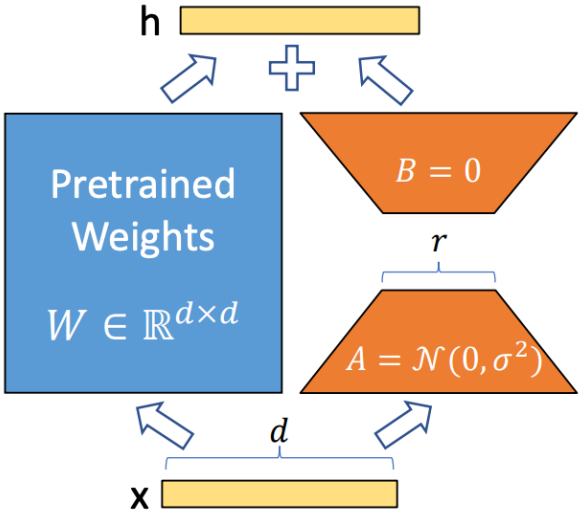


Houlsby et al. Parameter-Efficient Transfer Learning for NLP. 2022.

LoRA

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

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



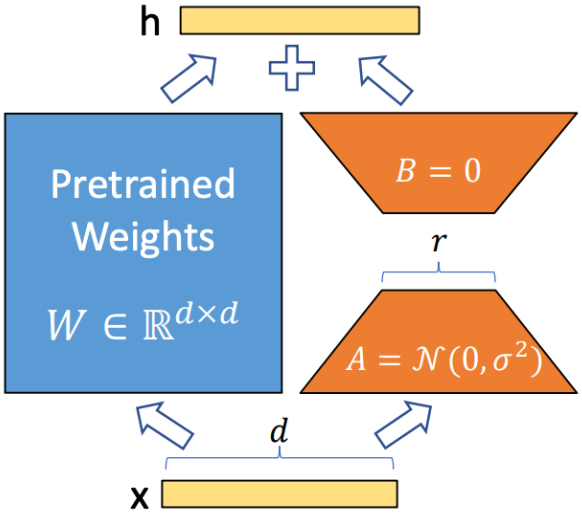
- By default, only W_Q and W_V have the AB pairs
  - Just changing which part to focus on works!

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

LoRA

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

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

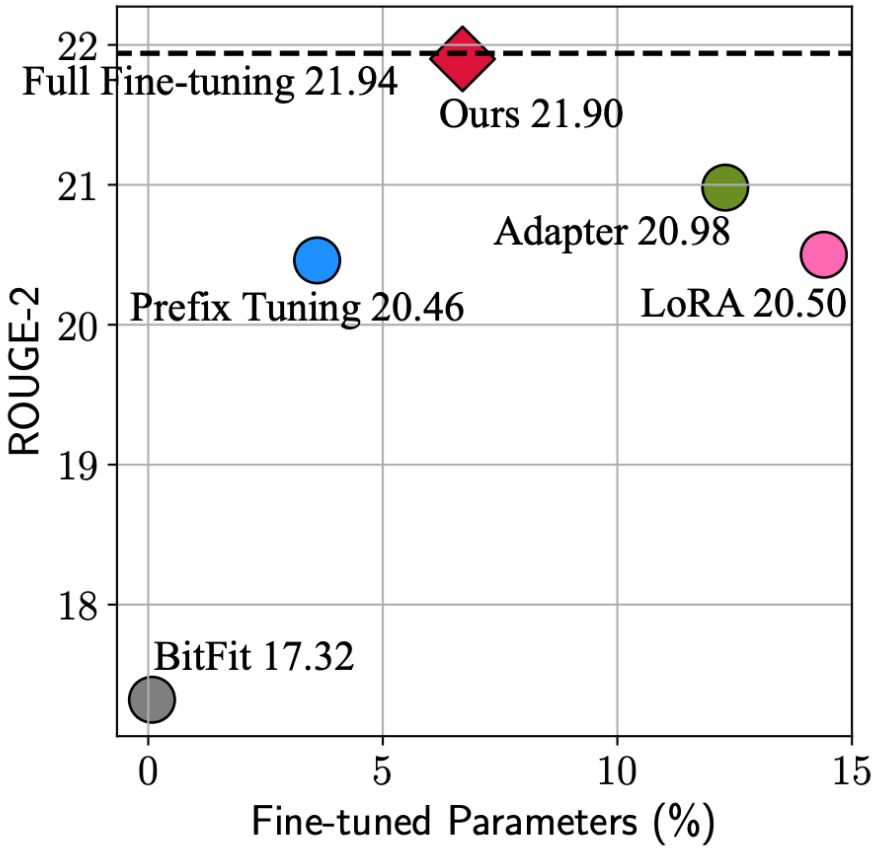


- By default, only W_Q and W_V have the AB pairs
  - Just changing which part to focus on works!
- For MLP when we want to add more knowledge

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

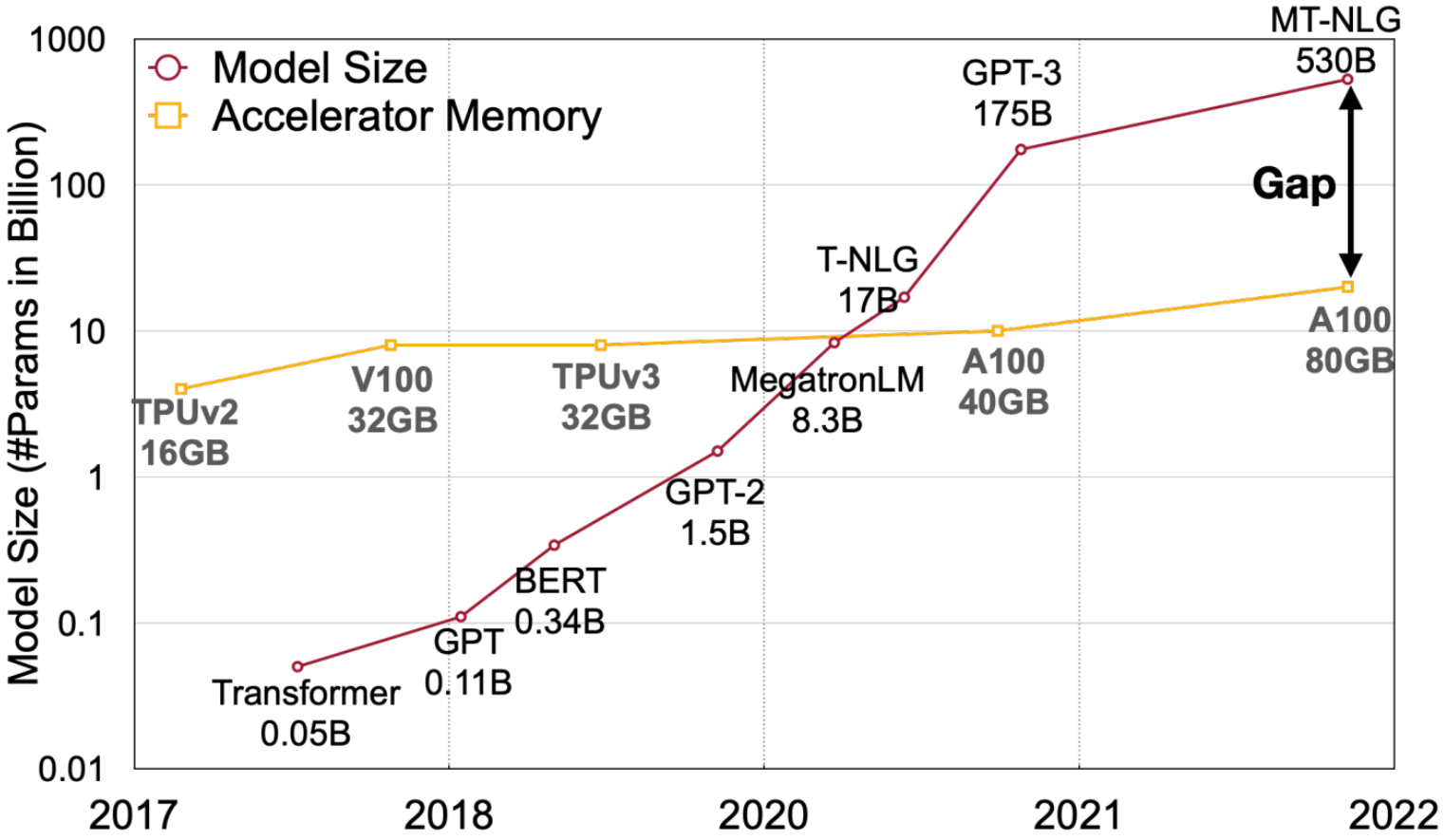


Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws
- Quantizing Transformers
- Interpreting Transformer – attention, logitlens

Scaling Laws

“Magic” of Transformer - Scaling



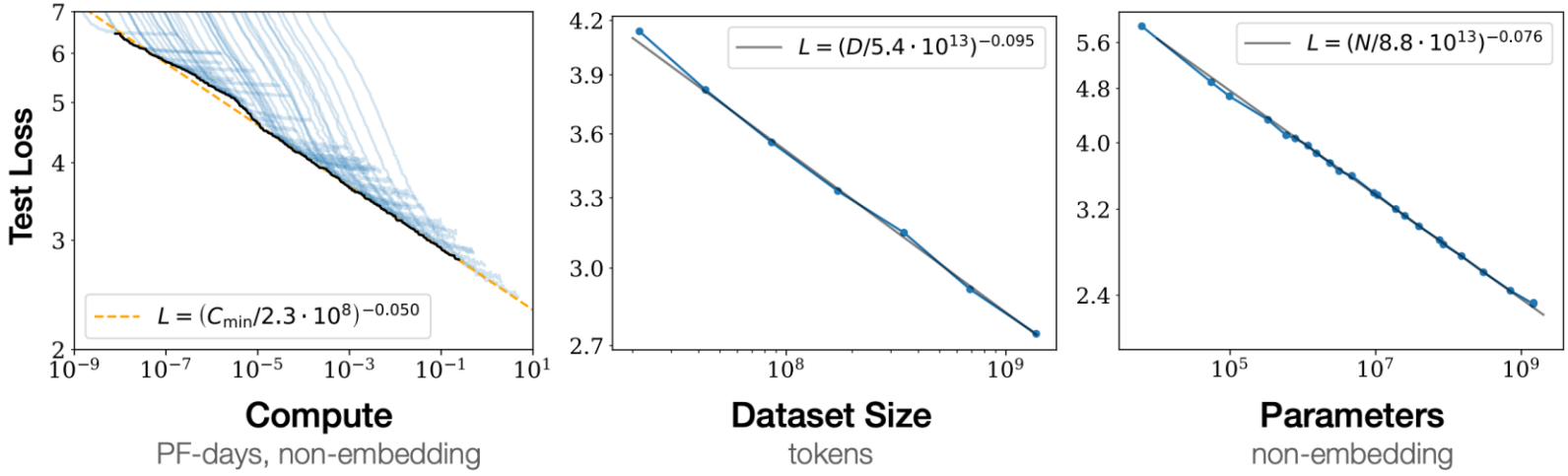
- Performance gets better as transformer scales up

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

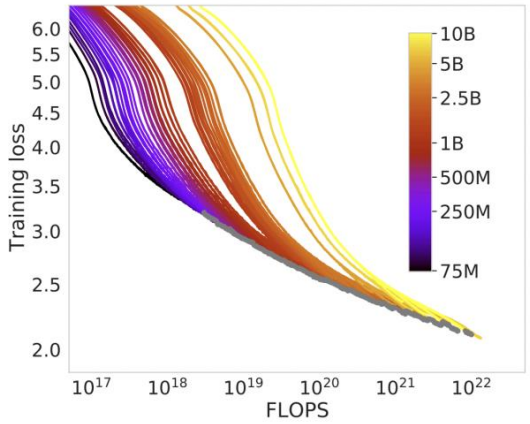
$C = \alpha DP$



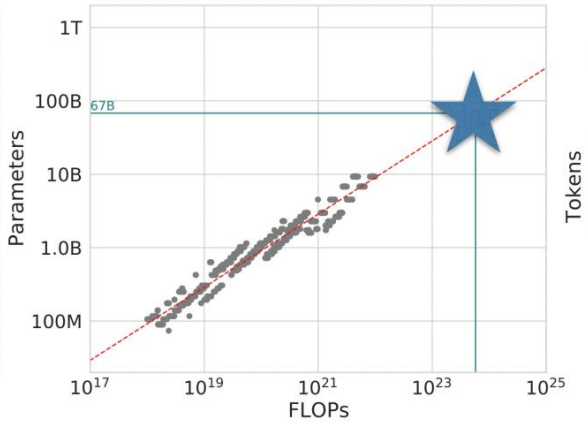
$$\text{PF-day} = 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19} \text{ floating point operations.}$$

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

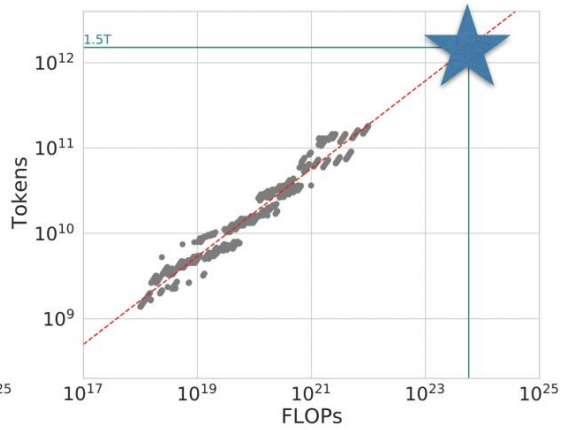
Scaling Law



Run experiments



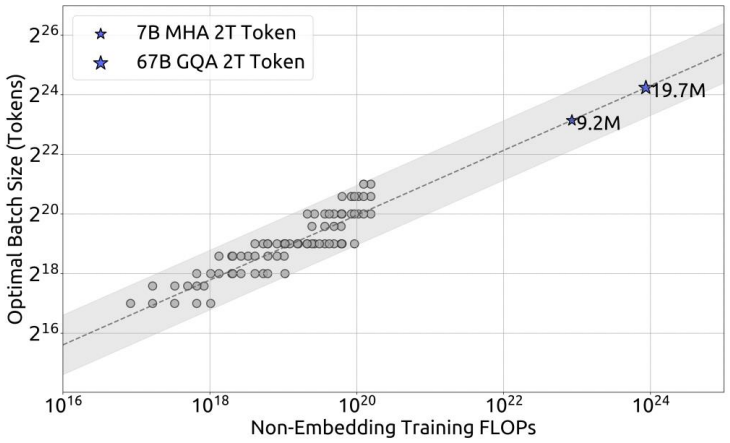
Fit a line and predict optimal model size



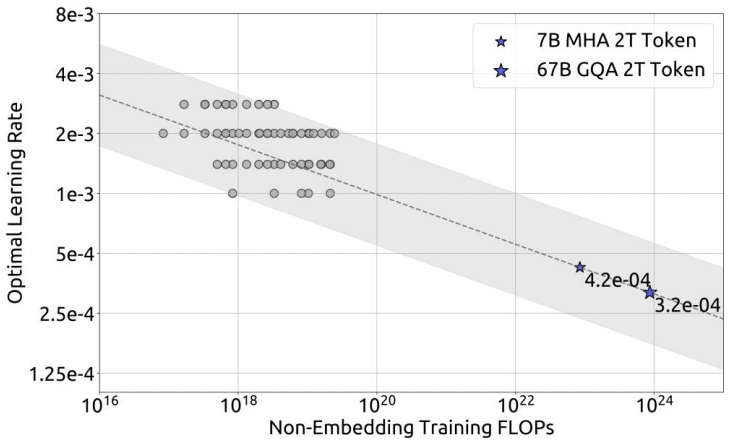
Fit a line and predict optimal # of tokens

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

Scaling Law



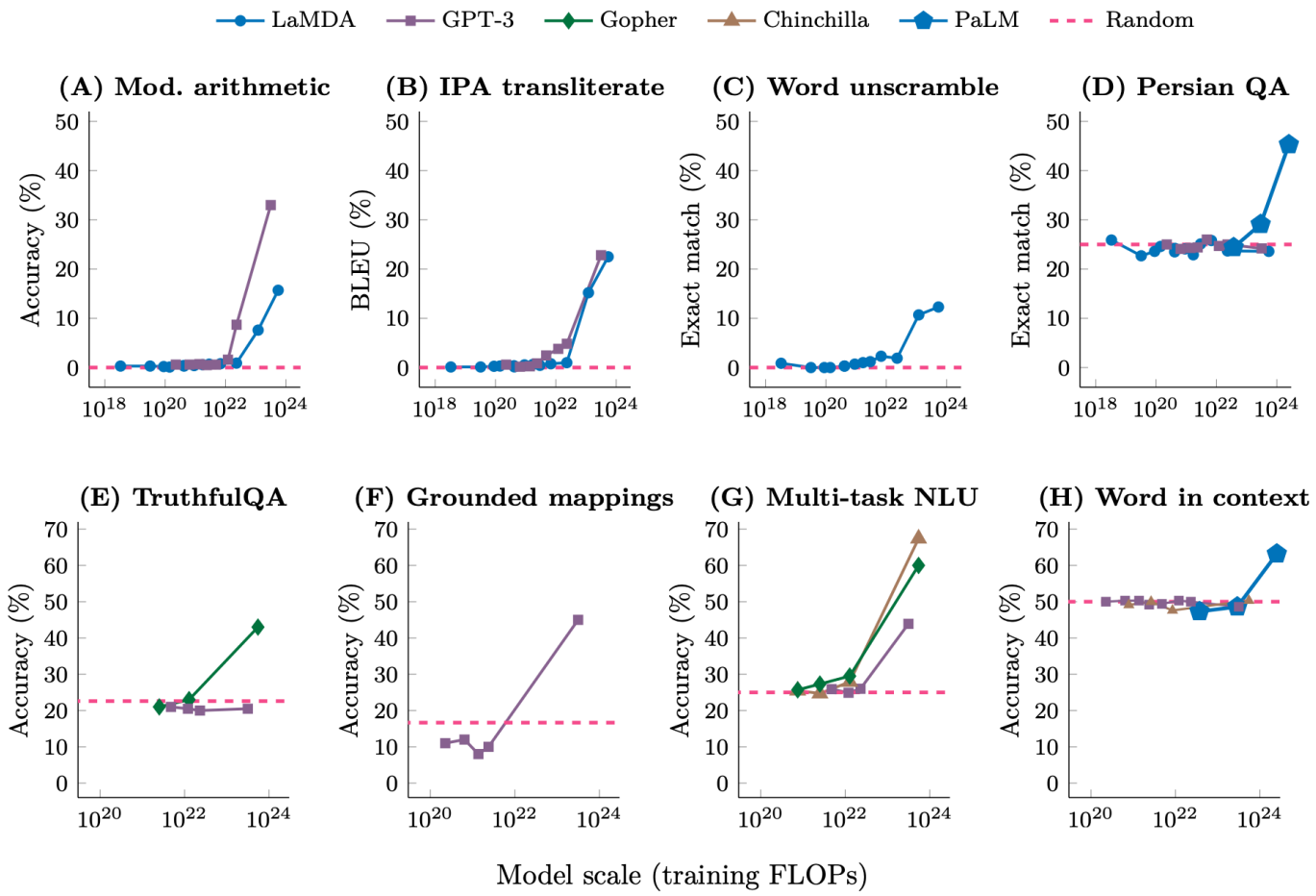
(a) Batch size scaling curve



(b) Learning rate scaling curve

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

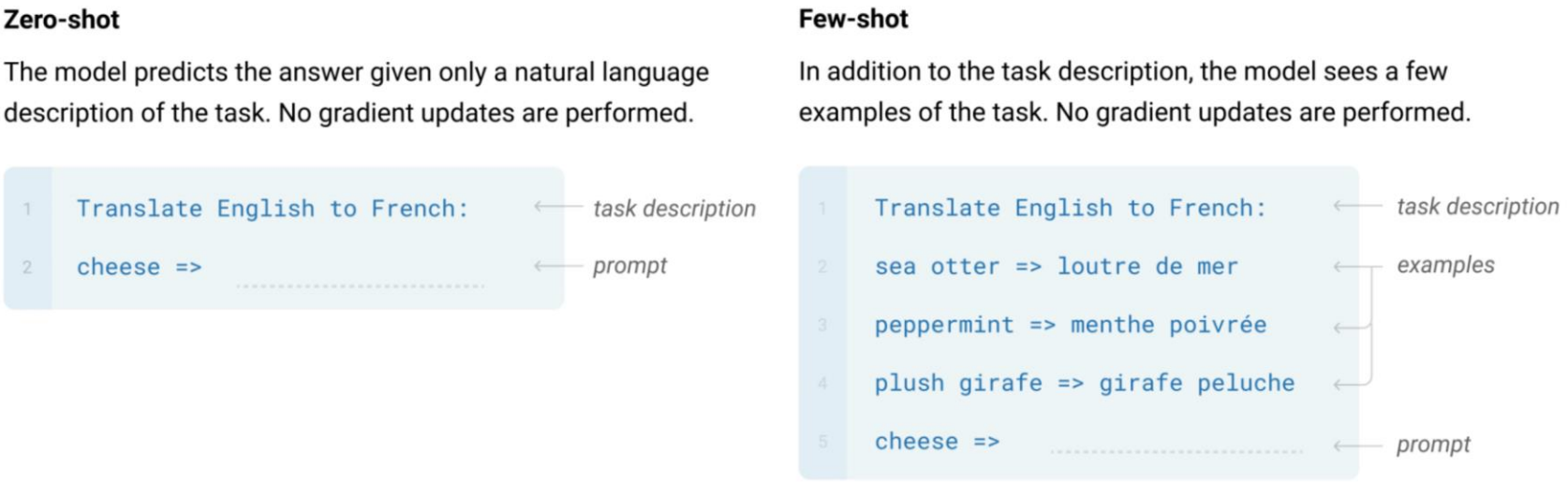
“Emergent” Capability



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.

Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws
- Quantizing Transformers
- Interpreting Transformer – attention, logitlens

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

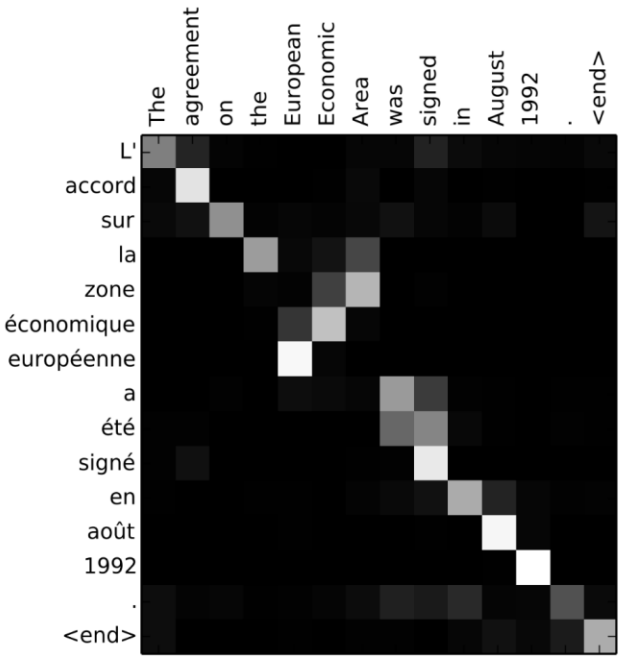
https://doi.org/10.1162/neco_a_01679; https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-quantization

Content

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws
- Quantizing Transformers
- Interpreting Transformer – attention, logitlens

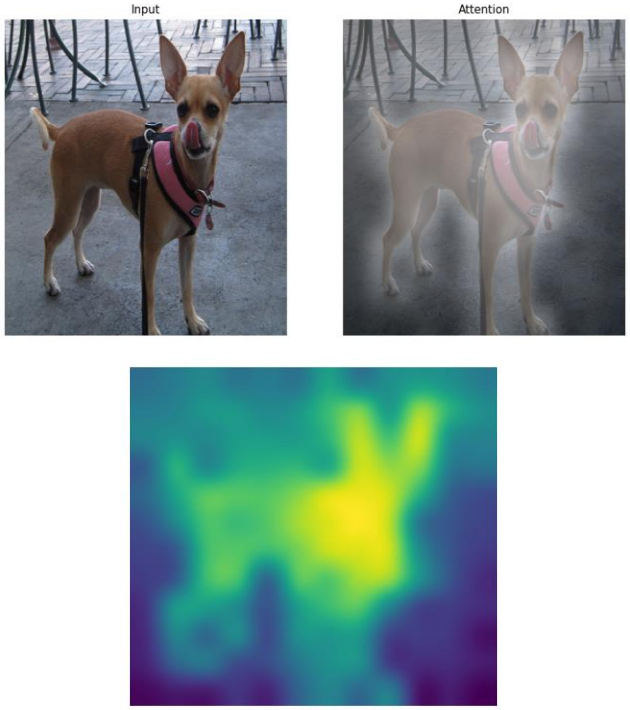
Attention Maps

- Many methods exist, common ones are:
  - Attention maps: aggregate attention weights across all heads in last layer to get avg weights for different tokens



Attention-matrix heatmap

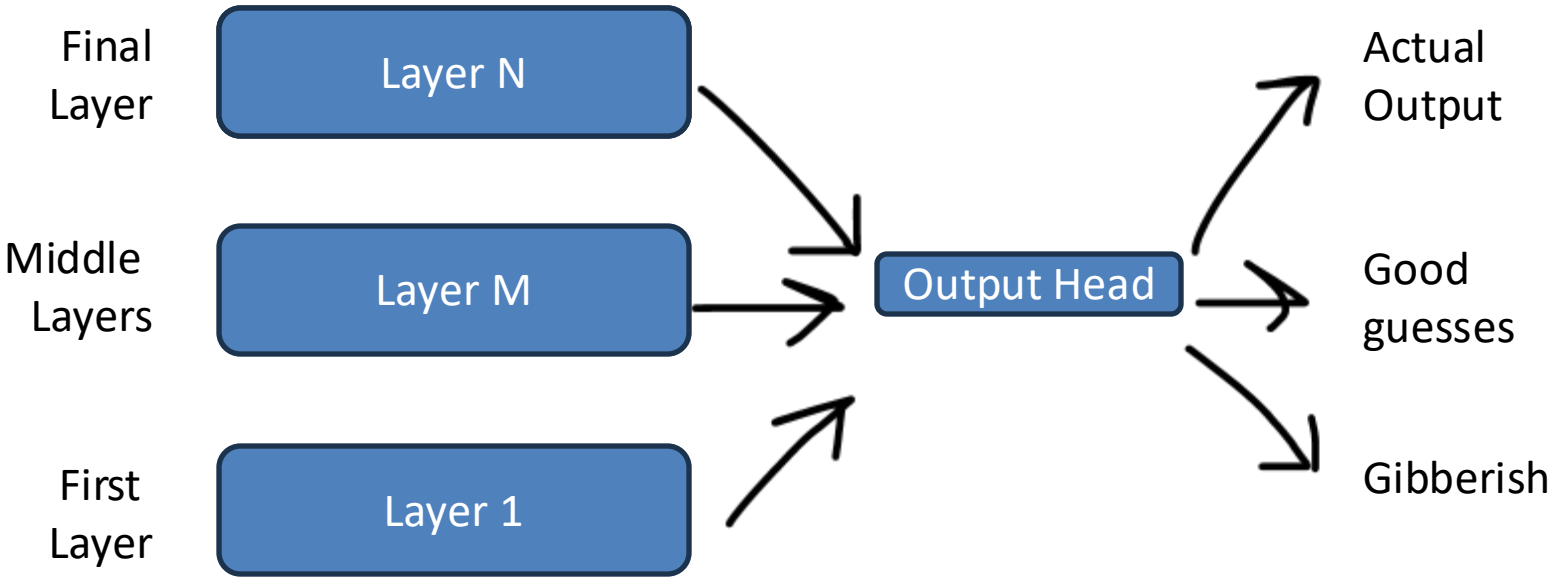
Bahdanau, et al. 2015. Neural machine translation by jointly learning to align and translate. In Proc. ICLR.



https://arxiv.org/pdf/1409.0473; https://medium.com/@nivonl/exploring-visual-attention-in-transformer-models-ab538c06083a

Logit Lens

- Many methods exist, common ones are:
  - Logit Lens: output head applied to intermediate layers



https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens

We learned…

- Transformer Architecture
- Improvements on Transformers
- Transformer for different modalities
- Parameter Efficient Tuning
- Scaling Laws
- Quantizing Transformers
- Interpreting Transformer – attention, logitlens