Lecture 03:
Transformer and Large Model

Notes

- Quiz 1 score can be found in Brightspace
- Course website: https://www.saiqianzhang.com/COURSE/
- I use Brightspace to post announcements and grades
- I provide an online zoom meeting option for people interested in auditing the class. However, enrolled students are required to attend in person unless special condition.
- Discussion groups has been created in the Brightspace
- Course email: efficientaiaccelerator@gmail.com

Recap

- Convolutional Neural Network
  - Basic building blocks
  - Popular CNN architectures
    - VGG
    - ResNet
    - MobileNet
    - ShuffleNet
    - SqueezeNet
    - DenseNet
    - EfficientNet
    - ConvNext
    - ShiftNet
  - CNN architectures for other vision tasks
    - Image Segmentation, Object Detection

Topics

- Transformer basics
- Bert
- Vision transformer
- Large Language Model
- Self-supervised learning

Transformers

- Proposed in ”Attention Is All You Need“ in 2017
- The vanilla Transformer is a sequence-to-sequence model and consists of transformer blocks.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

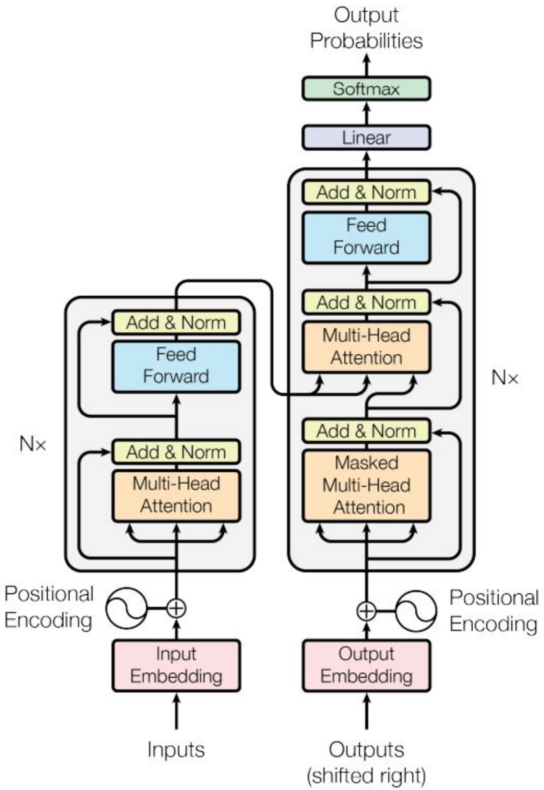
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

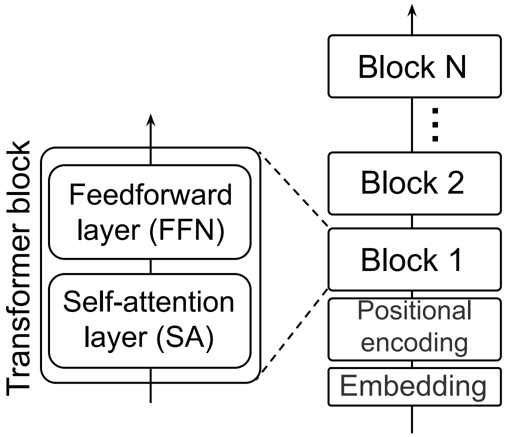
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

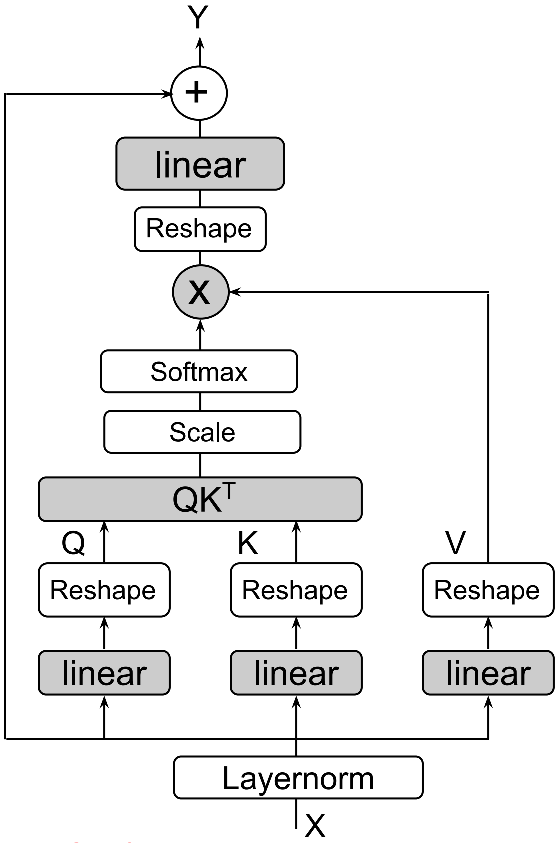
Illia Polosukhin* ‡ illia.polosukhin@gmail.com



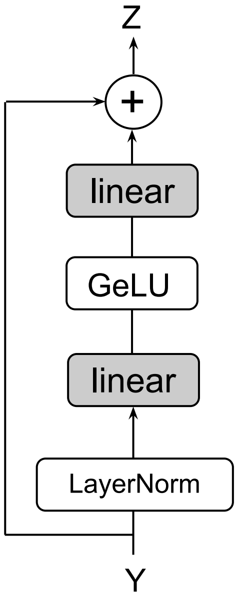
Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Transformers





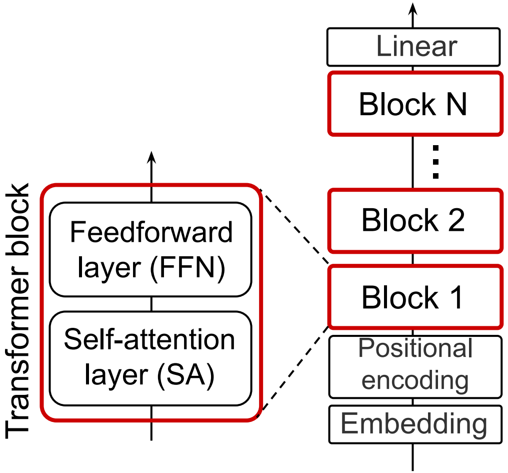
Self attention block (SA)



Feed forward block (FFN)

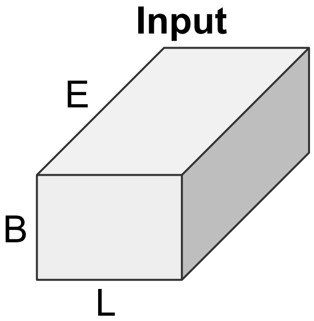
- Each transformer block includes a self-attention layer and a feedforward layer.

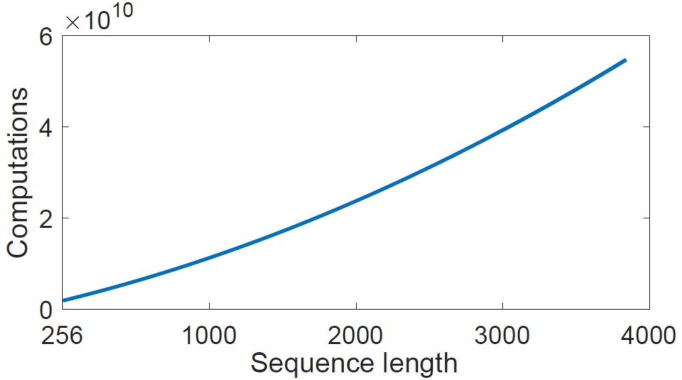
Transformers: Transformer Block



NYU SAI LAB

Transformers

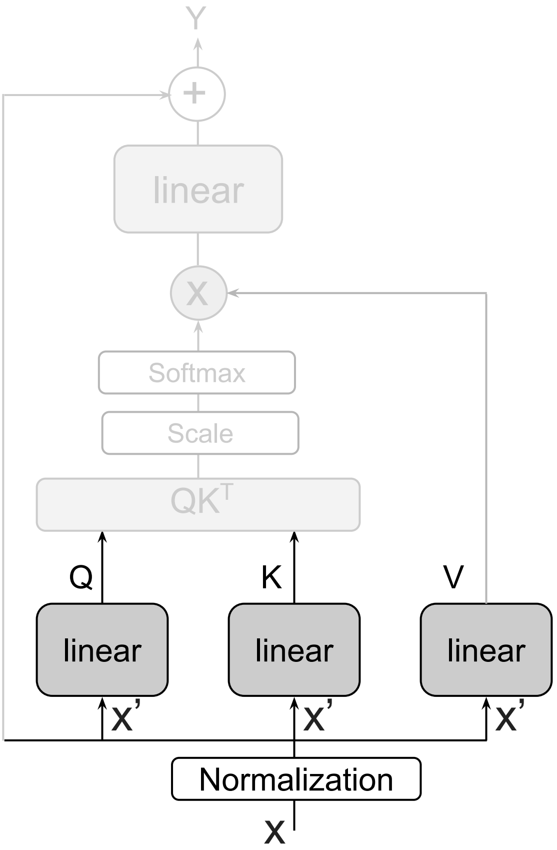




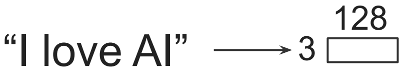
- The input contains three dimensions:
  - B: batch
  - L: token length
  - E: embeddings
- The amount of computation is closely related to the token length L.
- Longer sequences are disproportionately expensive because attention is quadratic to the sequence length.

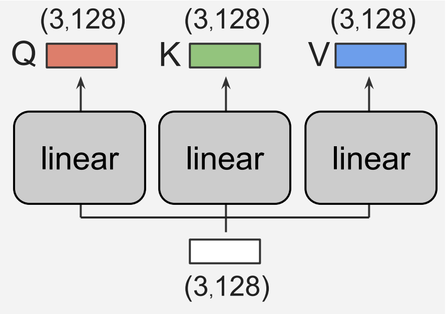
Self-Attention Block

- The input x is first normalized, then the first step in calculating self-attention is to create three vectors from the input x’, denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E) (BLE^2)
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
  - QK^T→(B, L✖E) ✖ (B,E✖L) → (B, L✖L)
- Scale and normalize the score using softmax.
  - Softmax(QK^T) → (B, L✖L)
- Multiply each value vector by the softmax score.
  - Softmax(QK^T) ✖ V
  - (B, L✖L) ✖ (B, L✖E) → (B, L✖E)
- Pass the result to the linear layer, sum with the input.



Example

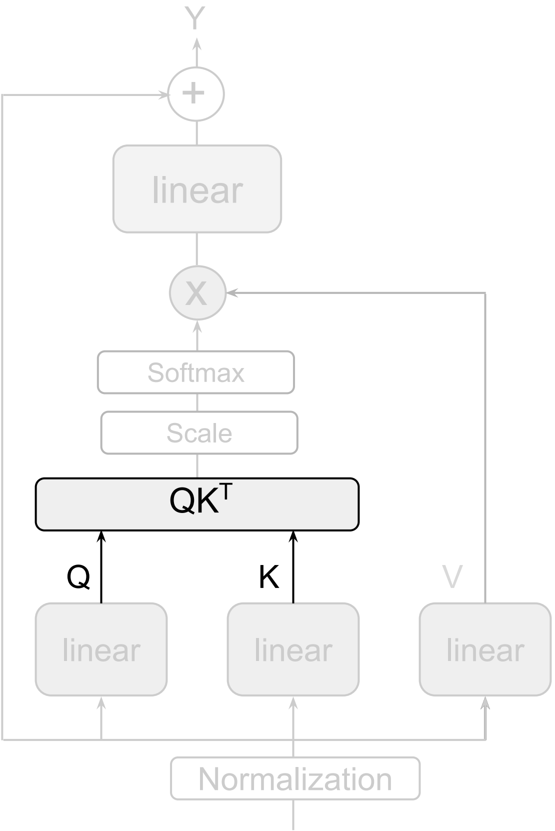




Step 1

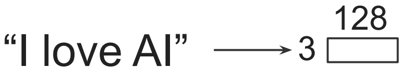
Self-Attention Block

- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
    - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
    - QK^T→(B, L✖E) ✖ (B, E✖L) → (B, L✖L) (BL^2E)
- Scale and normalize the score using softmax.
    - Softmax(QK^T) → (B, L✖L)
- Multiply each value vector by the softmax score.
    - Softmax(QK^T) ✖ V
    - (B, L✖L) ✖ (B, L✖E) → (B, L✖E)
- Pass the result to the linear layer, sum with the input.

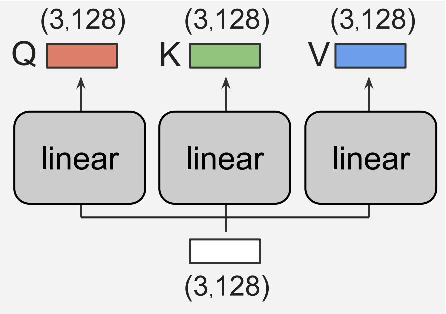


NYU SAI LAB

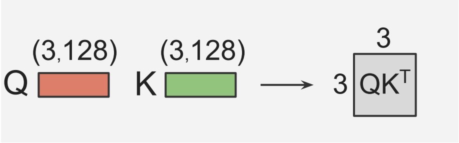
Example



Step 1

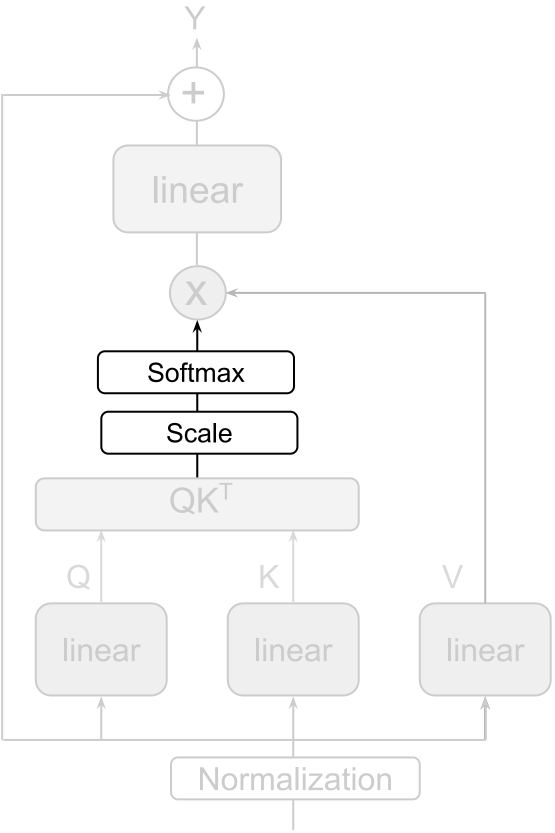


Step 2

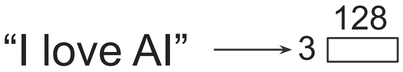


Self-Attention Block

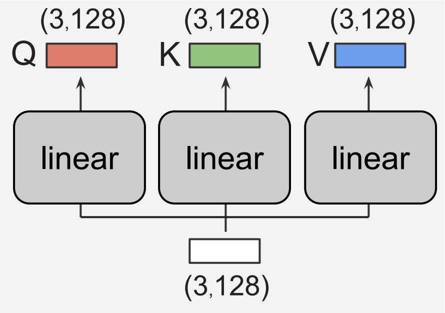
- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
  - QK^T→(B, L✖E) ✖ (B,E✖L) → (B, L✖L)
- Scale and normalize the score using softmax.
  - Softmax(QK^T) → (B, L✖L)
- Multiply each value vector by the softmax score.
  - Softmax(QK^T) ✖ V
  - (B, L✖L) ✖ (B, L✖E) → (B, L✖E)
- Pass the result to the linear layer, sum with the input.



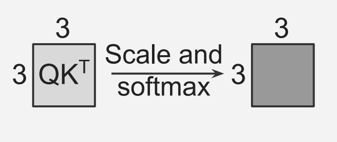
Example



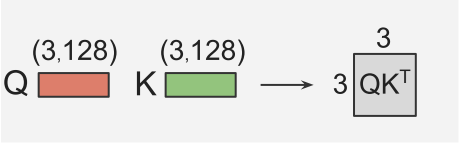
Step 1



Step 3

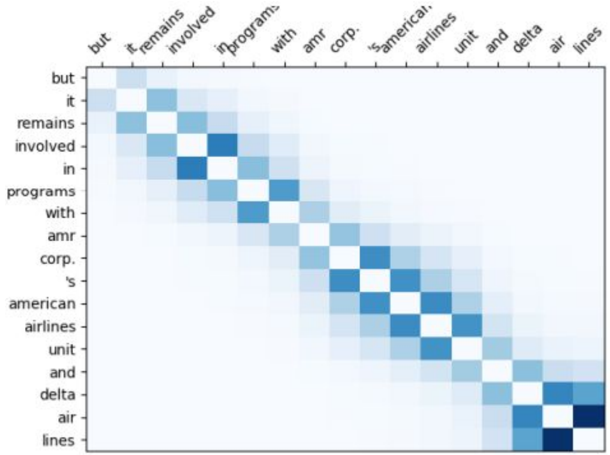


Step 2



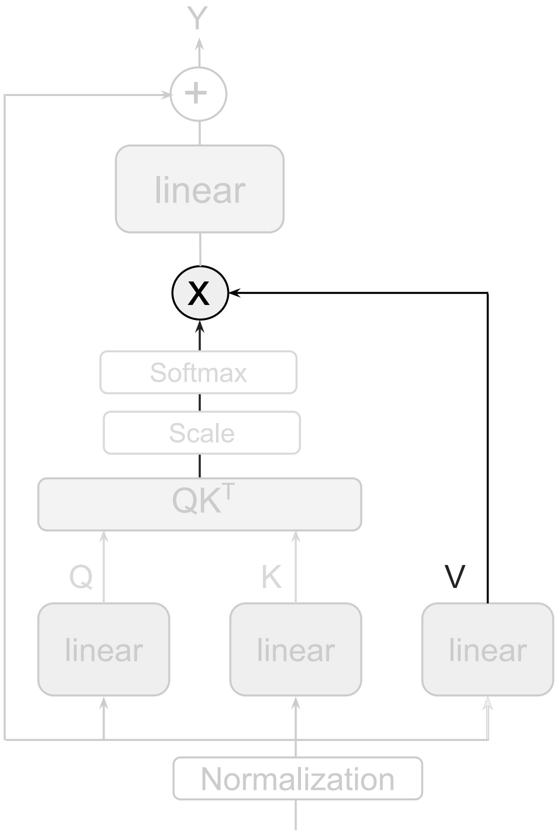
Self-Attention Block

- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
  - $QK^{\top}$→(B, L✖E) ✖ (B,E✖L) → (B, L✖L)
- Scale and normalize the score using softmax.
  - Softmax($QK^{\top}$) → (B, L✖L)
- Multiply each value vector by the softmax score.
  - Softmax($QK^{\top}$) ✖ V
  - (B, L✖L) ✖ (B, L✖E) → (B, L✖E)
- Pass the result to the linear layer, sum with the input.



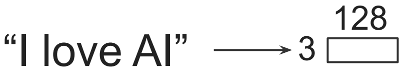
Self-Attention Block

- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
  - QK^T→(B, L✖E) ✖ (B,E✖L) → (B, L✖L)
- Scale and normalize the score using softmax.
  - Softmax(QK^T) → (B, L✖L)
- Multiply each value vector by the softmax score.
  - Softmax(QK^T) ✖ V
  - (B, L✖L) ✖ (B, L✖E) → (B, L✖E) (BL^2E)
- Pass the result to the linear layer, sum with the input.

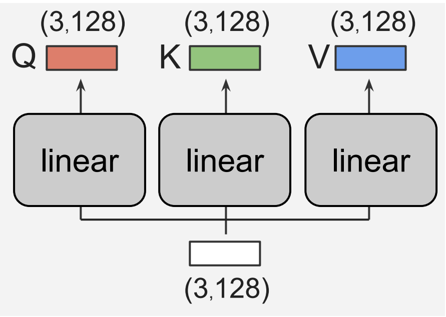


NYU SAI LAB

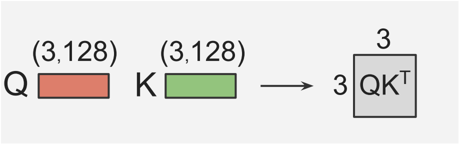
Example



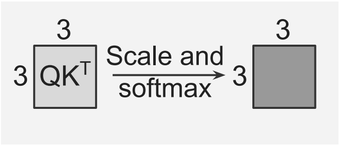
Step 1



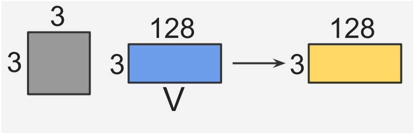
Step 2



Step 3

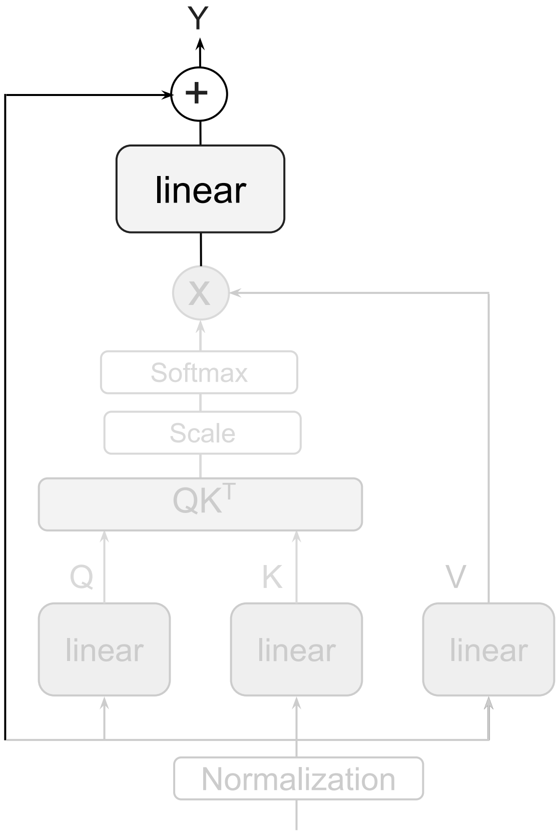


Step 4



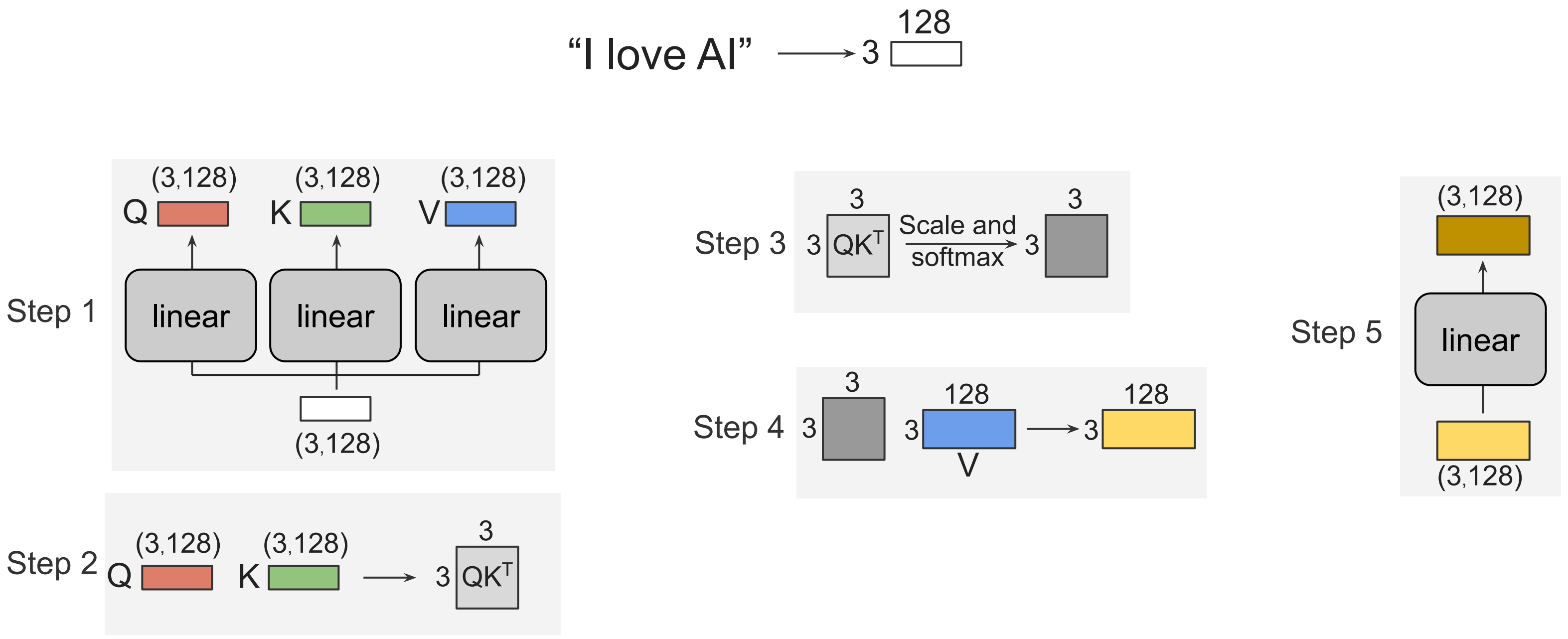
Self-Attention Block

- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input tokens.
  - QK^T→(B, L✖E) ✖ (B,E✖L) → (B, L✖L)
- Scale and normalize the score using softmax.
  - Softmax(QK^T) → (B, L✖L)
- Multiply each value vector by the softmax score.
  - Softmax(QK^T) ✖ V
  - (B, L✖L) ✖ (B, L✖E) → (B, L✖E)
- Pass the result to the linear layer, sum with the input.



NYU SAI LAB

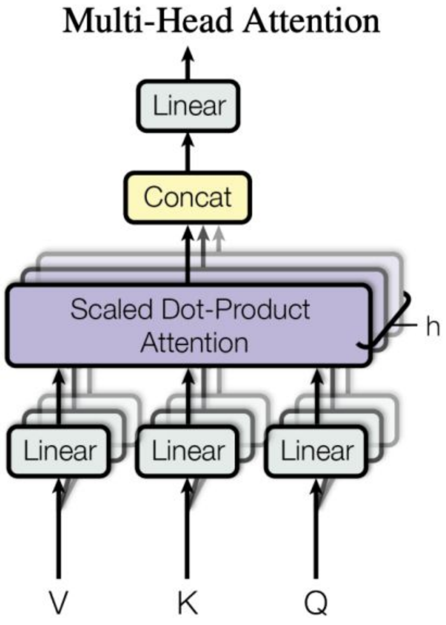
Example

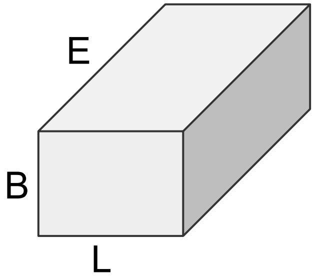


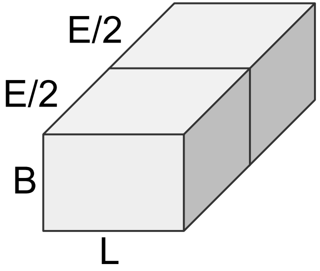
"I love AI"
3
128
Step 1
(3,128)
Q
K
V
linear
linear
linear
(3,128)
Step 2
(3,128)
(3,128)
Q
K
3
3 QK^T
Step 3
3
3 QK^T
Scale and softmax
3
3
Step 4
3
3
128
3
V
128
3
Step 5
(3,128)
linear
(3,128)

Multi-headed Attention

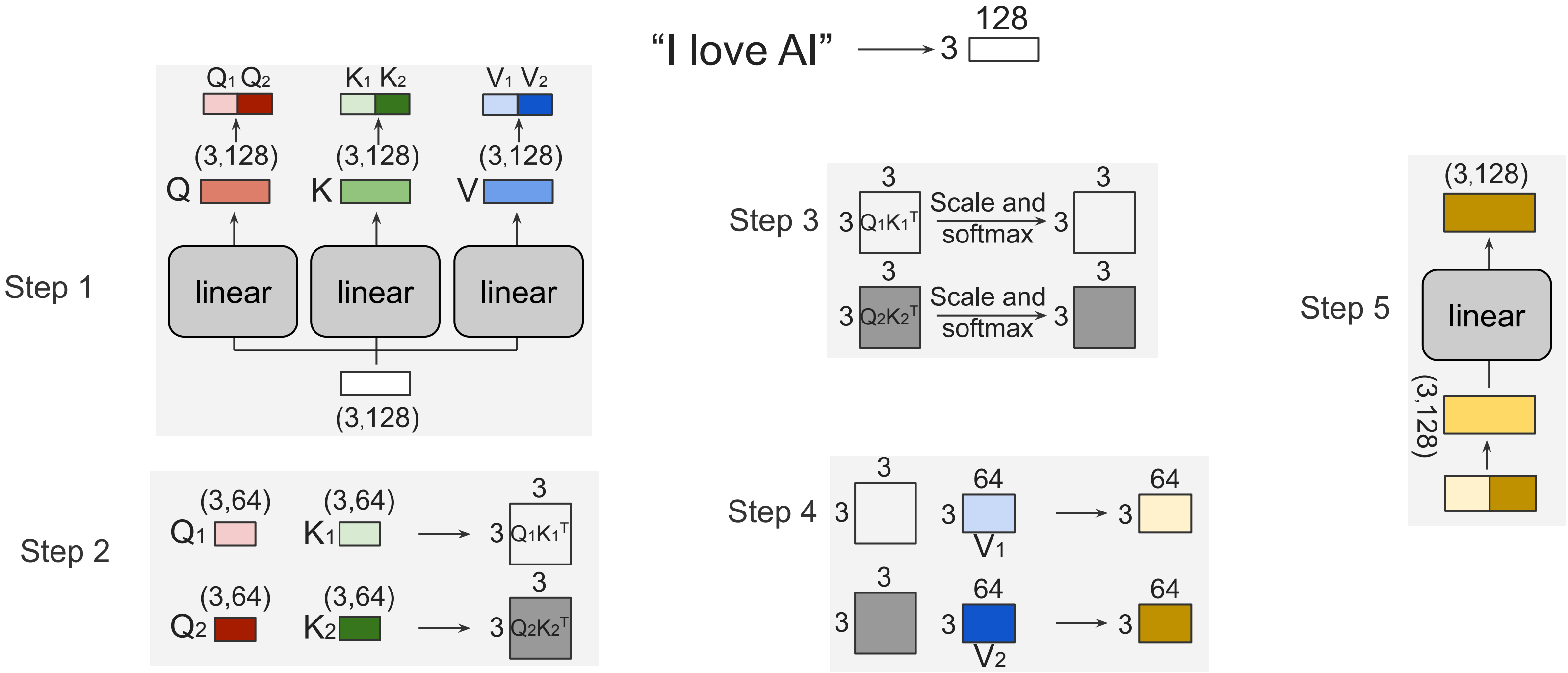
- Q, K, V tensors are broken into multiple components along the embedding dimension.
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
  - (B,L,E) → (B, M, L, E/M) → (B, M , L, D) , where D=E/M
- All the following operations can be performed independently over each head M.
  - QK^{T}→(B, M, L✖D) ✖ (B, M, D✖L) → (B, M, L✖L)
  - Softmax(QK^{T}) → (B, M, L✖L)
  - Softmax(QK^{T}) ✖ V → (B, M, L✖L) ✖ (B, M, L✖D) → (B, M, L✖D) → (B✖L✖E)





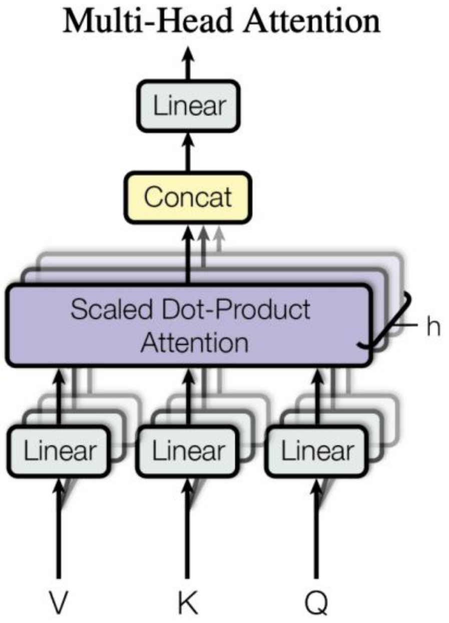


Example


"I love AI"
3
128
Step 1
Q1 Q2
K1 K2
V1 V2
(3,128)
(3,128)
(3,128)
Q
K
V
linear
linear
linear
(3,128)
Step 2
(3,64)
(3,64)
Q1
K1
3
3 Q1K1T
(3,64)
(3,64)
Q2
K2
3
3 Q2K2T
Step 3
3
3 Q1K1T
Scale and softmax
3
3
3
3 Q2K2T
Scale and softmax
3
3
Step 4
3
3
64
3
V1
64
3
3
3
64
3
V2
64
3
Step 5
(3,128)
linear
(3,128)

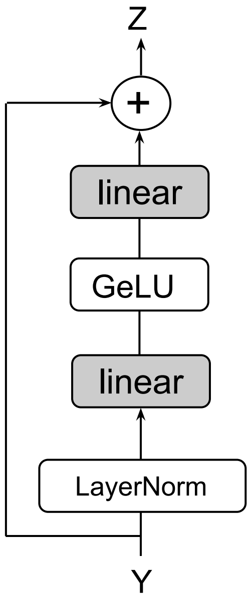
Multi-headed Attention

- Why we need multiple heads?
  - Multiple attention heads in transformers are used to enhance the expressive power and modeling capabilities of the network.
  - By using multiple attention heads, transformers can capture different types of dependencies and relationships between words or elements in a sequence.
  - Having multiple heads allows the model to perform attention calculations in parallel, which can improve computational efficiency.



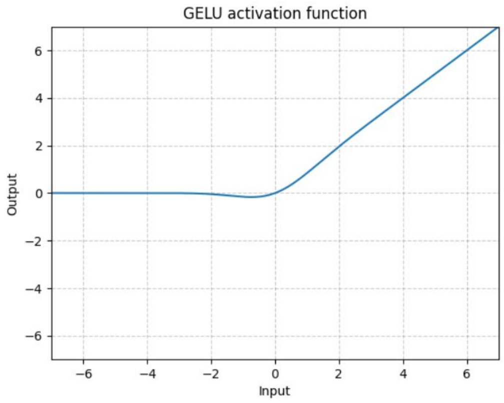
Michel, Paul, Omer Levy, and Graham Neubig. "Are sixteen heads really better than one?." *Advances in neural information processing systems* 32 (2019).

Feed Forward Layer



- The two linear layers are big:
  - (Ex4E) and (4ExE), E can be large (e.g., 4096)
  - This is expensive to implement.
- GeLU:
  - $GeLU(x) = x\Phi(x)$

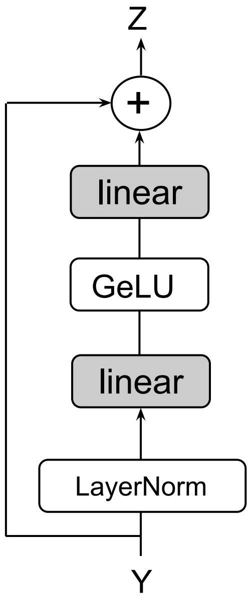
    $\Phi(x) = P(y \leq x), \text{ where Y} \sim N(0,1)$

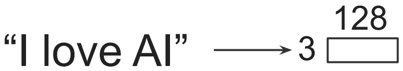


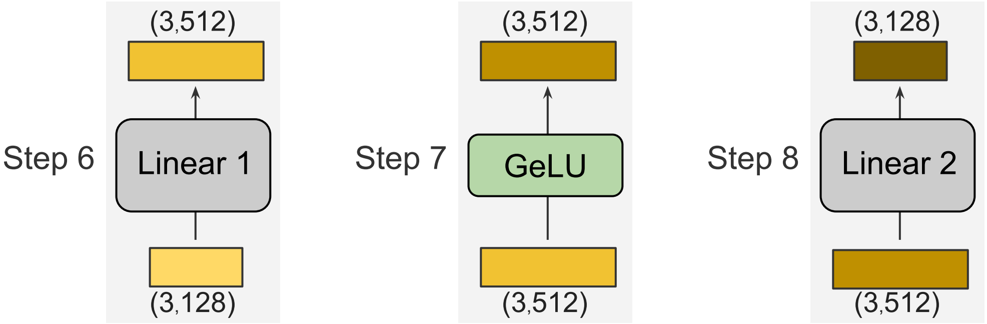
NYU SAI LAB

Hendrycks, Dan, and Kevin Gimpel. "Gaussian error linear units (gelus)." *arXiv preprint arXiv:1606.08415* (2016).

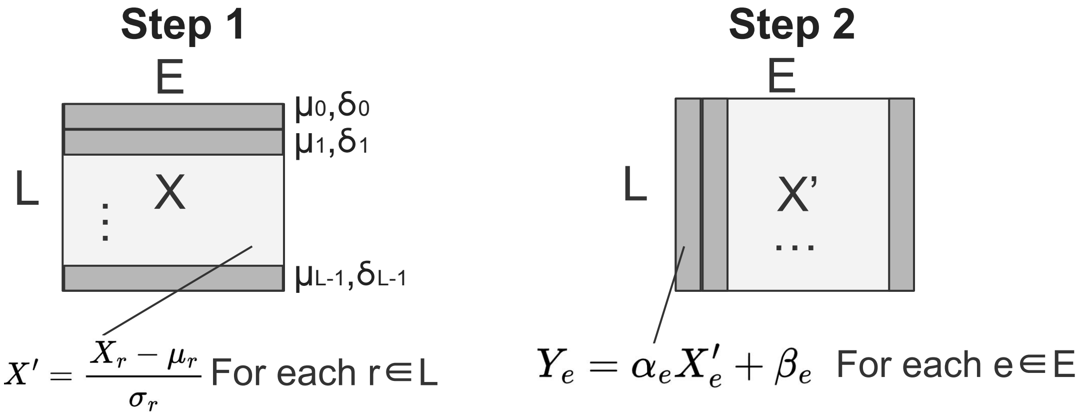
Example


Z
+
linear
GeLU
linear
LayerNorm
Y


"I love AI"
3
128


Step 6
(3,512)
Linear 1
(3,128)
Step 7
(3,512)
GeLU
(3,512)
Step 8
(3,128)
Linear 2
(3,512)

Layer Normalization



$$X' = \frac{X_r - \mu_r}{\sigma_r}$$ For each r∈L

$$Y_e = \alpha_e X'_e + \beta_e$$ For each e∈E

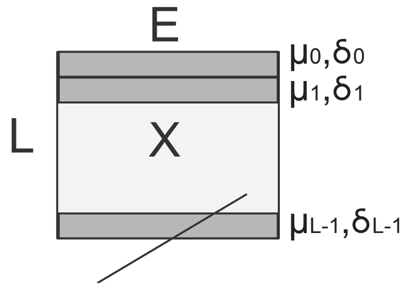
- LayerNorm is applied on each input sample.
- Both α and β have a length of E.

NYU SAI LAB

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." *arXiv preprint arXiv:1607.06450* (2016).

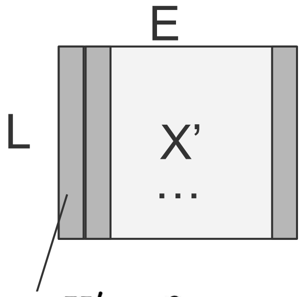
Layer Normalization

Step 1



$X' = \frac{X_r - \mu_r}{\sigma_r}$ For each r∈L

Step 2



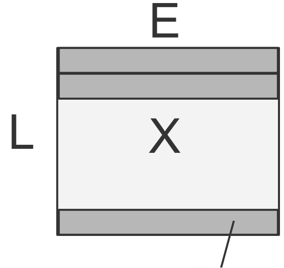
$Y_e = \alpha_e X'_e + \beta_e$ For each e∈E

- Layer Norm does not store the running mean and running variance, so during the inference time, the mean and variance need to be computed.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." *arXiv preprint arXiv:1607.06450* (2016).

RMS Normalization

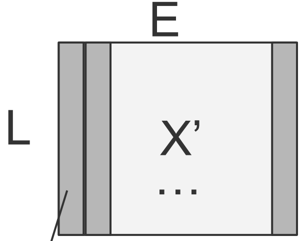
Step 1



$$X'_r = \frac{X_r}{\sqrt{\frac{\sum_i X_{r,i}^2}{L}}}$$

For each r∈L

Step 2



$$Y_e = \alpha_e X'_e$$

For each e∈E

- The experiments demonstrate RMSNorm achieves similar and even better results than LayerNorm.

Zhang, Biao, and Rico Sennrich. "Root mean square layer normalization." *Advances in Neural Information Processing Systems* 32 (2019).

Transpose & Reshape

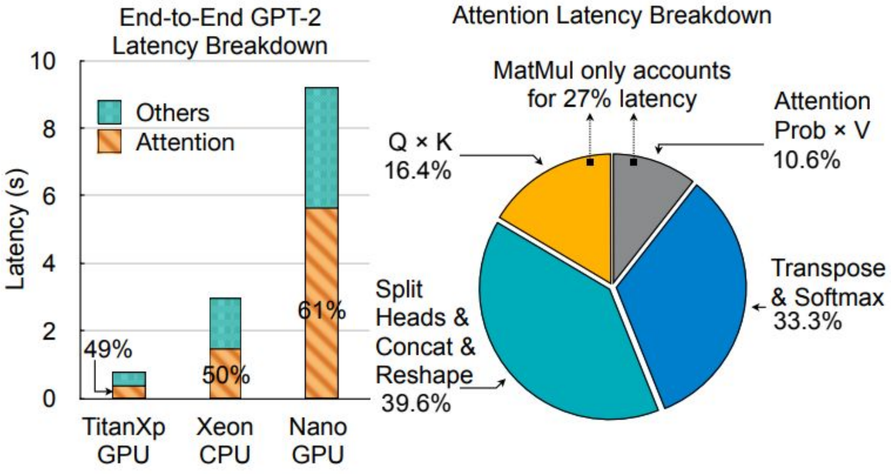


Fig. 2. End-to-End GPT-2 latency breakdown on various platforms, and attention latency breakdown on TITAN Xp GPU. Attention accounts for over 50% of total latency. Data movements account for 73% of attention latency.

Reshaping operation

$$(B, L, E) \xrightarrow{\quad\quad} (B, M, L, E/M)$$

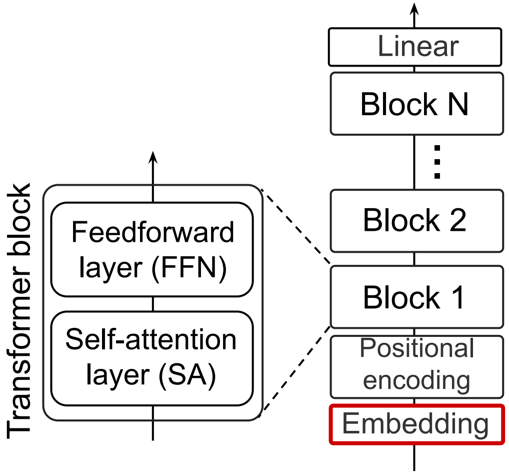
Transpose operation

$$(B, L, E) \xrightarrow{\quad\quad} (B, E, L)$$

NYU SAI LAB

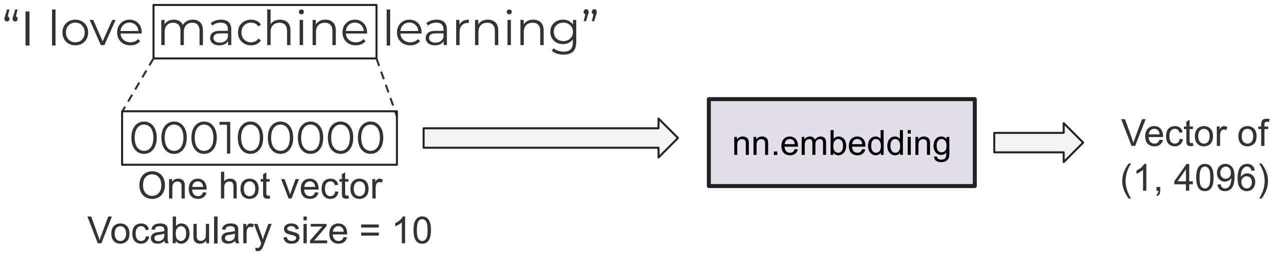
Wang, Hanrui, Zhekai Zhang, and Song Han. "Spatten: Efficient sparse attention architecture with cascade token and head pruning." *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021.

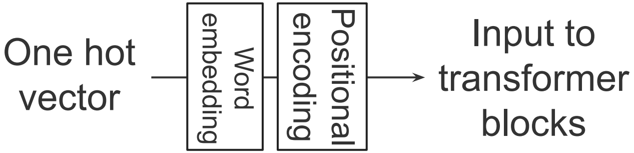
Transformers: Word Embedding



Transformers: Word Embedding

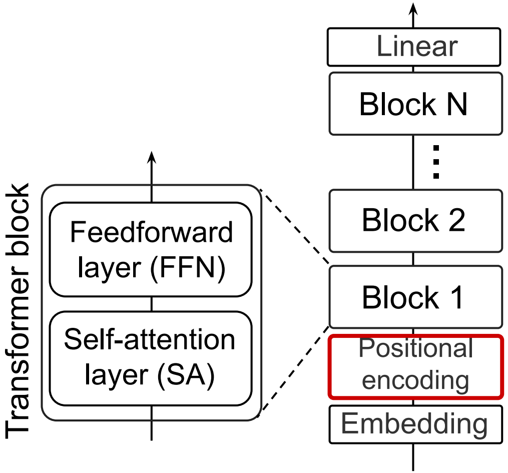
- For each word, we can convert them into a one-hot vector.
- We use the embedding layer to encode these one-hot vectors, which acts like a trainable lookup table.
- Dictionary: {are, is, machine, good, awesome, learning, love, ….}





NYU SAI LAB Mikolov, Tomas. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

Transformers: Positional Encoding

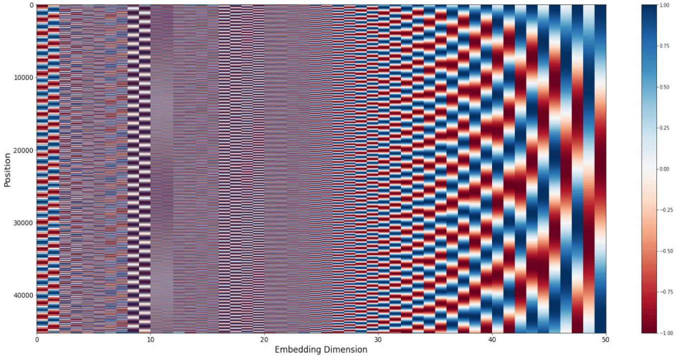


Transformers: Positional Encoding

- One thing that’s missing from the model as we have described it so far is a way to account for the order of the words in the input sequence.
- Transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

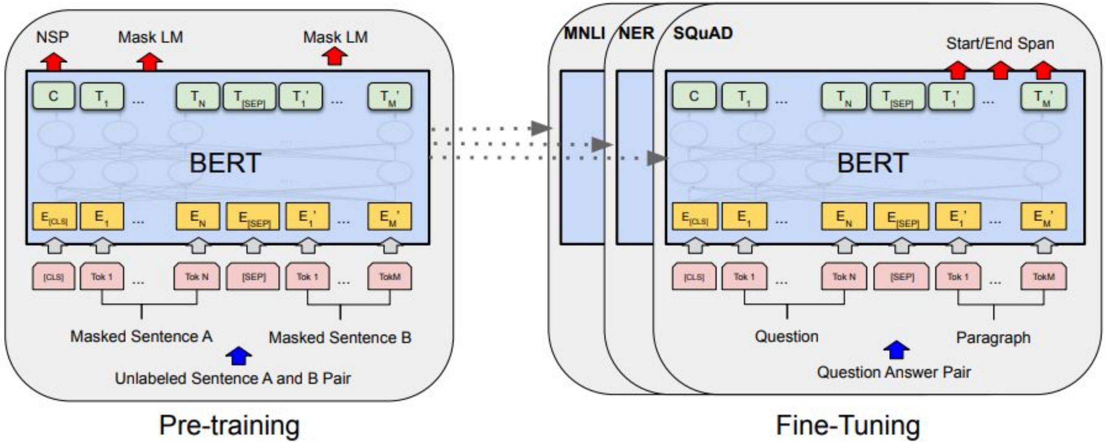
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



- pos is the positional of the token, i is the index of the embedding.

Case Study: BERT

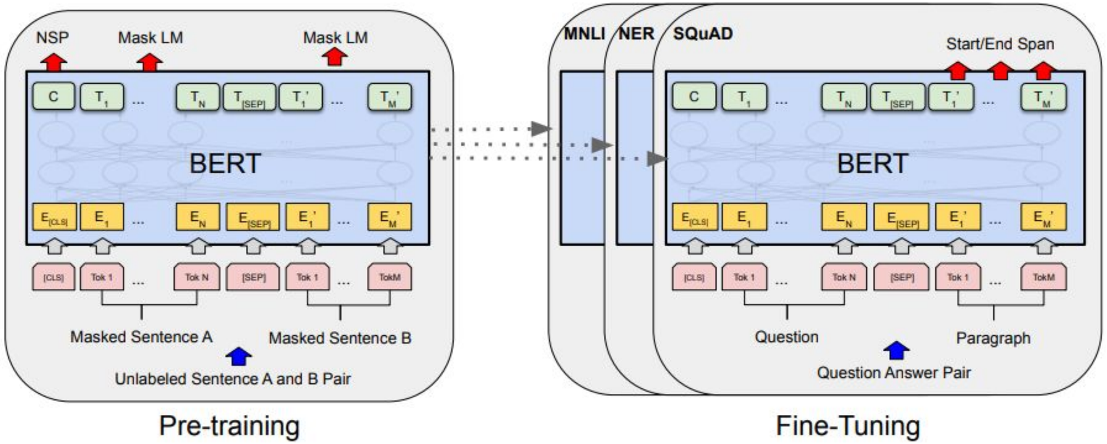
- Bidirectional Encoder Representations from Transformers.
- BERT is designed to understand the text by considering both the words before and after it.
- BERT consists of transformer encoders and takes the entire sentence as the input.
- BERT can not generate new text.



Devlin, Jacob. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

Case Study: BERT

- Use the encoder’s output embeddings as input features for downstream tasks.
- Represent a sentence as a vector for semantic similarity, clustering, or search.
- The pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks.

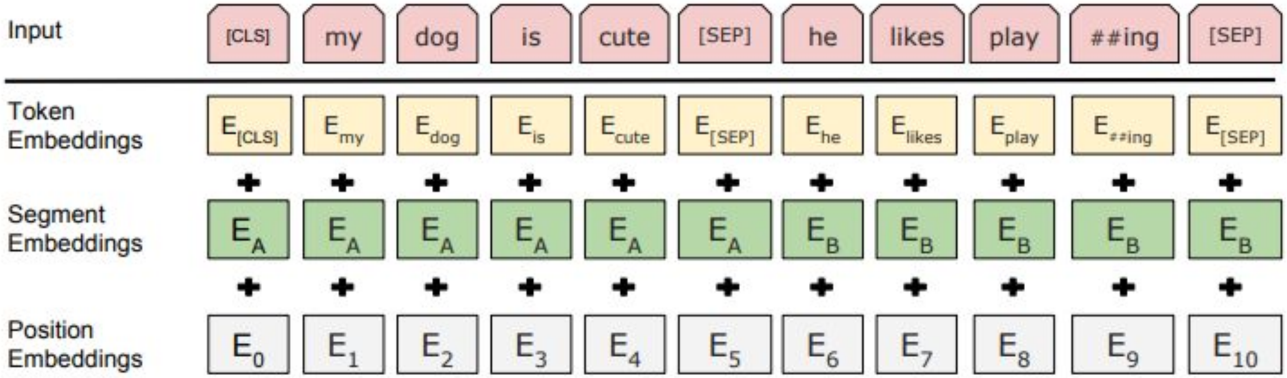


NYU SAI LAB

Devlin, Jacob. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

BERT

- A [CLS] token is inserted at the start of every sequence, and the two sentences in the sequence are separated by a [SEP] token.
- The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.
- In addition to the positional information, BERT contains a segment embeddings to differentiate the sentences.

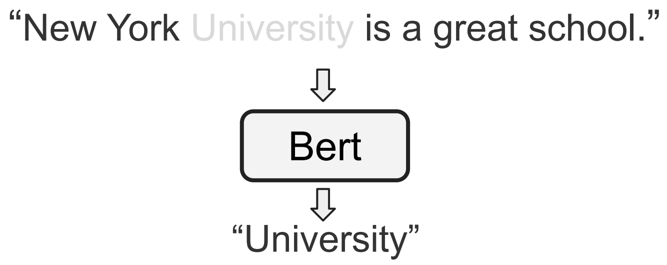


Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\#\#ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

BERT Pretraining

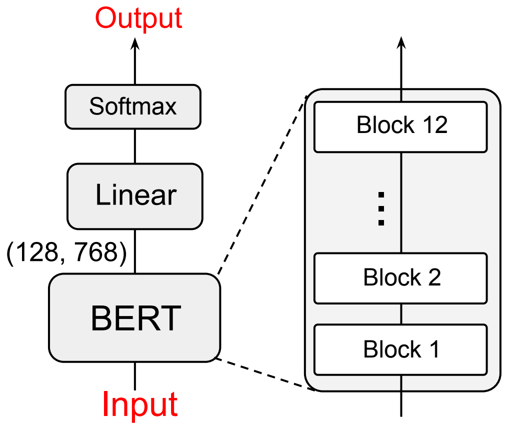
- BERT is pretrained using two unsupervised tasks:
  - Masked Language Modeling (MLM)
    - We simply mask some percentage of the input tokens at random, and then predict those masked tokens.
  - Next Sentence Prediction (NSP)
    - Given two sentences, A and B, predict whether B is A’s following sentence.

“New York University is a great school.”



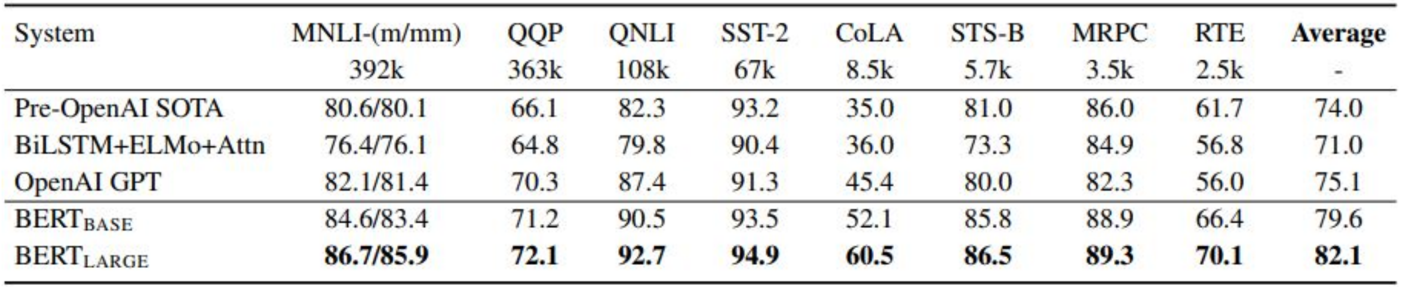
Downstream Tasks

- For text classification task, Bert will return a binary output.
  - Single sentence task (SST-2): The task is to predict the sentiment of a given sentence.
- The input may contain a single sentence, or a pair of sentences.
  - Similarity and Paraphrase tasks (MRPC): Is the second sentence a paraphrase of the first sentence.



NYU SAI LAB

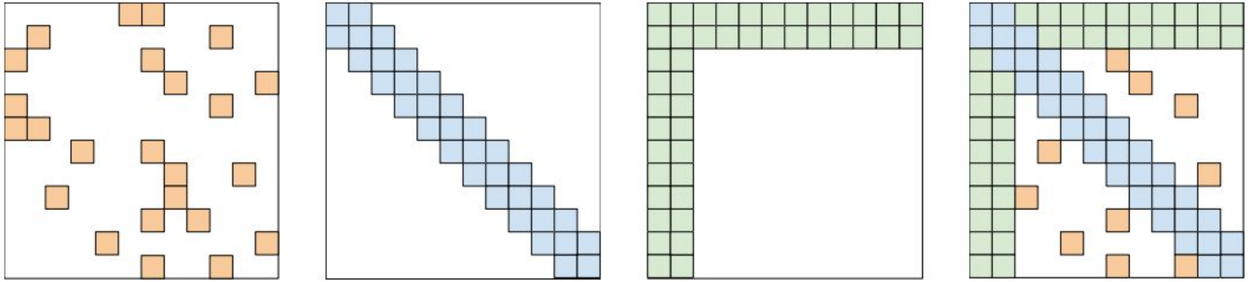
BERT Performance



System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

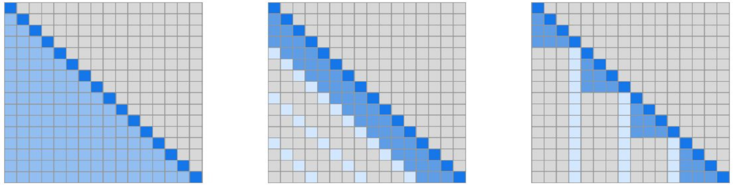
- BERT can achieve better performance over GLUE datasets than GPT-1.

Efficient Self-Attention Design



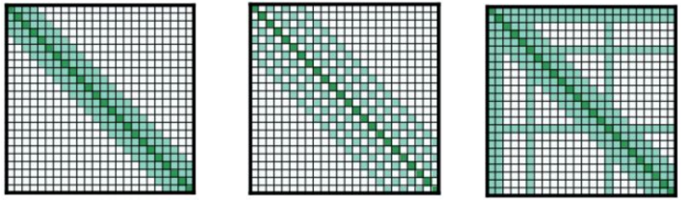
(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Sparse Transformer (2019)



Original Strided Fixed

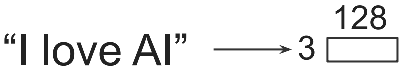
Longformer (2020)



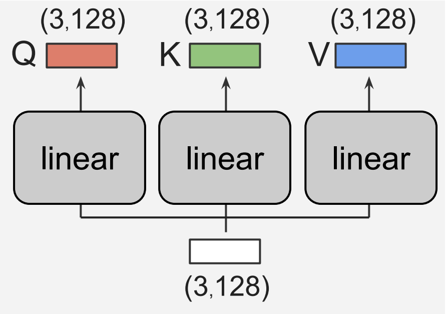
Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).

Child, Rewon, et al. "Generating long sequences with sparse transformers." *arXiv preprint arXiv:1904.10509* (2019).

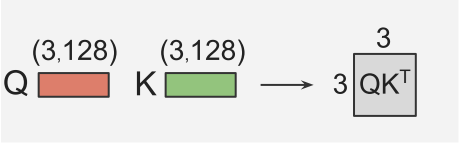
Efficient Self-Attention Design



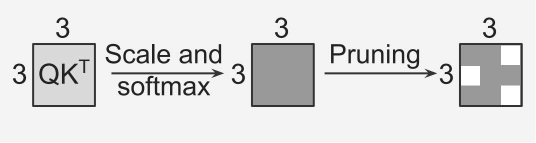
Step 1



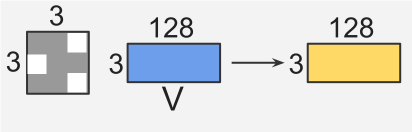
Step 2



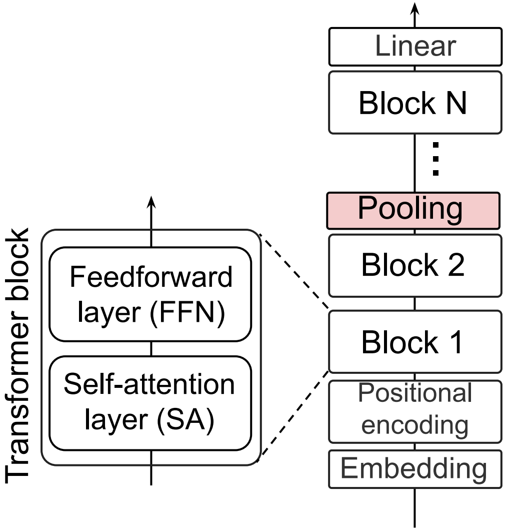
Step 3



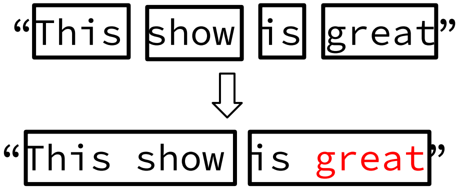
Step 4



Token Merging



Pooling layer will reduce the number of tokens.

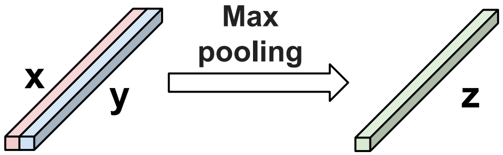


- We can reduce the number of tokens by merging them together.

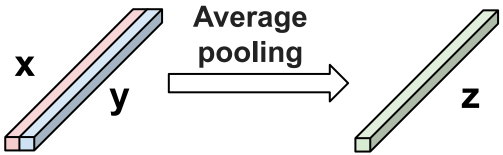
NYU SAI LAB Bolya, Daniel, et al. "Token merging: Your vit but faster." *arXiv preprint arXiv:2210.09461* (2022).

Token Merging

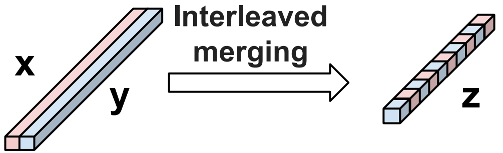
- x,y are two token vectors with length of E



$$z_i = \begin{cases} x_i & \text{if } (|x_i| >= |y_i|) \\ y_i & \text{otherwise} \end{cases}$$



$$z_i = (x_i + y_i)/2$$



$$z_i = \begin{cases} x_i & i \text{ is odd} \\ y_i & \text{otherwise} \end{cases}$$

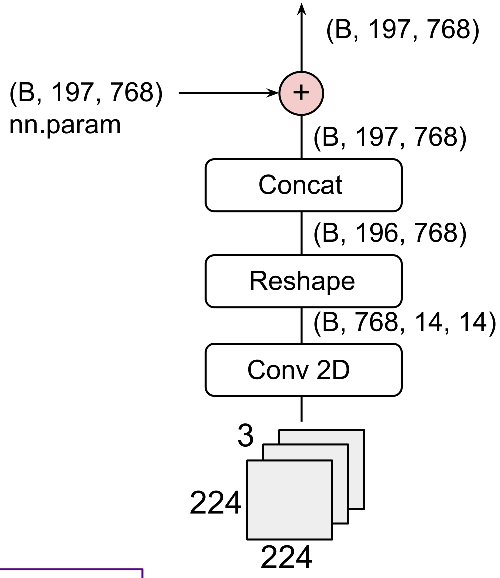
Bolya, Daniel, et al. "Token merging: Your vit but faster." *arXiv preprint arXiv:2210.09461* (2022).

Topics

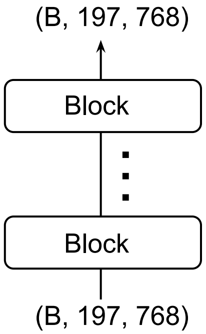
- Transformer basics
- Bert
- Vision transformer
- Large Language Model
- Self-supervised learning

Vision Transformer

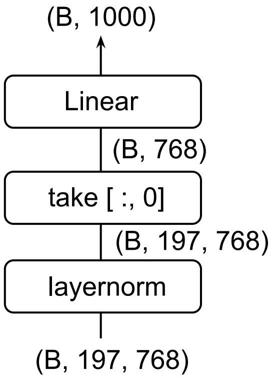
- Transformer architecture can also be applied over the computer vision tasks.



(part 1)



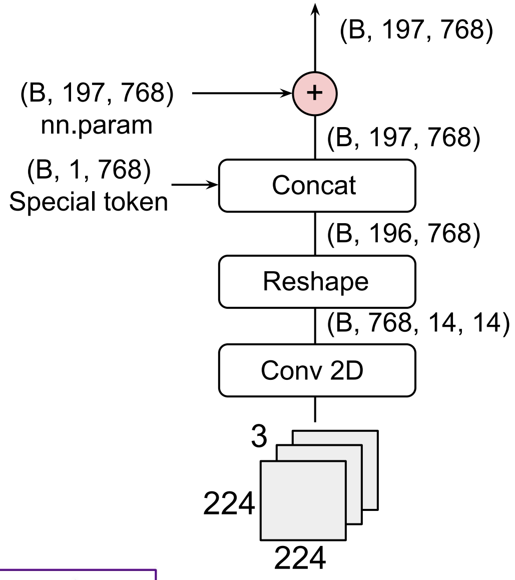
(part 2)



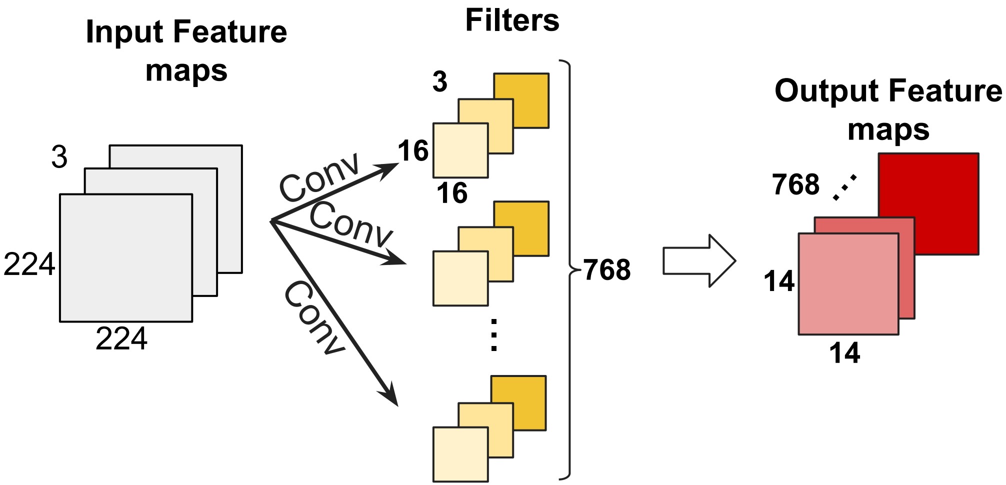
(part 3)

Vision Transformer

- A special placeholder is introduced to aggregate global information about the whole image.



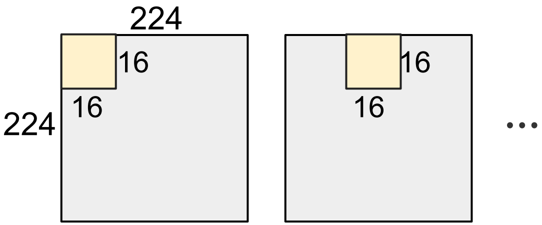
(part 1)

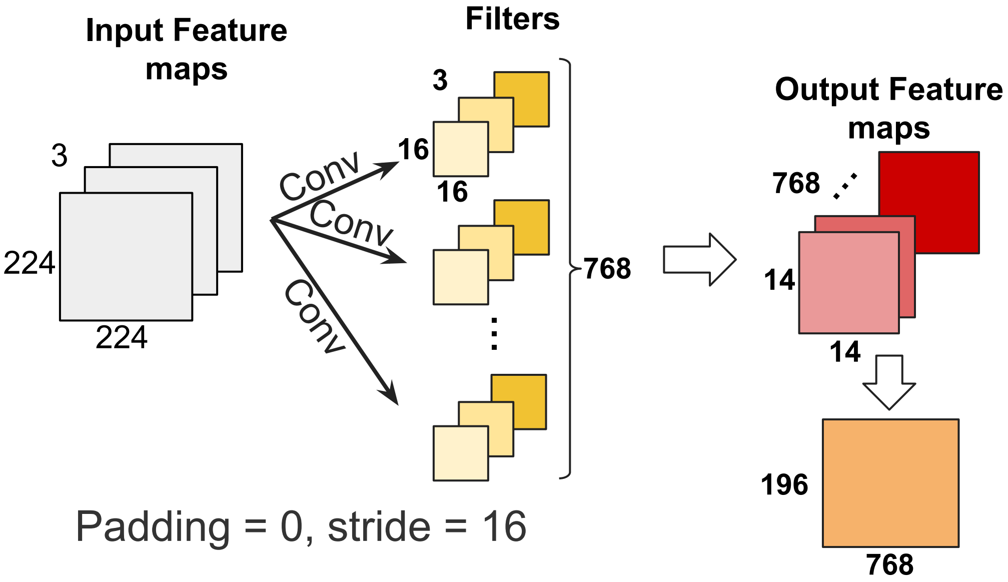


Padding = 0, stride = 16

Vision Transformer

- Transformer architecture can also be applied over the computer vision tasks.

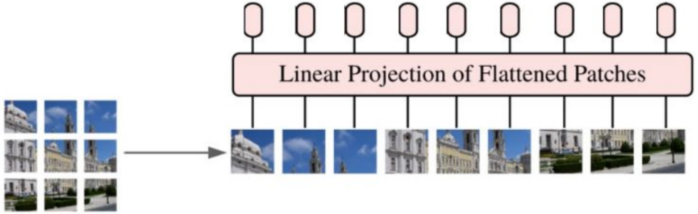


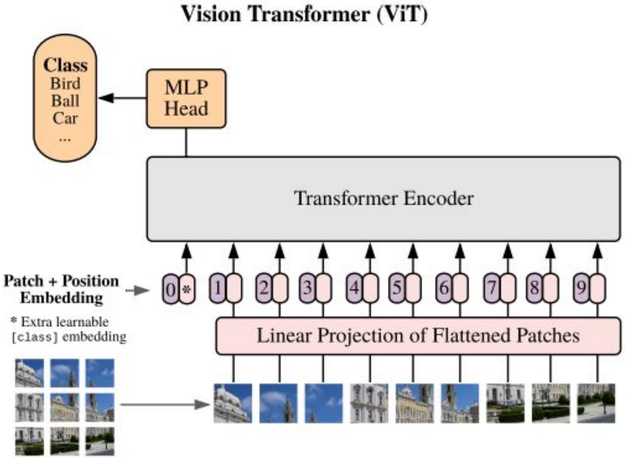


Padding = 0, stride = 16

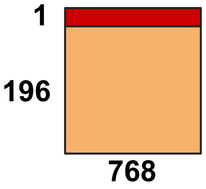
Vision Transformer

- An image C×H×W is divided into patches of C×P×P. P is 16✖16 in the previous example.
- They are then flattened and linearly projected to E (e.g., 768) dimensions for a sequence of (H/P) × (W/P) tokens.



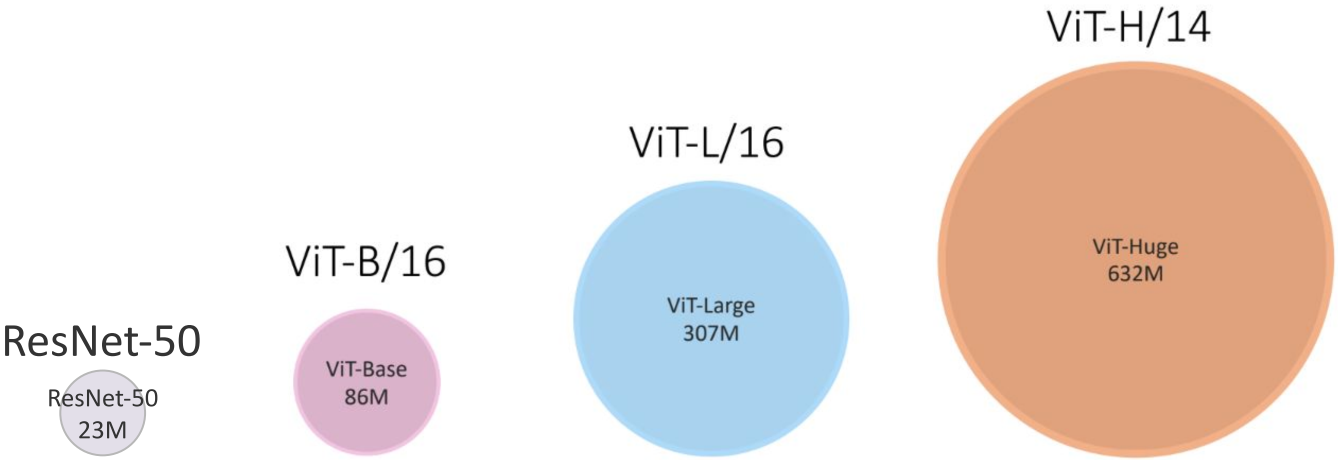

Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

- A classification [CLS] token is inserted at the start of the tokens.

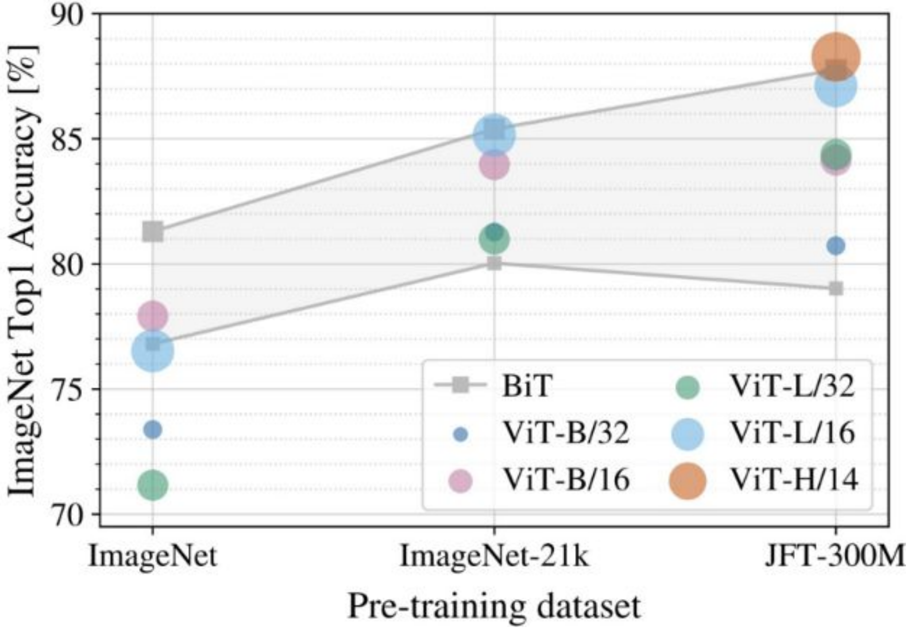


Dosovitskiy, Alexey. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

Vision Transformer



Vision Transformer



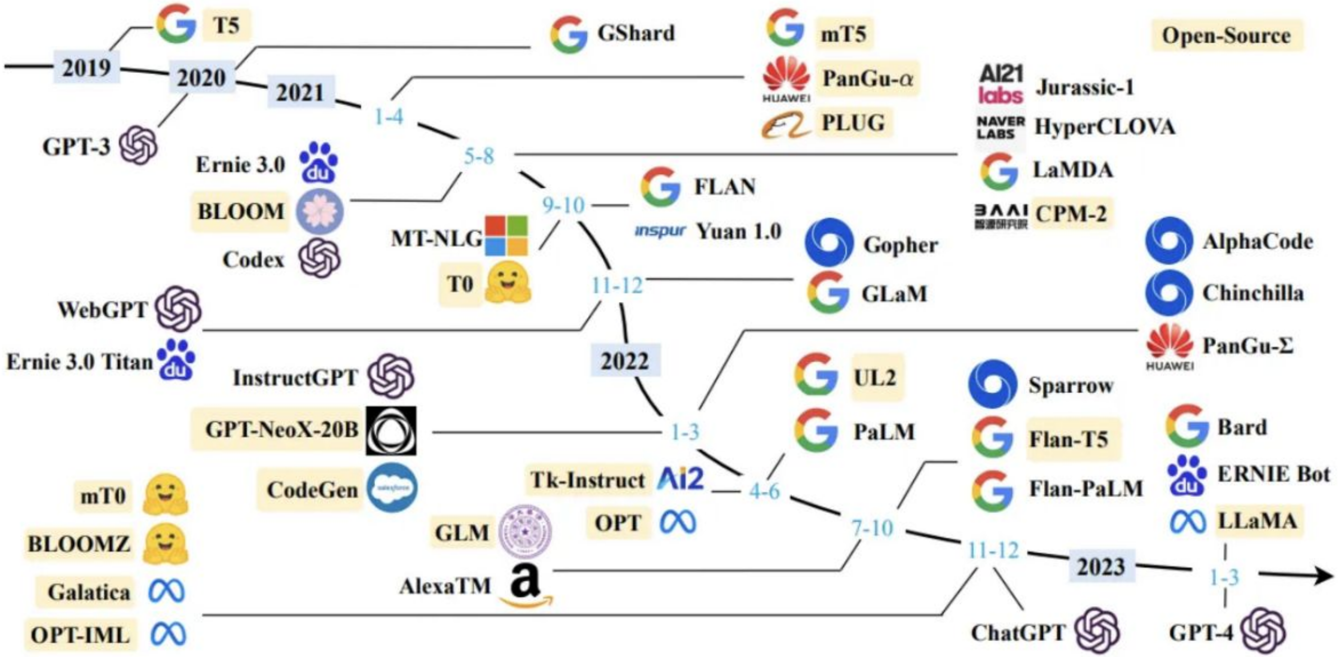
- Accuracy increases as the training dataset grow.
- ResNet 50 can achieves an accuracy between 75-80% on ImageNet.
- ViT does not strike an efficient balance between parameter count and accuracy when dataset is small.

NYU SAI LAB

Topics

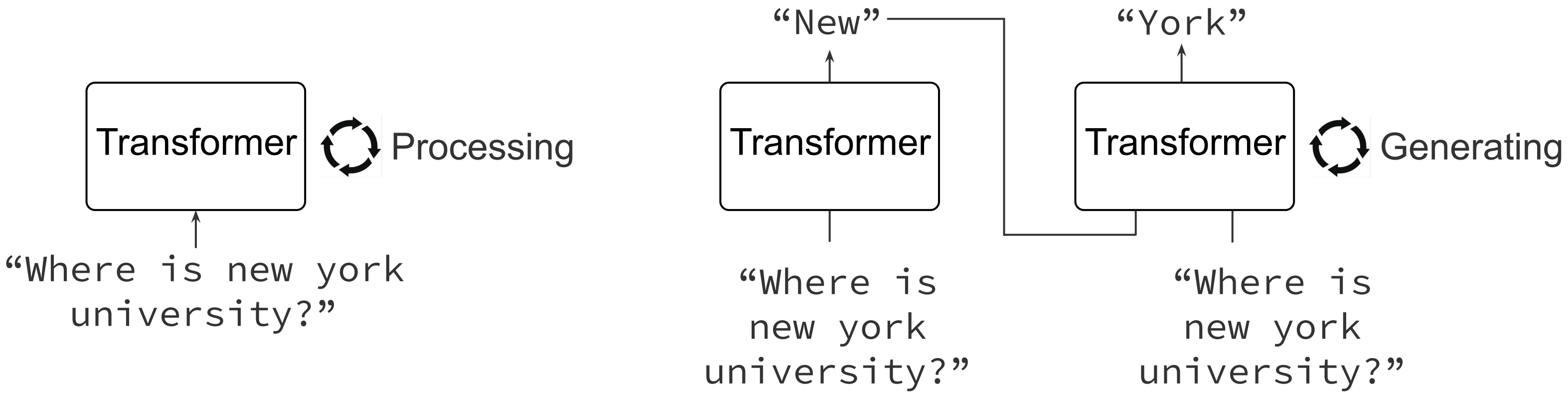
- Transformer basics
- Bert
- Vision transformer
- Large Language Model
- Self-supervised learning

Large Language Models (LLMs)



NYU SAI LAB

Transformers as a Generative AI Tool



Transformer

Processing

“Where is new york university?”

“New”

Transformer

“Where is new york university?”

“York”

Transformer

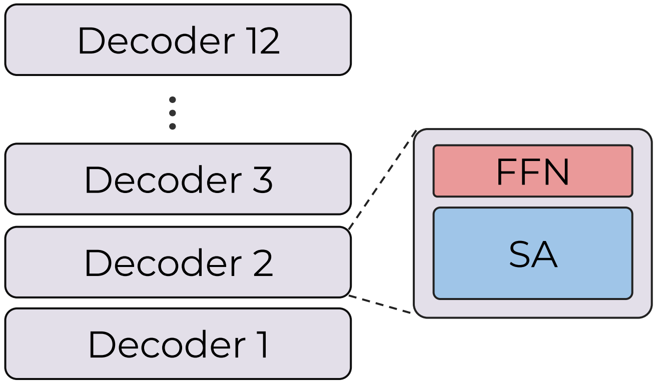
Generating

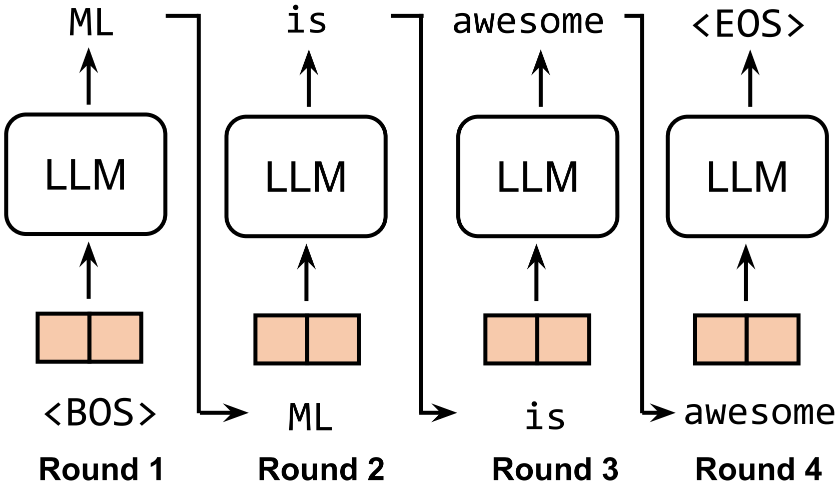
“Where is new york university?”

Step 1: Prefilling

Step 2: Decoding

Transformers as a Generative AI Tool

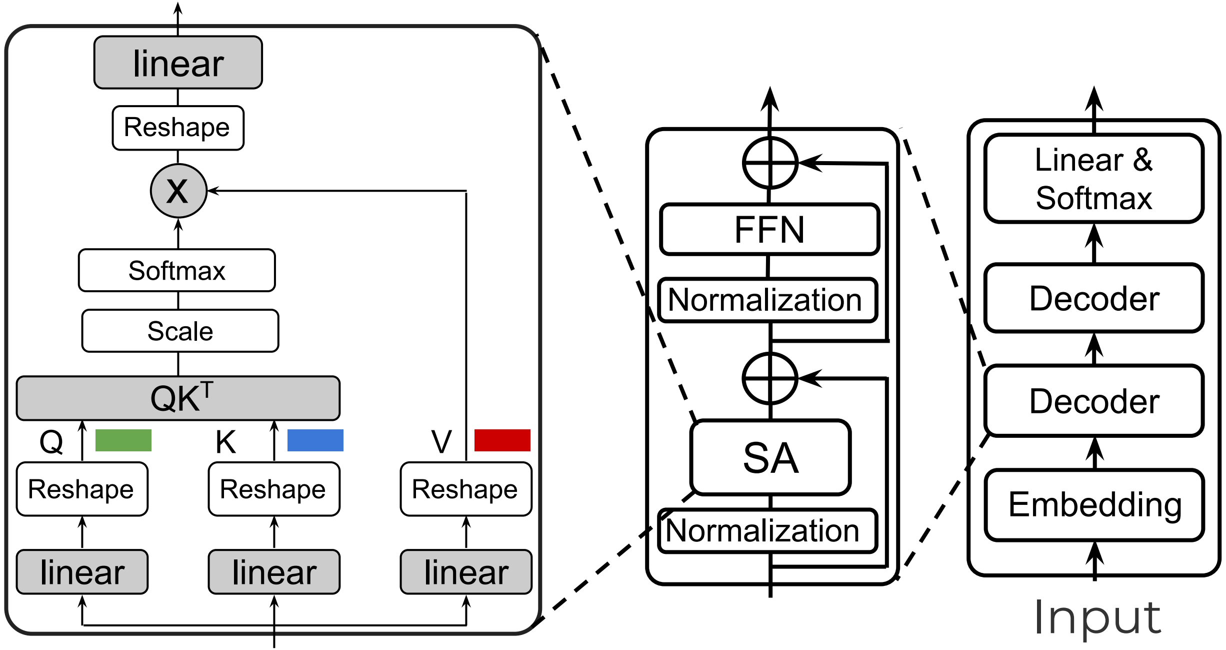




- Each token is generated in an autoregressive manner.

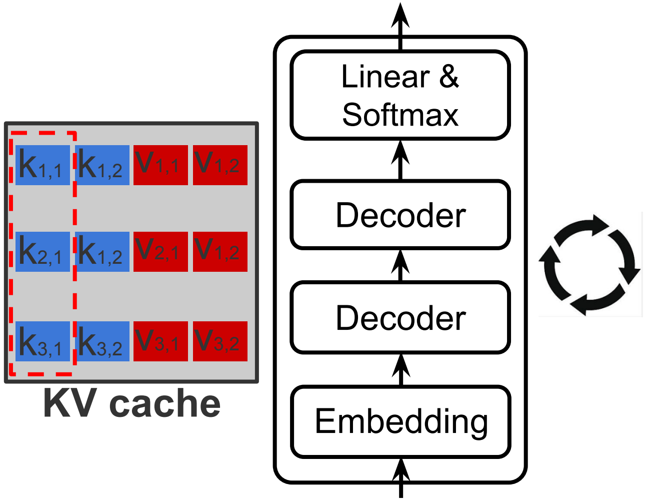
Radford, Alec, et al. "Language models are unsupervised multitask learners." *OpenAI blog* 1.8 (2019): 9.

Transformers as a Generative AI Tool



- We need to buffer the v and k for later usage.

GPT-2: Prefilling



$k_{1,1}$ $k_{1,2}$ $v_{1,1}$ $v_{1,2}$
$k_{2,1}$ $k_{1,2}$ $v_{2,1}$ $v_{1,2}$
$k_{3,1}$ $k_{3,2}$ $v_{3,1}$ $v_{3,2}$
KV cache
Linear & Softmax
Decoder
Decoder
Embedding

“How are you”

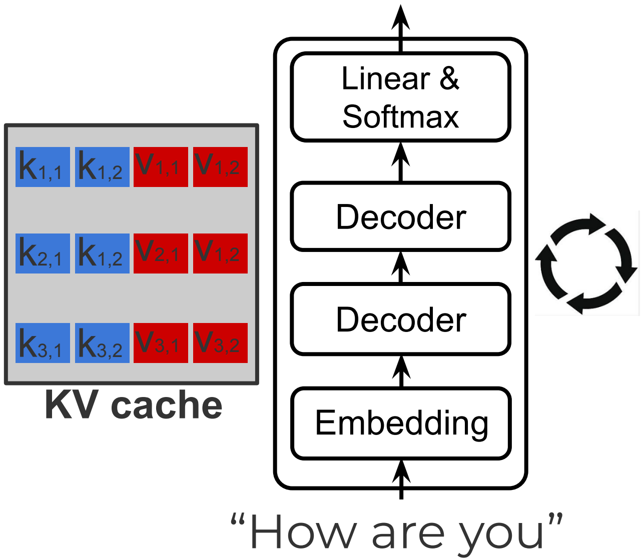
$k_{i,j}$ Key vector for ith token in jth layer (1×E)

$v_{i,j}$ Value vector for ith token in jth layer (1×E
)

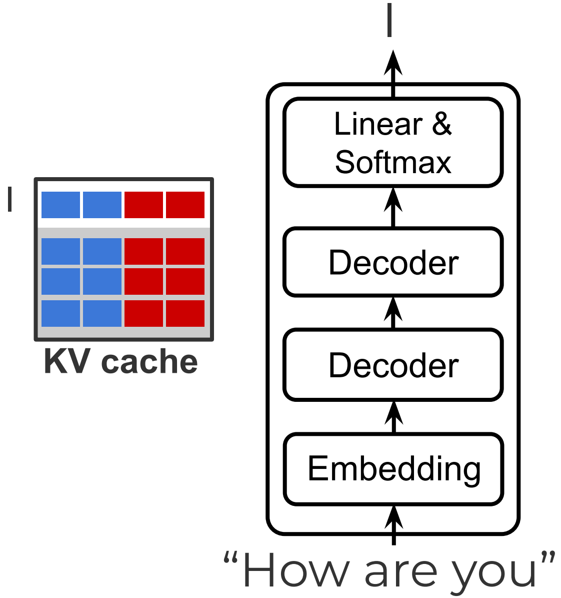
- During the prefilling stage, LLM processes the entire prompt, or context tokens jointly, saving the KV vectors into the memory.

NYU SAI LAB

GPT-2: Decoding



$k_{1,1}$ $k_{1,2}$ $v_{1,1}$ $v_{1,2}$
$k_{2,1}$ $k_{1,2}$ $v_{2,1}$ $v_{1,2}$
$k_{3,1}$ $k_{3,2}$ $v_{3,1}$ $v_{3,2}$
KV cache
Linear & Softmax
Decoder
Decoder
Embedding
“How are you”

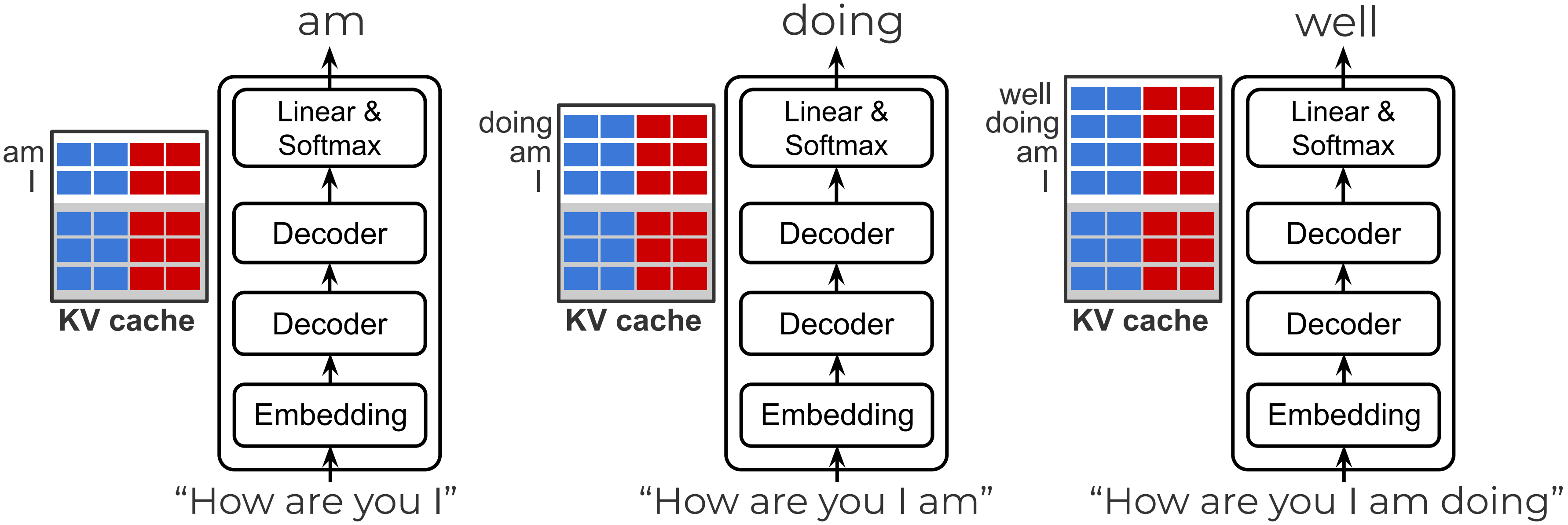


Round 1

- During the decoding stage, LLM generates the responses in an autoregressive way.

NYU SAI LAB

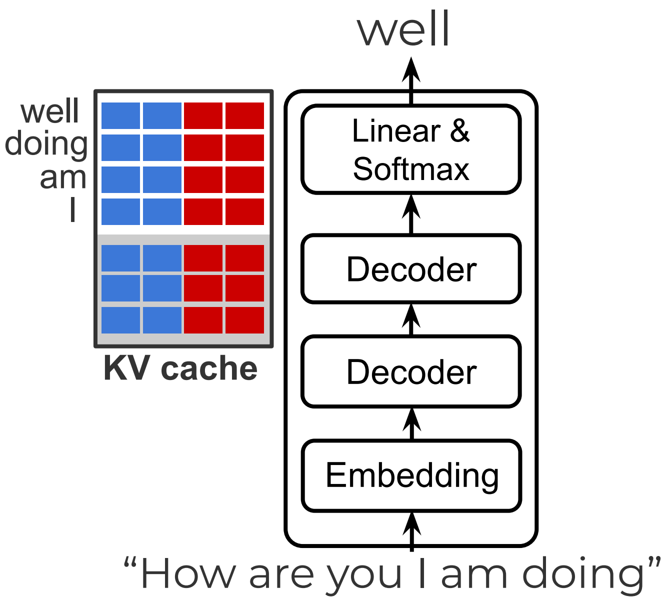
GPT-2: Decoding

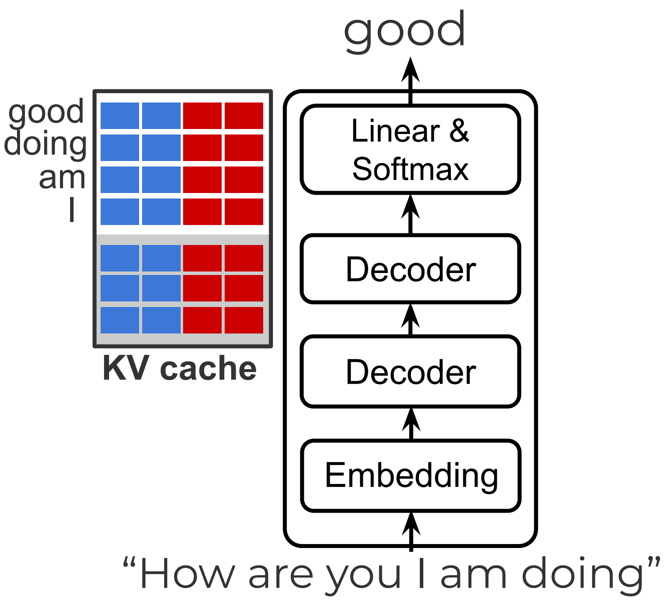


Round 1 **Round 2** **Round 3**

NYU SAI LAB

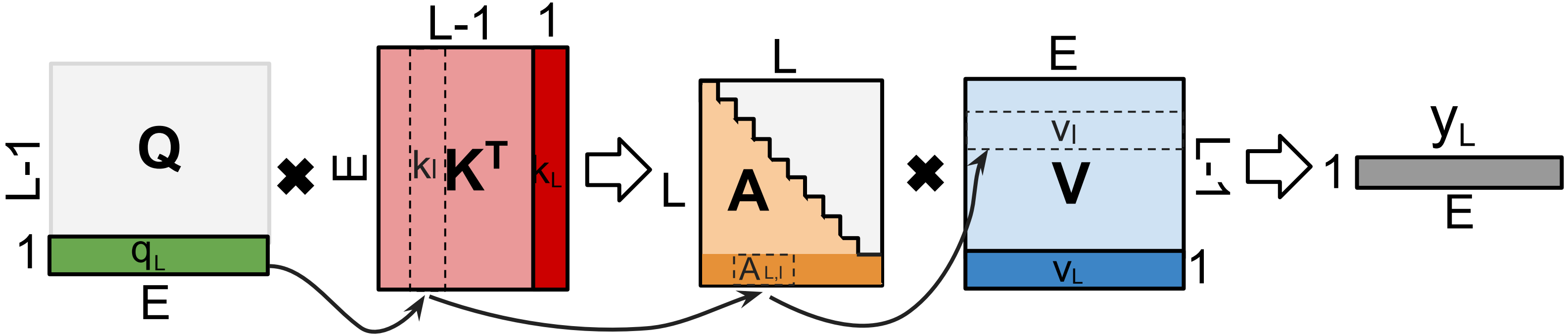
GPT-2: Decoding





- We can simply select the token with the highest score. But better results are achieved if the model considers other words as well. So a better strategy is to sample a word from the entire list using the score as the probability of selecting that word.

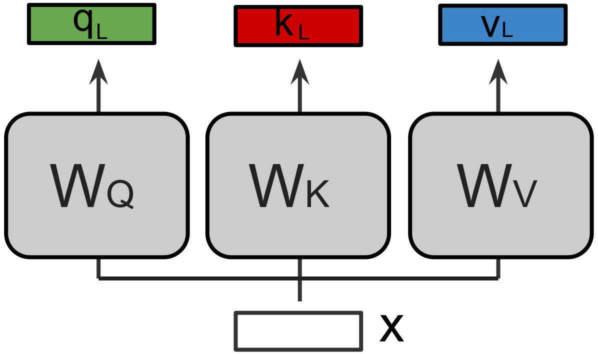
Why KV Cache Saves Computation?



- During the decoding phase, new tokens are continuously generated and must be processed using the buffered K and V vectors to generate subsequent tokens.
- Without a KV cache, all previous K and V vectors must be recomputed, resulting in significant computational overhead.

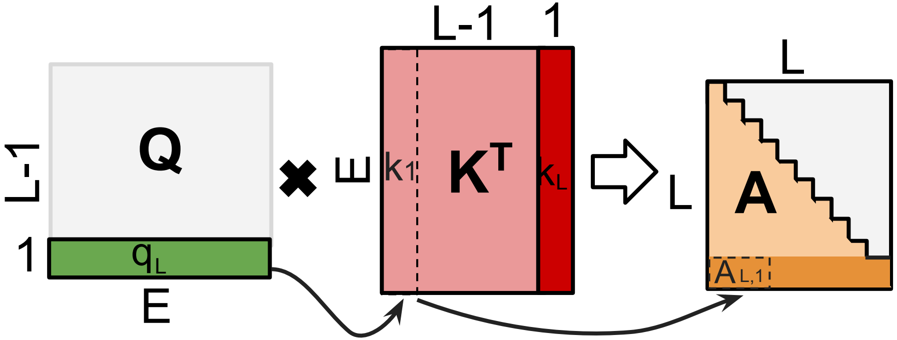
NYU SAI LAB

Why KV Cache Saves Computation?



- Given the input, the q_L, k_L, v_L are first computed by passing through the linear layers.
- After that, the k_l, v_l vectors (l=1,...,L-1) are also loaded from the memory.

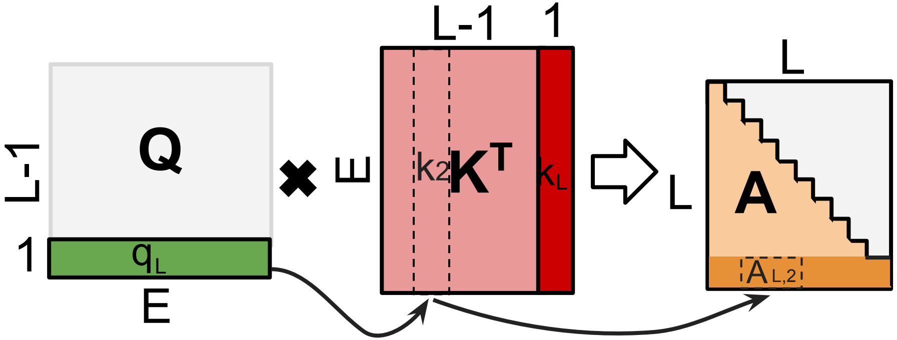
Why KV Cache Saves Computation?



- K and V are loaded from the memory, the q vector of the current token (q_L) is multiplied with the each of the key vector k_i, (i=1…L) to produce the result $A_{L,i}$

NYU SAI LAB

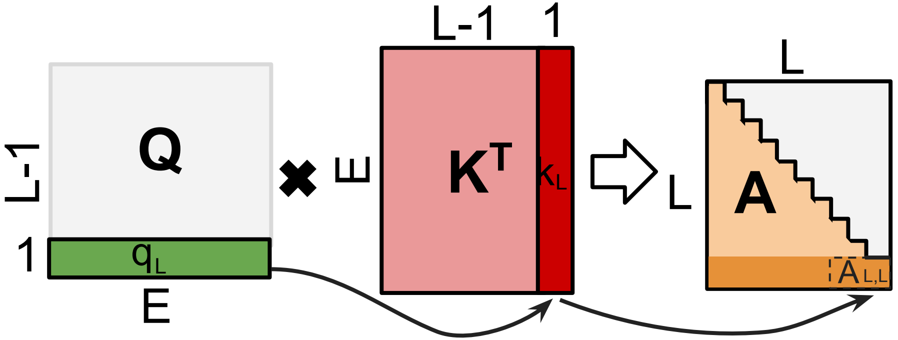
Why KV Cache Saves Computation?



- K and V are loaded from the memory, the q vector of the current token (q_L) is multiplied with the each of the key vector k_i, (i=1…L) to produce the result $A_{L,i}$

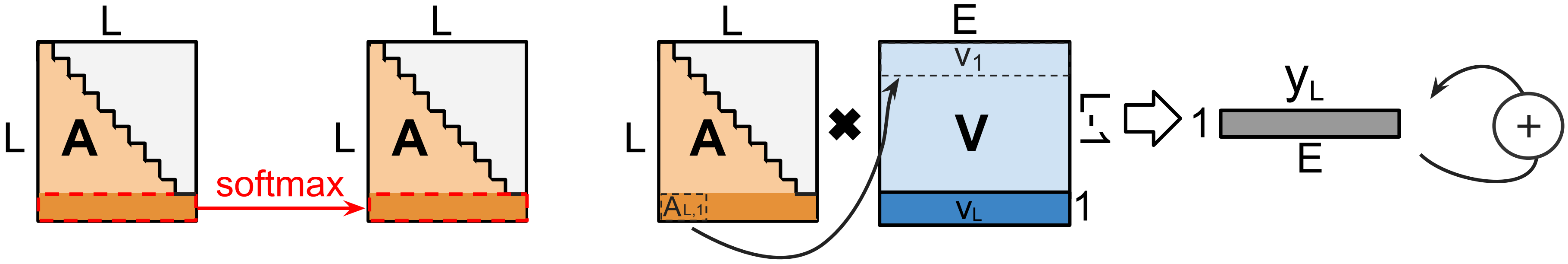
NYU SAI LAB

Why KV Cache Saves Computation?



- K and V are loaded from the memory, the q vector of the current token (q_L) is multiplied with the each of the key vector k_i, (i=1…L) to produce the result $A_{L,i}$

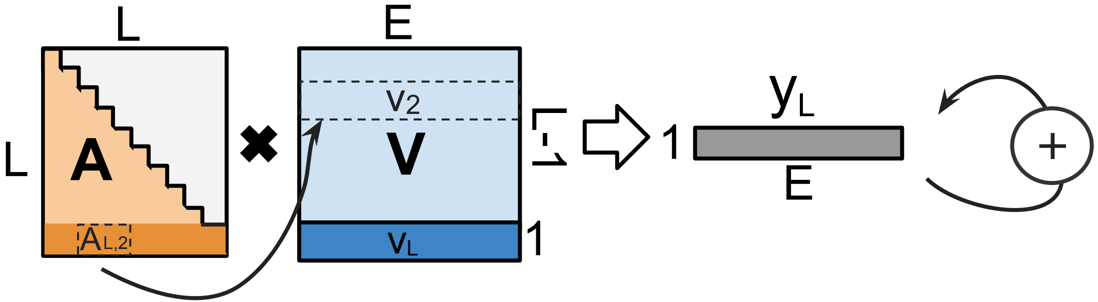
Why KV Cache Saves Computation?



- Afterwards, the computed $A_{L,i}$ (i=1…L) will then passed to softmax function.
- Then each element of A_L will then multiplied with vi (i=1…L) and elementwise sum together.

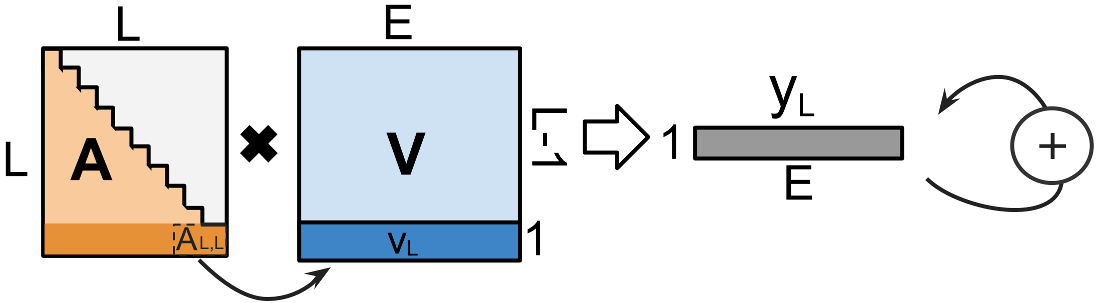
NYU SAI LAB

Why KV Cache Saves Computation?



- Afterwards, the computed $A_{L,i}$ (i=1…L) will then passed to softmax function
- Then each element of A_L will then multiplied with vi (i=1…L) and elementwise sum together.

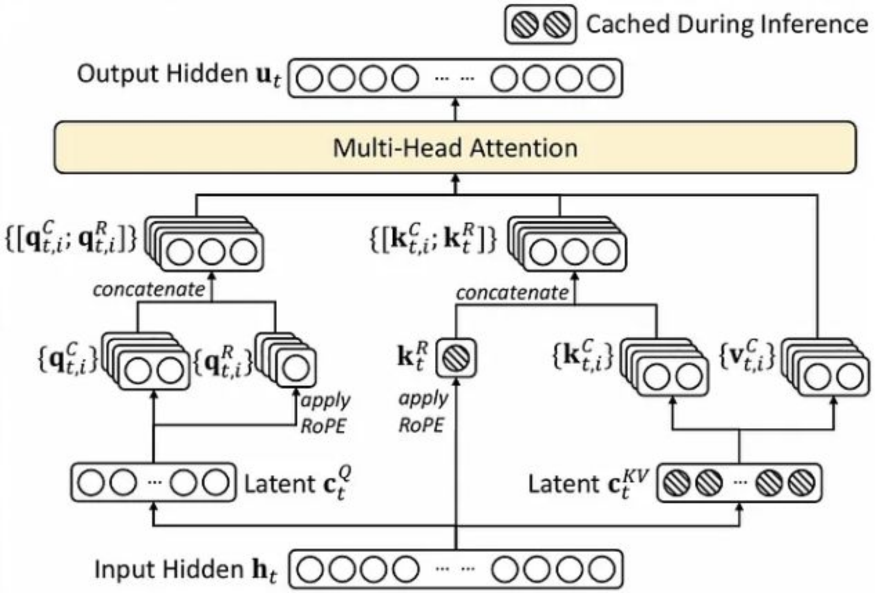
Why KV Cache Saves Computation?

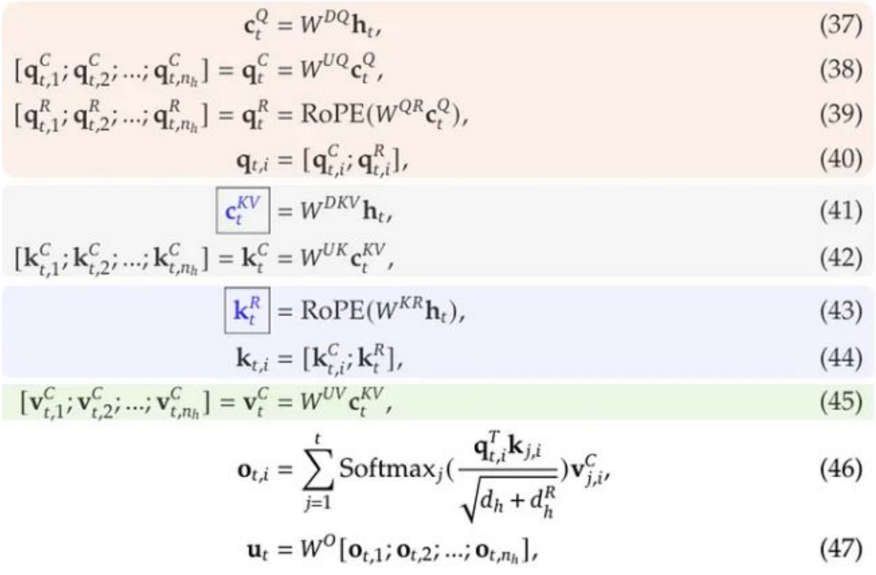


- Afterwards, the computed $A_{L,i}$ (i=1…L) will then passed to softmax function
- Then each element of A_L will then multiplied with vi (i=1…L) and elementwise sum together.

NYU SAI LAB

Deepseek V3





$$\mathbf{c}_t^Q = W^{DQ}\mathbf{h}_t, \tag{37}$$

$$[\mathbf{q}^C_{t,1}; \mathbf{q}^C_{t,2}; \ldots; \mathbf{q}^C_{t,n_h}] = \mathbf{q}^C_t = W^{UQ}\mathbf{c}^Q_t, \tag{38}$$

$$[\mathbf{q}^R_{t,1}; \mathbf{q}^R_{t,2}; \ldots; \mathbf{q}^R_{t,n_h}] = \mathbf{q}^R_t = \mathrm{RoPE}(W^{QR}\mathbf{c}^Q_t), \tag{39}$$

$$\mathbf{q}_{t,i} = [\mathbf{q}^C_{t,i}; \mathbf{q}^R_{t,i}], \tag{40}$$

$$\boxed{\mathbf{c}^{KV}_t} = W^{DKV}\mathbf{h}_t, \tag{41}$$

$$[\mathbf{k}^C_{t,1}; \mathbf{k}^C_{t,2}; \ldots; \mathbf{k}^C_{t,n_h}] = \mathbf{k}^C_t = W^{UK}\mathbf{c}^{KV}_t, \tag{42}$$

$$\boxed{\mathbf{k}^R_t} = \mathrm{RoPE}(W^{KR}\mathbf{h}_t), \tag{43}$$

$$\mathbf{k}_{t,i} = [\mathbf{k}^C_{t,i}; \mathbf{k}^R_t], \tag{44}$$

$$[\mathbf{v}^C_{t,1}; \mathbf{v}^C_{t,2}; \ldots; \mathbf{v}^C_{t,n_h}] = \mathbf{v}^C_t = W^{UV}\mathbf{c}^{KV}_t, \tag{45}$$

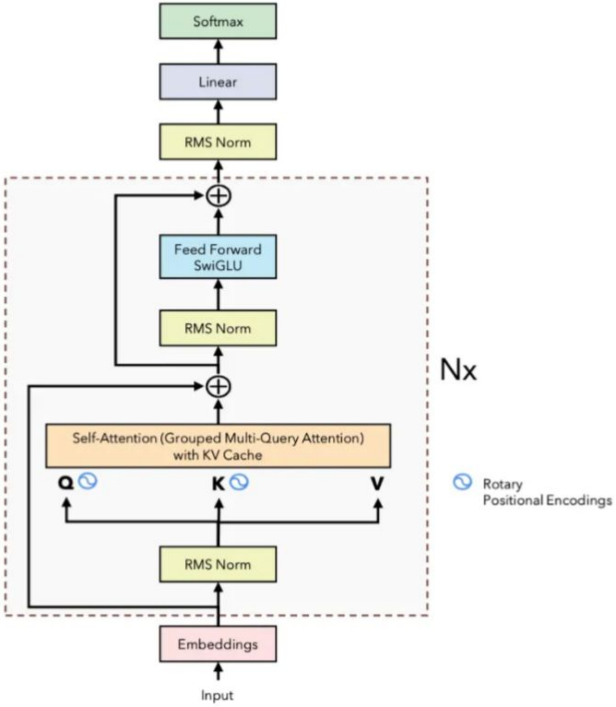
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}^T_{t,i}\mathbf{k}_{j,i}}{\sqrt{d_h + d^R_h}}\right)\mathbf{v}^C_{j,i}, \tag{46}$$

$$\mathbf{u}_t = W^O[\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \ldots; \mathbf{o}_{t,n_h}], \tag{47}$$

Liu, Aixin, et al. "Deepseek-v3 technical report." *arXiv preprint arXiv:2412.19437* (2024).

LLaMA

- LLaMA has a similar architecture as GPT-2, with some minor differences:
  - RMSNorm is used to replace LayerNorm
  - SwiGLU
  - MLPs with gating

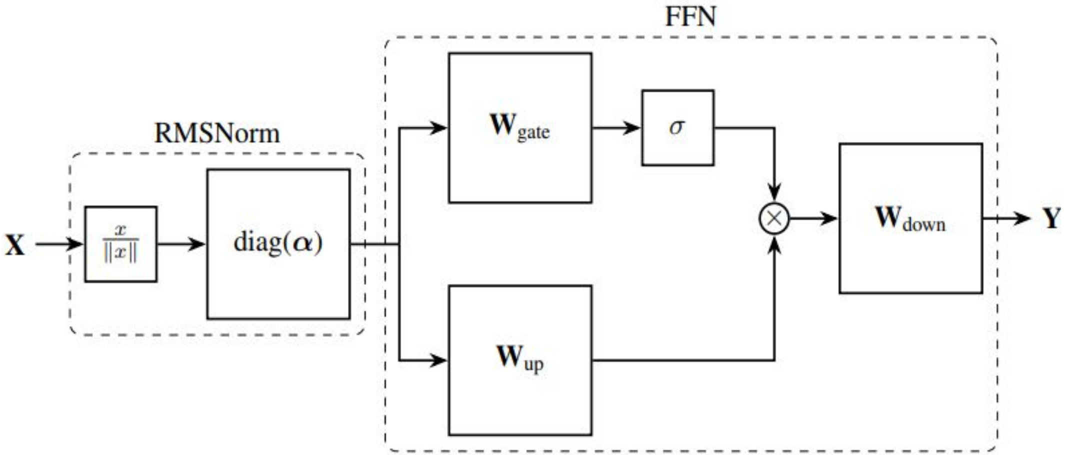

Softmax
Linear
RMS Norm
Feed Forward SwiGLU
RMS Norm
Nx
Self-Attention (Grouped Multi-Query Attention) with KV Cache
Q
K
V
Rotary Positional Encodings
RMS Norm
Embeddings
Input

LLaMA

NYU SAI LAB

Touvron, Hugo, et al. "Llama: Open and efficient foundation language models." *arXiv preprint arXiv:2302.13971* (2023).

MLPs with Gating



- A lot of LLM models applies gated feed-forward network to replace the conventional FFN in the transformer.

Liu, Hanxiao, et al. "Pay attention to mlps." *Advances in neural information processing systems* 34 (2021): 9204-9215.

How LLM is Trained?

- The loss function consists of two parts:

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

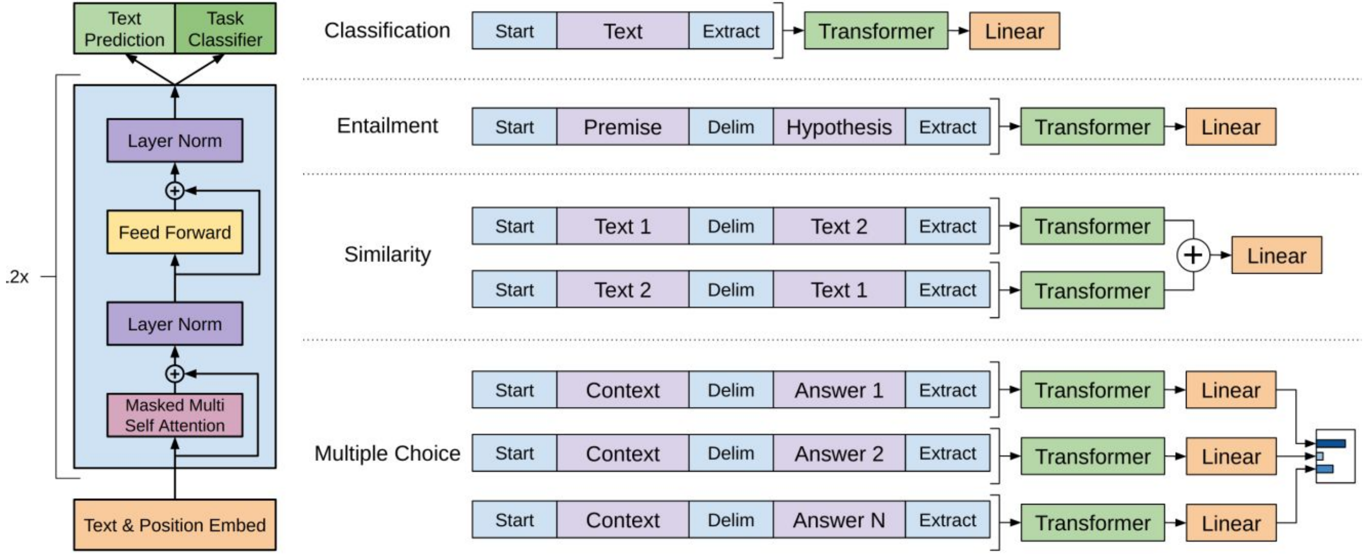
“A newspaper article should contain these five main components: a headline, a byline, a lead/lede paragraph, an explanation, and any other additional information.”

A newspaper article should contain these five main **xxx** → “components” (GPT)

A newspaper article should **xxx** these five main components: a headline, a byline, a lead/lede paragraph, an explanation, and any other additional information. → “contain” (BERT)

Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018).
Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

How LLM is Trained?

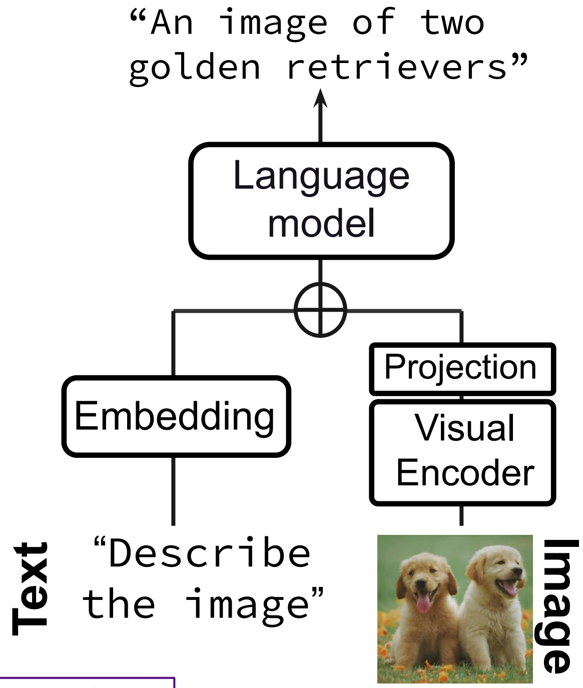


$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

NYU SAI LAB

Radford, Alec, et al. "Improving language understanding by generative pre-training." (2018).

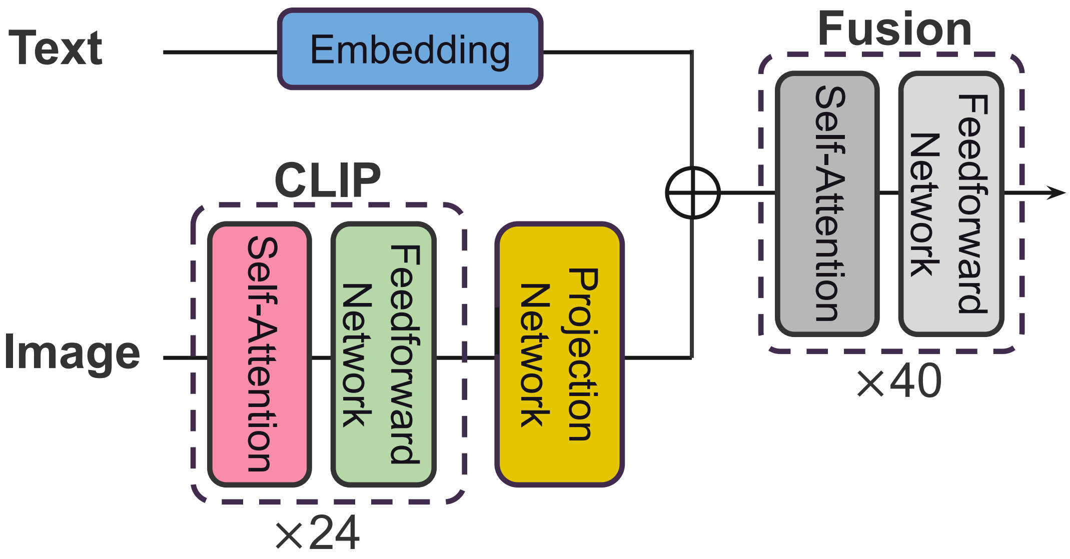
Vision Language Model



- A **Vision-Language Model (VLM)** is an large model that jointly processes from visual data (e.g., images, video) and textual data to understand, align, and generate multimodal content.

NYU SAI LAB

LLaVA



- In Llava, the visual encoder takes the input images and produced the visual embeddings.

- The visual embeddings and textual embeddings are concatenated, which is then forwarded to the fusion model.

NYU SAI LAB

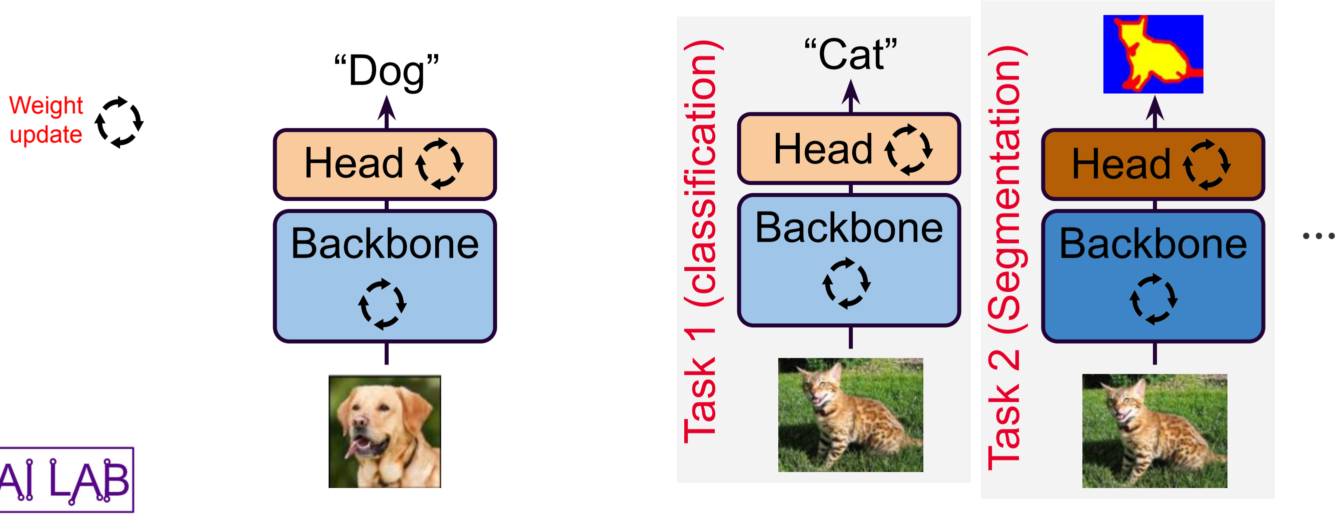
Liu, Haotian, et al. "Visual instruction tuning." *Advances in neural information processing systems* 36 (2023): 34892-34916.

Topics

- Transformer basics
- Bert
- Vision transformer
- Large Language Model
- Self-supervised learning for visual model

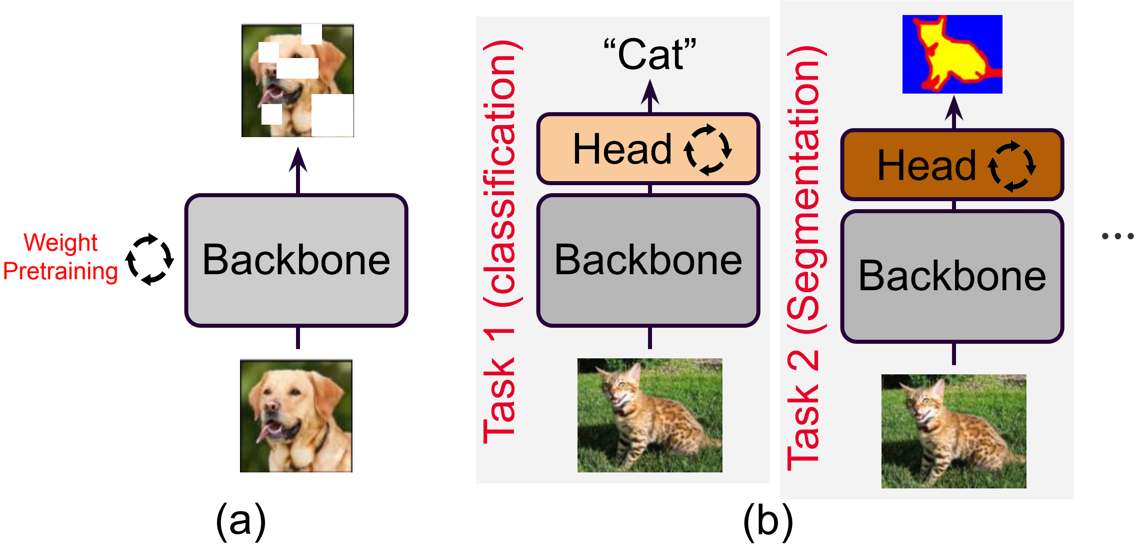
Self-Supervised Learning

- Self-Supervised Learning (SSL) is a paradigm that leverages intrinsic structures within unlabeled data to create pretext tasks, enabling models to learn meaningful representations that can be fine-tuned for downstream applications.



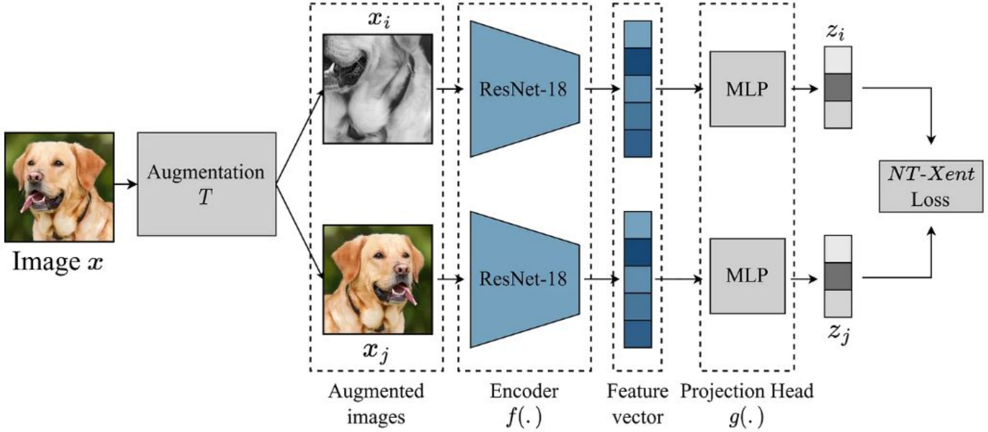
Self-Supervised Learning

- Self-Supervised Learning (SSL) is a paradigm that leverages intrinsic structures within unlabeled data to create pretext tasks, enabling models to learn meaningful representations that can be fine-tuned for downstream applications.



Popular SSL Methods: Contrastive Learning

- Contrastive Learning is a framework in which models learn meaningful representations by contrasting positive pairs (similar data points) with negative pairs (dissimilar data points), encouraging the embedding space to capture semantic similarities and differences.

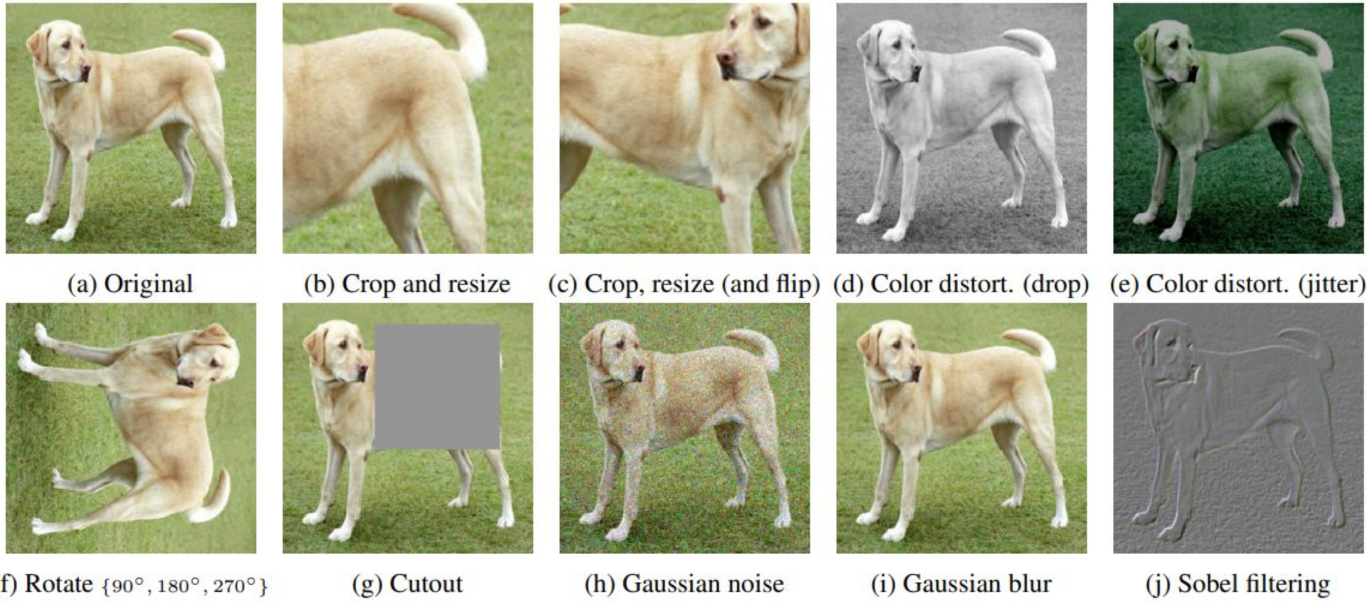


SimCLR framwork. Image by author.

NYU SAI LAB

Chen, Ting, et al. "A simple framework for contrastive learning of visual representations." *International conference on machine learning*. PMLR, 2020.

Popular SSL Methods: Contrastive Learning

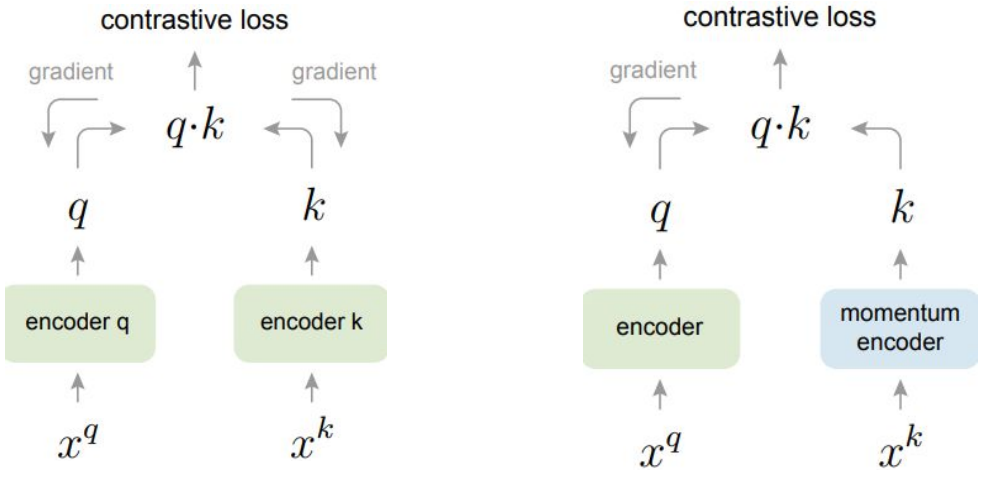


(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate $\{90^\circ, 180^\circ, 270^\circ\}$ (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

- The current augmentation approaches adopted by the AI community including random crop (with flip and resize), color distortion, and Gaussian blur.

Momentum Contrast SSL (MoCo)



- In the training process, only query encoder is updated, momentum encoder doesn't change.

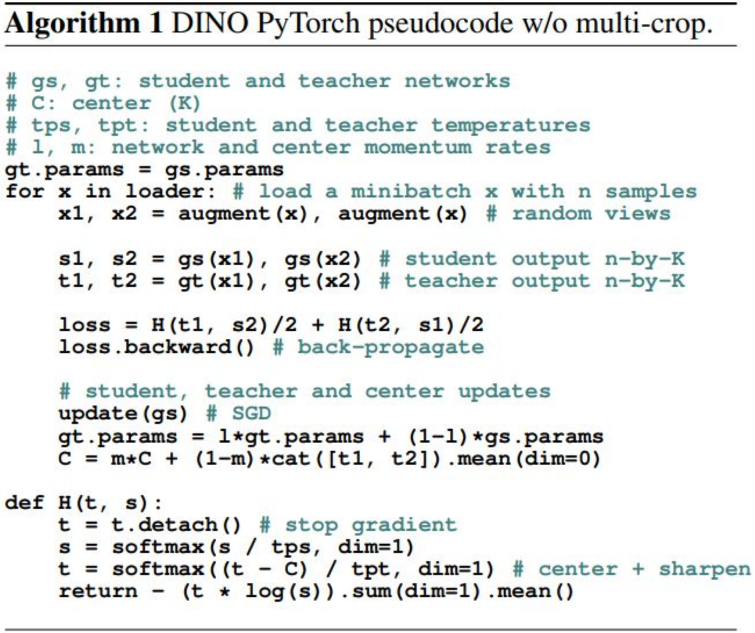
$$\theta_{\mathrm{k}} \leftarrow m\theta_{\mathrm{k}} + (1 - m)\theta_{\mathrm{q}}.$$

- Only the encoder is updated.

NYU SAI LAB

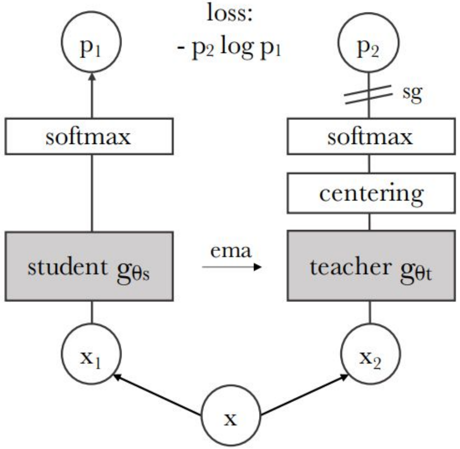
He, Kaiming, et al. "Momentum contrast for unsupervised visual representation learning." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020.

Knowledge Distillation with No Labels (DINO)



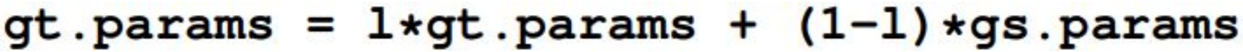
Algorithm 1 DINO PyTorch pseudocode w/o multi-crop.

```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2)/2 + H(t2, s1)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s):
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```



DINO (2021)

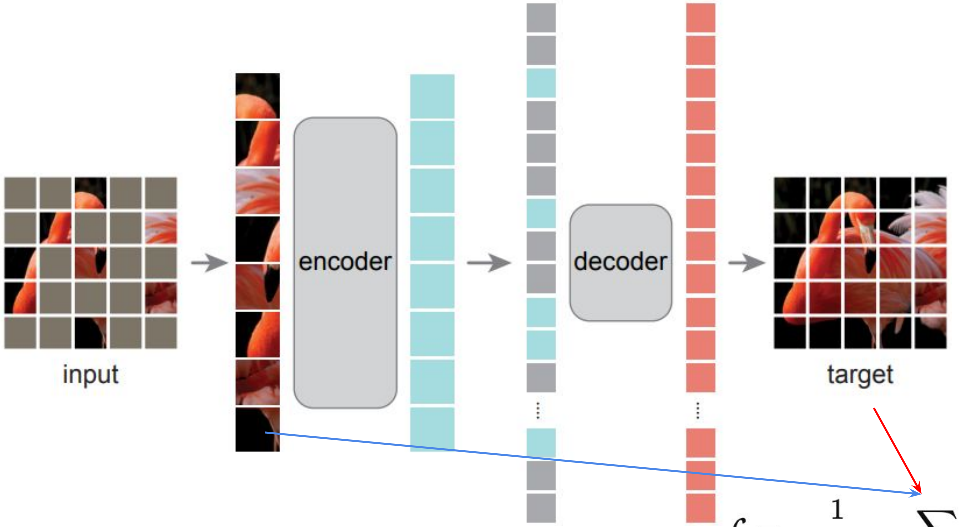
- During the backpropagation, only the student DNN is updated.
- The teacher updates its weight periodically using the following formula:



```
gt.params = l*gt.params + (1-l)*gs.params
```

NYU SAI LAB

Masked AutoEncoder (MAE)

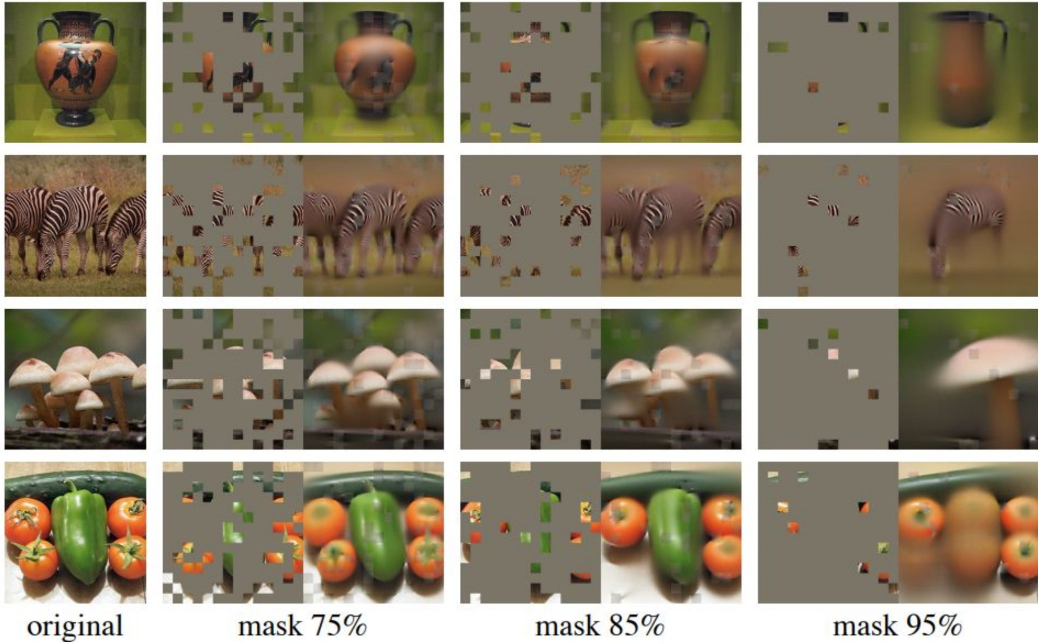


$$\mathcal{L} = \frac{1}{N_{\text{masked}}} \sum_{i \in \text{masked}} \|\hat{x}_i - x_i\|^2$$

- The input image is masked, the unmasked image patches will be sent to the encoder.
- The decoder will infer the masked portion of the image.

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

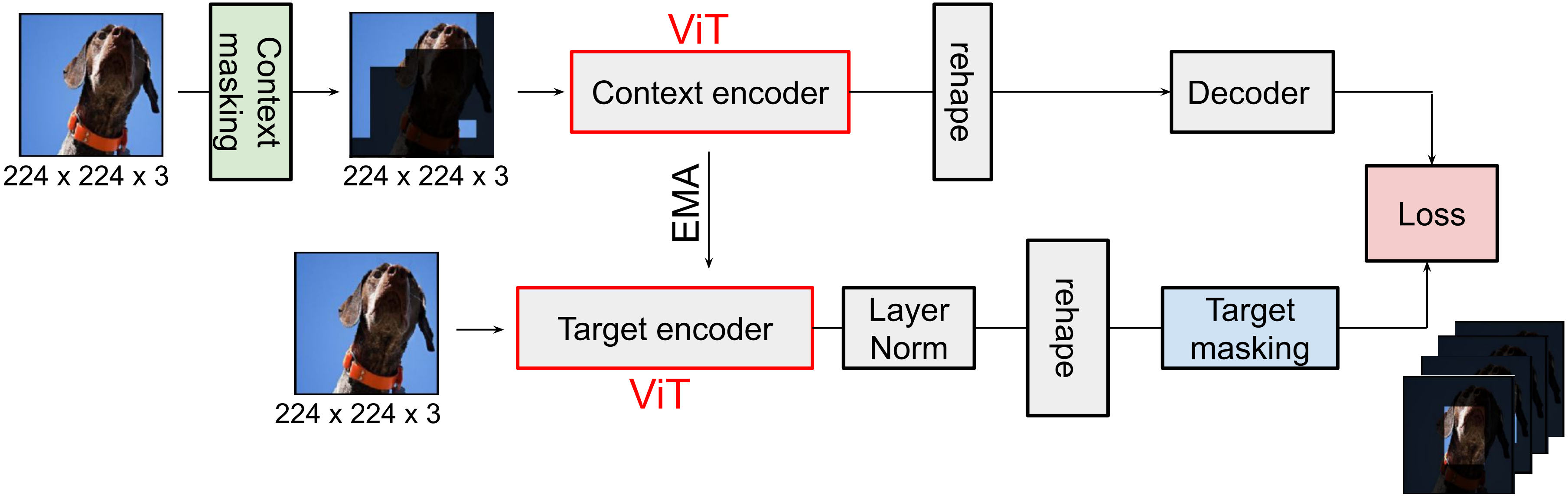
Self-Supervised Learning: Masked Autoencoder



NYU SAI LAB

He, Kaiming, et al. "Masked autoencoders are scalable vision learners." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

JEPA



Assran, Mahmoud, et al. "Self-supervised learning from images with a joint-embedding predictive architecture." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.