Lecture 03:
Transformer and Large Model

Course Information

- 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

Notes

- Quiz next week
- Lab 1 will be posted next weekend.

Recap

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

NYU SAI LAB

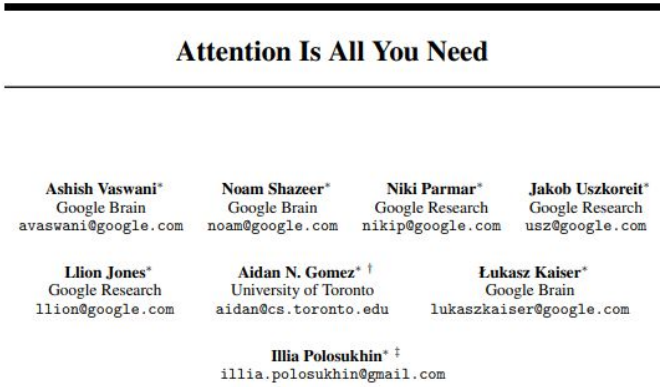
Topics

- Transformer basics
- Bert
- Vision transformer
- Large Language Model

NYU SAI LAB

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

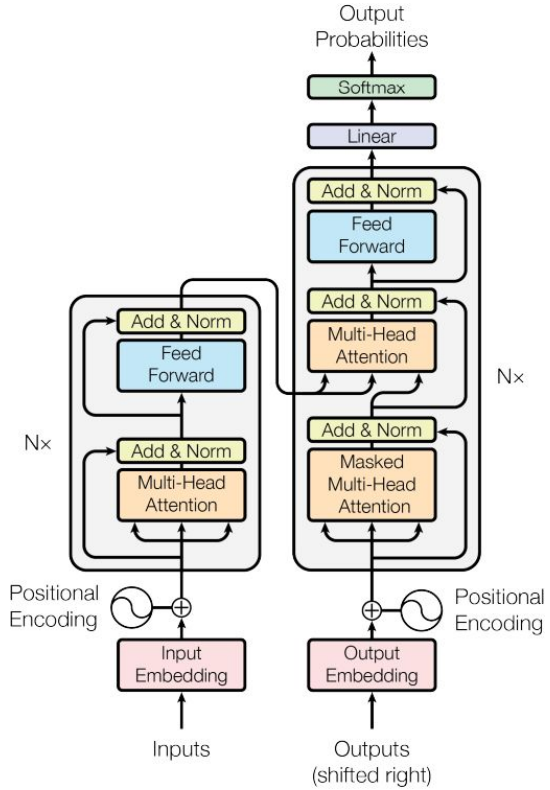
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

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

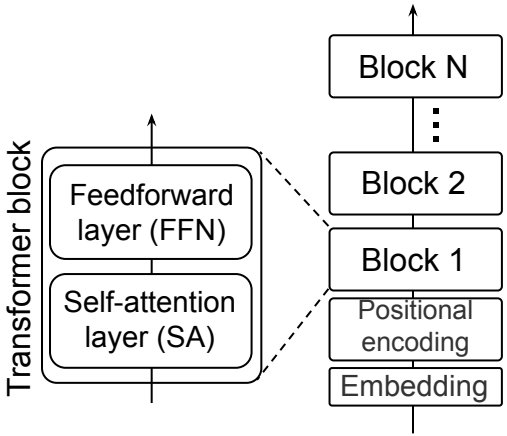
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

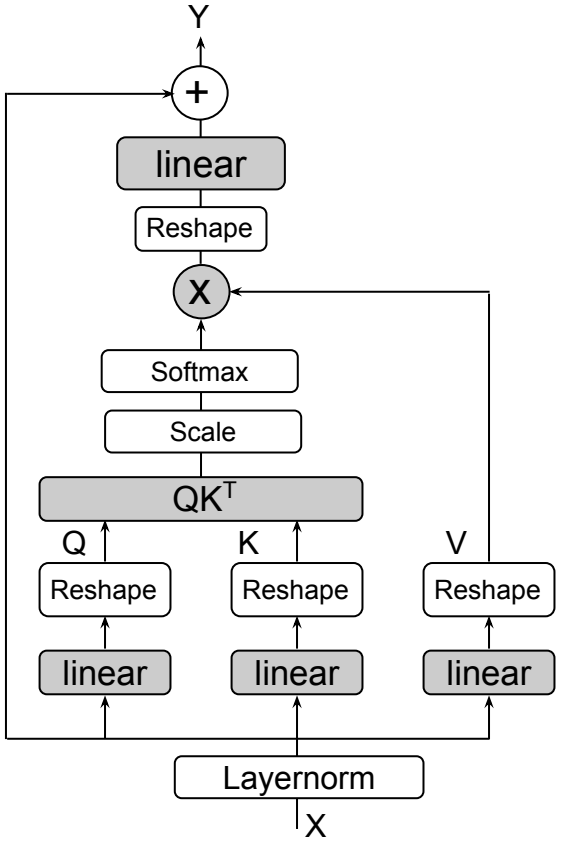


Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

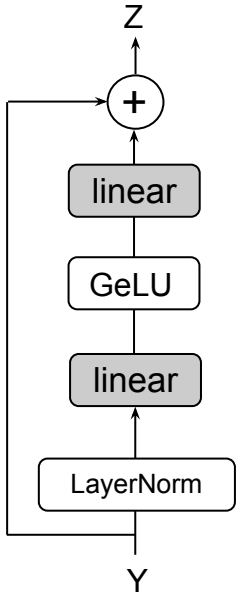
Transformers



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



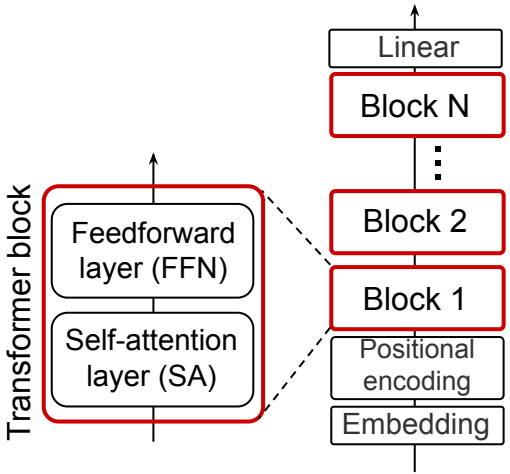
Self attention block (SA)



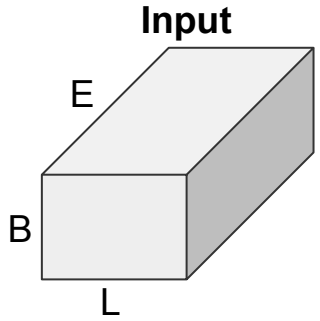
Feed forward block (FFN)

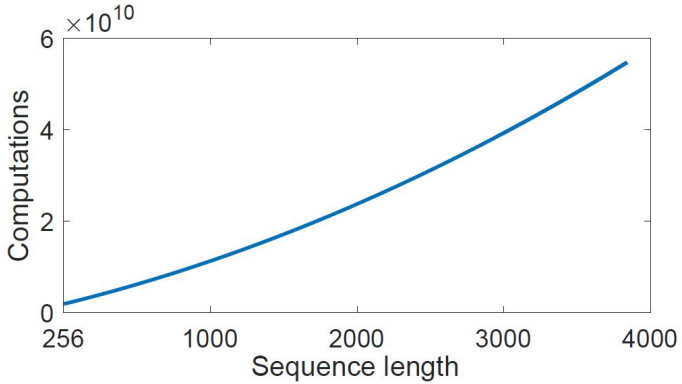
NYU SAI LAB

Transformers: Transformer Block



Transformers



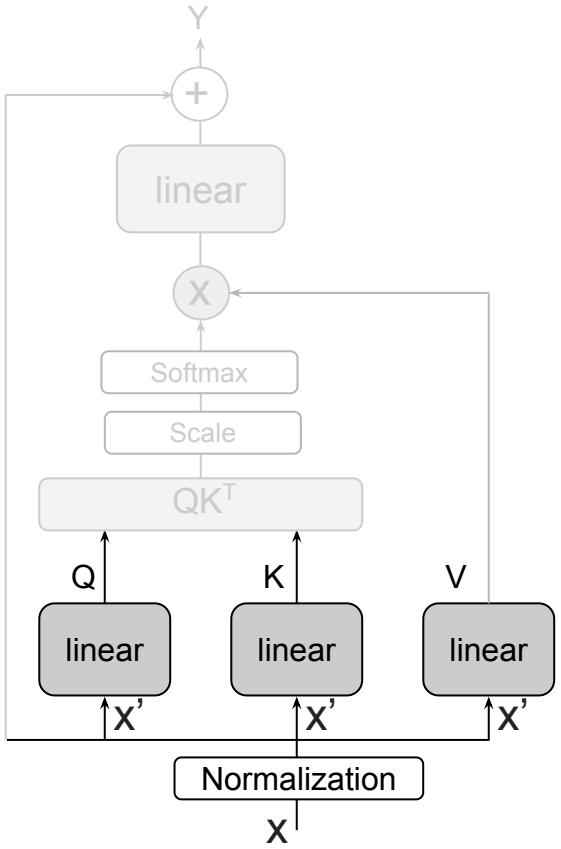


- The input contains three dimensions:
  - B: batch
  - L: token length
  - E: embeddings
- The amount of computation is closely related to the token length L.
- Longer sequences are disproportionately expensive because attention is quadratic to the sequence length.

NYU SAI LAB

Self-Attention Block

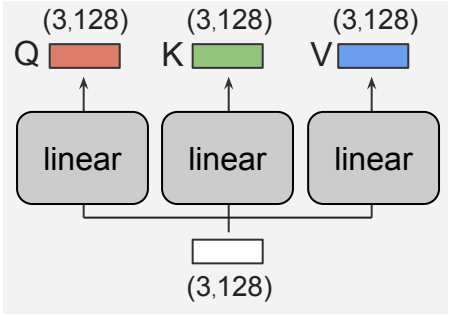
- The input x is first normalized, then the first step in calculating self-attention is to create three vectors from the input x', denoted as: Query (Q), Key (K), Value (V).
  - $(B,L,E) \times (E \times E) \rightarrow (B \times L \times E)$ (BLE^2)
- The second step in calculating self-attention. This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



NYU SAI LAB

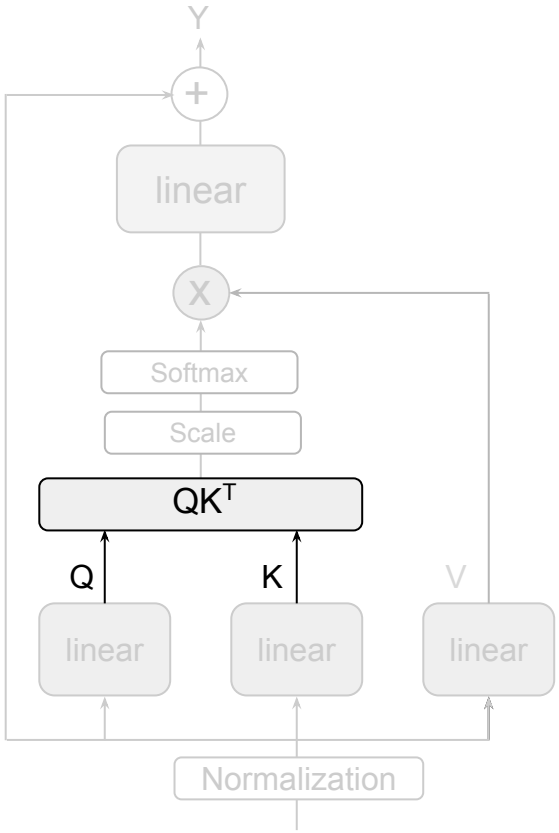
Example

“I love AI” → 3 × 128



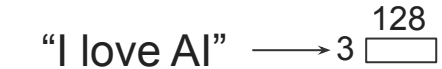
Self-Attention Block

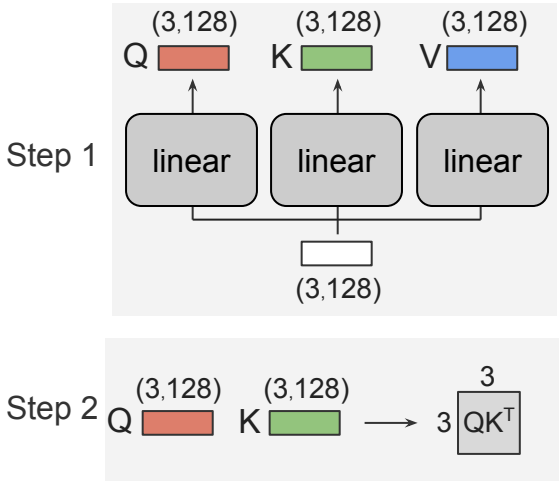
- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input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



NYU SAI LAB

Example

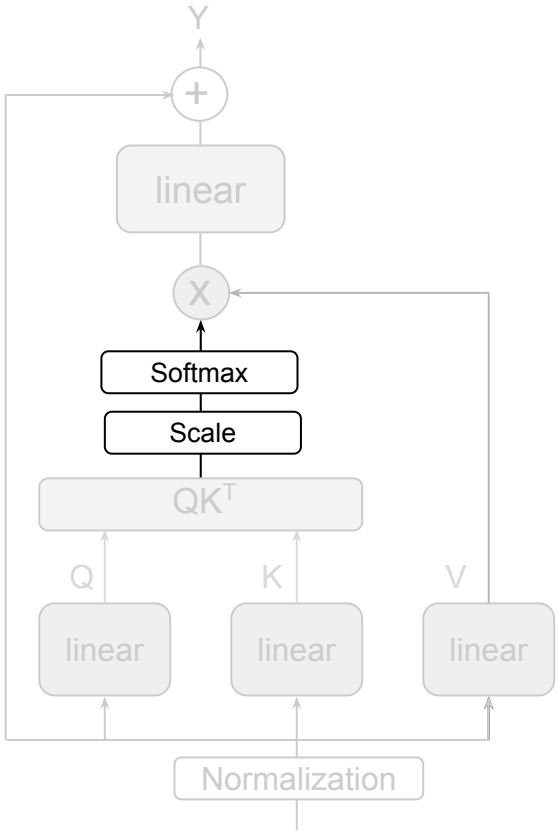




NYU SAI LAB

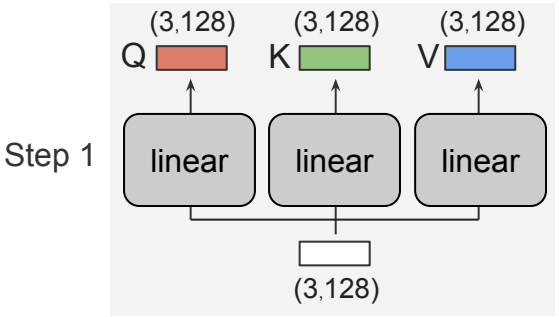
Self-Attention Block

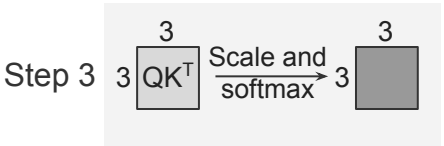
- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input 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.

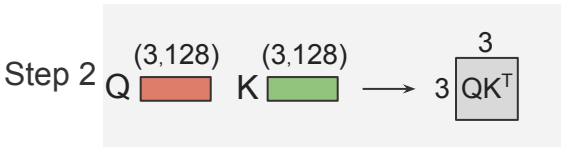


NYU SAI LAB

Example



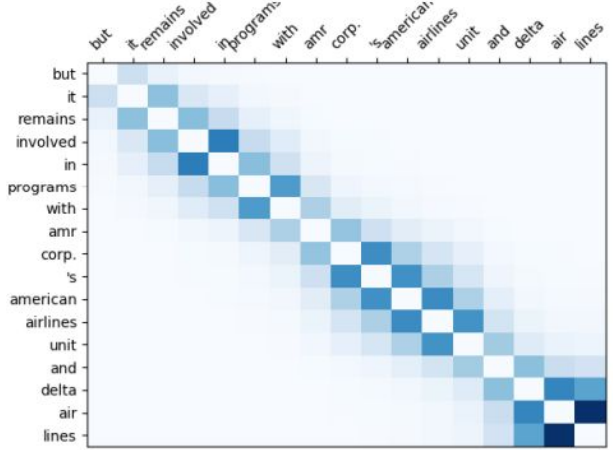




NYU SAI LAB

Self-Attention Block

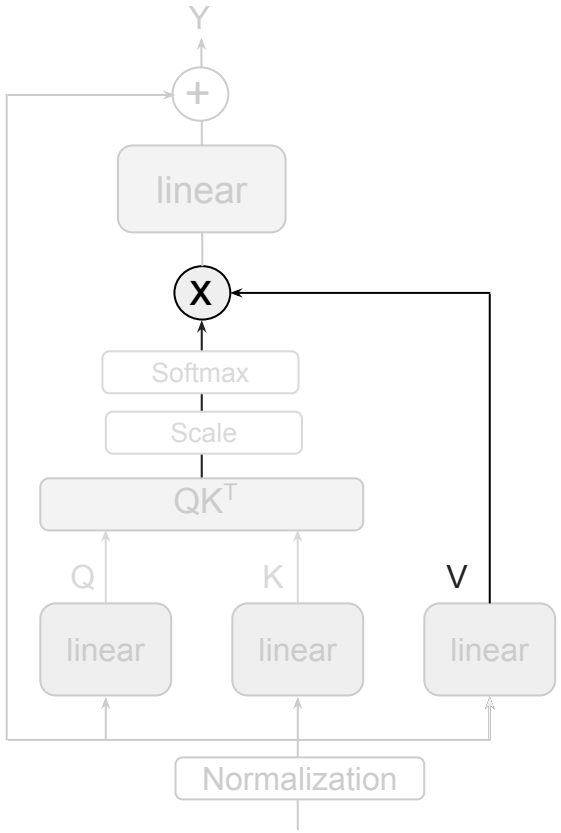
- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input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



NYU SAI LAB

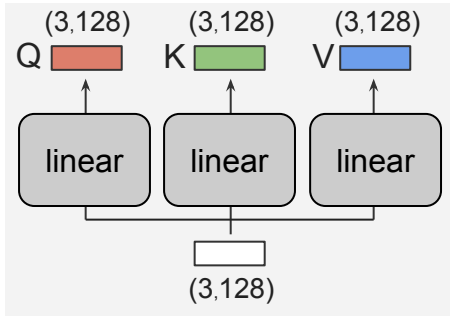
Self-Attention Block

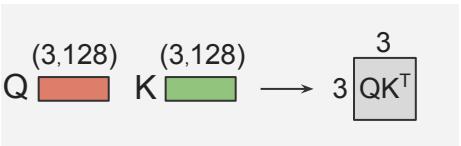
- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input 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.

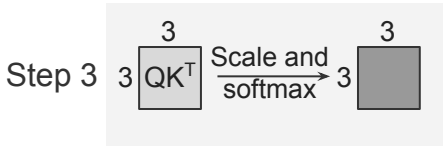


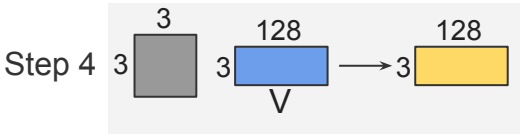
NYU SAI LAB

Example





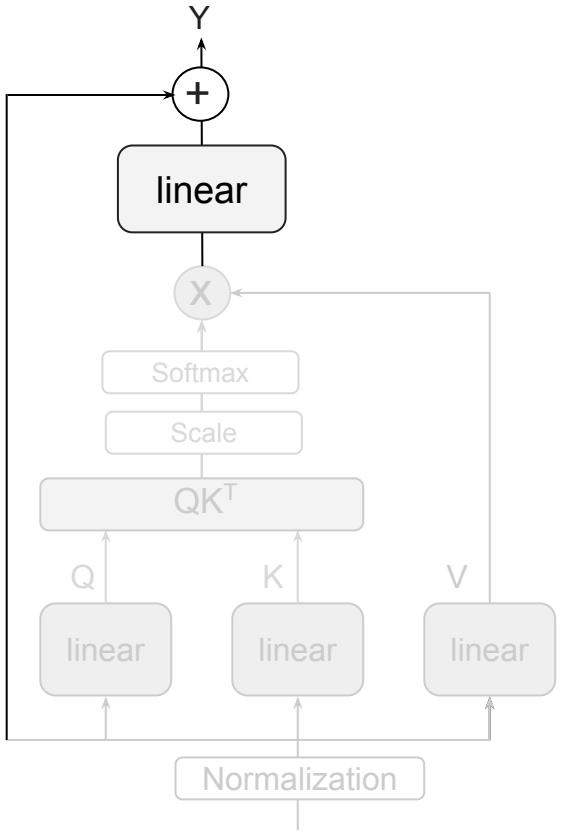




NYU SAI LAB

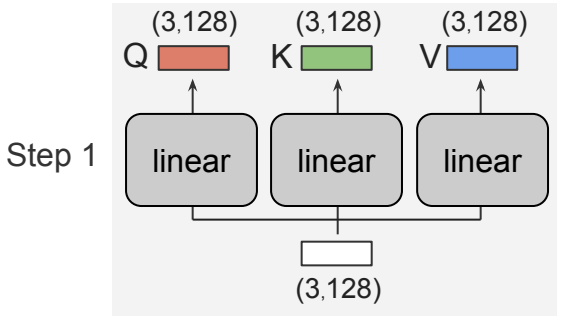
Self-Attention Block

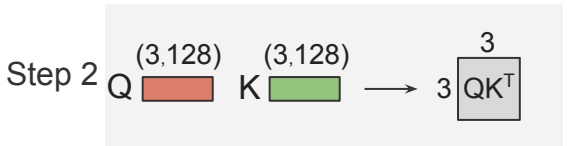
- Given input x, the first step in calculating self-attention is to create three vectors from each of the input x', denoted as: Query (Q), Key (K), Value (V).
  - (B,L,E) ✖ (E✖E) → (B✖L✖E)
- The second step in calculating self-attention. This will compute the attention score between each pair of input 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.

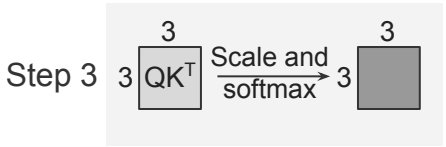


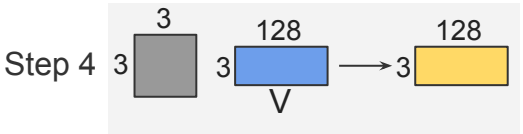
NYU SAI LAB

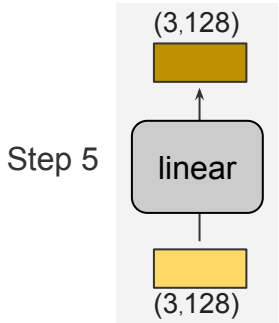
Example







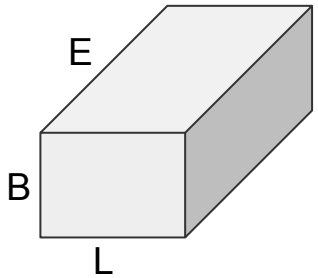


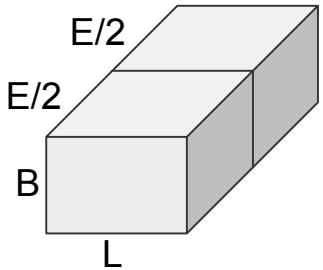


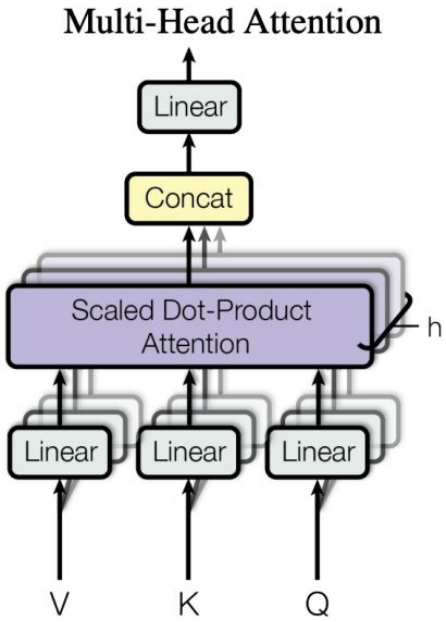
NYU SAI LAB

Multi-headed Attention

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

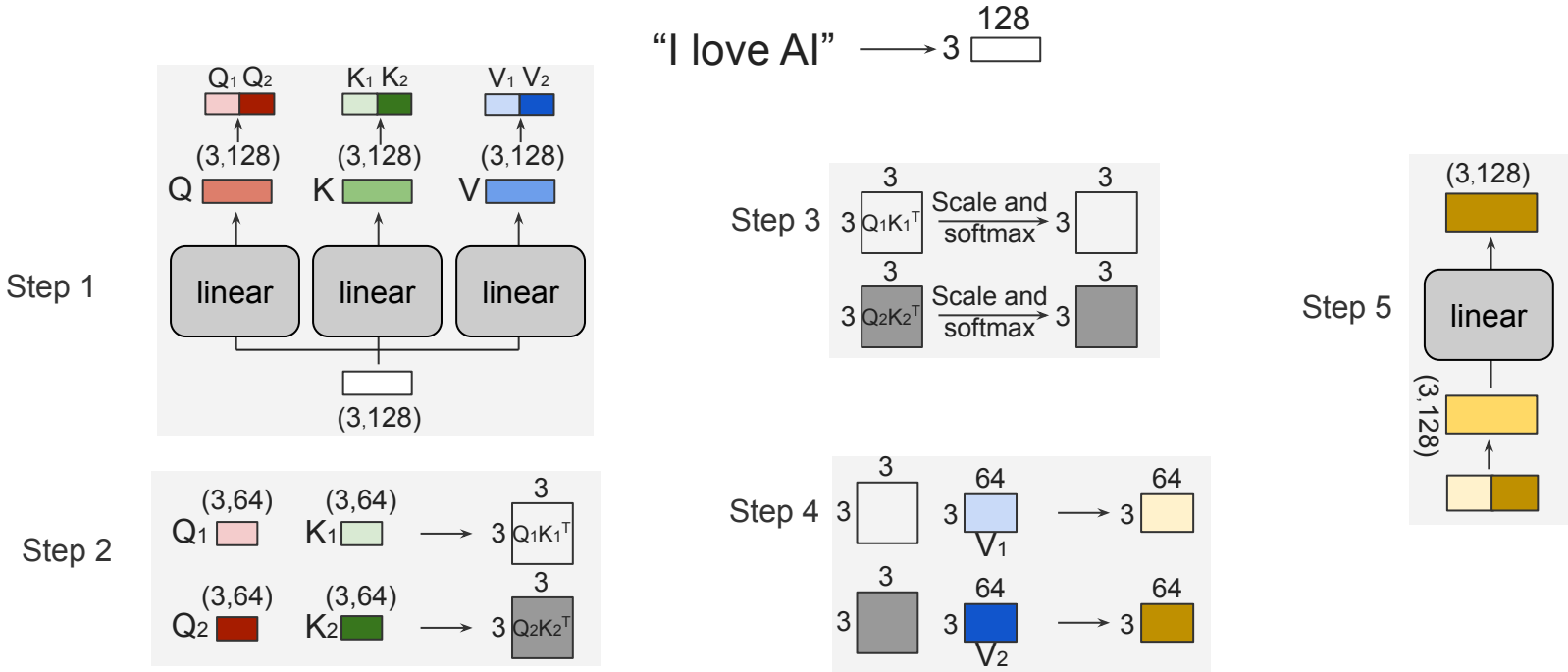






NYU SAI LAB

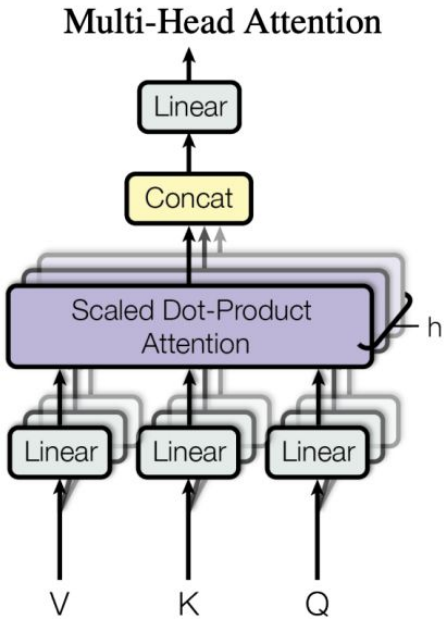
Example



NYU SAI LAB

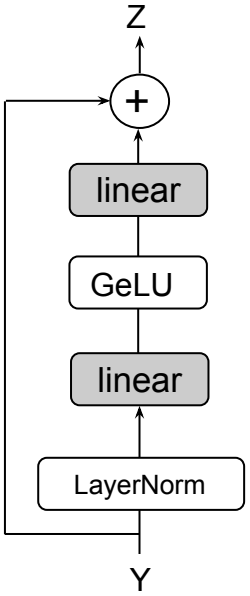
Multi-headed Attention

- 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.

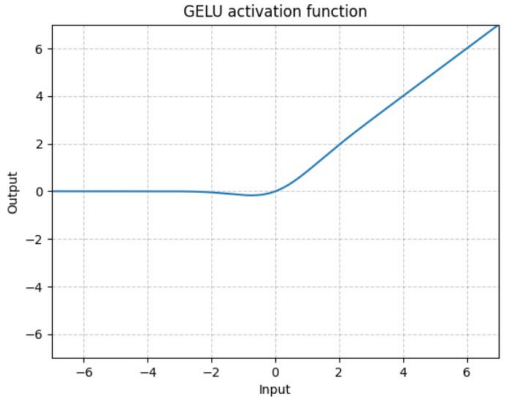


Michel, Paul, Omer Levy, and Graham Neubig. "Are sixteen heads really better than one?." *Advances in neural information processing systems* 32 (2019).

Feed Forward Layer

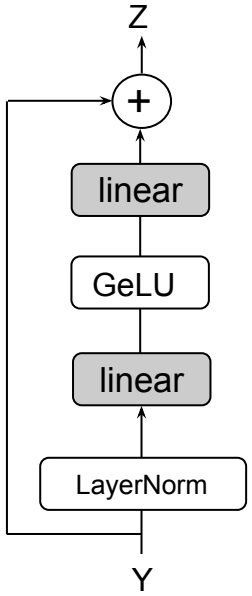


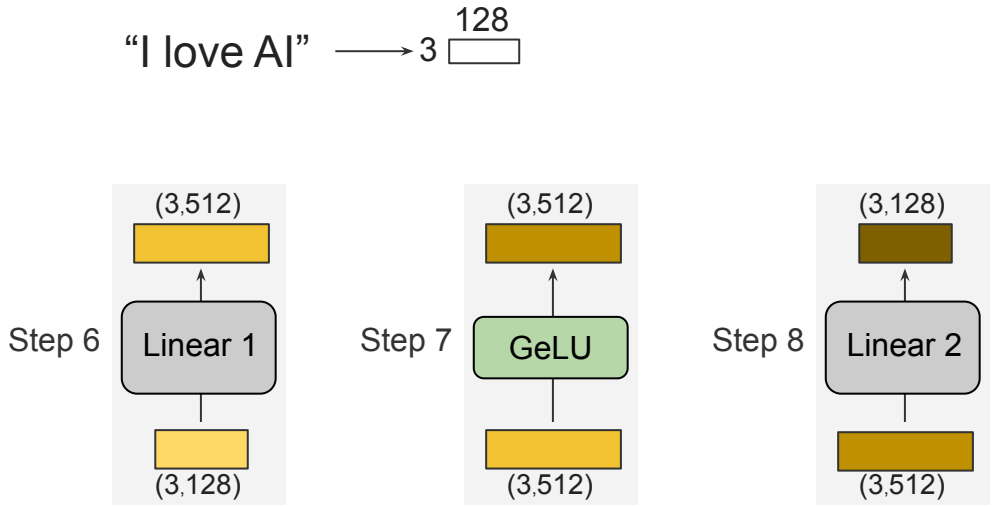
- The two linear layers are big:
  - (Ex4E) and (4ExE), E can be large (e.g., 4096)
  - This is expensive to implement.
- GeLU:
  - $GeLU(x) = x\Phi(x)$
    $\Phi(x) = P(y \leq x), \text{ where } Y \sim N(0,1)$



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

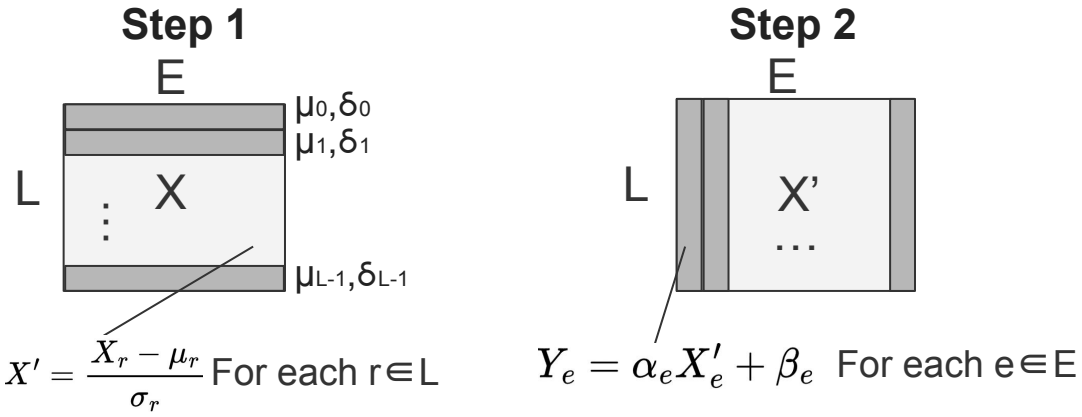
Example





NYU SAI LAB

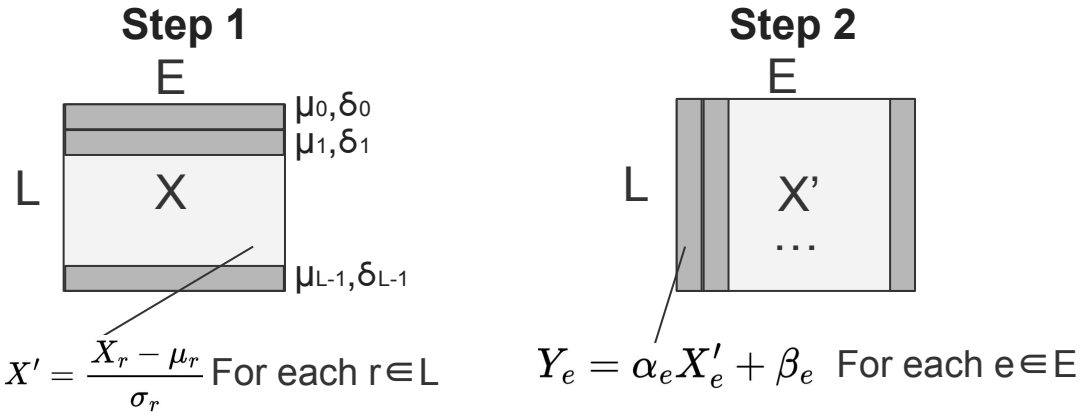
Layer Normalization



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

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

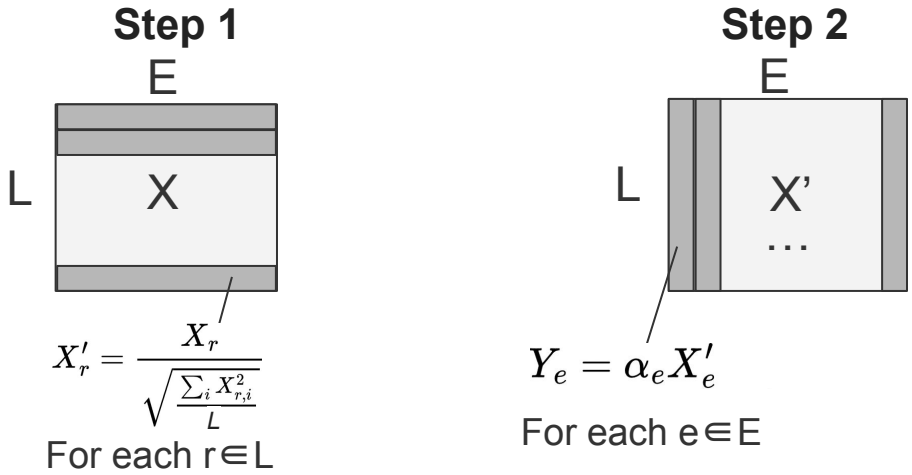
Layer Normalization



- 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



- 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

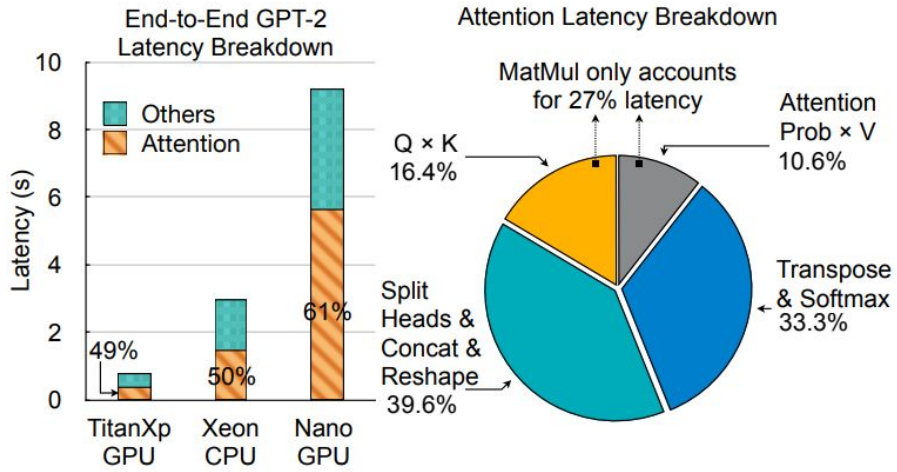


Fig. 2. End-to-End GPT-2 latency breakdown on various platforms, and attention latency breakdown on TITAN Xp GPU. Attention accounts for over 50% of total latency. Data movements account for 73% of attention latency.

Reshaping operation

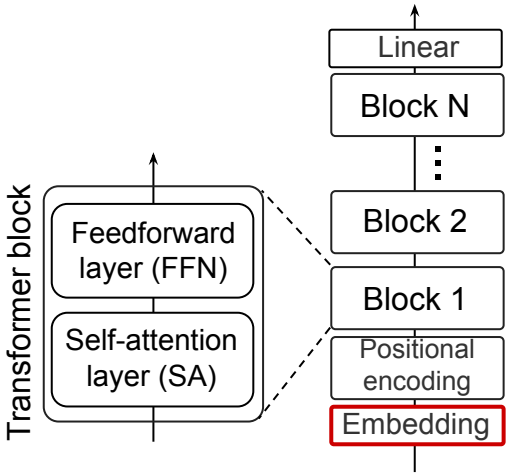
(B, L, E) ⟶ (B, M, L, E/M)

Transpose operation

(B, L, E) ⟶ (B, E, L)

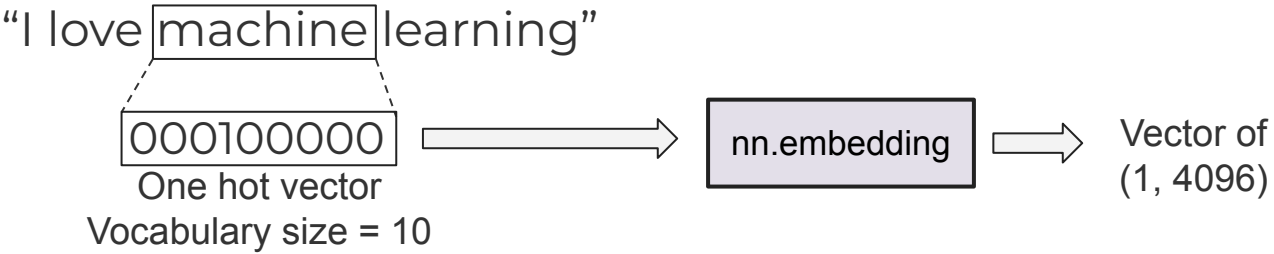
Wang, Hanrui, Zhekai Zhang, and Song Han. "Spatten: Efficient sparse attention architecture with cascade token and head pruning." *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021.

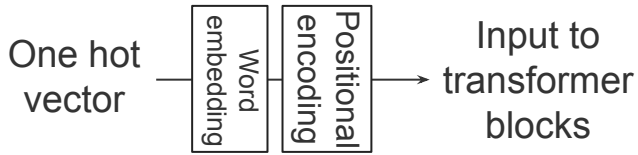
Transformers: Word Embedding



Transformers: Word Embedding

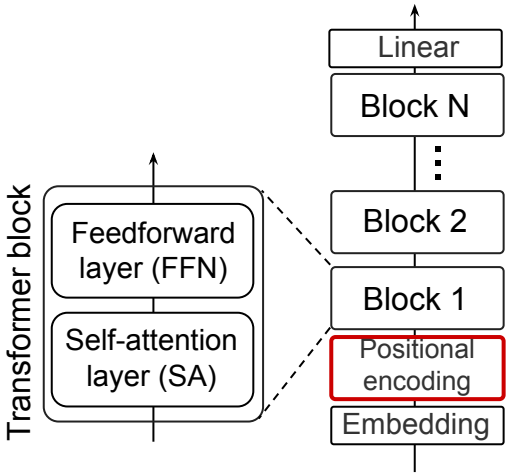
- For each word, we can convert them into a one-hot vector.
- We use the embedding layer to encode these one-hot vectors, which acts like a trainable lookup table.
- Dictionary: {are, is, machine, good, awesome, learning, love, ….}





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

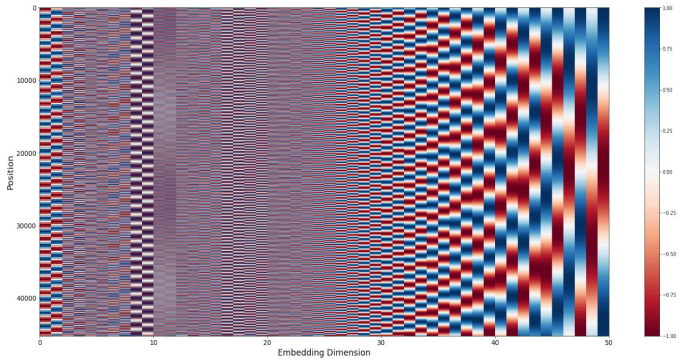
Transformers: Positional Encoding



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.

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

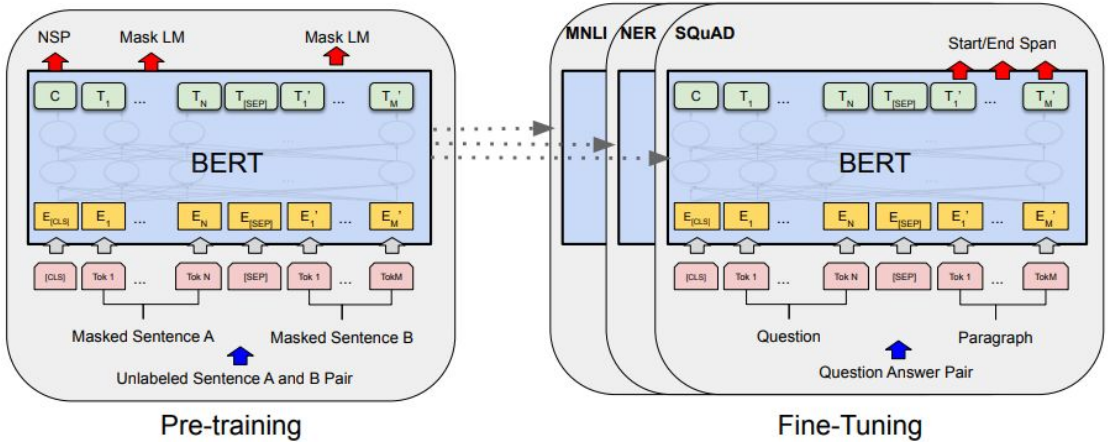


- For long sequences, the indices can become quite large. Normalizing these index values to a range between 0 and 1 can cause issues with variable-length sequences, as each sequence would be normalized differently.
- pos is the positional of the token, i is the index of the embedding.

NYU SAI LAB

Case Study: BERT

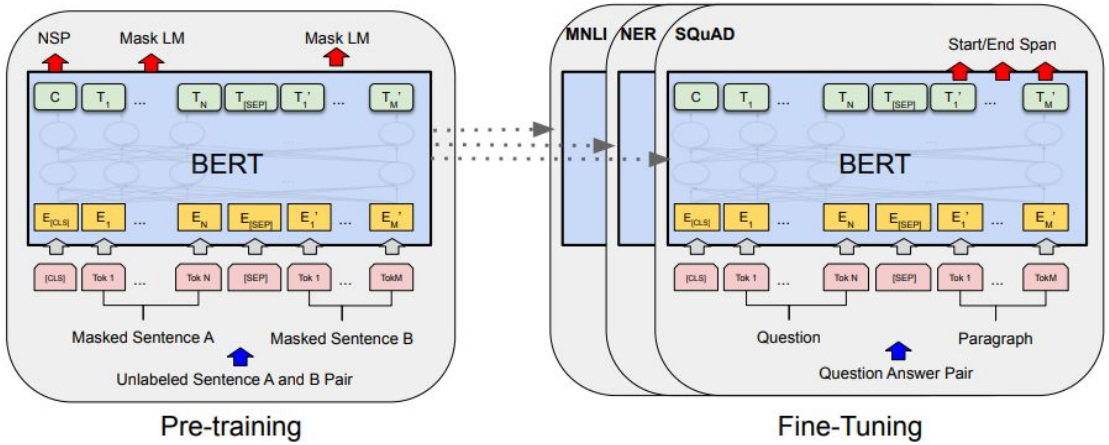
- Bidirectional Encoder Representations from Transformers.
- BERT is designed to understand the text by considering both the words before and after it.
- BERT consists of transformer encoders and takes the entire sentence as the input.
- BERT can not generate new text.



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.



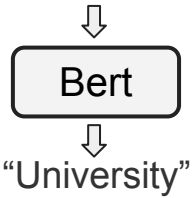
Devlin, Jacob. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

BERT Pretraining

- BERT is pretrained using two self-supervised 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.

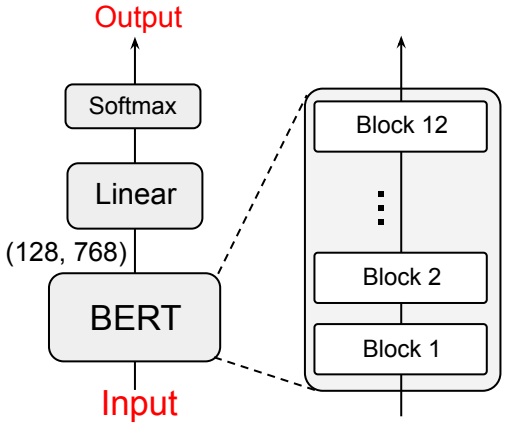
"New York University is a great school."

"New York University is a great school."

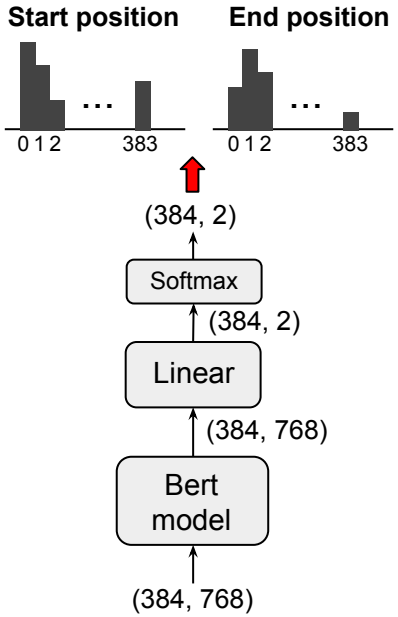


Downstream Tasks

- 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.



Downstream Tasks



Context paragraph

"Similarly, movies and television often revert to standard, clichéd snatches of classical music to convey refinement or opulence: some of the most-often heard pieces in this category include Bach´s Cello Suite No. 1, Mozart's Eine kleine Nachtmusik, Vivaldi's Four Seasons, Mussorgsky's Night on Bald Mountain (as orchestrated by Rimsky-Korsakov), and Rossini's William Tell Overture."

Question

"Who wrote William Tell Overture?"

Answer

Classical_music (12, 13)

- In Question Answering tasks, an input sample consists of a context paragraph and a question with a total length of 384.
- The goal is to find, for each question, a span of text in the context paragraph that answers that question.
- BERT produces the answers by generating the start and end positions of the text span.

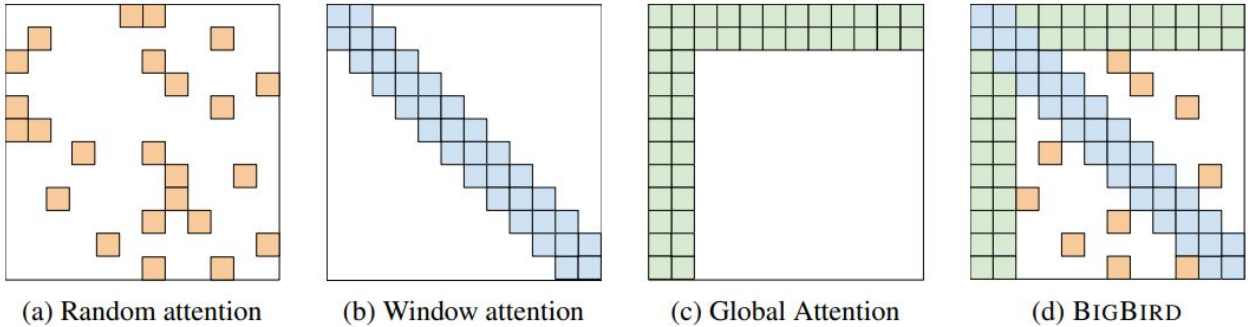
NYU SAI LAB

BERT Performance

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
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**

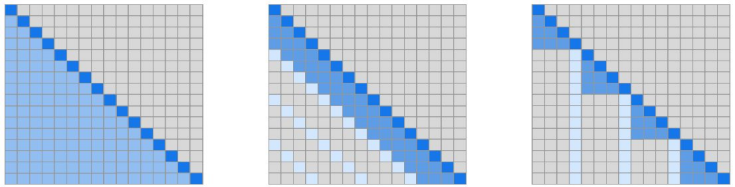
- BERT can achieve better performance over GLUE datasets than GPT-1.

Efficient Self-Attention Design



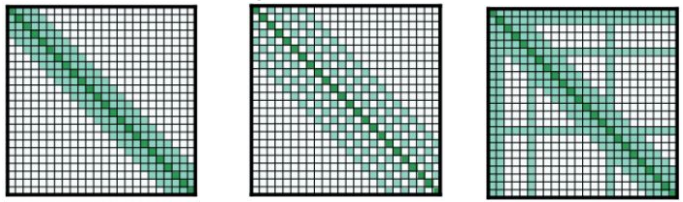
(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Sparse Transformer (2019)



Original Strided Fixed

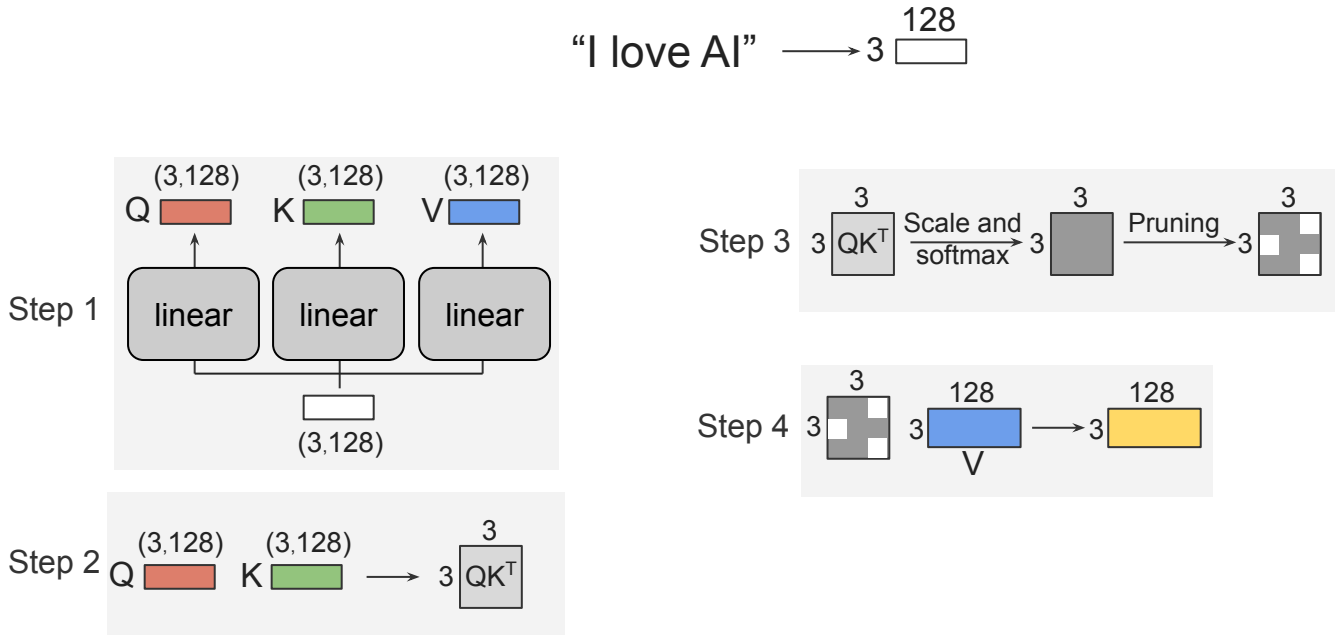
Longformer (2020)



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

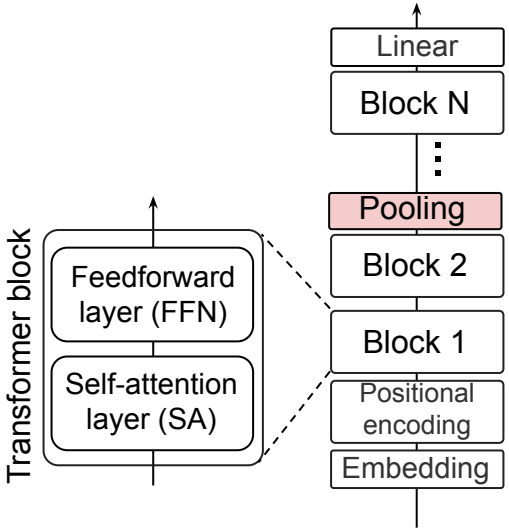
Child, Rewon, et al. "Generating long sequences with sparse transformers." *arXiv preprint arXiv:1904.10509* (2019).

Efficient Self-Attention Design

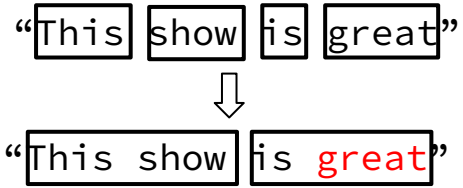


NYU SAI LAB

Token Merging



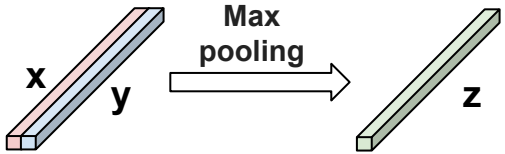
Pooling layer will reduce the number of tokens.



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

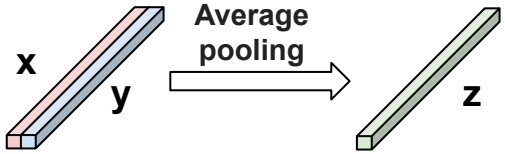
Bolya, Daniel, et al. "Token merging: Your vit but faster." *arXiv preprint arXiv:2210.09461* (2022).

Token Merging



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

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



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

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

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

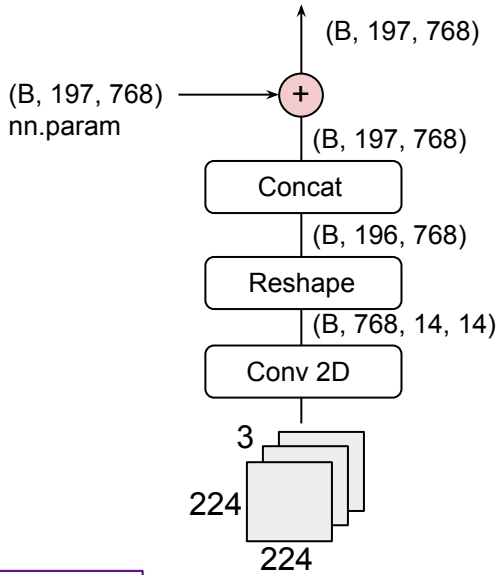
Topics

- Transformer basics
- Bert
- Vision transformer
- Large Language Model

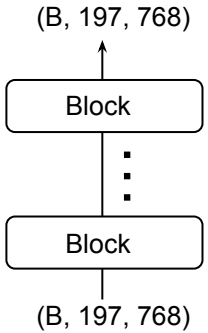
NYU SAI LAB

Vision Transformer

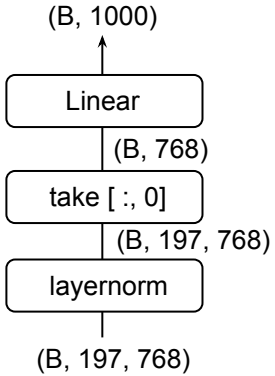
- Transformer architecture can also be applied over the computer vision tasks.



(part 1)



(part 2)

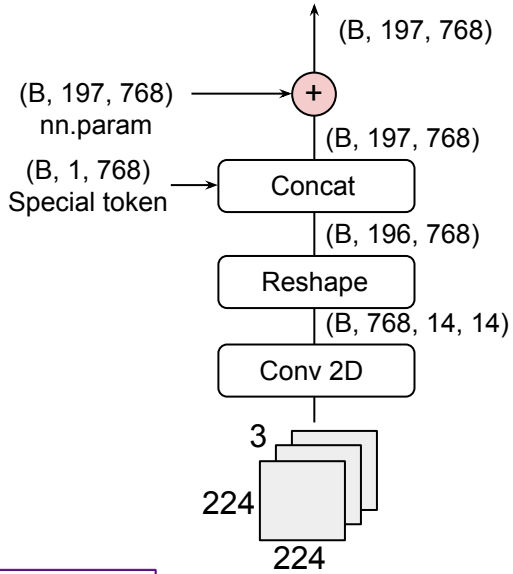


(part 3)

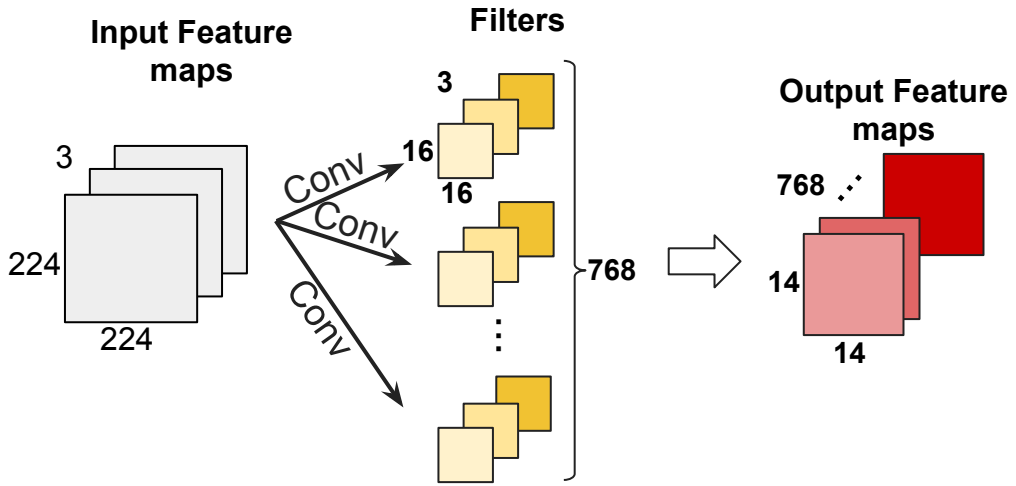
NYU SAI LAB

Vision Transformer

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



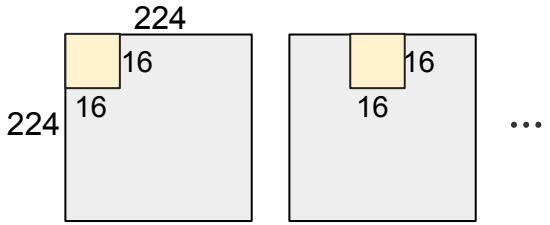
(part 1)

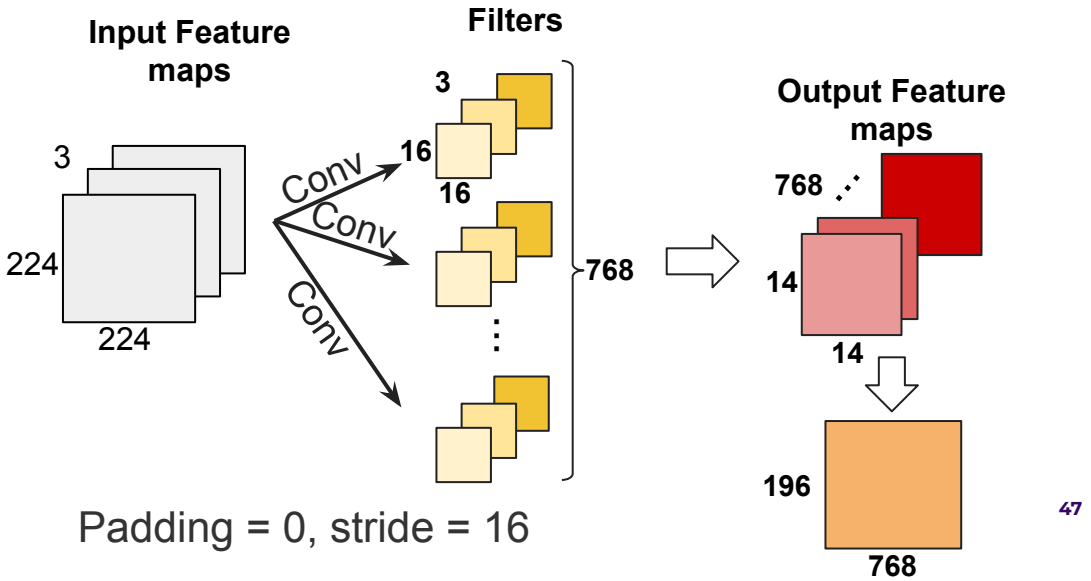


Padding = 0, stride = 16

Vision Transformer

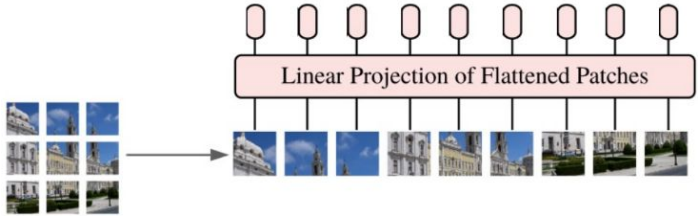
- Transformer architecture can also be applied over the computer vision tasks.

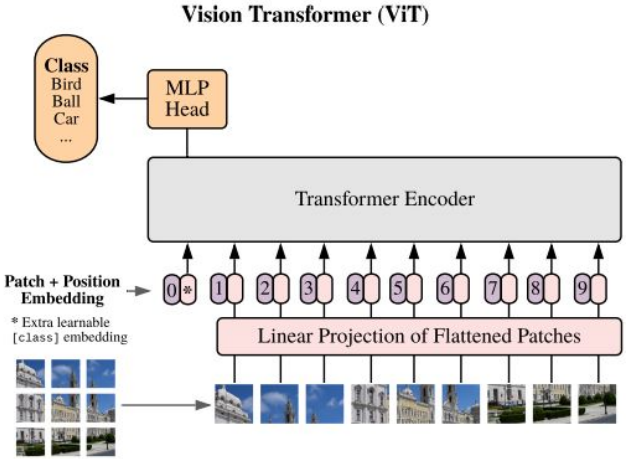




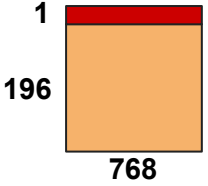
Vision Transformer

- 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.





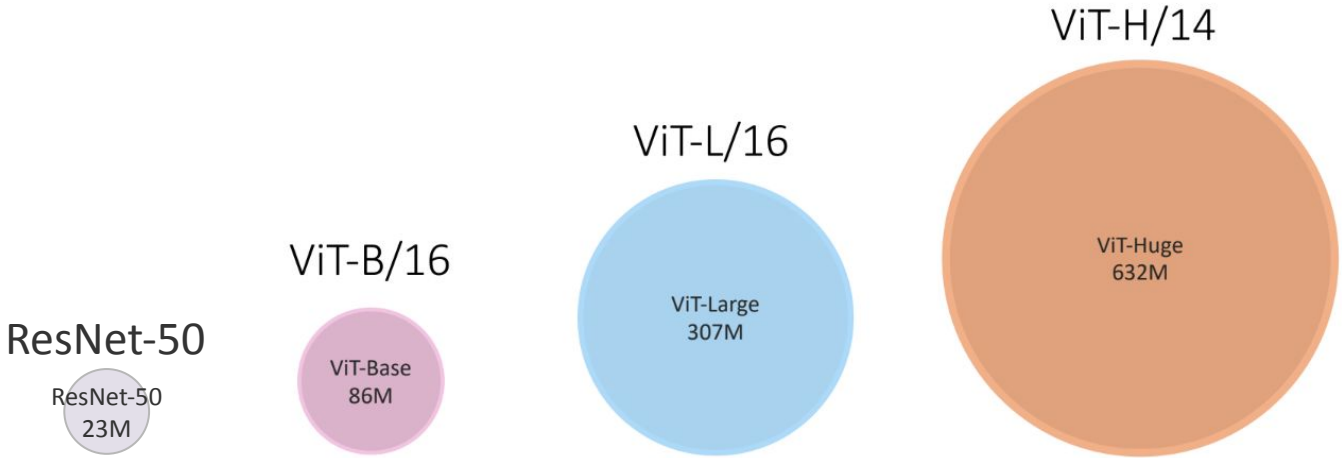
- A classification [CLS] token is inserted at the start of the tokens.



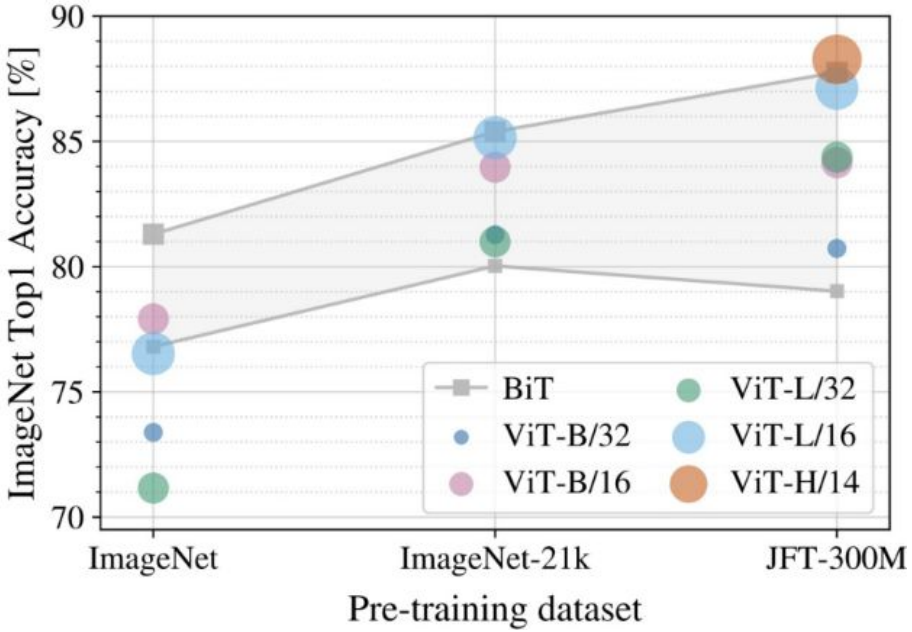
NYU SAI LAB

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.

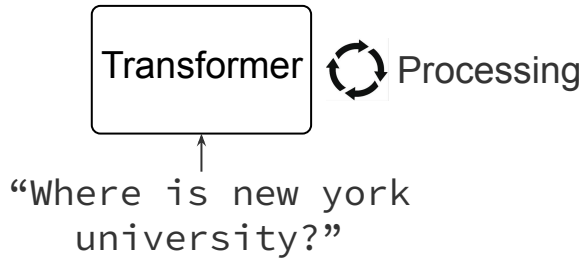
NYU SAI LAB

Topics

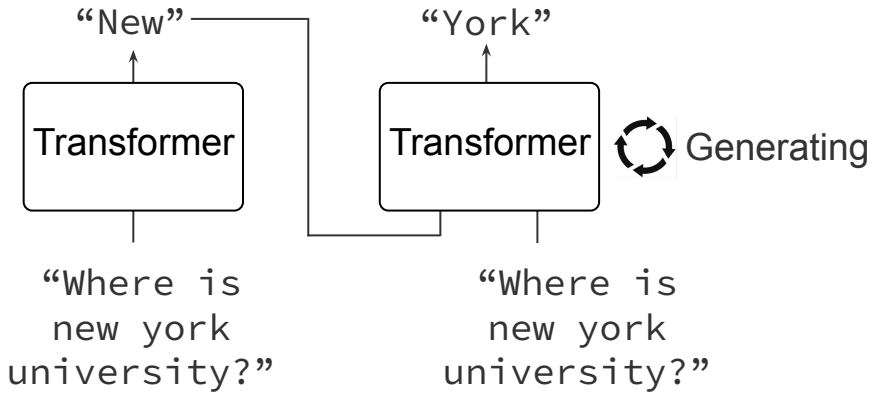
- Transformer basics
- Bert
- Vision transformer
- Large Language Model

NYU SAI LAB

Transformers as a Generative AI Tool



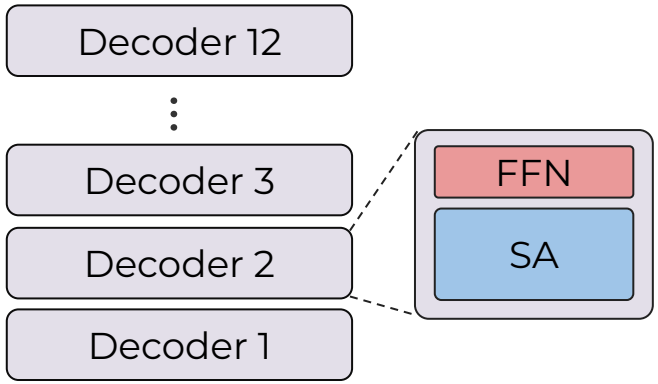
Step 1: Prefilling

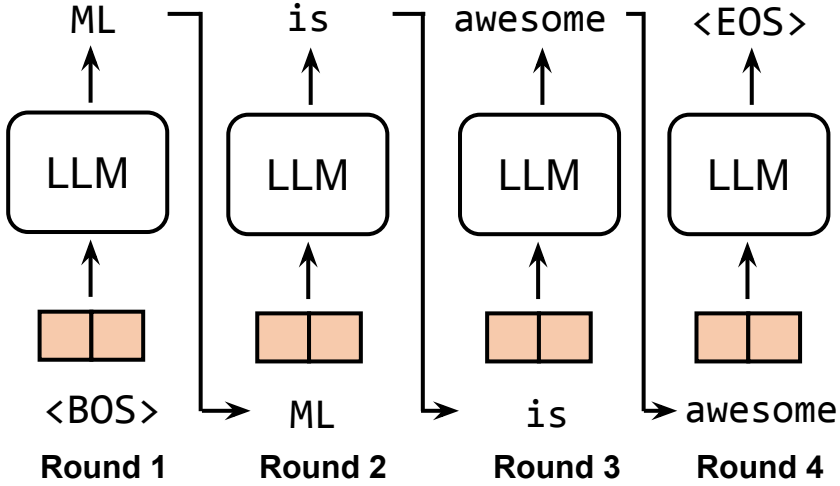


Step 2: Decoding

NYU SAI LAB

Transformers as a Generative AI Tool

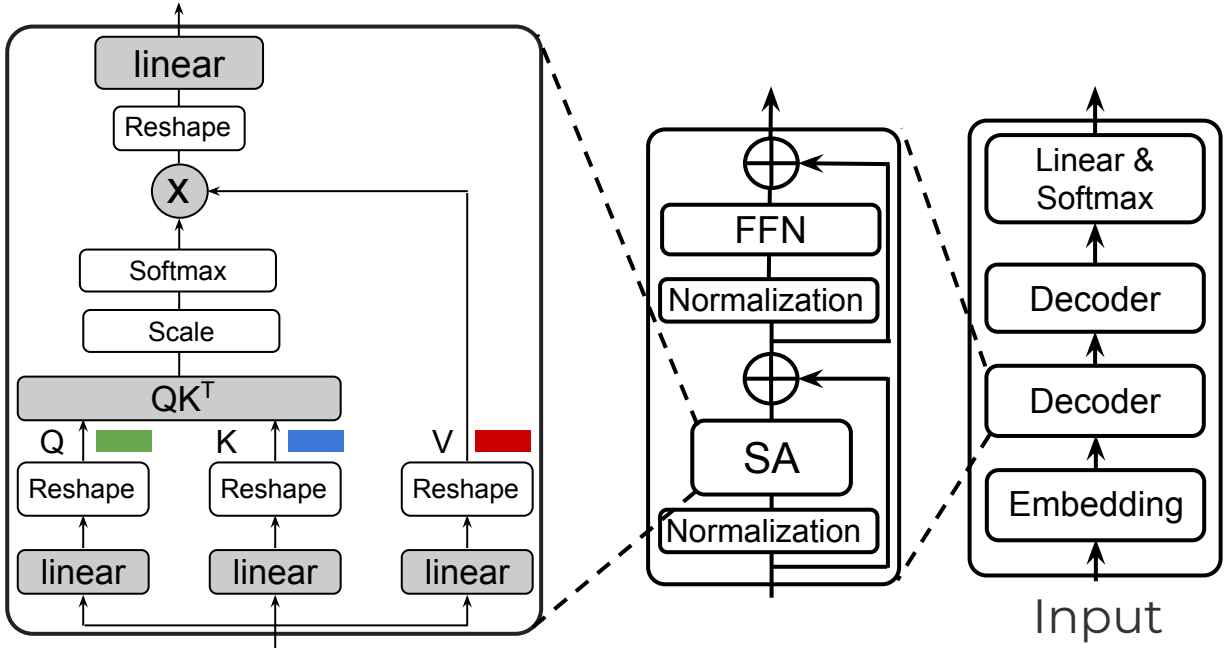




- Each token is generated in an autoregressive manner.

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

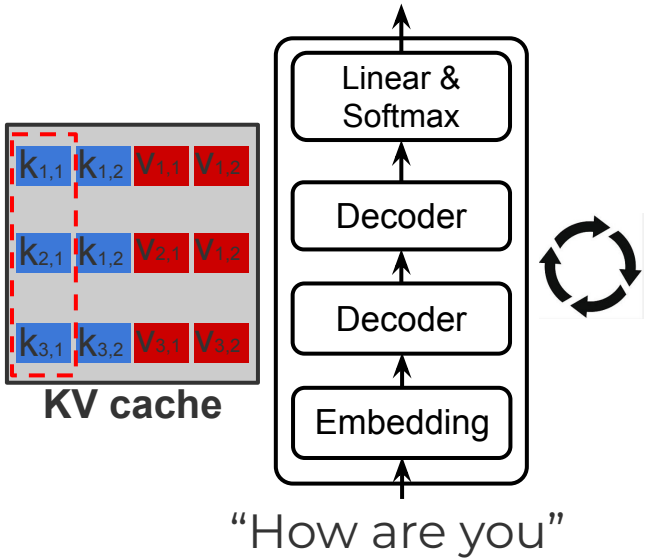
Transformers as a Generative AI Tool



NYU SAI LAB

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

GPT-2: Prefilling



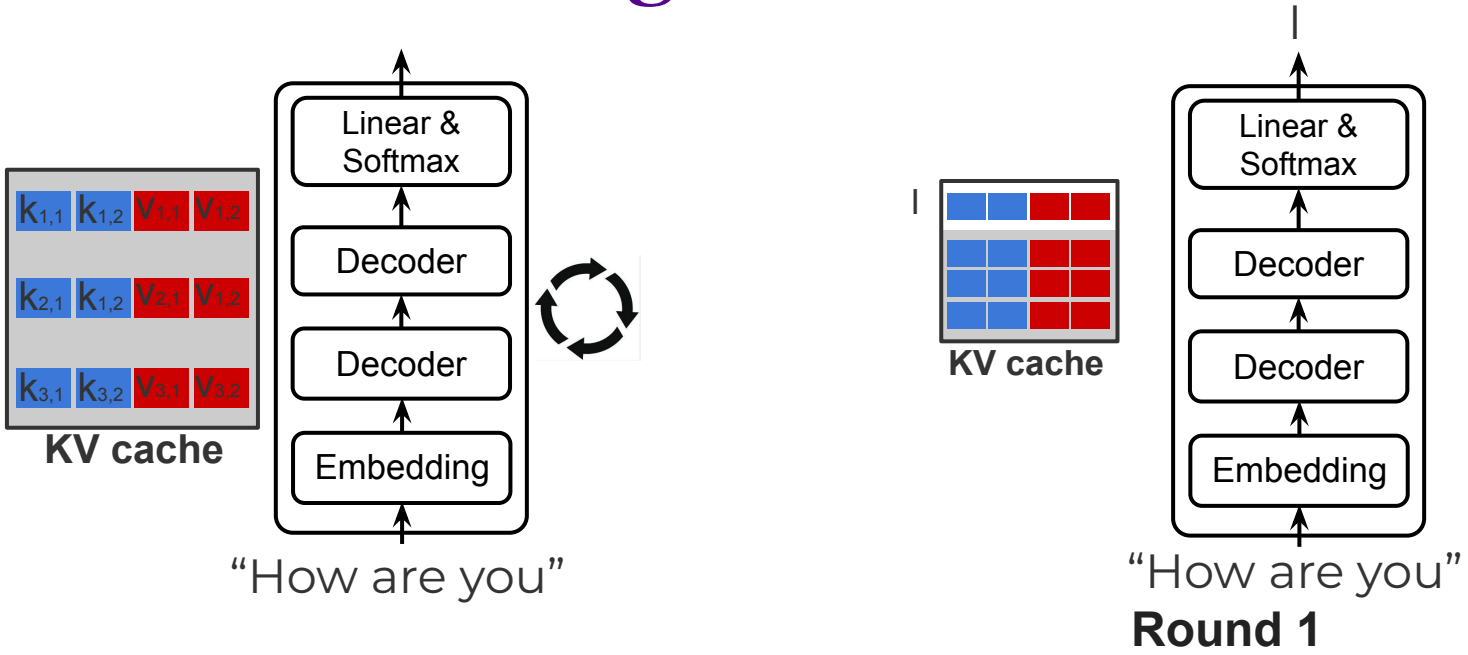
$k_{i,j}$ Key vector for i^{th} token in j^{th} layer
$(1 \times E)$

$v_{i,j}$ Value vector for i^{th} token in j^{th} layer
$(1 \times E$
$)$

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

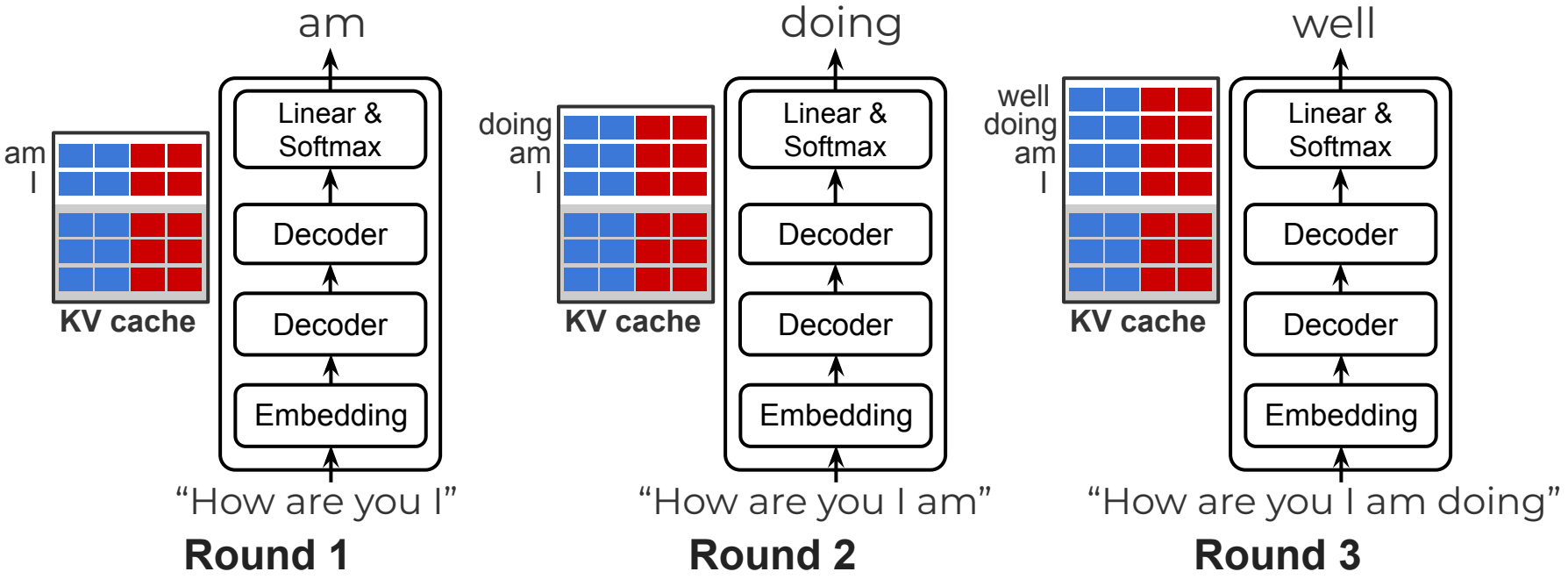
NYU SAI LAB

GPT-2: Decoding



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

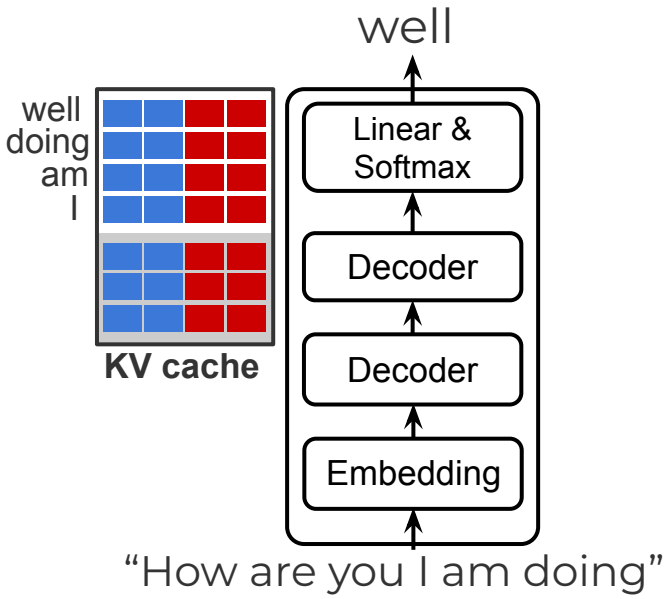
GPT-2: Decoding

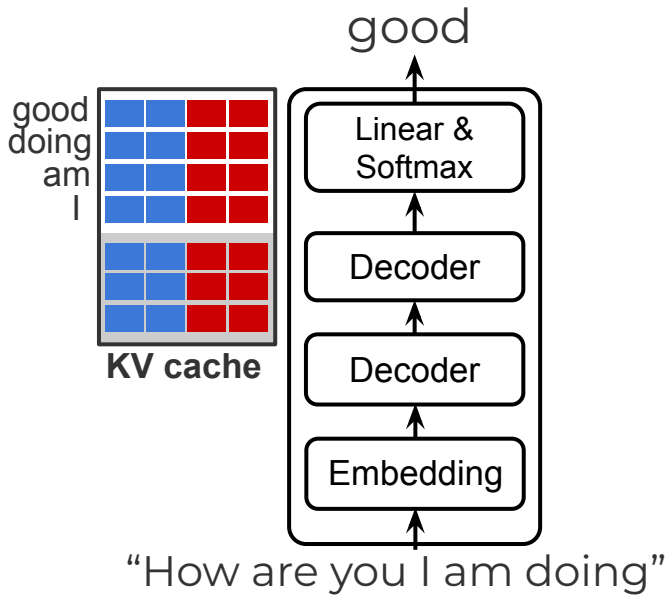


NYU SAI LAB

- The storage complexity scales linearly with the number of tokens.

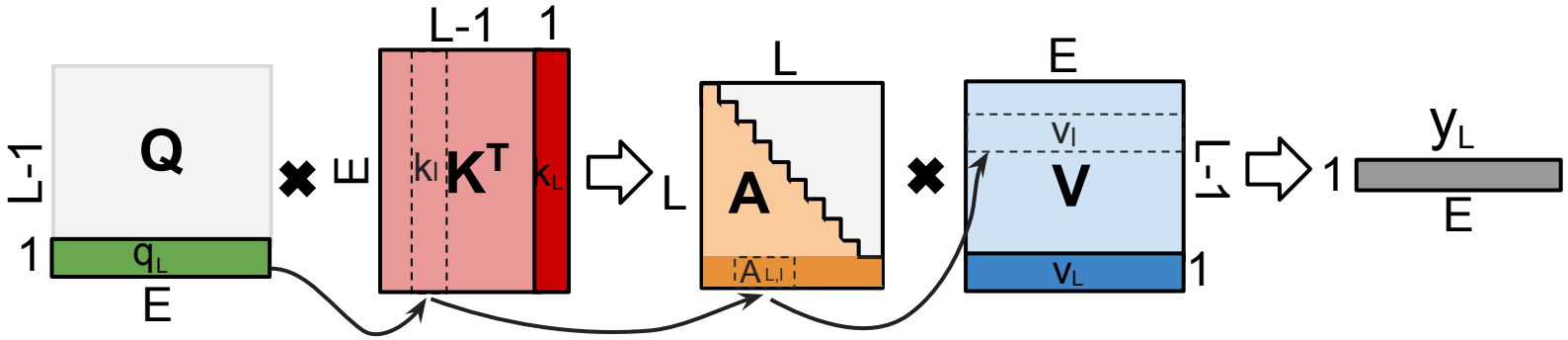
GPT-2: Decoding





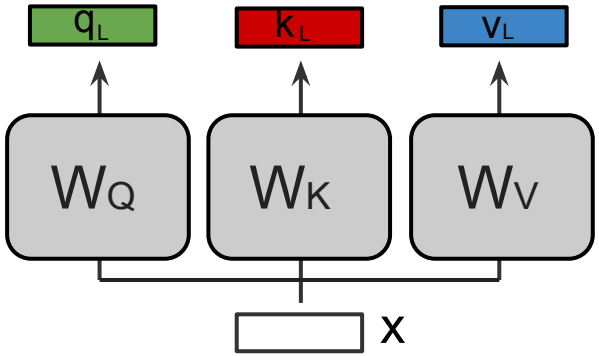
- We can simply select the token with the highest score. But better results are achieved if the model considers other words as well. So a better strategy is to sample a word from the entire list using the score as the probability of selecting that word.

Why KV Cache Saves Computation?



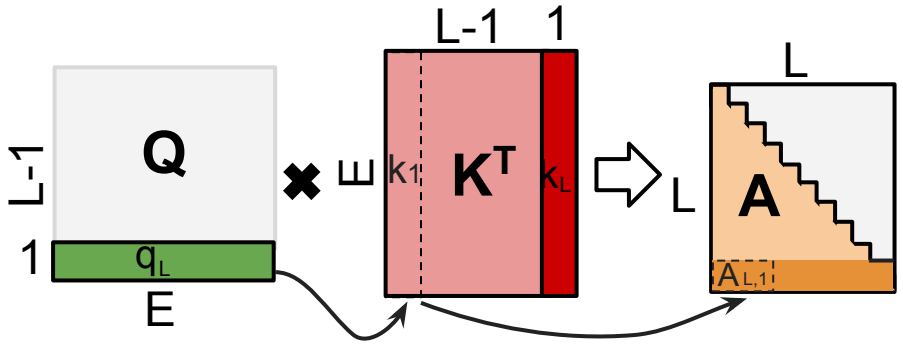
- During the decoding phase, new tokens are continuously generated and must be processed using the buffered K and V vectors to generate subsequent tokens.
- Without a KV cache, all previous K and V vectors must be recomputed, resulting in significant computational overhead.

Why KV Cache Saves Computation?



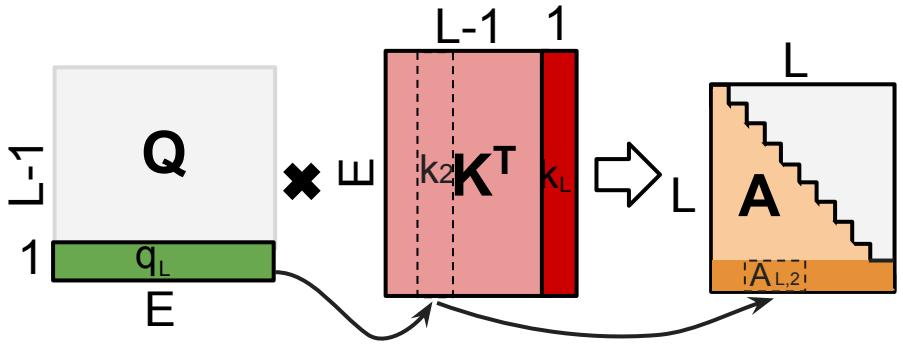
- Given the input, the q_L, k_L, v_L are first computed by passing through the linear layers.
- After that, the k_l, v_l vectors (l=1,...,L-1) are also loaded from the memory.

Why KV Cache Saves Computation?



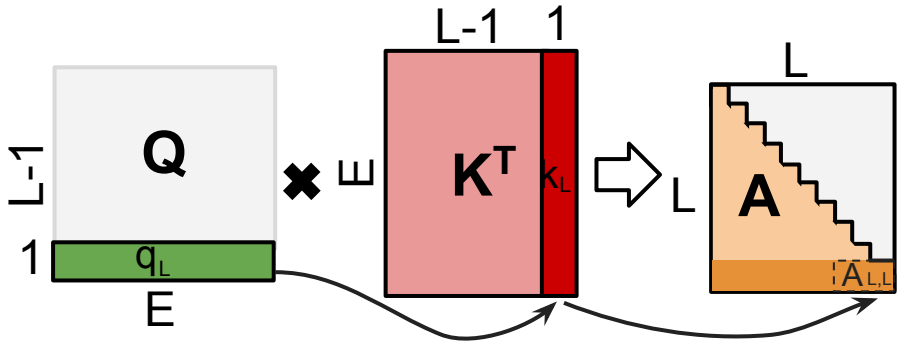
- K and V are loaded from the memory, the q vector of the current token (q_L) is multiplied with the each of the key vector k_i, (i=1…L) to produce the result $A_{L,i}$

Why KV Cache Saves Computation?



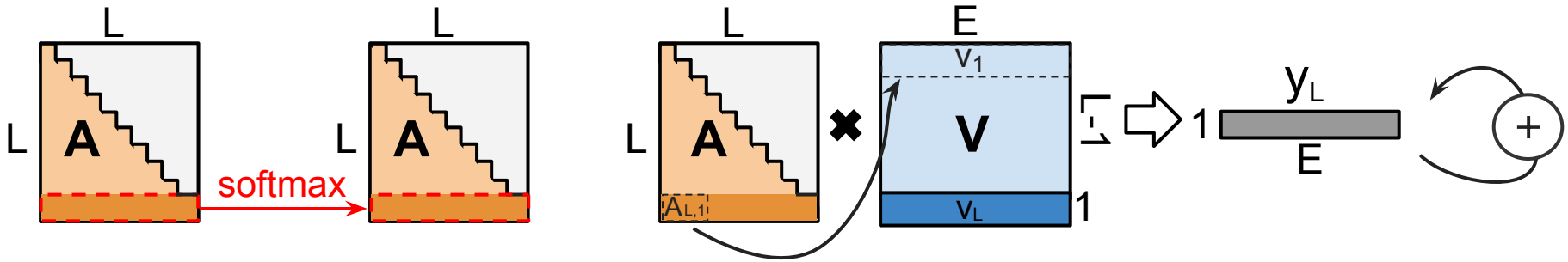
- K and V are loaded from the memory, the q vector of the current token (q_L) is multiplied with the each of the key vector k_i, (i=1…L) to produce the result $A_{L,i}$

Why KV Cache Saves Computation?



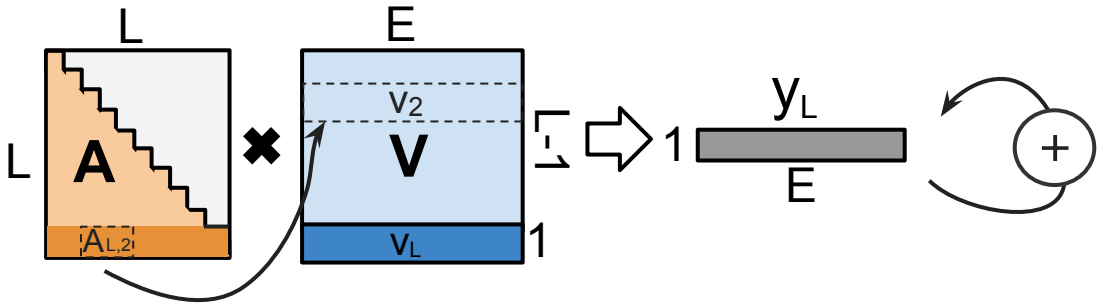
- K and V are loaded from the memory, the q vector of the current token (q_L) is multiplied with the each of the key vector k_i, (i=1…L) to produce the result $A_{L,i}$

Why KV Cache Saves Computation?



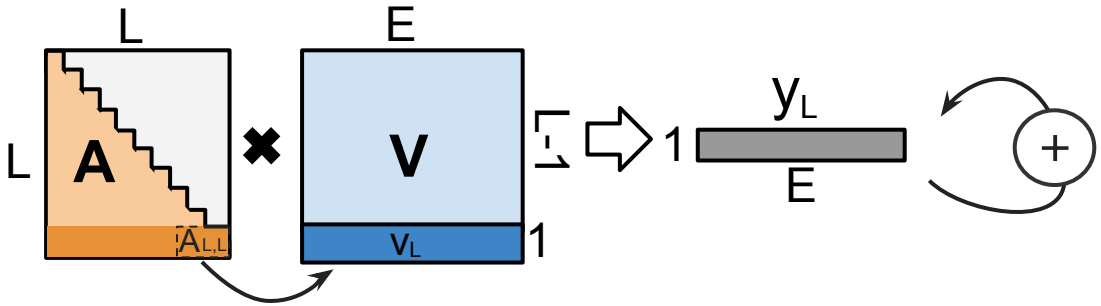
- Afterwards, the computed $A_{L,i}$ (i=1…L) will then passed to softmax function.
- Then each element of A_L will then multiplied with vi (i=1…L) and elementwise sum together.

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

Linear Attention

- The storage cost of KV cache grows linearly with the number of tokens.
- The computational cost grows quadratically with the number of tokens.

- The quadratic computation can be approximated with linear-complexity computation and constant storage overhead.

$$\text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V \Rightarrow V_i' = \frac{\sum_{j=1}^{N} \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)} \Rightarrow V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)} \Rightarrow V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}$$

$$\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

Const storage
Linear computational cost

Katharopoulos, Angelos, et al. "Transformers are rnns: Fast autoregressive transformers with linear attention." *International conference on machine learning*. PMLR, 2020.

Linear Attention

$$\text{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V \Rightarrow V_i' = \frac{\sum_{j=1}^{N} \text{sim}(Q_i, K_j) V_j}{\sum_{j=1}^{N} \text{sim}(Q_i, K_j)} \Rightarrow V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)} \Rightarrow V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}$$

$$\text{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

Const storage
Linear computational cost

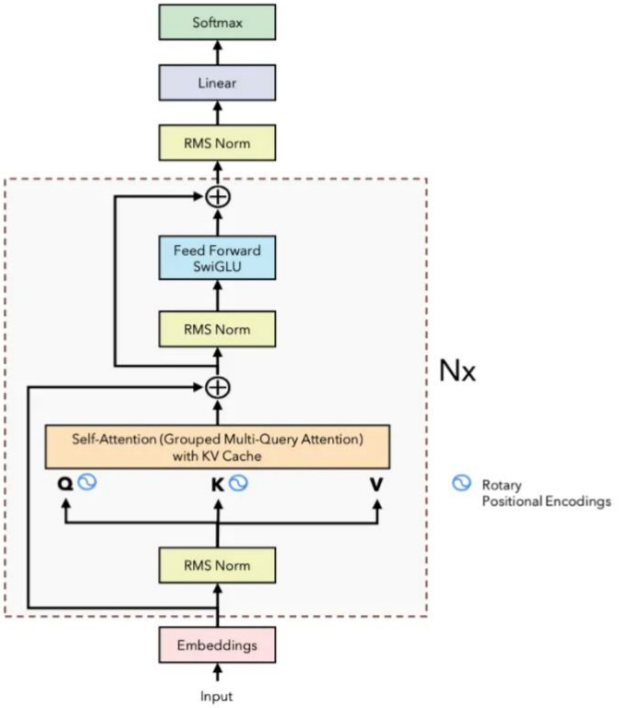
- Conventional self-attention uses softmax normalization, so each normalized attention weight depends on all keys for the same query.
- Linear attention replaces the softmax-based similarity $\phi(Q_i)^T\phi(K_j)$ with a kernel form $\phi(.)$

$$\phi(Q_i) \in \mathcal{R}^{E\times 1} \qquad \phi(K_j) \in \mathcal{R}^{E\times 1} \qquad V_j \in \mathcal{R}^{E\times 1}$$

Katharopoulos, Angelos, et al. "Transformers are rnns: Fast autoregressive transformers with linear attention." *International conference on machine learning*. PMLR, 2020.

LLaMA

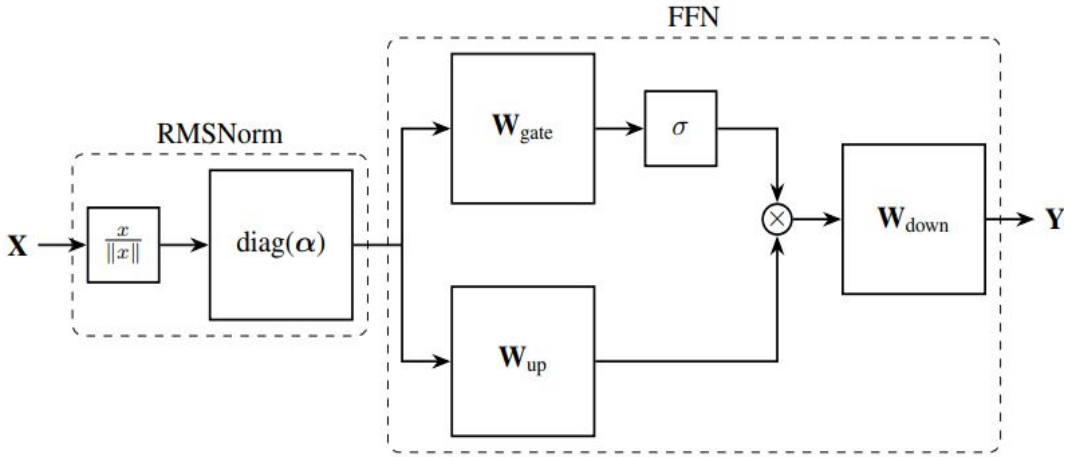
- LLaMA has a similar architecture as GPT-2, with some minor differences:
  - RMSNorm is used to replace LayerNorm
  - SwiGLU
  - MLPs with gating



LLaMA

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."

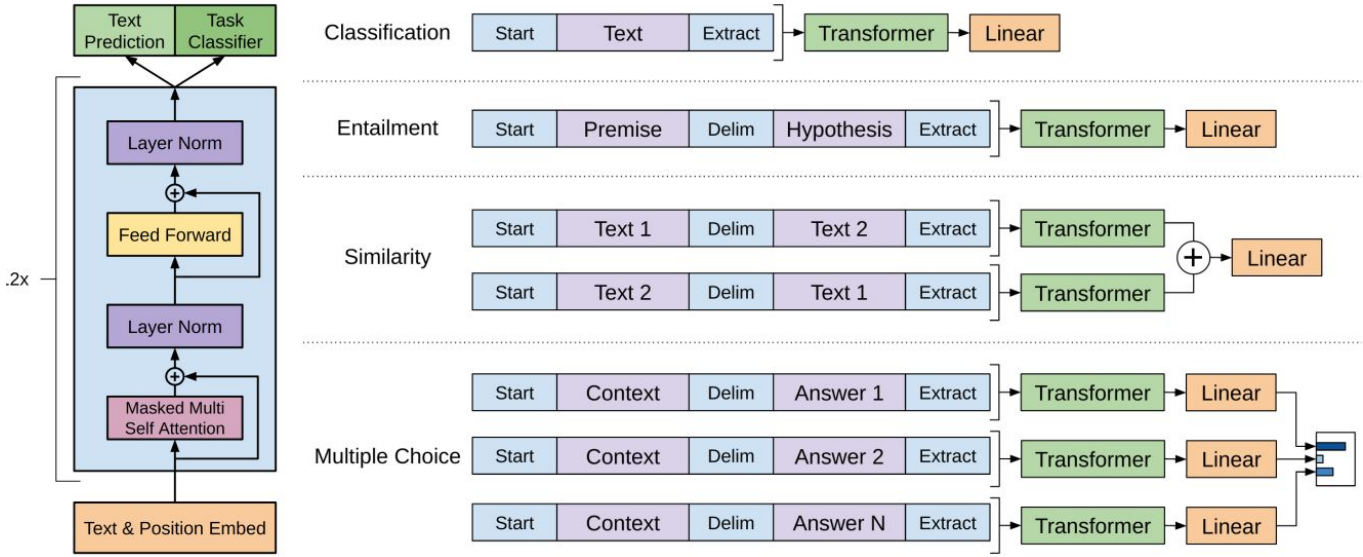
A newspaper article should contain these five main **xxx** ⟶ "components" (GPT)

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)

NYU SAI LAB

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).

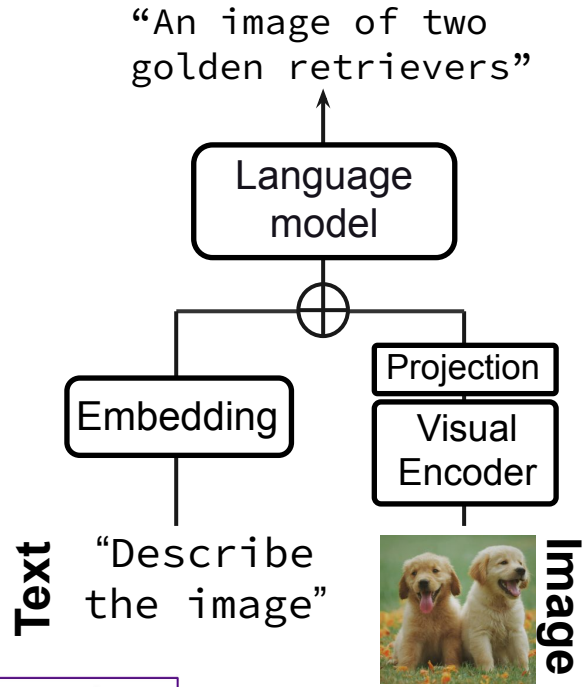
How LLM is Trained?



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

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

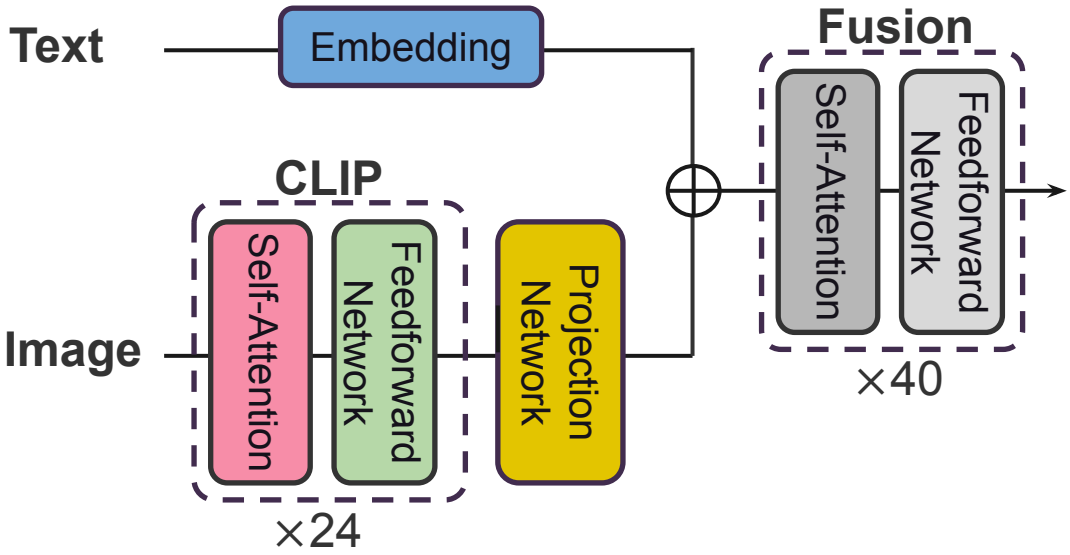
Vision Language Model



- 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.

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