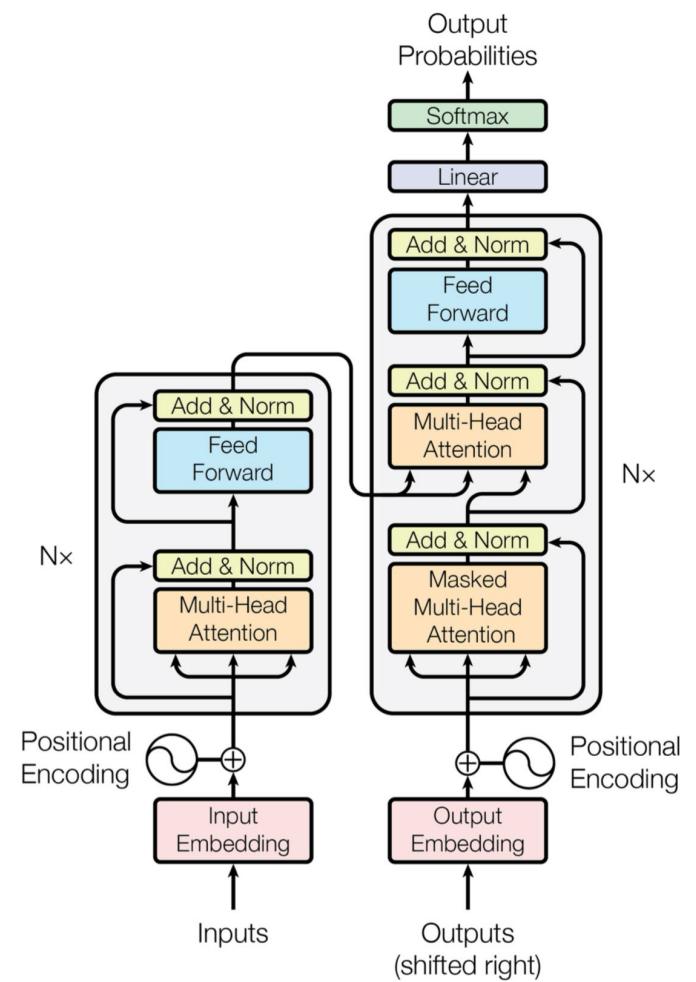


# Computer Vision

## Lecture 10b: Transformers In Vision



SAPIENZA  
UNIVERSITÀ DI ROMA



# Summary

- From RNNs to Transformers
- Transformer's attention and self-attention mechanism
- Transformer Encoder
- From Text to Images: Vision Transformers
- The scale and data problem
- Hybrid Transformers

# History

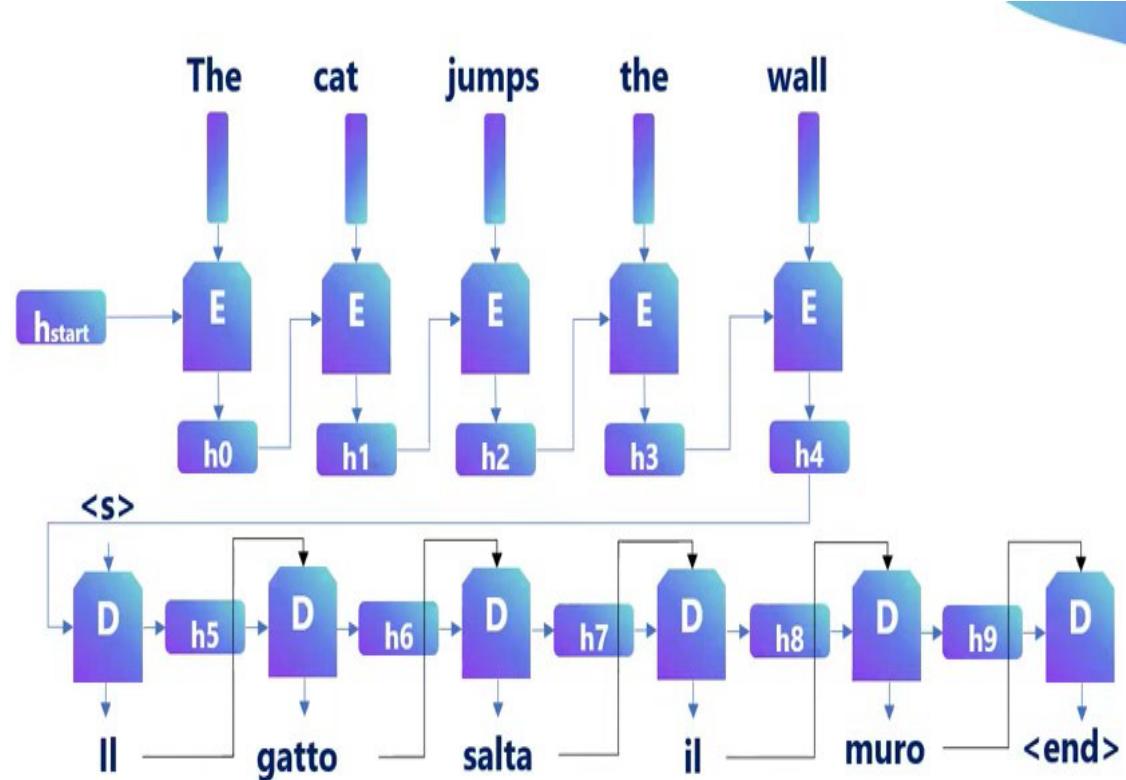
- 2017 Introduced transformers in NLP
- 2020 Vision Transformers
- 2021 Transformers for video understanding
- Now used everywhere!

# Recurrent Networks (RNNs)

- Recurrent models generate a sequence of hidden states  $h_t$ , as a function of the previous hidden state  $h_{t-1}$  and the input for position  $t$

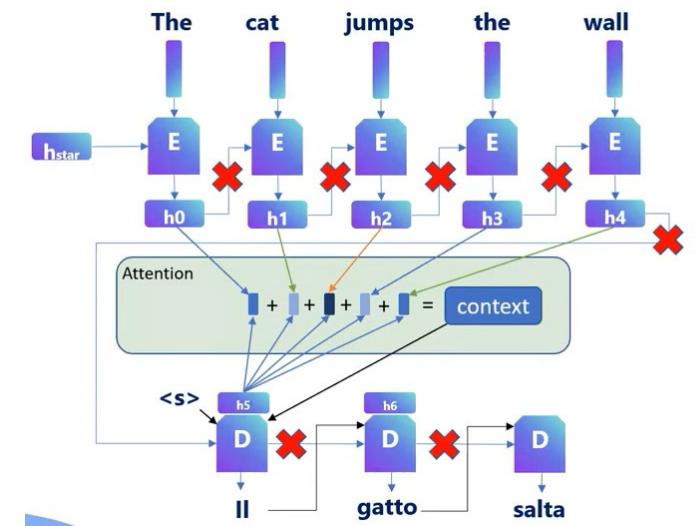
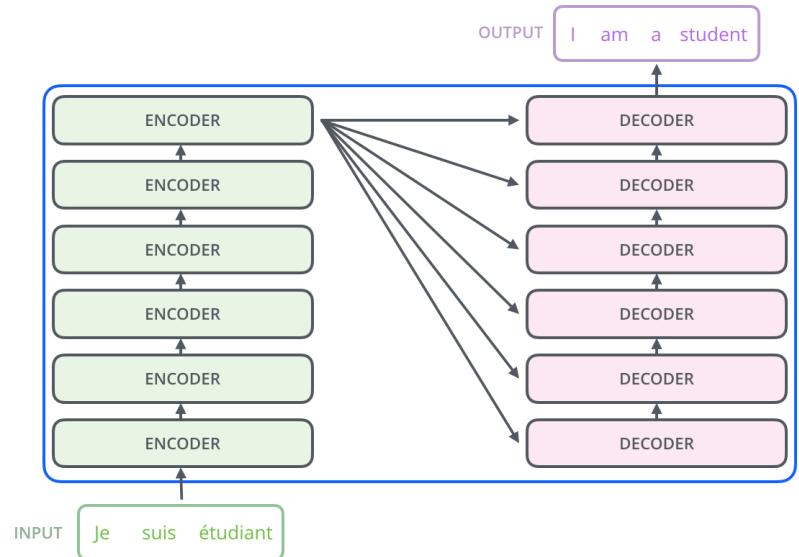
Problems:

1. We forget tokens too far in the past
2. This inherently sequential nature **precludes parallelization** within training examples
  - Need to wait the previous token to compute the next hidden state

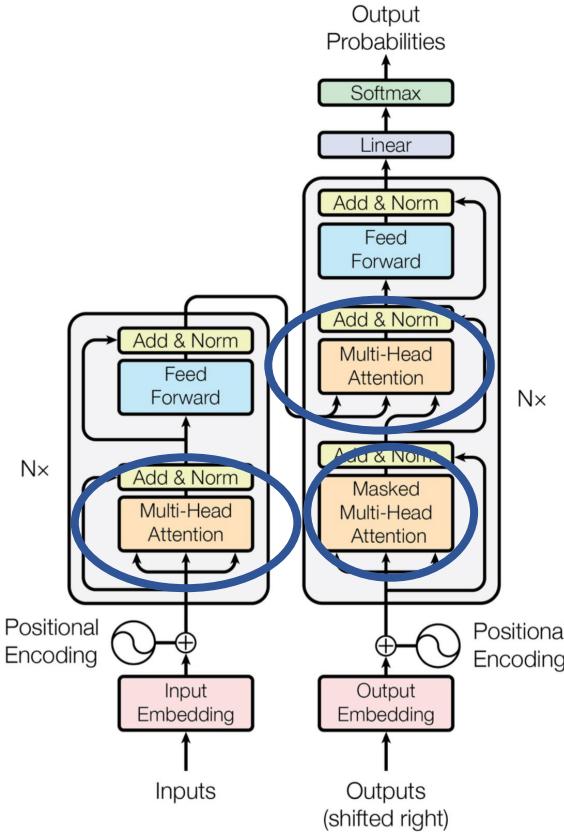


# How to solve those problems?

- Problem 1: Add an attention mechanism
- Problem 2 : Remove recurrent connections
- Paper from 2017: «Attention is all you need»

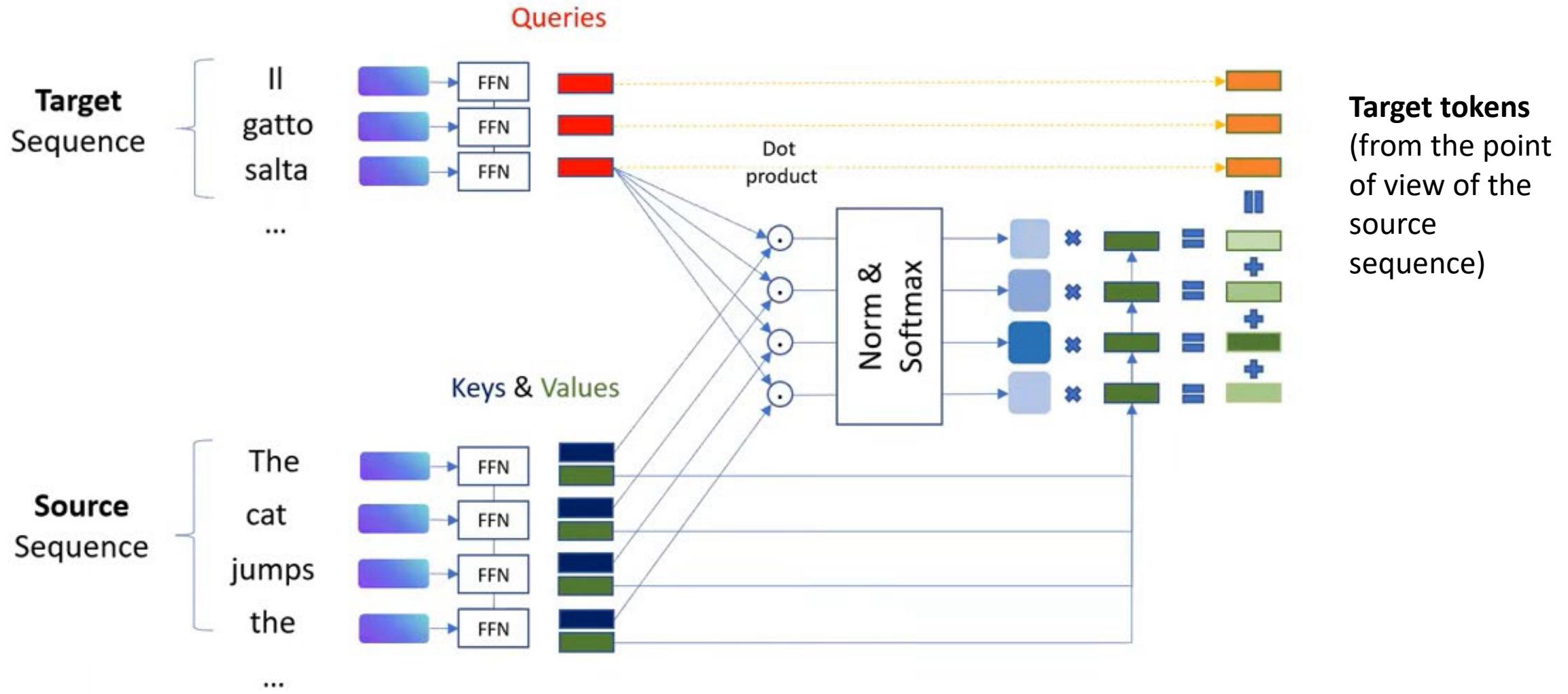


# Attention is all you need



The Transformer is a simple network architecture based on the attention mechanism which **boosts the speed** with which these models can be trained

# Transformer's Attention mechanism

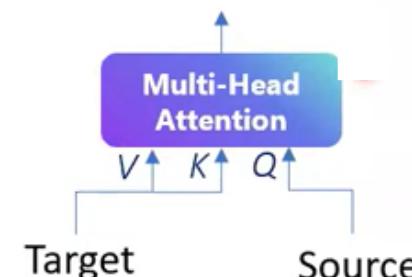


# Attention and Self-Attention

## Attention



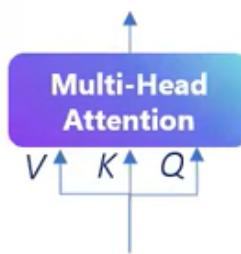
- Source  $\neq$  Target
  - Queries from Source
  - Key, Values from Target
- Captures **inter-sequence** dependencies



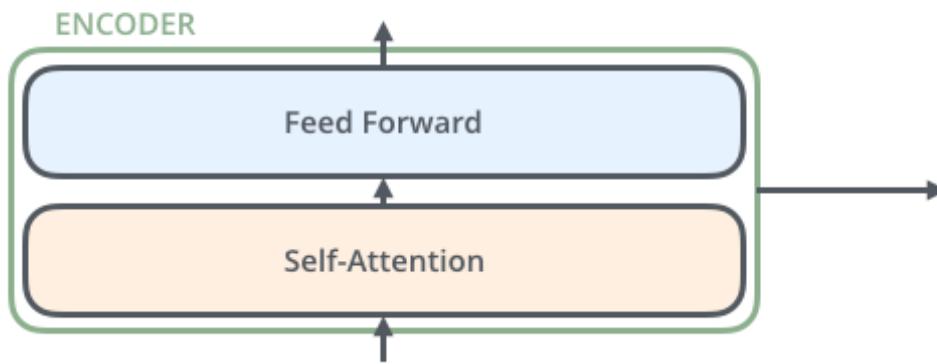
## Self-Attention



- Source = Target
  - Key, Queries, Values obtained from the same sentence
- Captures **intra-sequence** dependencies

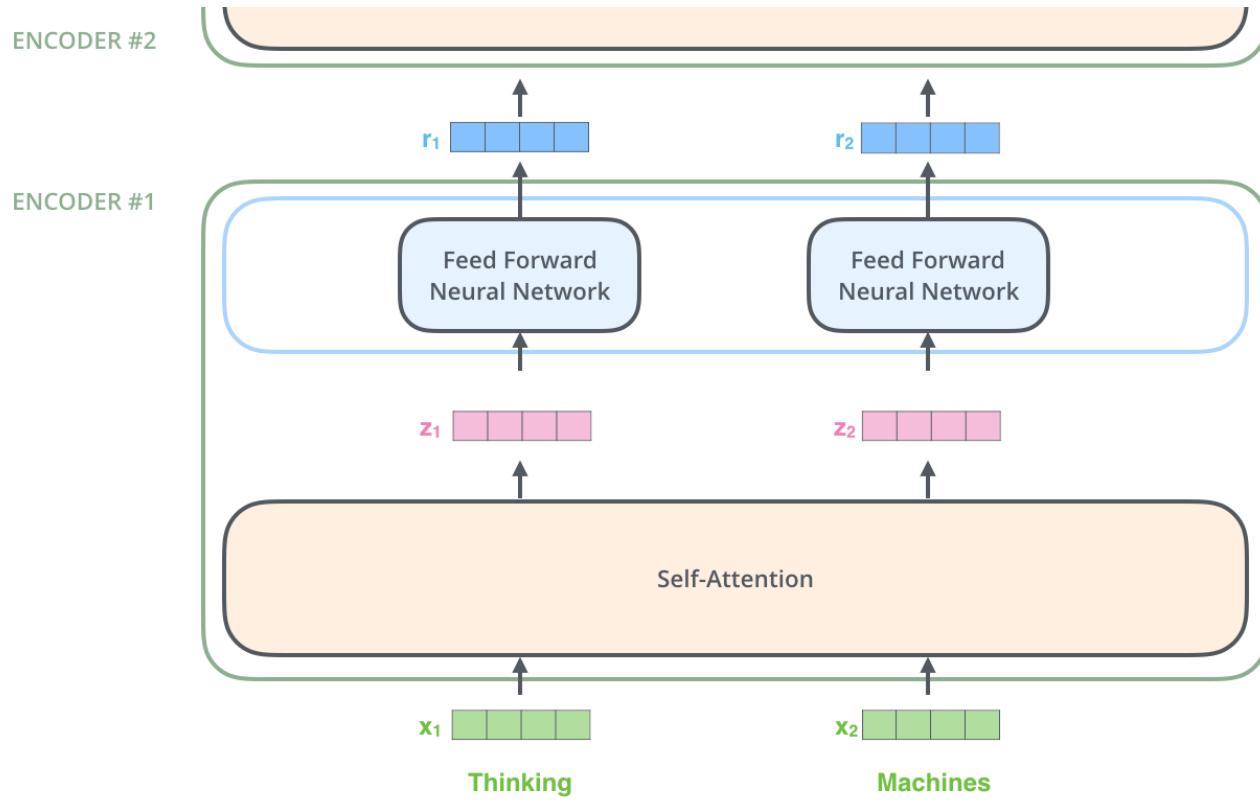


# Attention is all you need



- The encoders are **all identical** in structure (yet they do not share weights)
- The *self-attention layer* helps the encoder to **look at other words** in the input sentence as it encodes a specific word
- The outputs of the self-attention layer are fed to a feed-forward neural network

# The encoder



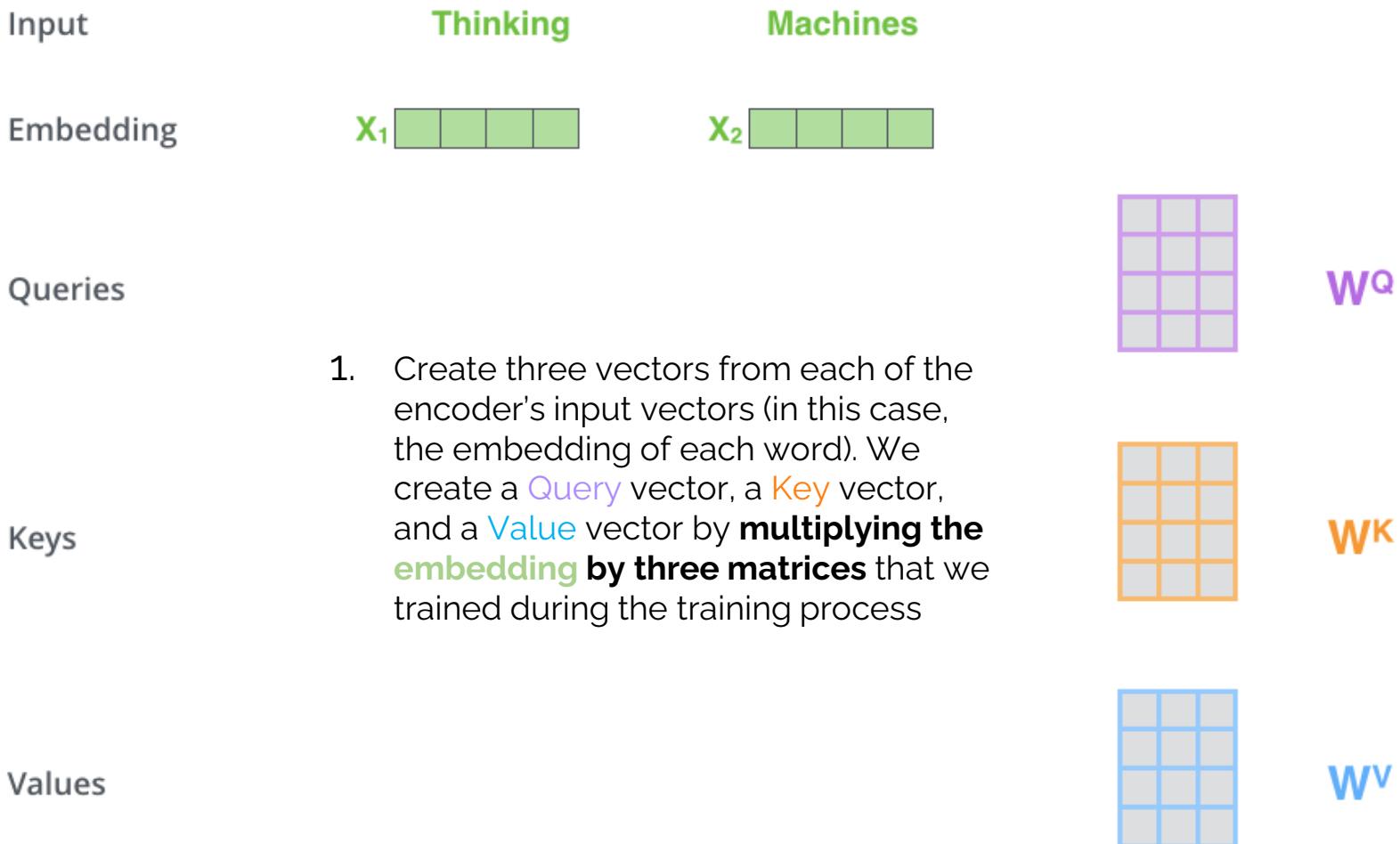
Key property of the Transformer: each word flows through its own path in the encoder.

- There are dependencies between these paths in the self-attention layer.
- The feed-forward layer does not have those dependencies and thus the various paths can be executed in parallel while flowing through the feed-forward layer

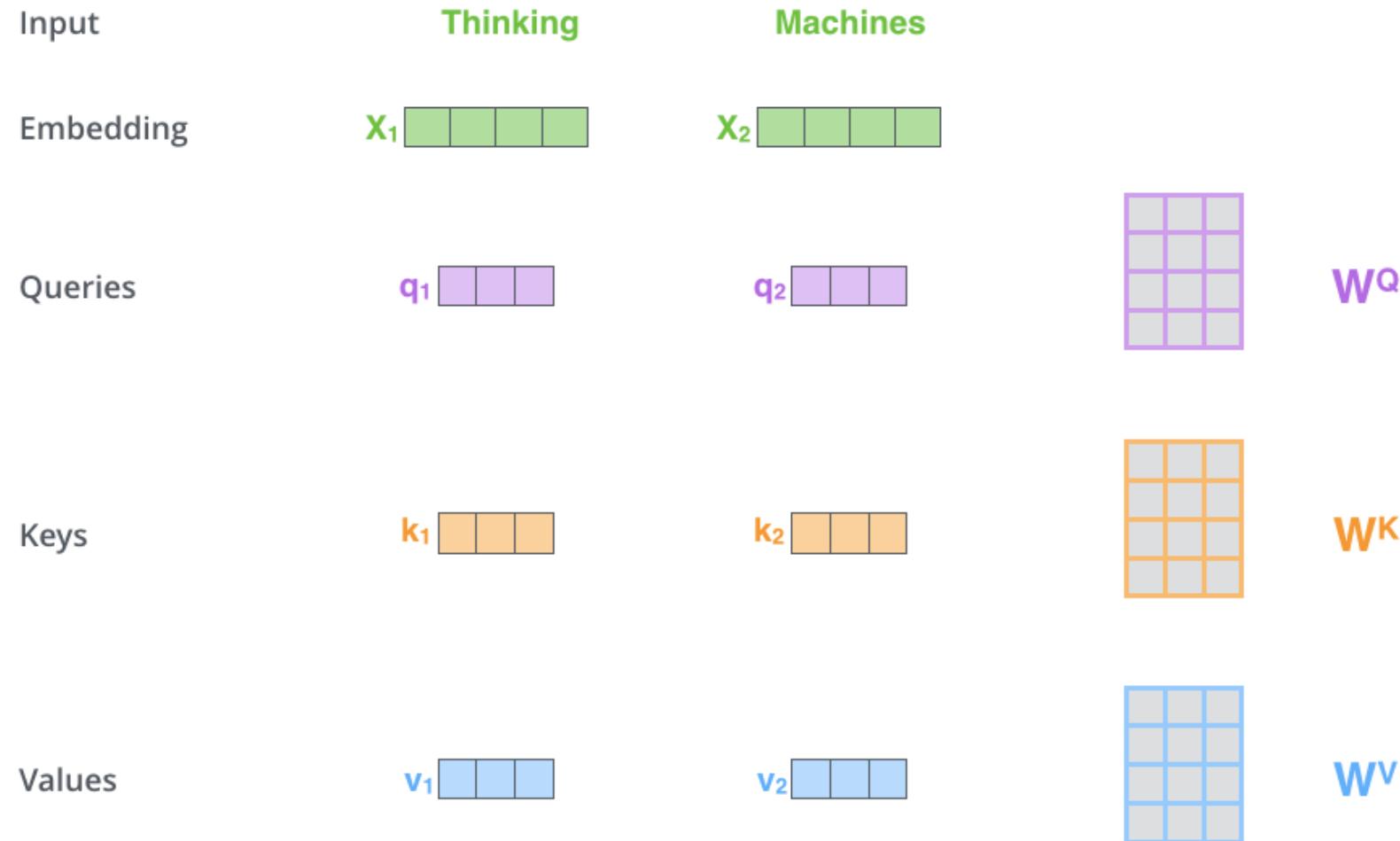
An encoder receives a list of vectors as input. It processes this list by passing these vectors into a 'self-attention' layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder.

# Self-attention

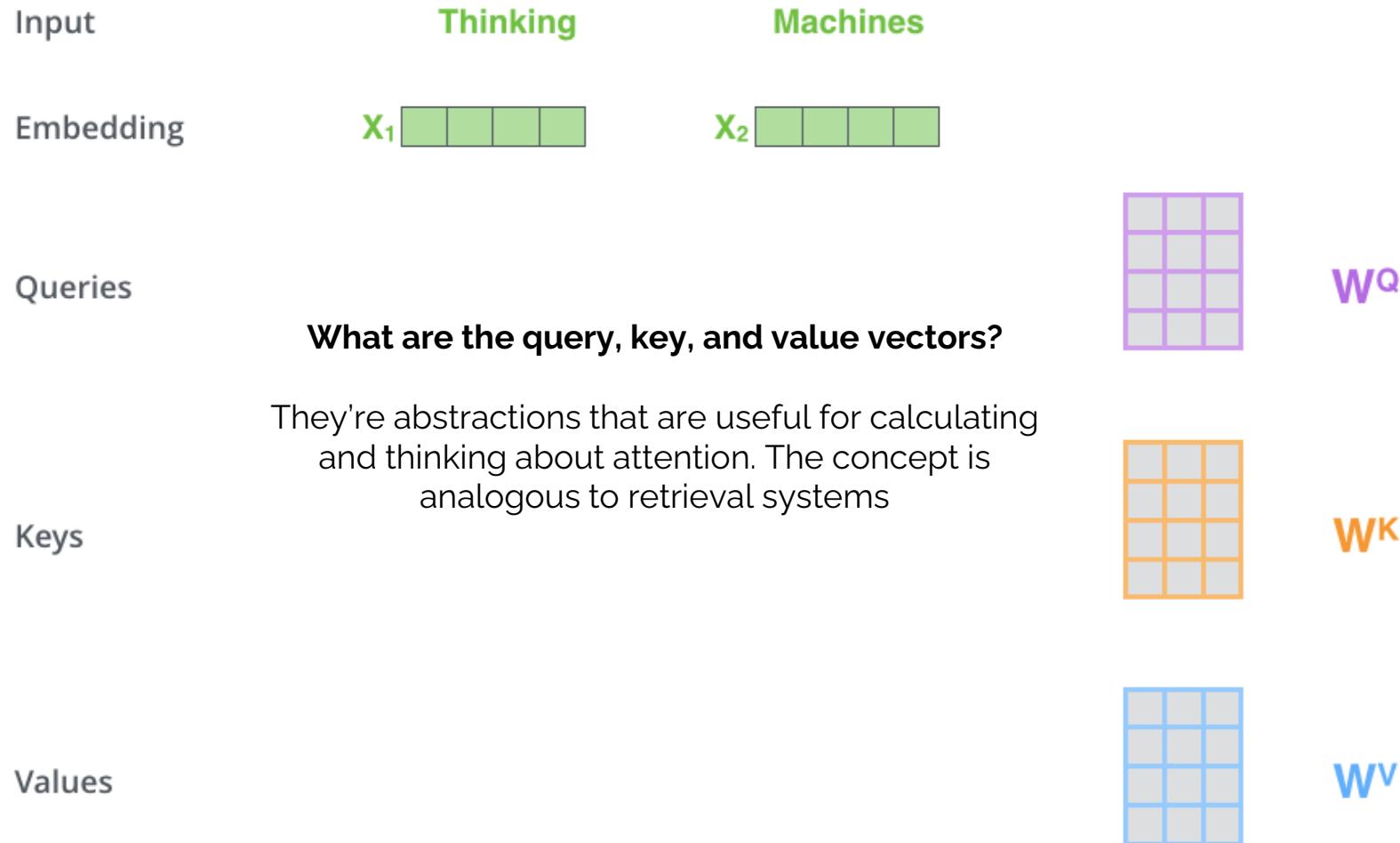
- As the model processes each word (each position in the input sequence), self attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word.
- Captures **intra-sequence** dependencies



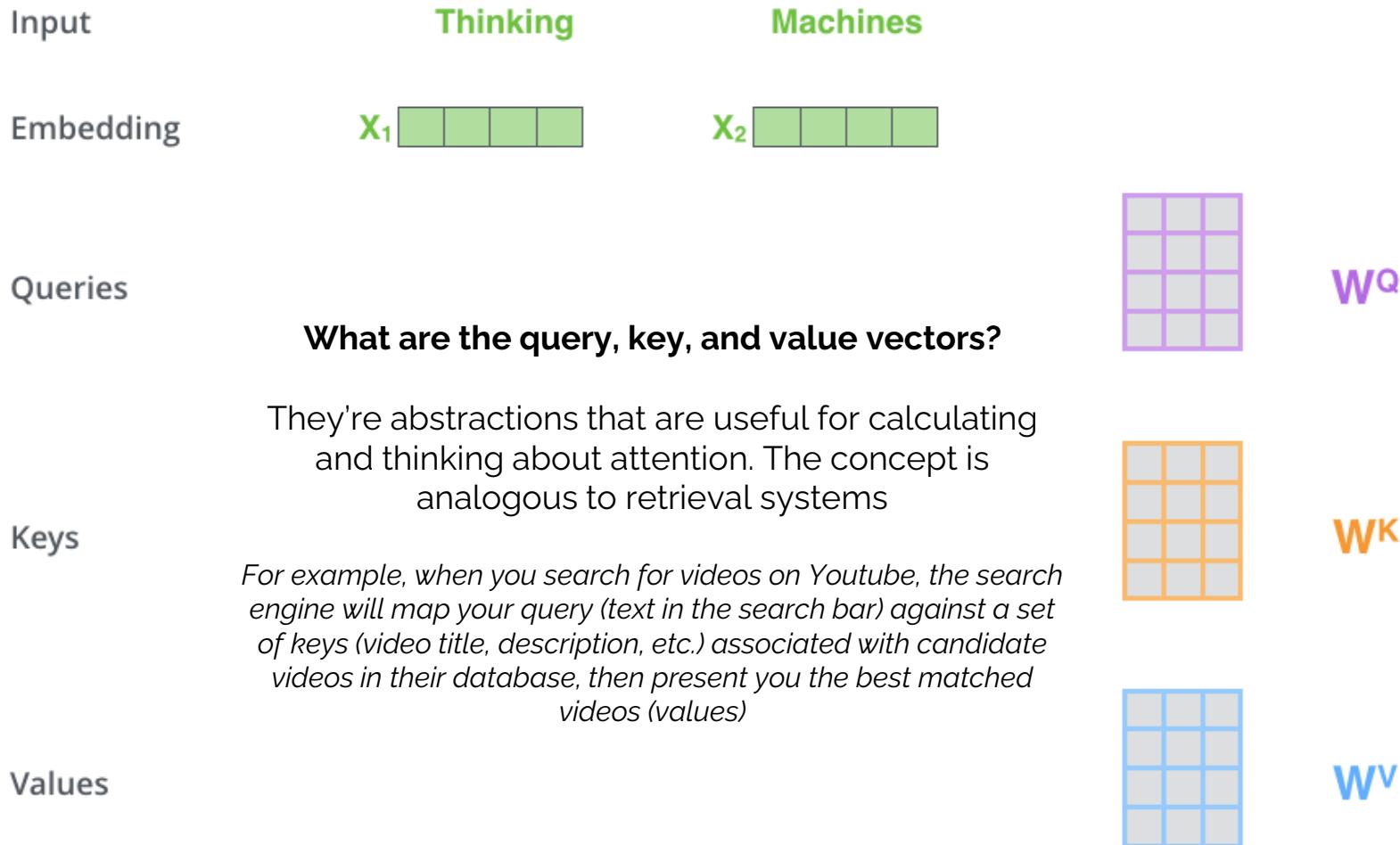
# Self-attention



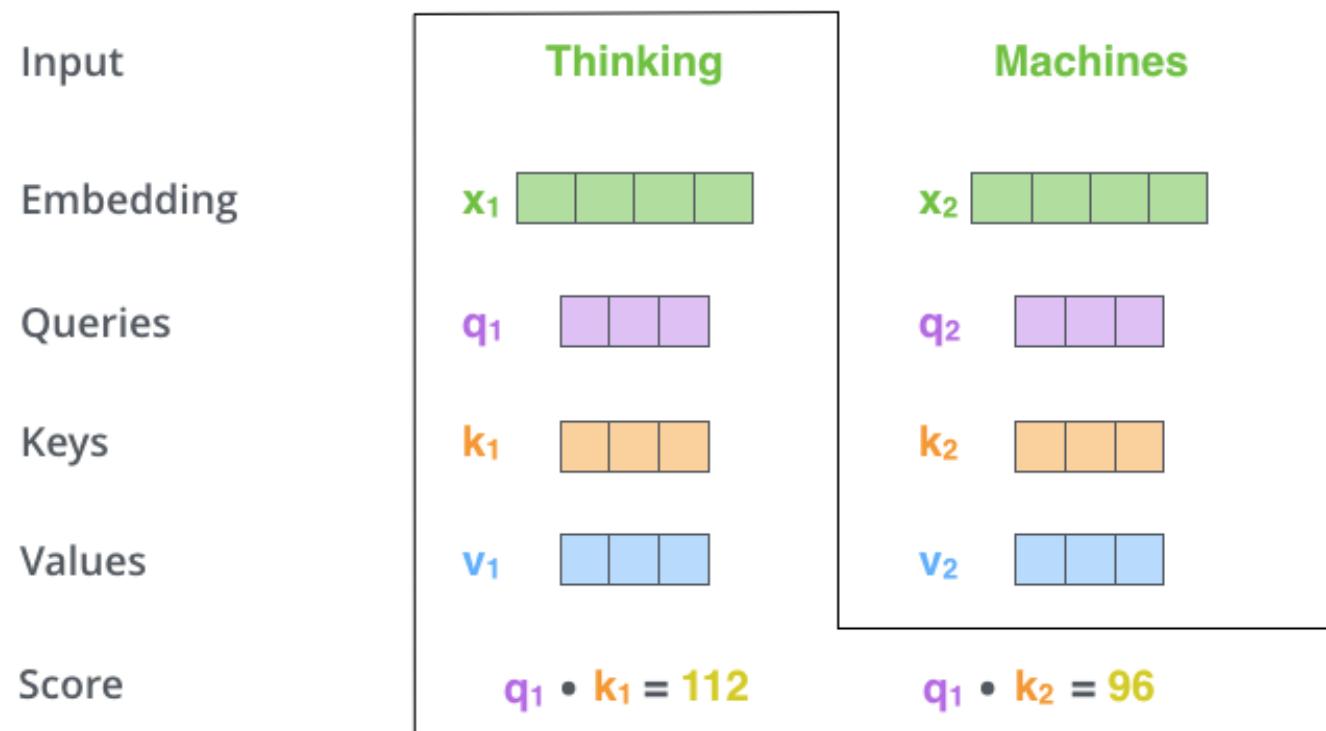
# Self-attention



# Self-attention



# Self-attention



2. Calculate the **score** by taking the **dot product** of the **query** vector with the **key** vector of the respective word we're scoring

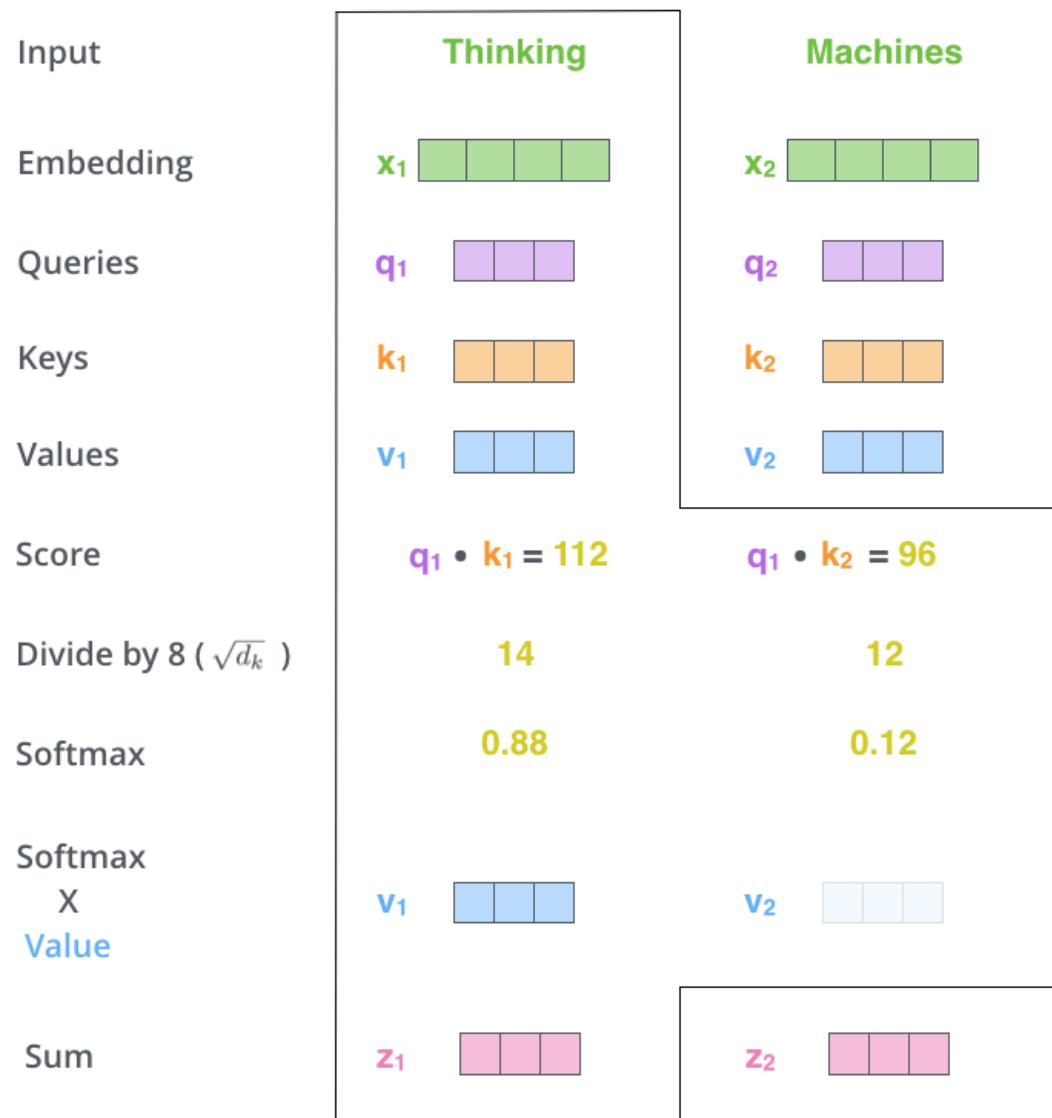
# Self-attention

Input		
Embedding	$x_1$	$x_2$
Queries	$q_1$	$q_2$
Keys	$k_1$	$k_2$
Values	$v_1$	$v_2$
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ( $\sqrt{d_k}$ )	14	12
Softmax	0.88	0.12

3. **Divide** the scores by the **square root** of the **dimension of the key** vectors (64 in the paper). This leads to having more stable gradients.
4. Pass the result through a **softmax** operation (softmax score determines how much each word will be expressed at this position)

# Self-attention

5. **Multiply** each **value** vector by the softmax score. The intuition here is to keep intact the values of the word(s) we want to **focus** on, and drown-out irrelevant words
6. **Sum** up the weighted value vectors



# Self-attention: matrix calculation

This calculation is done in matrix form for faster processing!

**The first step** is to calculate the Query, Key, and Value matrices. We do that by packing our embeddings into a matrix  $X$  and multiplying it by the weight matrices we have trained ( $WQ$ ,  $WK$ ,  $WV$ ).

$$X \times WQ = Q$$

A diagram illustrating the calculation of the Query matrix. It shows a green 4x4 matrix labeled 'X' multiplied by a purple 4x4 matrix labeled 'WQ', resulting in a purple 4x4 matrix labeled 'Q'. The matrices are represented as 4x4 grids of colored squares.

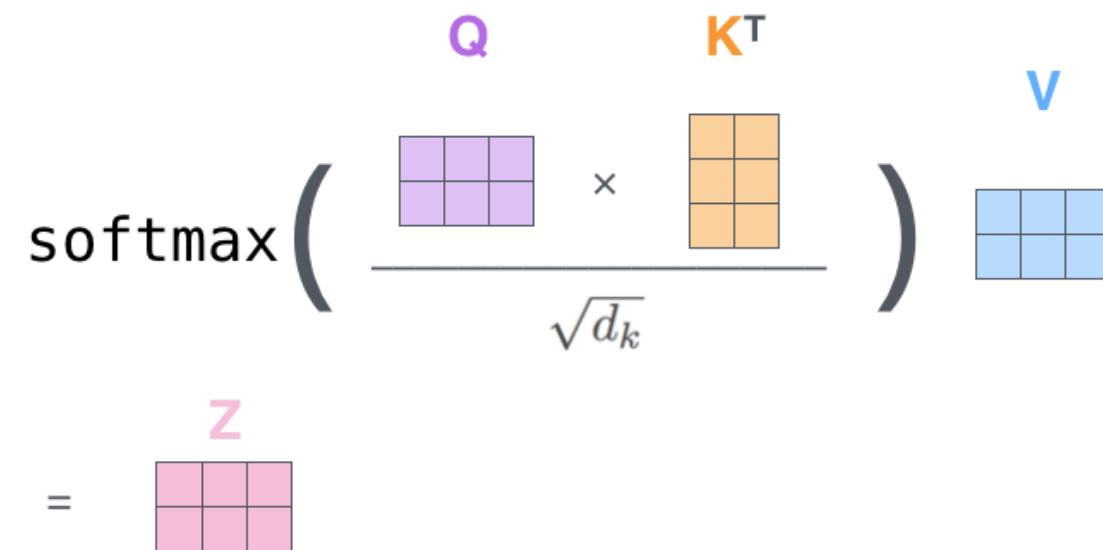
$$X \times WK = K$$

A diagram illustrating the calculation of the Key matrix. It shows a green 4x4 matrix labeled 'X' multiplied by an orange 4x4 matrix labeled 'WK', resulting in an orange 4x4 matrix labeled 'K'. The matrices are represented as 4x4 grids of colored squares.

$$X \times WV = V$$

A diagram illustrating the calculation of the Value matrix. It shows a green 4x4 matrix labeled 'X' multiplied by a blue 4x4 matrix labeled 'WV', resulting in a blue 4x4 matrix labeled 'V'. The matrices are represented as 4x4 grids of colored squares.

# Self-attention: matrix calculation

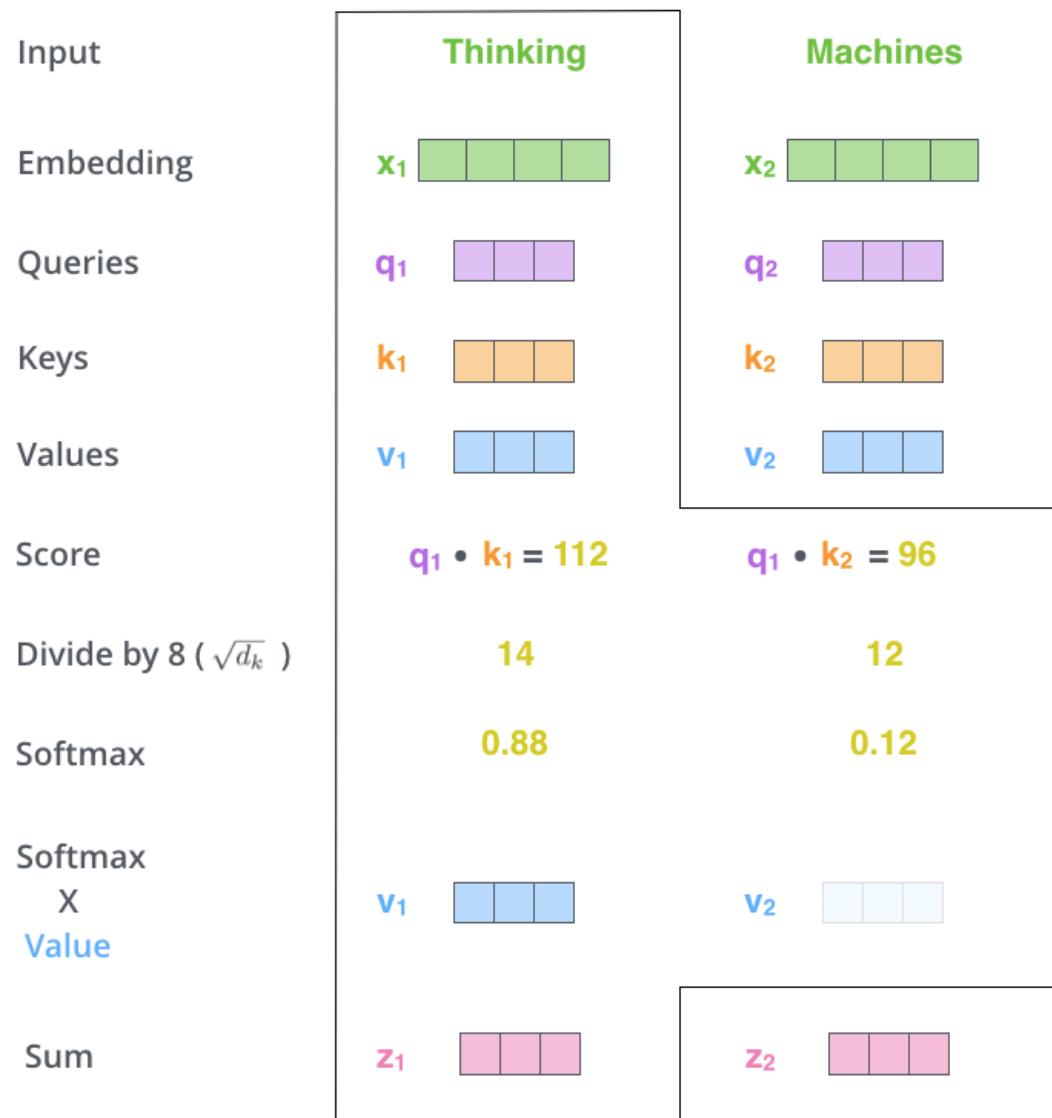
$$\text{softmax} \left( \frac{\begin{matrix} \mathbf{Q} & \mathbf{K}^T \\ \begin{matrix} \text{---} & \times \\ \hline \end{matrix} & \begin{matrix} \mathbf{K}^T \\ \text{---} \end{matrix} \end{matrix}}{\sqrt{d_k}} \right) \mathbf{V}$$
$$= \mathbf{Z}$$


# Self-attention: limitation

## Note

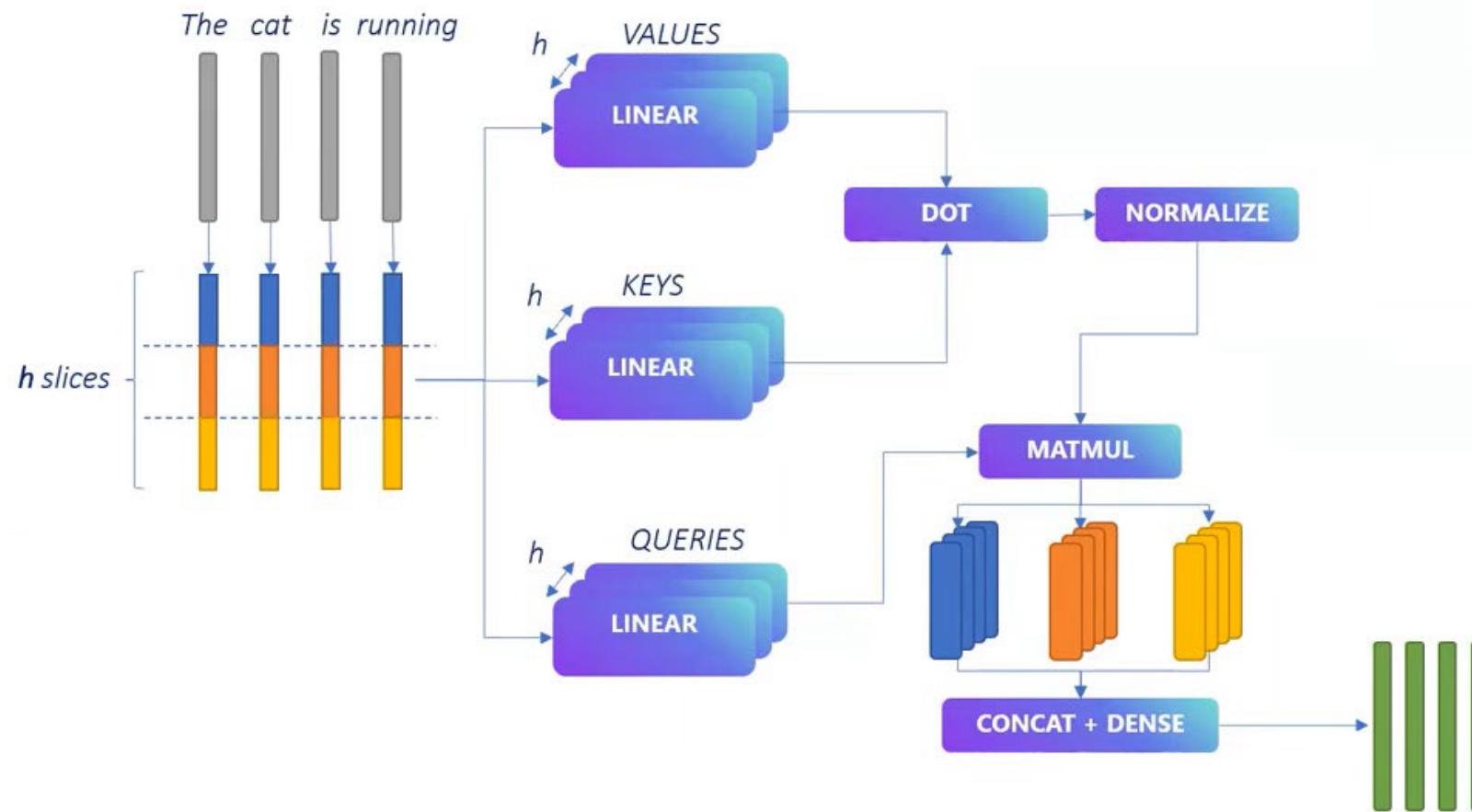
$z_1$  contains a little bit of every other encoding, but it could be **dominated by the the actual word itself**

It would be useful if we're translating a sentence like "**The animal** didn't cross the street because **it** was too tired", we would want to know which word "it" refers to



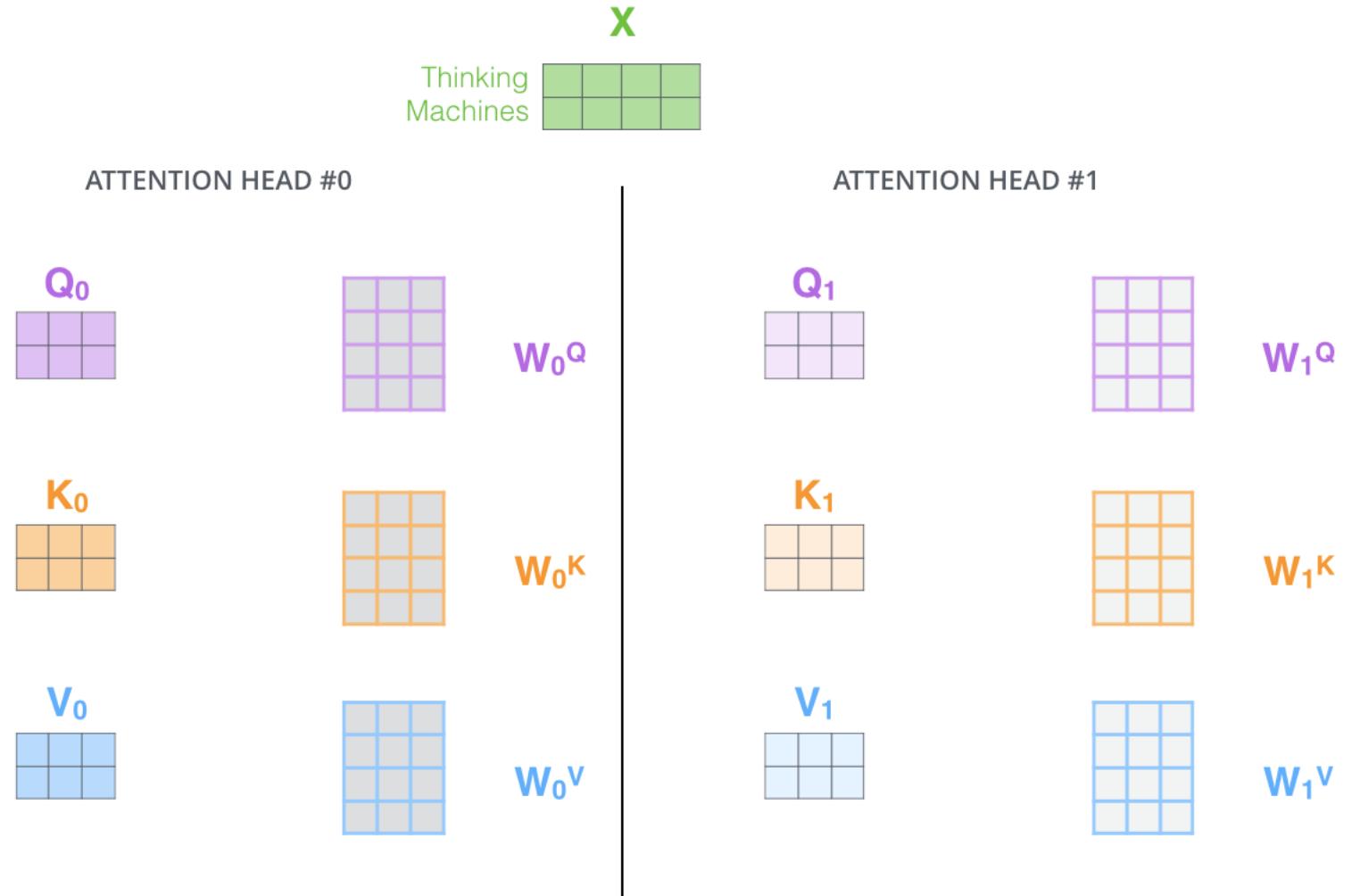
# Multi-head attention

Multiple instantiations of the attention mechanism



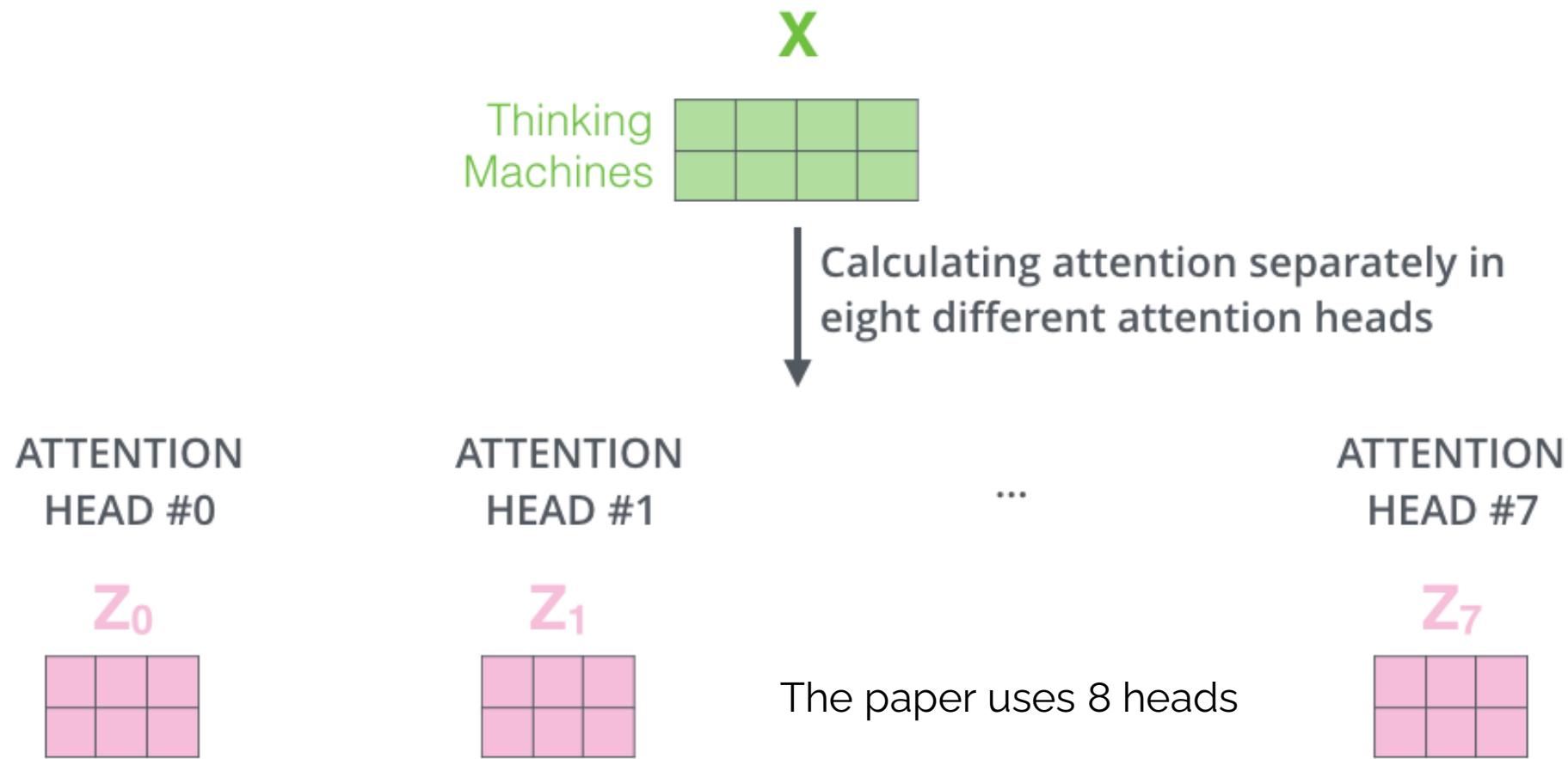
# Multi-head attention

1. It expands the model's ability to focus on **different positions**
2. It gives the attention layer **multiple “representation subspaces”**

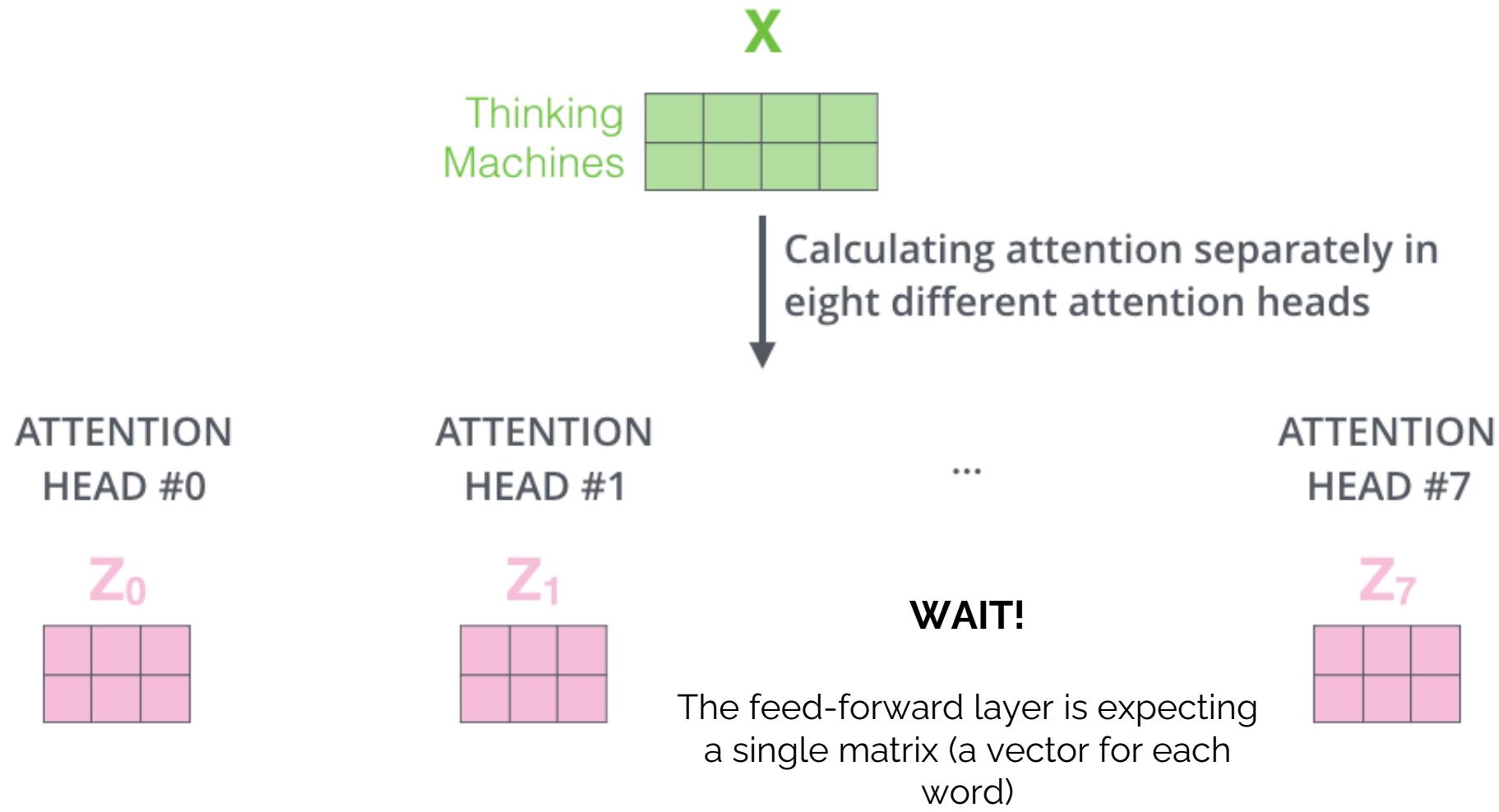


Each of these sets is randomly initialized. Then, after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace

# Multi-head attention

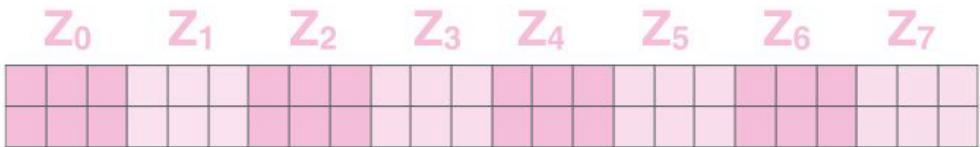


# Multi-head attention



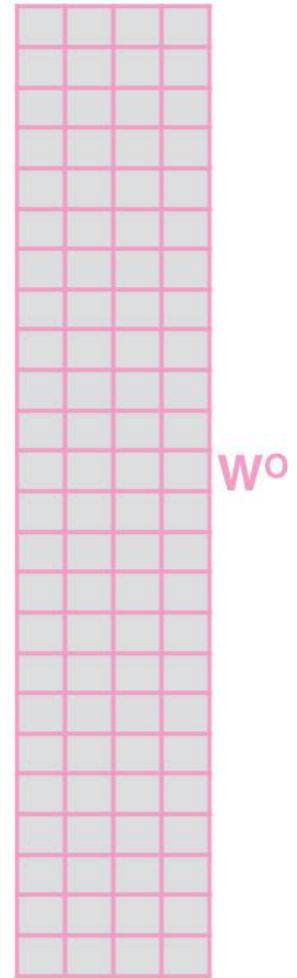
# Multi-head attention

1) Concatenate all the attention heads



2) Multiply with a weight matrix  $W^o$  that was trained jointly with the model

$X$



3) The result would be the  $Z$  matrix that captures information from all the attention heads. We can send this forward to the FFNN

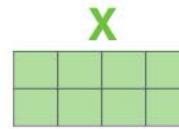
$$= \begin{matrix} Z \\ \hline \end{matrix}$$

# Multi-head attention: put all together!

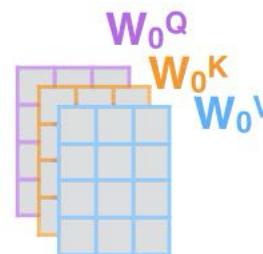
1) This is our input sentence\*

Thinking  
Machines

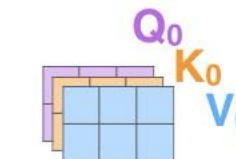
2) We embed each word\*



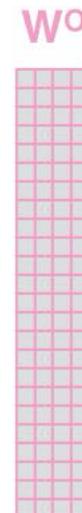
3) Split into 8 heads.  
We multiply  $X$  or  $R$  with weight matrices



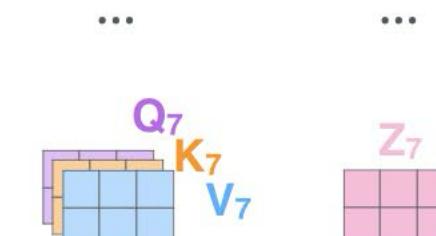
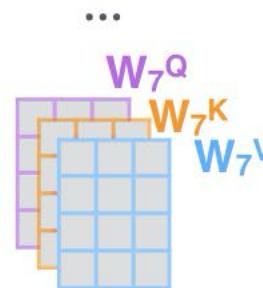
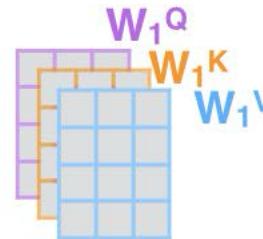
4) Calculate attention using the resulting  $Q/K/V$  matrices



5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer

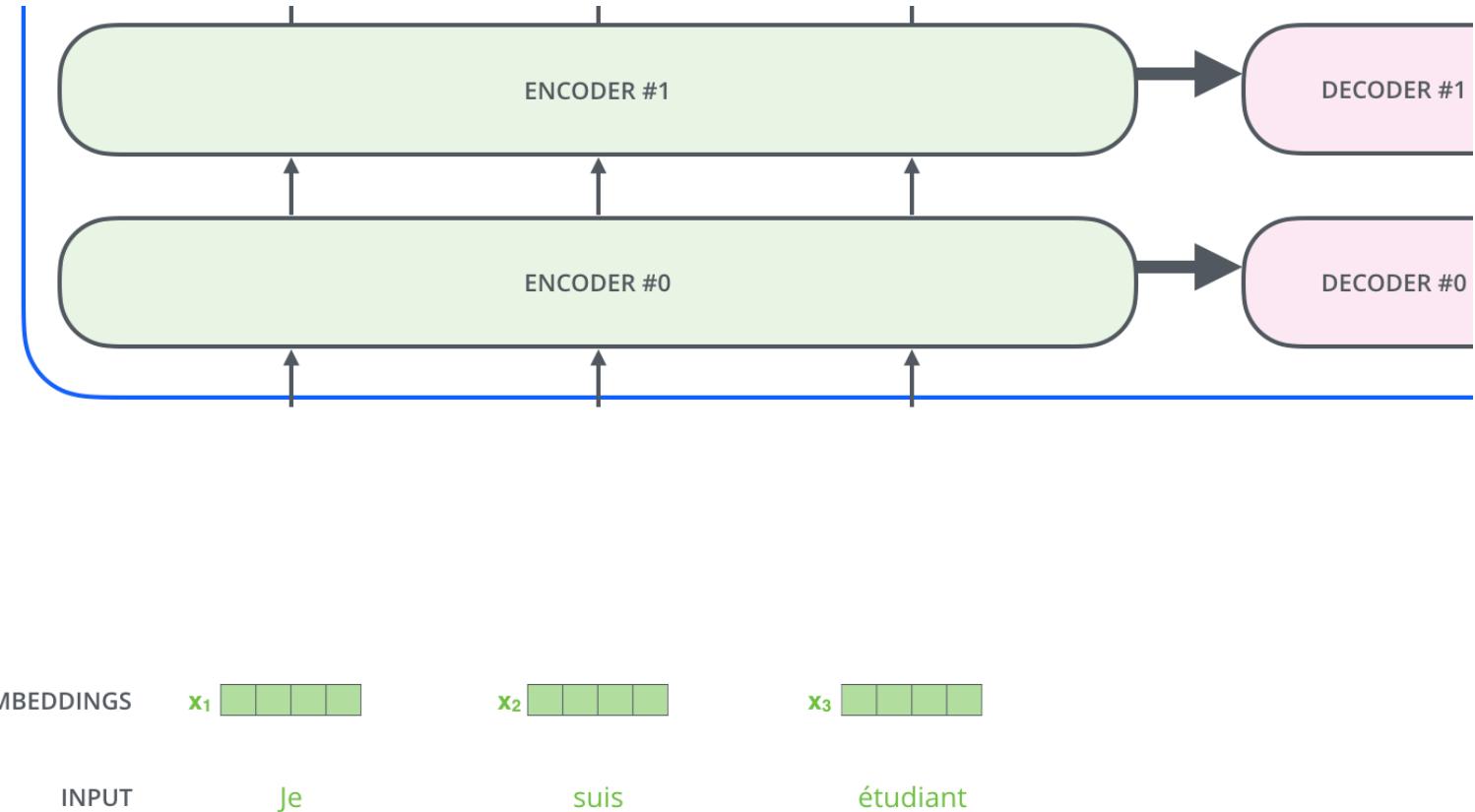


\* In all encoders other than #0, we don't need embedding.  
We start directly with the output of the encoder right below this one



# Positional Encoding

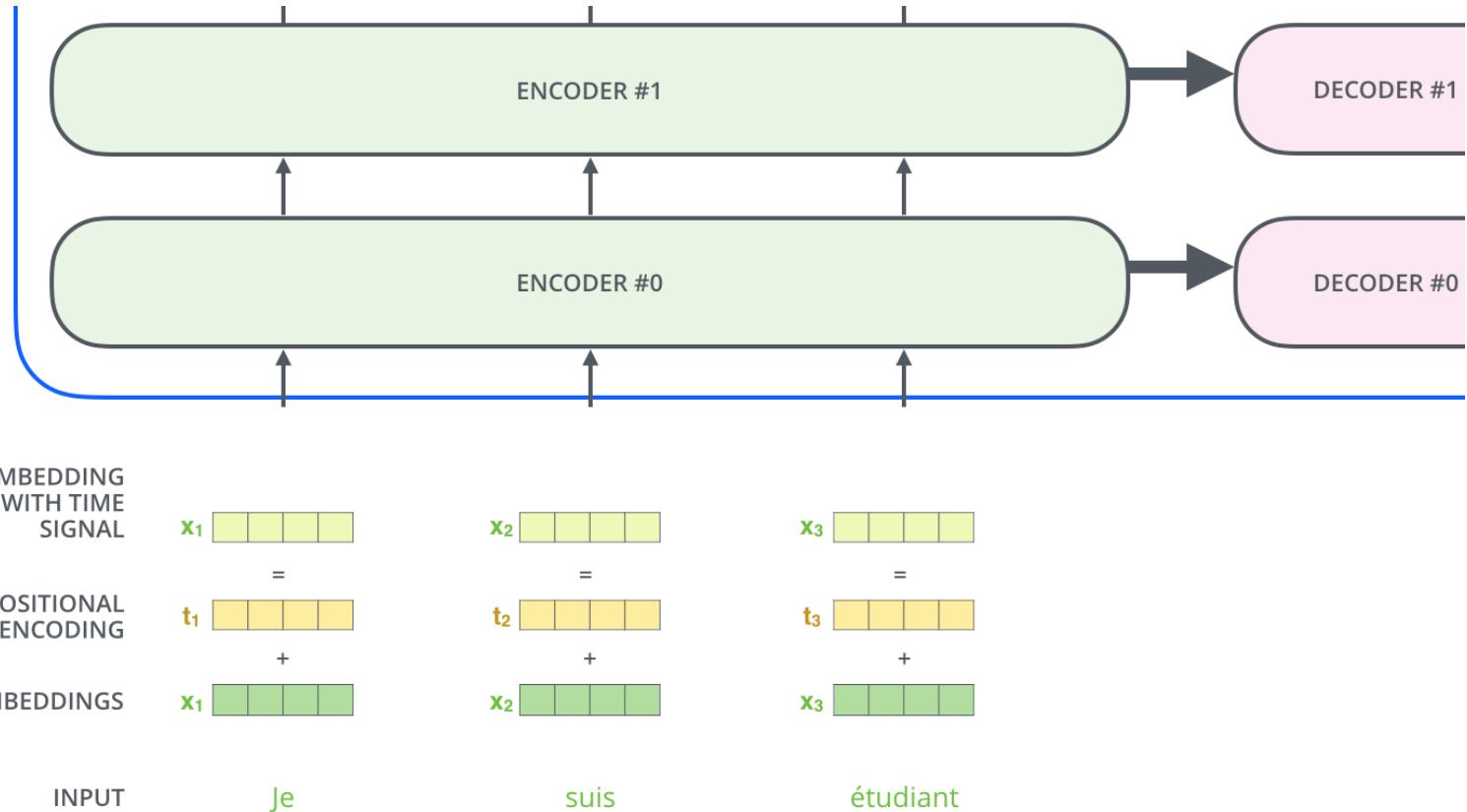
One thing that's missing from the model as we have described it so far is a way to account for the **order of the words** in the input sequence!



# Positional Encoding

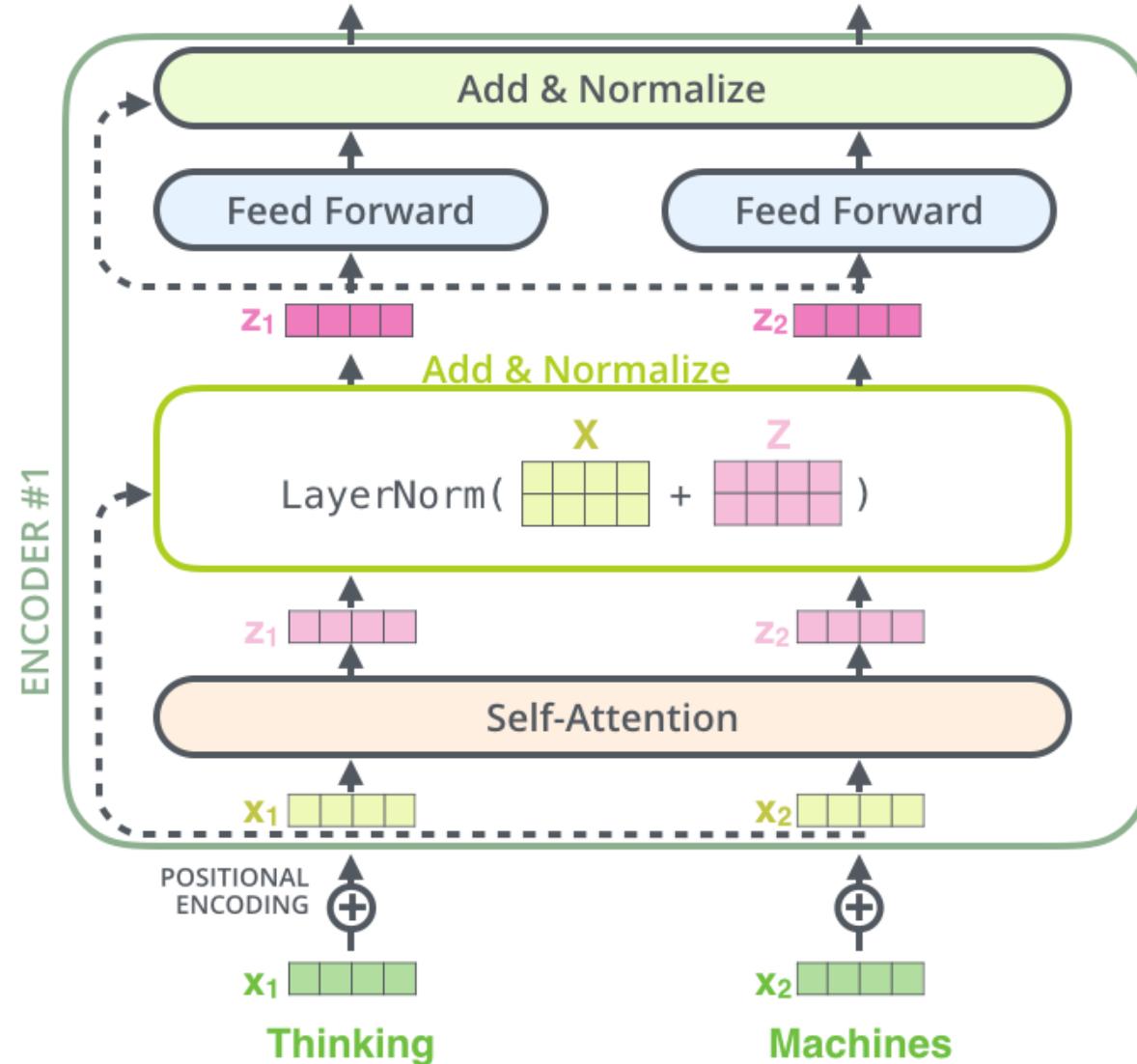
One thing that's missing from the model as we have described it so far is a way to account for the **order of the words** in the input sequence!

To address this, the transformer adds a vector to each input embedding which helps it determine the position of each word

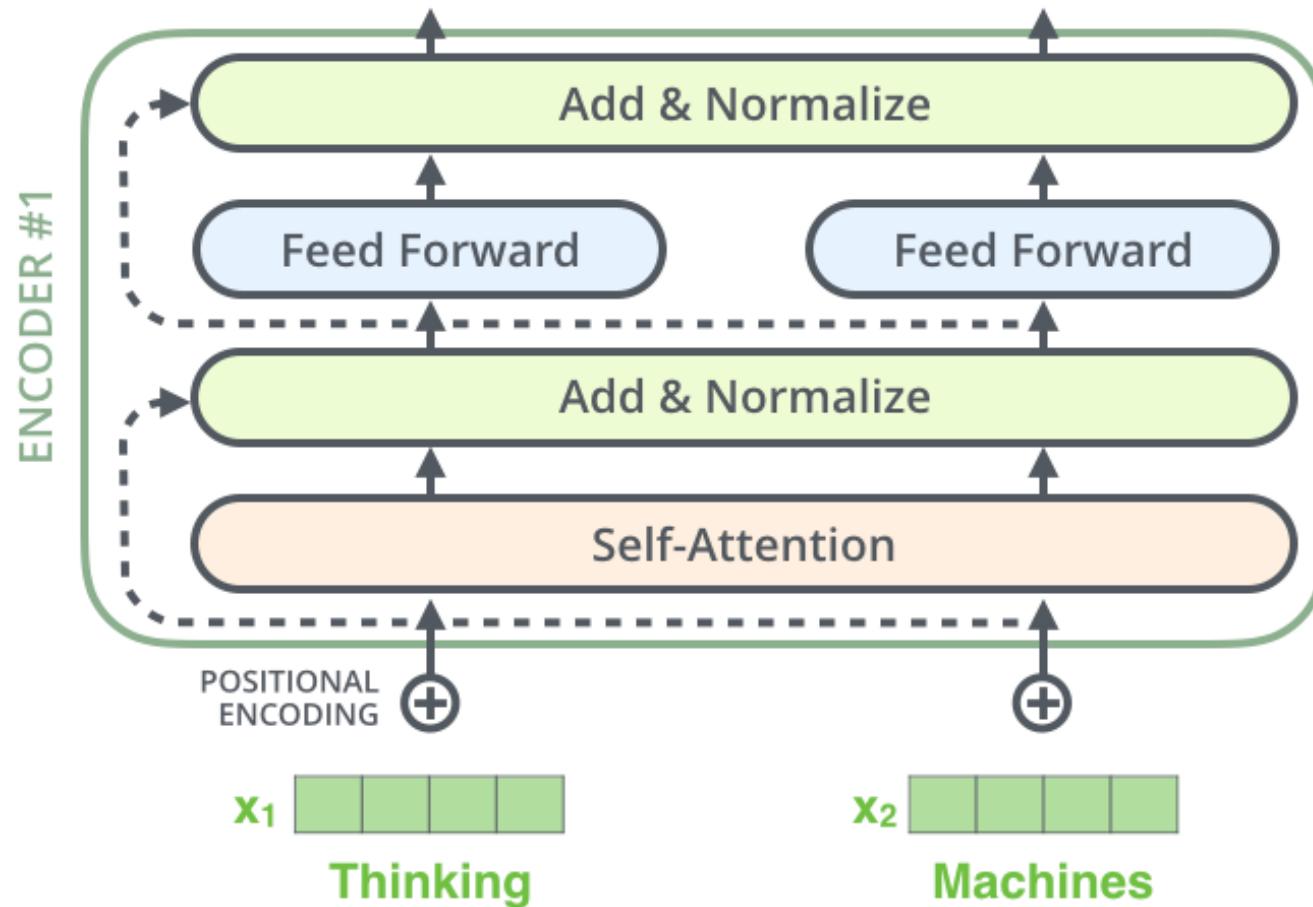


# The residuals

Each sub-layer (self-attention, feed forward neural net) in each encoder has a residual connection around it, and is followed by a **layer-normalization** step



# The encoder



# Comparison to other layers

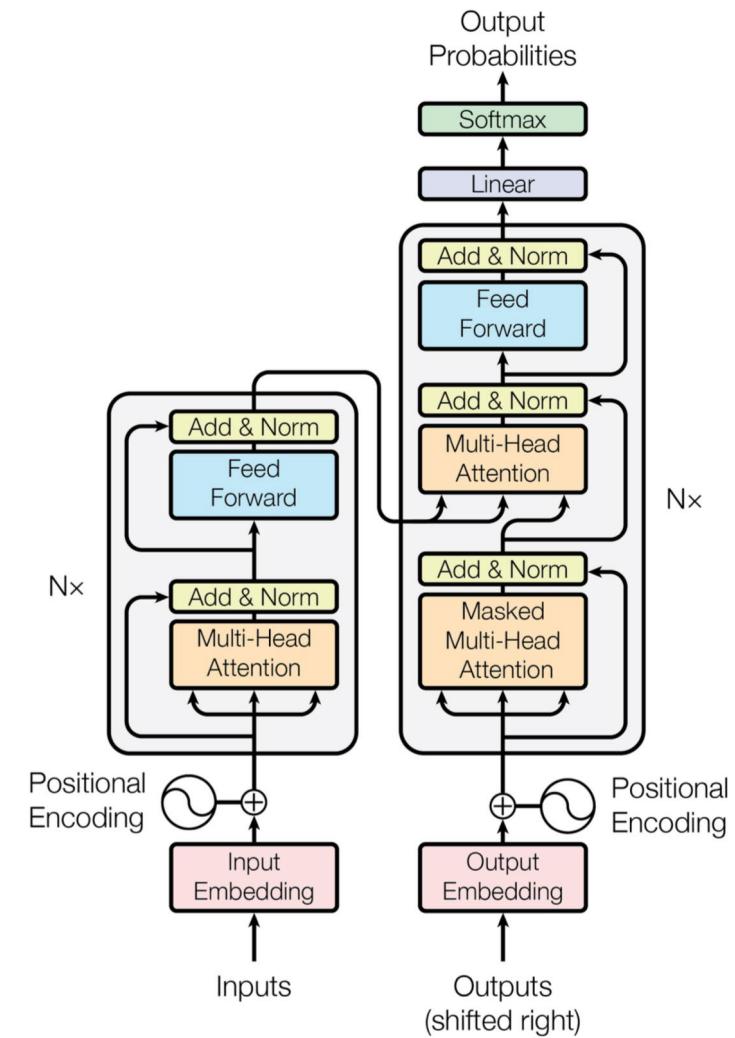
- Self-attention layers were found to be **faster than recurrent layers** for shorter sequence lengths, and can be restricted to consider only a neighbourhood in the input sequence for very long sequence lengths
- The number of sequential operations required by a recurrent layer is based upon the **sequence length**, whereas this number remains **constant** for a self-attention layer
- Tracking long-term dependencies with convolutional layers would require the use of **large kernels**, or **stacks of convolutional layers** that could increase the computational cost. However, a single layer of MHA has a receptive field of  $n$  (*i.e. sequence length*), while the convolutive layer has a receptive field of  $k$  (*i.e. kernel size*)

# The transformer network: key concepts

- The **encoder attends to all words** in the input sequence, irrespective if they precede or succeed the word under consideration
- The decoder receives as input its own predicted  $y_{t-1}$
- At the second sublayer, the decoder also receives the output of the encoder, which now allows the decoder to **attend to all of the words in the input sequence**

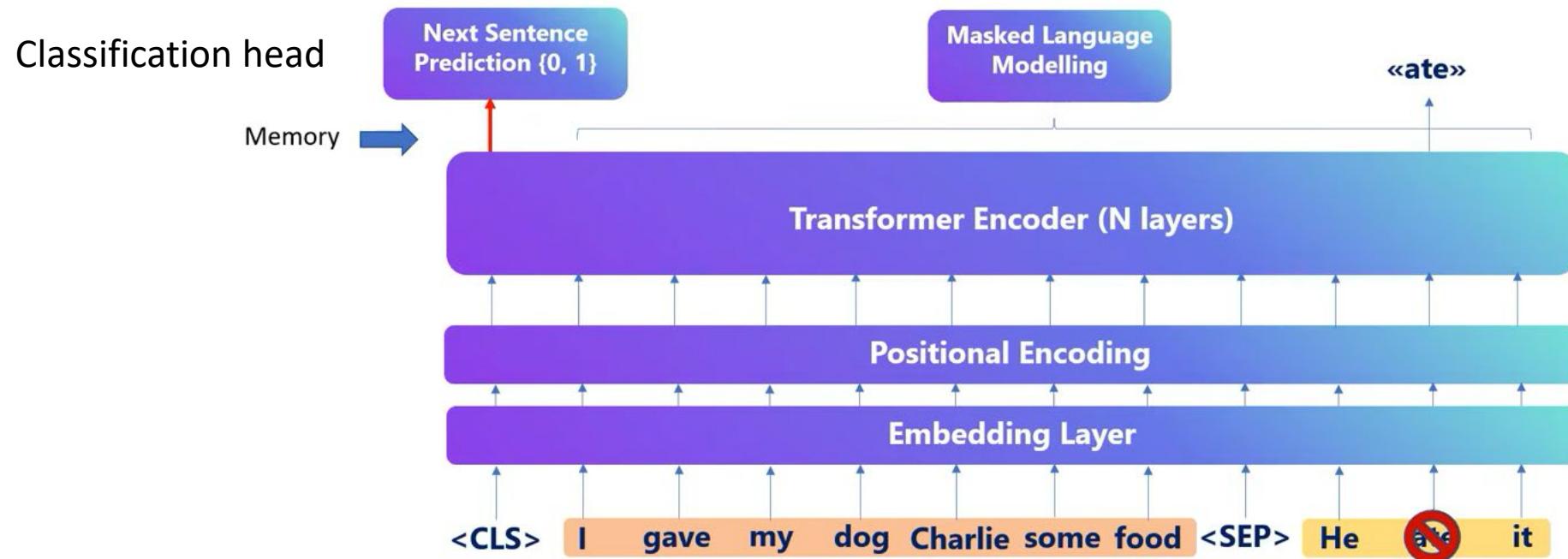
# The transformer network: limitations

- Self- Attention calculation is  $O(n^2)$
- Is possible to construct an Attention map



# Transformer Encoder

- BERT (Devlin et al. 2018)



sentence-level representation for classification

# Expanding the universe of self-attention

- **Attention** dominates **language**
- **Convolution** dominates (dominated?) **vision**
- Can we bring **attention** to **vision**?

# Why self-attention for vision?

# Attention Models: Motivation

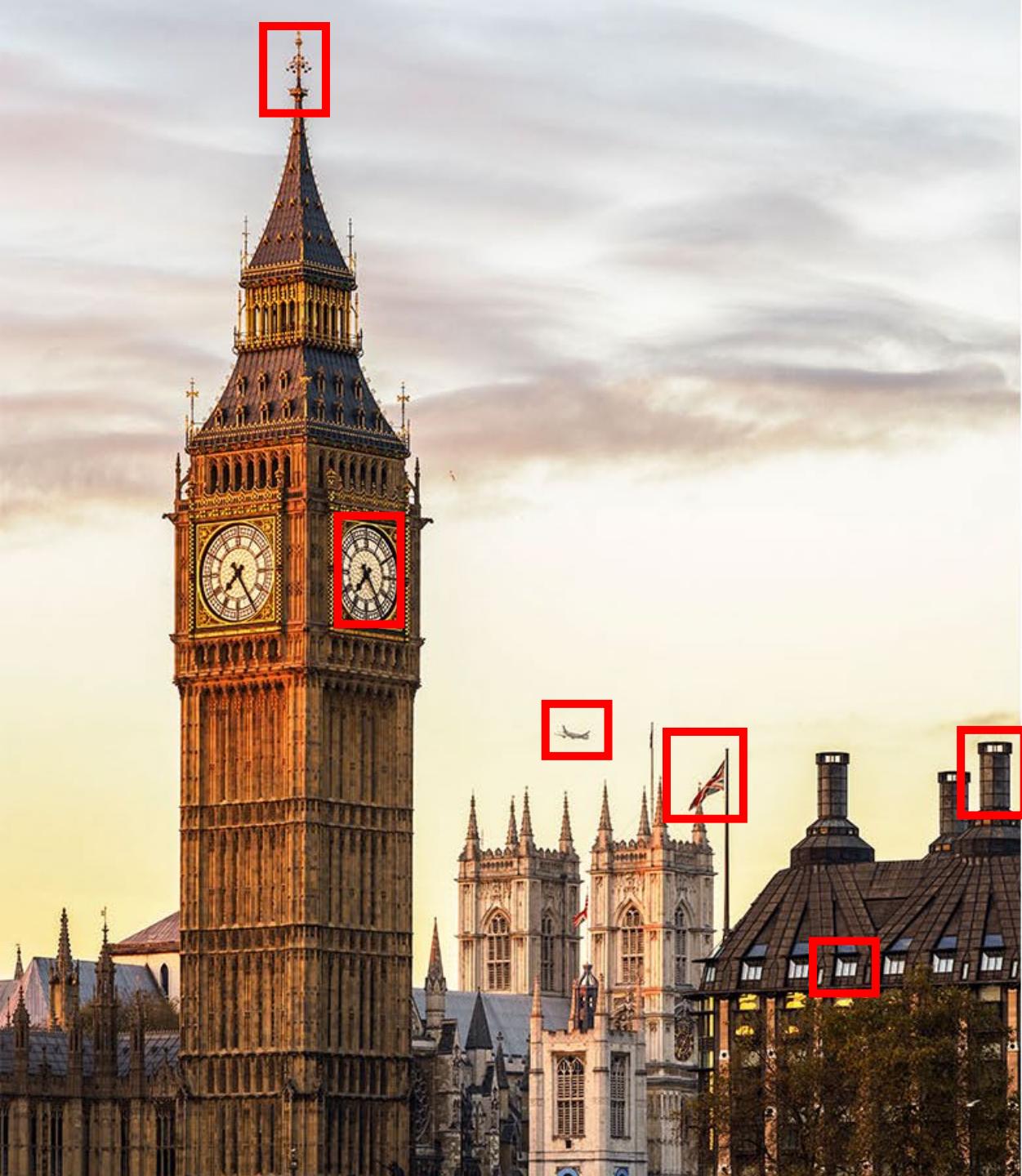


Can we use self-attention in images?

- What are the images patches relevant to a given query patch
- Concentrate on relevant parts of the image
- Given non local nature of S-A enable to discover relationship between spatial distant image patches

# Possible motivations

- Modeling long-range interactions between words (pixels).
- Useful for longer sentences (images).
- Different heads can model different kinds of interactions between words (pixels)
- Self-attention can model the self-similarity within images



FIND THE DIFFERENCES



Who is wearing glasses?



Where is the child sitting?



Is the umbrella upside down?



How many children are in the bed?





# DALLE: Creating images from text

Text prompt

an armchair in the shape of an avocado [...]

AI-generated images



[View more images or edit prompt ↓](#)

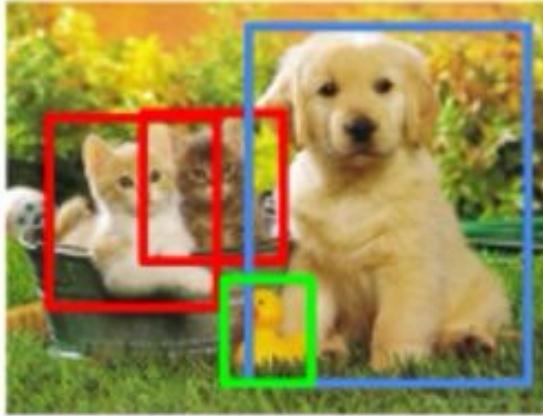
# Vision tasks

Classification



Cat

Detection



Cat, Duck

Segmentation

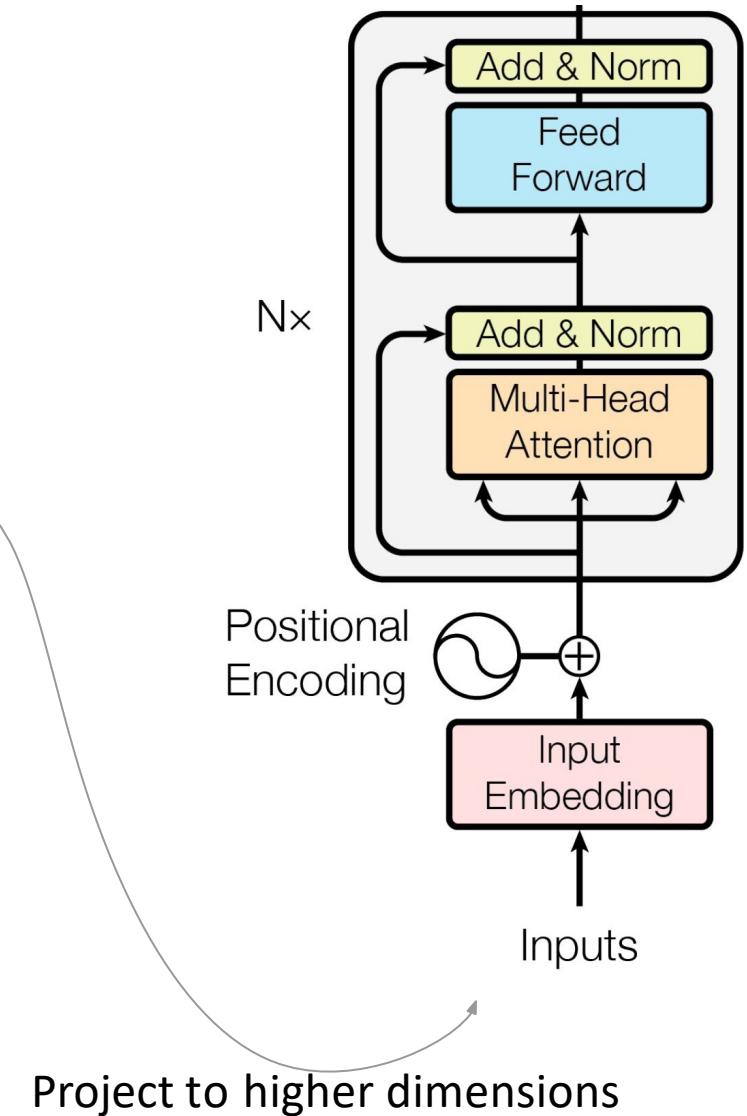
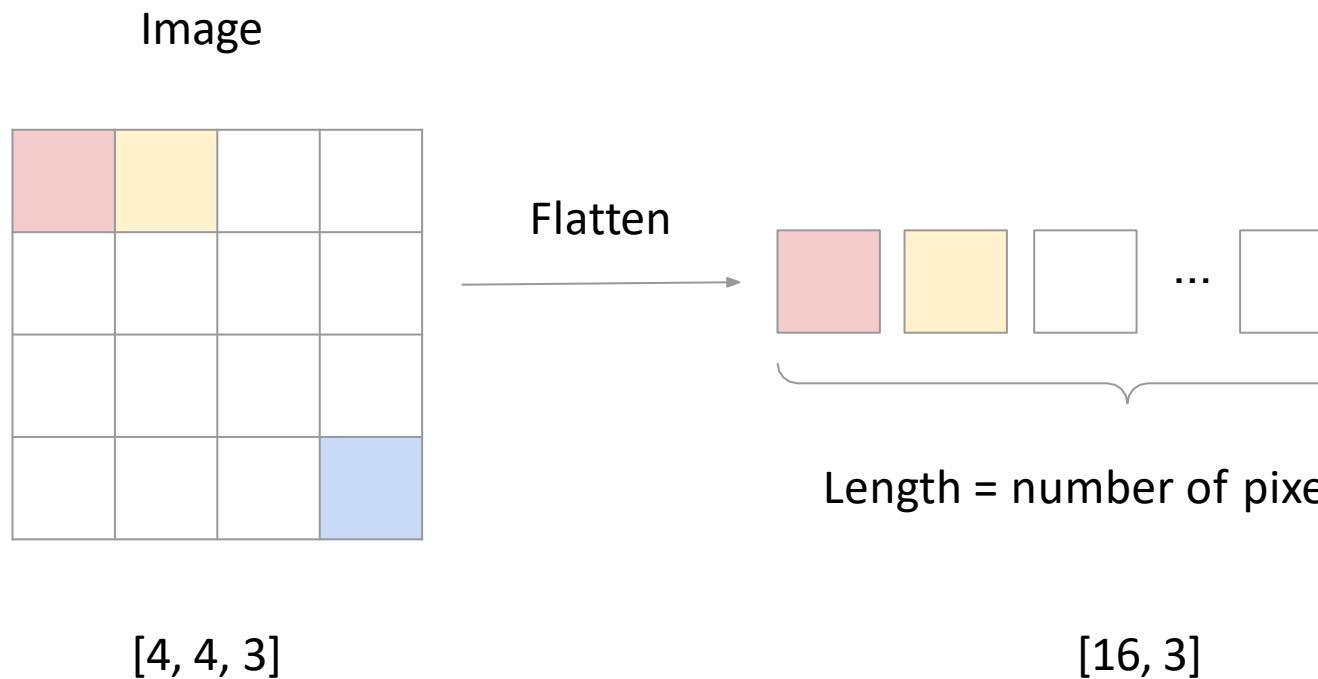


Cat, Duck

[Source](#)

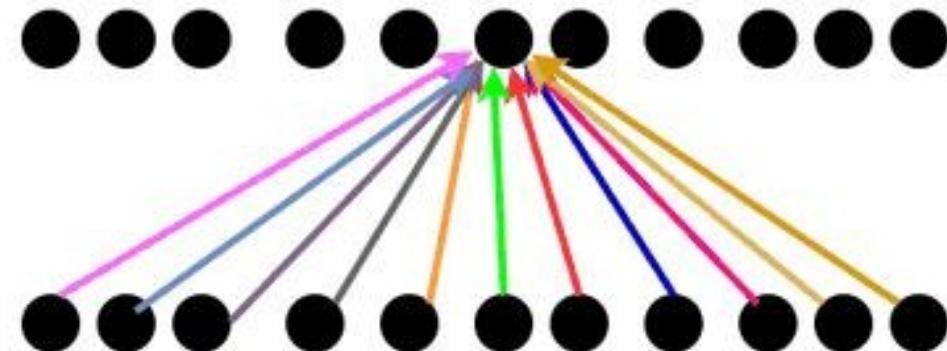
How do we design vision attention models?  
Adapt a pre-existing method

# Idea: treat each pixel as a token, and pass to Transformer



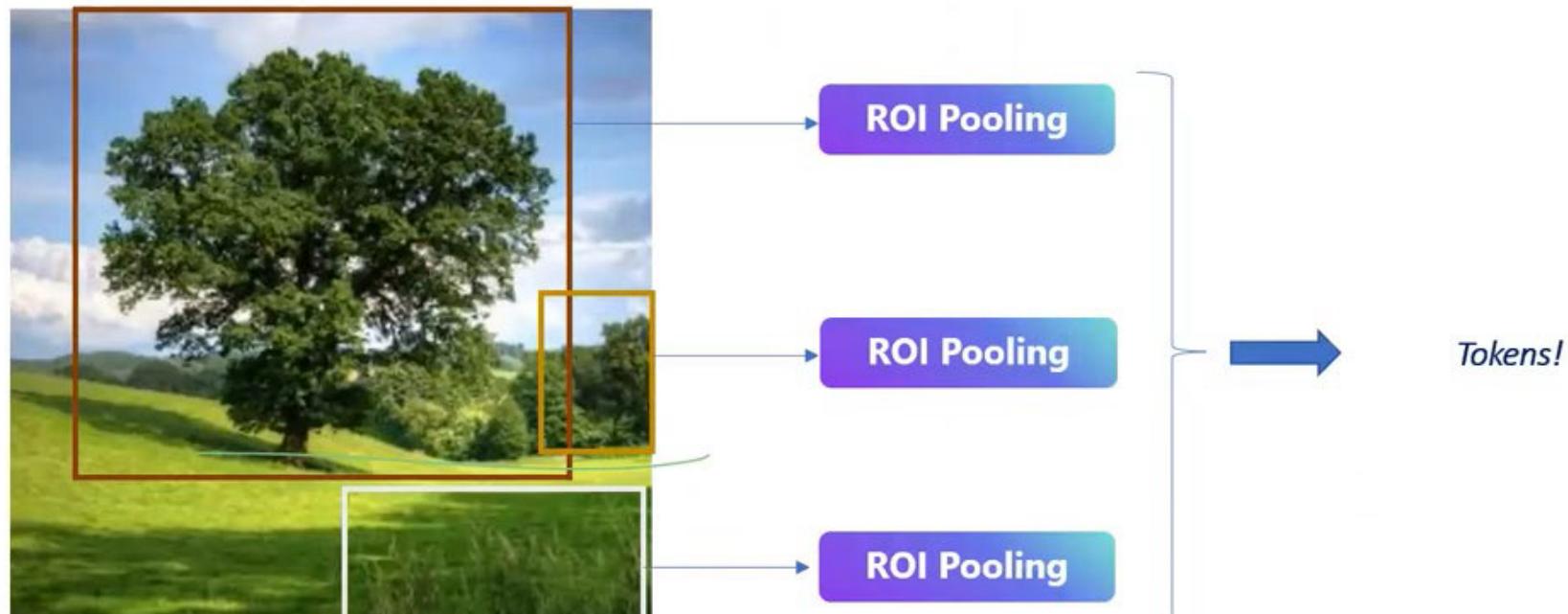
# Problem: it's too expensive!

- For a 256x256 images, there 62500 pixels
- Attention cost scales quadratically with the input length
- So 3906250000 calculations!!!



# Transformers in Computer Vision

- Tokens as the features from an object detector



Used in visual textual matching

# Idea: use a smaller image size

- Will reduce the input length, which makes the Transformer cheaper



[Source](#)

# Problem: loses a lot of detail



horse (7)



ship (8)



deer (4)



deer (4)



frog (6)



dog (5)



bird (2)



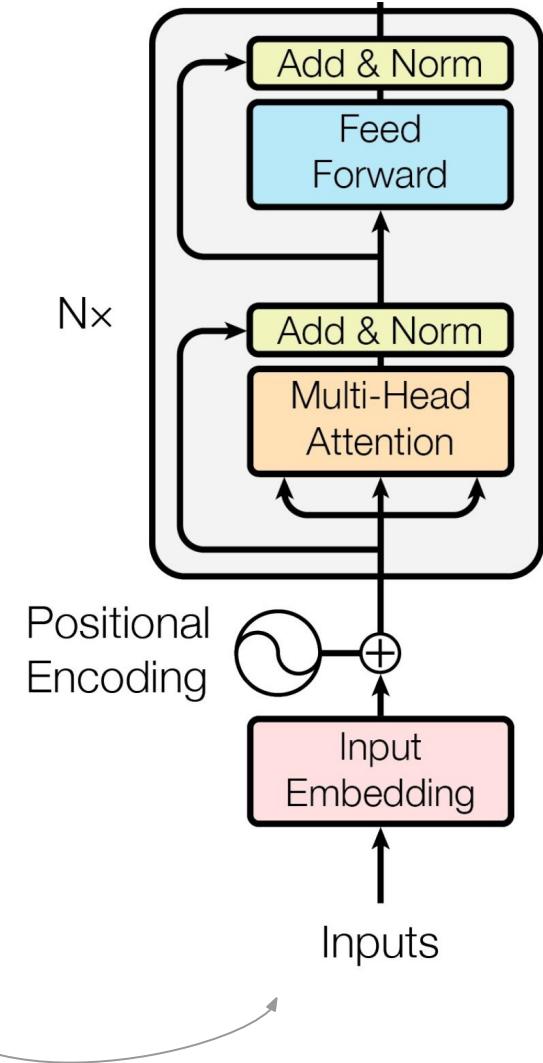
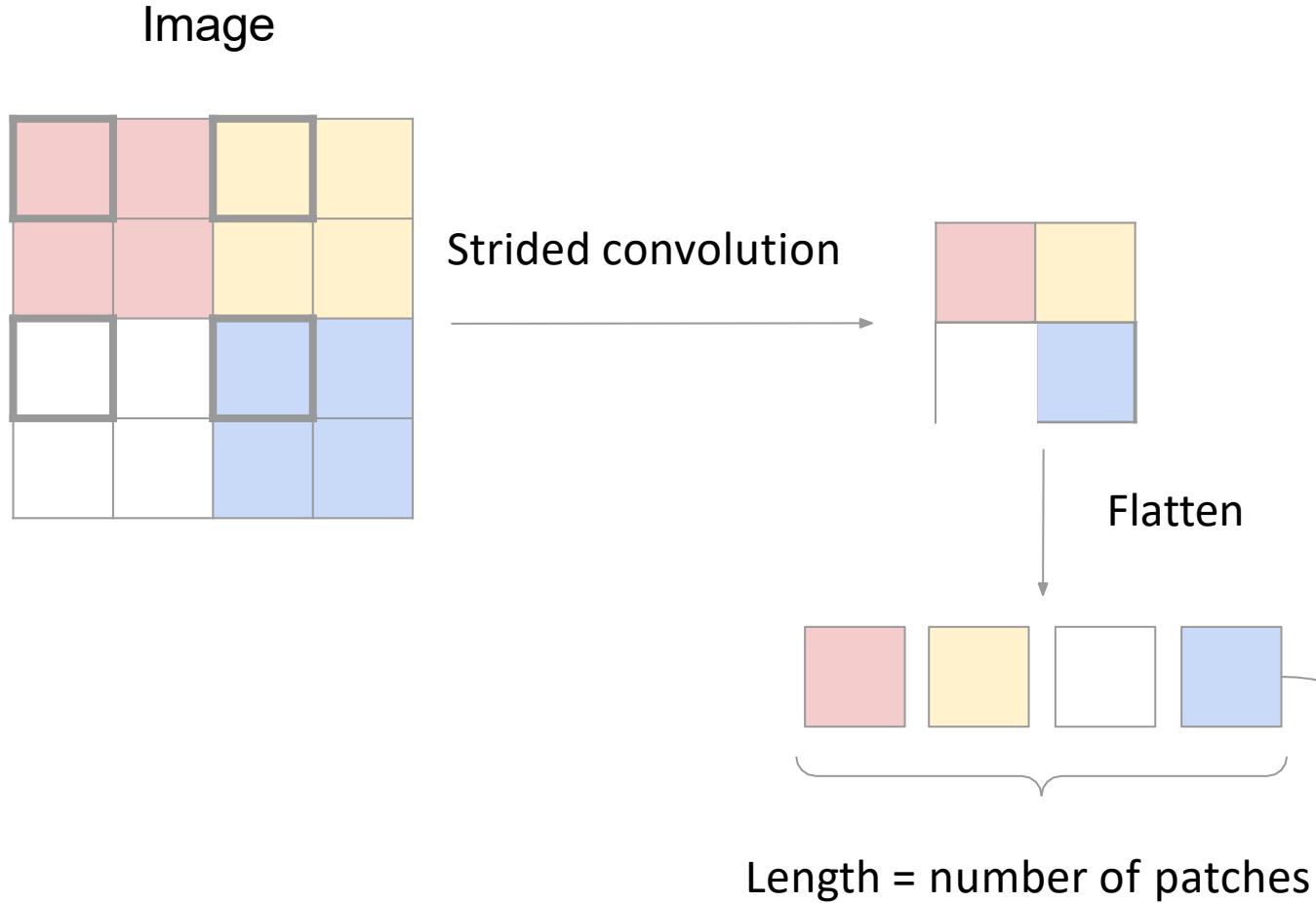
truck (9)



frog (6)

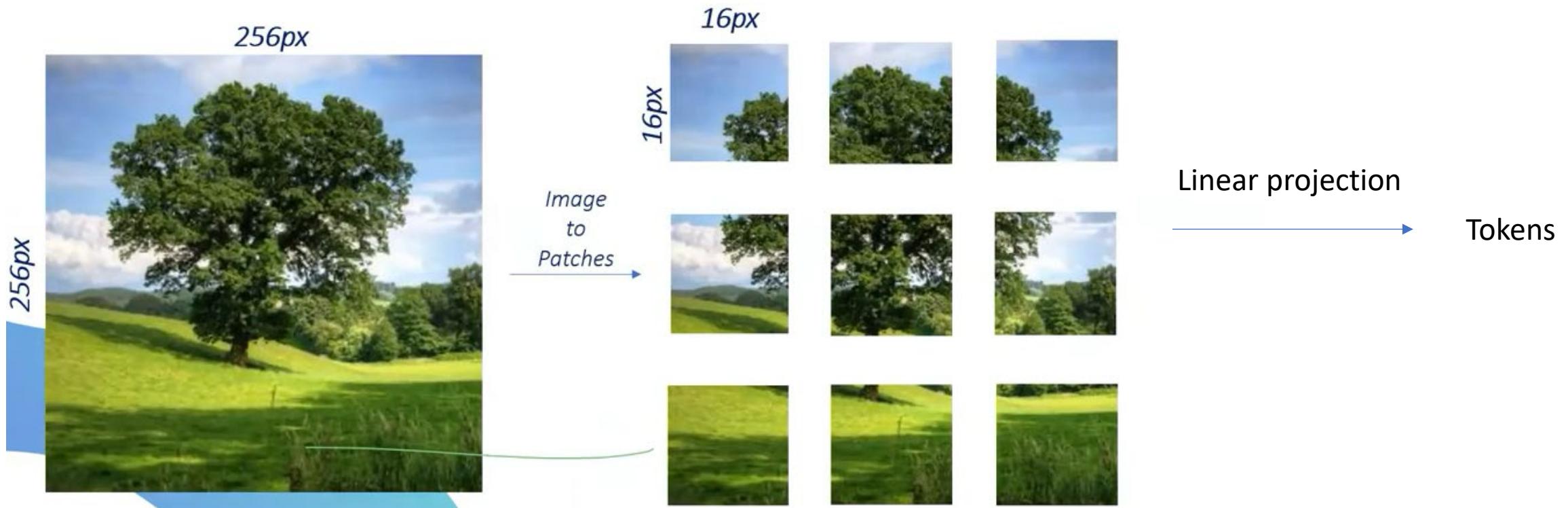
[Source](#)

# Idea: learnable downsampling

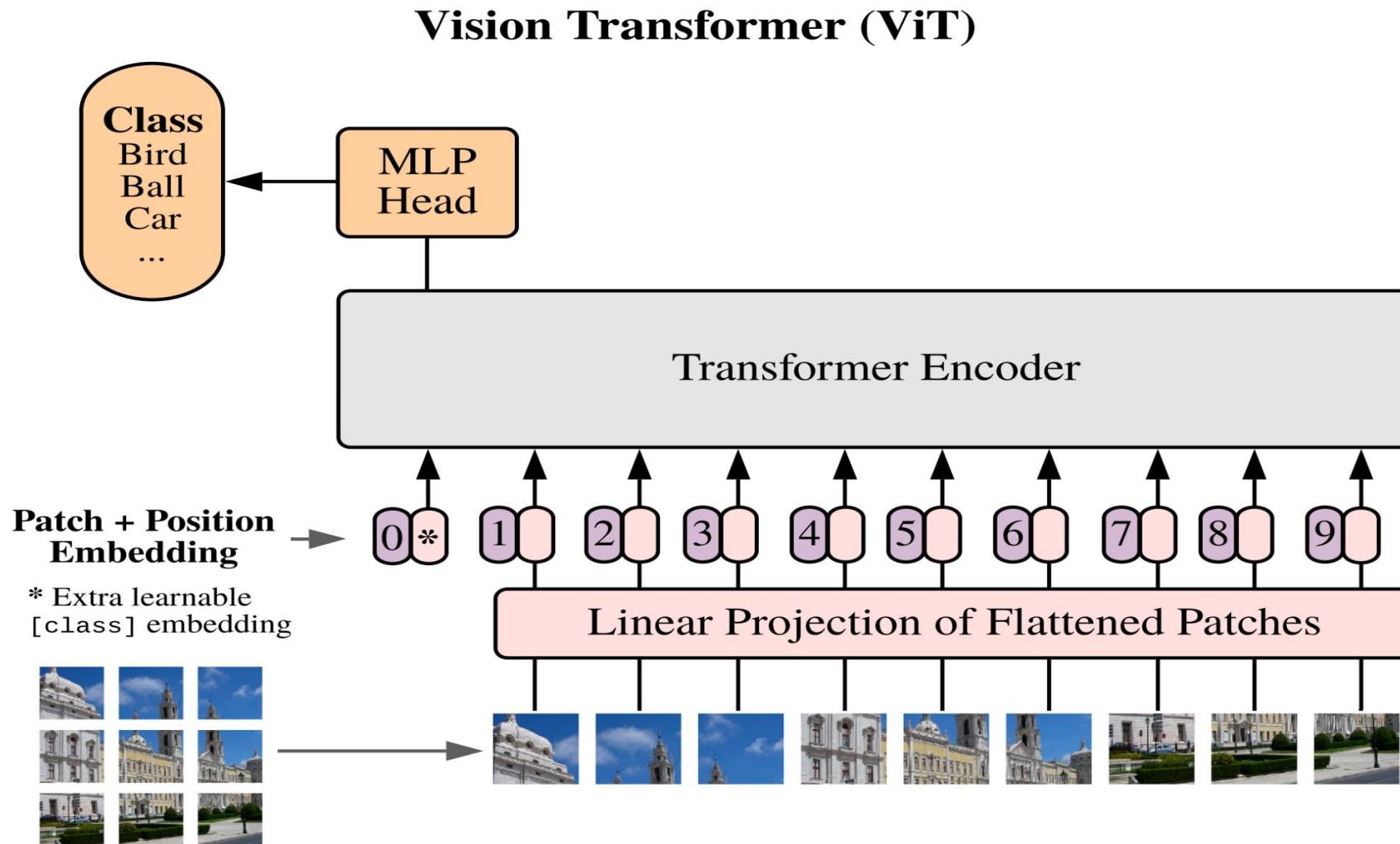


# Vision Transformers (ViTs)

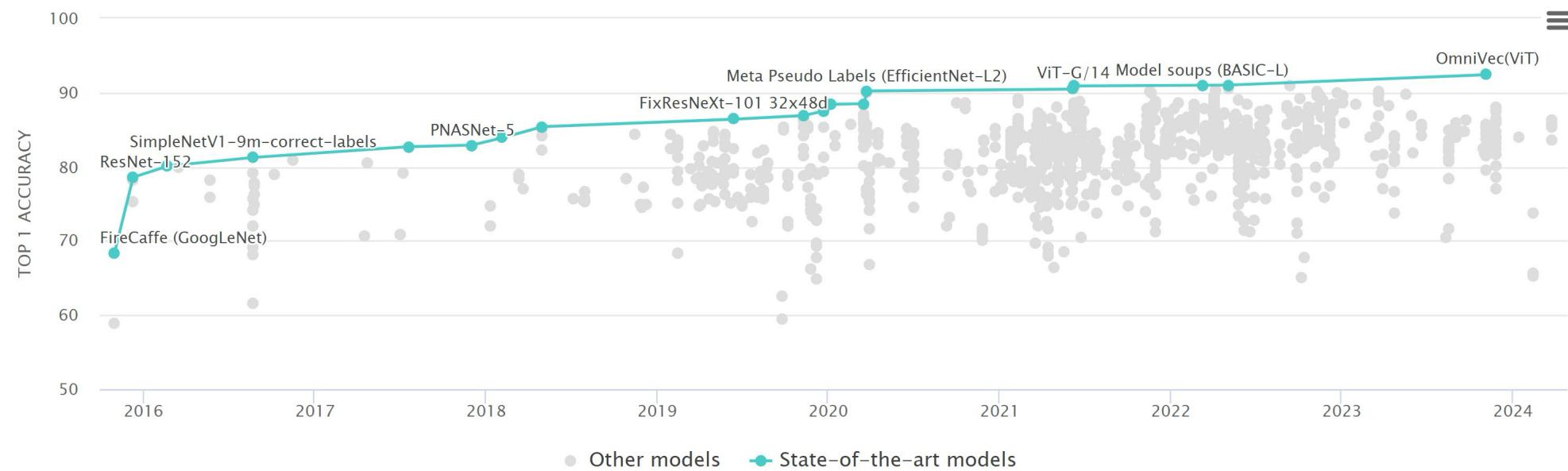
«An image is worth 16x16 words» paper



# Vision Transformer



# Image classification on ImageNet



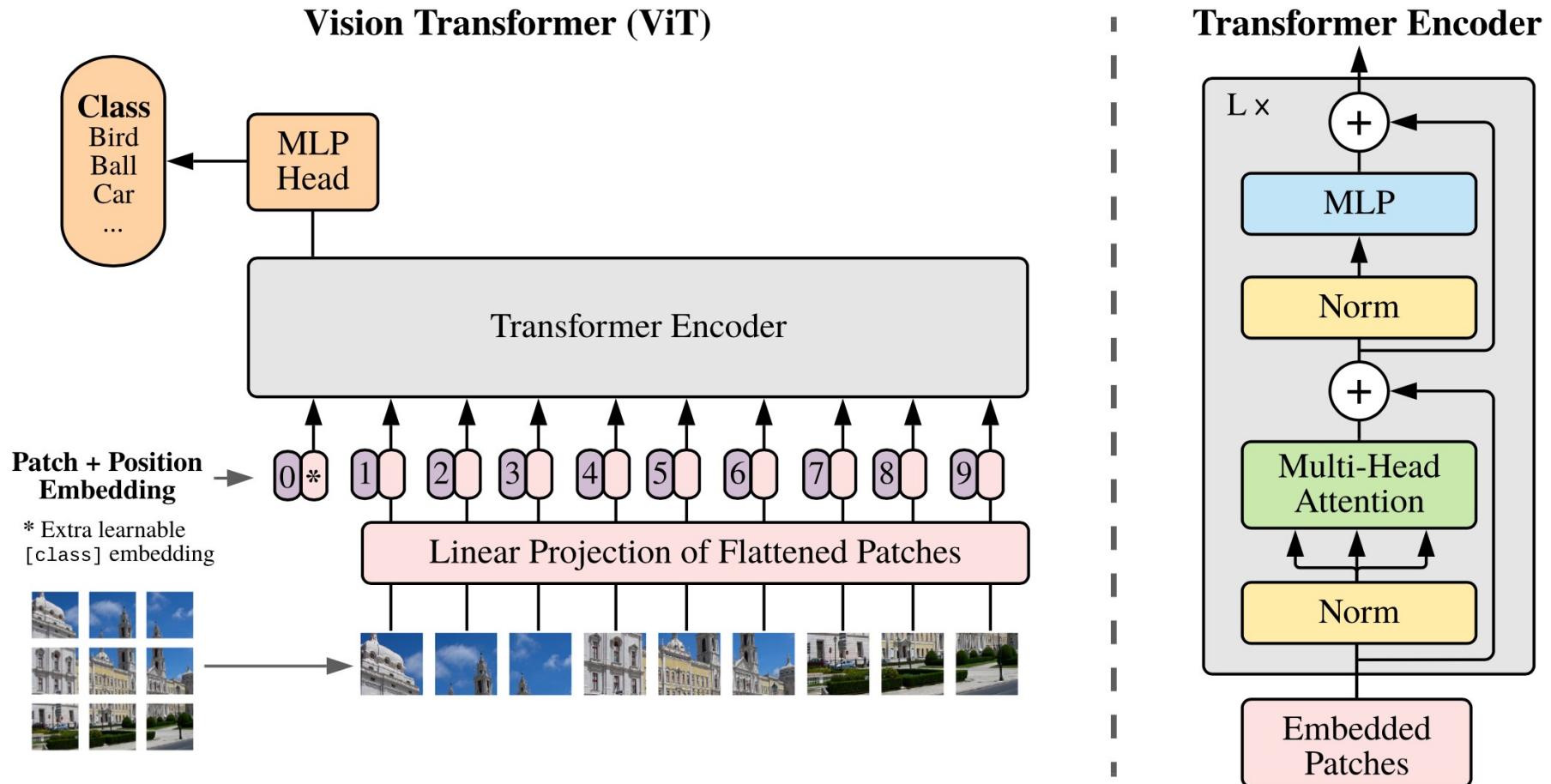
# Key concepts

- **Applying a Transformer Directly to Images**
  - with the fewest possible modifications
  - provide the sequence of linear embeddings of the patches as an input
  - image patches = tokens (words) in NLP
- **Small Scale Training**
  - achieved accuracies below ResNets of comparable size
  - Transformers lack some inductive biases inherent to CNNs (such as translation equivariance and locality)
    - Convolutions are translation invariant, locality sensitive, and lack a global understanding of images.
    - Transformers can be used in convolutional pipelines to produce global representations of images.

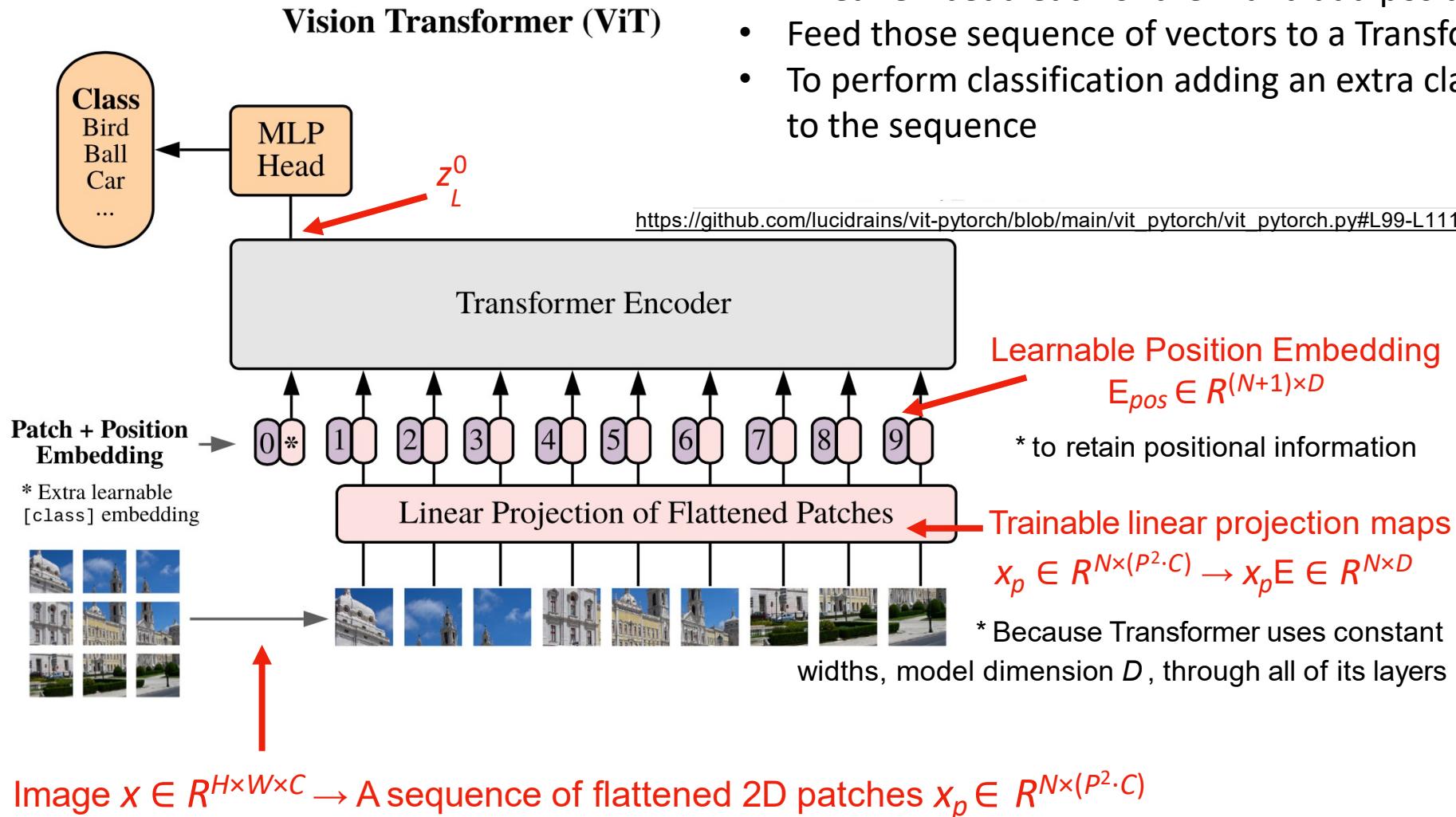
## Large Scale Training

- surpass inductive bias
- excellent results when pre-trained and transferred

# Method



# Method



# Step by step

1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the flattened patches
4. Add positional embeddings
5. Feed the sequence as an input to a standard transformer encoder
6. Pretrain the model with image labels (fully supervised on a huge dataset)
7. Finetune on a smaller dataset for image classification

# Experiments

- **Datasets**

Pre-training

- ILSVRC-2012 ImageNet dataset : 1k classes / 1.3M images
- ImageNet-21k : 21k classes / 14M images
- JFT : 18k classes / 303M images

Downstream (Fine-tuning)

- ImageNet, ImageNet ReaL, CIFAR-10/100, Oxford-IIIT Pets, Oxford Flowers-102, VTAB

- **Model Variants**

Model	Layers	Hidden size $D$	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

# Datasets

	# of Images	# of Classes
ImageNet (Small)	1.3 Million	1 Thousand
ImageNet-21K (Medium)	14 Million	21 Thousand
JFT (Big)	300 Million	18 Thousand

# Experiments

- Comparison to State of the Art

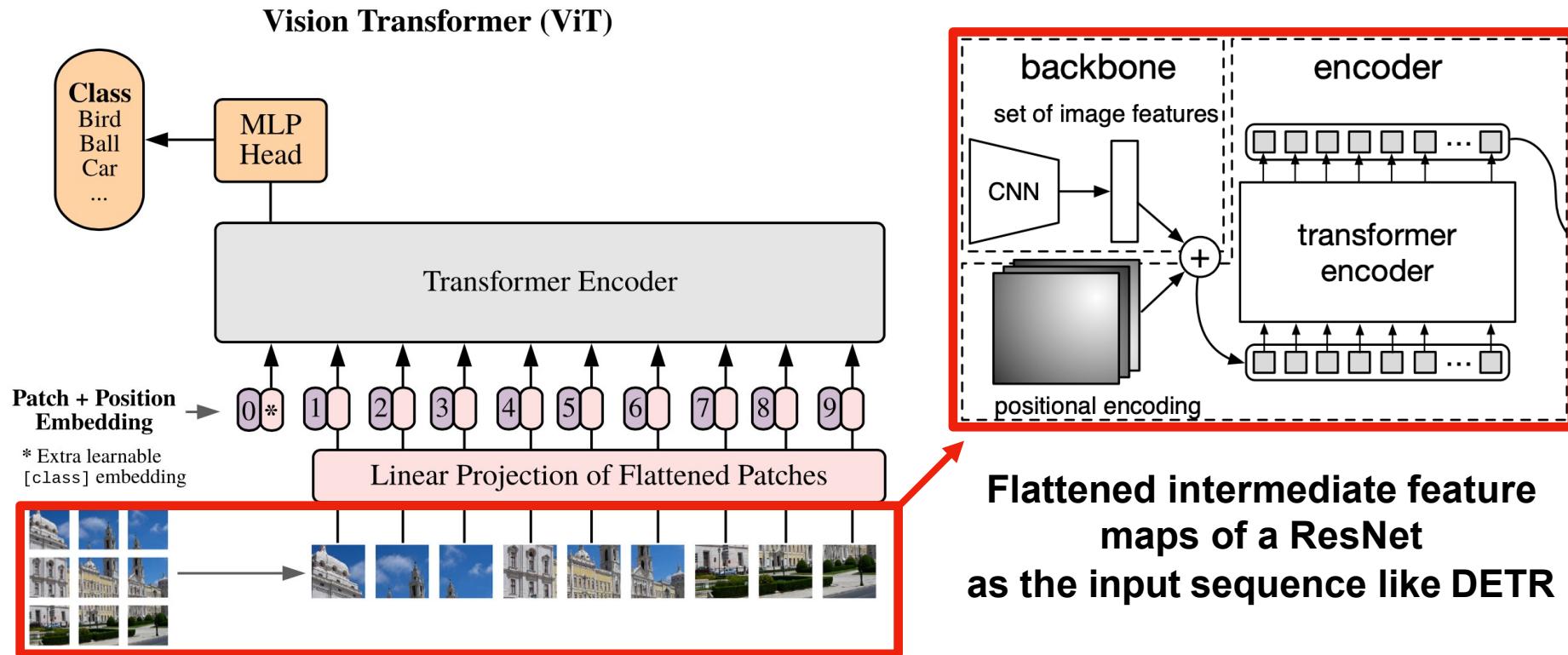
	Ours (ViT-H/14)	Ours (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.36	87.61 ± 0.03	87.54 ± 0.02	88.4/ <b>88.5*</b>
ImageNet ReAL	<b>90.77</b>	90.24 ± 0.03	90.54	90.55
CIFAR-10	<b>99.50</b> ± 0.06	99.42 ± 0.03	99.37 ± 0.06	—
CIFAR-100	<b>94.55</b> ± 0.04	93.90 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	<b>97.56</b> ± 0.03	97.32 ± 0.11	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	<b>99.74</b> ± 0.00	99.63 ± 0.03	—
VTAB (19 tasks)	<b>77.16</b> ± 0.29	75.91 ± 0.18	76.29 ± 1.70	—
TPUv3-days	2.5k	0.68k	9.9k	12.3k

Table 2: Comparison with state of the art on popular image classification datasets benchmarks. Vision Transformer models pre-trained on the JFT300M dataset often match or outperform ResNet-based baselines while taking substantially less computational resources to pre-train. \*Slightly improved 88.5% result reported in Touvron et al. (2020).

\* BiT-L : Big Transfer, which performs supervised transfer learning with large ResNets

\* Noisy Student : a large EfficientNet trained using semi-supervised learning

# Hybrid Architecture



# Conclusion and Challenges

- **Application of Transformers to Image Recognition**
  - no image-specific inductive biases in the architecture
  - interpret an image as sequence of patches and process it by a standard Transformer encoder (self-attention and layer norm)
  - it is highly scalable and highly parallelizable
  - faster training, larger models, better performance across vision and language tasks
  - matches or exceeds the S.O.T.A being cheap to pre-train
- **Challenges**
  - The scale and data problem

# Survey of self-attention applications in Computer Vision

# Some cool applications

- [Image GPT \(Generative Pretraining from Pixels\)](#), uses transformer for pixel level image completion, just like other GPT for text completion
- Dall-e 2 <https://openai.com/dall-e-2/> and Dall-e 3

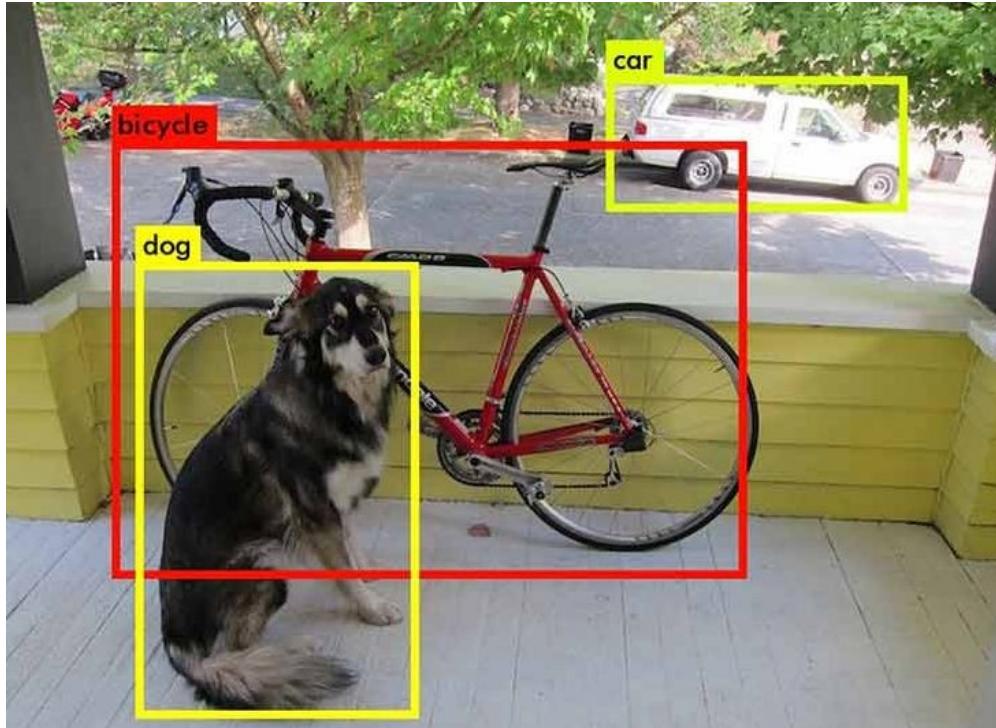


"A photo of a confused grizzly bear in Computer programming class"  
**#dalle2 #dalle**

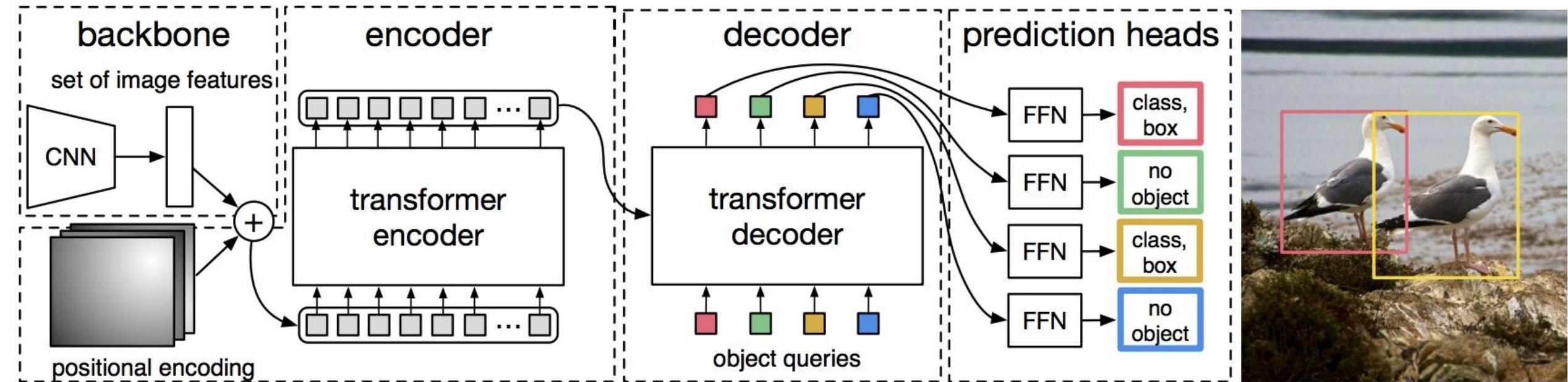
[Traduci il Tweet](#)



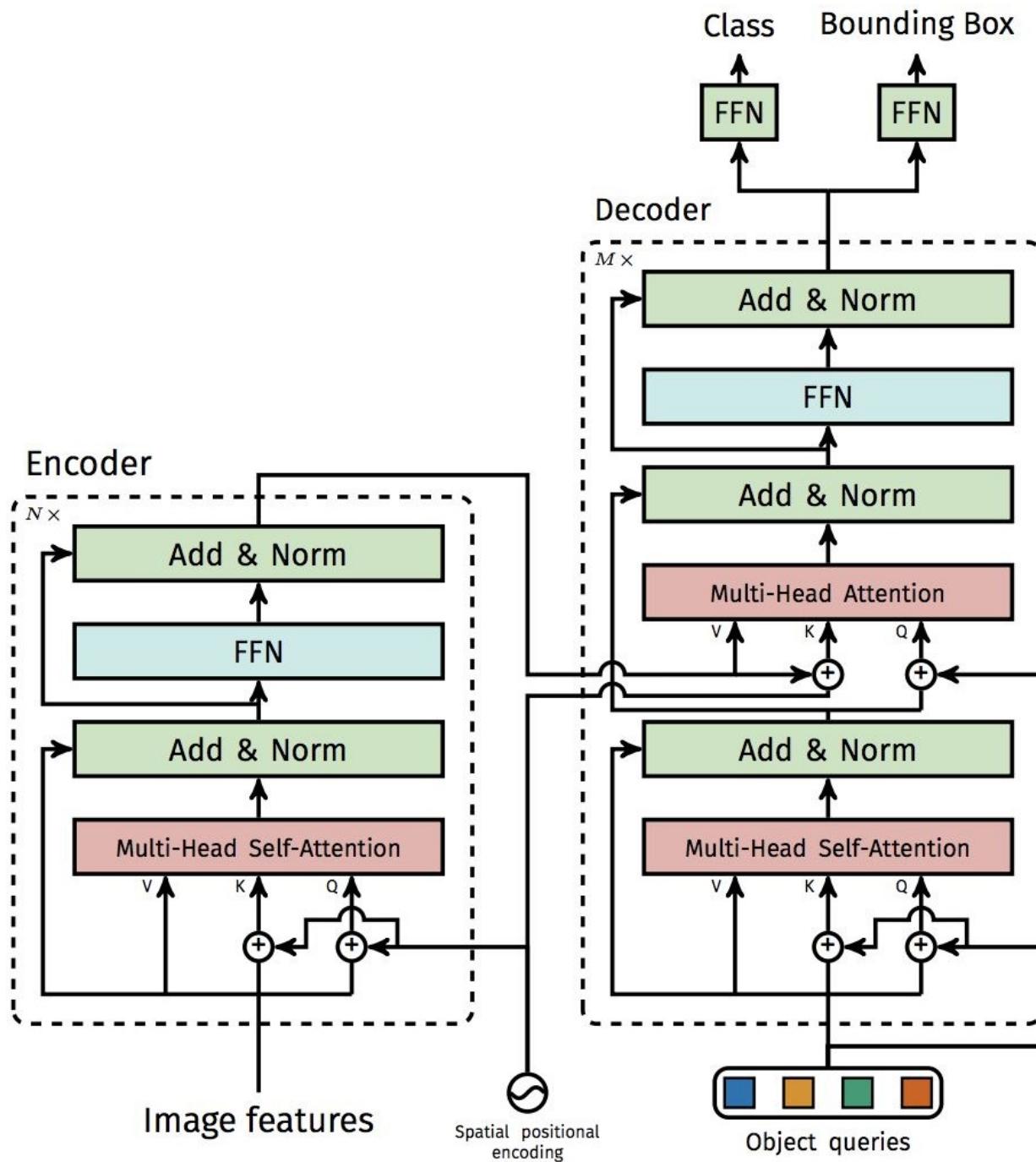
# Transformers for Object Detection



# DETR: End-to-End Object Detection with Transformers



# DETR



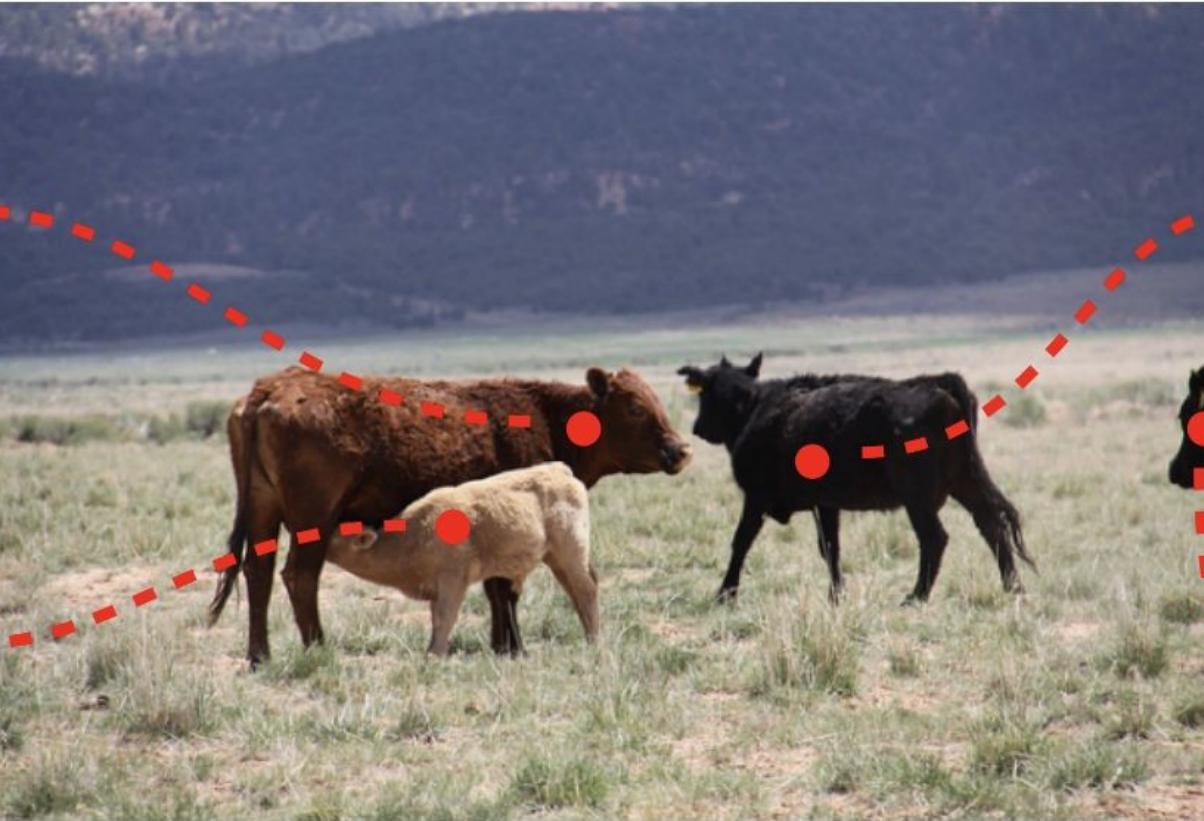
Carion et al 2020

# DETR

Model	GFLOPS/FPS	#params	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	<b>47.8</b>	<b>27.2</b>	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	<b>44.9</b>	<b>64.7</b>	47.7	23.7	<b>49.5</b>	<b>62.3</b>

# DETR

self-attention(430, 600)



self-attention(450, 830)



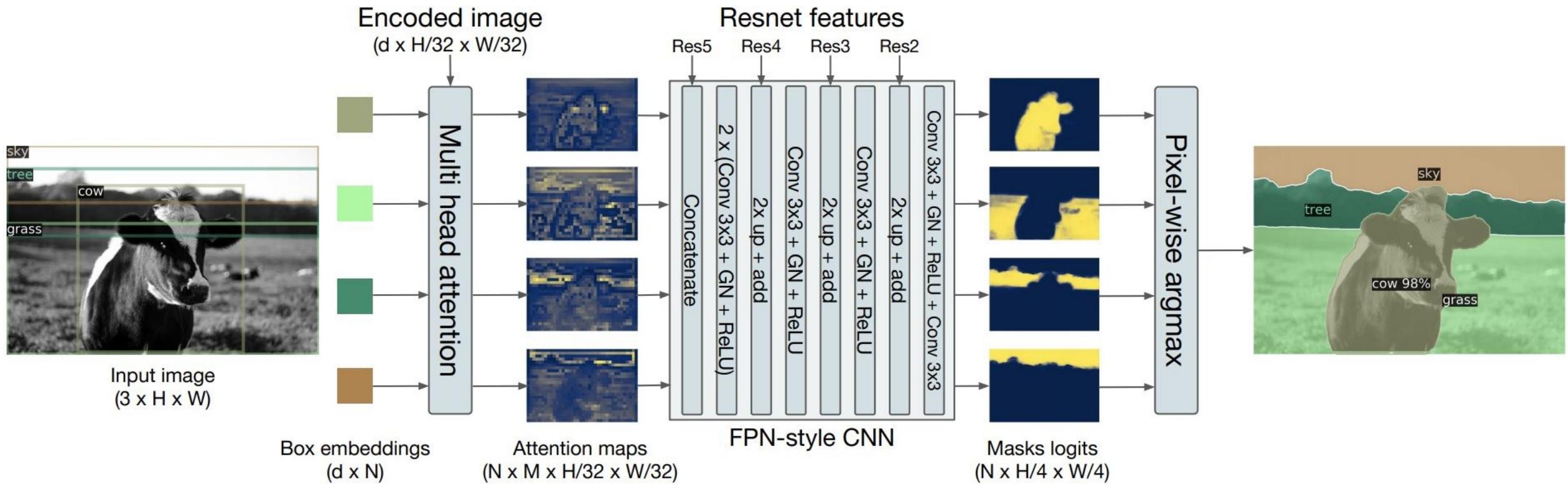
self-attention(520, 450)



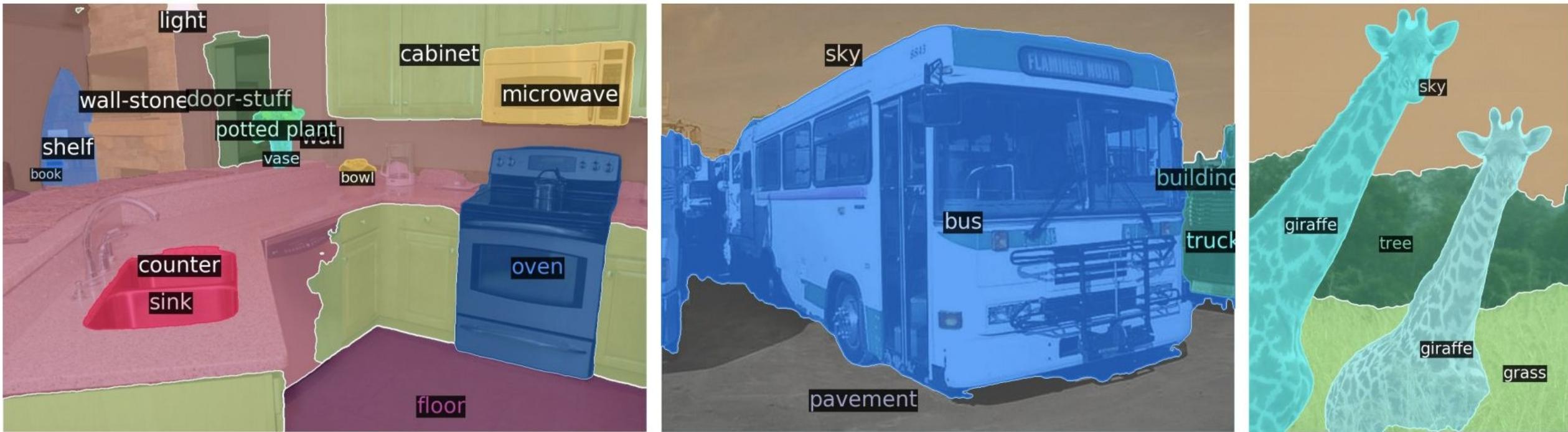
self-attention(440, 1200)



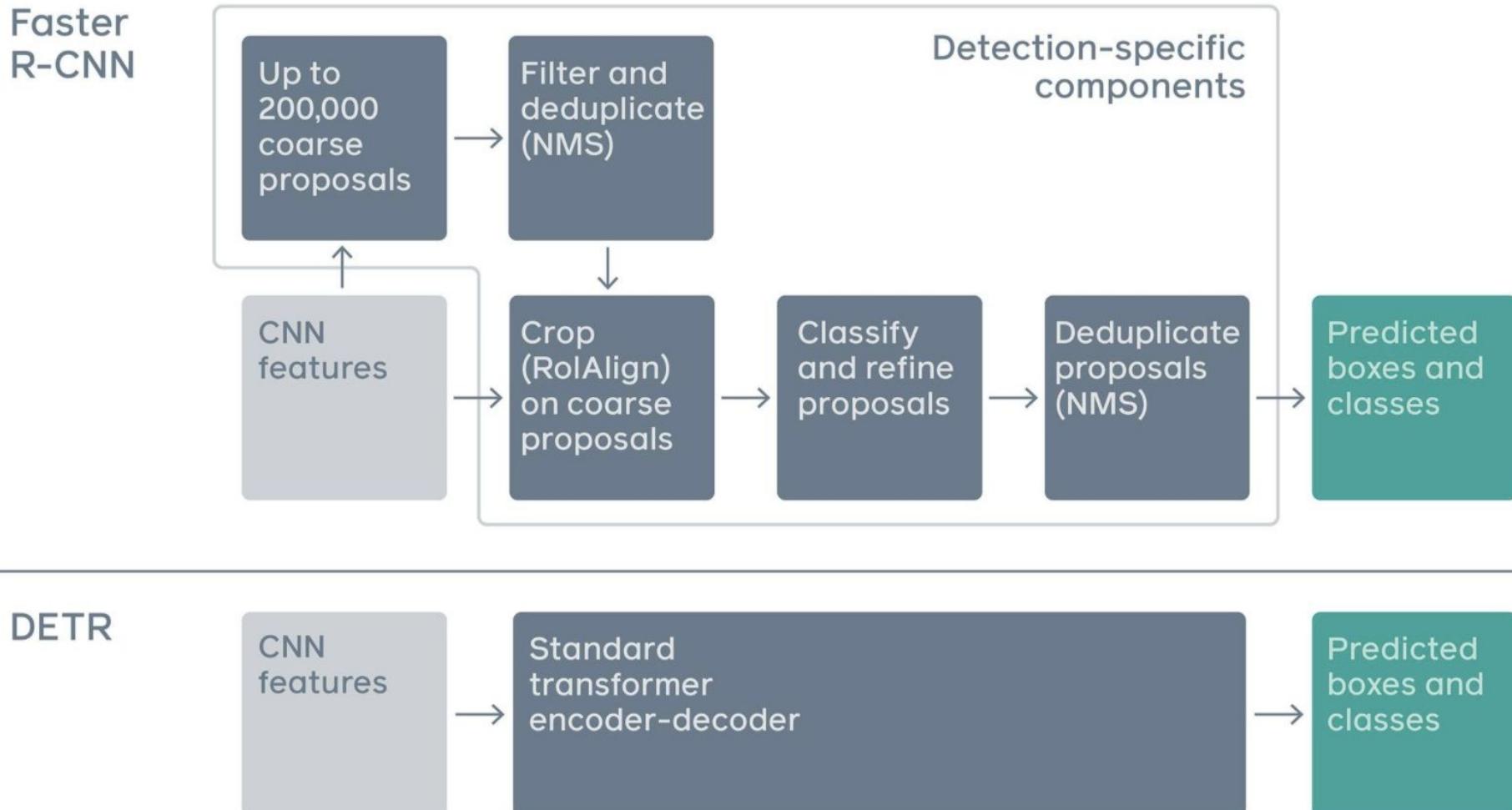
# DETR can be modified to perform panoptic segmentation



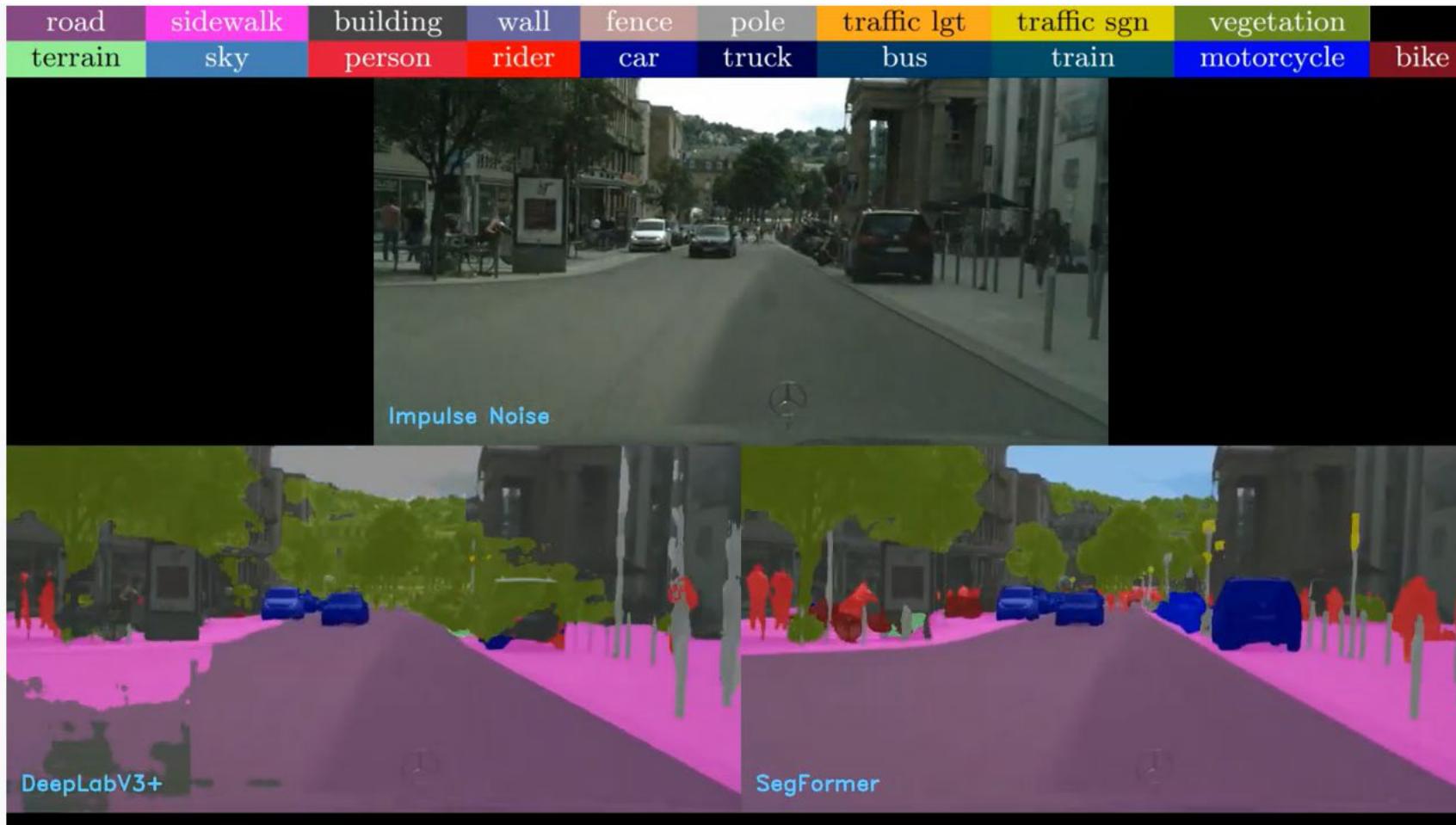
# DETR can be modified to perform panoptic segmentation



# DETR Inference Code is simpler

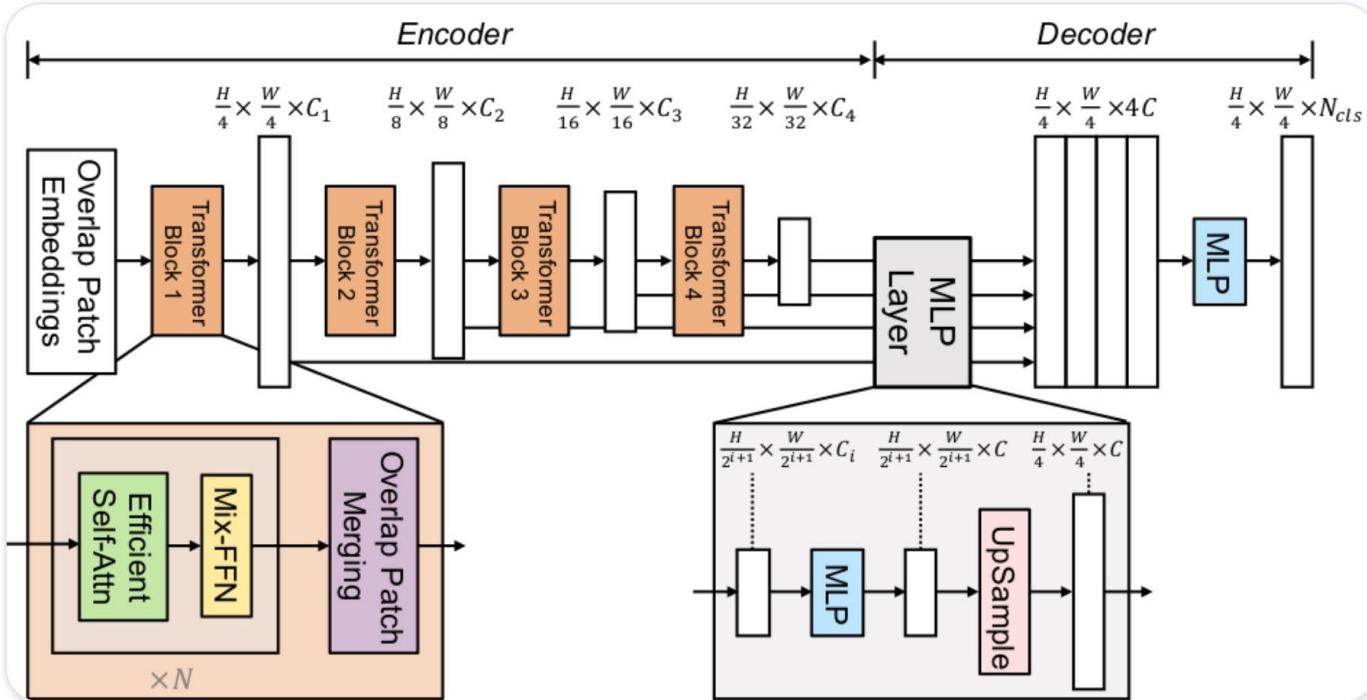


# Transformers for Semantic Segmentation



<https://www.youtube.com/watch?v=J0MoRQzZe8U>

# SegFormer



- 1) SegFormer comprises a novel hierarchically structured Transformer encoder which outputs multiscale features. It does not need positional encoding.
- 2) SegFormer avoids complex decoders. The proposed MLP decoder aggregates information from different layers, and thus combining both local attention and global attention to render powerful representations.

# Transformers for Video Recognition

# TimeSformer

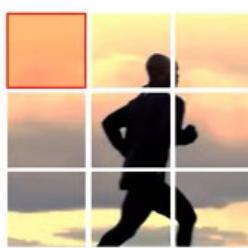
Combine space and time attention with Divided Space-Time Attention



*Time*

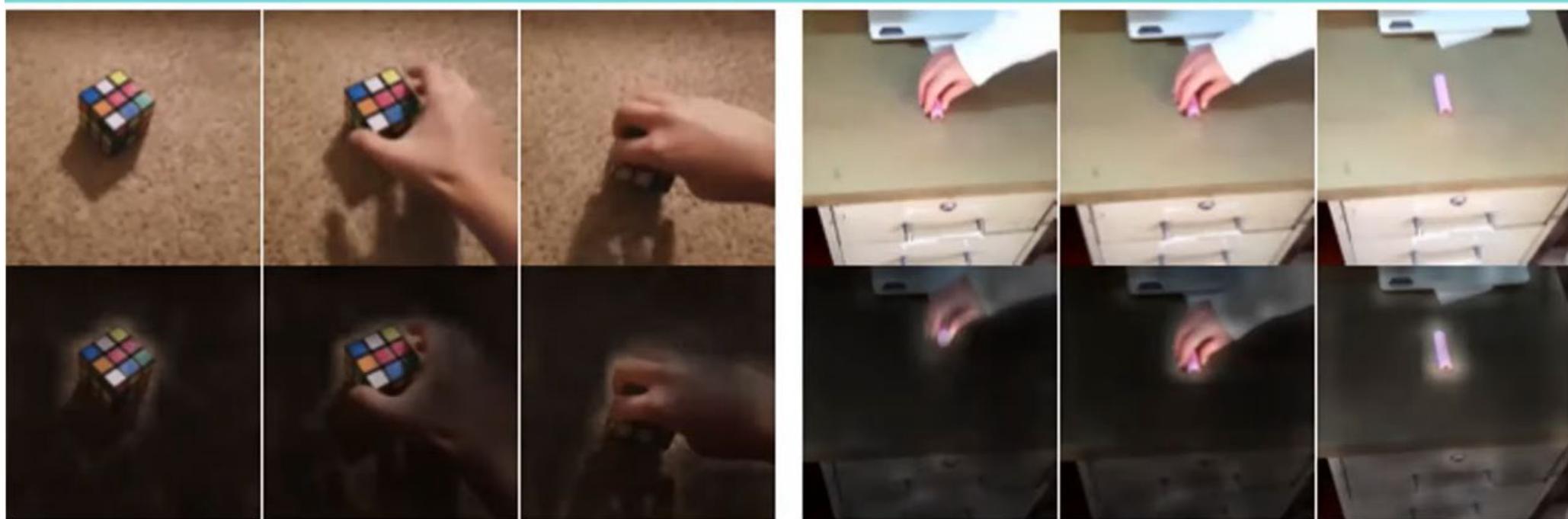


*Space*



*Time*

# TimeSformer

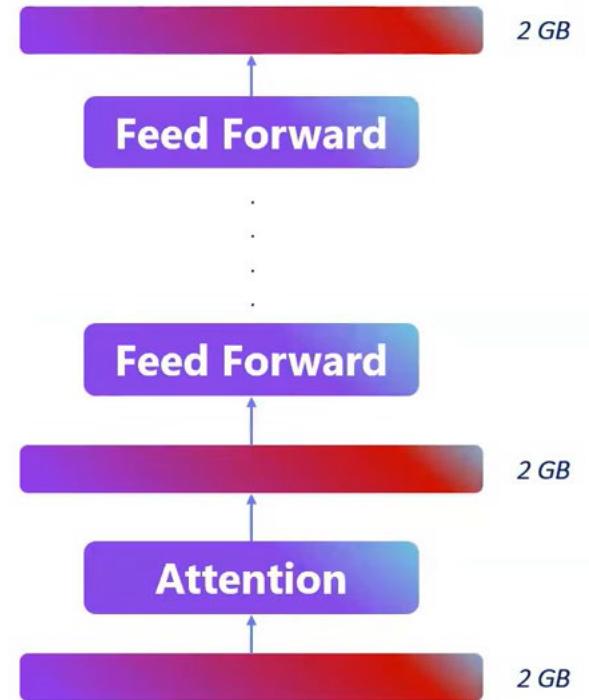
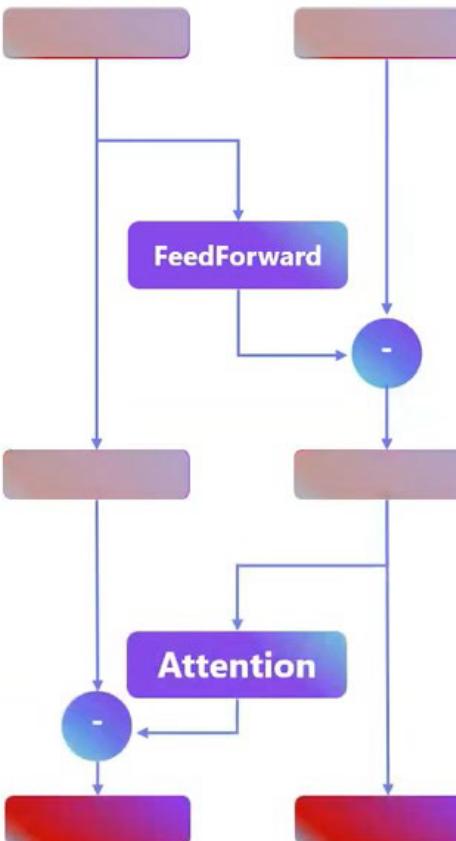
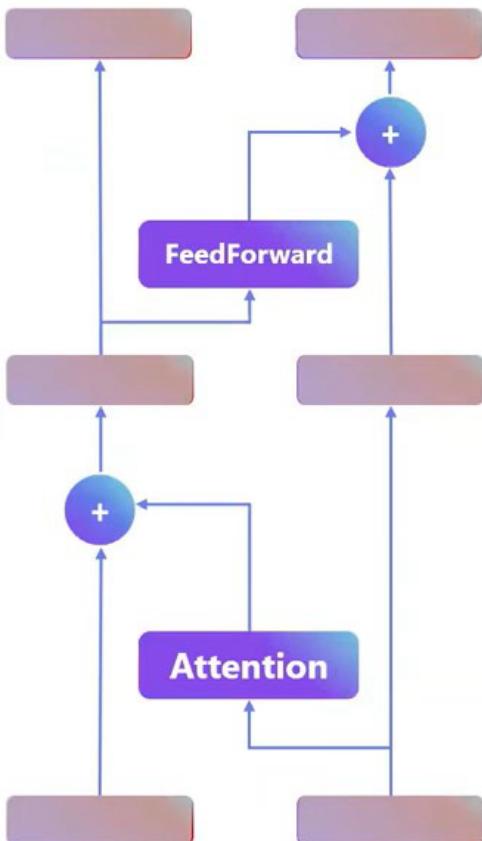


Focus on the main object  
Track the object through the frames

# Efficient Transformer

Transformers in the original form use a lot of memory

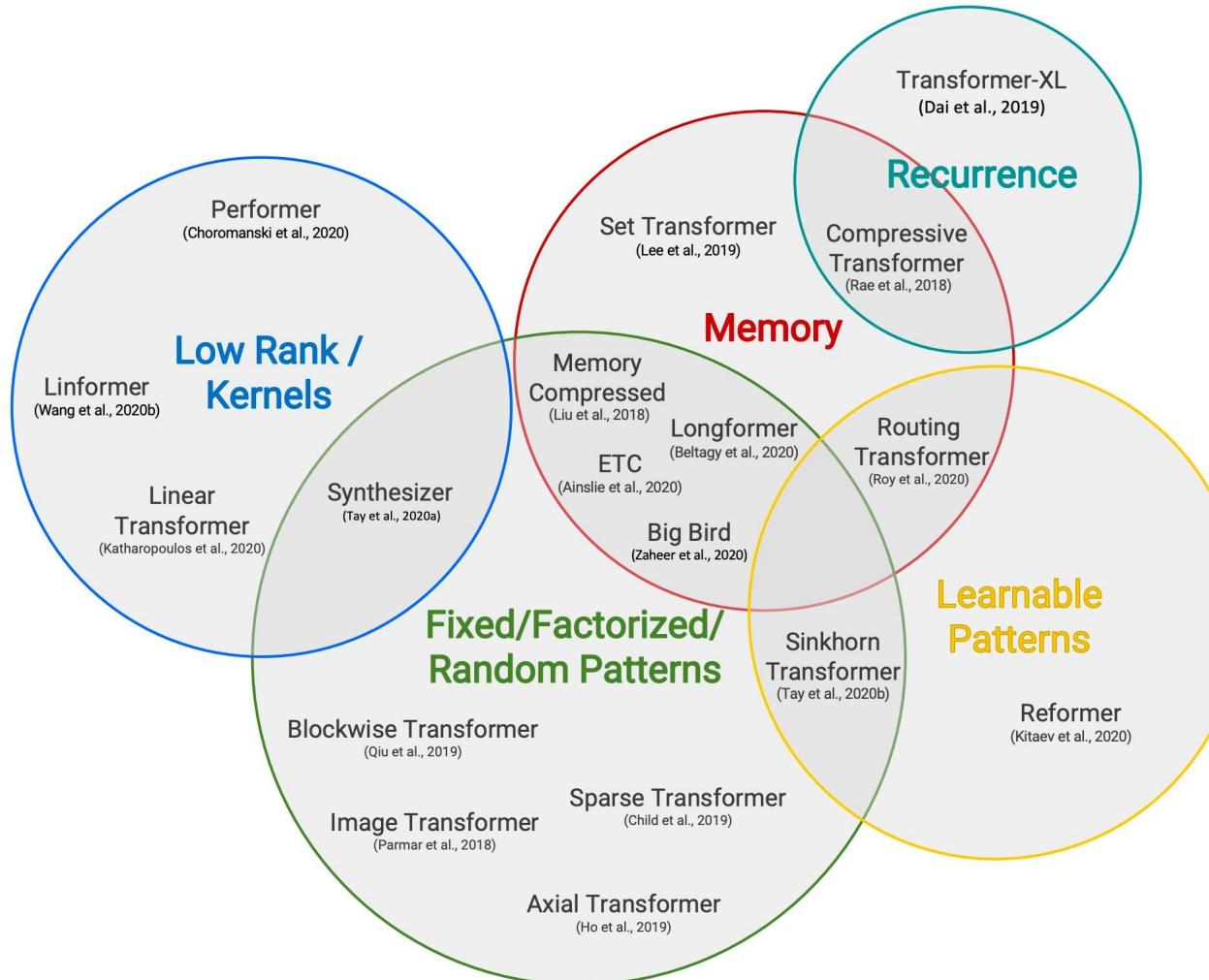
Only final tokens are stored



List of the input tokens  
At each step a tensor is produced  
to be maintained in memory

# Lot of ideas to try out!

Transformers in the original form use a lot of memory



[Tay et al. 2020. Efficient Transformers: A Survey.](#)

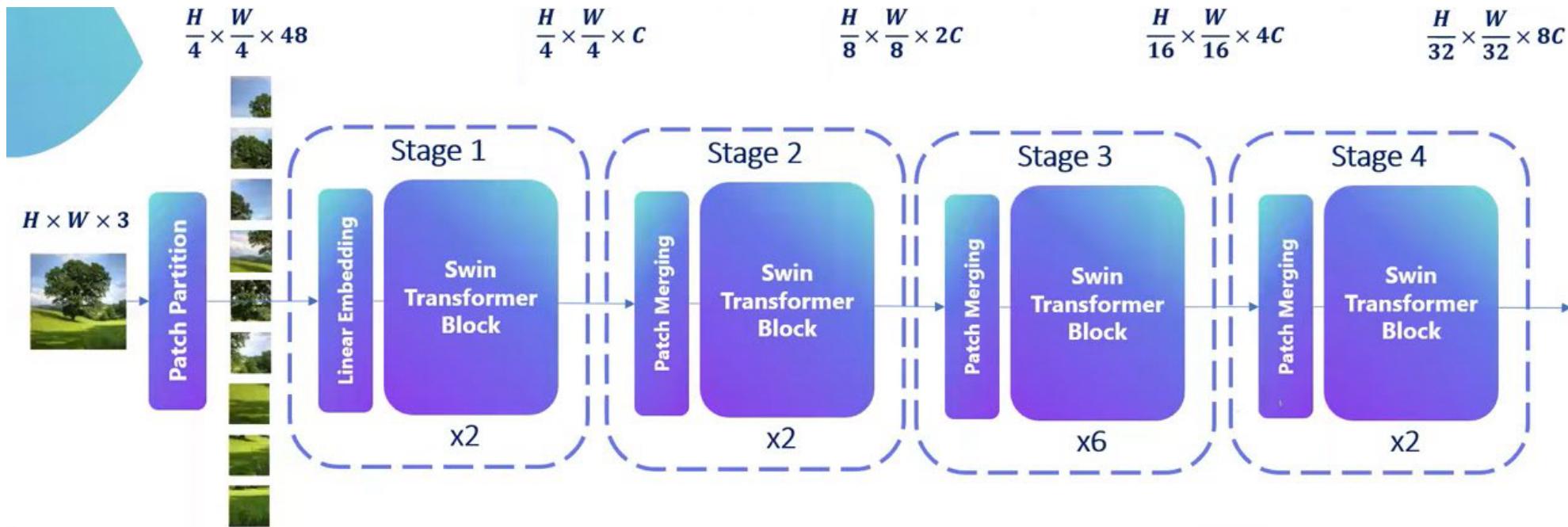
Bohan Zhuang et al. "A Survey on Efficient Training of Transformers". In: Aug. 2023, pp. 6823–6831. DOI: [10.24963/ijcai.2023/764](https://doi.org/10.24963/ijcai.2023/764).

L. Papa, P. Russo, I. Amerini and L. Zhou, "A Survey on Efficient Vision Transformers: Algorithms, Techniques, and Performance Benchmarking," in IEEE Transactions on Pattern Analysis and Machine Intelligence 2024, doi: [10.1109/TPAMI.2024.3392941](https://doi.org/10.1109/TPAMI.2024.3392941)

# Multi-Scale Features in Transformers

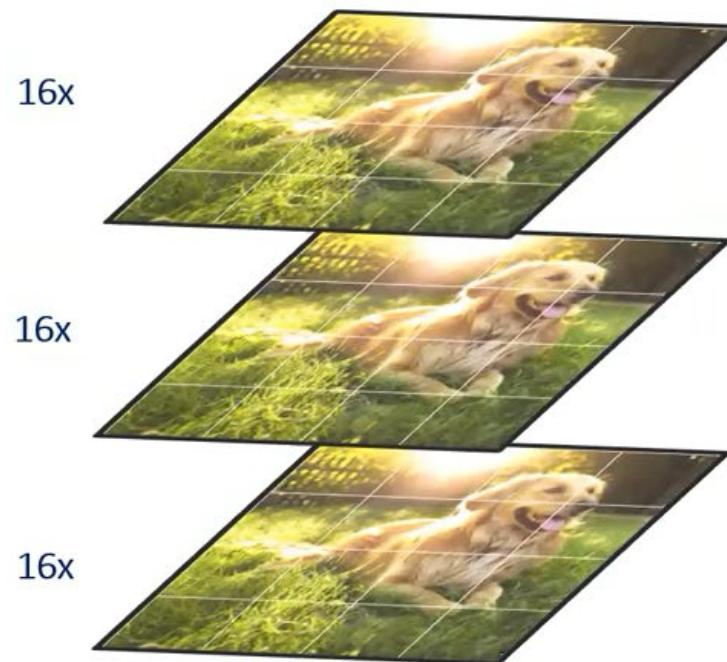
# Swin Transformer

Shifted-window based self-Attention

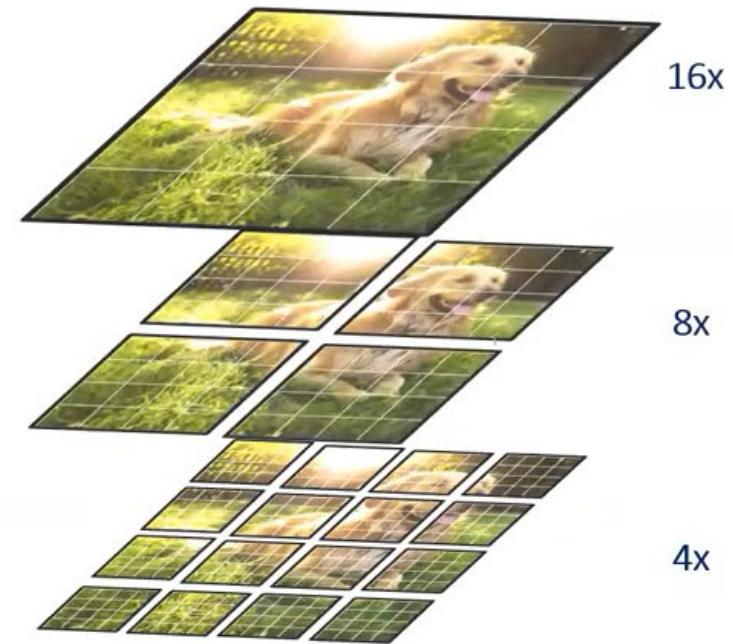


# Swin Transformer

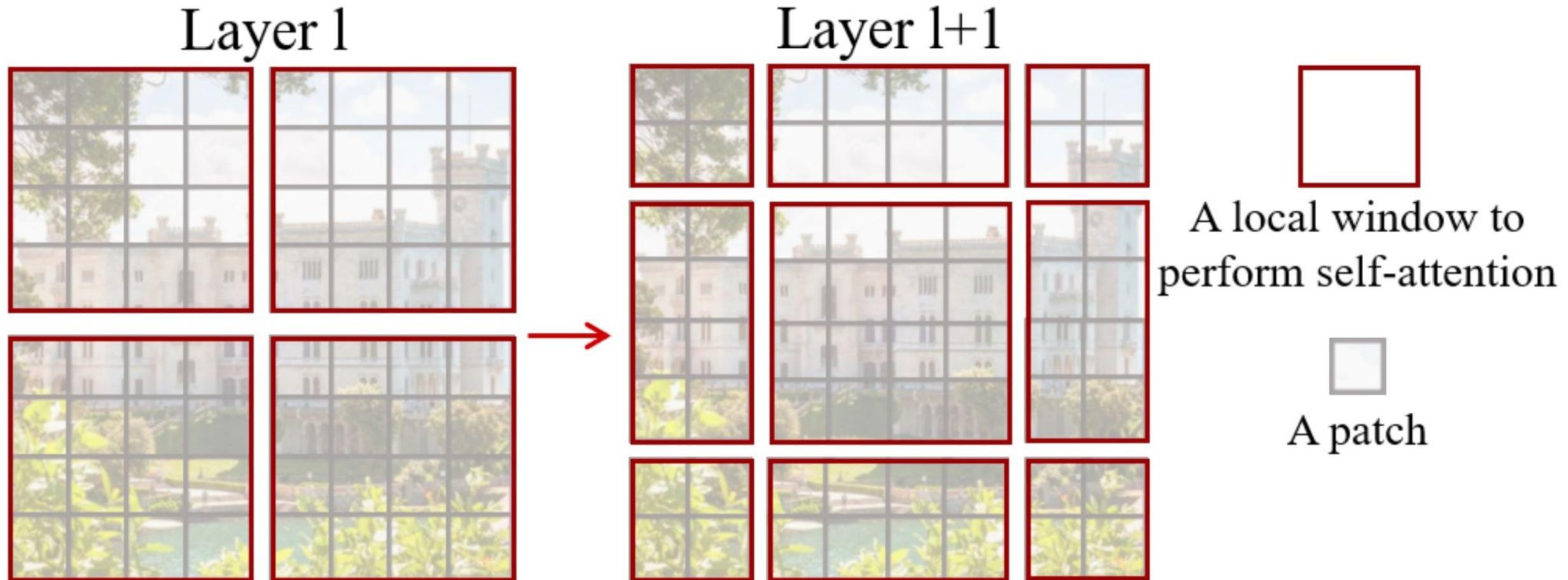
**Vision Transformer**



**Swin Transformer**



# Swin Transformer: Shifted window



# Swin Transformer

[youtube.com/watch?v=FQVS\\_0Bja6o](https://youtube.com/watch?v=FQVS_0Bja6o)

State of the art on COCO dataset 2021-2022 now Co-DETR <https://paperswithcode.com/sota/object-detection-on-coco>



# Transformers for Self-Supervised Learning

# DINO



<https://www.youtube.com/watch?v=h3ij3F3cPlk>

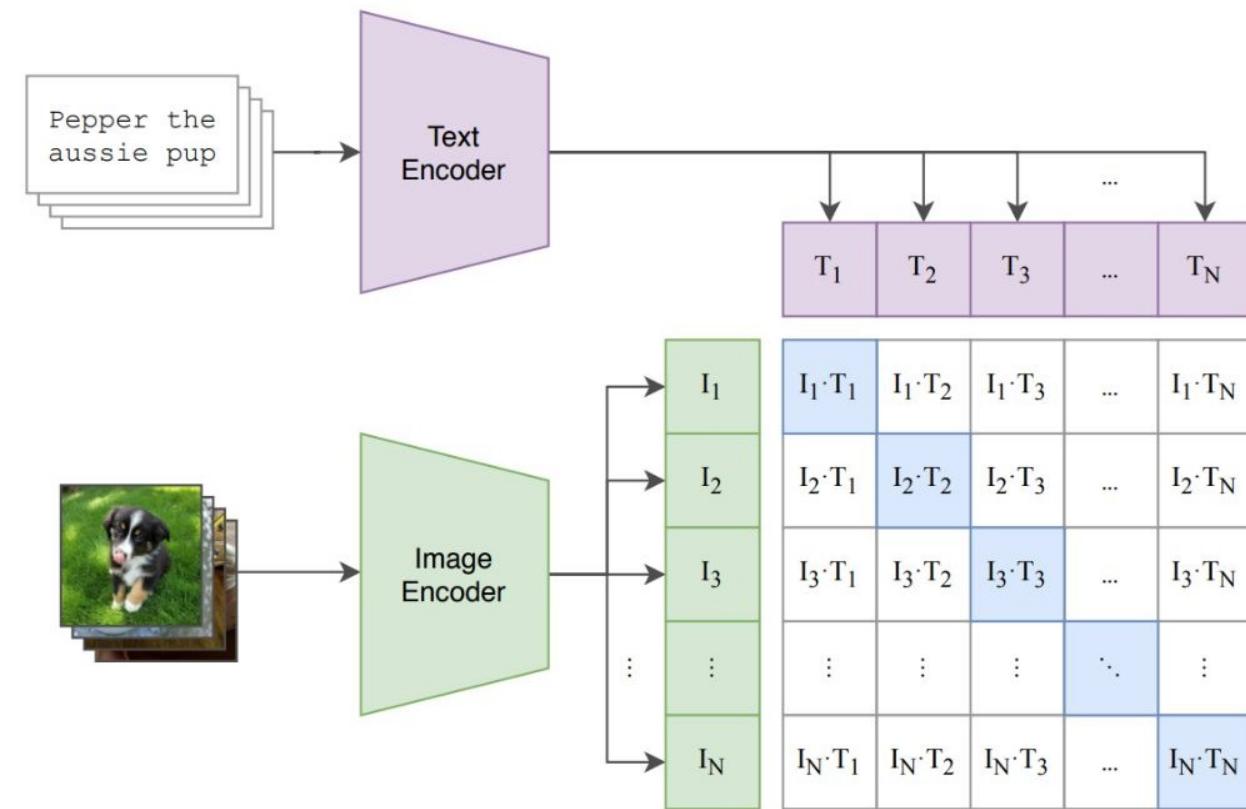
The network learns through a process called ‘self-distillation’. There is a teacher and student network both having the same architecture, a **Vision Transformer(ViT)**.

[Caron et al 2020](#)

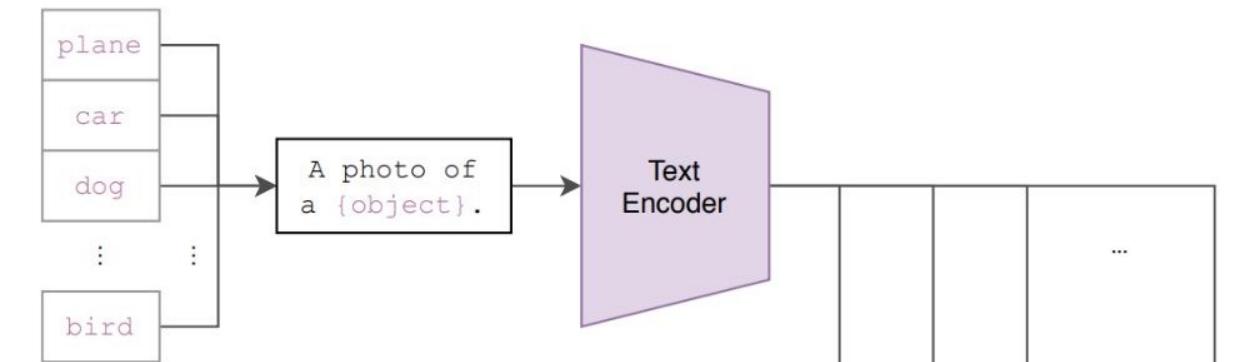
# Transformers for Multi-Modal Learning

# CLIP (*Contrastive Language–Image Pre-training*)

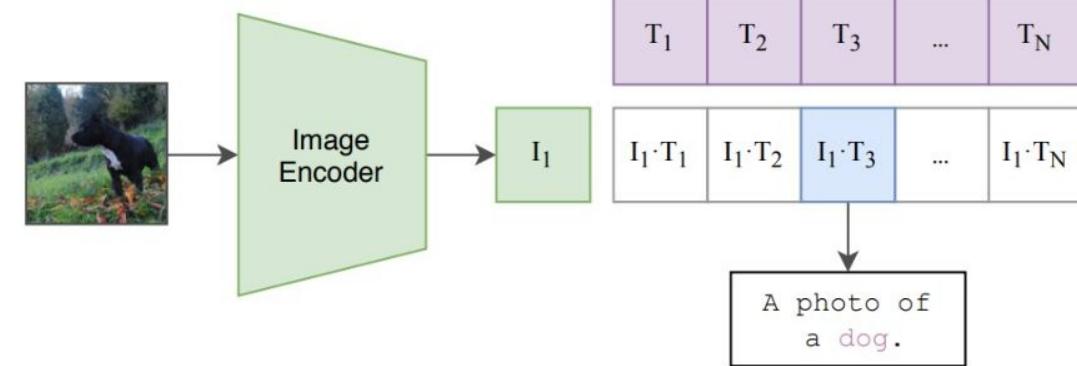
(1) Contrastive pre-training



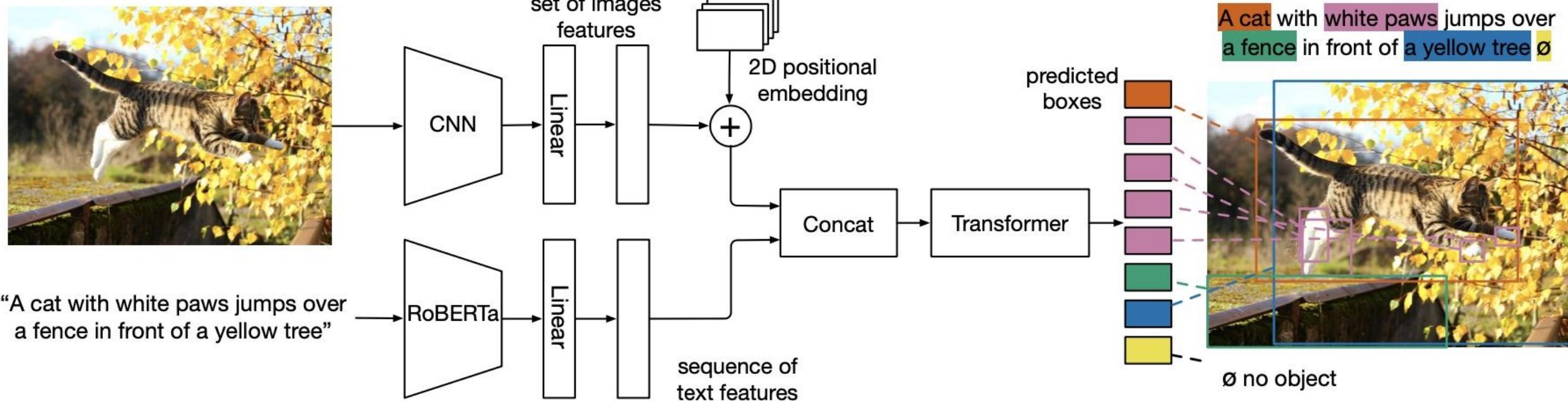
(2) Create dataset classifier from label text



(3) Use for zero-shot prediction



# Multimodal DETR (M-DETR)



# M-DETR



# Takeaways

- Pure attention models require a lot of data OR data-augmentations and regularization for ~SoTA performance
- Hybrid and (or) multi-scale models perform best (efficient for the same high accuracy) across all data regimes
- Huge promise for multimodal (combining with language) and the promise is fulfilled (see ChatGPT image generator and not only!)

