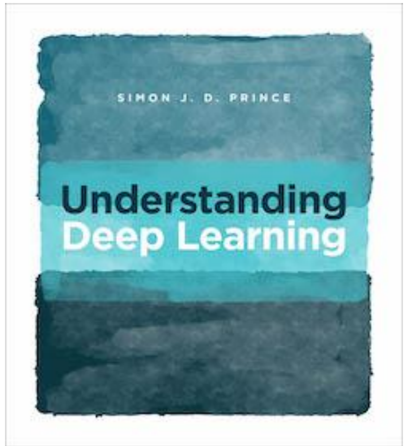
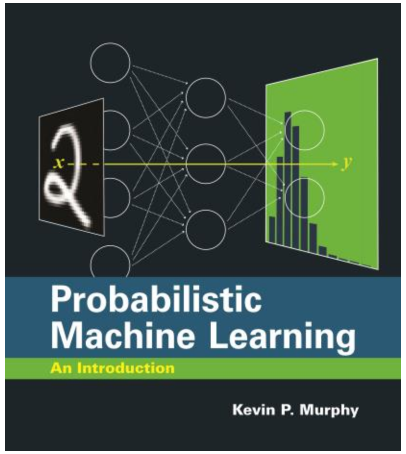
Transformers for Vision

Computer Vision (SJK02)

Universitat Jaume I



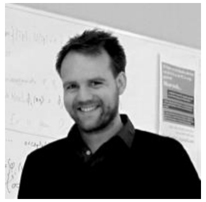
[Understanding Deep Learning (MIT, 2023)]
Chapter 12: Transformers



[Probabilistic Machine Learning (MIT 2022)]
Chapter 15: NN for sequences

Luis Serrano
[Attention mechanisms]
[Attention with maths]
[Transformers]
[Transformer models]

Jay Alammar
[The Illustrated Transformer]



Peter Bloem
[Transformers from scratch]

Why are they called "transformers"?

Hypothesis 1: Just marketing
Hypothesis 2: Paradigm shift
Hypothesis 3: Inputs are transformed
Other?

A bottom-up explanation

Attention mechanism

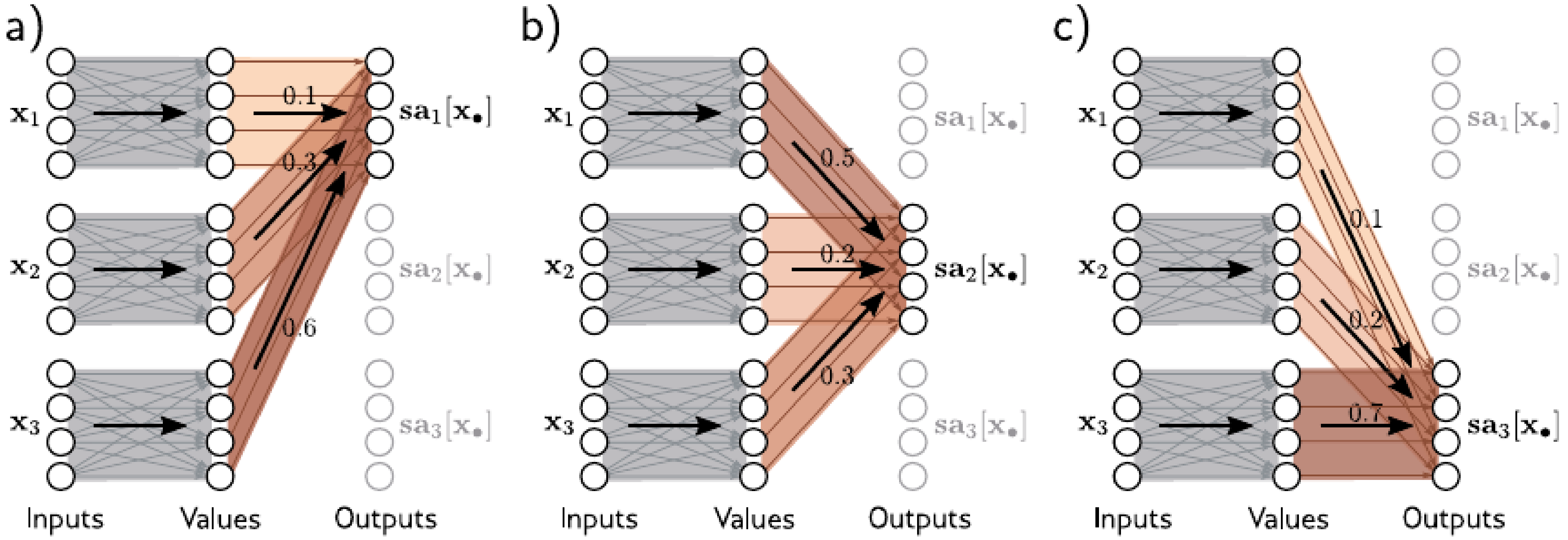
Standard layer: fixed operation

$$z = \sigma(Wv)$$

fixed

What about something more **flexible**, input-dependent?

Self-attention as "routing" of values



$$v_i = A_v \cdot x_i + b_v$$

$$sa_i = \sum_{j=1}^{N} \alpha_{ij} \cdot v_j$$

How many attention weights needed for input sequence of *N* elements?

Attention = soft dictionary lookup

q = query

V = values (set of feature vectors)

K = keys

If q is most similar to key *i, then use more value i*

$$\mathrm{Attn}\left(q, \{(k_i, v_i)\}\right) = \sum \alpha_i \cdot v_i$$

$0 \le \alpha_i \le 1$

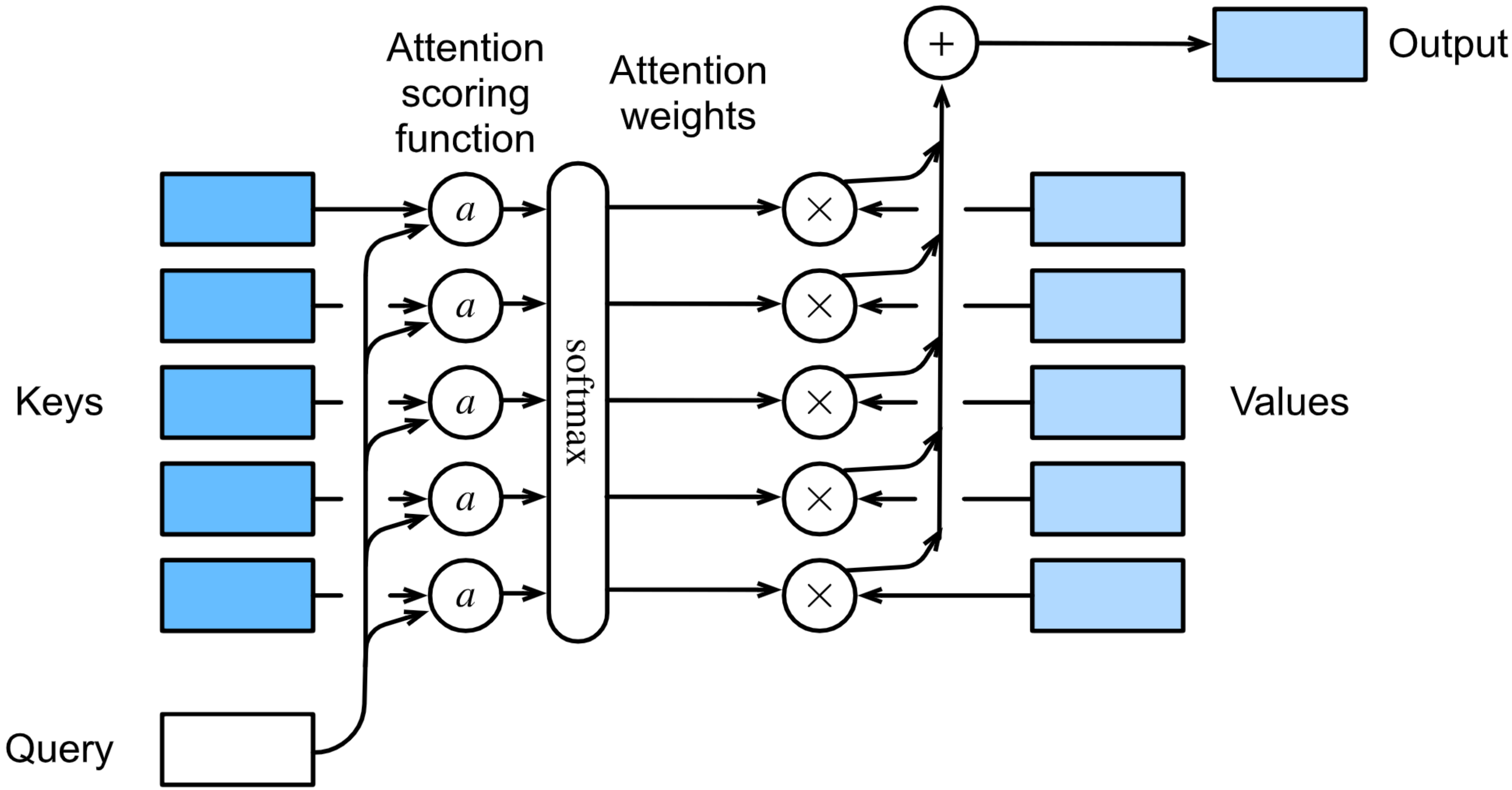
$\sum \alpha_i = 1$

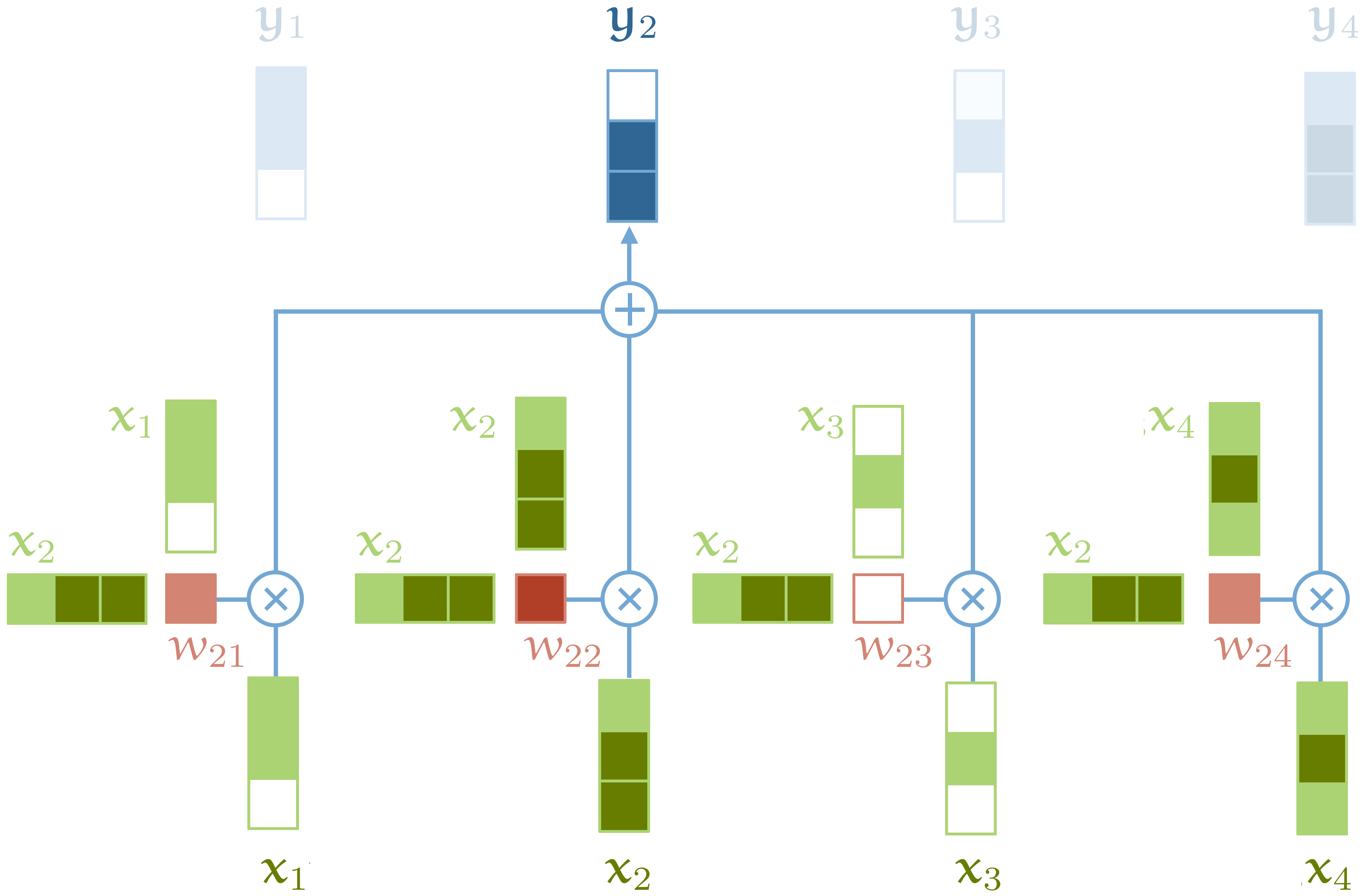
attention weight

attention score

$$\alpha_i = \frac{e^{a(q,k_i)}}{\sum_j e^{a(q,k_j)}}$$

Differentiable lookup


Attention
scoring
function
Attention
weights
Output
Keys
Values
Query
softmax
a
a
a
a
a



https://peterbloem.nl/blog/transformers

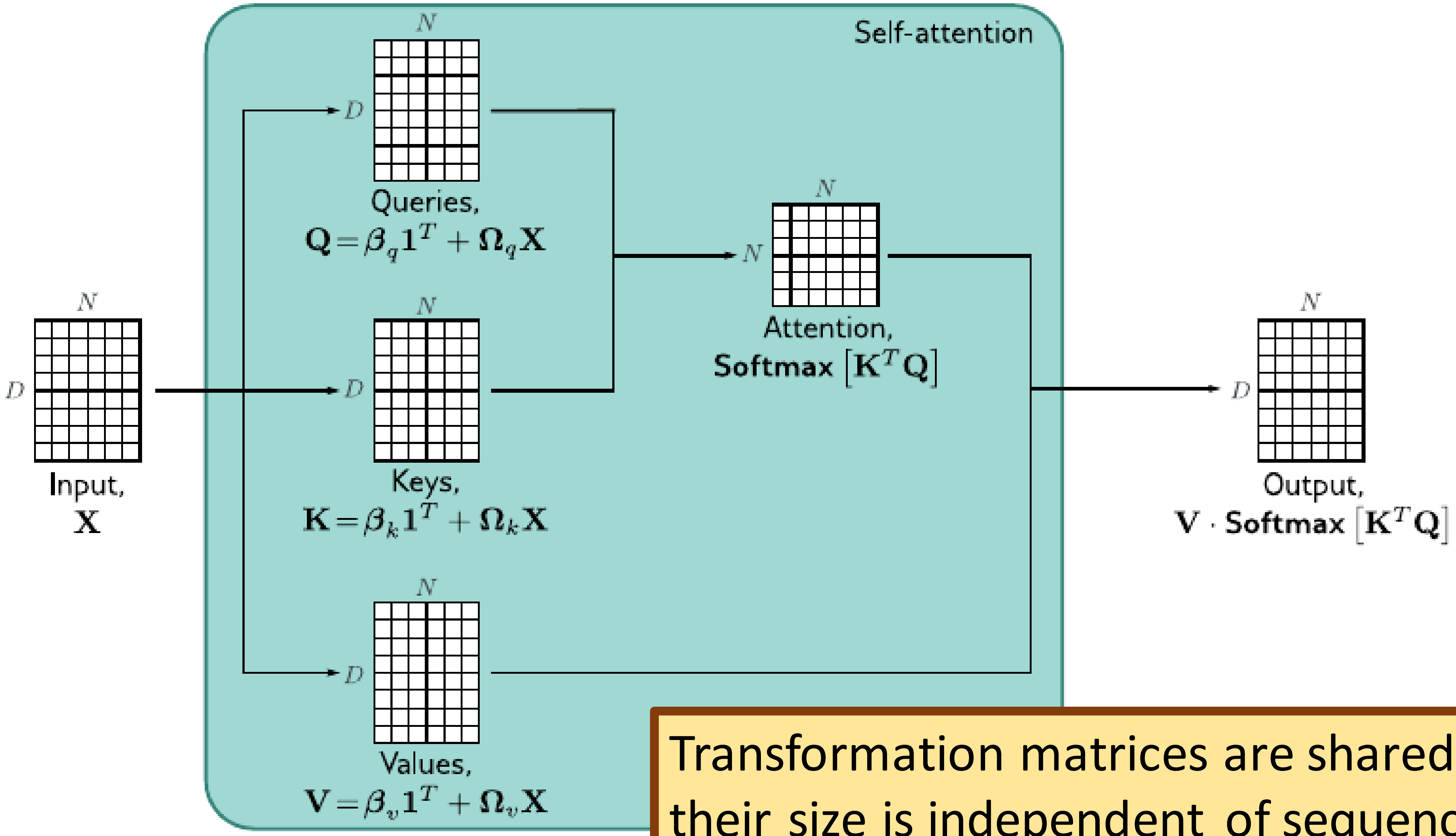
Scaled dot-product attention

$q, k \in \mathbb{R}^d$ $\quad a(q,k) = \frac{q^T k}{\sqrt{d}}$

$$\text{Attn}(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$Q \in \mathbb{R}^{n \times d}$ $\quad K \in \mathbb{R}^{m \times d}$ $\quad V \in \mathbb{R}^{m \times d}$

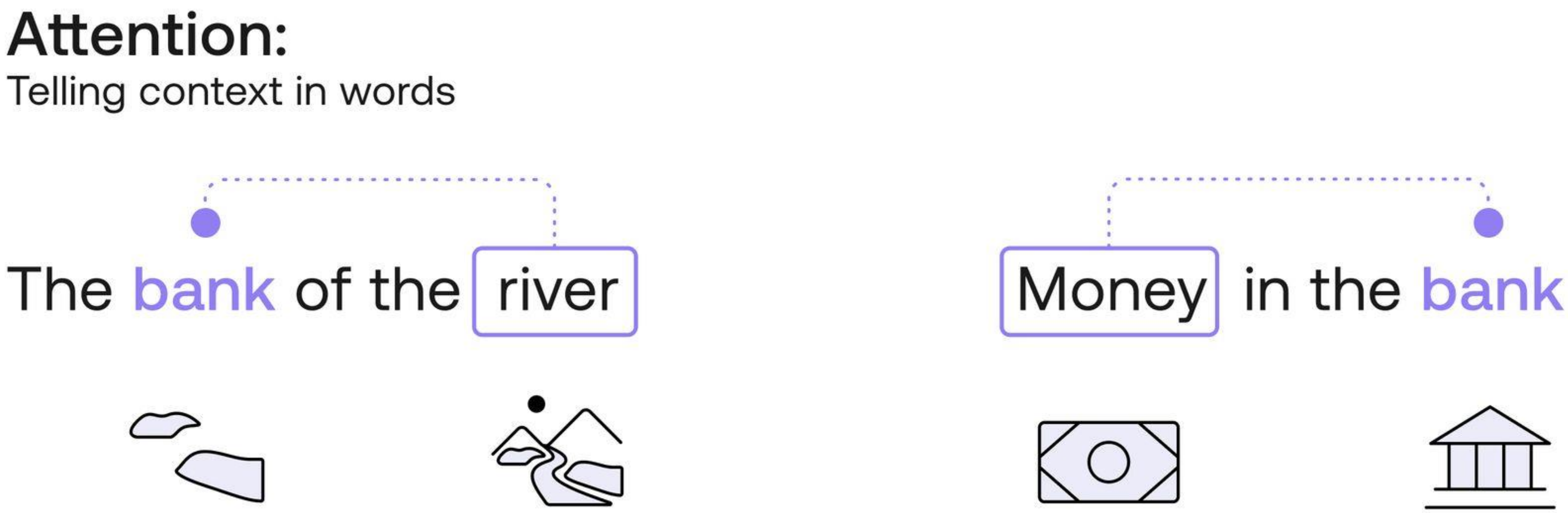
Block diagram



Transformation matrices are shared among inputs; their size is independent of sequence length!

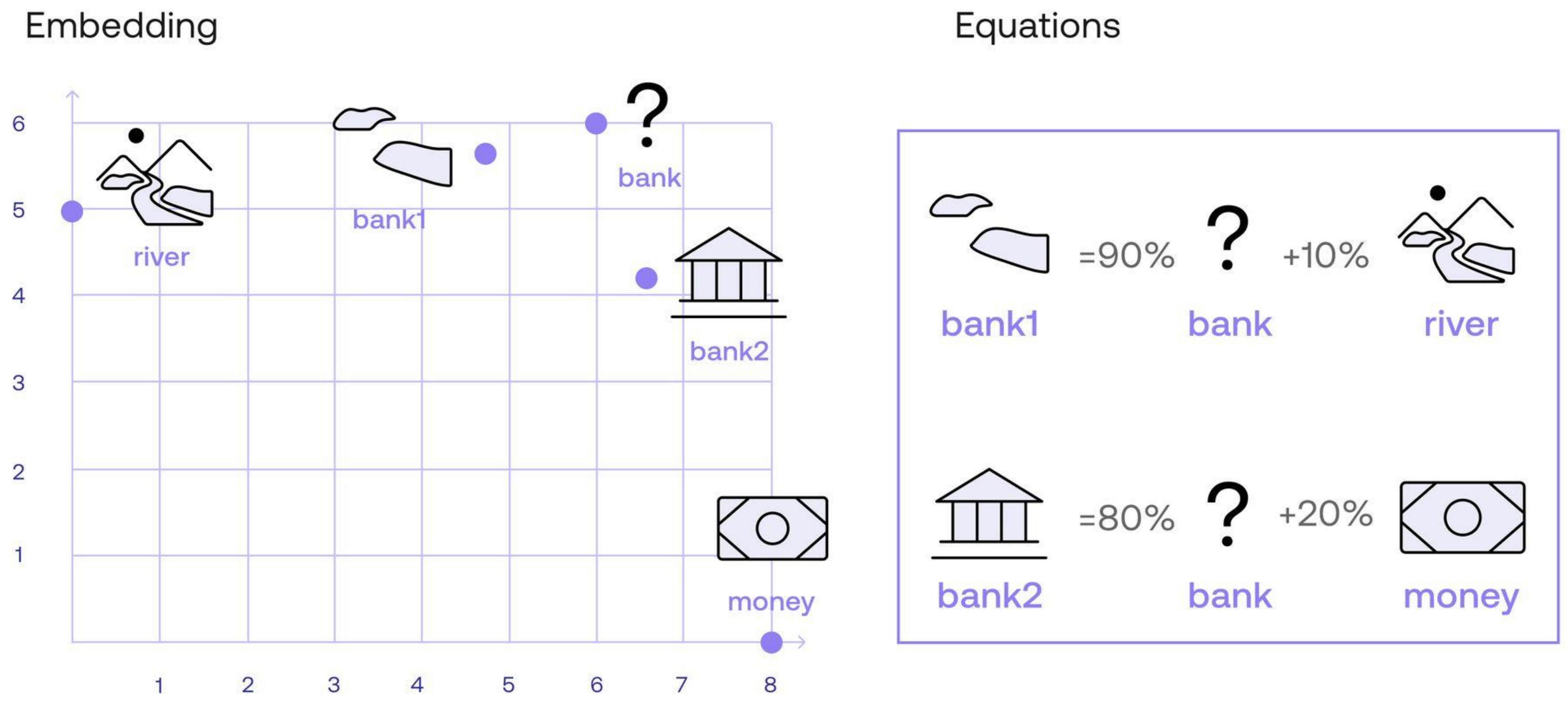
Visual explanation

Sentence 1: The bank of the river.
Sentence 2: Money in the bank.



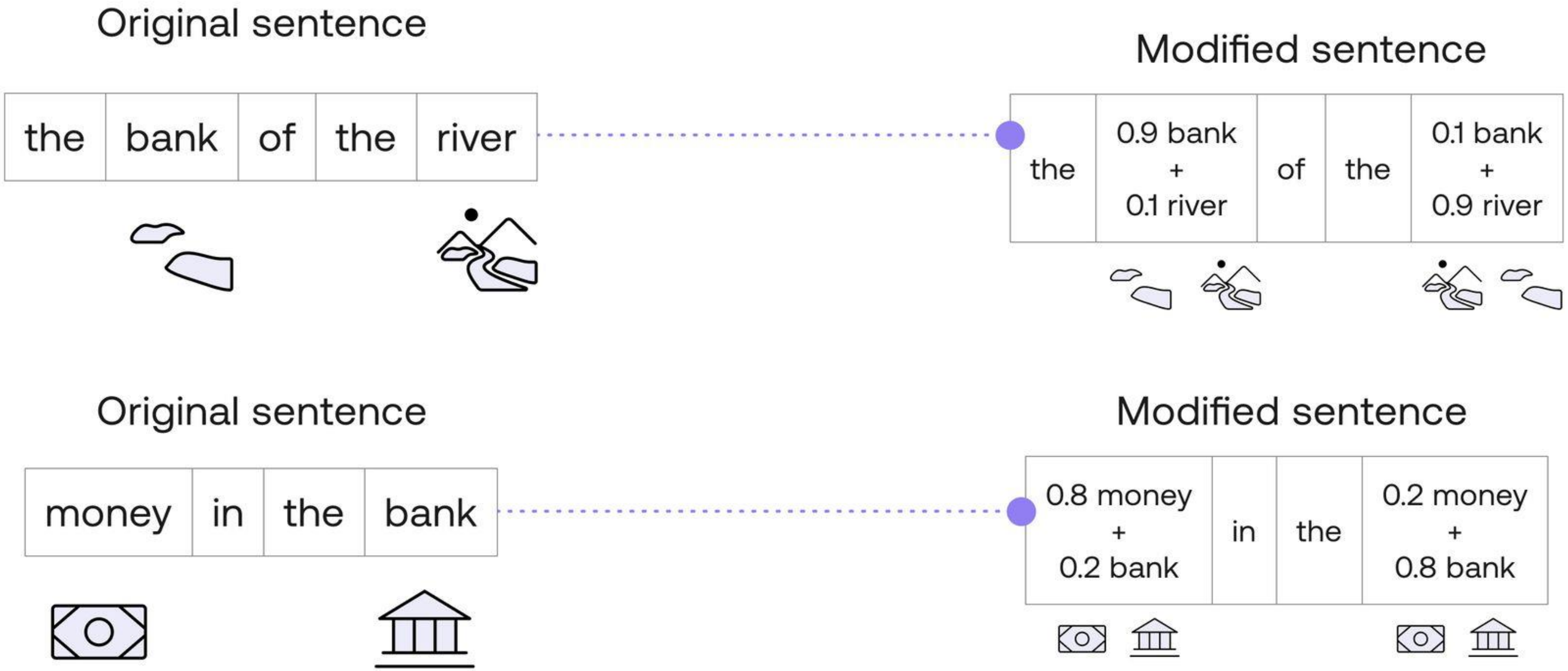
https://txt.cohere.com/what-is-attention-in-language-models/

Modified sentence 1: The **bank1** of the river.
Modified sentence 2: Money in the **bank2**.



Bank1 = 0.9*Bank + 0.1*River
Bank2 = 0.8*Bank + 0.2*Money

We find these "weights" through mechanisms such as similarity and attention



Similarity matrices

The bank of the river

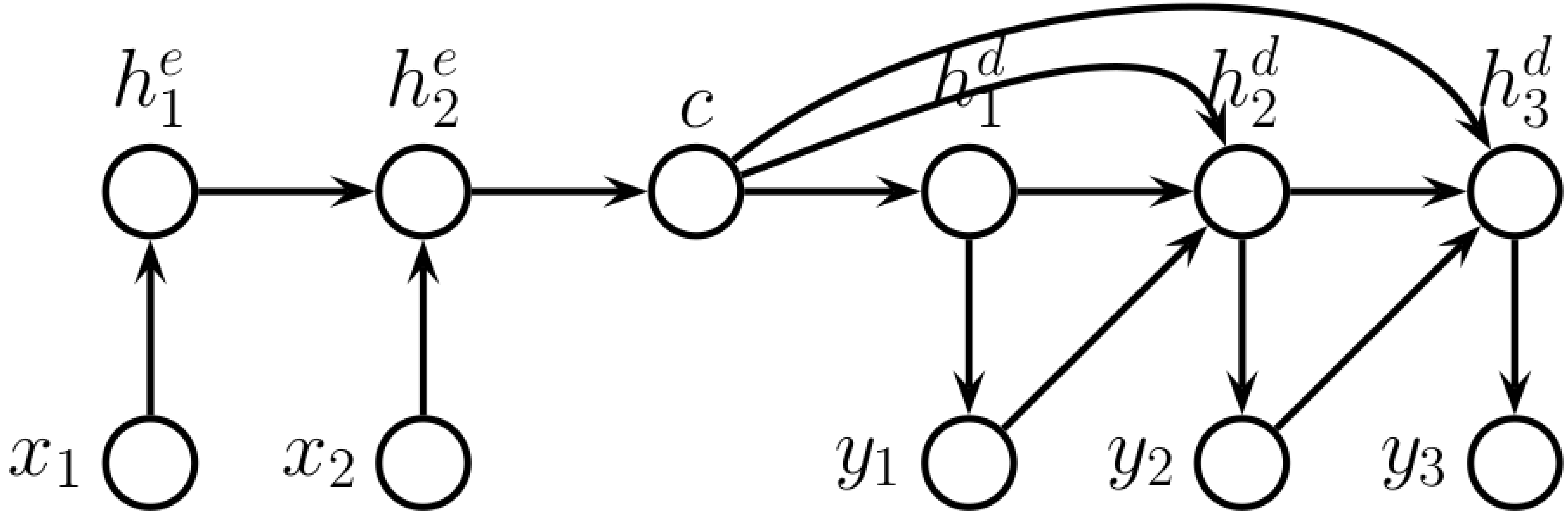
	the	bank	of	river
the	1	0	0	0
bank	0	1	0	0.11
of	0	0	1	0
river	0	0.11	0	1

Money in the bank

	money	in	the	bank
money	1	0	0	0.25
in	0	1	0	0
the	0	0	1	0
bank	0.25	0	0	1

Seq2Seq model

E → c → D



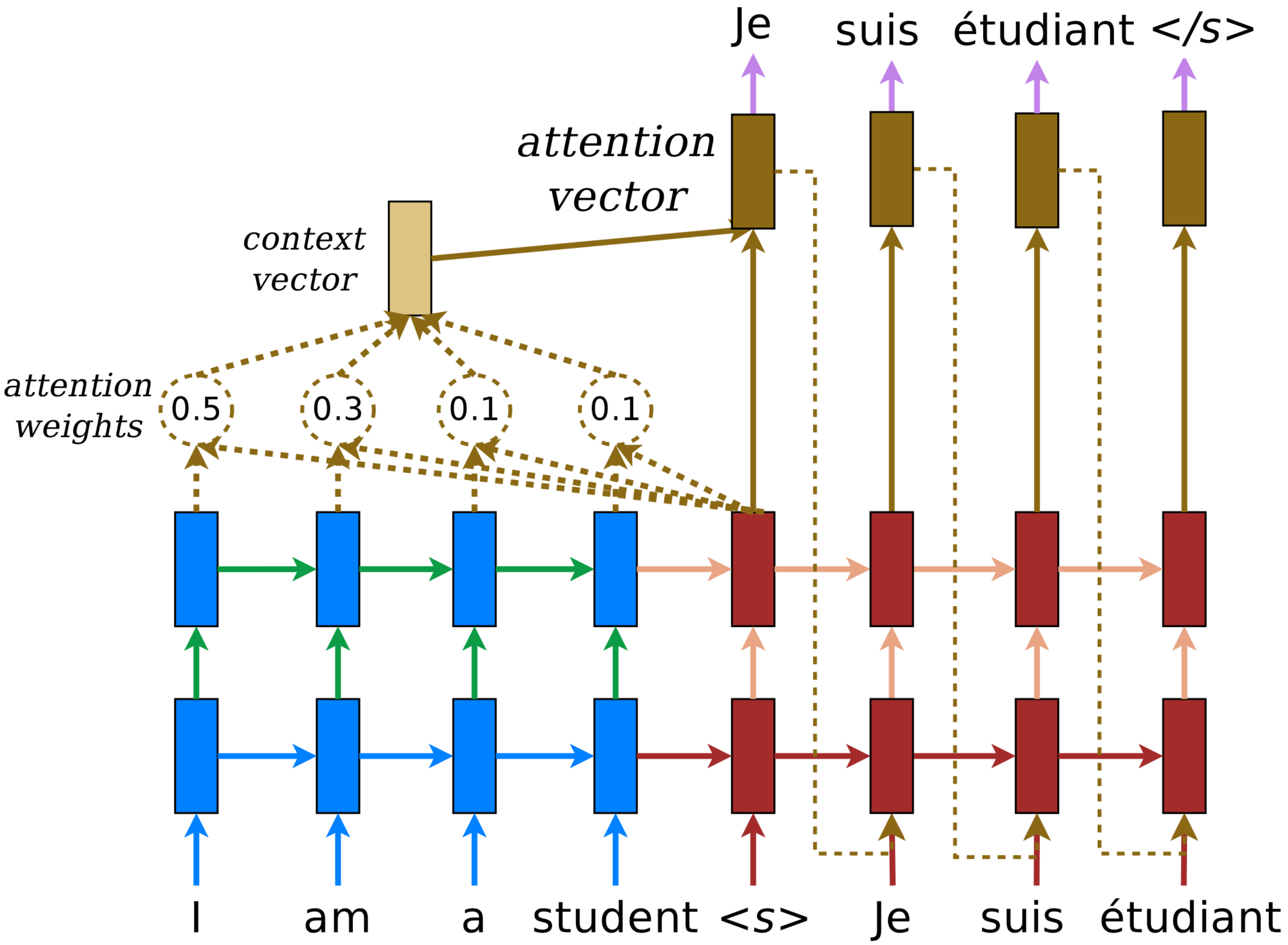
Context vector **c** encodes the *whole* input sequence **x**
Output has no access to input tokens
If we had access, which token should we "pay attention to"?

Seq2Seq with attention

$q = h^d_{t-1}$

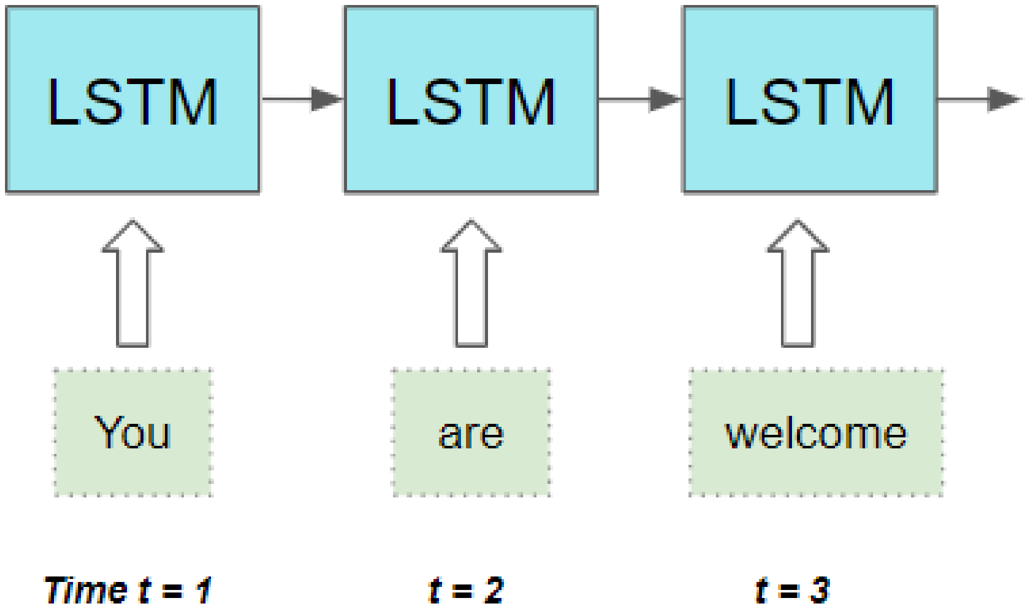
$k_i = h^e_i$

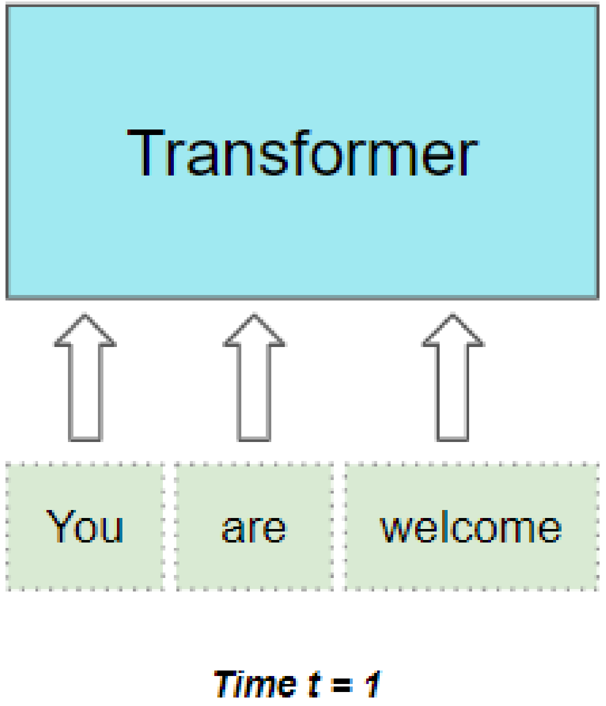
$v_i = h^e_i$



Are transformers better than RNNs?

- Transformers deal better with long-range dependencies
- Transformers process the input at same time (not step-by-step)

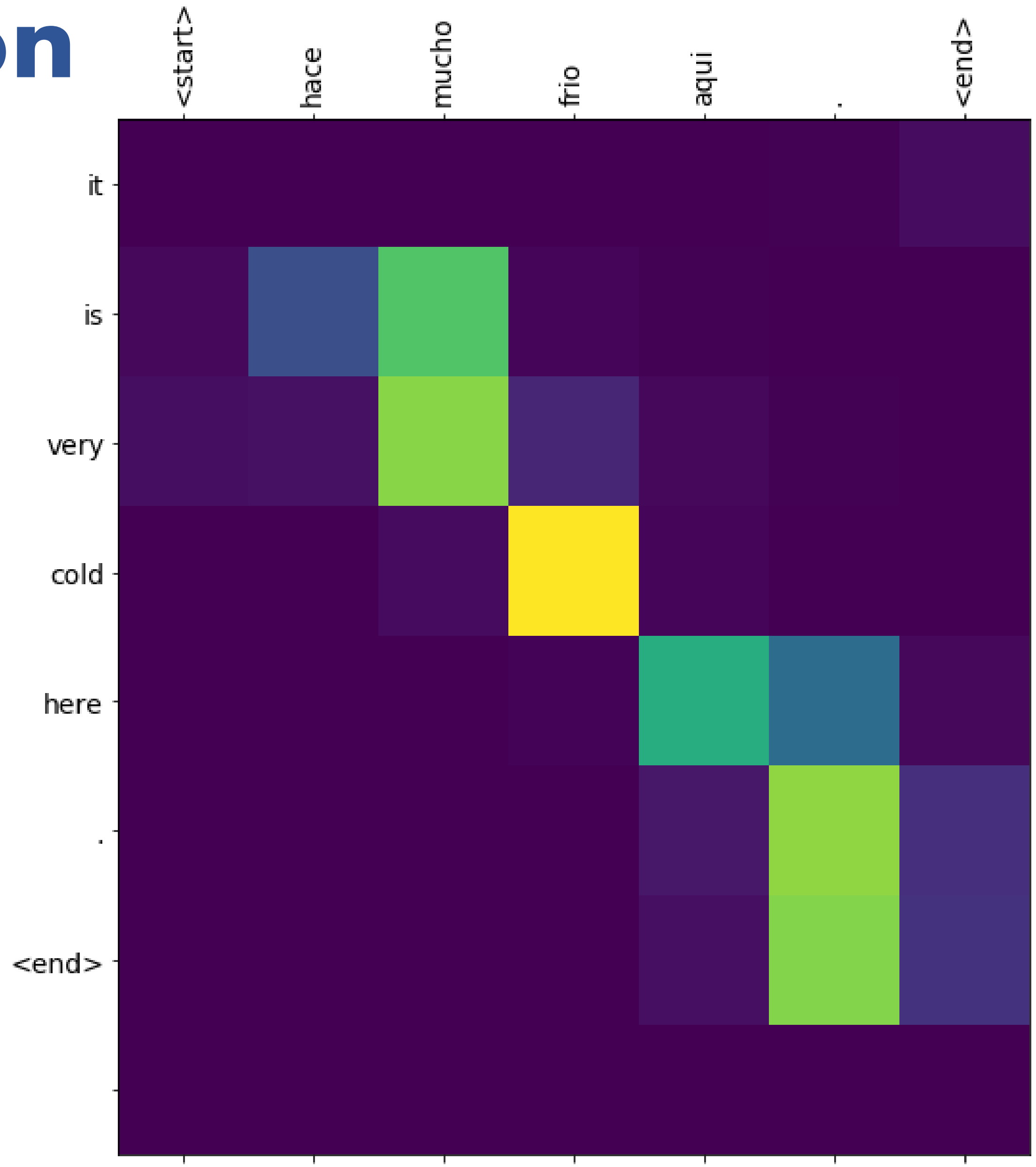




Example: NLP translation

x = Hace mucho frío aquí.
y = It is very cold here.

$\alpha_i \alpha_i$



Transformer

Transformers

Transformer = Seq2Seq model with attention at both E and D

Do we really need both *encoder* and decoder?

"Any architecture designed to process a connected set of units—such as the tokens in a sequence or the pixels in an image—where the only interaction between units is through self-attention."

Don't we need also cross-attention?

Self attention (at encoder)

Input attends to itself!

$$y_i = \mathrm{Attn}(x_i, \{(x_j, x_j)\})$$

$$q_i = x_i$$

$$k_j = x_j$$

$$v_j = x_j$$

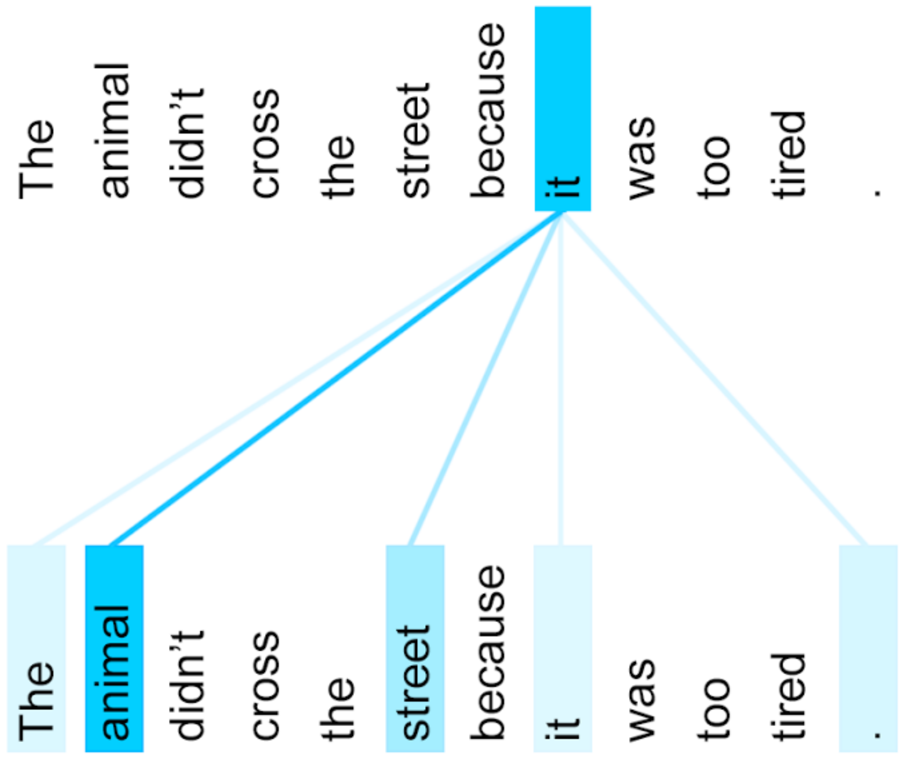
At decoder: masked self-attention

$$y_i = \mathrm{Attn}\left(y_{i-1}, \{(y_1, y_1), \dots (y_{i-1}, y_{i-1})\}\right)$$

q_i $\qquad$ k_i v_i

Example: Translation English-French

x = The animal didn't cross the street because it was too tired



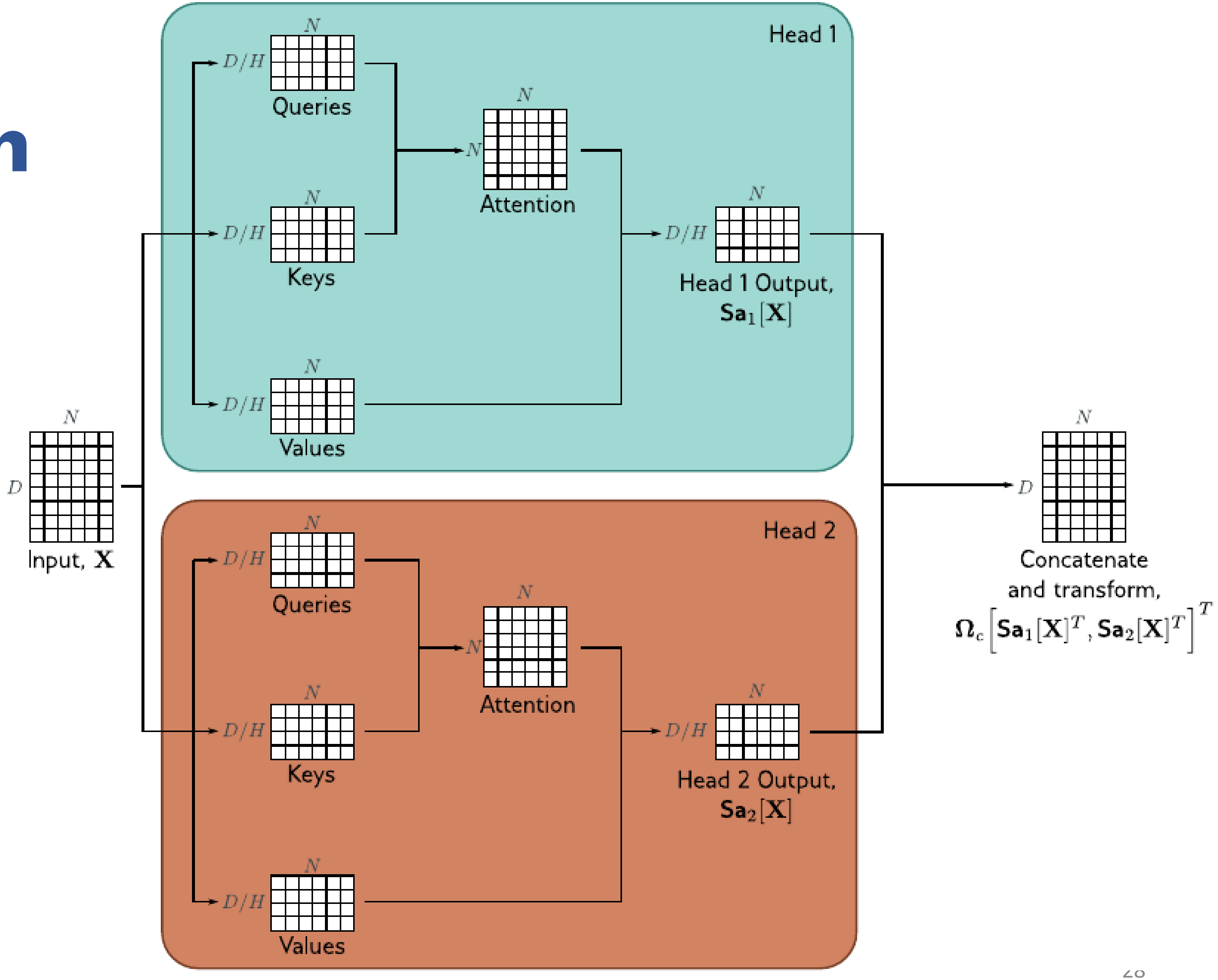
Multi-head attention

Enrich the representation: having several notions of similarity

$$h_i = \mathrm{Attn}\left(W_i^{(q)} q, \left\{W_i^{(k)} k_j, W_i^{(v)} v_j\right\}\right)$$

$$h = \mathrm{MHA}\left(q, \left\{(k_j, v_j)\right\}\right) = W_o \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_N \end{bmatrix}$$

Multihead – block diagram



Positional encoding

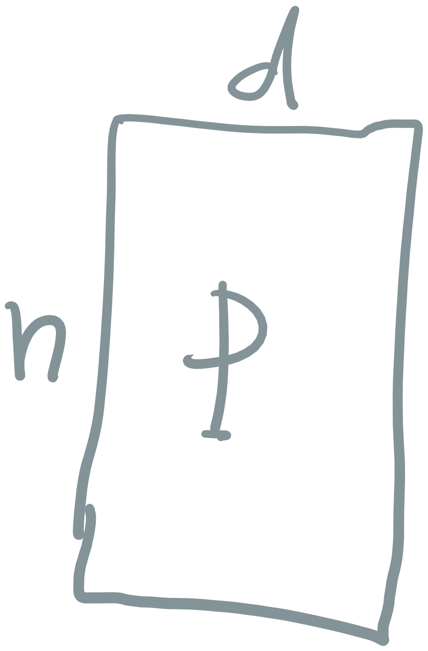
Sequences have order, but self-attention is permutation invariant!

$x_1, x_2, \dots, x_n \qquad x_i \in \mathbb{R}^d$

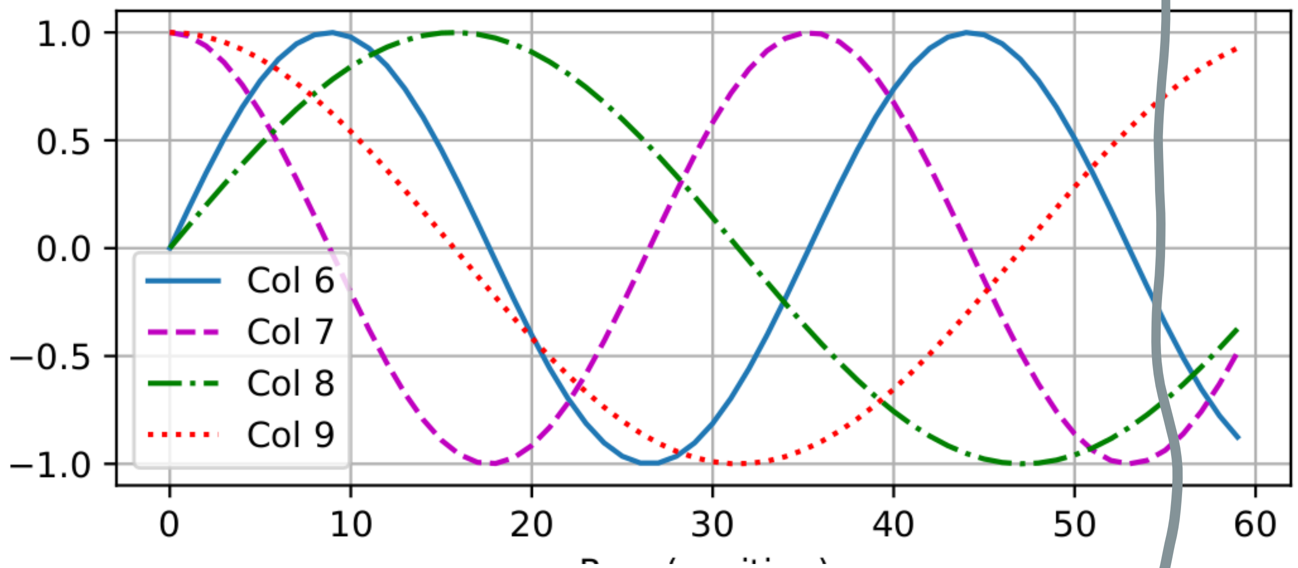
$$P_{i,j} = \sin\left(\frac{i}{C^{2j/d}}\right) \quad \text{if } j \text{ is even}$$

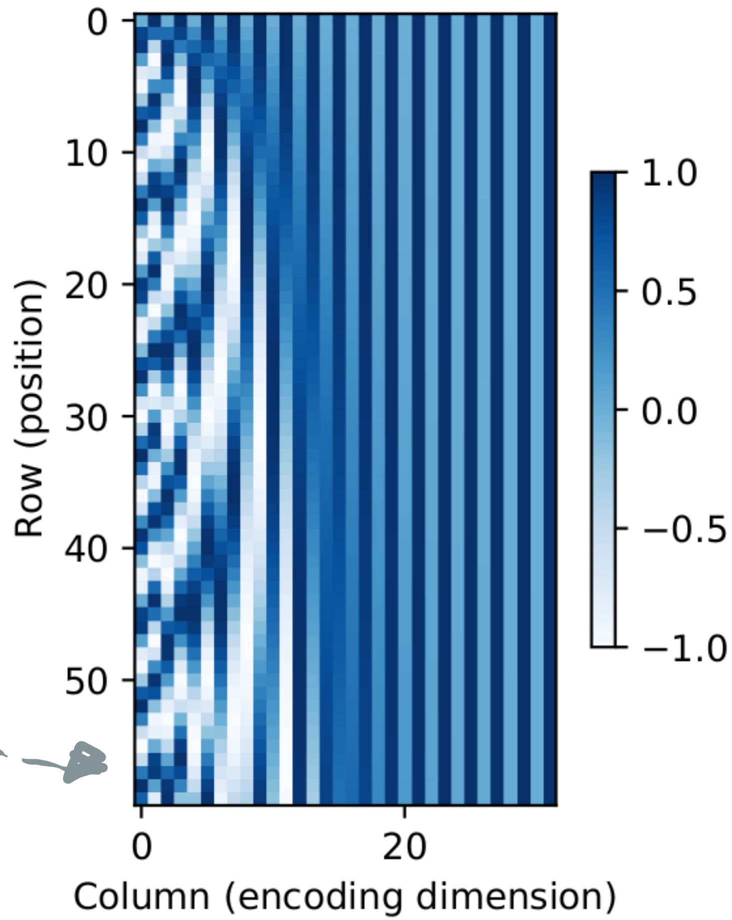
$$P_{i,j} = \cos\left(\frac{i}{C^{2j/d}}\right) \quad \text{if } j \text{ is odd}$$

$C =$ max seq length



$n = 60, \ d = 32$



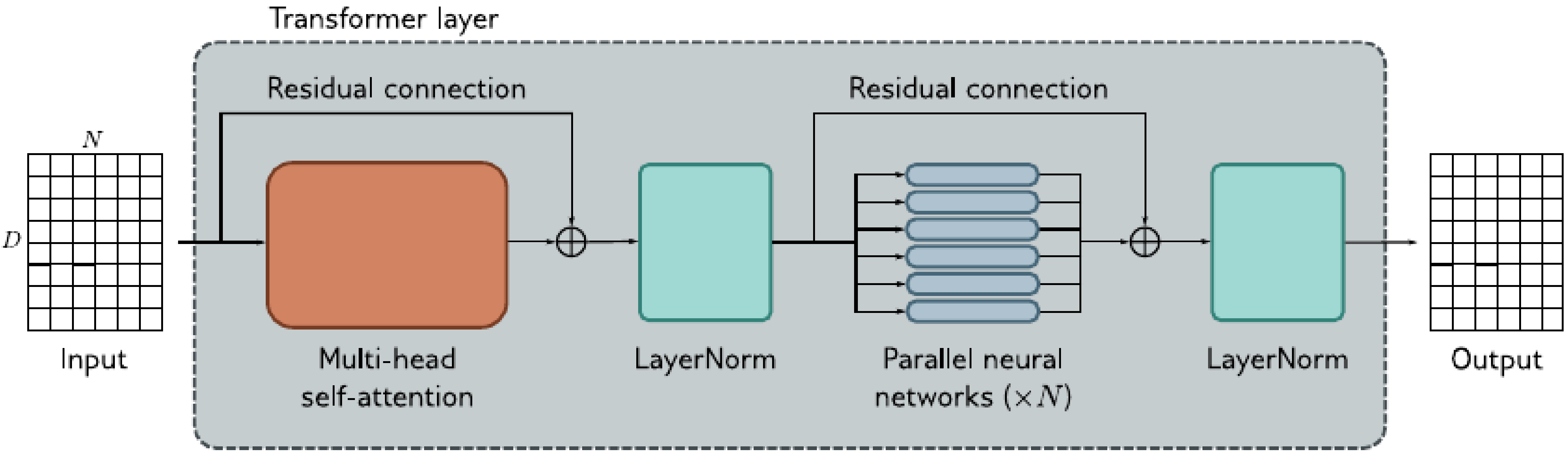


X = original embeddings

P = positional embeddings

new embeddings: $X + P$

Transformer: MH SA + something else



Encoder [+ decoder] transformers

Self-attention:

- input tokens attend to one another

Masked self-attention

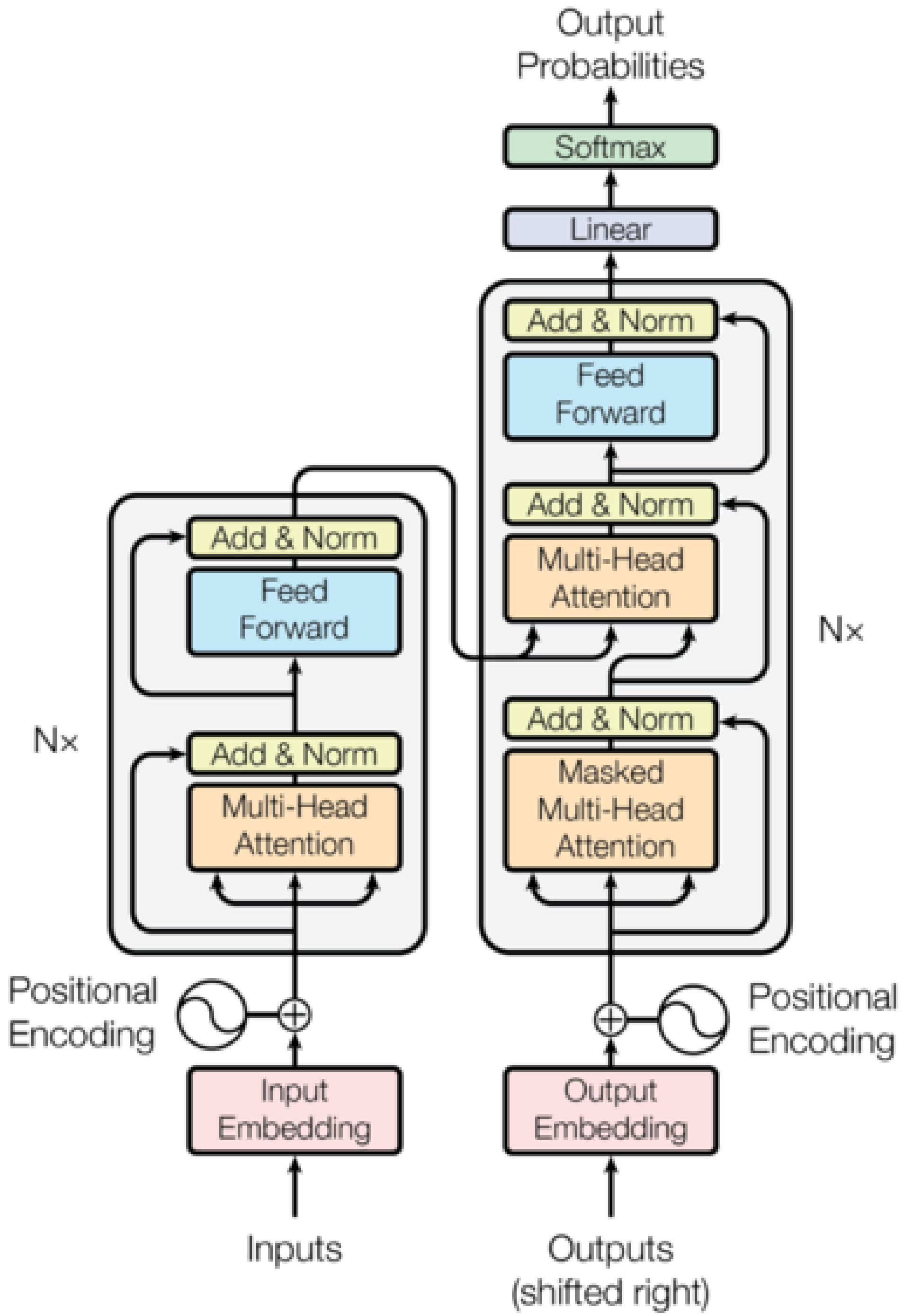
- output tokens attend to *previous* output tokens

Cross-attention

- output tokens attend to input tokens

Let's identify the (q,k,v) at the three different attention blocks

k_i v_i q_i



Pseudocode

Encoder = series of *M* encoder blocks

```
def EncoderBlock(X):
    Z = LayerNorm(MHA(Q=X, K=X, V=X) + X) # note the residual connection
    E = LayerNorm(FeedForward(Z) + Z)
    return E


def Encoder(X, M): # M = number of layers
    E = POS(Embed(X))
    for _ in range(M):
        E = EncoderBlock(E)
    return E
```

Decoder has access to:

- encoder (via another MHA)
- Previously generated output

```
def DecoderBlock(X, E):
    Z  = LayerNorm(MHA(Q=X, K=X, V=X) + X)
    Z' = LayerNorm(MHA(Q=Z, K=E, V=E) + Z)
    D  = LayerNorm(FeedForward(Z') + Z')
    return D

def Decoder(X, E, N):
    D = POS(Embed(X))
    for _ in range(N):
        E = DecoderBlock(D, E)
    return D
```

Attention is not explanation

Or is it?

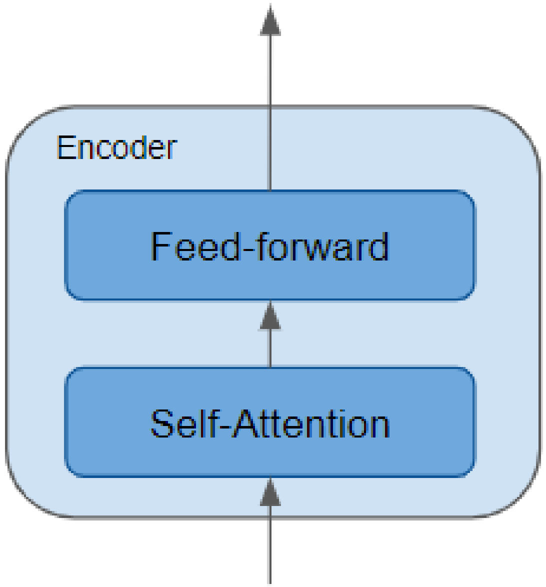
https://towardsdatascience.com/is-attention-explanation-b609a7b0925c

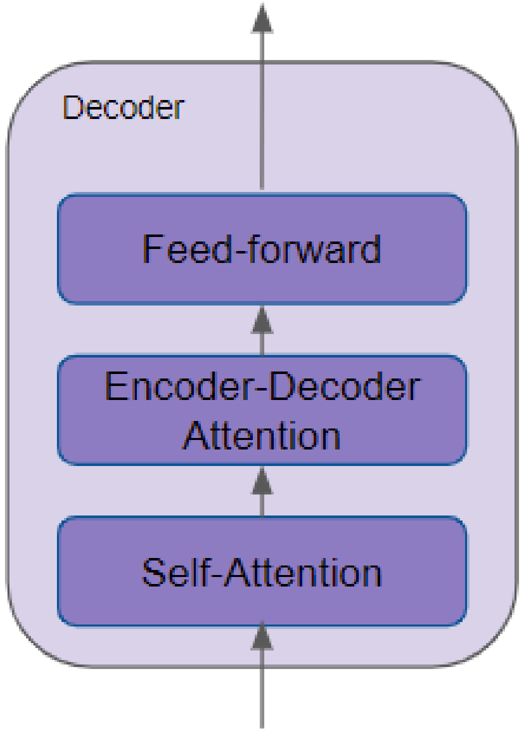
Many variations proposed

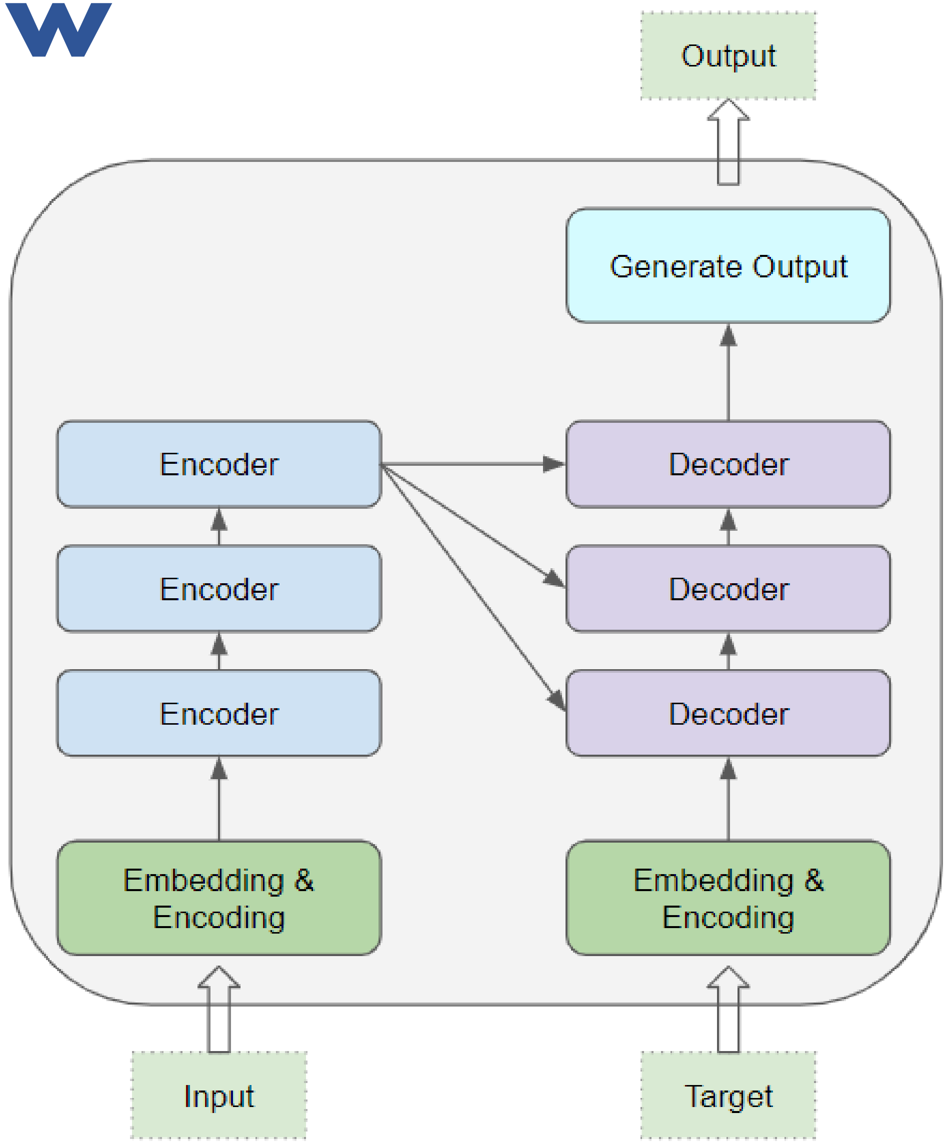
Conformer (conv layers inside Transformer)
Reformer
Linformer
Performer
...

A top-down explanation

Encoder, Decoder and how they relate

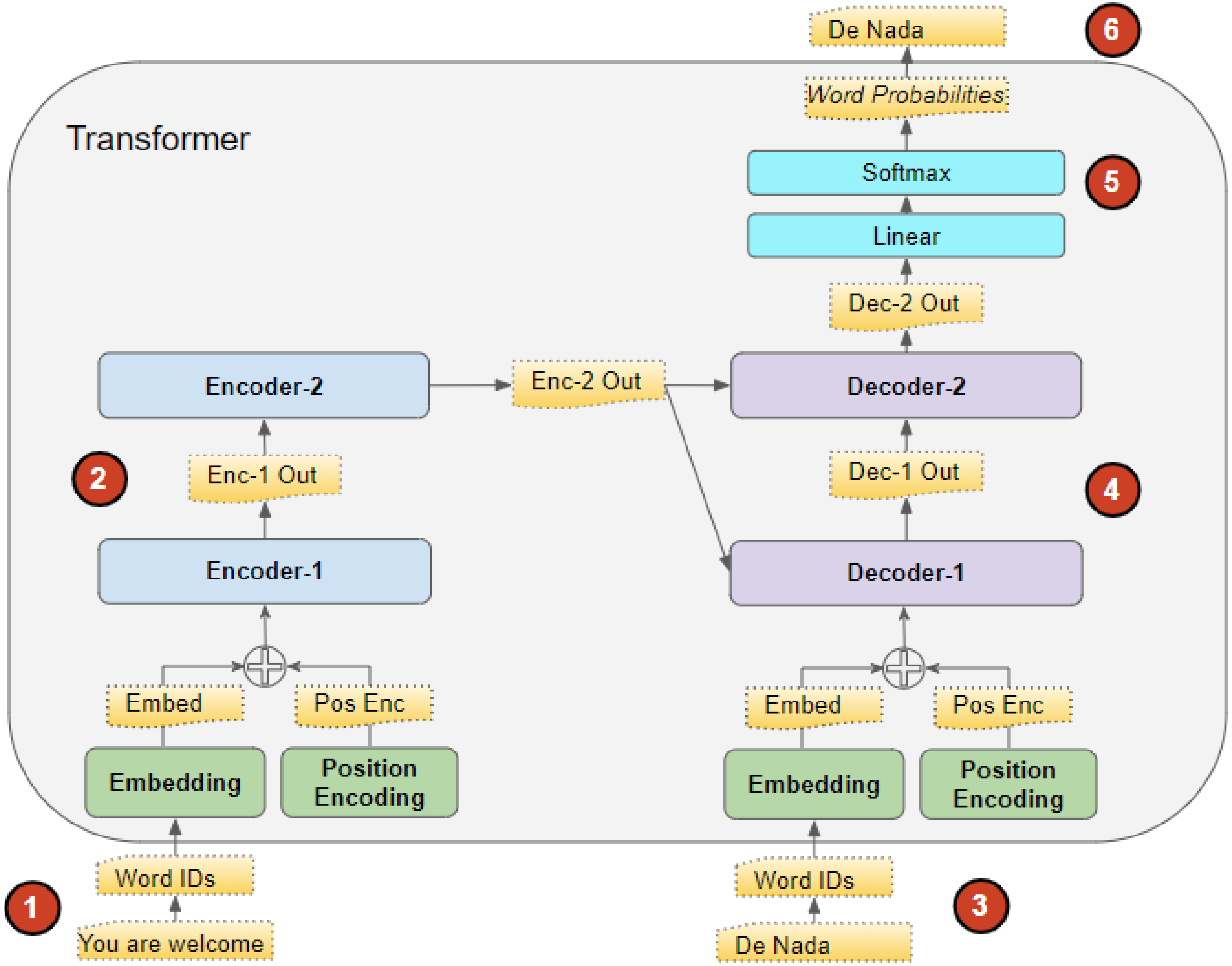






https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

At training...



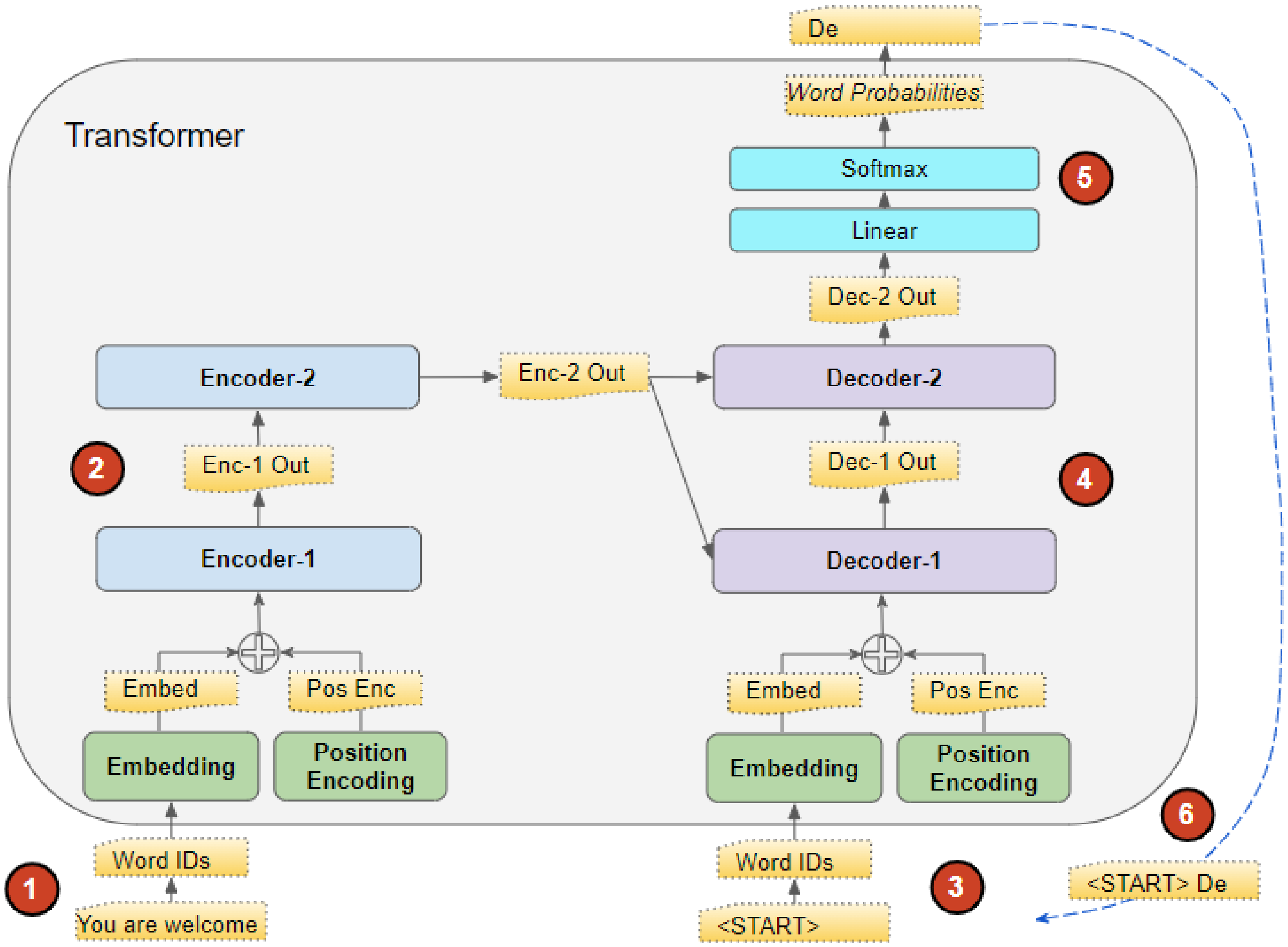
Loss:
compares output sequence with target sequence (known because it's in the training data)

Teacher Forcing

At inference...

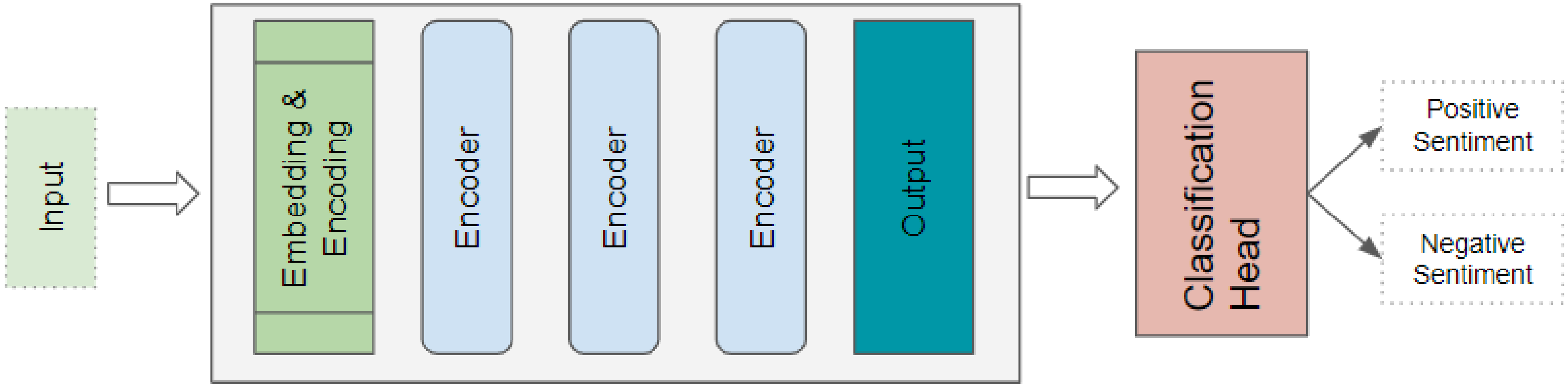
Last word of output sequence added at the end of
the decoder input

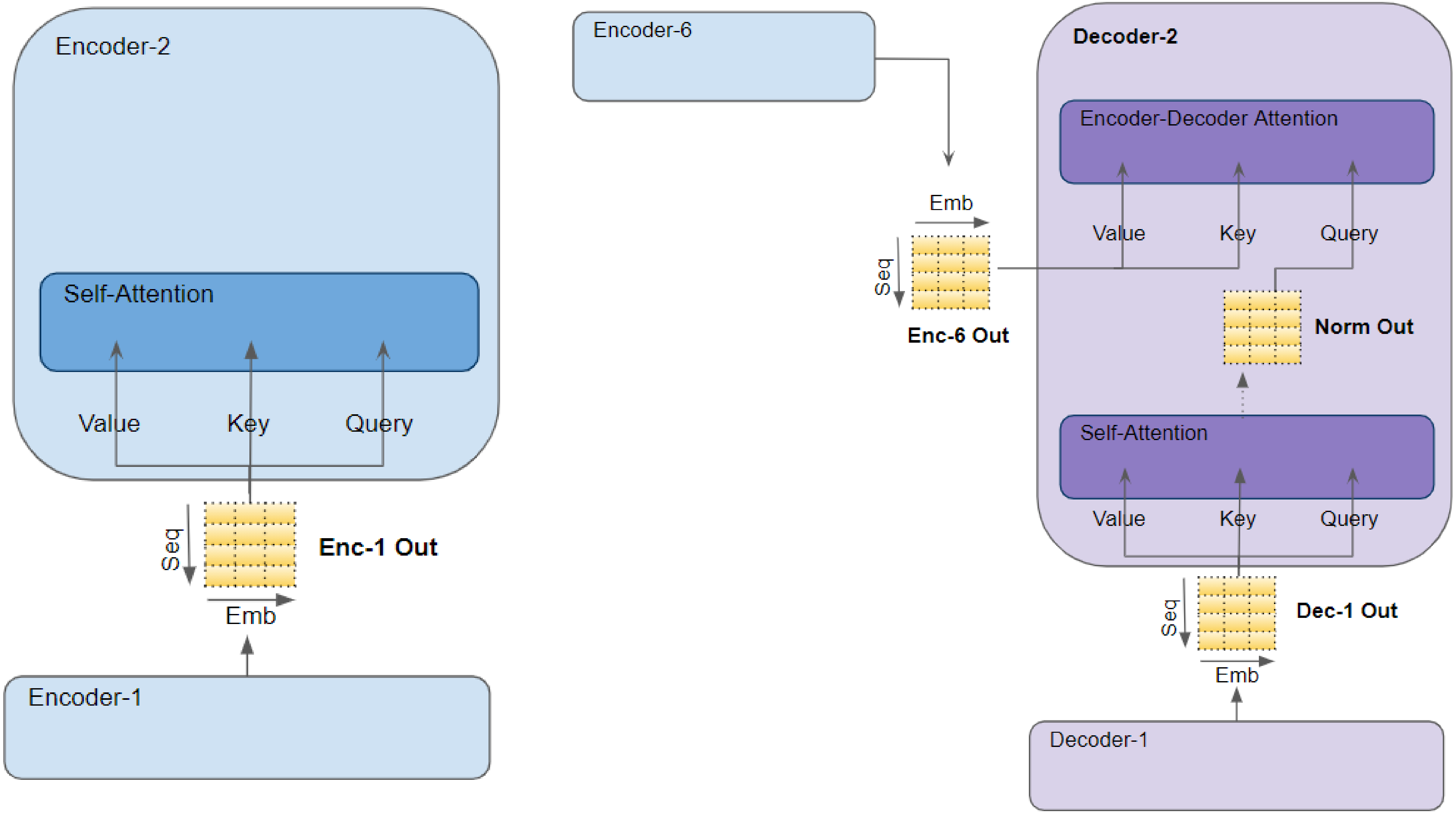
Repeat 3, 4, 5, 6 until "end of sequence"



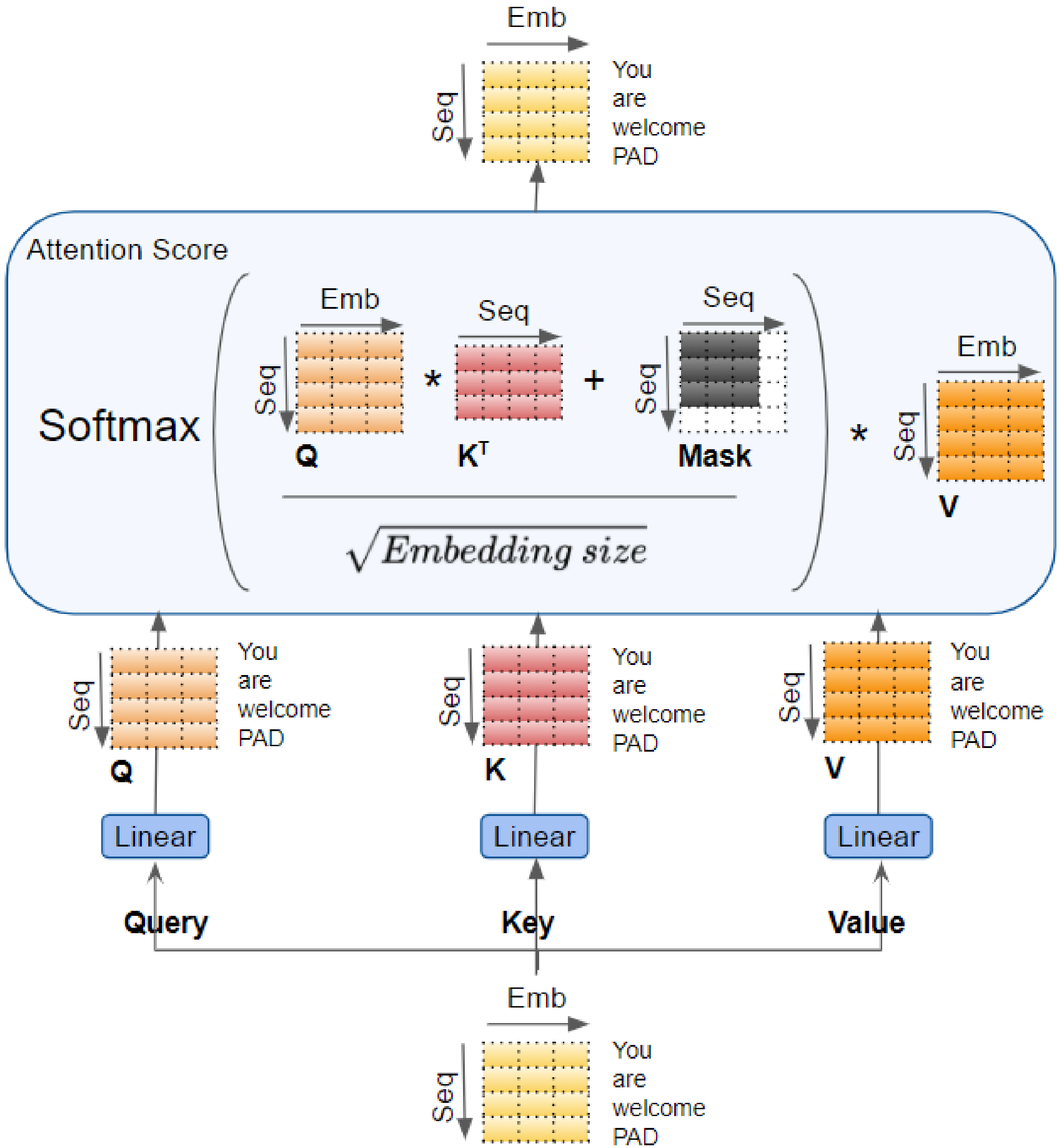
We don't alwyas have a decoder…

Encoder as a building block for different applications

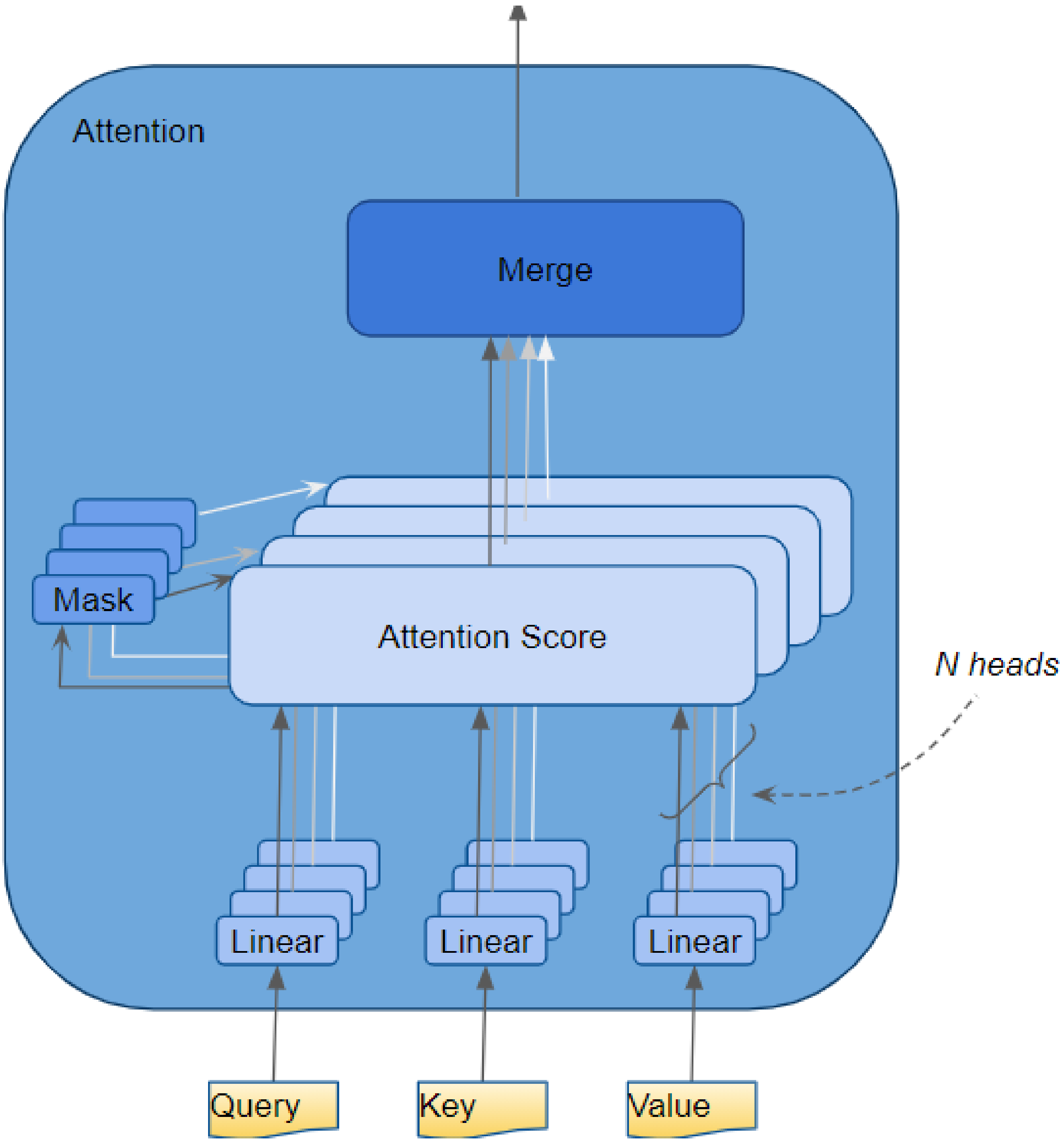


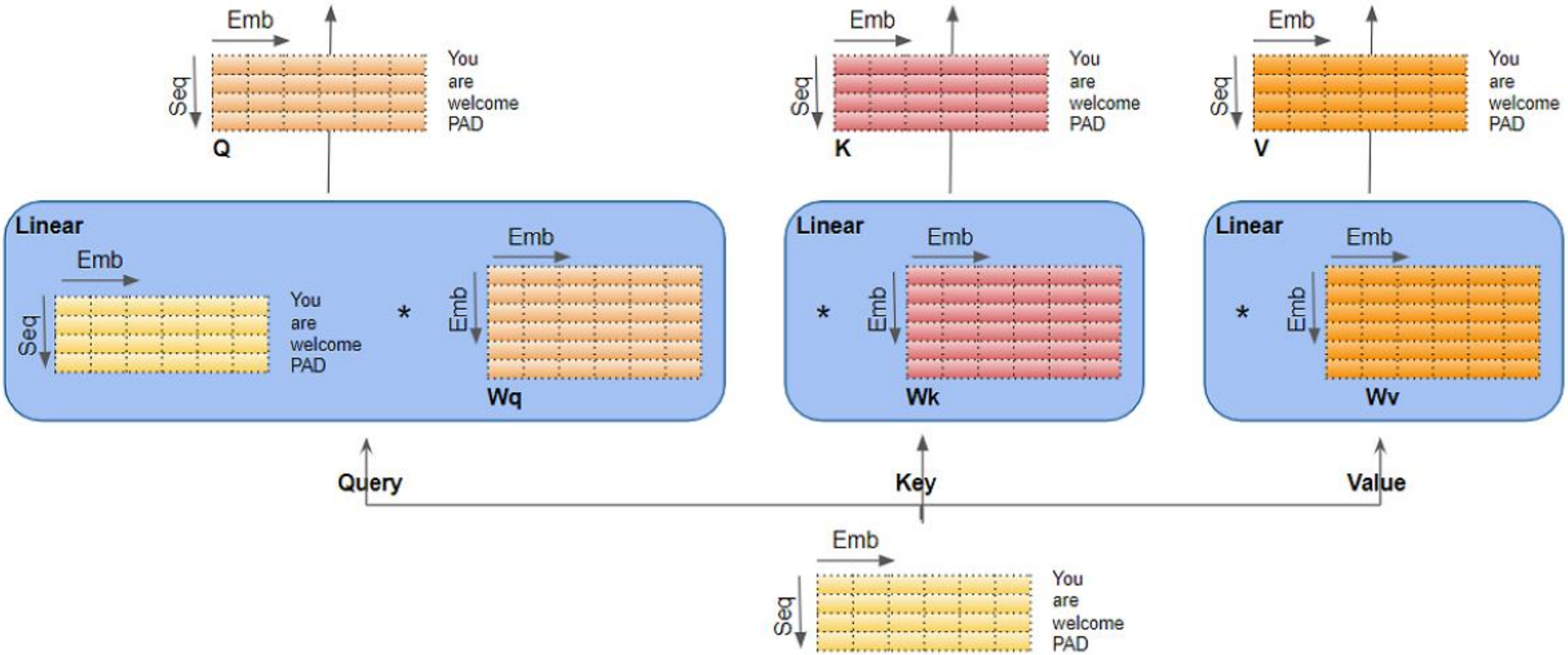


https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34



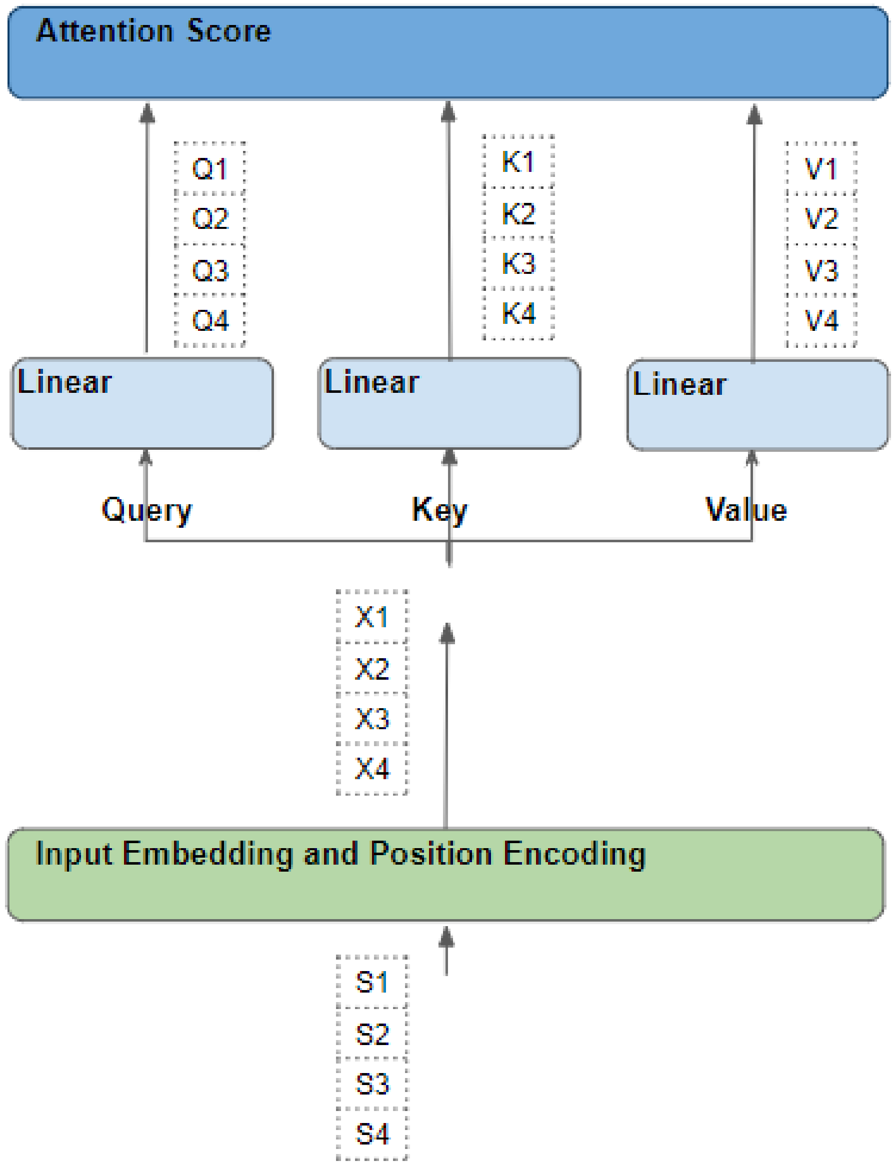
Emb
Seq
You
are
welcome
PAD
Attention Score
Softmax
Emb
Seq
Q
*
Seq
K^T
+
Seq
Seq
Mask
$\sqrt{Embedding\ size}$
*
Emb
Seq
V
Seq
Q
You
are
welcome
PAD
Seq
K
You
are
welcome
PAD
Seq
V
You
are
welcome
PAD
Linear
Linear
Linear
Query
Key
Value
Emb
Seq
You
are
welcome
PAD


Attention
Merge
Mask
Attention Score
N heads
Linear
Linear
Linear
Query
Key
Value



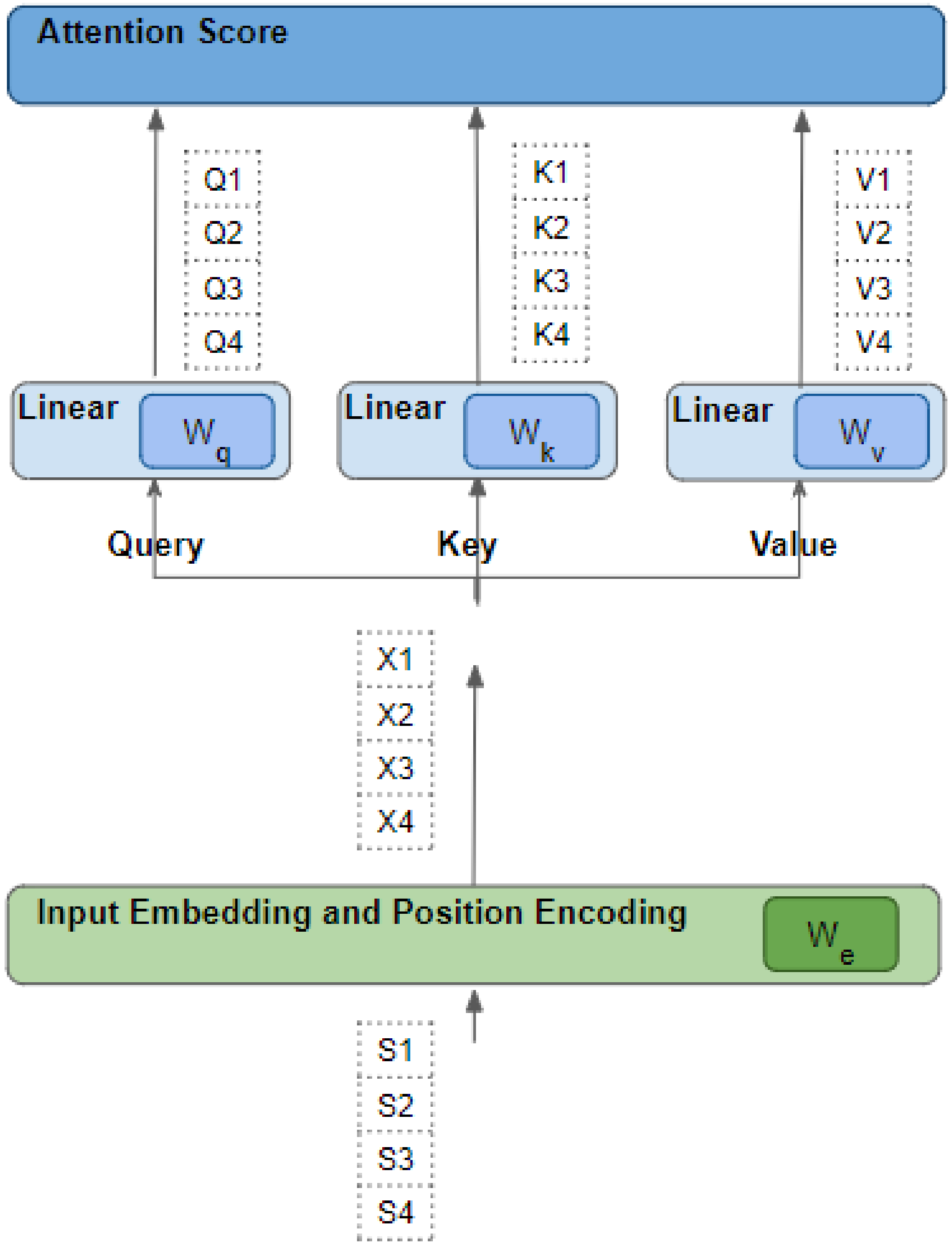
https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853

Words (sequence items) go through a series of transformations



https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3

Transformations are learnable!



Role of Q, K, V

The **Query** word: word **for which** we are calculating Attention

The **Key** and **Value** word: word ***to which*** we are paying attention (how relevant is that word to the Query word)

And how is "relevance" learned?

The more "aligned" (similar embeddings) two words are, the higher their attention score

So, the transformer learns to produce low or high scores

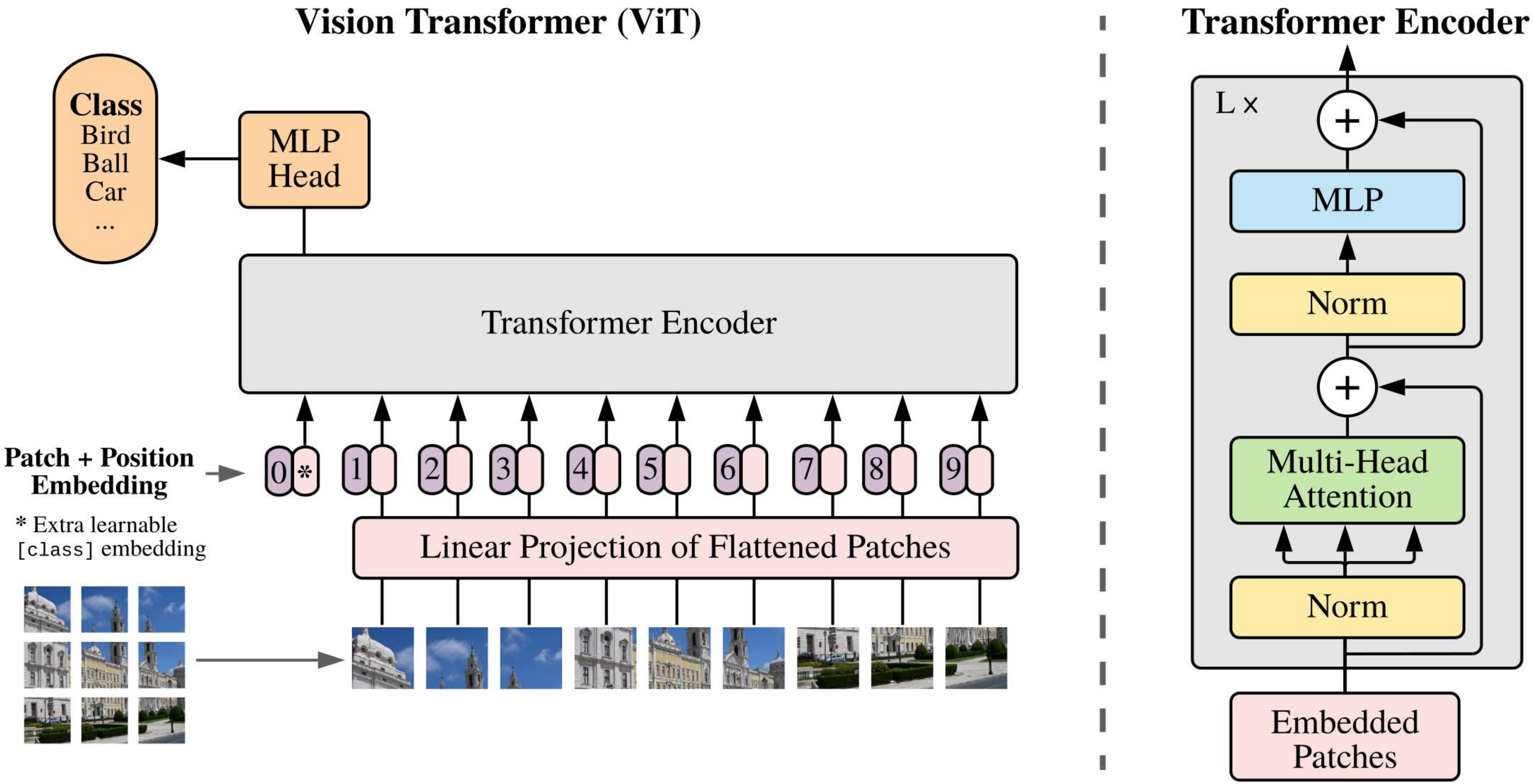
And how are such scores found?
From the training instances

Vision Transformers

CNNs vs Transformers for images

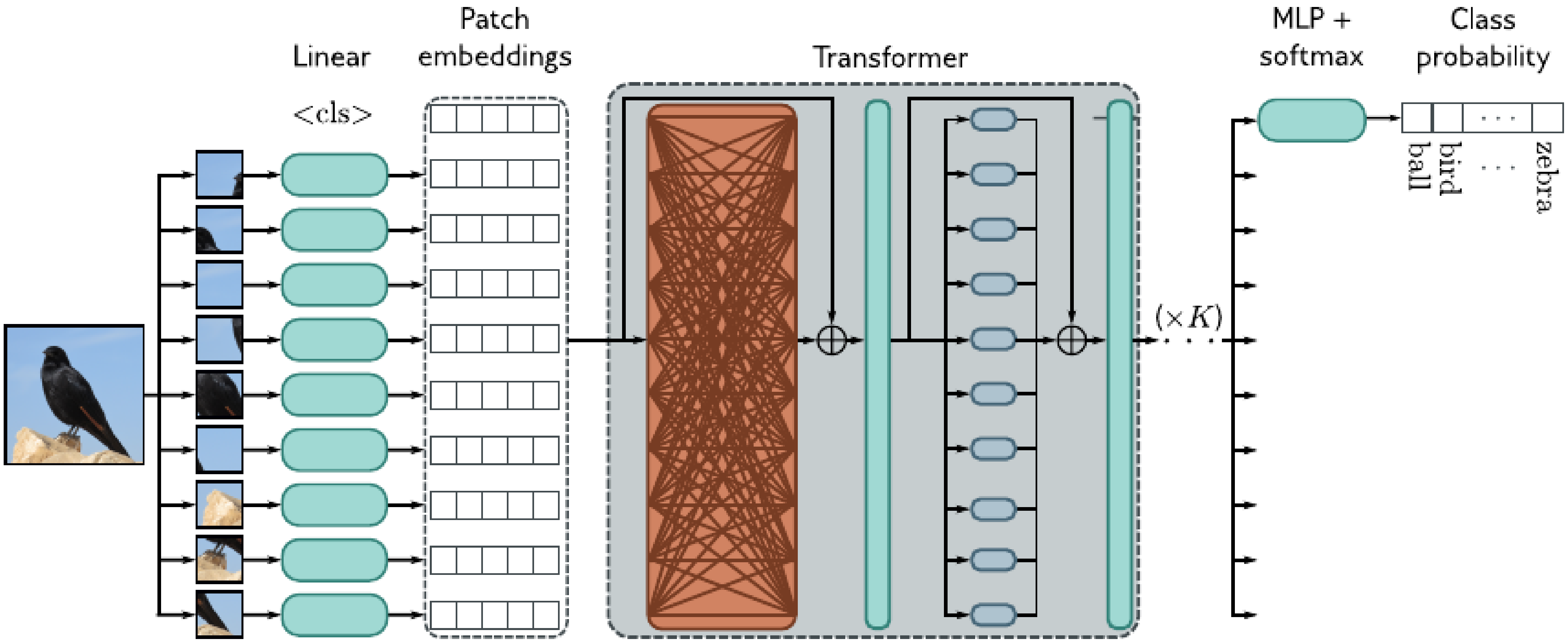
CNNs have built-in **inductive bias**:
- **locality** (small kernel)
- **equivariance** (weight sharing)
- **location invariance** (pooling)

Transformers don't seem a good fit for 2D data...

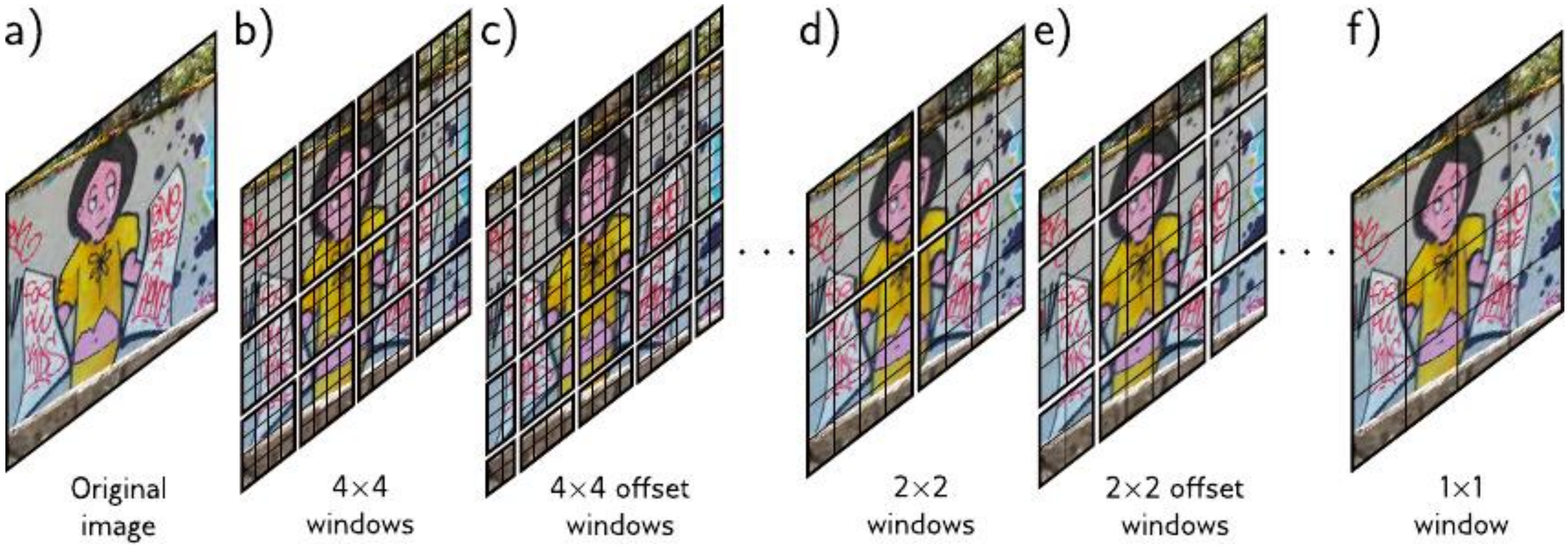


An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (ICLR 2021)

ViT: transformer encoder



SWin (shifted-window): multi-scale transformer



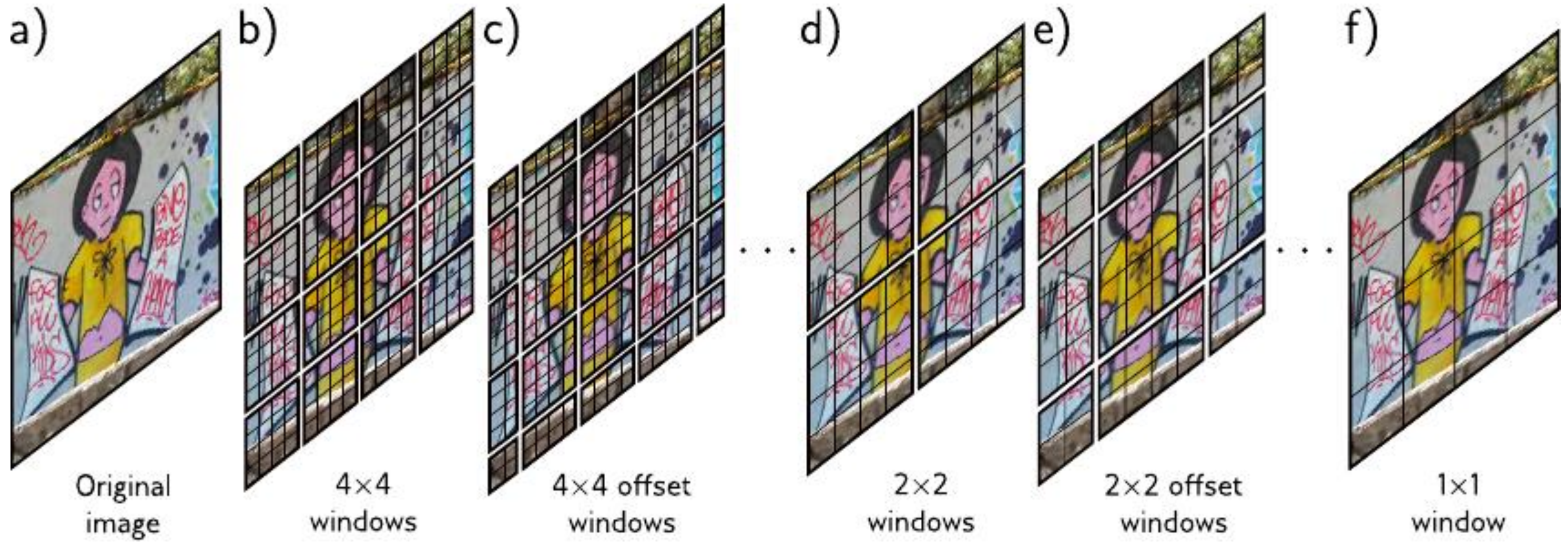
Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows (ICCV 2021)

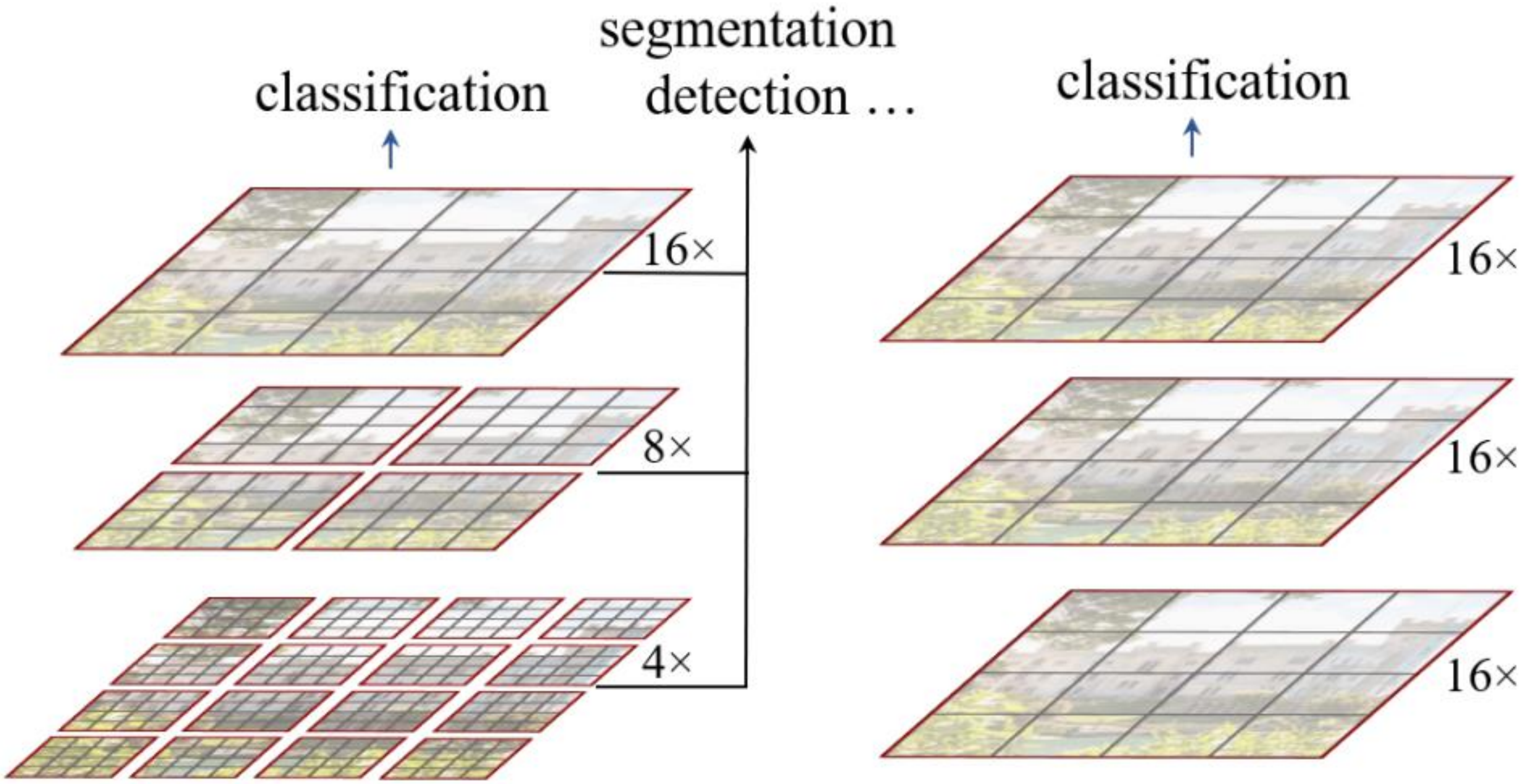
Efficiency:

- Limiting self-attention to non-overlapping **local** windows

Hierarchical architecture:

- Flexibility to model at various **scales**
- **Linear** computational complexity with respect to image size



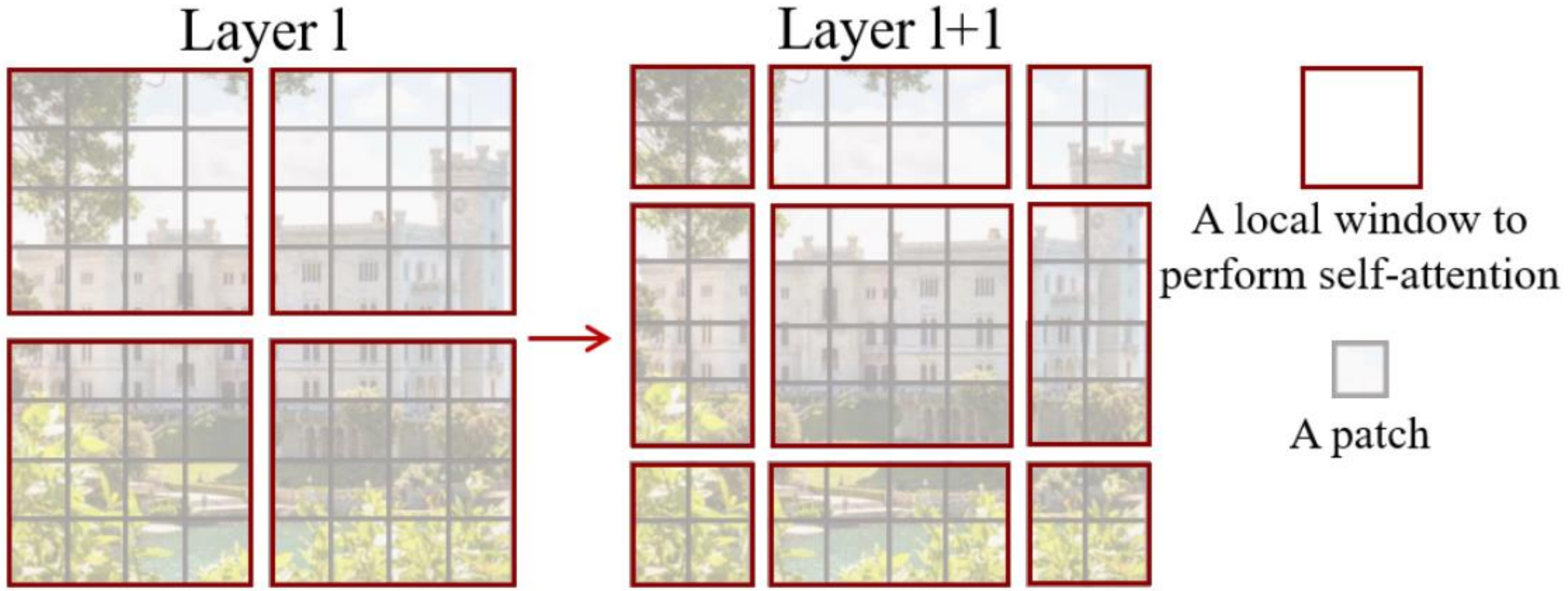


(a) Swin Transformer (ours)

(b) ViT

Local attention
Hierarchical
O(N)

Global attention
Single and coarse level
O(N^2)



Window partitioning is shifted in alternate layers
--> this changes the interacting patches (and connect windows)

Ablation study (SWin)

	ImageNet		COCO		ADE20k
	top-1	top-5	AP^{box}	AP^{mask}	mIoU
w/o shifting	80.2	95.1	47.7	41.5	43.3
shifted windows	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**
no pos.	80.1	94.9	49.2	42.6	43.8
abs. pos.	80.5	95.2	49.0	42.4	43.2
abs.+rel. pos.	81.3	95.6	50.2	43.4	44.0
rel. pos. w/o app.	79.3	94.7	48.2	41.9	44.1
rel. pos.	**81.3**	**95.6**	**50.5**	**43.7**	**46.1**

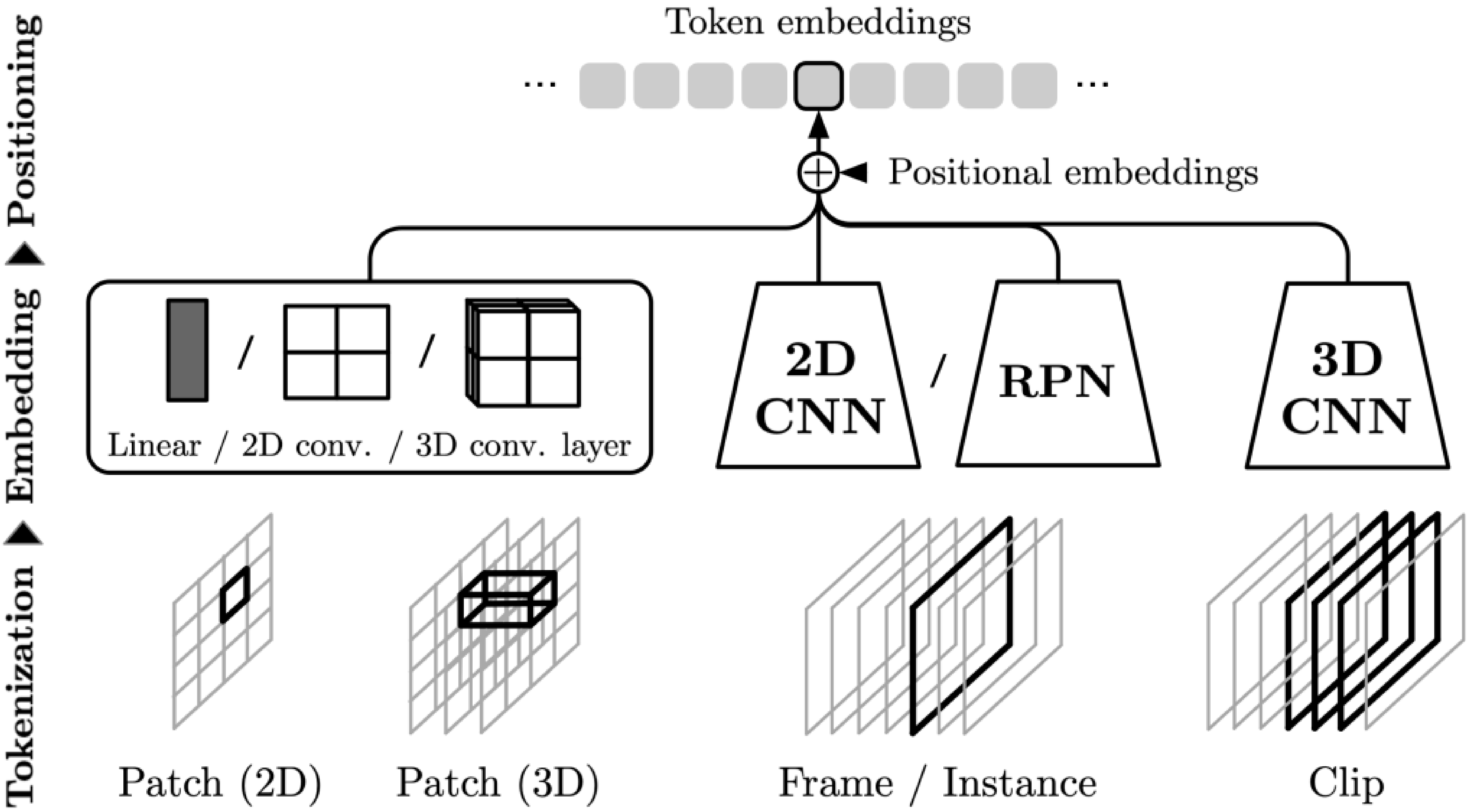
Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows (ICCV 2021)

Findings

Transformers also work well (despite the lack of inductive bias!)
- If trained with **enough data** (to compensate this lack)

Expensive training: 30 days on ImageNet-21k on Google Clout TPUx3 with 8 cores

Video Transformers



Video Transformers: A Survey (PAMI 2023)

Video Transformers: A Survey (PAMI 2023)