

# Advanced NLP -

## Session 3: Attention & Transformers

Prof. Dr. Richard Sieg  
TH Köln IWS - WS 25/26

# 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

# The Illustrated Transformer

by Jay Alammar

# Agenda

01.

Attention

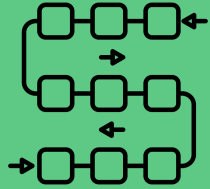
02.

Transformer

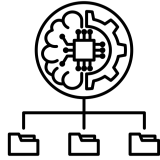
03.

Tutorial

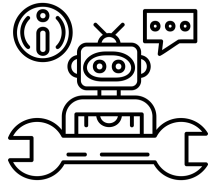
# The 3 Ingredients of LLMs



Process long sequences and context



Efficient training on huge datasets



Follow (human) instructions

01.

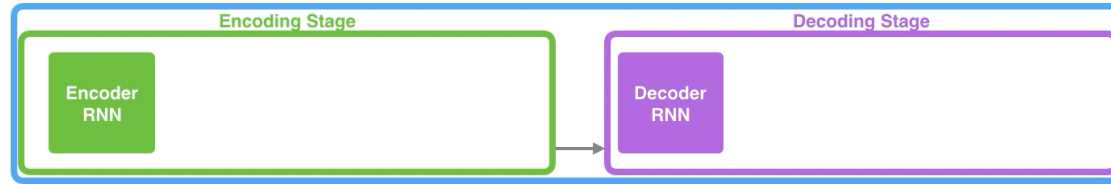
Attention

## Recap: RNNs & LSTMs Issues

- The entire meaning of the source sentence is encapsulated in the last hidden state
- The decoder only uses this last hidden state but has not other interaction with the encoder
- Words are processed in sequential order and thus it is difficult to learn long-distance dependencies
- We have to compute the hidden states sequentially so we cannot parallelize and use GPUs for training and inference

## Neural Machine Translation

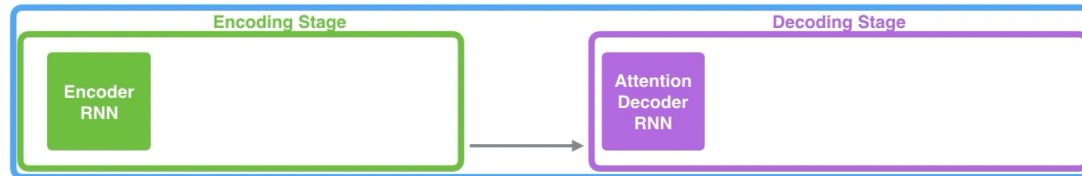
### SEQUENCE TO SEQUENCE MODEL



Je                    suis                    étudiant

## Neural Machine Translation

### SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



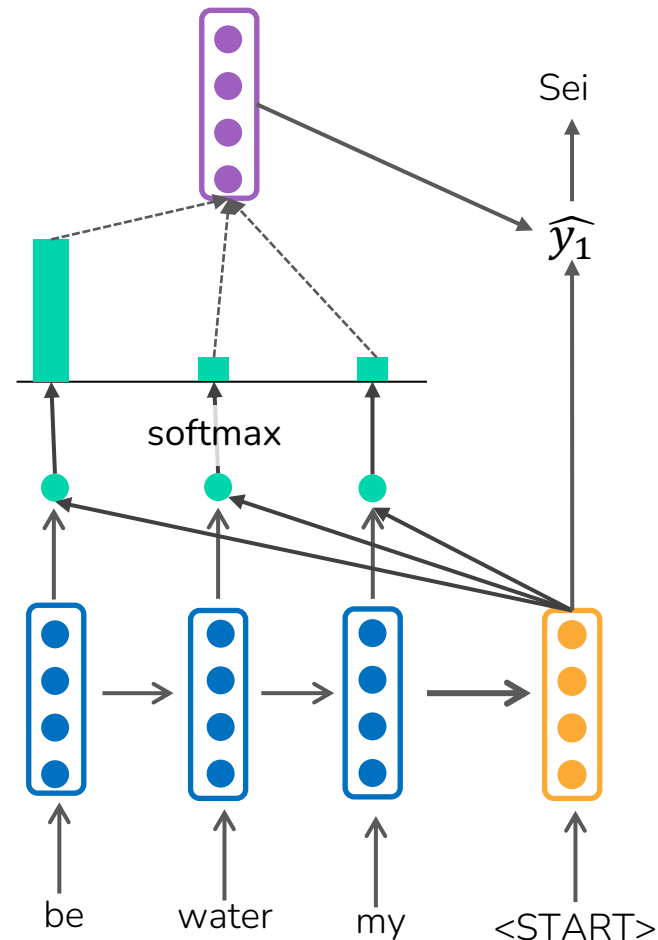
Je                    suis                    étudiant

Attention Decoder  
Bahdanau et al2014  
Luong et al 2015



# Attention - Core Idea

- At each time step  $t$  in the decoder, we take the average of all encoder hidden states as an additional **context vector** combined with the hidden state
- Instead of just taking the normal average of all hidden states we take a **weighted average** depending on the current hidden state
- Weights are given by softmaxed **attention scores** so that they sum up to 1
- And these **attention scores** are learned by the neural network



1. The **Attention Decoder** receives **all** hidden states of the Encoder.
2. At each time step, we compute a score for each encoder hidden state (**attention**)
3. We multiply each hidden state with its **softmaxed** score
4. And add those vectors to get our new **context vector**
5. The context vector is then joined with the hidden state vector to generate the output distribution

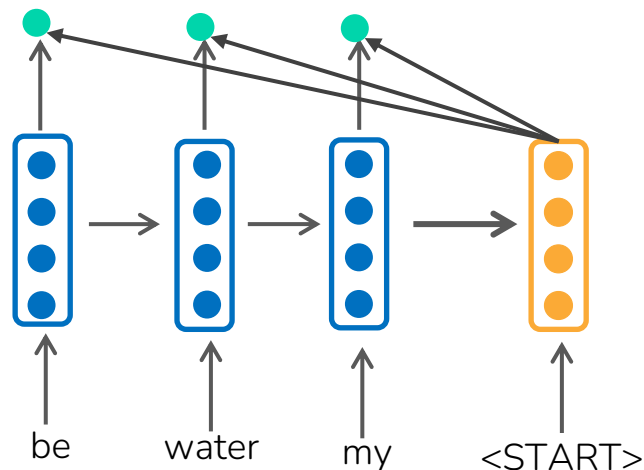
Attention at time step 4





# Computing Attention Scores

- Easiest approach: Take the scalar/dot product between encoder and decoder hidden state  $h_E \cdot h_D$
- More generally: Introduce a weight matrix:  $h_E^T W h_d$
- Use a feed forward layer and project it to a scalar:  $v^T \tanh(W_1 h_E + W_2 h_D)$



But still...

- The encoder and decoder work sequentially and we cannot parallelize computations
- The attention decoder now has all encoder hidden states but only the current decoder hidden state to work with
- Vanishing Gradients: Difficult to learn long-distance relationship

**As it turns out:**

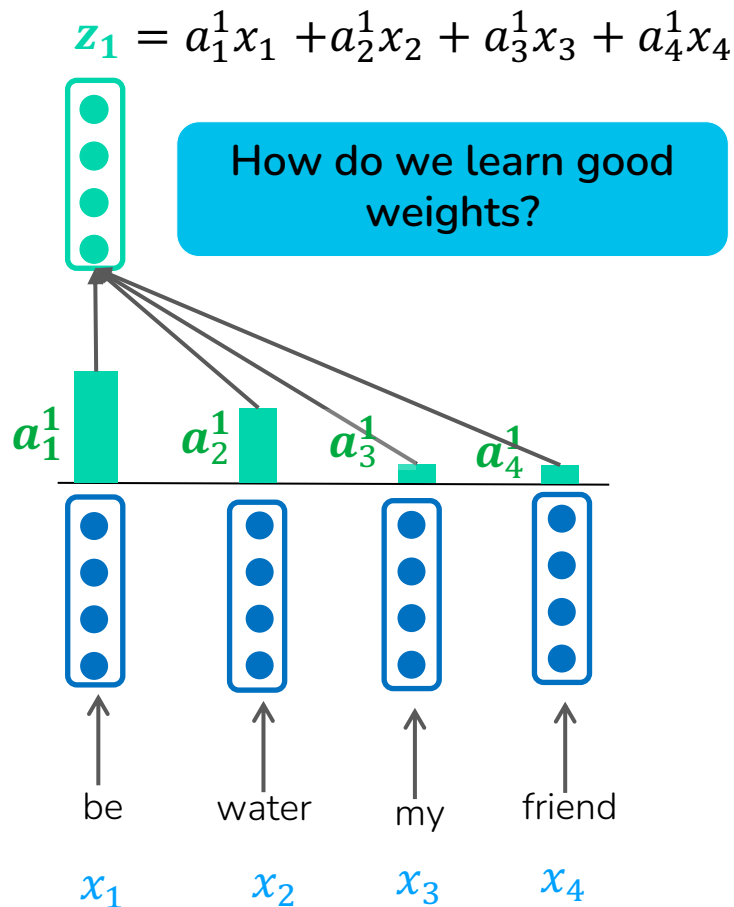
**Attention Is All You Need**

02.

Transformers

# Self-Attention

- Main idea: “Translate” a sequence into itself and calculate attention scores between the tokens of the sequence
- How much do the other (or previous) tokens in the sequence influence a token?
- This **self-attention** score yields a powerful representation of each token that encapsulates the whole context in a **context vector**
- And: All computations can be performed in parallel



# Self-Attention

Divide by  $\sqrt{\dim(k_i)} = 8$  for more stability

$$s_4 = q_1 \cdot k_4 = 8$$

$$a_4 = \sigma(s_4/8) = 0$$

$$s_3 = q_1 \cdot k_3 = 16$$

$$a_3 = \sigma(s_3/8) = .01$$

$$s_2 = q_1 \cdot k_2 = 96$$

$$a_2 = \sigma(s_2/8) = .12$$

$$s_1 = q_1 \cdot k_1 = 112$$

$$a_1 = \sigma(s_1/8) = .87$$

We need three weight matrices

$$W_q, q_i = W_q x_i$$

Queries

$$W_k, k_i = W_k x_i$$

Keys

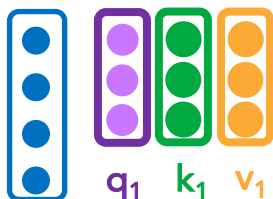
$$W_v, v_i = W_v x_i$$

Values



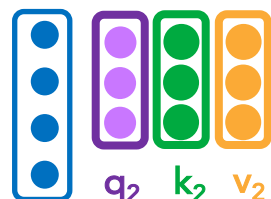
$$z_1 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

$$= 0.87 \cdot v_1 + 0.12 \cdot v_2 + 0.01 \cdot v_3 + 0 \cdot v_4$$



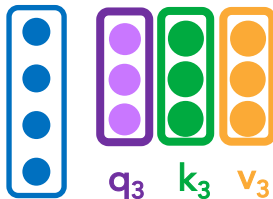
be

$x_1$



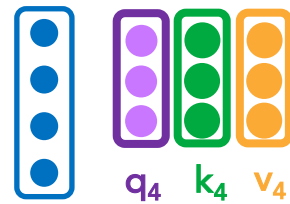
water

$x_2$



my

$x_3$



friend

$x_4$

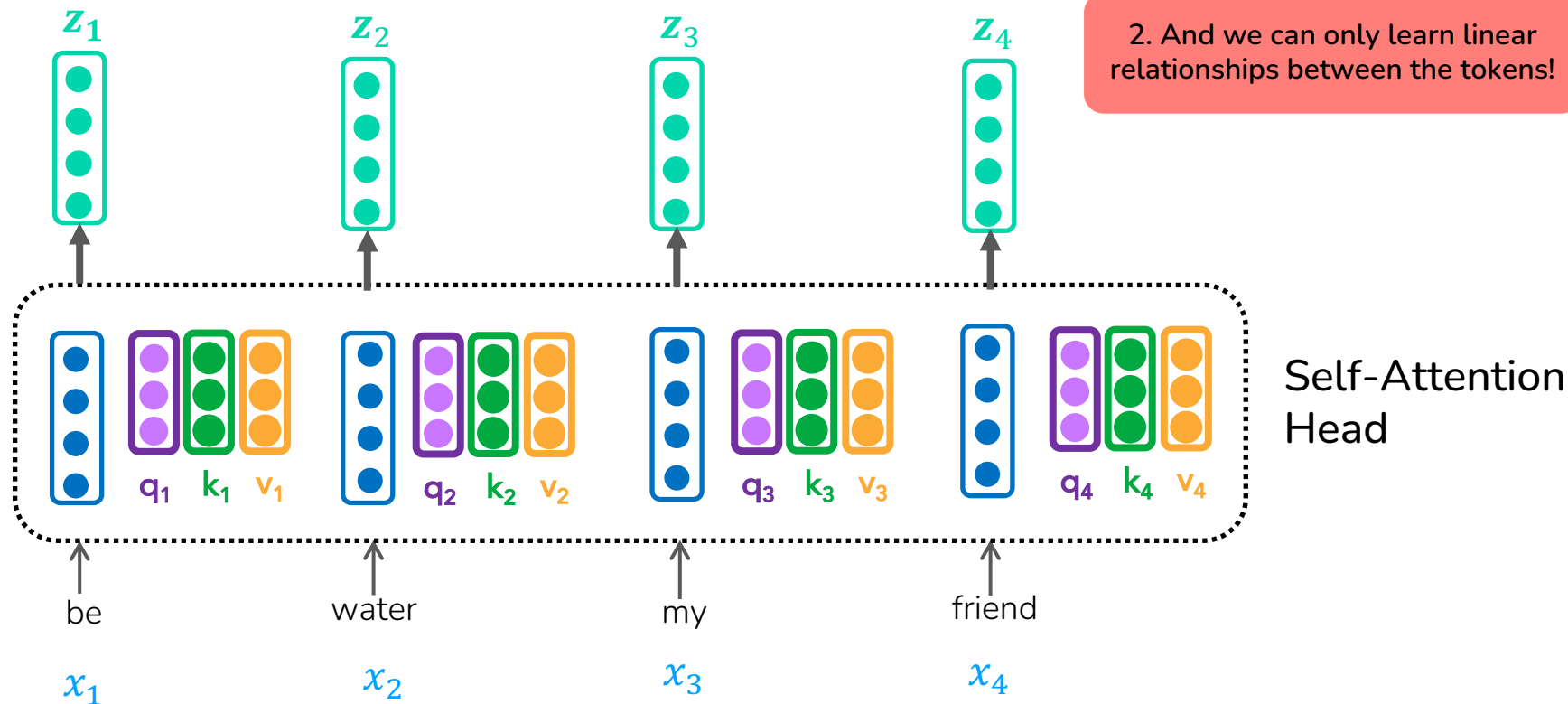


$$\textit{Attention}(Q, K, V) = \sigma(QK^T / \sqrt{d_k})V$$

# Self-Attention

1. But now we have no sense for the order of the tokens!

2. And we can only learn linear relationships between the tokens!

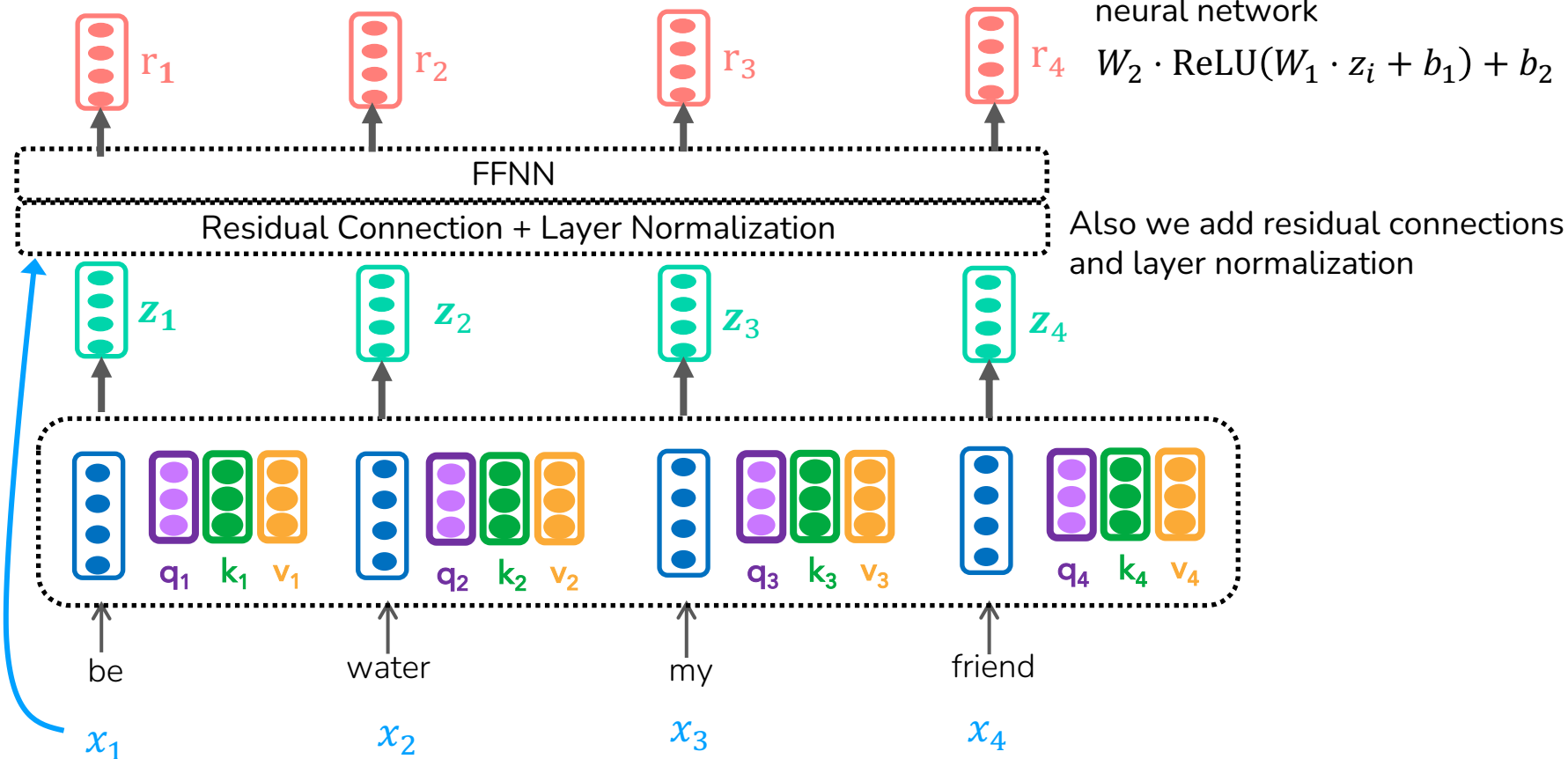


## Fix: Sequence Order

- We do not have a concept for the position of a word (same as bag of words)
- **Solution:** Add position vectors to the embedding inputs  $\tilde{x}_i = x_i + p_i$  where  $p_i \in \mathbb{R}^d$  (**positional embedding**)
- These position vectors can be fixed or learned (out of scope of this lecture)
- Examples of fixed position vectors
- One-hot encodings  $p_i = (0, \dots, 0, 1, 0, \dots, 0)$  (not useful in practice)
- **Sinusoidal** (original paper uses this)

$$p_i = (\sin(i/10000^{2 \cdot 1/d}), \cos(i/10000^{2 \cdot 1/d}), \dots, \sin(i/10000^{2 \cdot d/2/d}), \cos(i/10000^{2 \cdot d/2/d}))$$

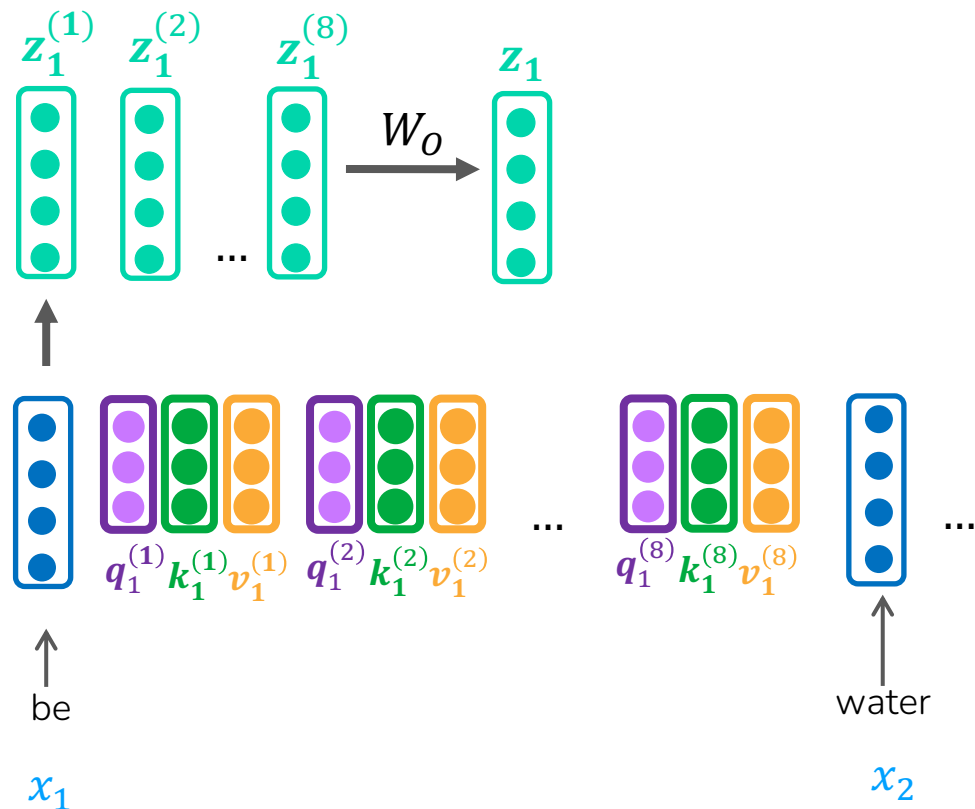
# Fix: Non-Linearity



Let's stack things up!



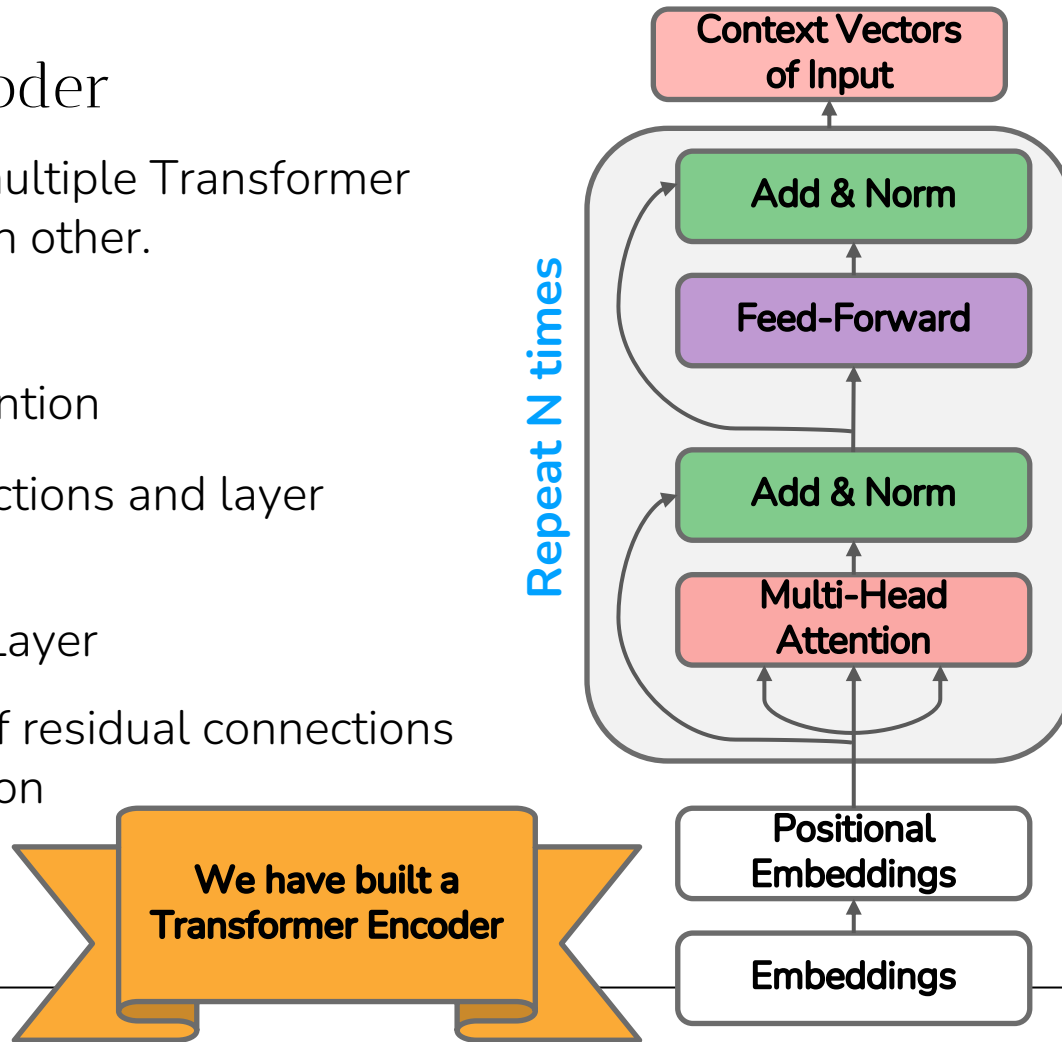
# Multi-Head Attention



- We can use multiple weight matrices  $W_q, W_k, W_v$  in parallel
- This allows tokens to focus on different positions
- We call each set of these matrices an **attention head**
- The output  $z_i$  of each attention head is then concatenated  $[z_1, \dots, z_h]$  and then projected back to the original dimension with a weight matrix  $W_o$

# Transformer Encoder

- Now we can stack multiple Transformer Blocks on top of each other.
- Each block has:
  - Multi-head attention
  - Residual connections and layer normalization
  - Feed-Forward Layer
  - Another layer of residual connections and normalization



# Transformer Decoder

- The Decoder has to generate the output sequence
- So, during training we need to make sure that the Decoder can't look into the future
- Solution: Mask out the attention by setting the attention of future words to  $-\infty$
- Our attention matrix becomes

$$e_{ij} = \begin{cases} q_i^T k_j, j \leq i \\ -\infty, j > i \end{cases}$$

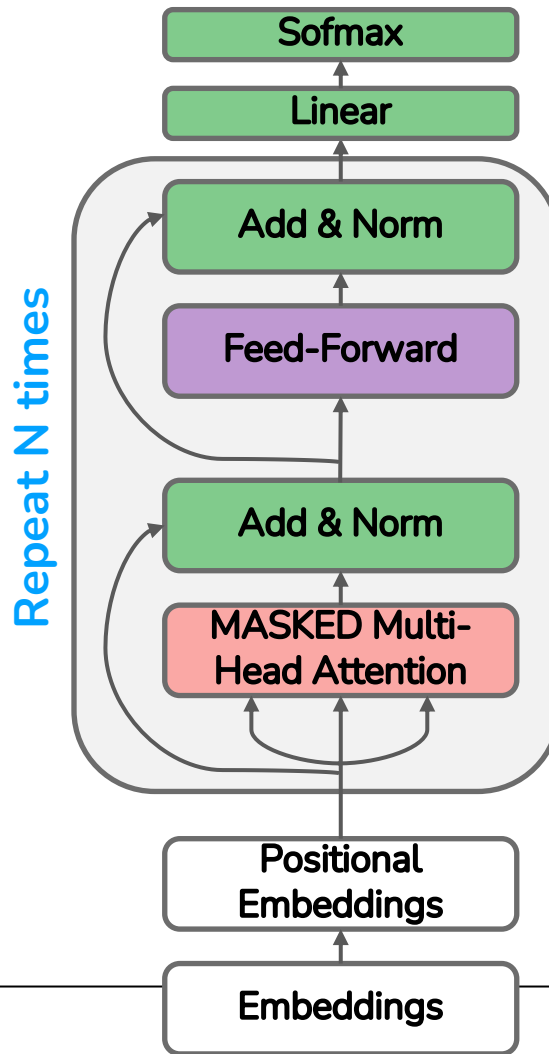
	<Start>	be	water	my	friend
<START>		$-\infty$	$-\infty$	$-\infty$	$-\infty$
be			$-\infty$	$-\infty$	$-\infty$
water				$-\infty$	$-\infty$
my					$-\infty$
friend					



# Transformer Decoder

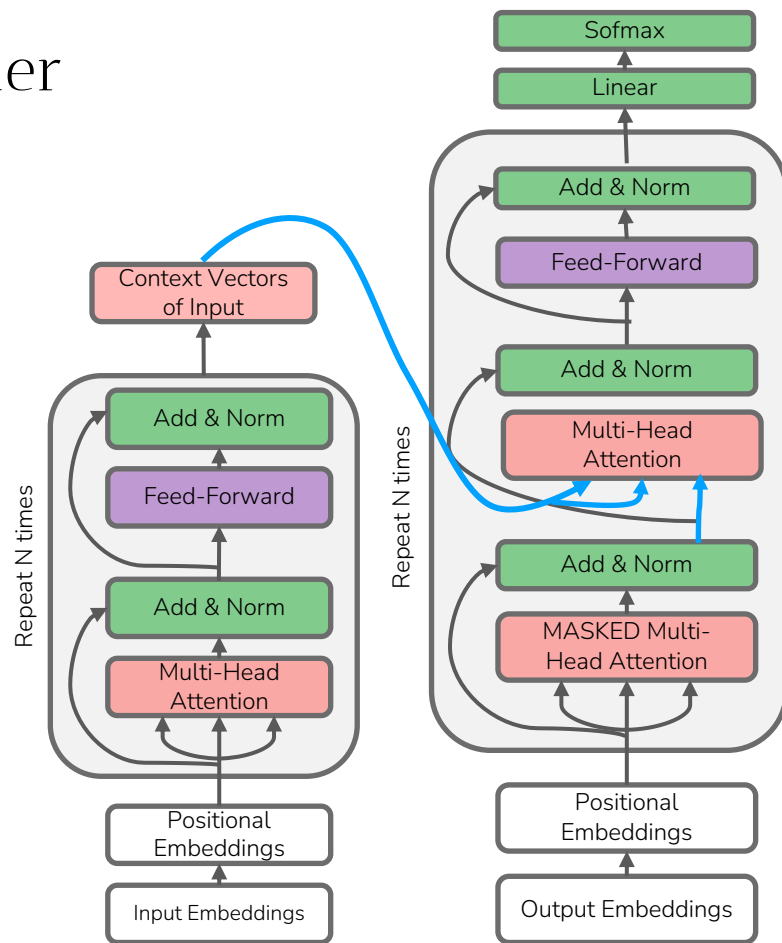
- The Transformer Decoder Block is the same as the Encoder just with Masked Multi-Head Attention
- So, the Decoder performs as an **auto-regressive** Language Model 🗯️
- In the end, we add a Linear and a Softmax Layer to get the the next word probabilities

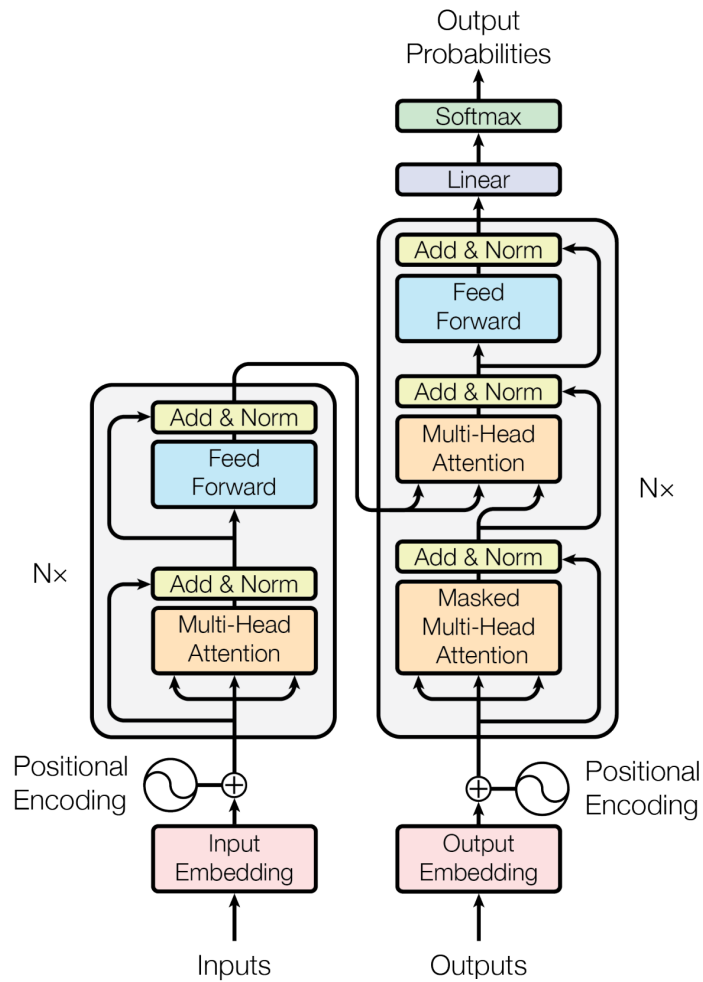
**We have built a  
Transformer Decoder**



# The Transformer Encoder-Decoder

- For seq2seq tasks like Machine Translation, we can couple the Encoder and Decoder
- For this we add another Multi-Head Attention Block to the Decoder that receives the output vectors  $z_1, \dots, z_n$  from the Encoder (**Cross-Attention**)
- In this block, the keys and values are computed using the Encoder output vectors and the queries are computed using the Decoder input vectors  $h_1, \dots, h_n$
- $k_i = W_k z_i, v_i = W_v z_i, q_i = W_q h_i$





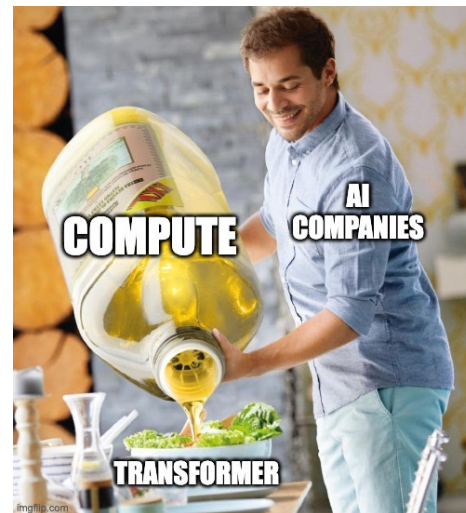
# The Success of the Transformer

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.8</b>	$2.3 \cdot 10^{19}$	

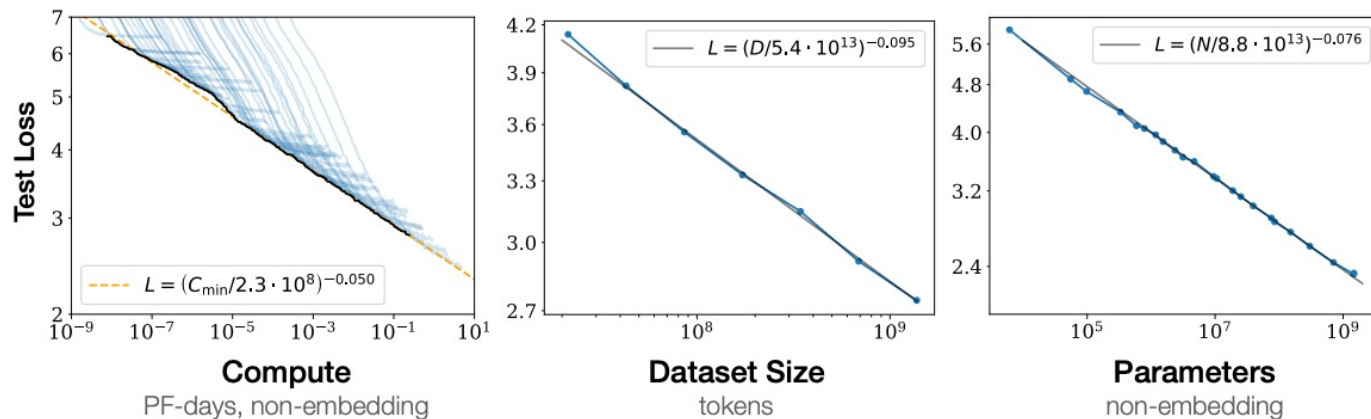
	Base	Big
Layers in Encoder & Decoder	6	6
#Heads	8	16
Model Dimension	512	1024
Feed-Forward Dimension	2048	4096

# The Success of the Transformer

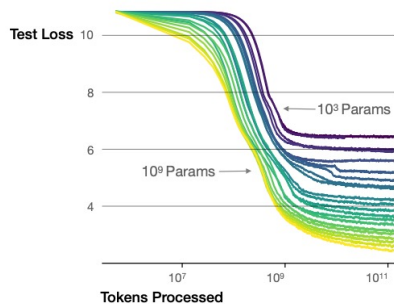
- As it turns out, the Transformer architecture is amazing for pre-training models on huge datasets (see next lecture)
- And the Encoder (BERT etc) or Decoder (GPT etc) themselves make great Language Models
- The rest is scaling (although attention has quadratic costs)



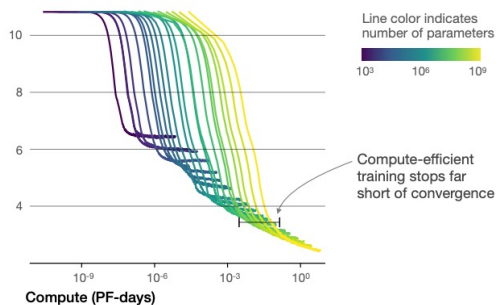
# Scaling Transformer Models



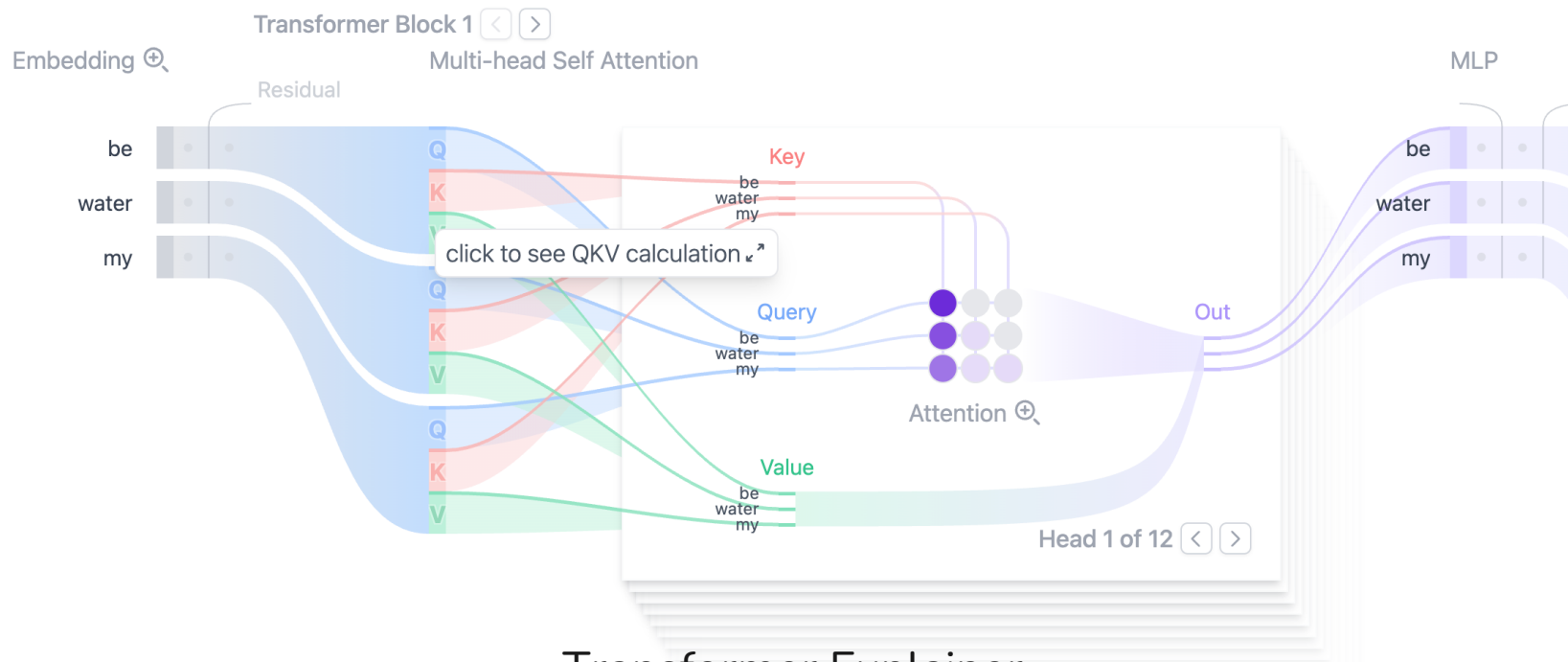
Larger models require **fewer samples** to reach the same performance



The optimal model size grows smoothly with the loss target and compute budget



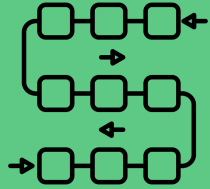
# Folks do great visualizations of the transformer



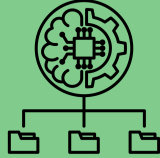
Transformer Explainer

Next lecture: BertViz in the tutorial session

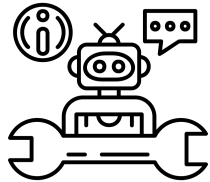
# The 3 Ingredients of LLMs



Process long sequences and context



Efficient training on huge datasets



Follow (human) instructions



# Tutorial

