



CS 394E: Contemporary Topics in LLMs

Transformers

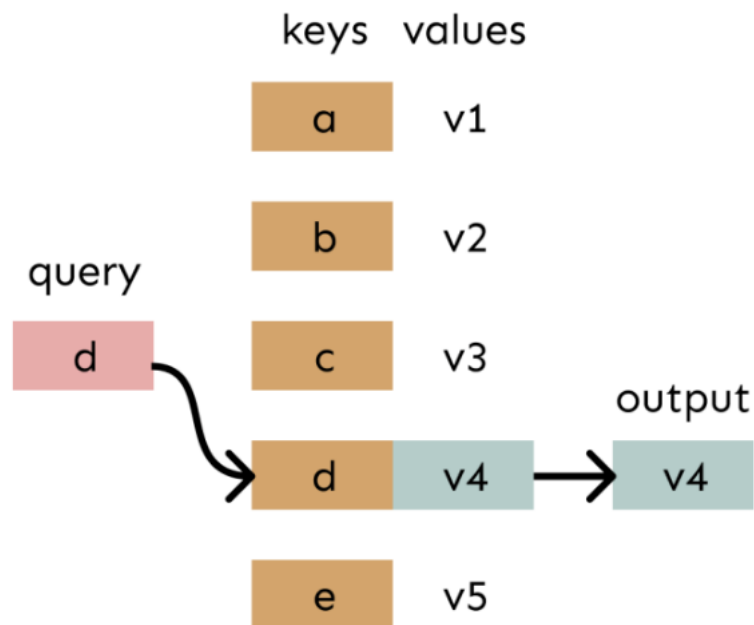
Marco Canini

Professor of Computer Science, KAUST

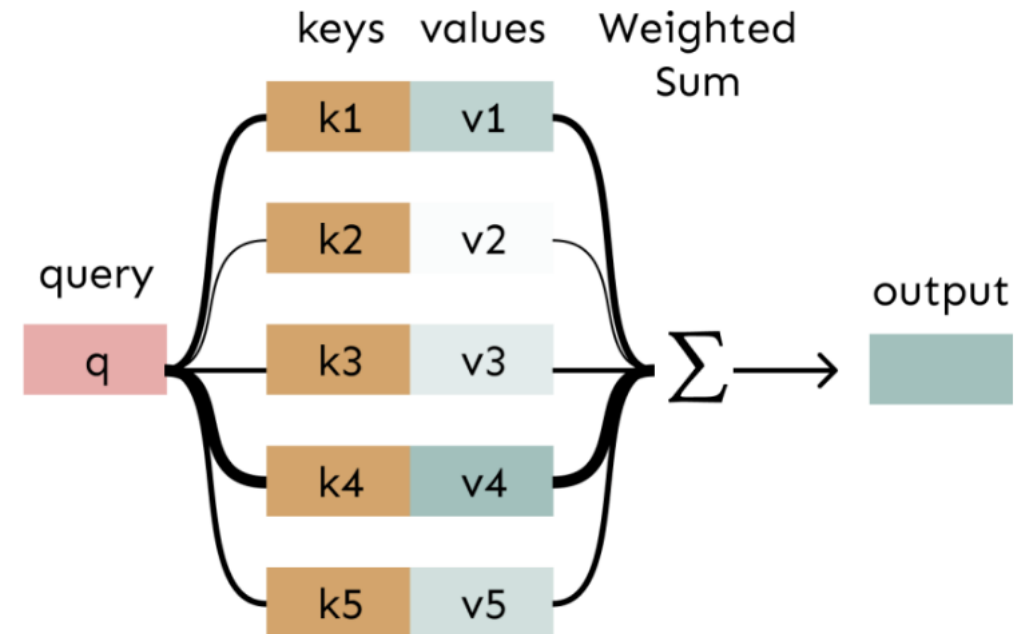
Intuition: Attention as an Averaging Lookup Table

We can think of **attention** as performing fuzzy lookup a in **key-value store**

Lookup table: a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



Attention: The **query** matches to all **keys** softly to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



Self-Attention in I-Slide

- Step 1: For each word x_i , calculate its **query**, **key**, and **value**

$$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

- Step 2: Calculate attention score between **query** and **keys**

$$e_{ij} = q_i \cdot k_j / \sqrt{D_Q}$$

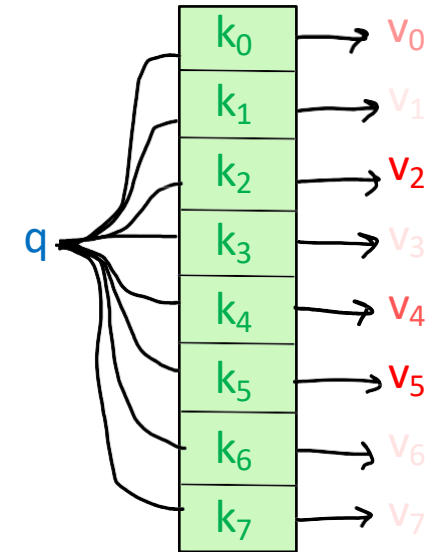
D_Q : Dimensions in the query and key embeddings

- Step 3: Take the softmax to normalize attention scores

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}$$

- Step 4: Take a weighted sum of **values**

$$\begin{aligned} \text{Output}_i &= \sum_j \alpha_{ij} v_j \end{aligned}$$

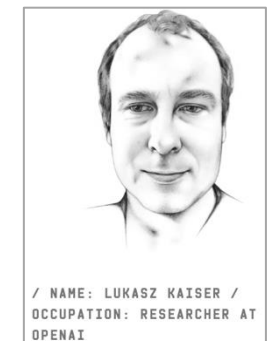
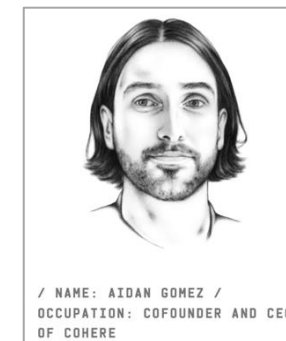
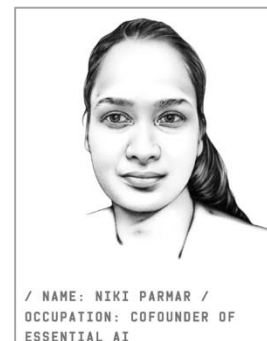
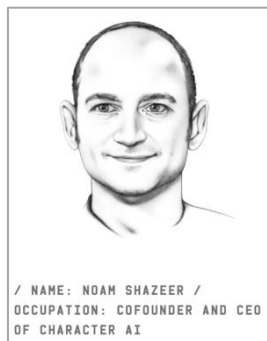


Enter Transformers



Attention is All You Need

Ashish Vaswani · Noam Shazeer · Niki Parmar · Jakob Uszkoreit · Llion Jones · Aidan N. Gomez · Lukasz Kaiser · Illia Polosukhin · NIPS(2017)

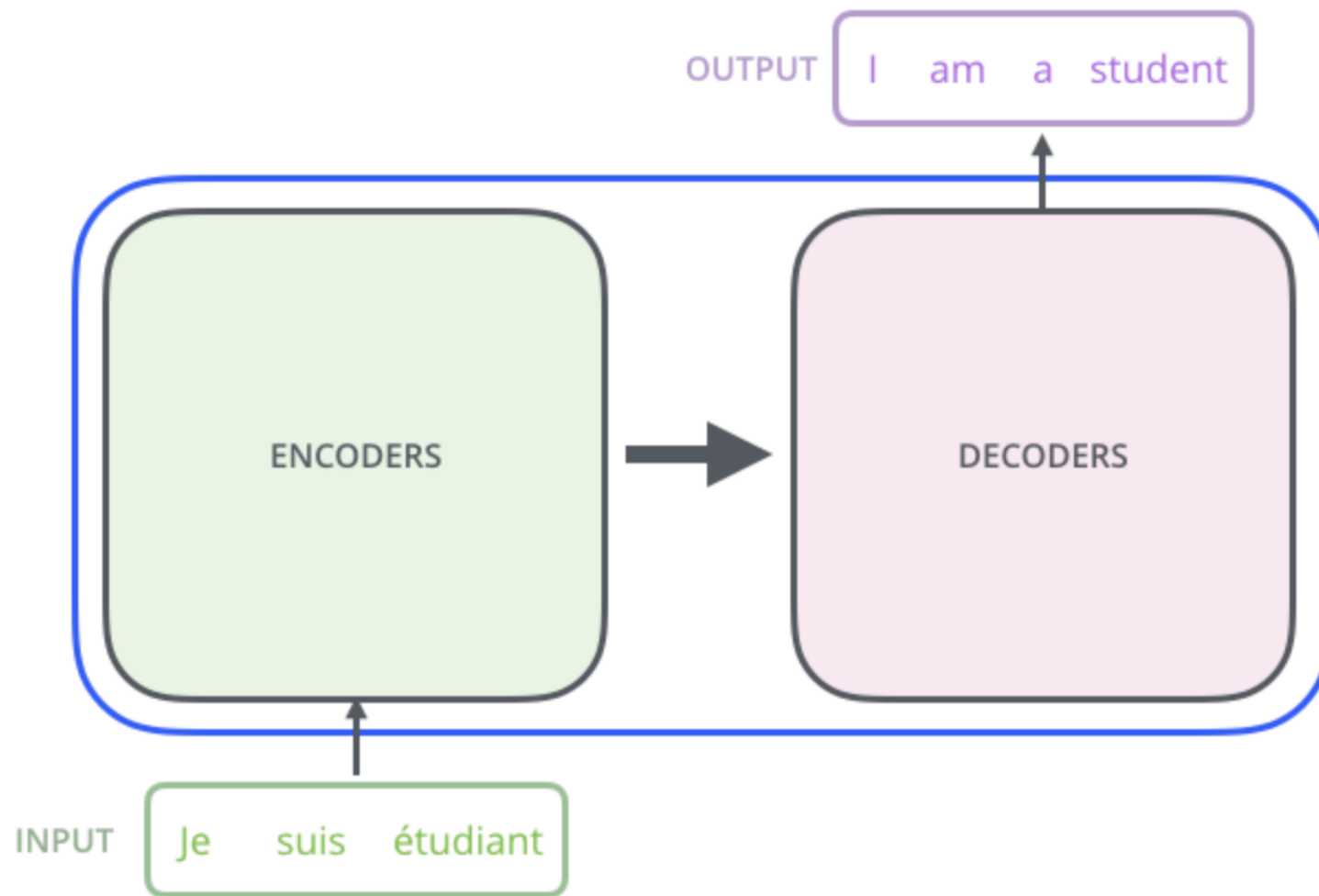


TITLE	CITED BY	YEAR
Attention is all you need A Vaswani, N Shazeer, N Parmar, J Uszkoreit, L Jones, AN Gomez, ... Advances in neural information processing systems 30	203486	2017

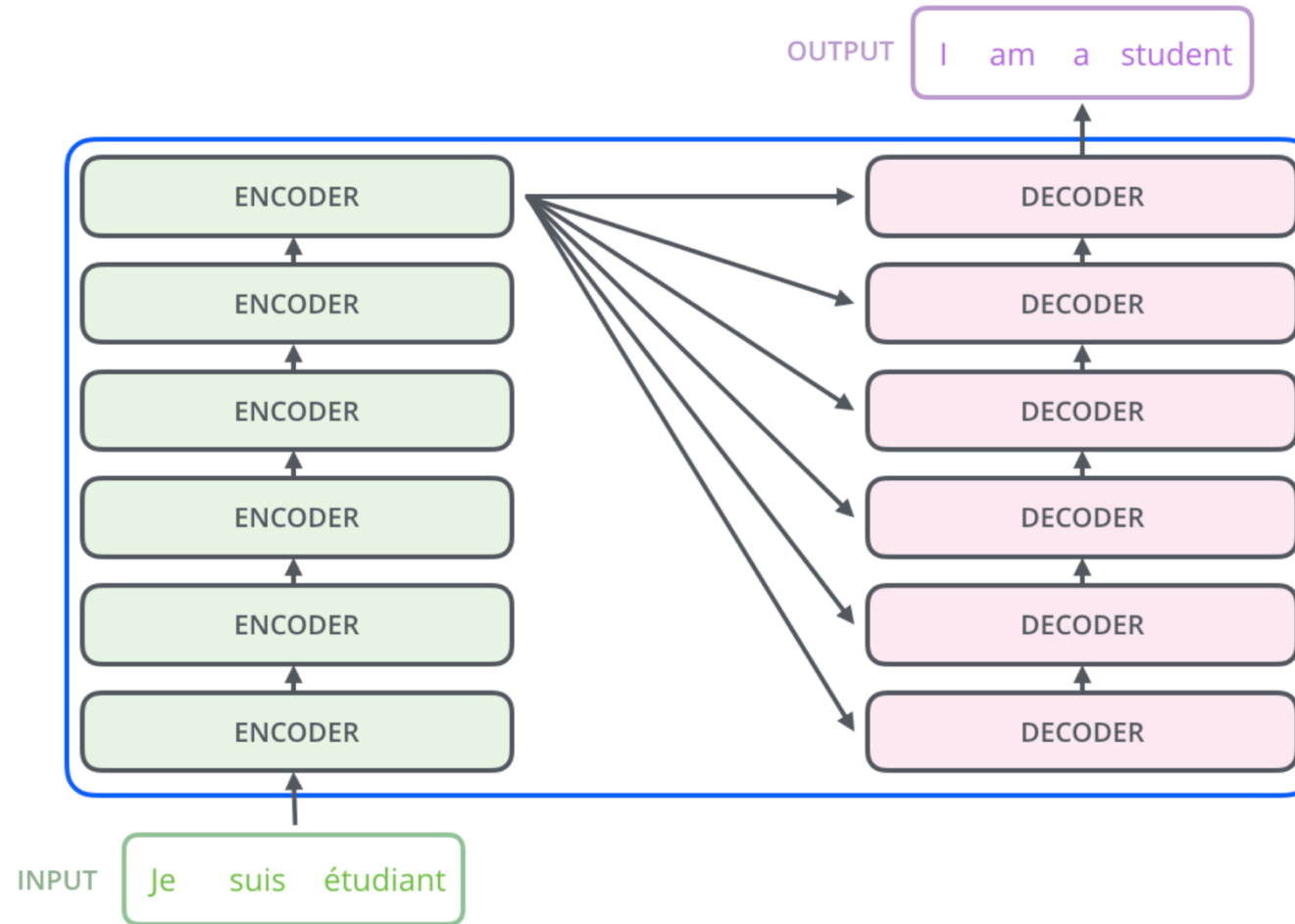
Neural networks, in particular recurrent neural networks (RNNs), are now at the core of the leading approaches to language understanding tasks such as language modeling, machine translation and question answering. In “Attention Is All You Need”, we introduce the Transformer, a novel neural network architecture based on a self-attention mechanism that we believe to be particularly well suited for language understanding.

Transformers

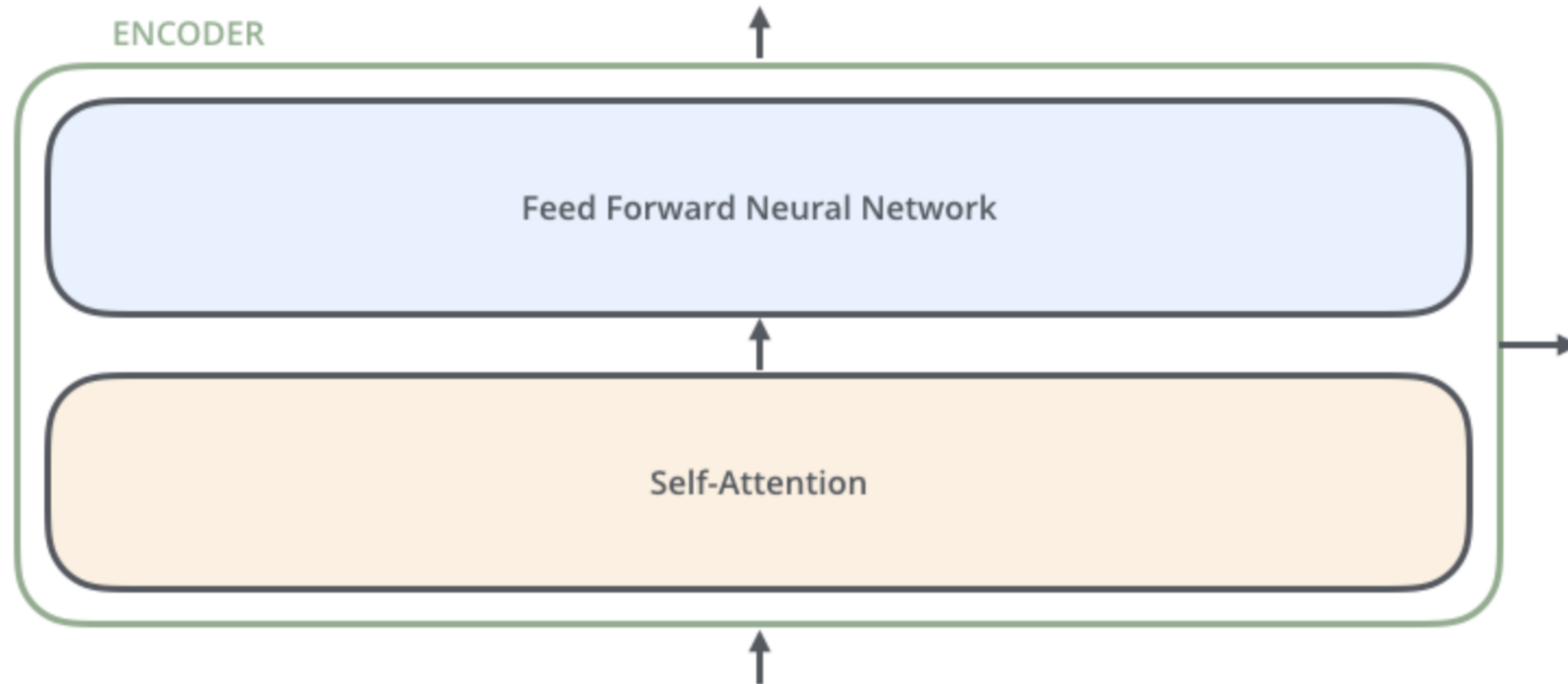
High-Level Architecture



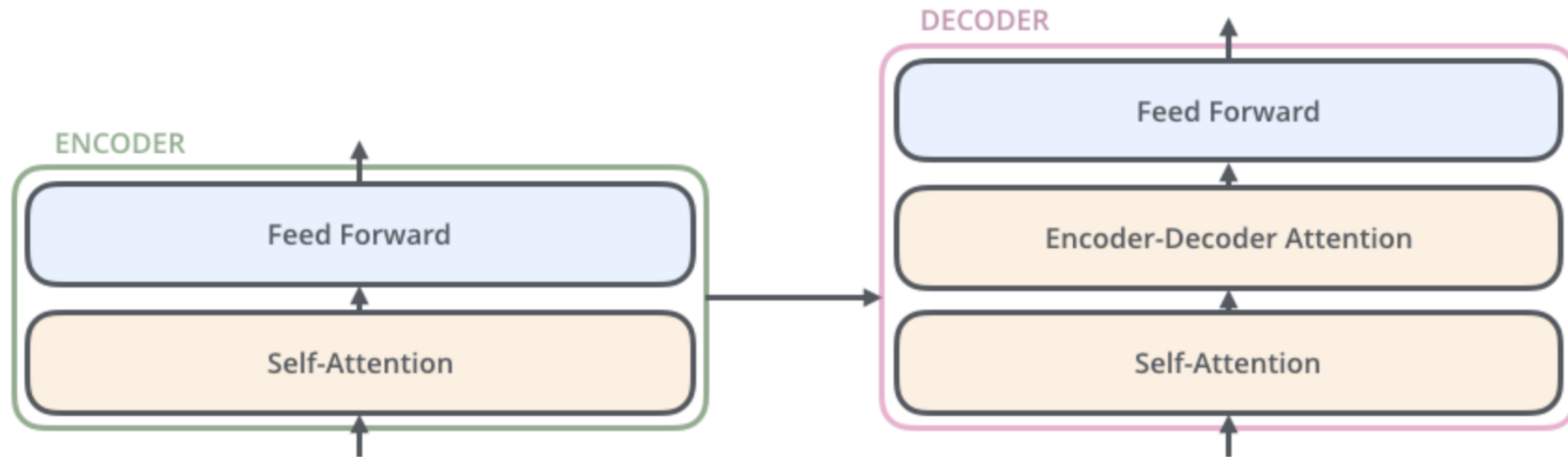
Architecture Top-Down



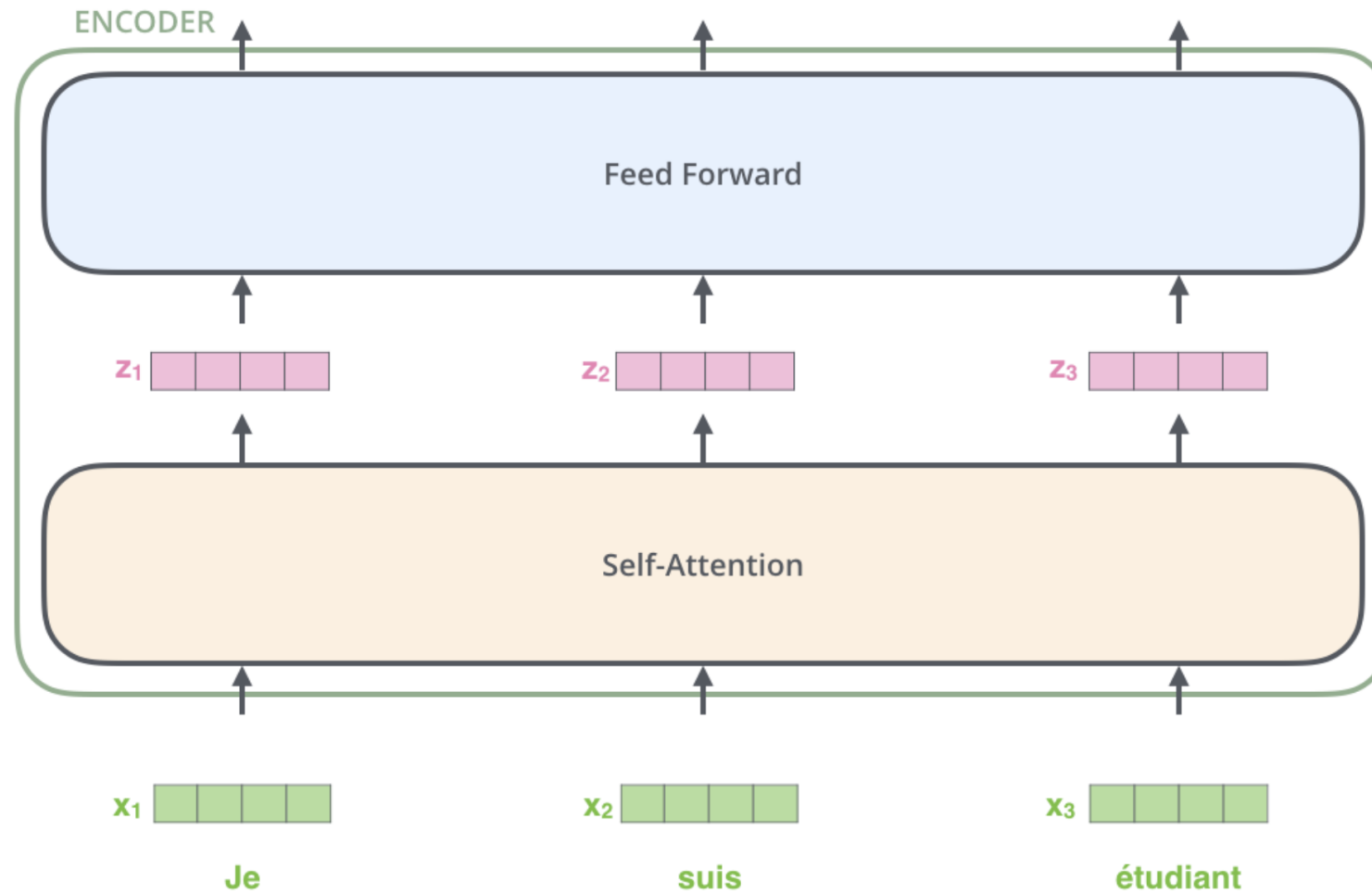
Encoder Layer



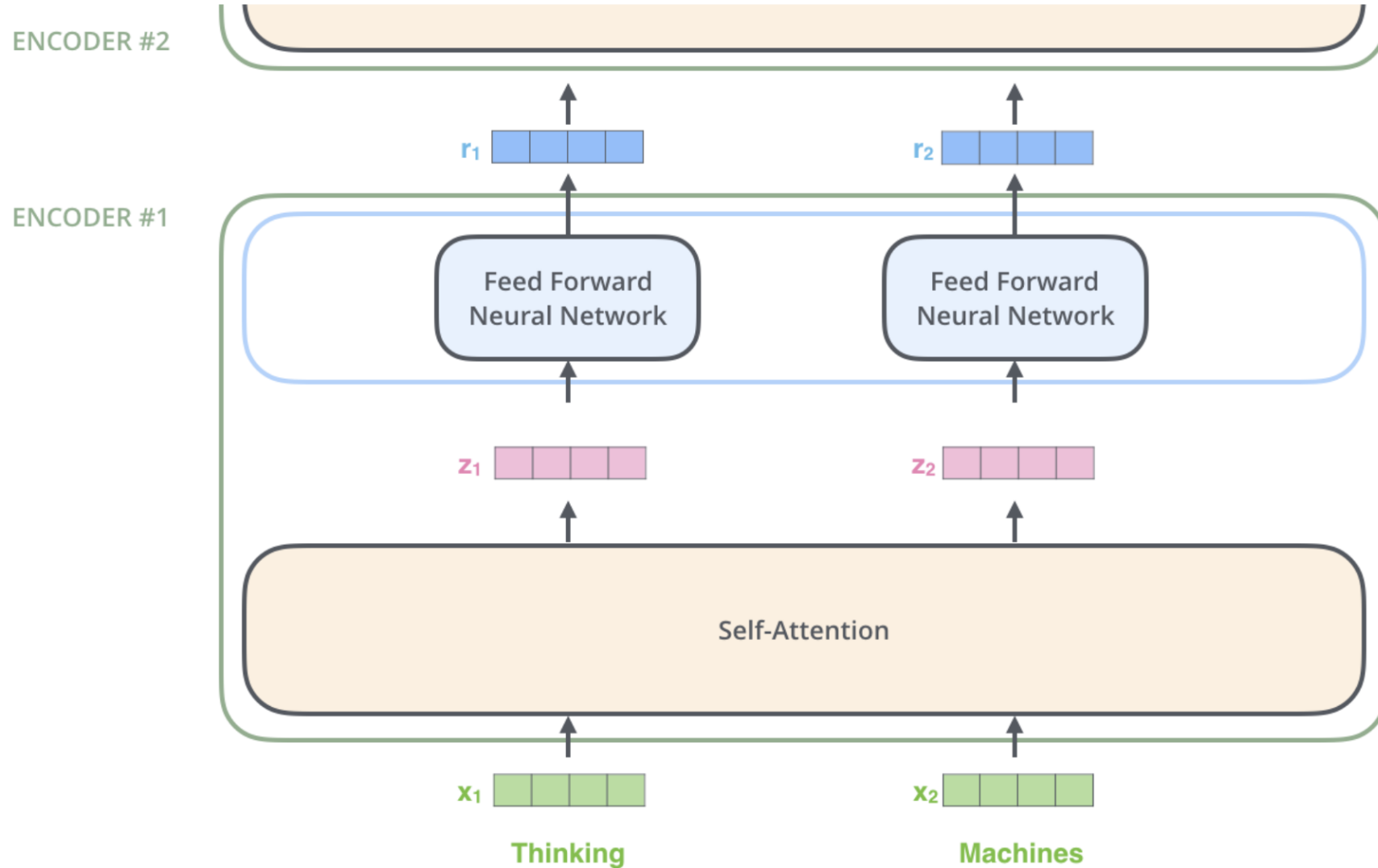
Encoder vs. Decoder Layer



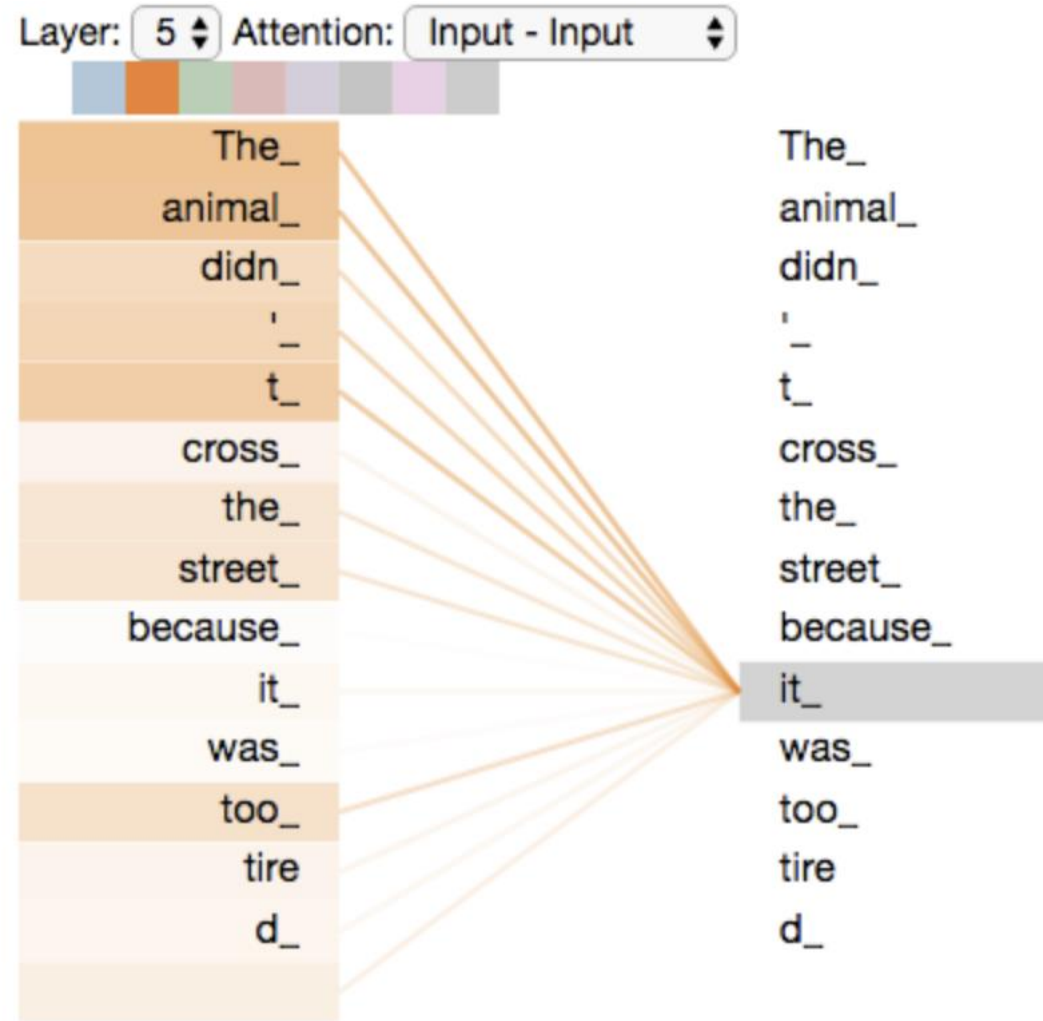
Intra-Encoder Information Flow



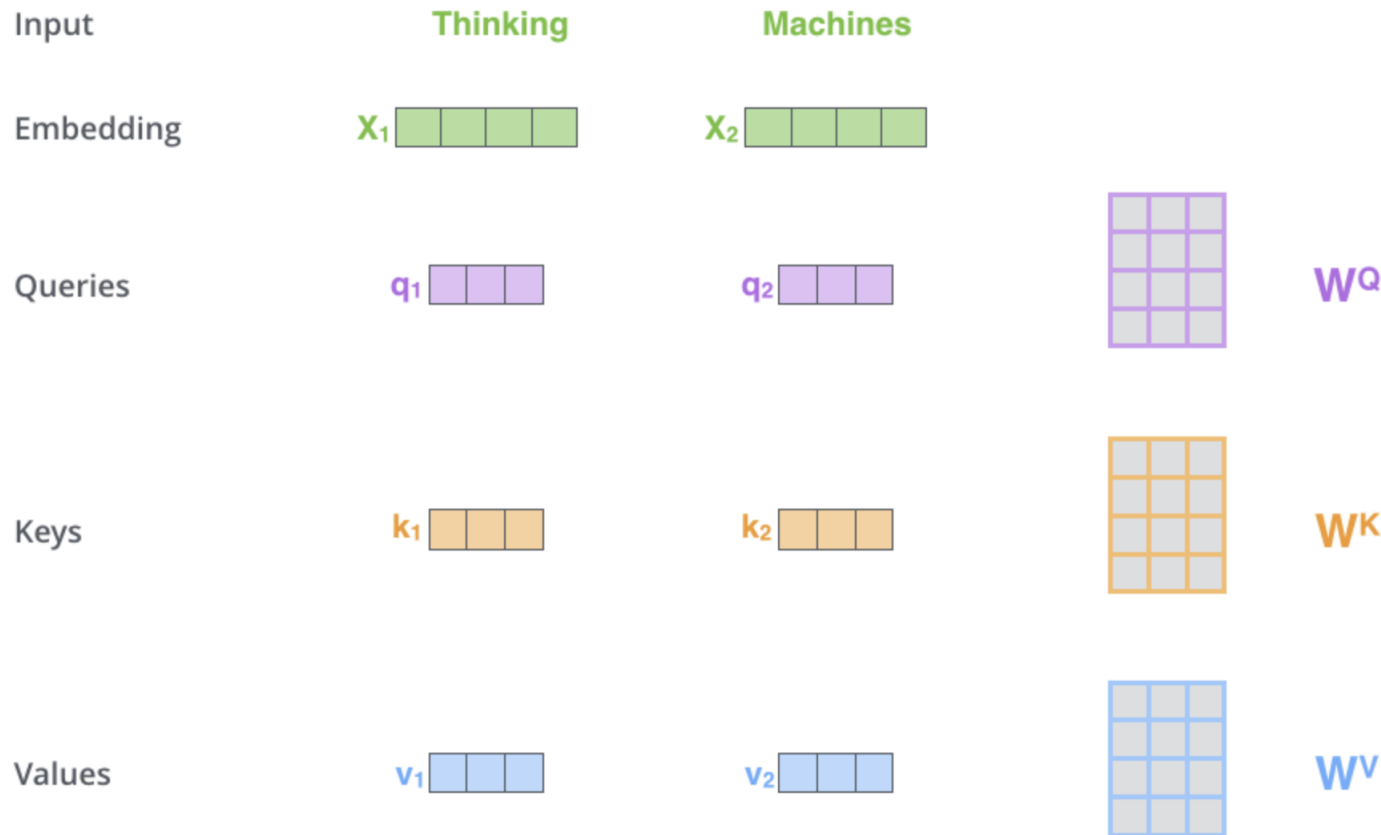
Across Encoder Information Flow



Self-Attention



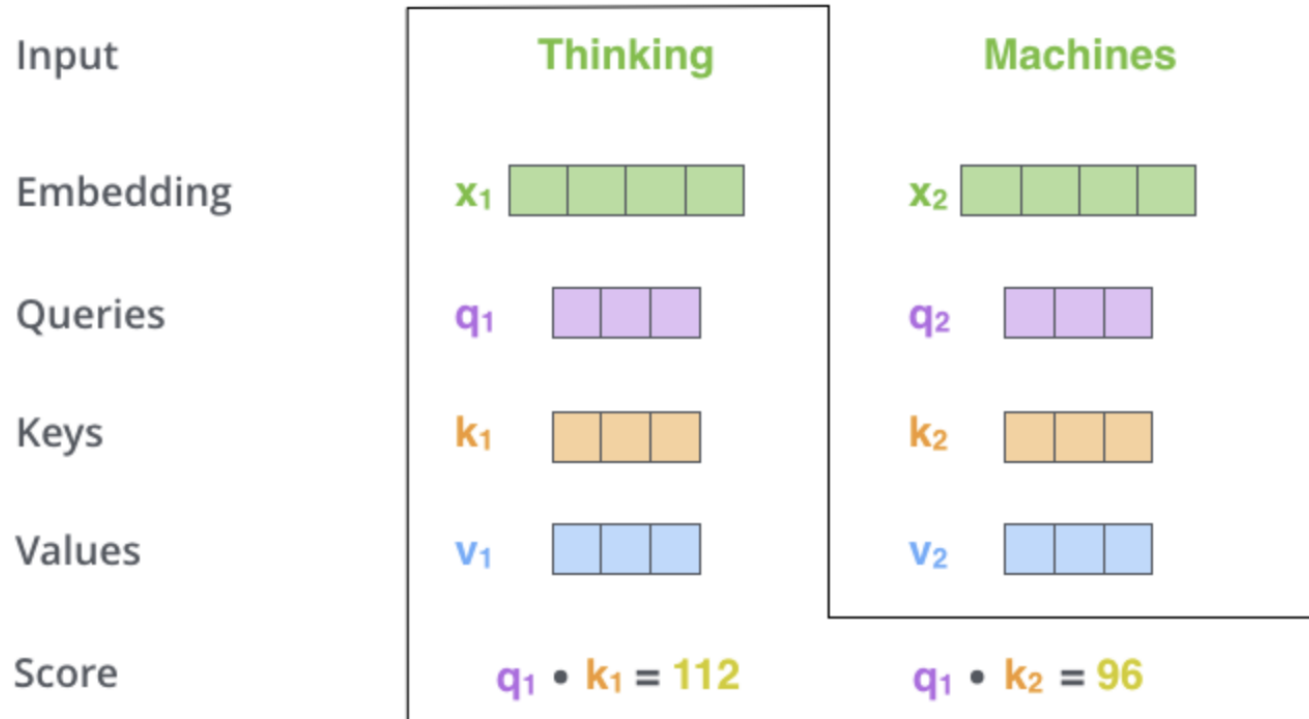
Self Attention: The Three Matrices



Step 1

Multiplying x_1 by the W^Q weight matrix produces q_1 , the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

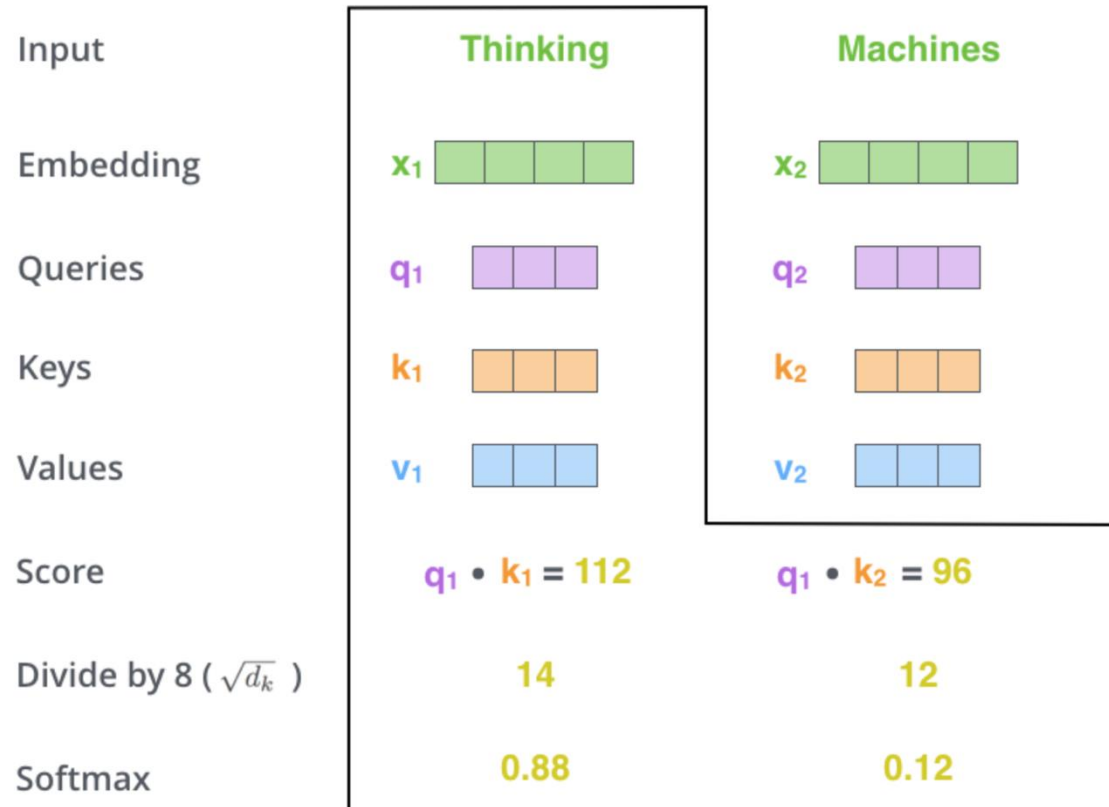
Self Attention Score: Dot Product



Step 2

The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring. So if we're processing the self-attention for the word in position #1, the first score would be the dot product of q_1 and k_1 . The second score would be the dot product of q_1 and k_2 .

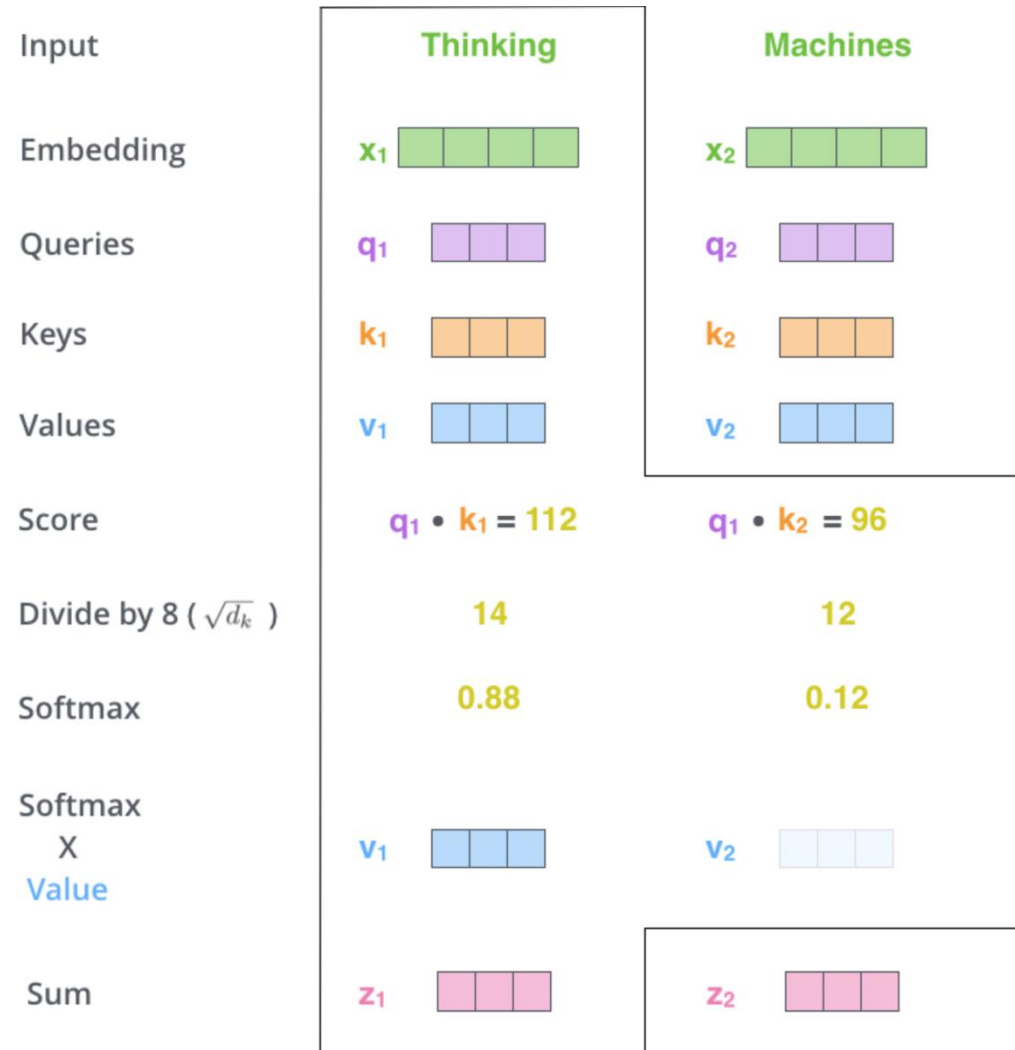
Self Attention Score: Normalization



Step 3 and 4

The **third and fourth steps** are to divide the scores by 8 (the square root of the dimension of the key vectors used in the paper – 64. This leads to having more stable gradients. Then pass the result through a softmax operation. Softmax normalizes the scores so they're all positive and add up to 1.

Self-Attention: Weights



Step 5 and 6

The **fifth step** is to multiply each value vector by the softmax score (in preparation to sum them up). The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words (by multiplying them by tiny numbers like 0.001, for example). The **sixth step** is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for the first word).

Matrix Calculation of Self-Attention

$$\begin{matrix} X \\ \text{2x4 grid} \end{matrix} \times \begin{matrix} W^Q \\ \text{4x4 grid} \end{matrix} = \begin{matrix} Q \\ \text{2x4 grid} \end{matrix}$$

$$\begin{matrix} X \\ \text{2x4 grid} \end{matrix} \times \begin{matrix} W^K \\ \text{4x4 grid} \end{matrix} = \begin{matrix} K \\ \text{2x4 grid} \end{matrix}$$

$$\begin{matrix} X \\ \text{2x4 grid} \end{matrix} \times \begin{matrix} W^V \\ \text{4x4 grid} \end{matrix} = \begin{matrix} V \\ \text{2x4 grid} \end{matrix}$$

$$\text{softmax} \left(\frac{\begin{matrix} Q \\ \text{2x4 grid} \end{matrix} \times \begin{matrix} K^T \\ \text{4x2 grid} \end{matrix}}{\sqrt{d_k}} \right) \begin{matrix} V \\ \text{2x4 grid} \end{matrix} = \begin{matrix} Z \\ \text{2x4 grid} \end{matrix}$$

Why Multi-head Attention?

- Each **attention head** focuses on different aspects of the input
 - Each head transforms the input differently, producing different Q, K, V spaces
- **Example:** “The cat sat on the mat”
 - **Head 1 – Subject Focus**
→ “cat” attends to itself (who is involved)
 - **Head 2 – Action Focus**
→ “cat” attends to “sat” (what it does)
 - **Head 3 – Location Focus**
→ “cat” attends to “mat” (where it happens)
- **Final Output:** Combines all heads → captures subject, action, and location

$$Multi - head\ output = Linear(Concat(Head_1, Head_2, \dots, Head_h))$$

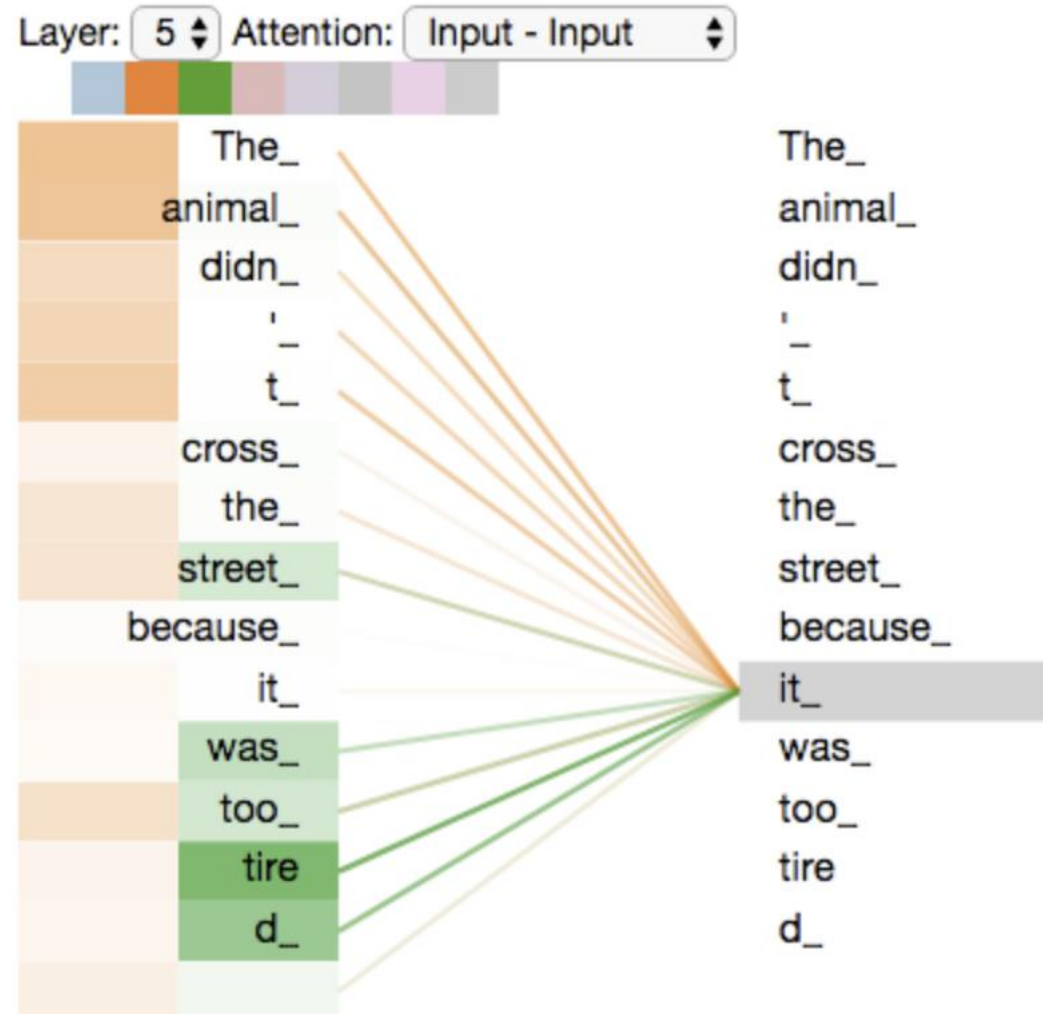
Why is Multi-head Attention Useful?

- Heads can focus on **different patterns** (short-range, long-range, syntax, semantics)
- Separate Q, K, V for each head $\rightarrow$ multiple **representation spaces**

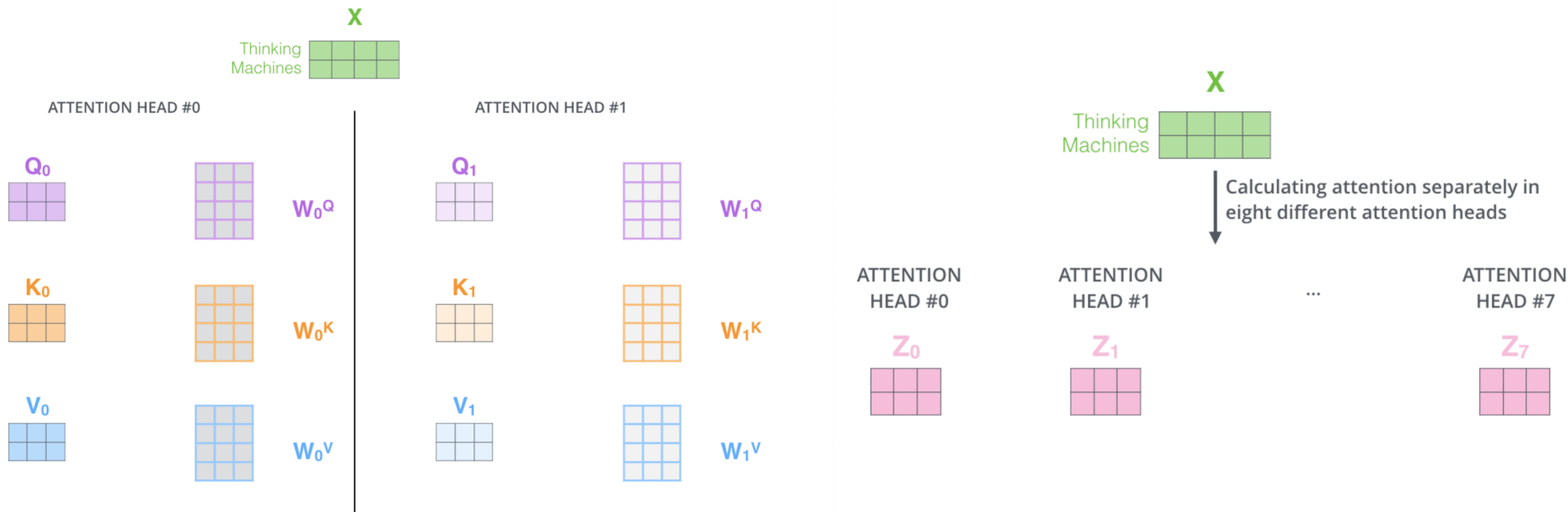
Why Multi-head Attention Specializes?

- **No explicit instruction:** The model is not told “*Head 1 = subject*”
- **Random initialization:** Each head starts with different random W_Q, W_K, W_V
- **Gradient updates:** During training (via backprop + loss minimization), each head learns transformations that reduce prediction error
- **Specialization emerges:**
 - Because different projections → different “views” of the same sequence
 - Some heads become useful at picking up syntax (e.g., subject-verb)
 - Others pick up semantics (who is related to what)
 - Redundancy is penalized indirectly; if two heads do the *same thing*, it’s less helpful for minimizing loss, so they tend to spread out to capture complementary patterns
- **Empirical evidence:** In practice, visualization of attention weights shows that some heads specialize
 - Others may remain redundant or less interpretable

Why Multi-Head Attention?



Multi-Head Attention: Separate Matrices



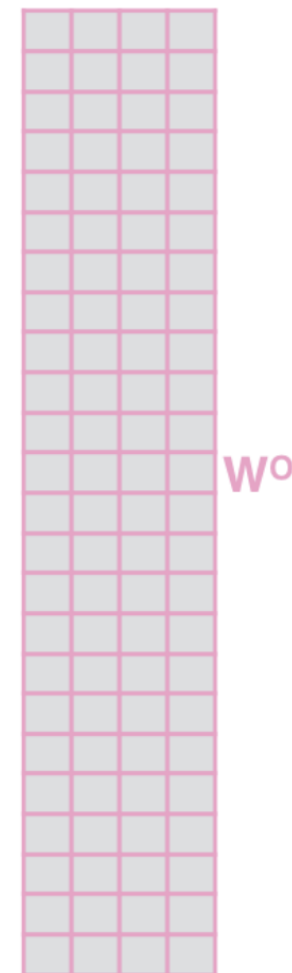
Multi-Head Attention: Concatenation

1) Concatenate all the attention heads

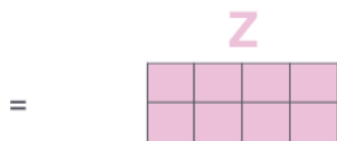


2) Multiply with a weight matrix W^O that was trained jointly with the model

$\times$



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



Putting All Together

1) This is our input sentence*

2) We embed each word*

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

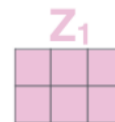
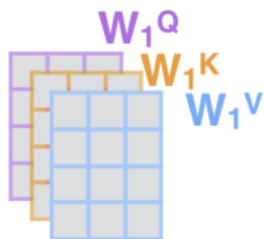
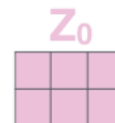
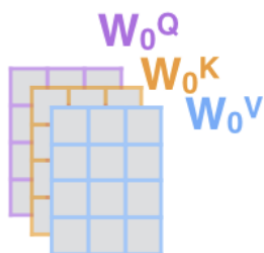
4) Calculate attention using the resulting $Q/K/V$ matrices

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

Thinking
Machines



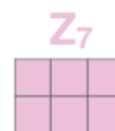
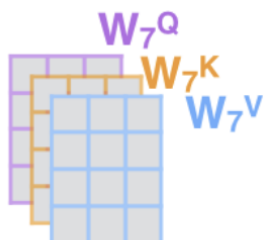
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



...

...

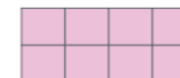
...



W^O

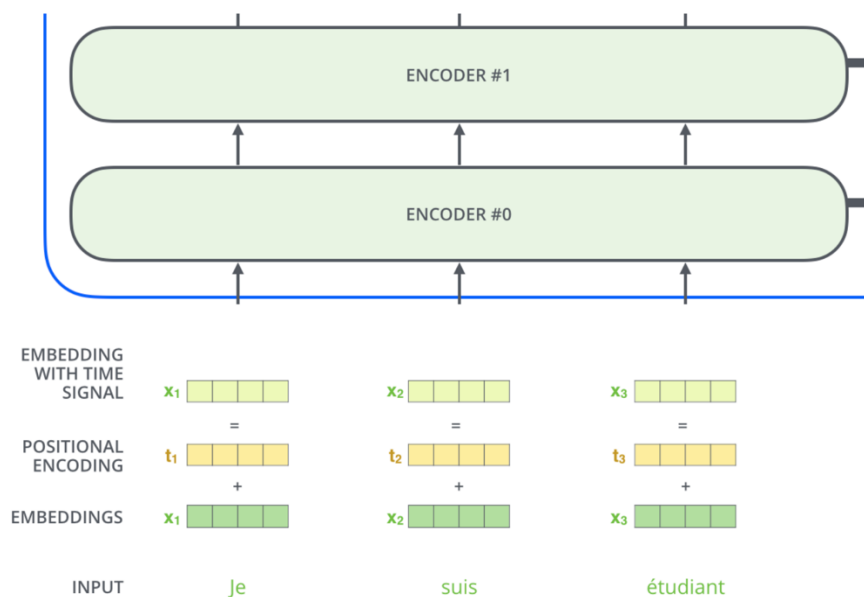


Z



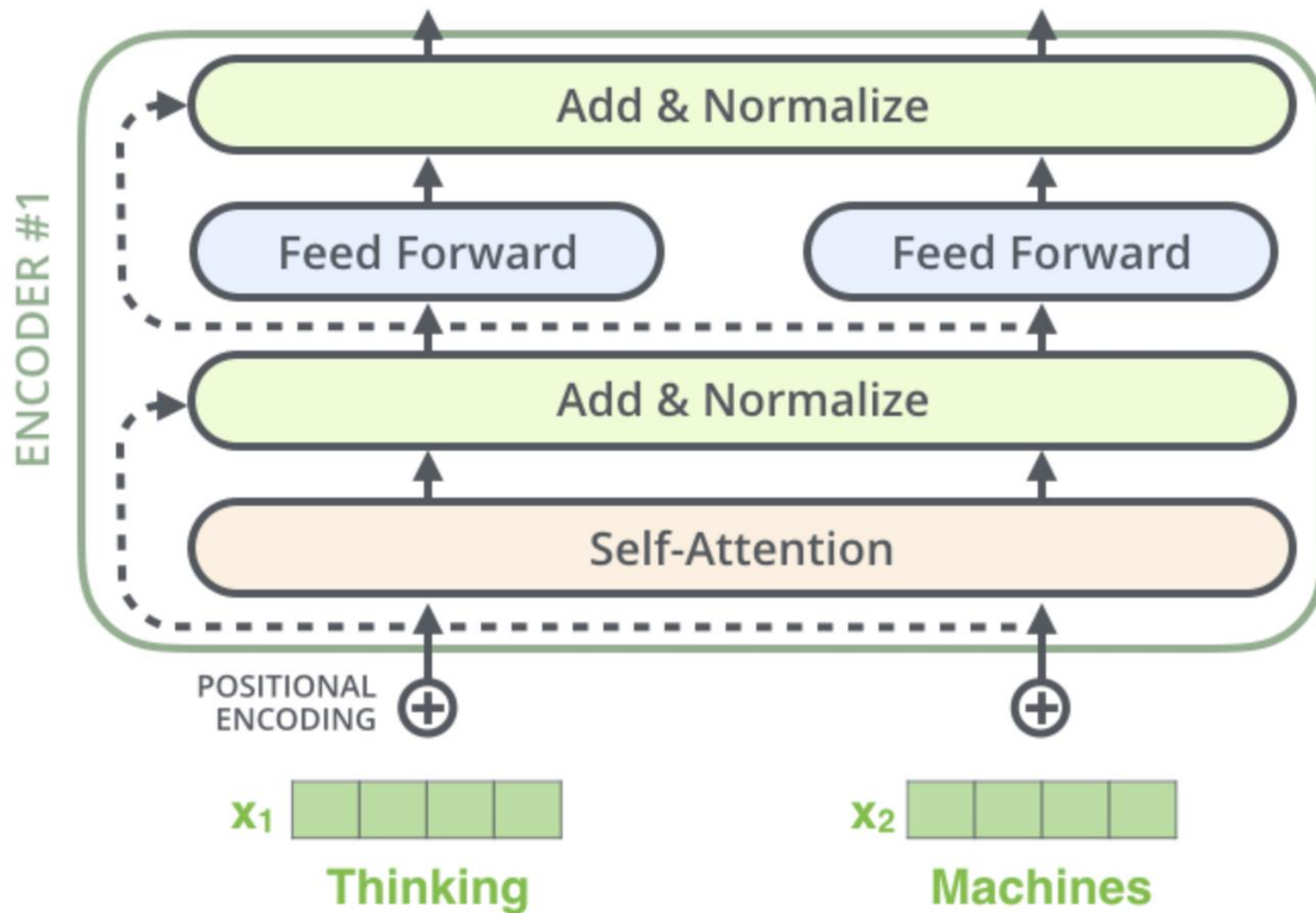
Positional Encodings

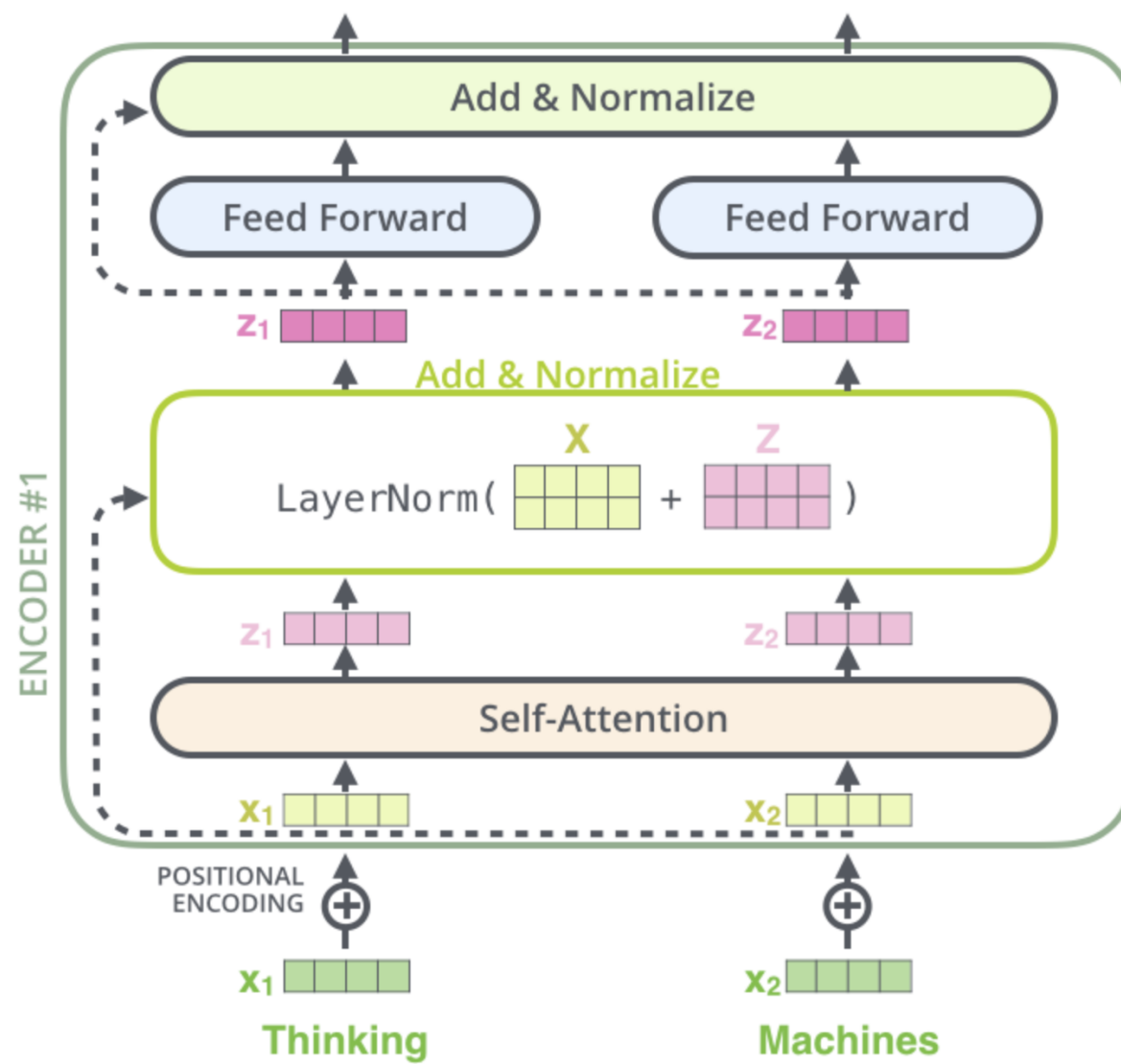
- **Still missing:** accounting for the order of the words in the input sequence
- Solution: Add a **vector** to each input embedding



Sinusoidal encodings = “position $\rightarrow$ vector of sine/cosine waves at multiple frequencies.”

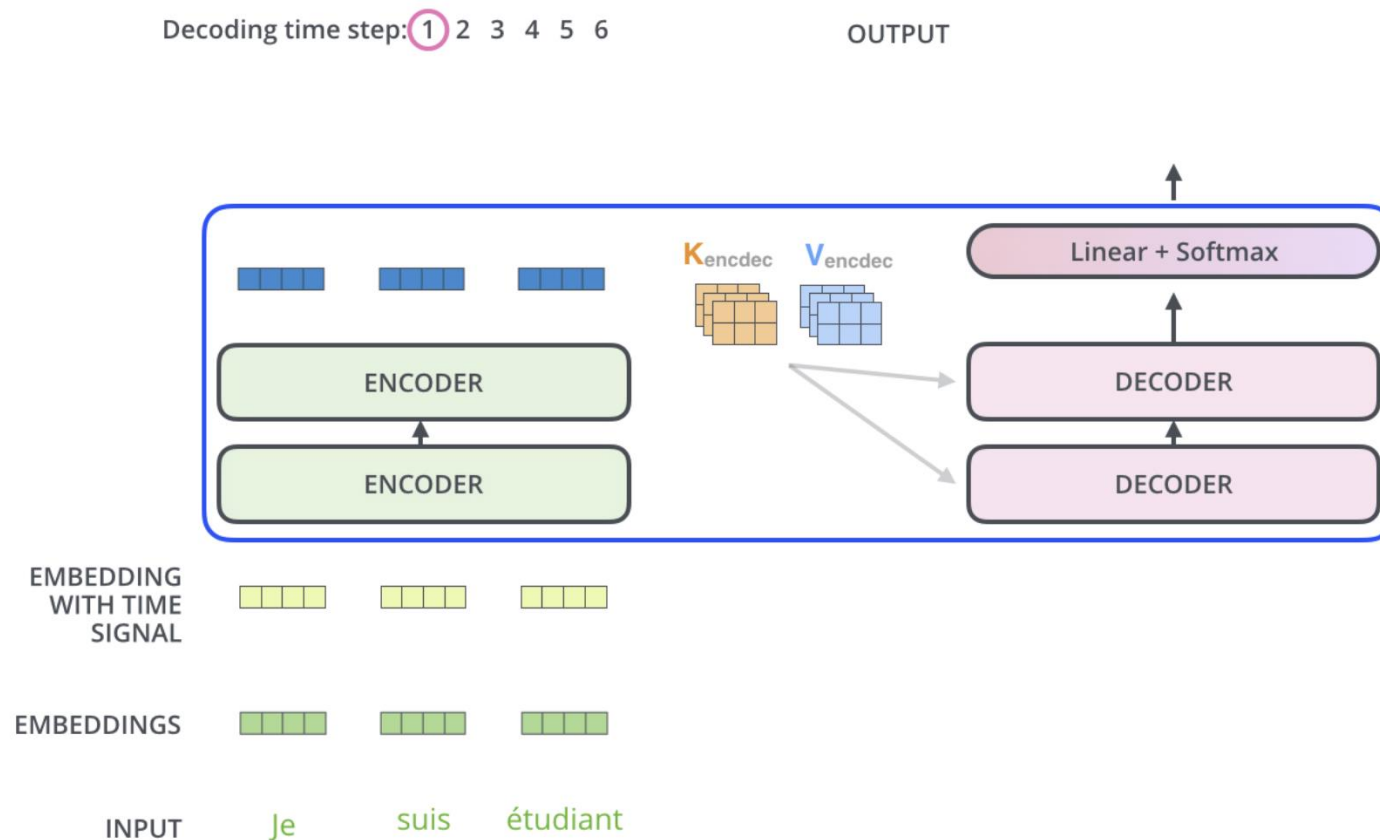
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$





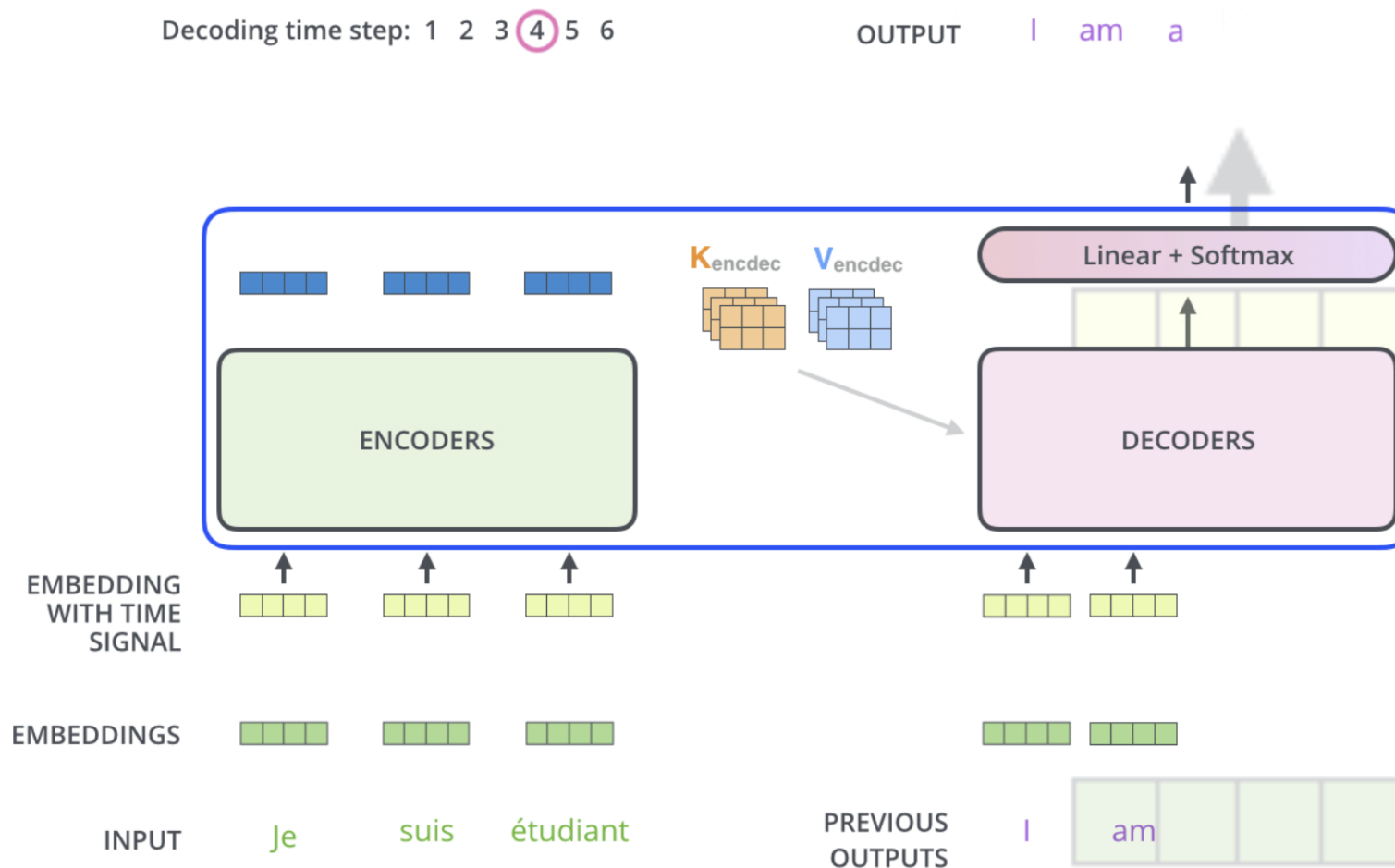
Decoder-Side

Encoder-Decoder Relation



The encoder start by processing the input sequence. The output of the top encoder is then transformed into a set of **attention vectors K and V** . These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence:

Decoder Outputs



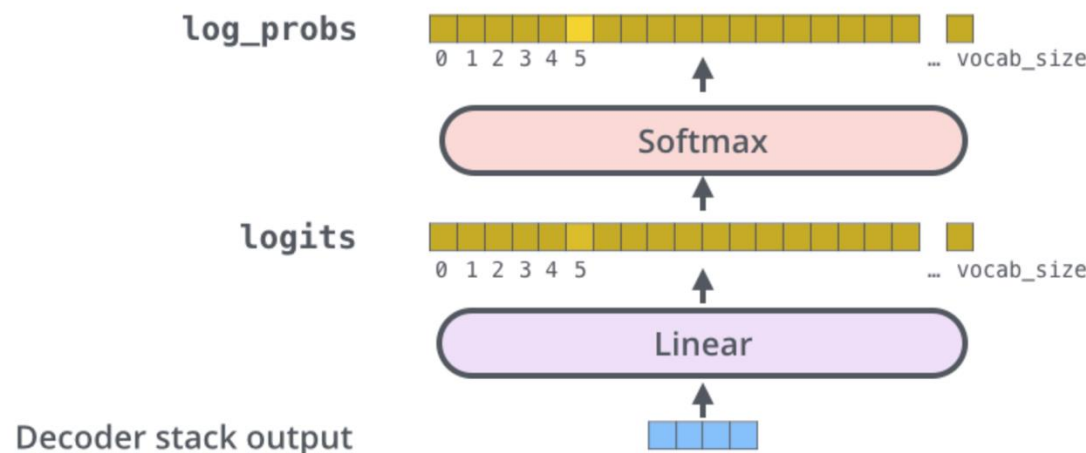
The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output. The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did. And just like we did with the encoder inputs, we embed and add positional encoding to those decoder inputs to indicate the position of each word.

Self Attention in Decoder Layer

- Only allowed to attend to **earlier positions** in the output sequence.
- This is done by masking future positions (setting them to $-\infty$) before the softmax step in the self-attention calculation.

The Final Linear and Softmax Layer

- Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector.
- The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0).



Performance Analysis



Setup

5 Training

This section describes the training regime for our models.

5.1 Training Data and Batching

We trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs. Sentences were encoded using byte-pair encoding [3], which has a shared source-target vocabulary of about 37000 tokens. For English-French, we used the significantly larger WMT 2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece vocabulary [38]. Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

5.2 Hardware and Schedule

We trained our models on one machine with 8 NVIDIA P100 GPUs. For our base models using the hyperparameters described throughout the paper, each training step took about 0.4 seconds. We trained the base models for a total of 100,000 steps or 12 hours. For our big models,(described on the bottom line of table 3), step time was 1.0 seconds. The big models were trained for 300,000 steps (3.5 days).

Results

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention			
Recurrent			
Convolutional			
Self-Attention (restricted)			

Result: Complexity

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Self-Attention layer connects all positions with a constant number of sequentially executed operations

Results: Parallelization

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Self Attention has lower complexity when sequence length is smaller than the representation dimension

Results: Capturing Long Range Dependencies

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

The shorter these paths between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies

Results: Model Performance and Cost

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	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

Constituency Parsing

- Is the task of analyzing a sentence and breaking it down into constituents
- Constituents: a word or a group of words that act together as a phrase
 - The cat (noun phrase)
 - At the cake (verb phrase)

Results

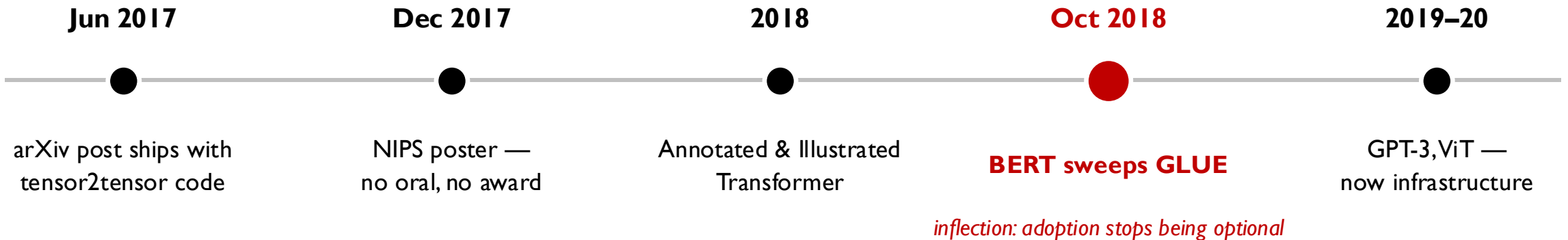
Parser	Training	WSJ 23 F1
Vinyals & Kaiser et al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser et al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

Ablation Study

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)		positional embedding instead of sinusoids								4.92	25.7	
big	6	1024	4096	16			0.3		300K	4.33	26.4	213

Why the paper took off

Vaswani et al., 2017 — reception, diffusion, and the gap between the paper and the code



1 Code, not just the paper

tensor2tensor was released alongside the arXiv post, with the authors' own configs. Code artifacts can be huge adoption enablers!

2 A field already primed

SOTA on WMT14 at a fraction of prior training cost, landing mid-race to drop recurrence (ConvS2S, one month earlier). Read as the sharp win in that race, not a new paradigm.

3 Transfer, not translation

ELMo → GPT-1 → BERT. The GLUE sweep forced adoption; GPT-3 and ViT confirmed the generality. The authors say they did not foresee the scaling story.

Reproducibility caveat: the paper describes post-LN while tensor2tensor configs used pre-LN — and warmup turned out to be load-bearing for post-LN stability, a lesson rediscovered painfully through 2019–20.

Learning Resources

- The Illustrated Transformer
 - <http://jalammar.github.io/illustrated-transformer/>
- The Annotated Transformer
 - <http://nlp.seas.harvard.edu/2018/04/03/attention.html>
- From GPT-2 to gpt-oss: Analyzing the Architectural Advances
 - <https://magazine.sebastianraschka.com/p/from-gpt-2-to-gpt-oss-analyzing-the>

Summary

- Introduced the **Transformer** architecture
- Eliminated recurrence & convolutions, relies only on self-attention
- Enabled massive parallelization → faster training
- Synthesis of Multiple Ideas
 - **Multi-Head Self-Attention**: captures diverse contextual relationships
 - **Positional Encodings**: inject word order information
 - **Encoder–Decoder Stacks**: layered attention blocks for sequence-to-sequence tasks
- **Impact**
 - Achieved state-of-the-art in translation (WMT 2014)
 - Foundation for GPT, and modern LLMs