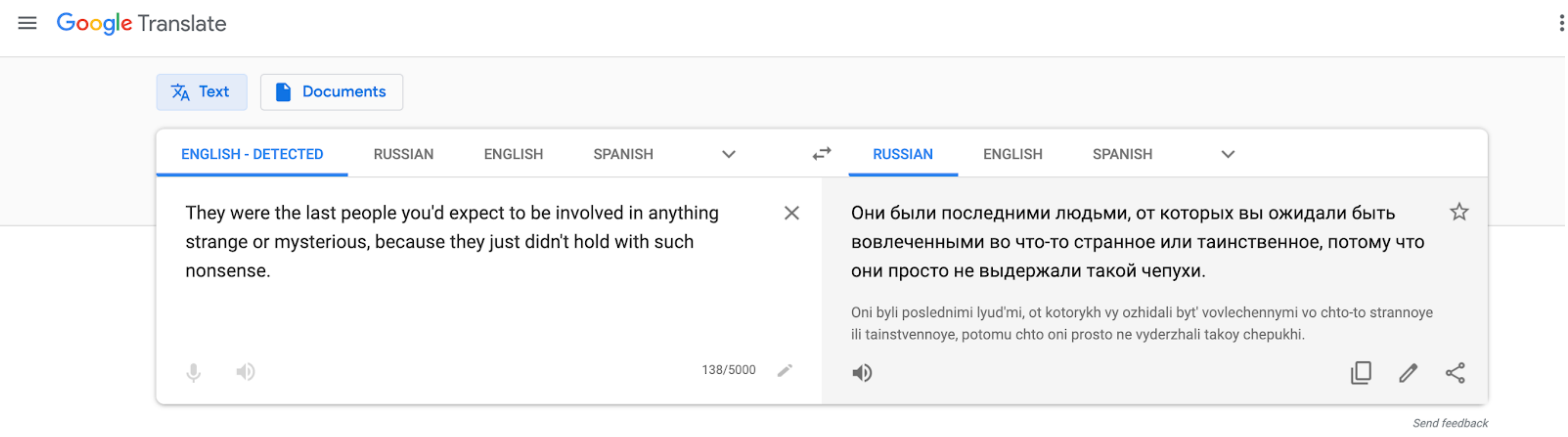
Глубинное обучение

Обработка естественного языка: Attention, Transformer

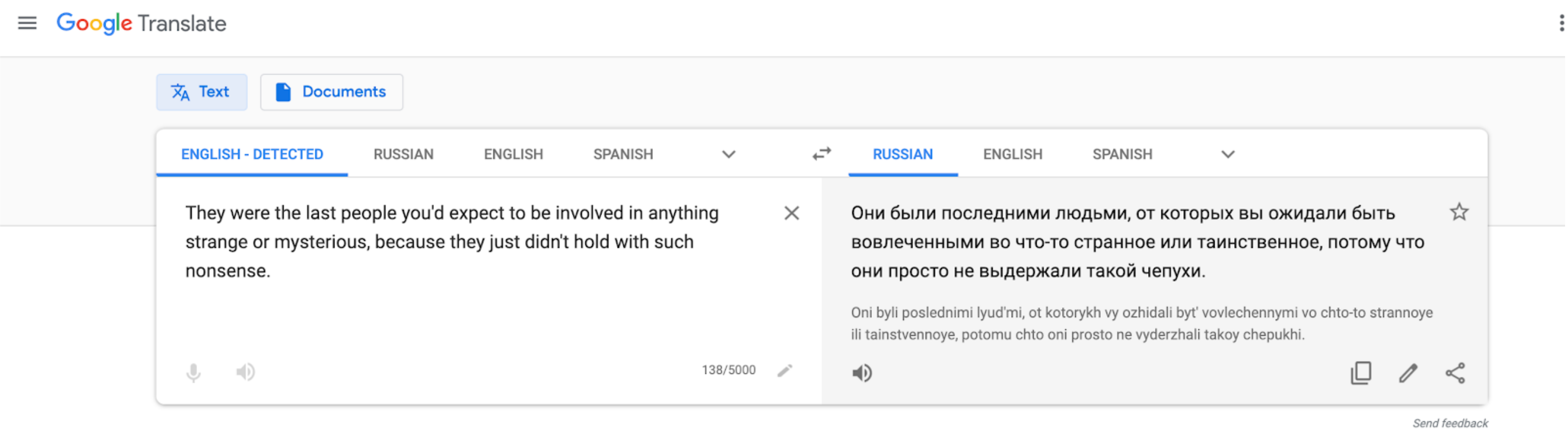
Михаил Лазарев

Sequence-to-sequence model

Приложения: перевод



Приложения: перевод



Авторегрессионная модель $$p(y\,|\,\underline{x}) = = \prod_{i=1}^{n} p(y_i\,|\,y_1, \ldots, y_{i-1}, \underline{x})$$

x - текст на исходном языке (source)

y - перевод текста на другой язык (target)

Sequence-to-sequence model

Последний hidden state - представление всего исходного текста

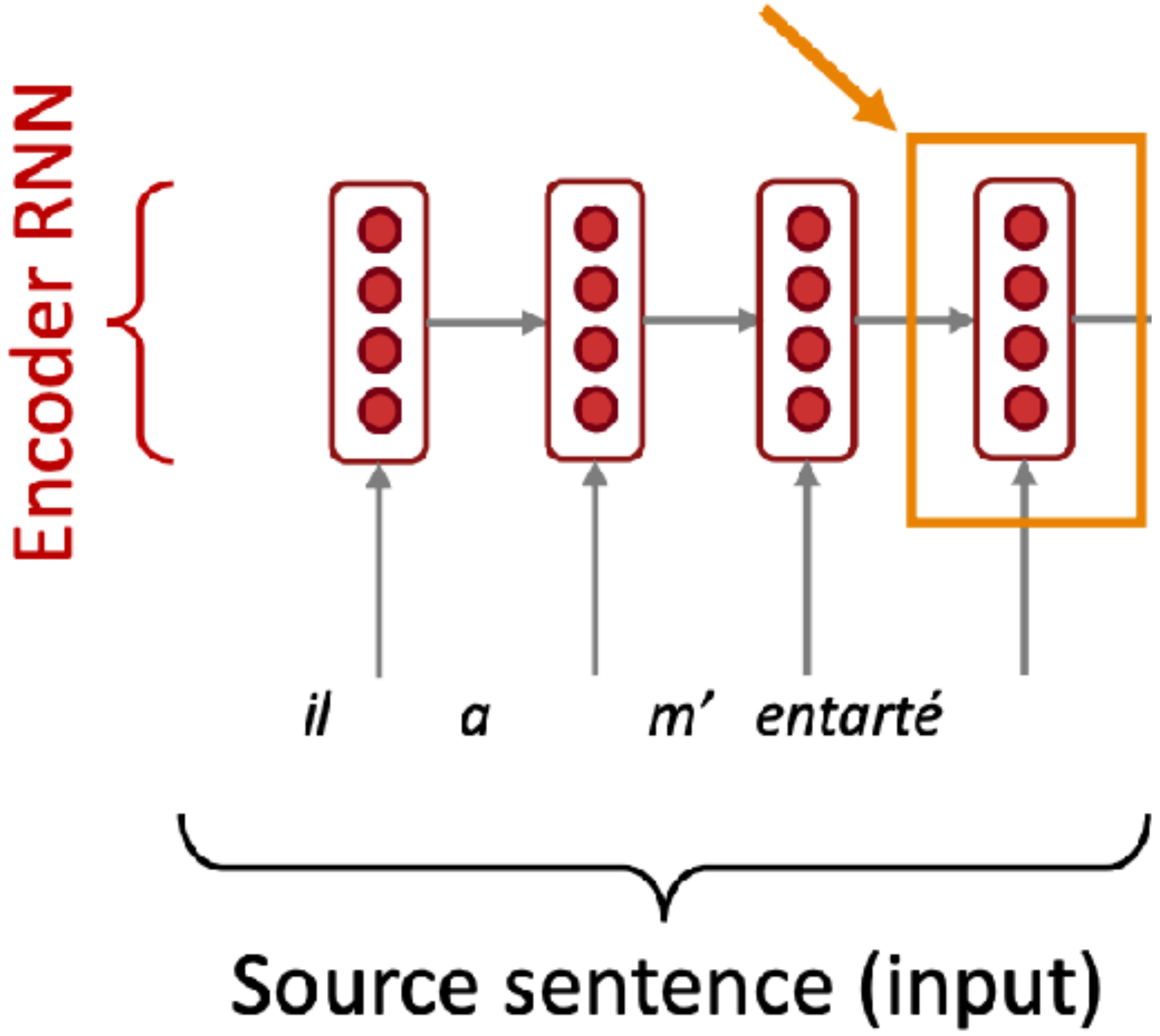


Image credit

Sequence-to-sequence model

Используем как h^0
для языковой модели
на другом языке

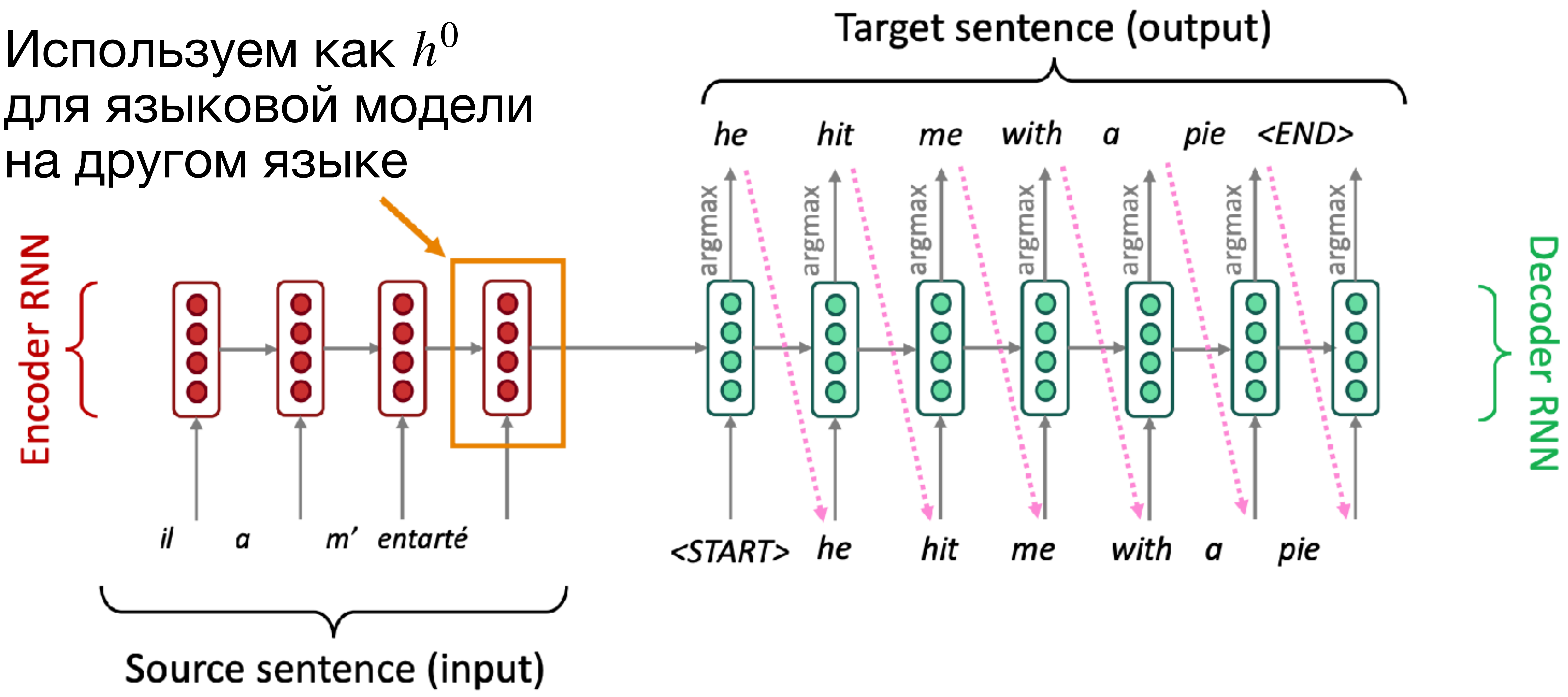


Image credit

Sequence-to-sequence model

Обучение: нужен параллельный корпус

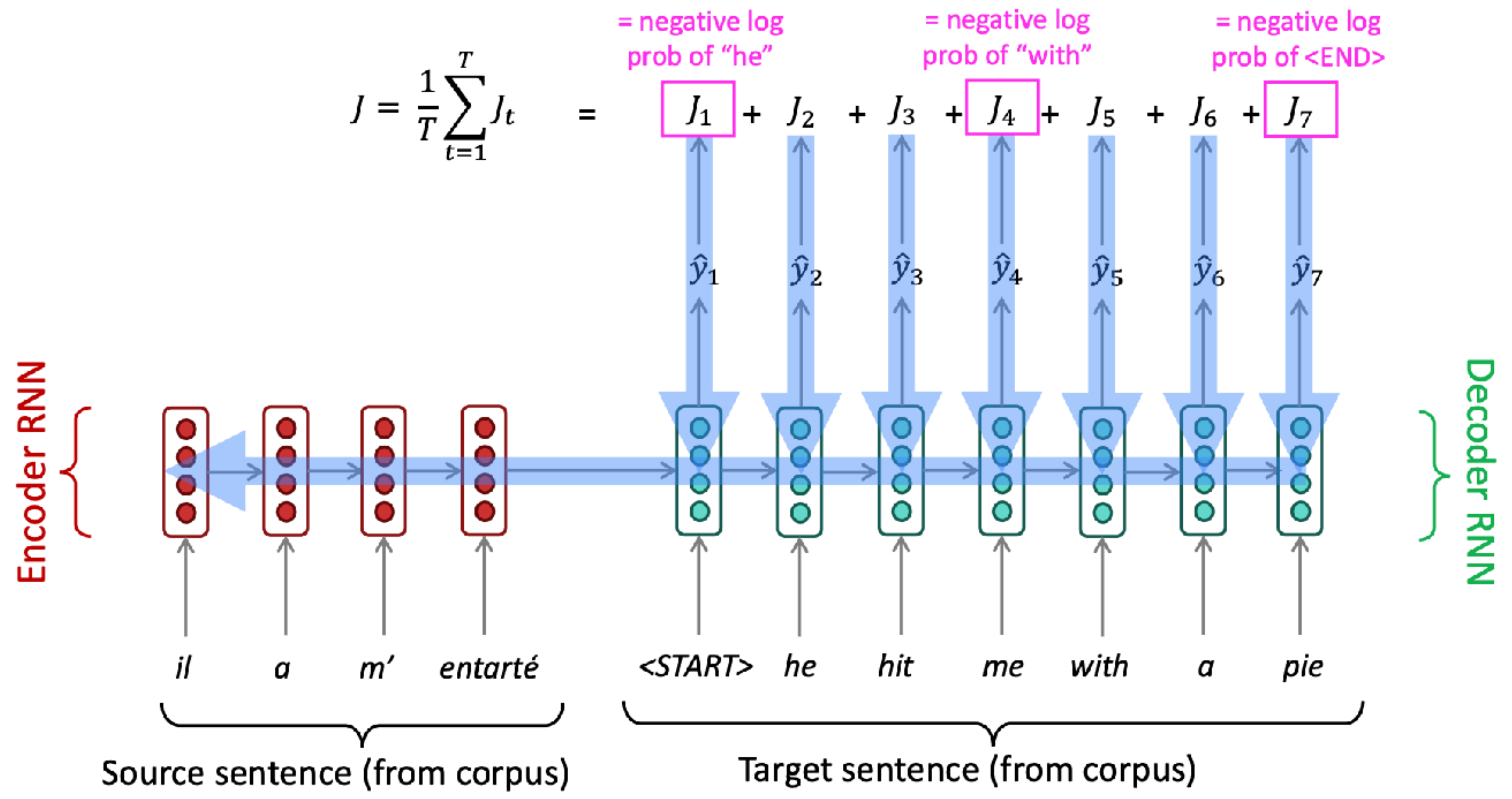
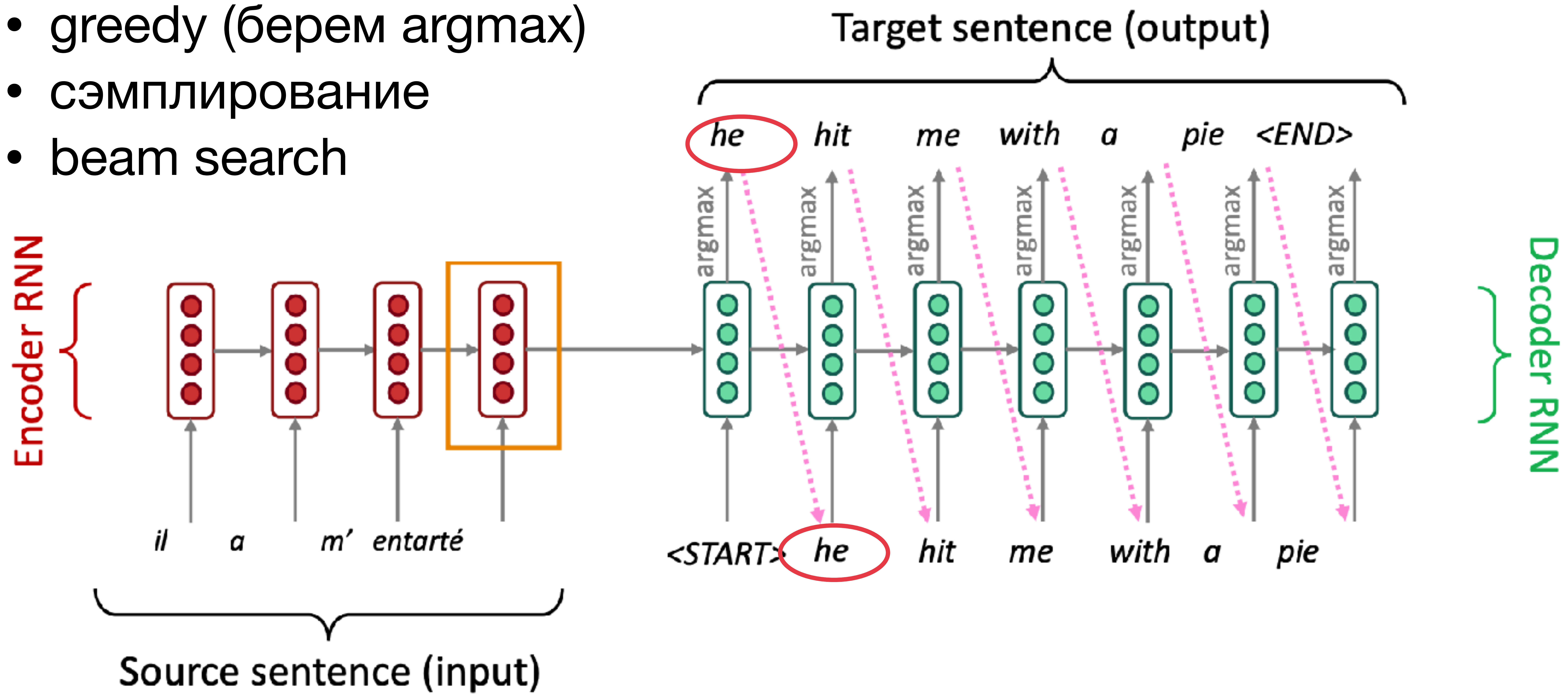


Image credit

Sequence-to-sequence model

Генерация:

- greedy (берем argmax)
- сэмплирование
- beam search



[Image credit](#)

Sequence-to-sequence model

Informational bottleneck:
В одном векторе должна быть вся информация

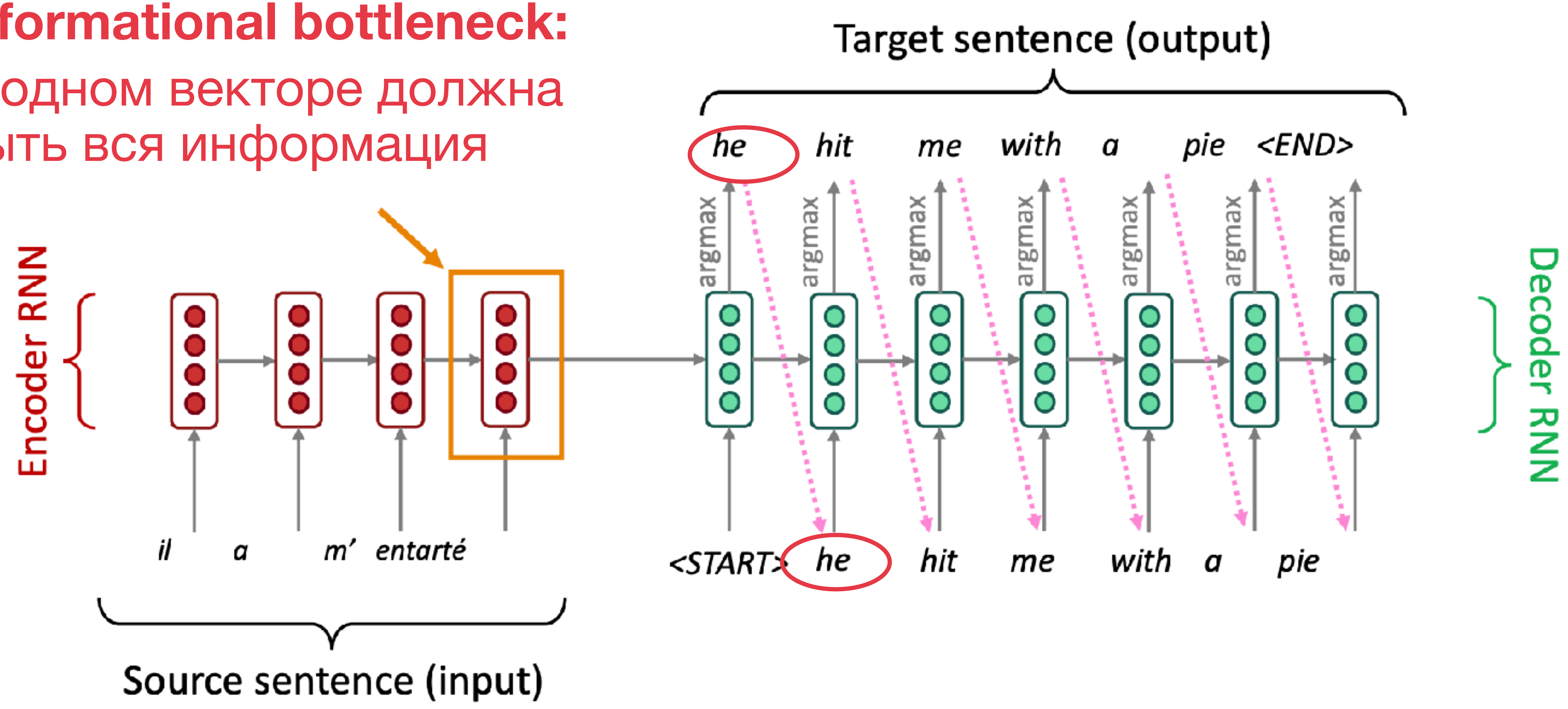


Image credit

Attention

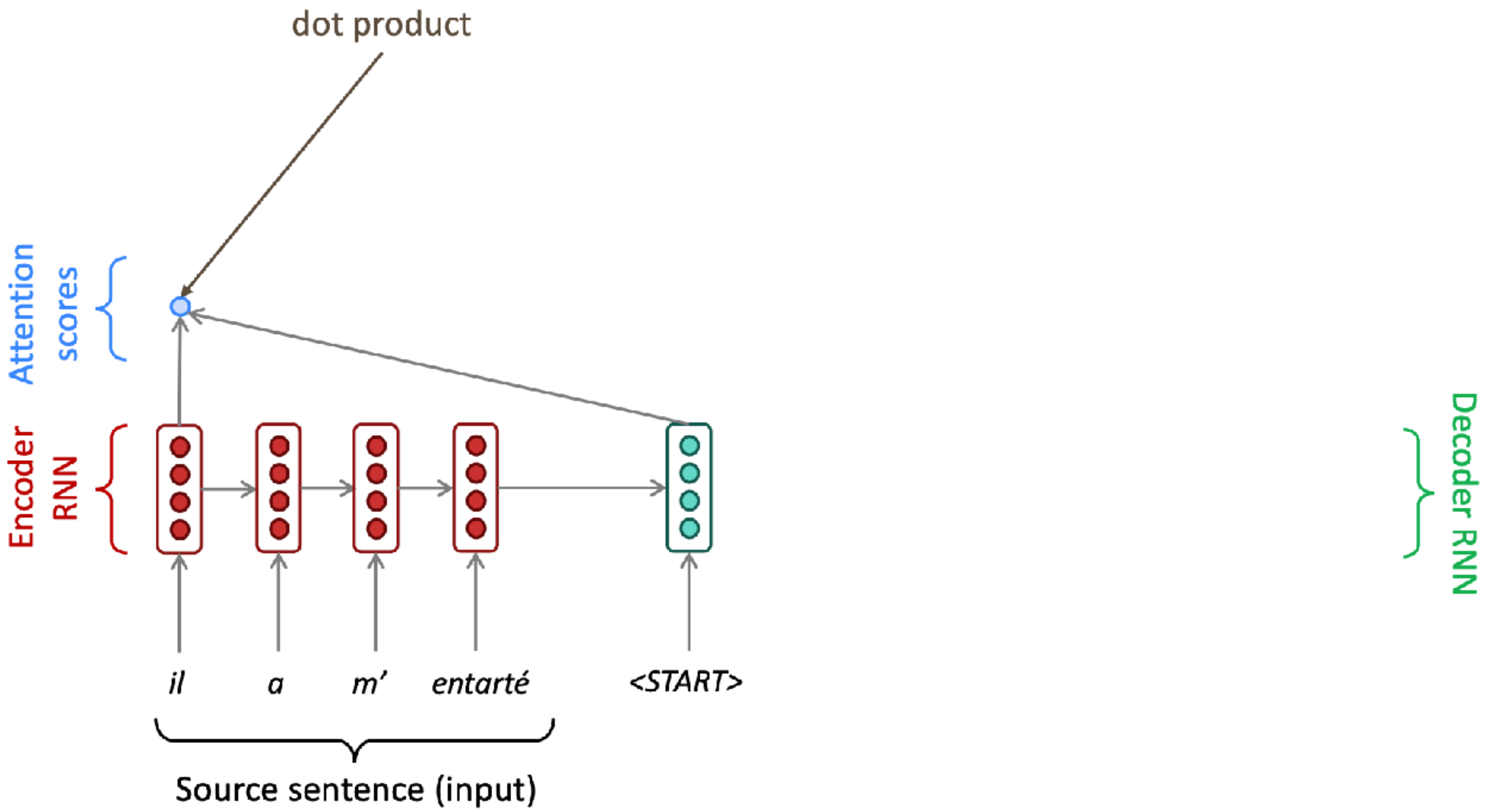


Image credit

Attention

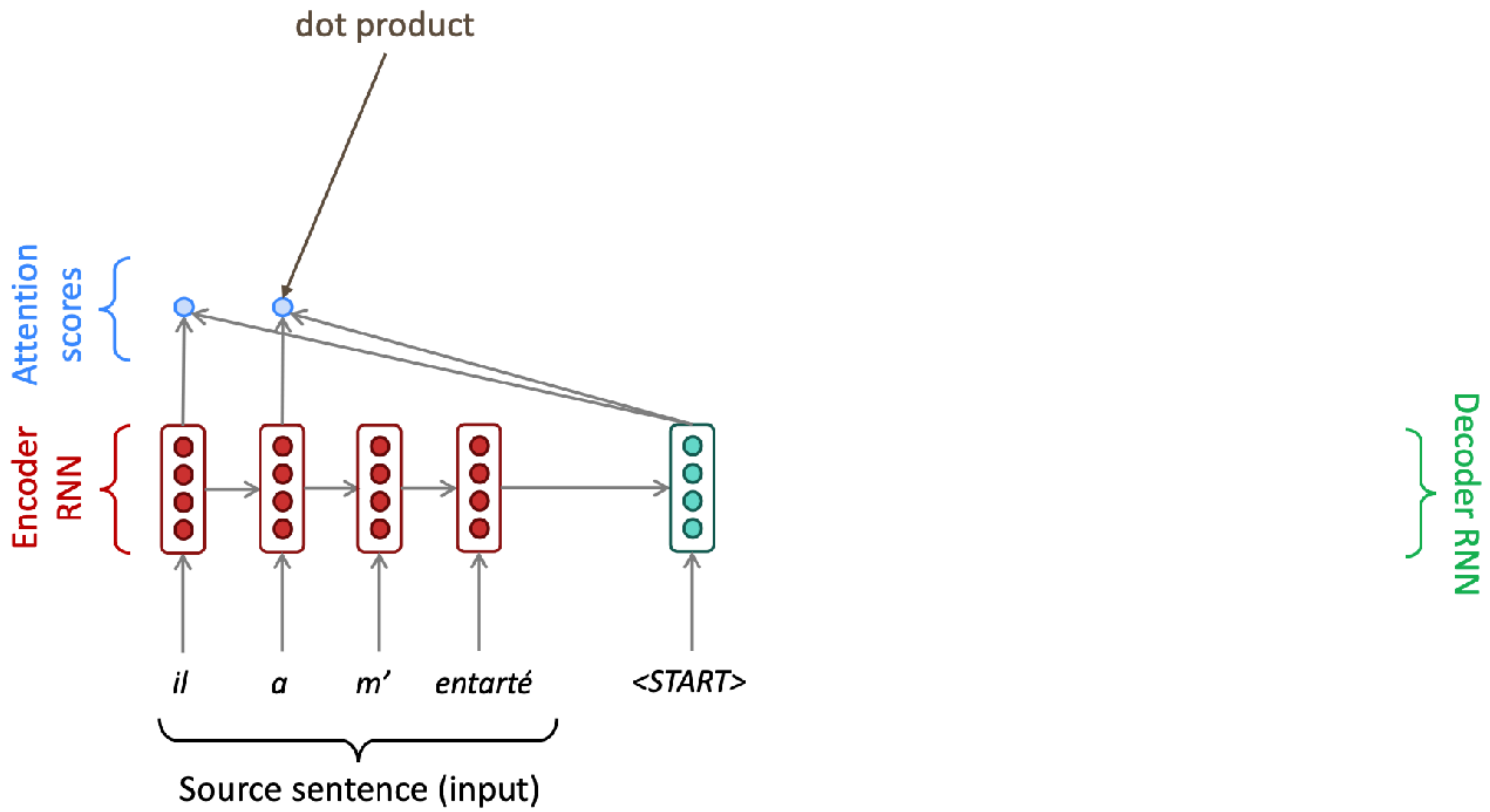


Image credit

Attention

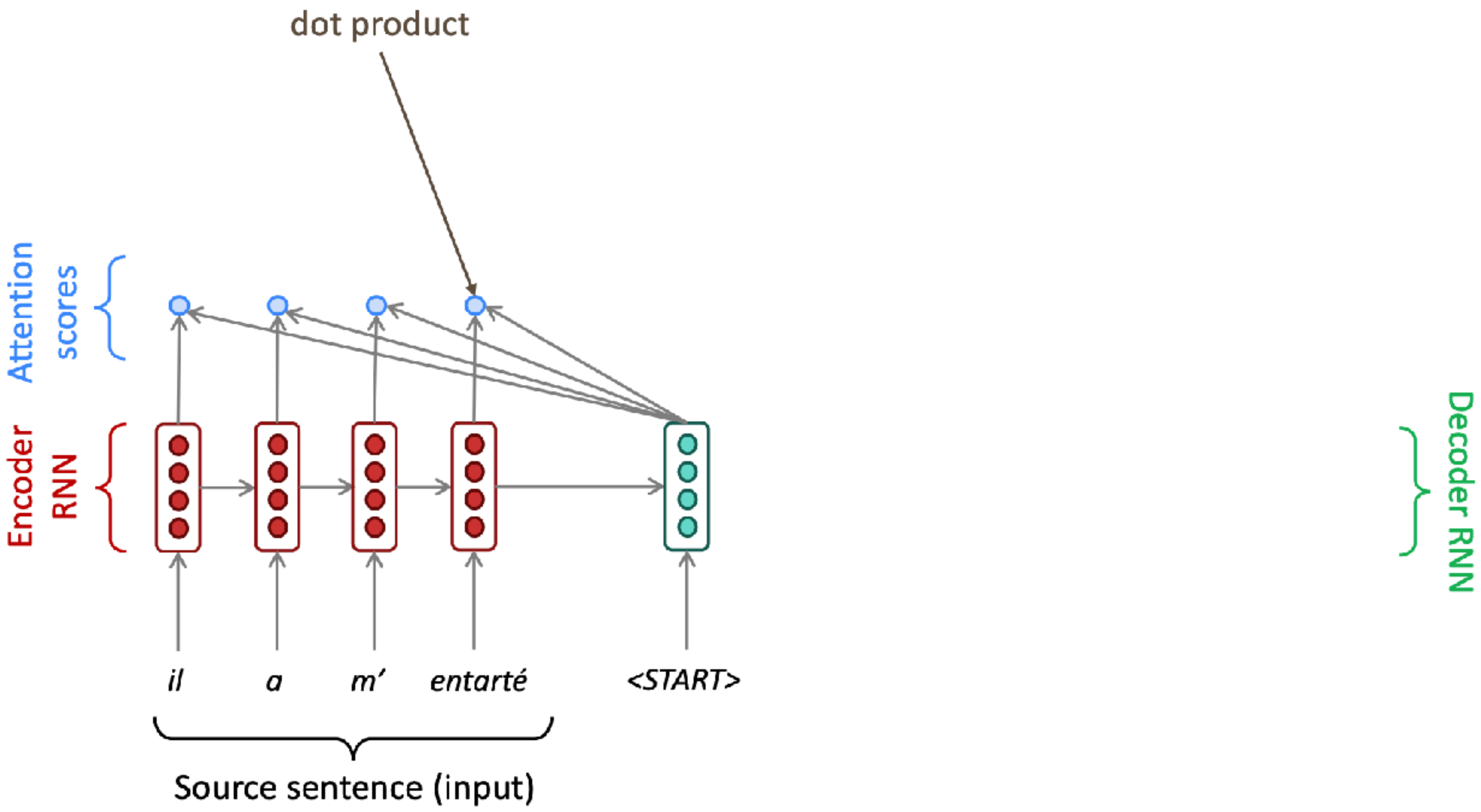


Image credit

Attention

Для первого слова в переводе
больше всего смотрим на *il*

Softmax

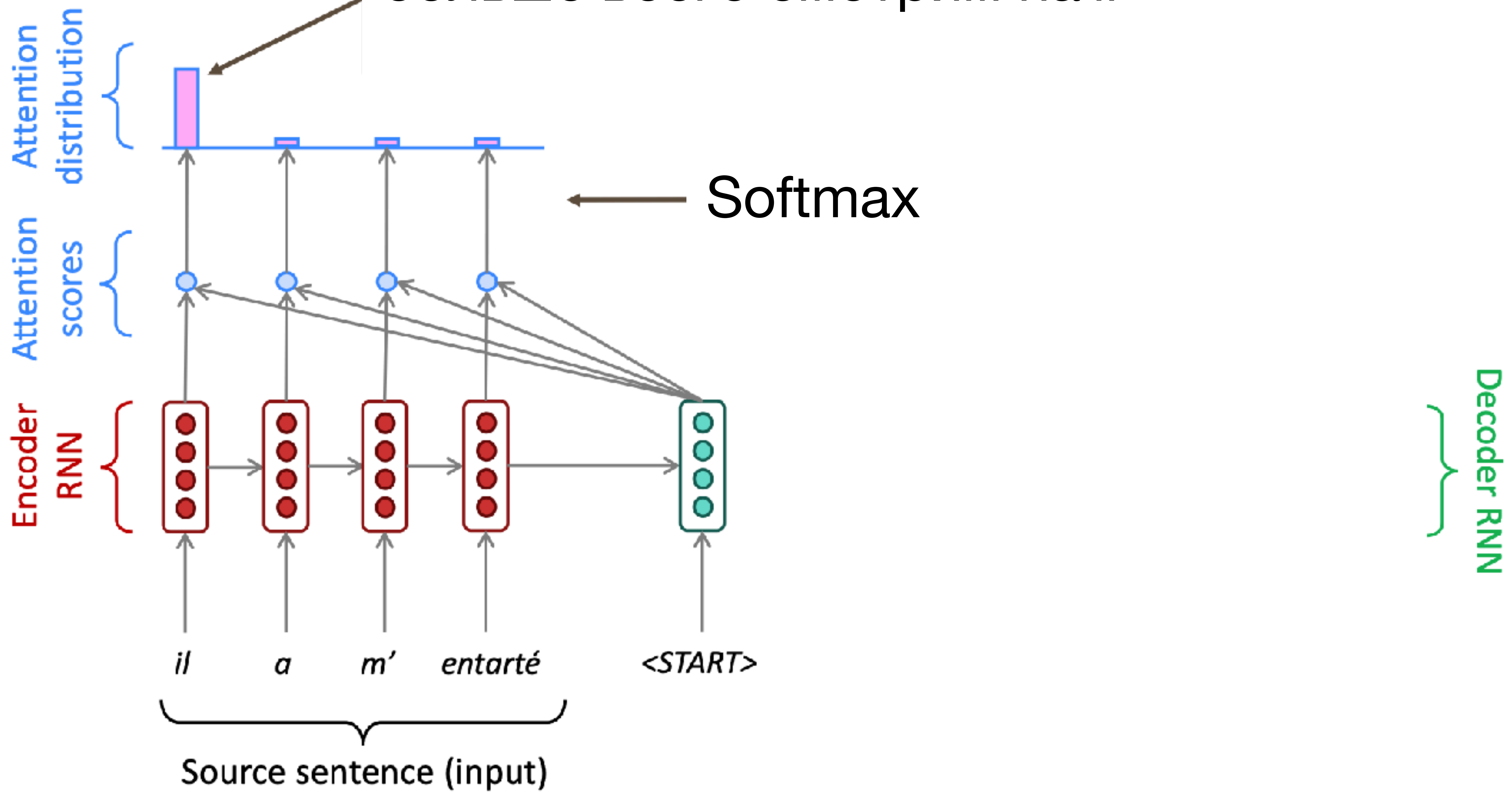
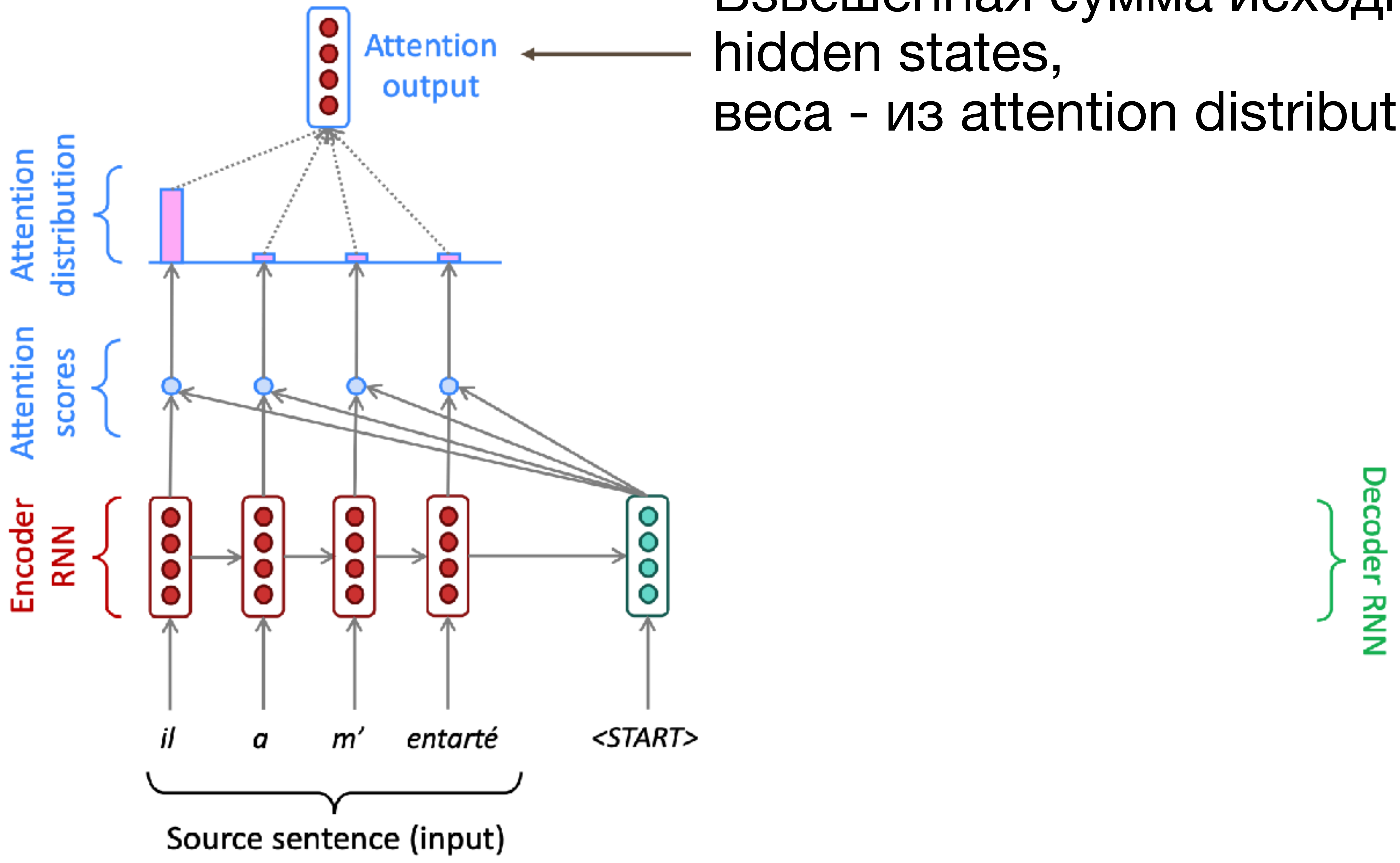


Image credit

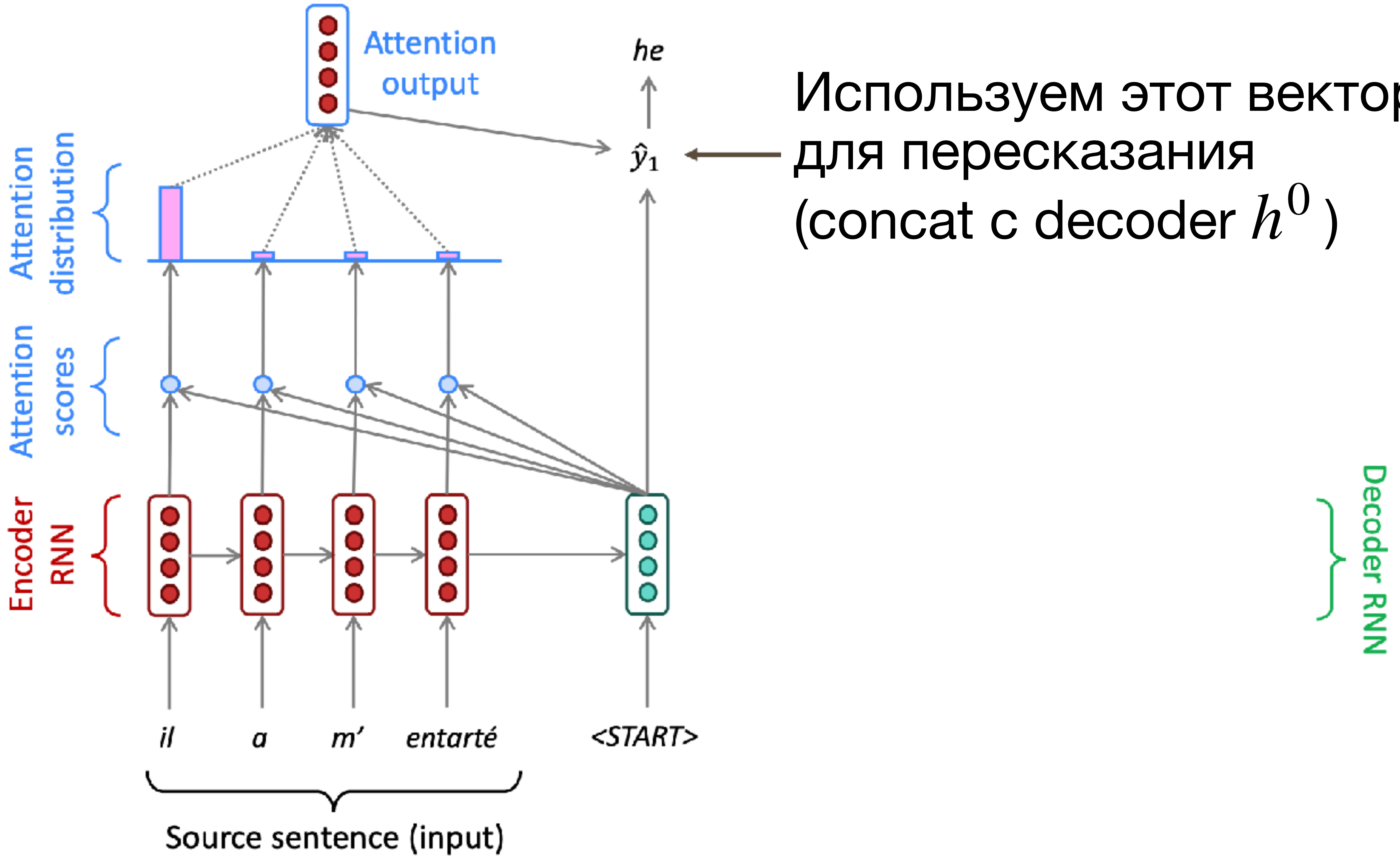
Attention



Взвешенная сумма исходных hidden states,
веса - из attention distribution

Image credit

Attention



Используем этот вектор для пересказания (concat с decoder h^0)

Image credit

Attention

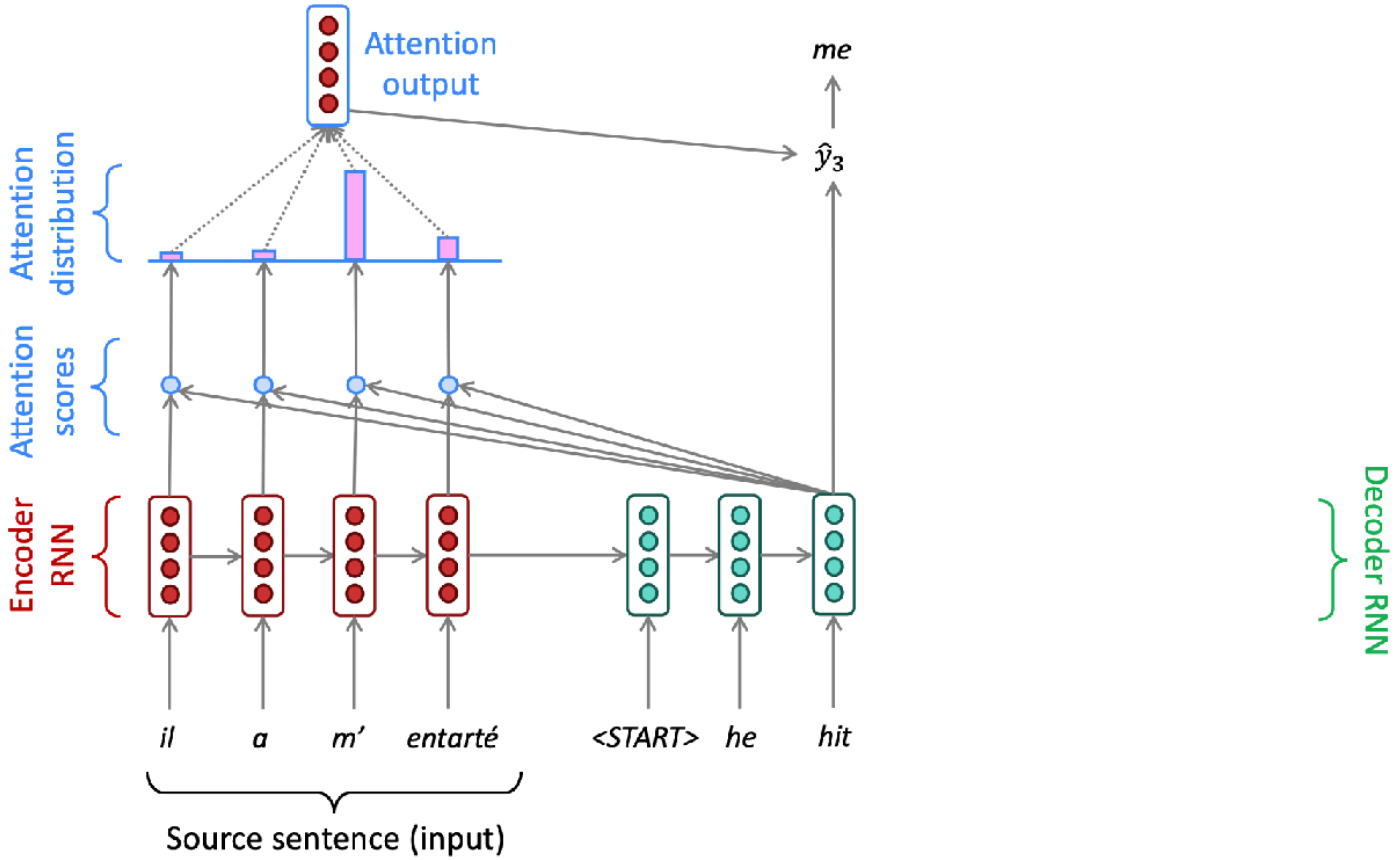


Image credit

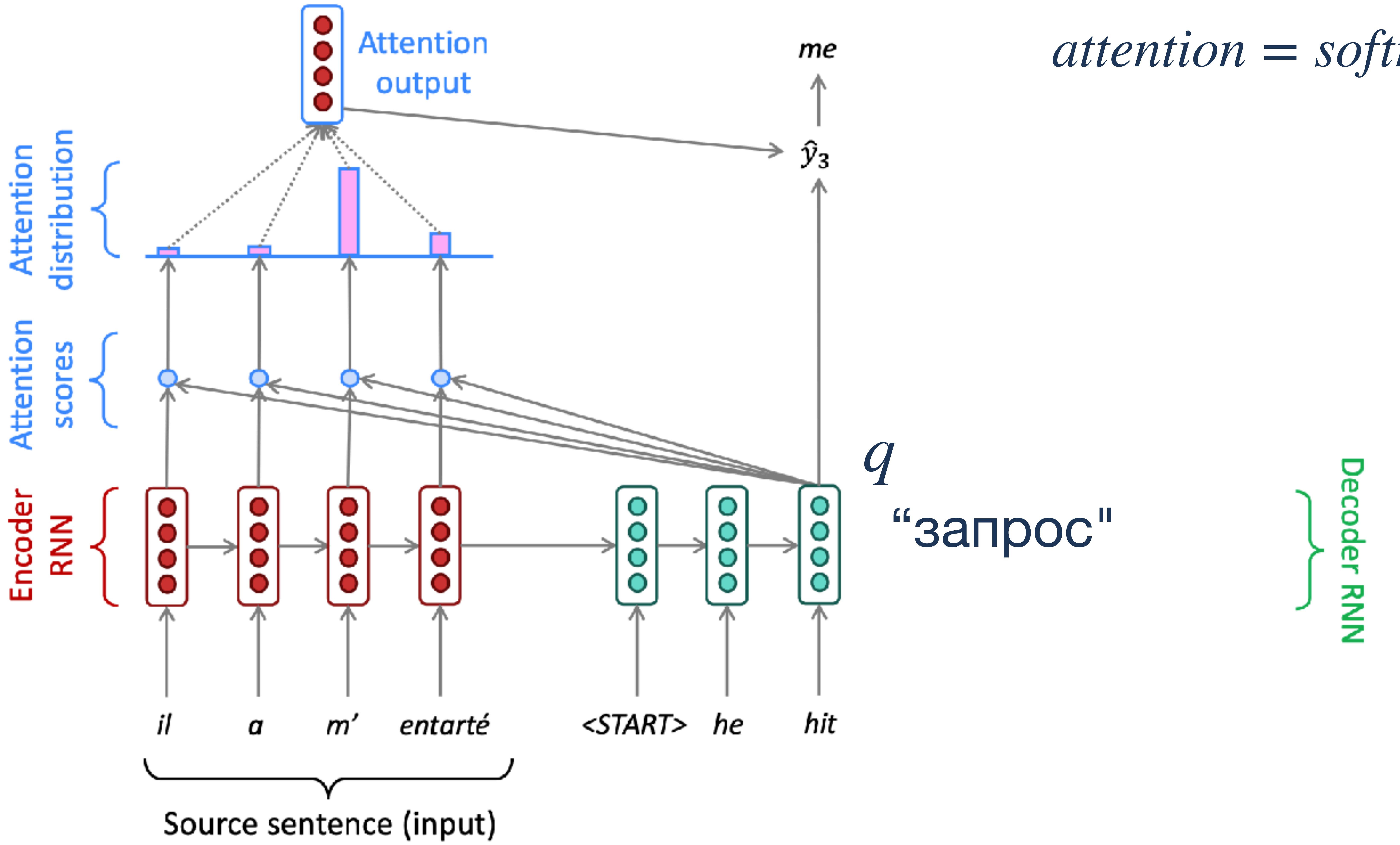
Attention

$$attention = softmax(qK^T)V$$

K, V

“ключи”

“значения”



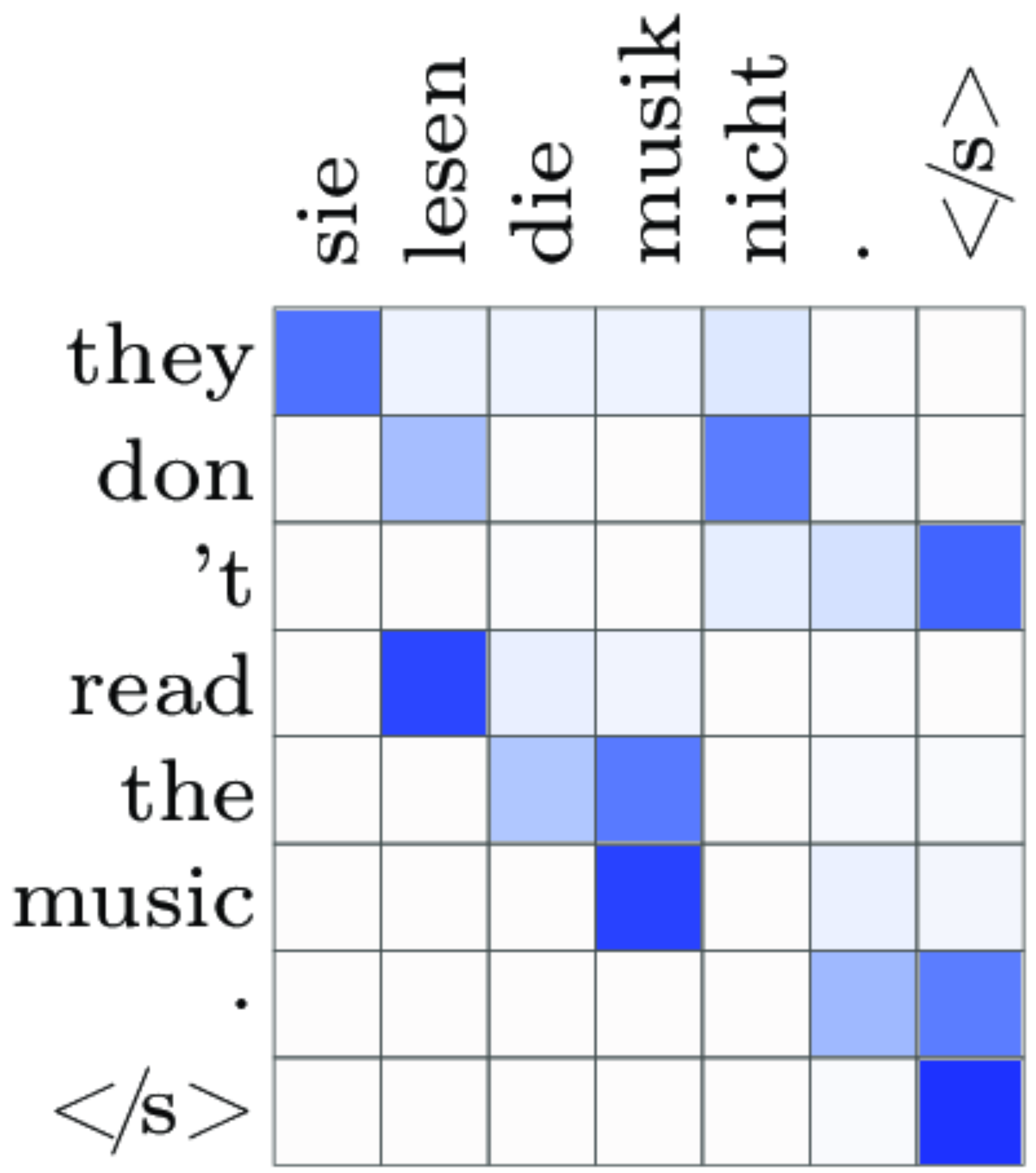
q

“запрос"

[Image credit](#)

Attention

- Убирает bottleneck, всегда лучше качество
- Дает интерпретируемость (можно визуализировать распределения)

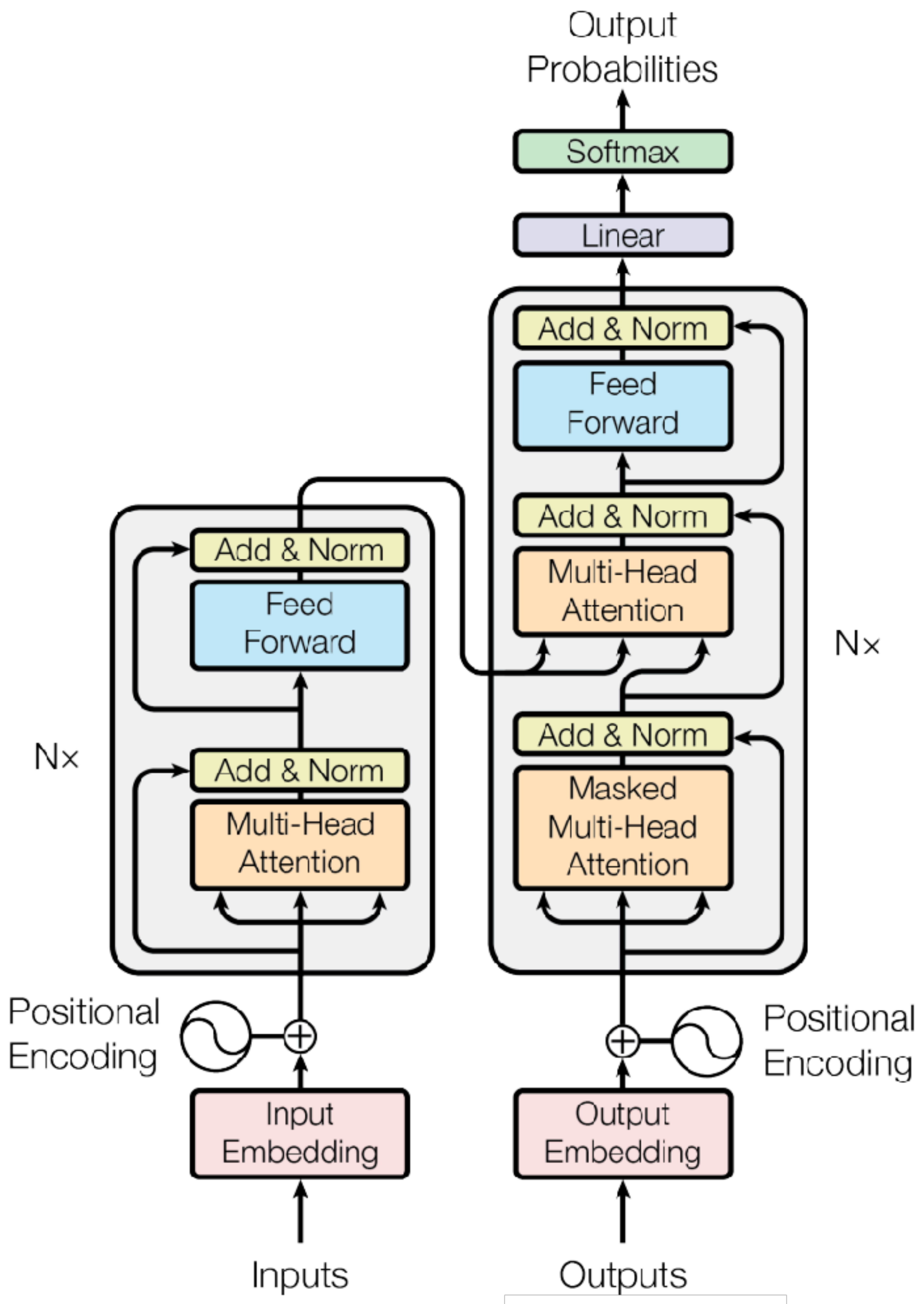


Transformer

Transformer

BERT

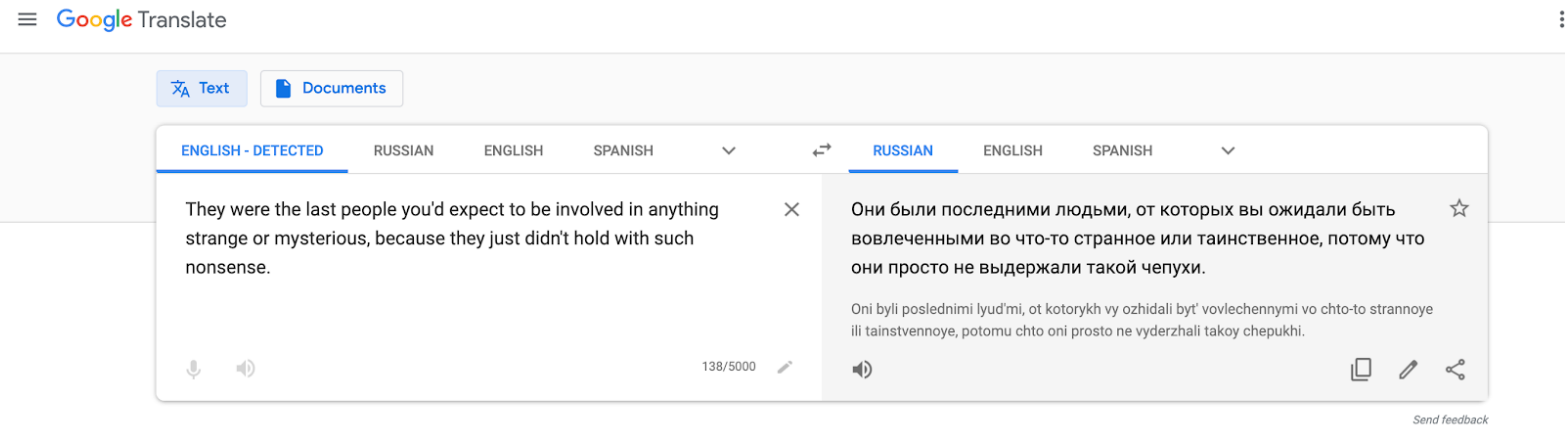
Encoder



GPT

Decoder

Transformer



Изначально предложен для перевода

Transformer

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training 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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Изначально предложен для перевода

- Лучше качество (в других задачах тоже!)
- Быстрее

Transformer

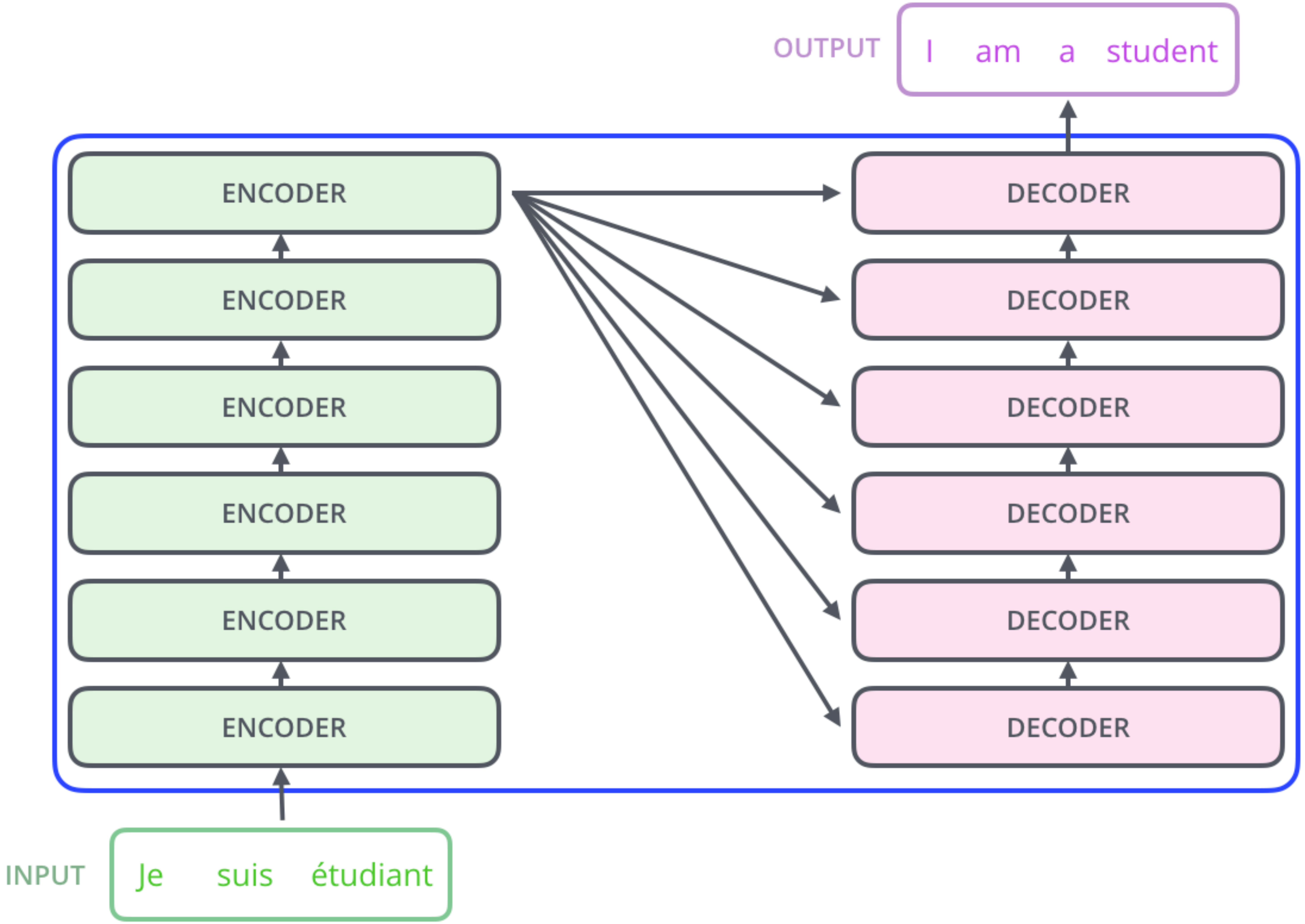


Image credit

Transformer

Encoder block/ Decoder block

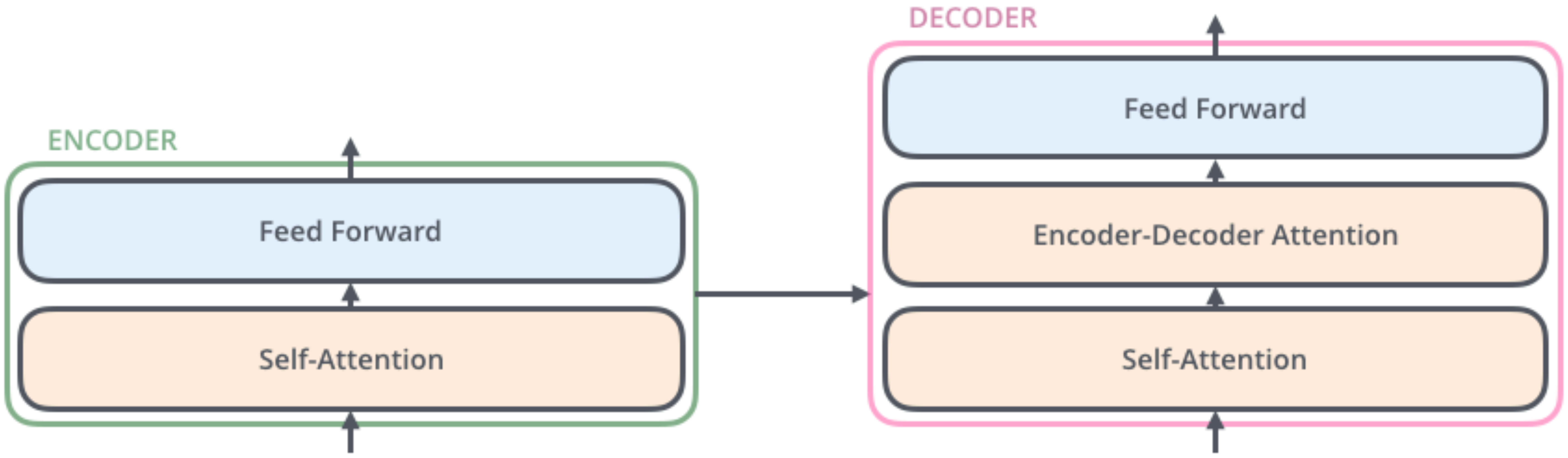


Image credit

Transformer: encoder block

Слой эмбеддингов -
для каждого входного слова достаем из таблицы вектор (обучаемый)

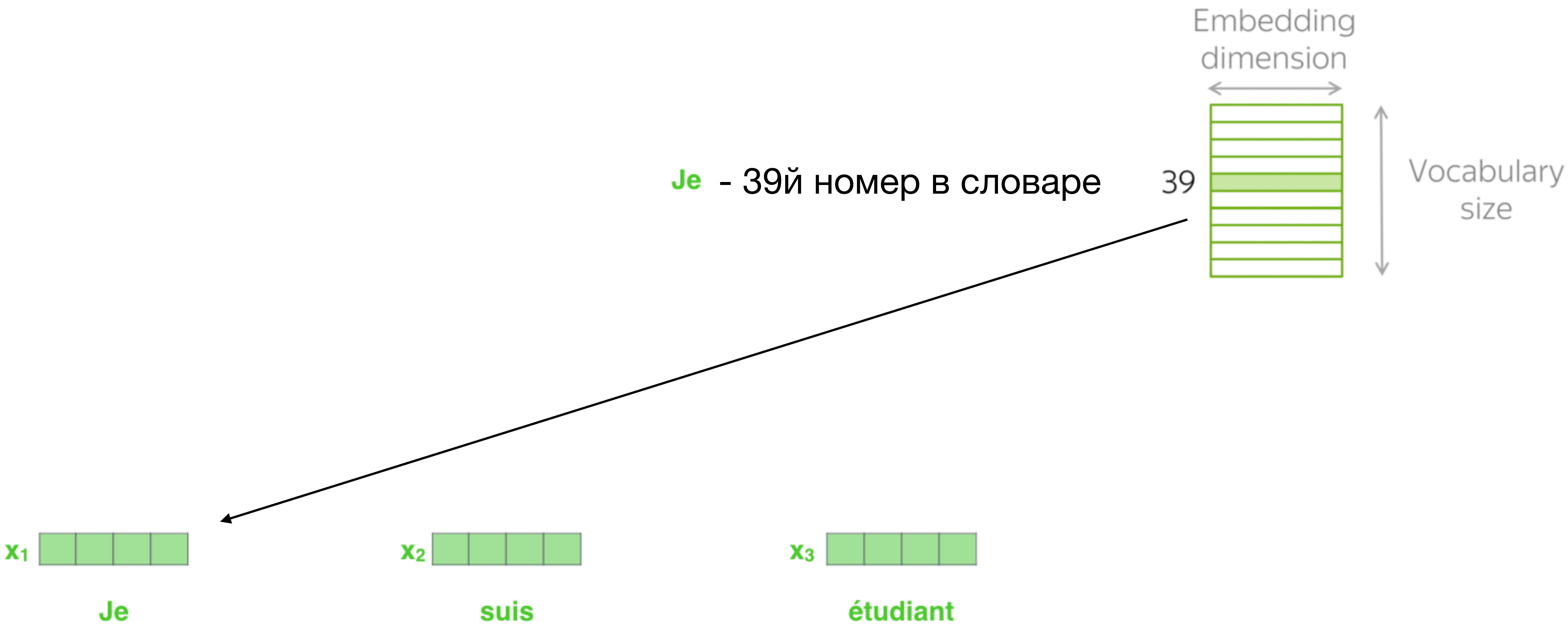


Image credit

Transformer: encoder block

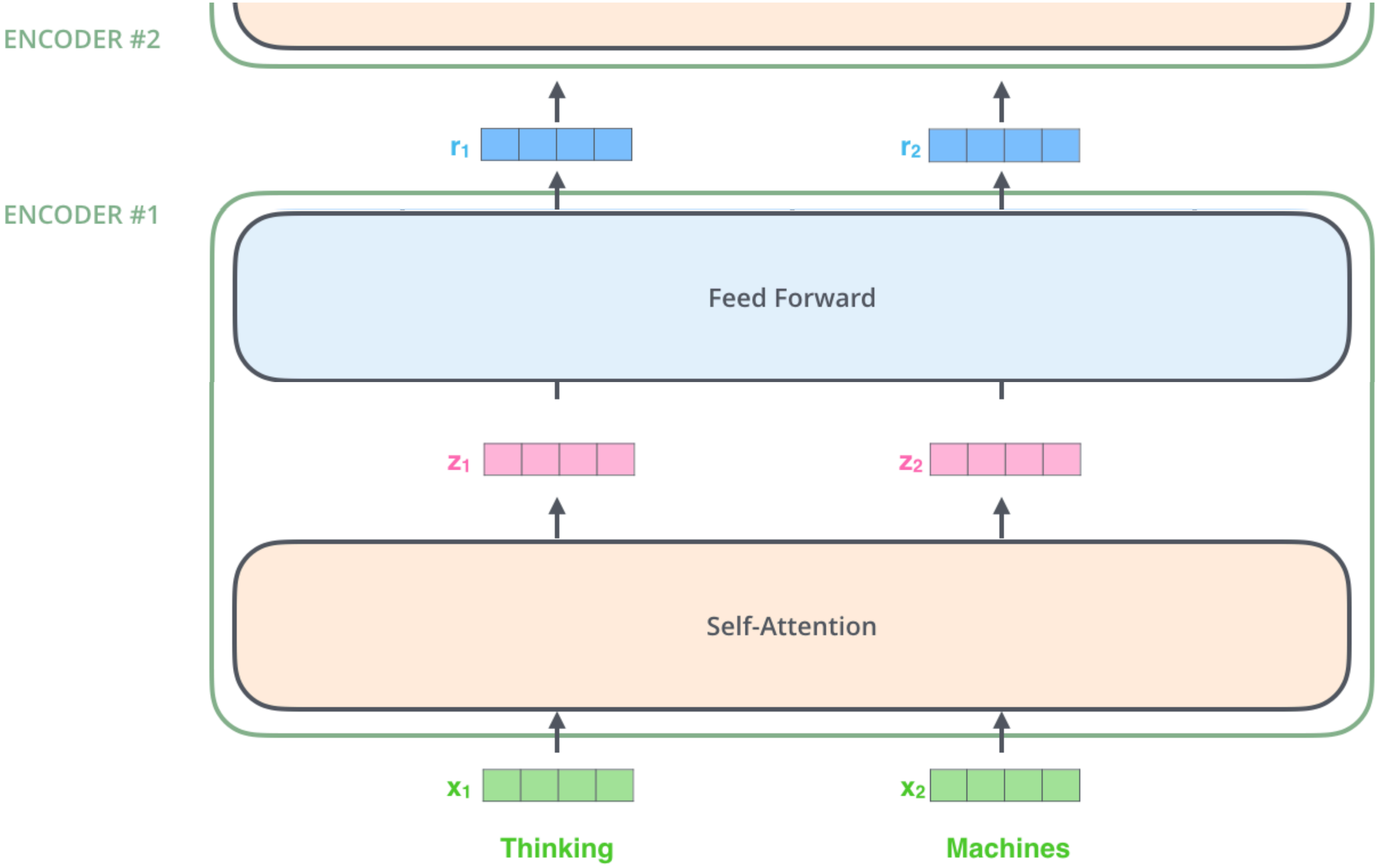


Image credit

Transformer: encoder block

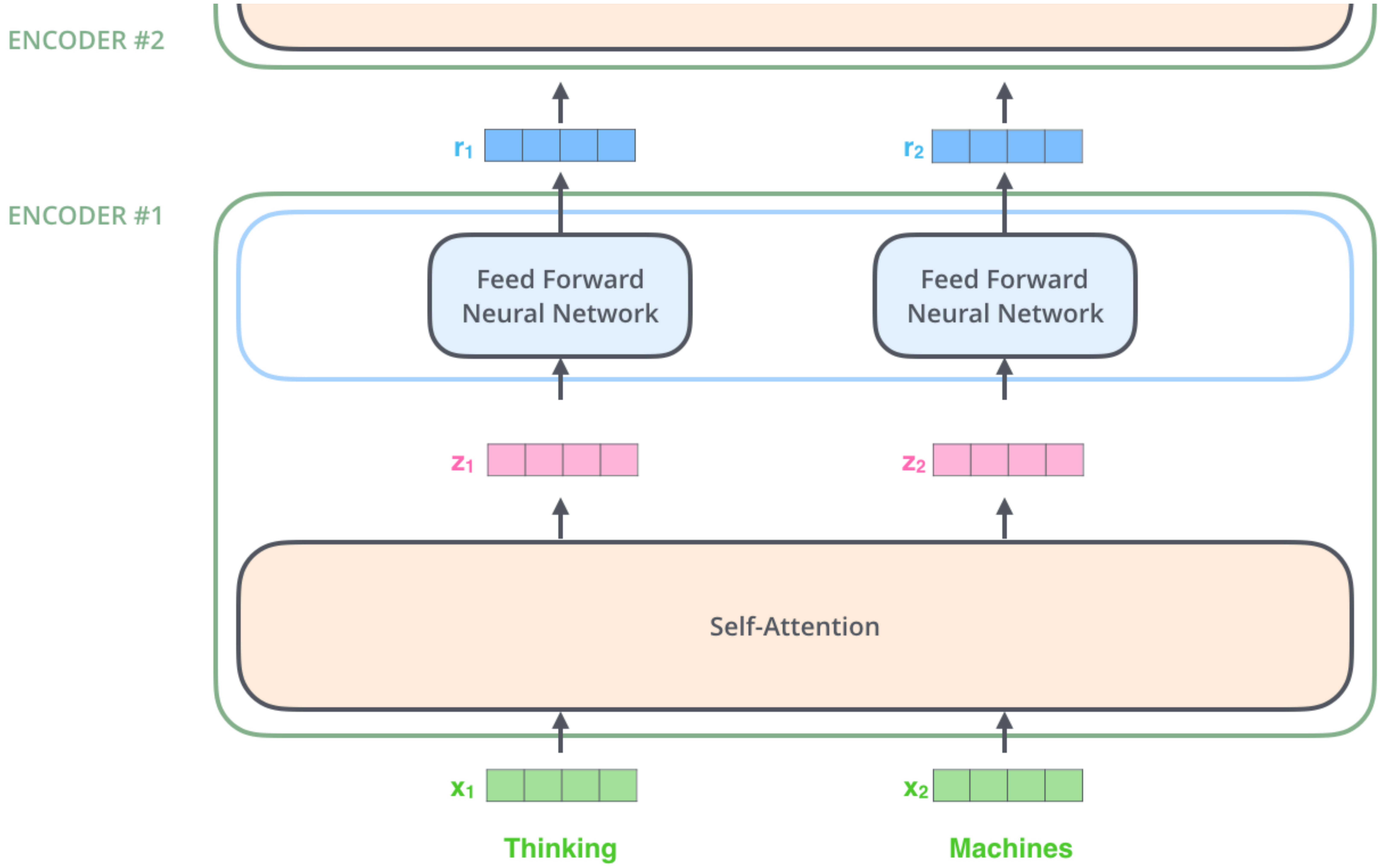


Image credit

Transformer: encoder block

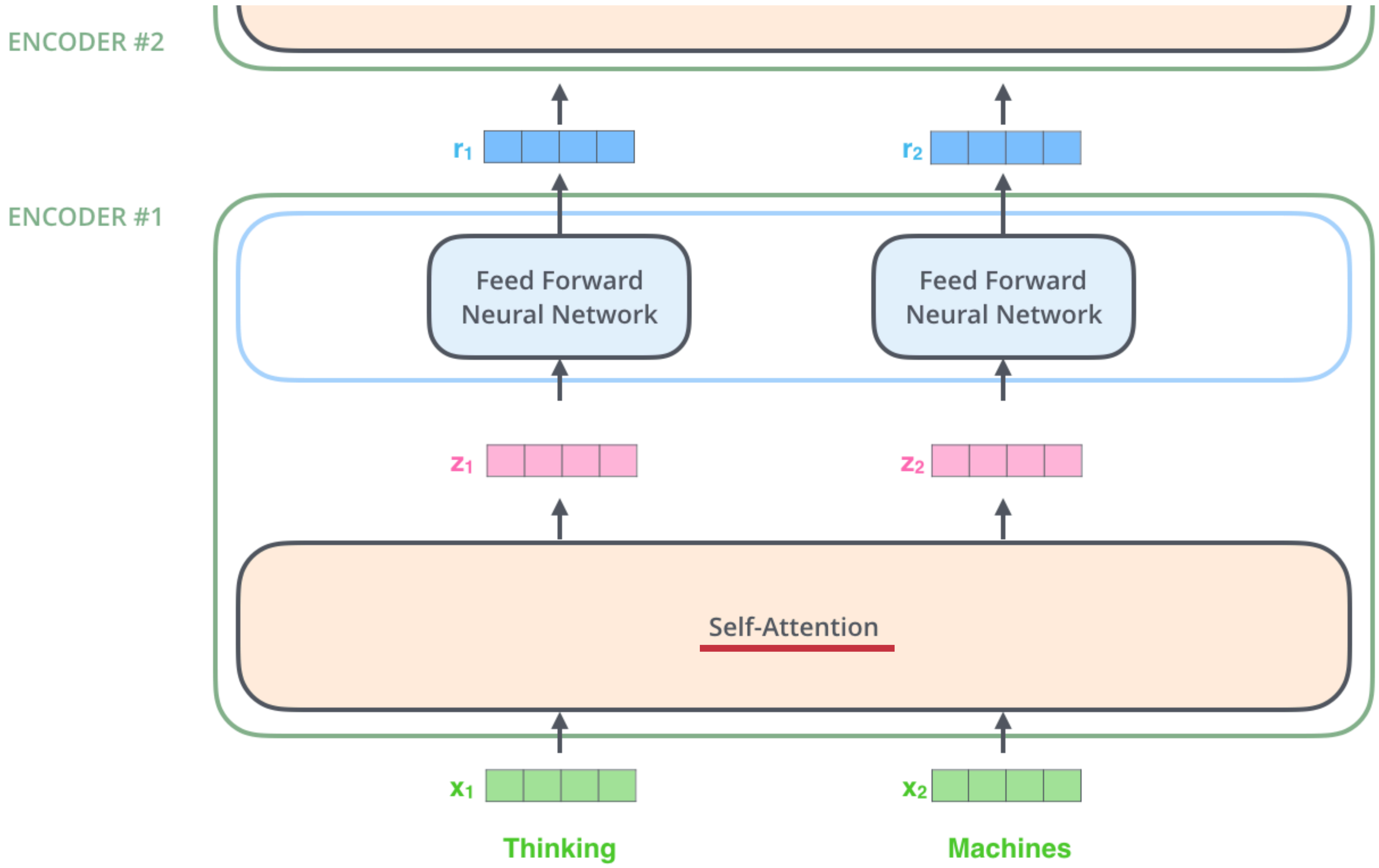


Image credit

Transformer: self-attention

Seq2seq attention: между decoder vector и encoder vectors

- какие слова в input “важны” для предсказания текущего output

$$attention = softmax(qK^T)V$$

Image credit

Transformer: self-attention

Seq2seq attention: между decoder vector и encoder vectors

- какие слова в input “важны” для предсказания текущего output

$$attention = softmax(qK^T)V$$

Self-attention (encoder): между encoder векторами

- какие слова в input как соотносятся между собой

$$attention = softmax(\frac{QK^T}{d})V$$

Image credit

Transformer: self-attention

”`The animal didn't cross the street because it was too tired`”

На что указывает it?

Image credit

Transformer: self-attention

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

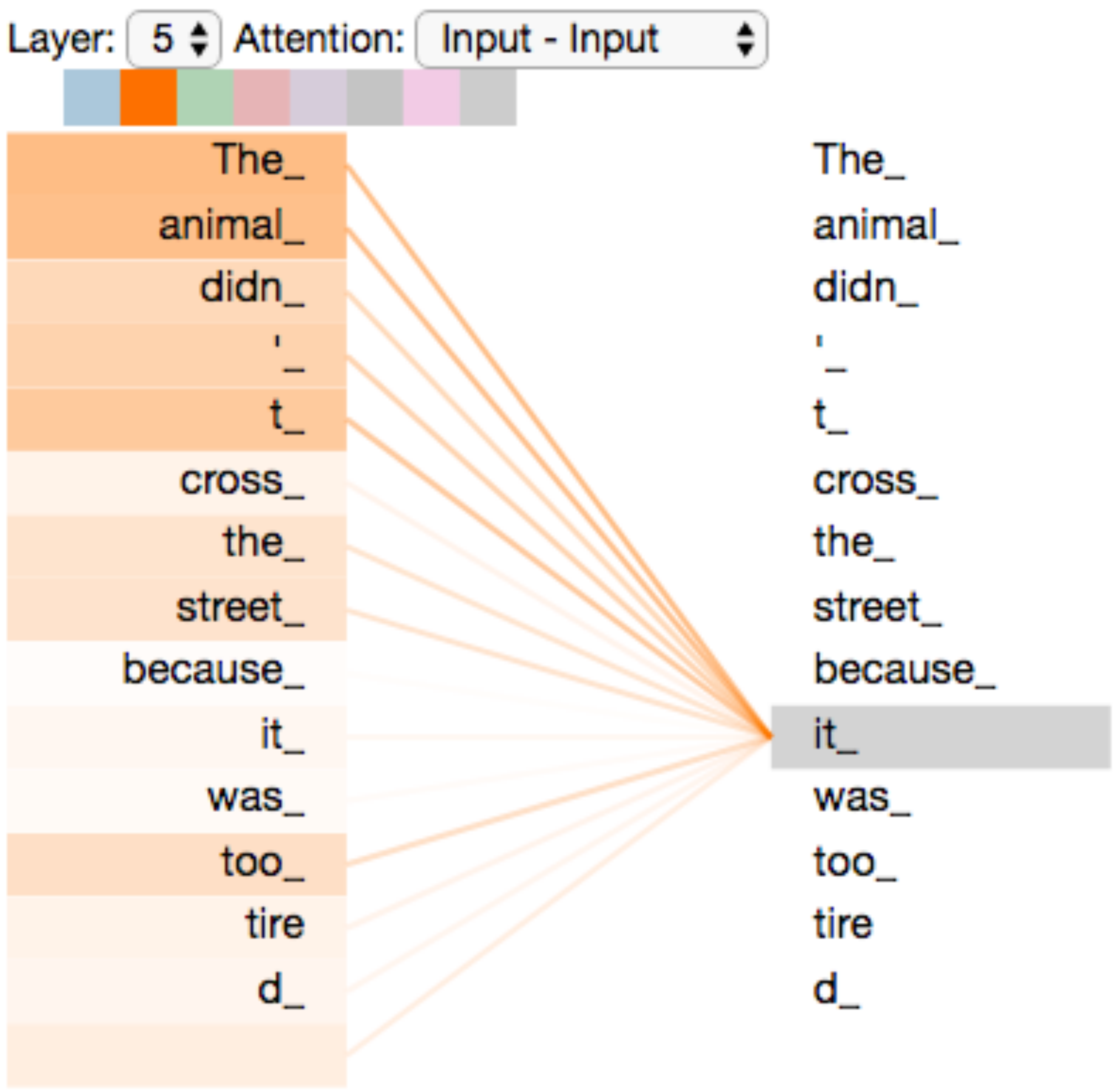


Image credit

Transformer: self-attention

$$attention = softmax(\frac{QK^T}{d})V$$

$$q_1 = W_Q x_1$$

$$k_1 = W_K x_1$$

$$v_1 = W_V x_1$$

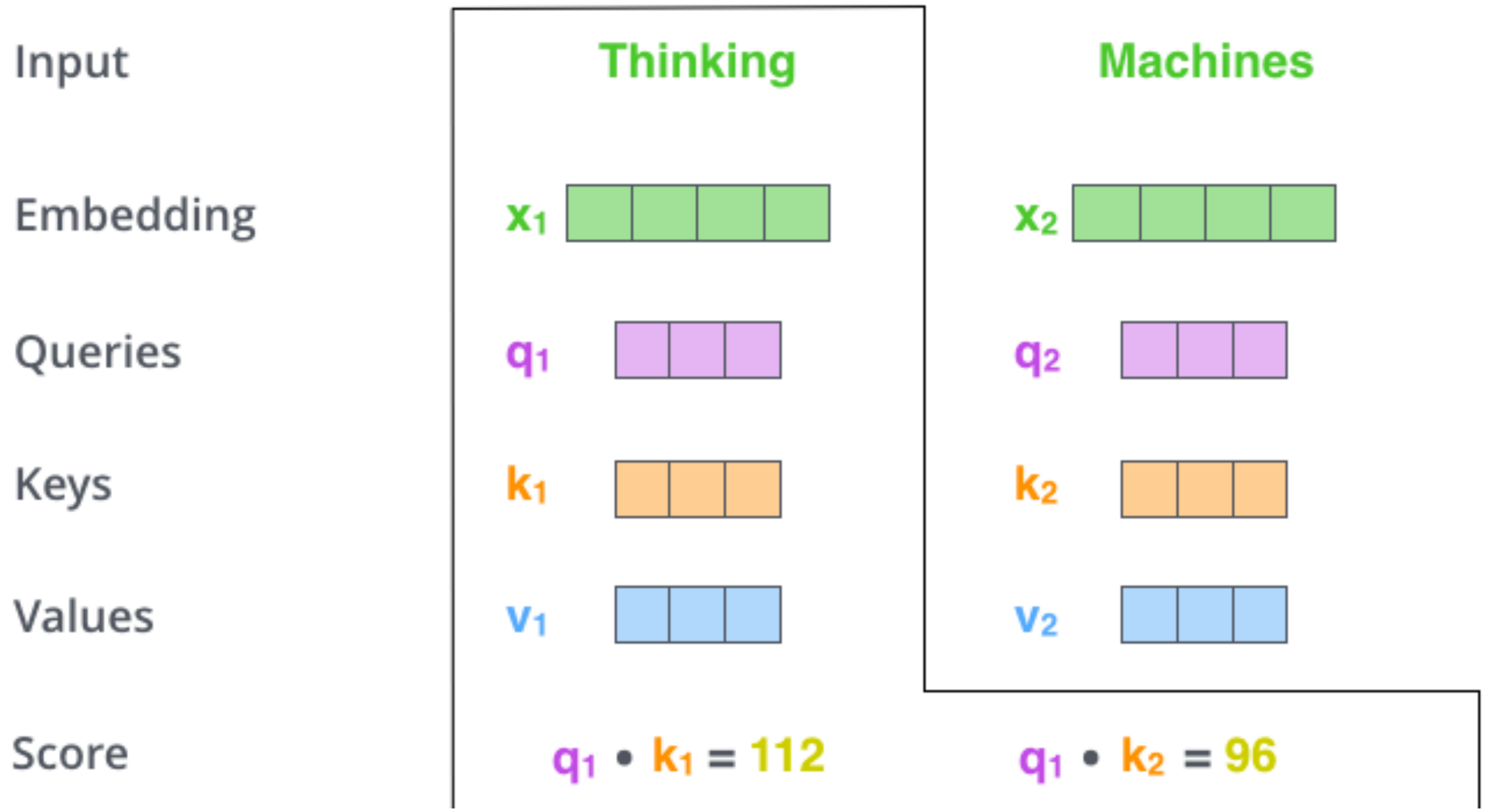


Image credit

Transformer: self-attention

$$attention = softmax(\frac{QK^T}{d})V$$

$$q_1 = W_Q x_1$$

$$k_1 = W_K x_1$$

$$v_1 = W_V x_1$$

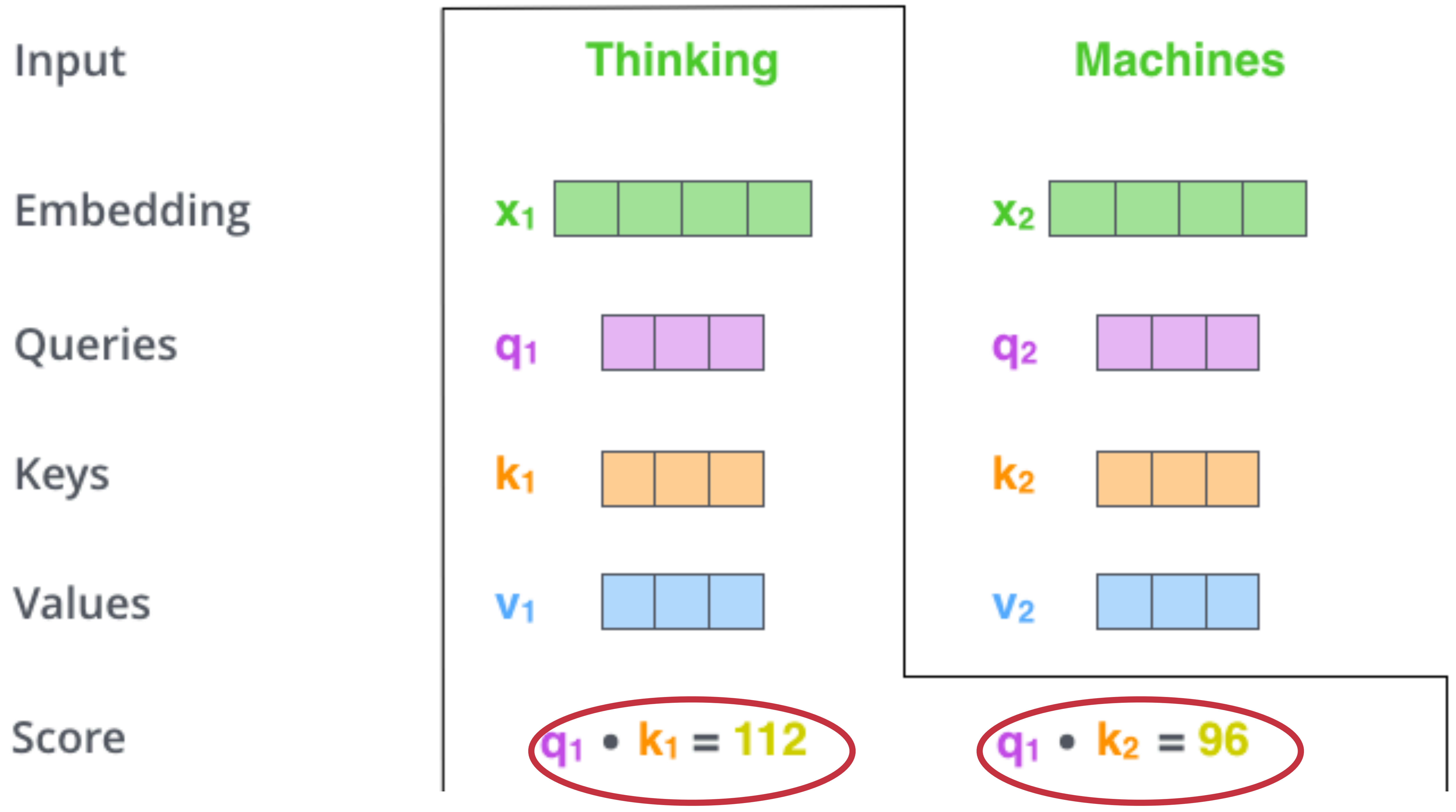


Image credit

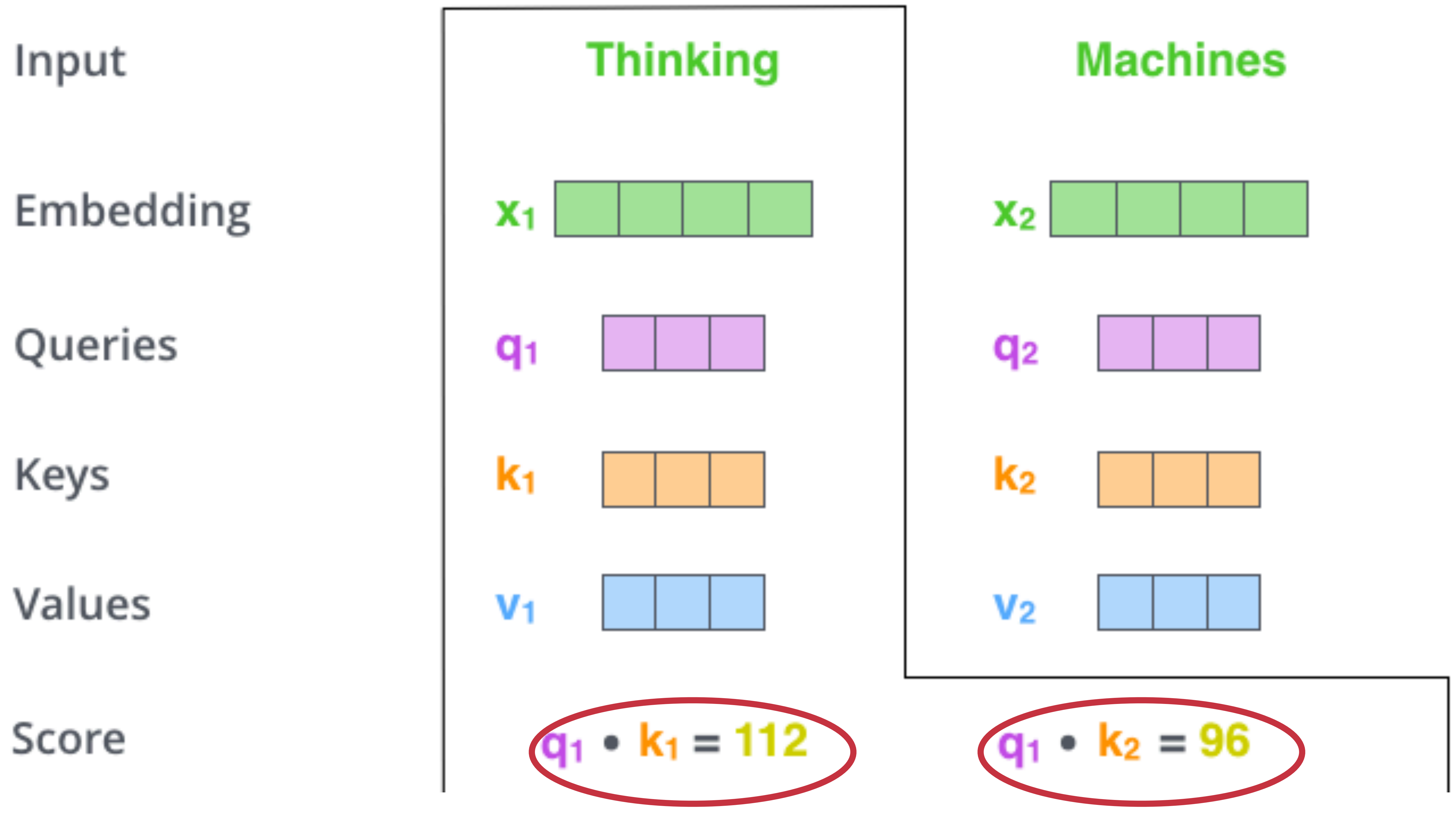
Transformer: self-attention

$$attention = softmax(\frac{QK^T}{d})V$$

$$q_1 = W_Q x_1$$

$$k_1 = W_K x_1$$

$$v_1 = W_V x_1$$



То же для x_2

Image credit

Transformer: self-attention

$$attention = softmax(\frac{QK^T}{d})V$$

$$q_1 = W_Q x_1$$

$$k_1 = W_K x_1$$

$$v_1 = W_V x_1$$

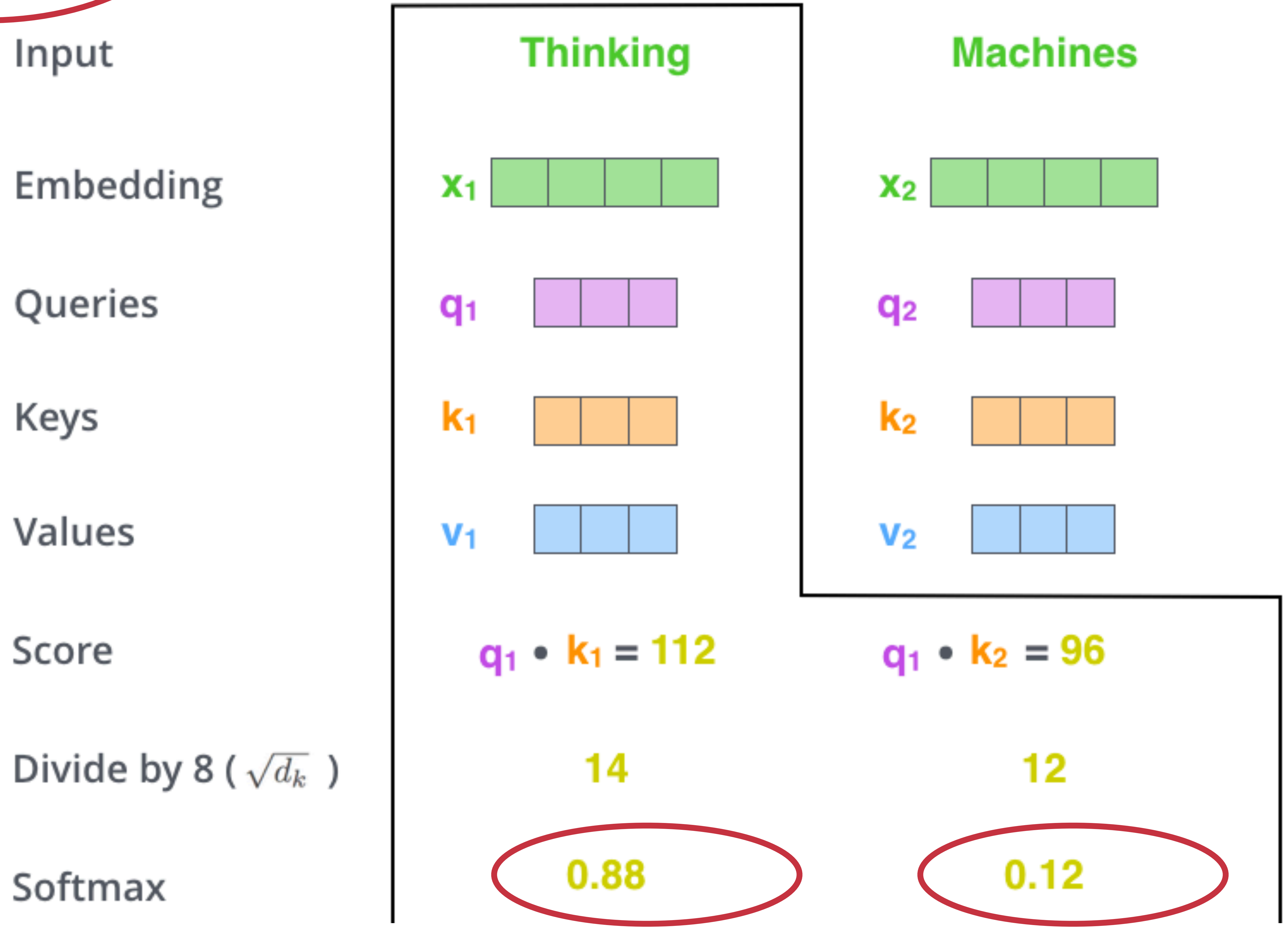


Image credit

Transformer: self-attention

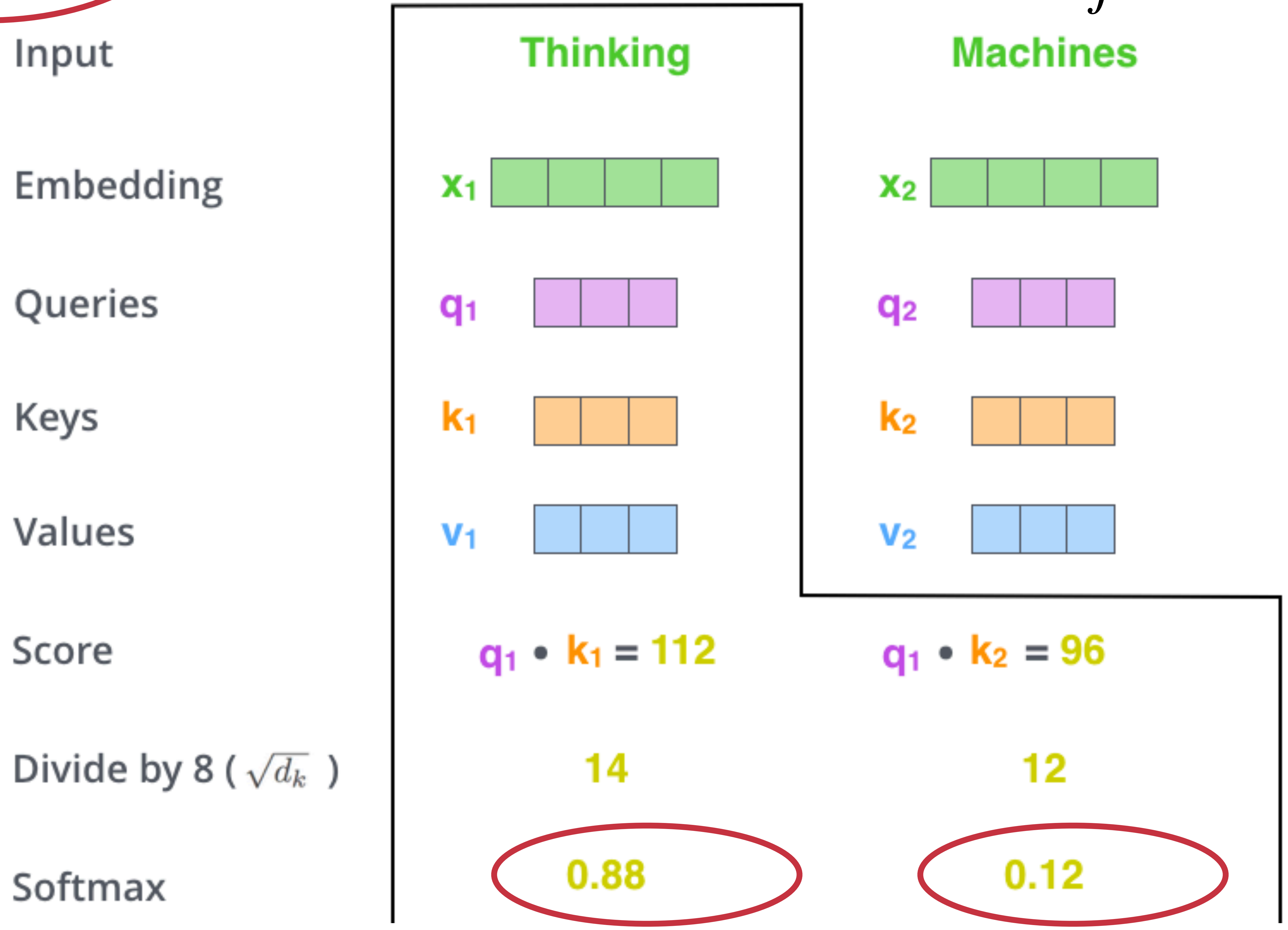
$$attention = softmax(\frac{QK^T}{d})V$$

Для каждого x_i получаем веса, насколько он соотносится с x_j - всеми элементами входа

$$q_1 = W_Q x_1$$

$$k_1 = W_K x_1$$

$$v_1 = W_V x_1$$



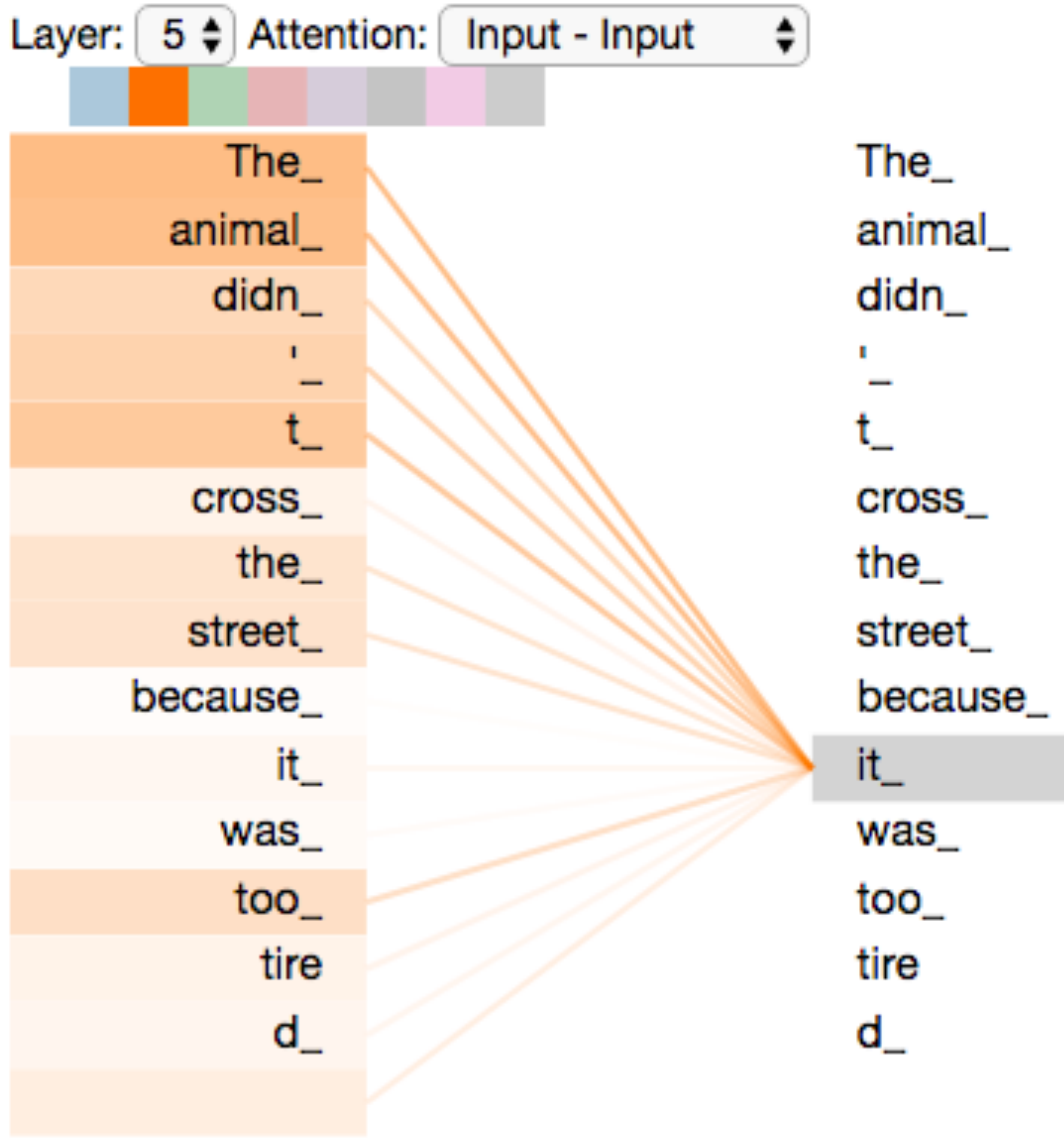
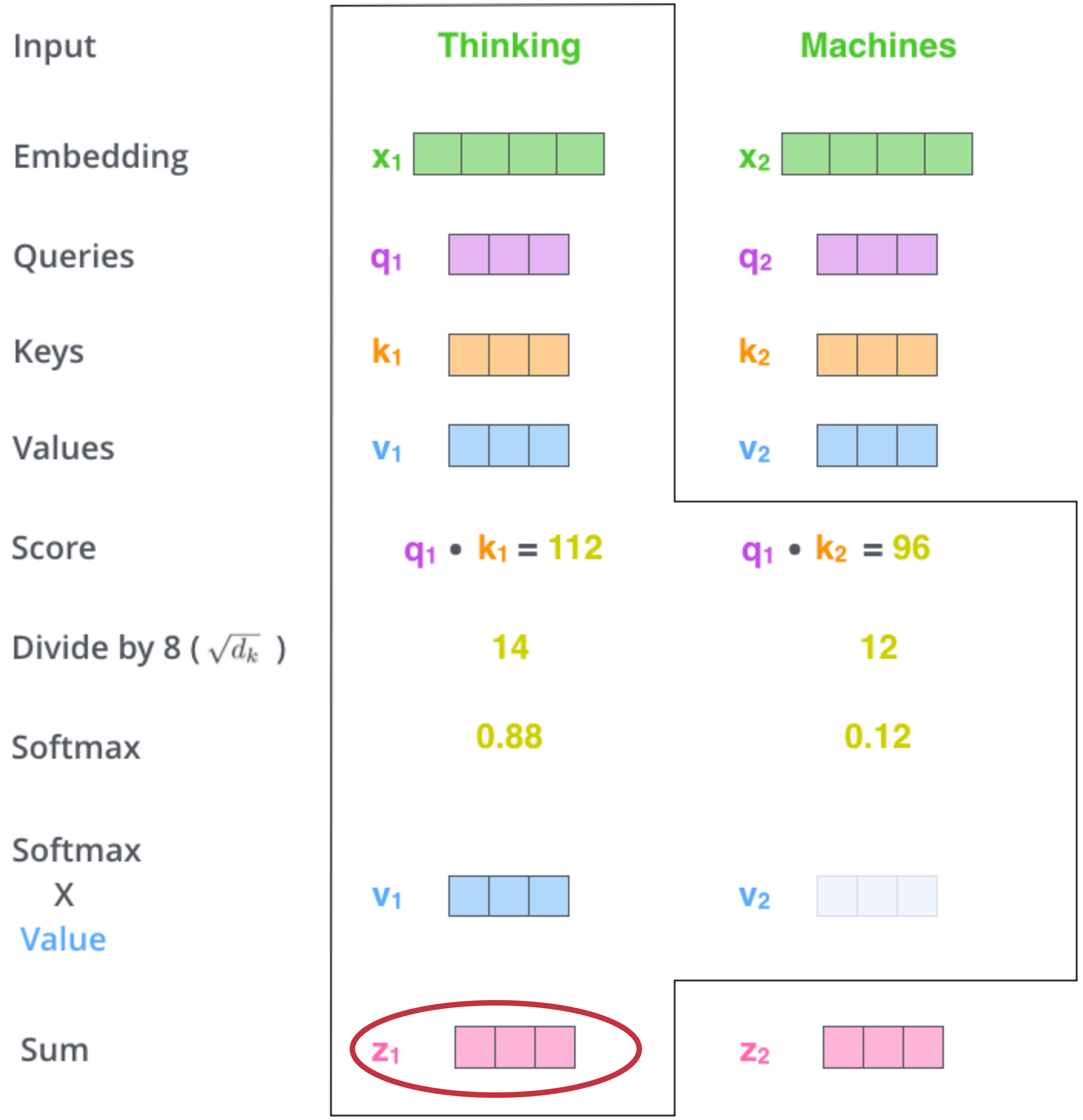


Image credit

Transformer: self-attention

$$attention = softmax(\frac{QK^T}{d})V$$



Взвешенная сумма всех v_j,
веса - из softmax

Image credit

Transformer: self-attention

$$attention = softmax(\frac{QK^T}{d})V$$

Матричная запись

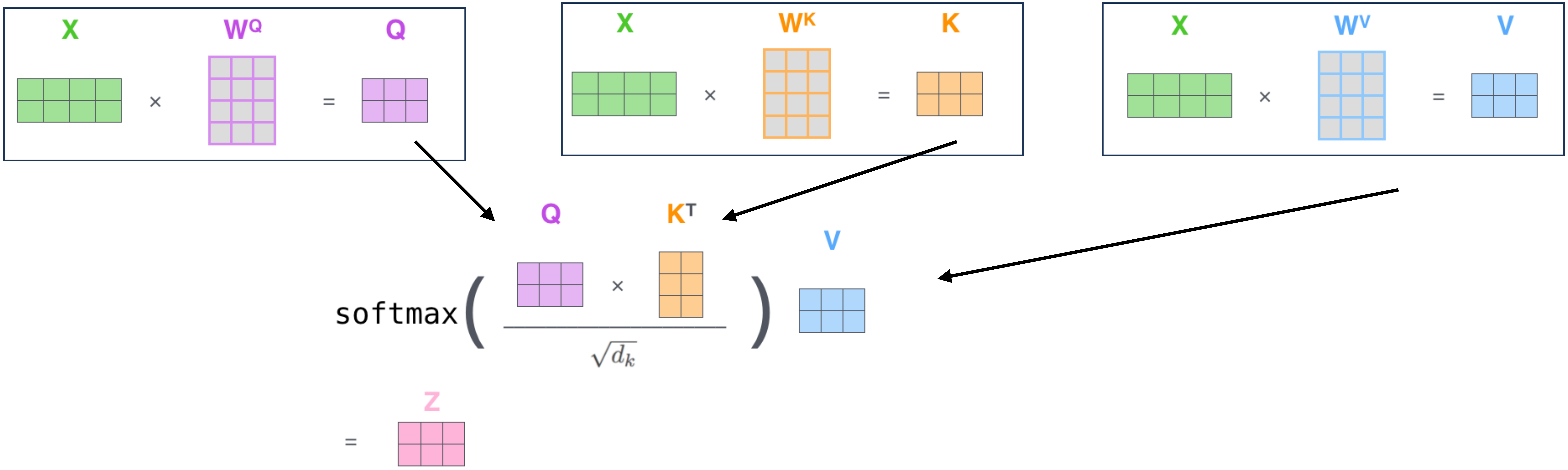


Image credit

Transformer: multihead self-attention

Используем несколько self-attention слоев сразу - с разными весами

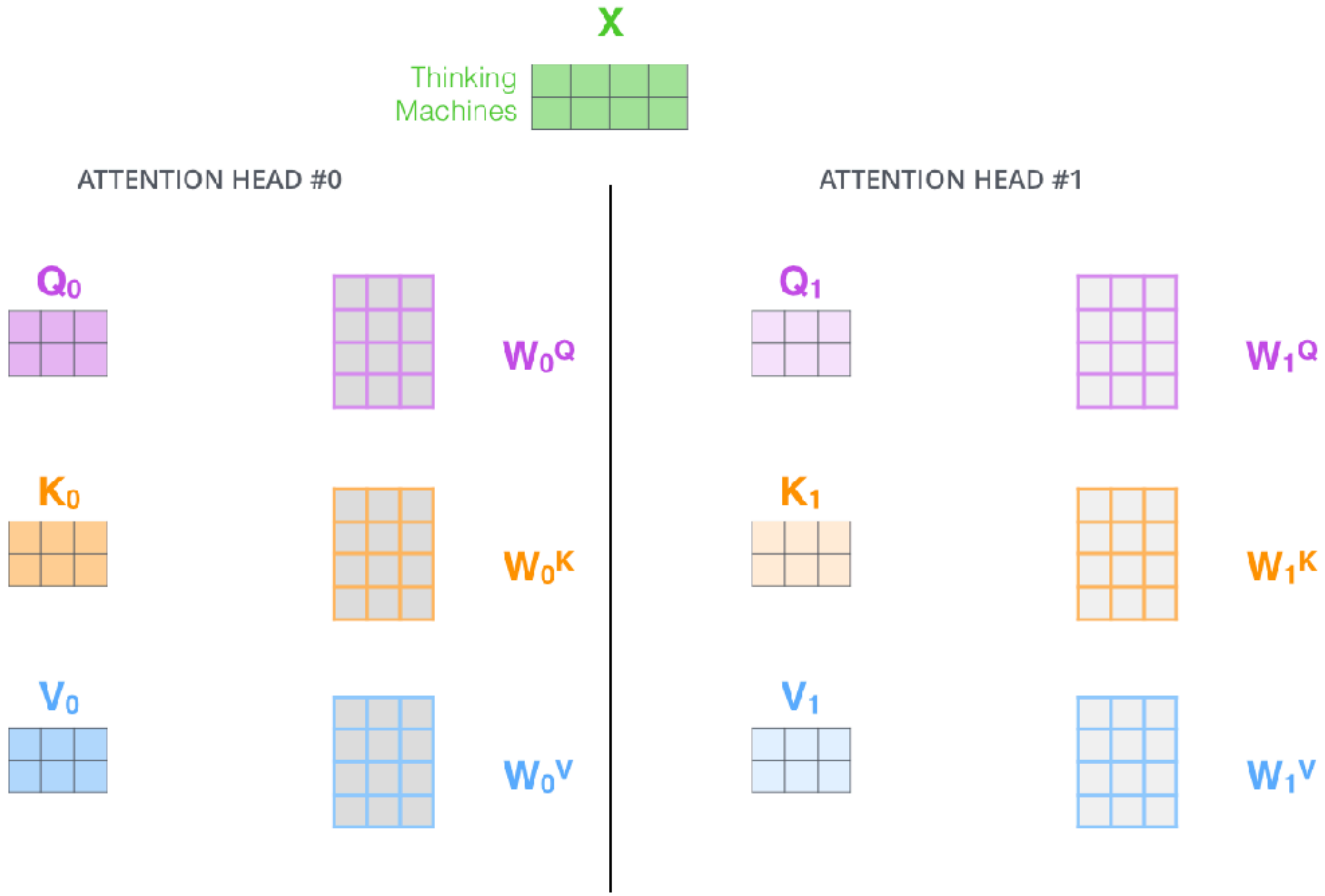


Image credit

Transformer: multihead self-attention

Используем несколько self-attention слоев сразу - с разными весами

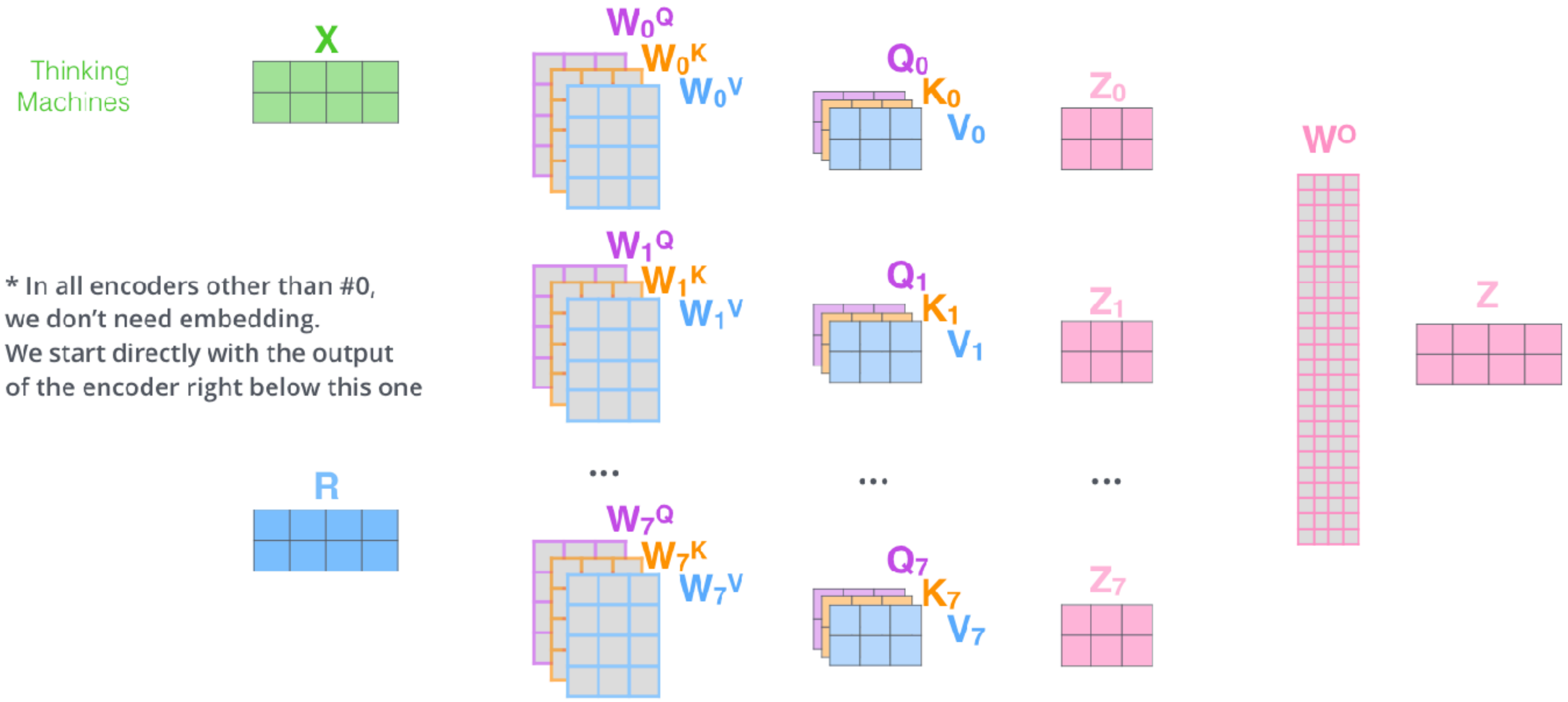


Image credit

Transformer: multihead self-attention

Используем несколько self-attention слоев сразу - с разными весами

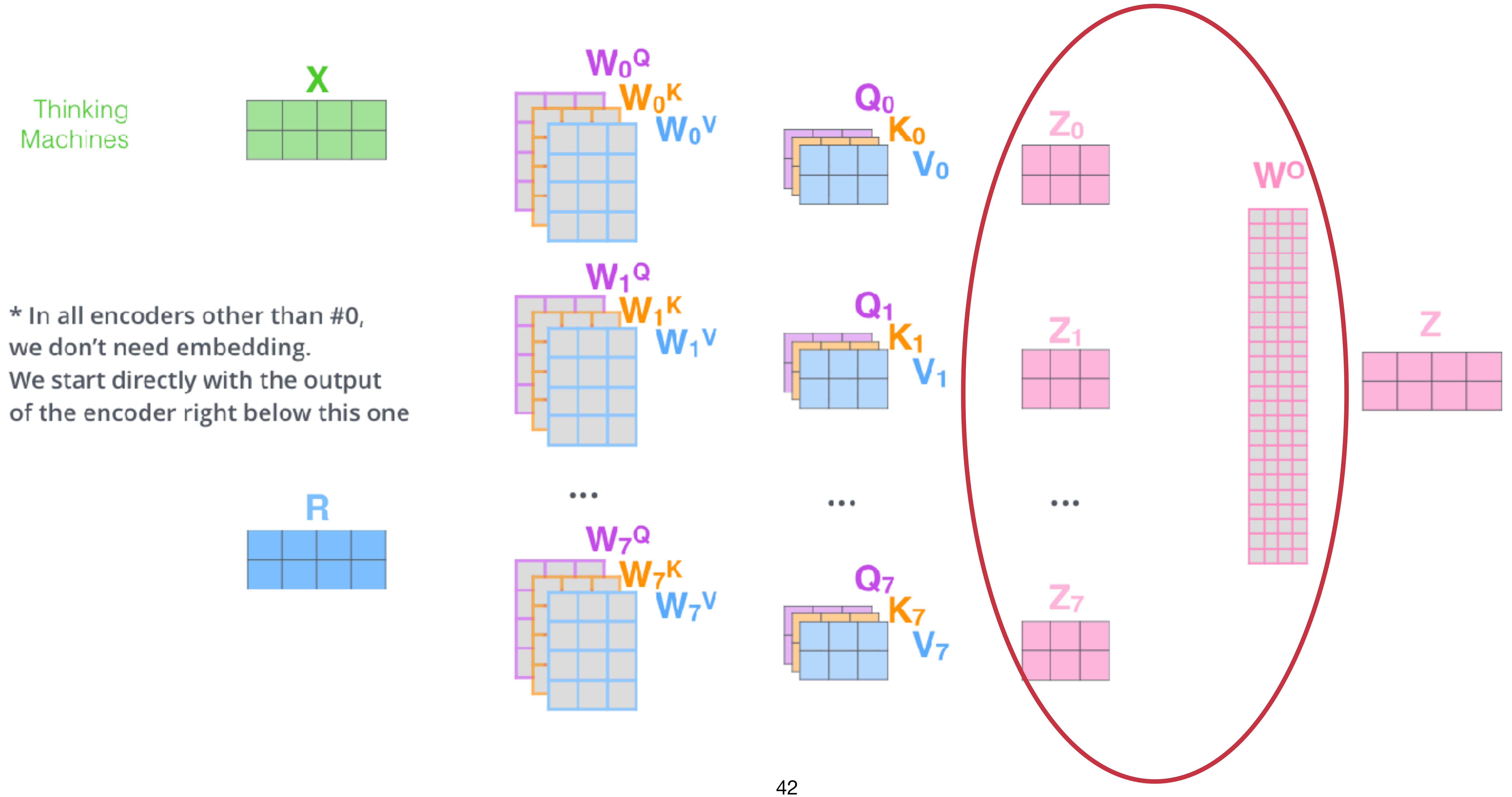
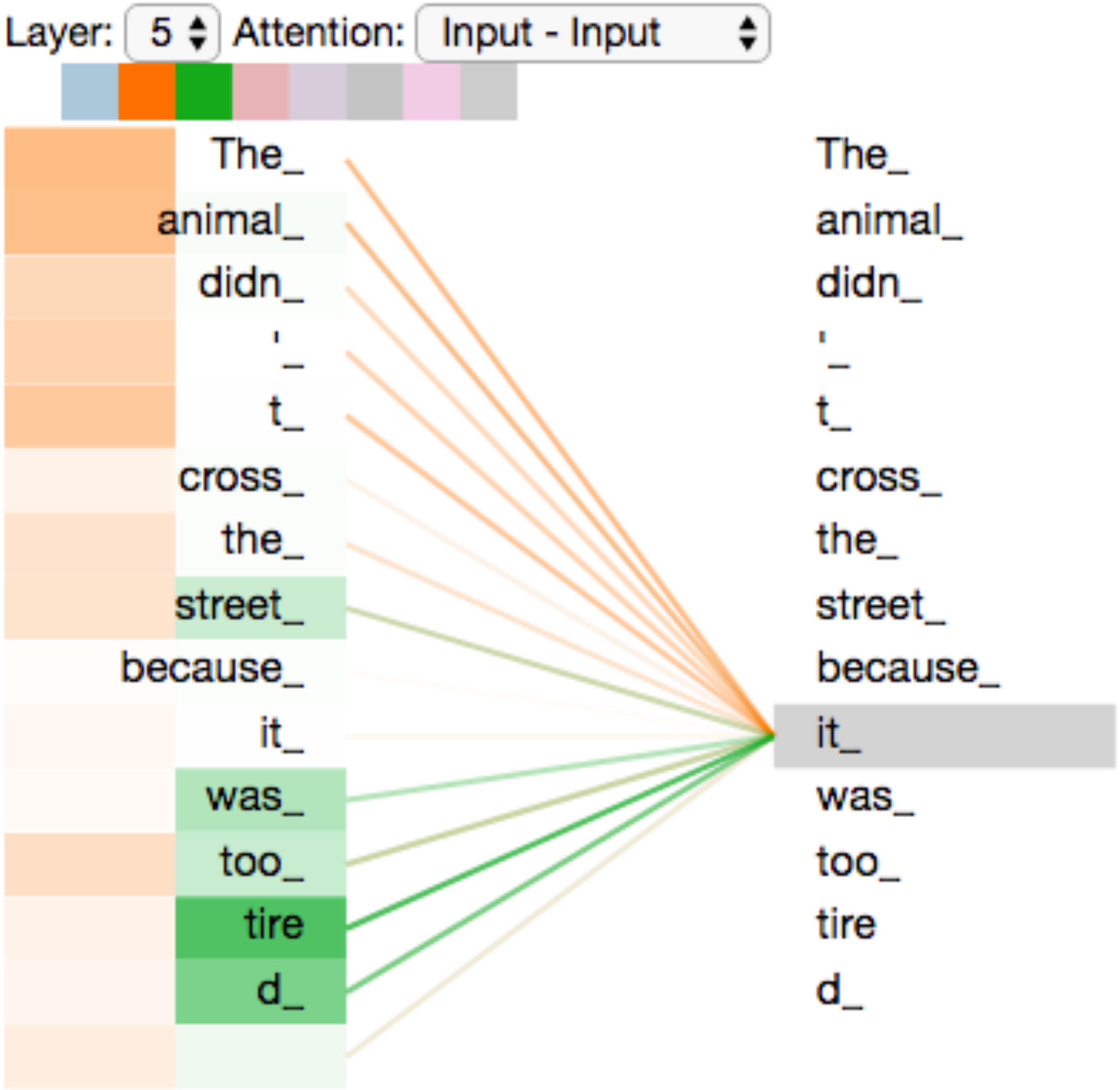


Image credit

Transformer: multihead self-attention

Мотивация: разные self-attention “обращают внимание” на разные признаки



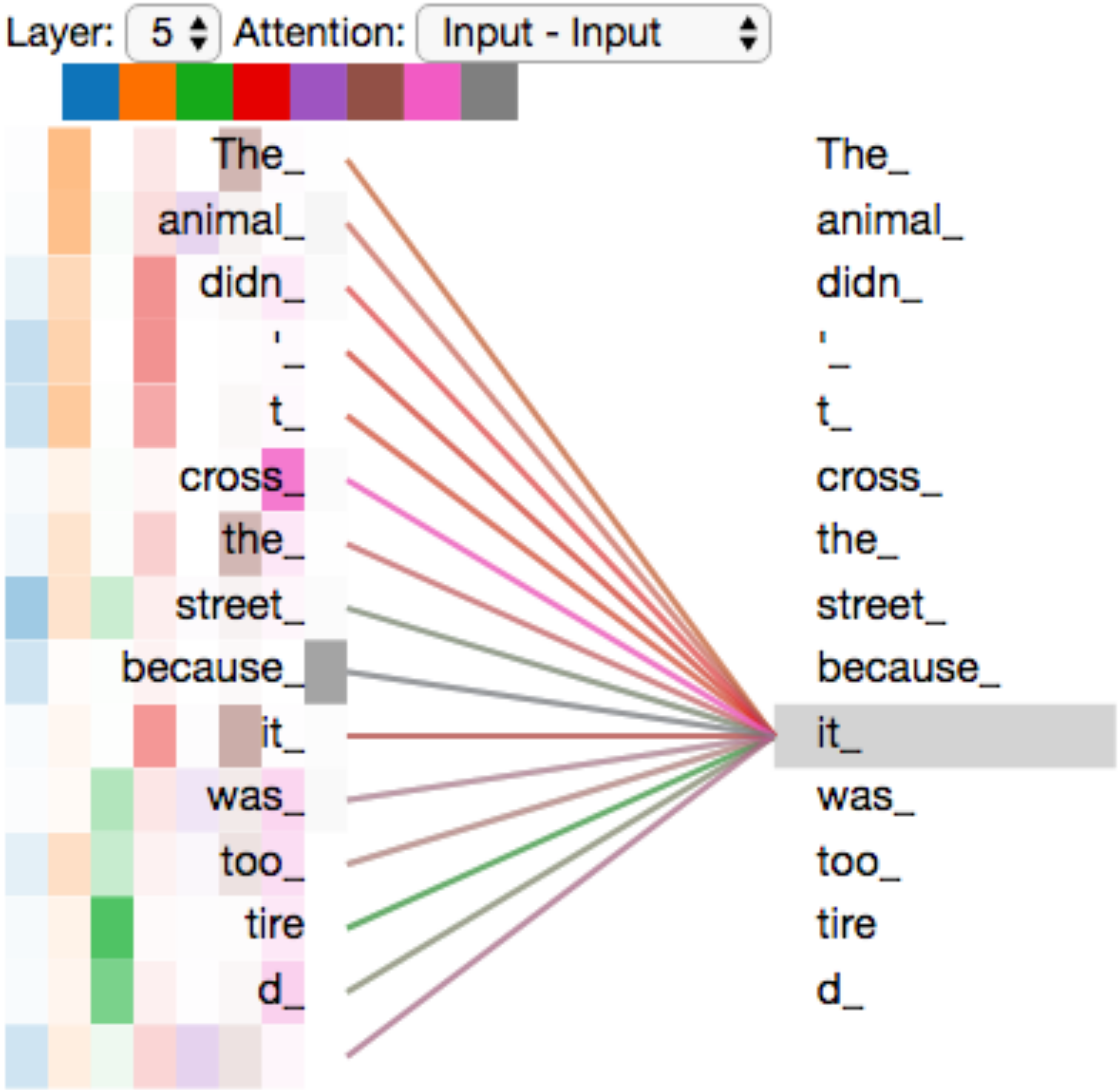
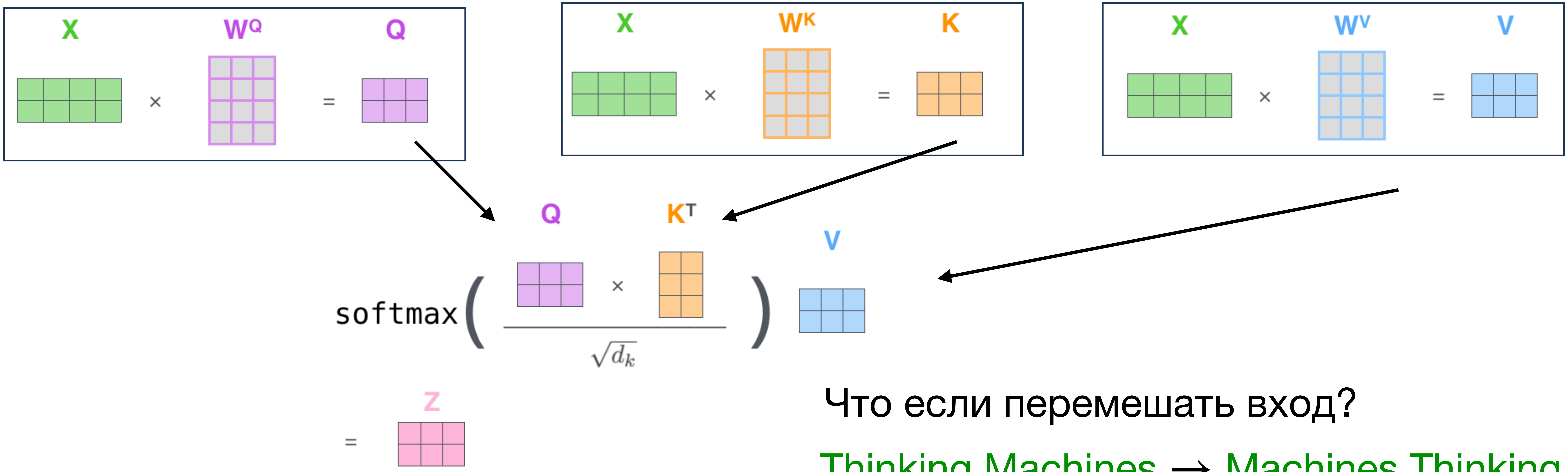


Image credit

Transformer: multihead self-attention

$$attention = softmax(\frac{QK^T}{d})V$$



Что если перемешать вход?

Thinking Machines → Machines Thinking

Transformer: positional embeddings

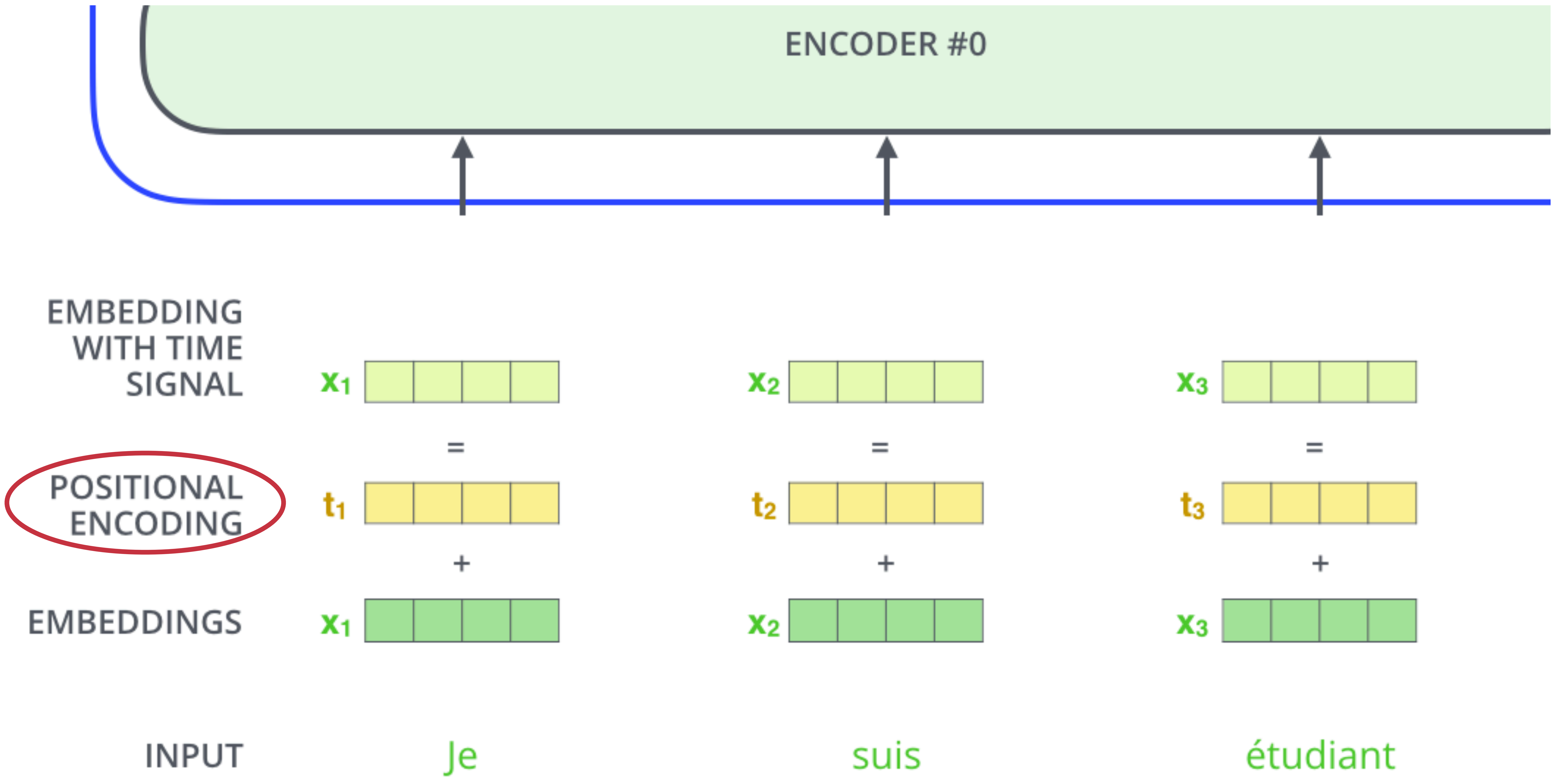


Image credit

Transformer: positional embeddings

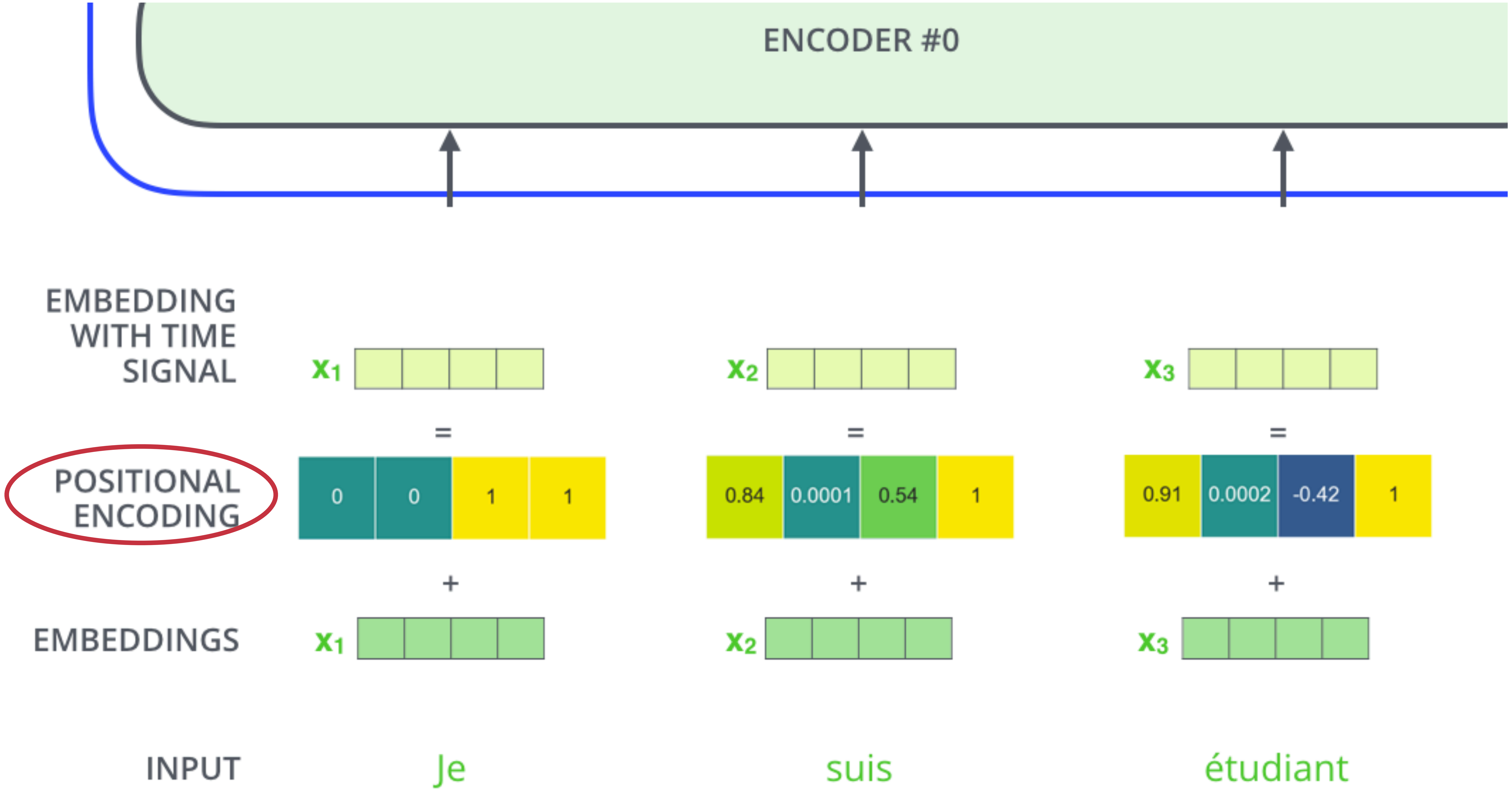


Image credit

Transformer: Residuals & LayerNorm

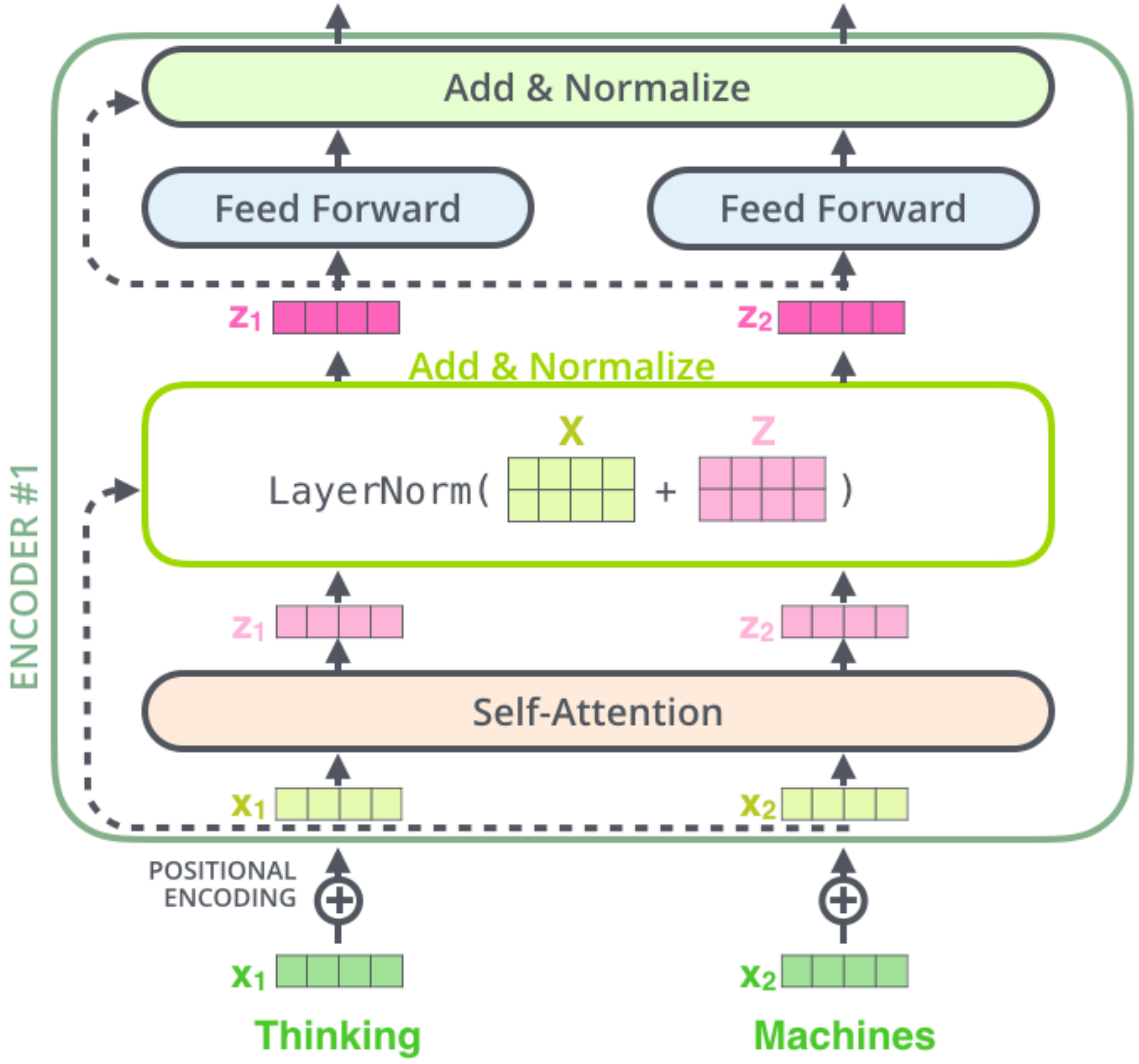
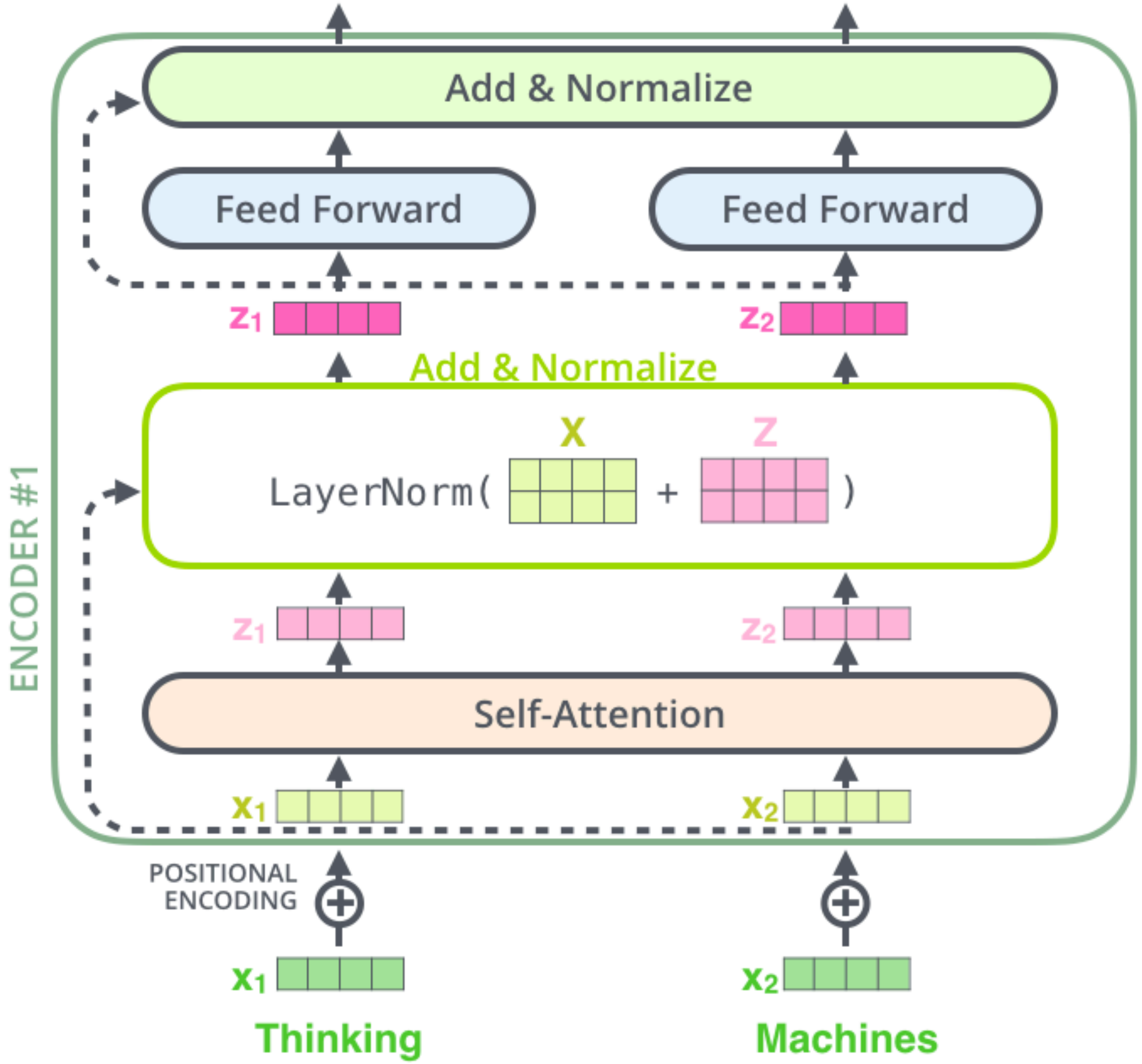


Image credit

Transformer: Residuals & LayerNorm



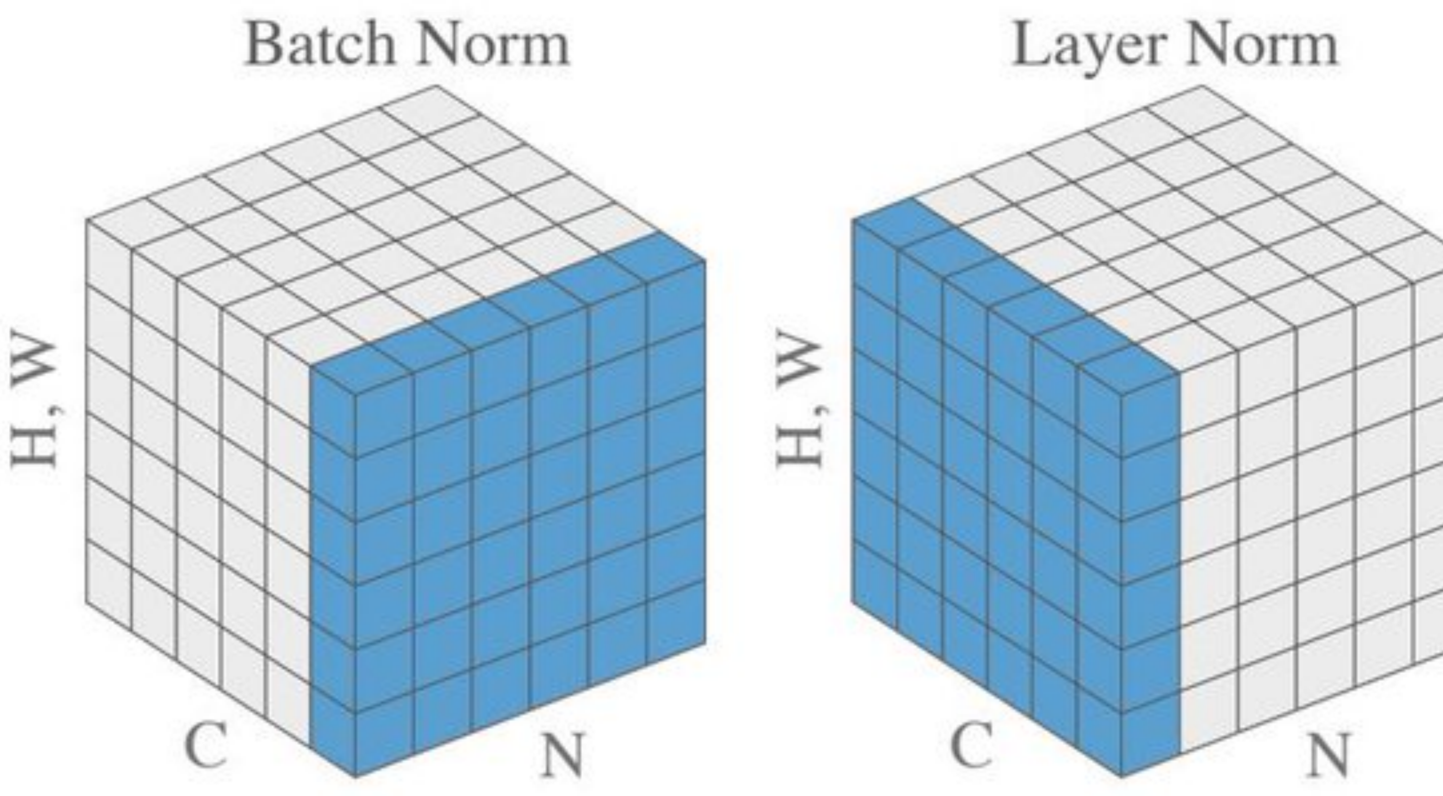


Image credit

Transformer: encoder-decoder attention

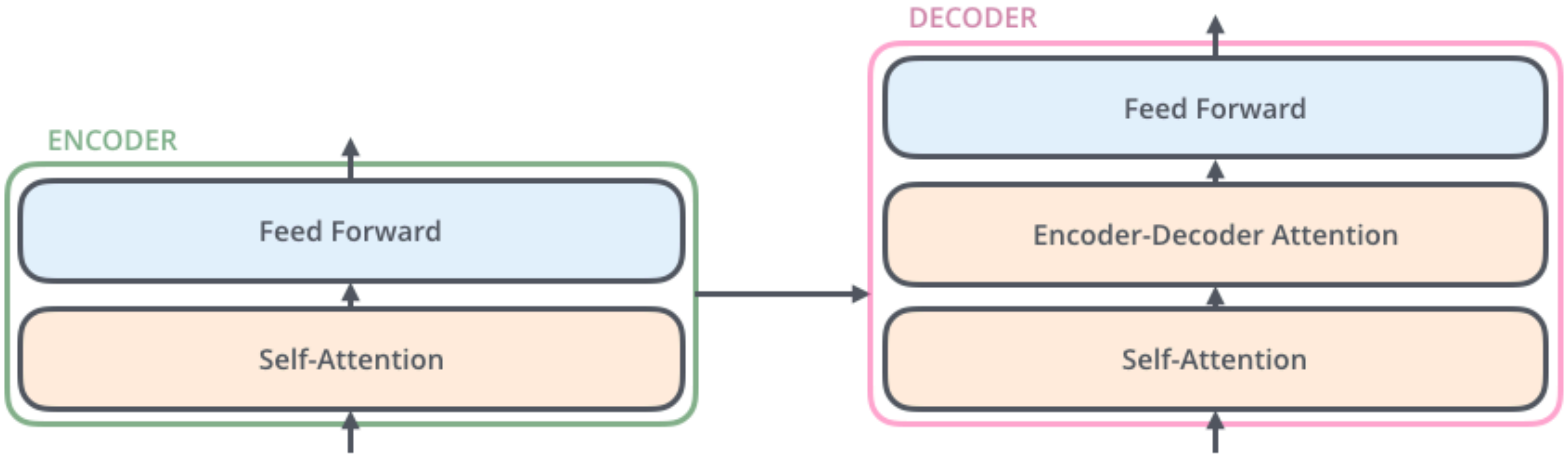
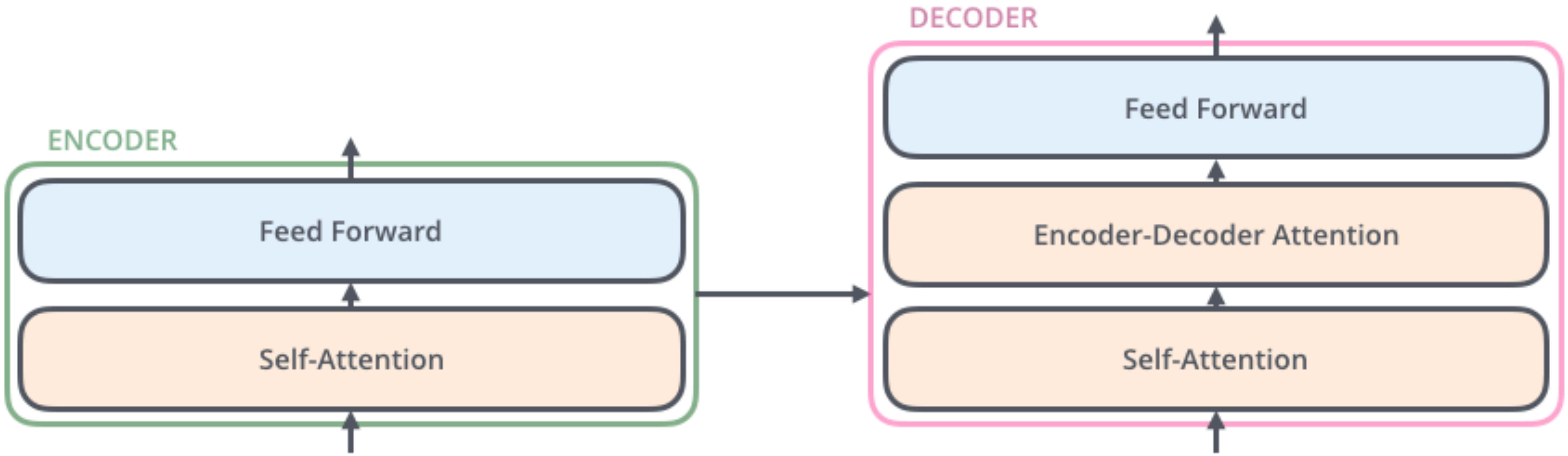


Image credit

Transformer: encoder-decoder attention



$$Enc\text{-}Dec\ attention = softmax(\frac{Q_{decoder}K^T_{encoder}}{d})V_{encoder}$$

Image credit

Transformer

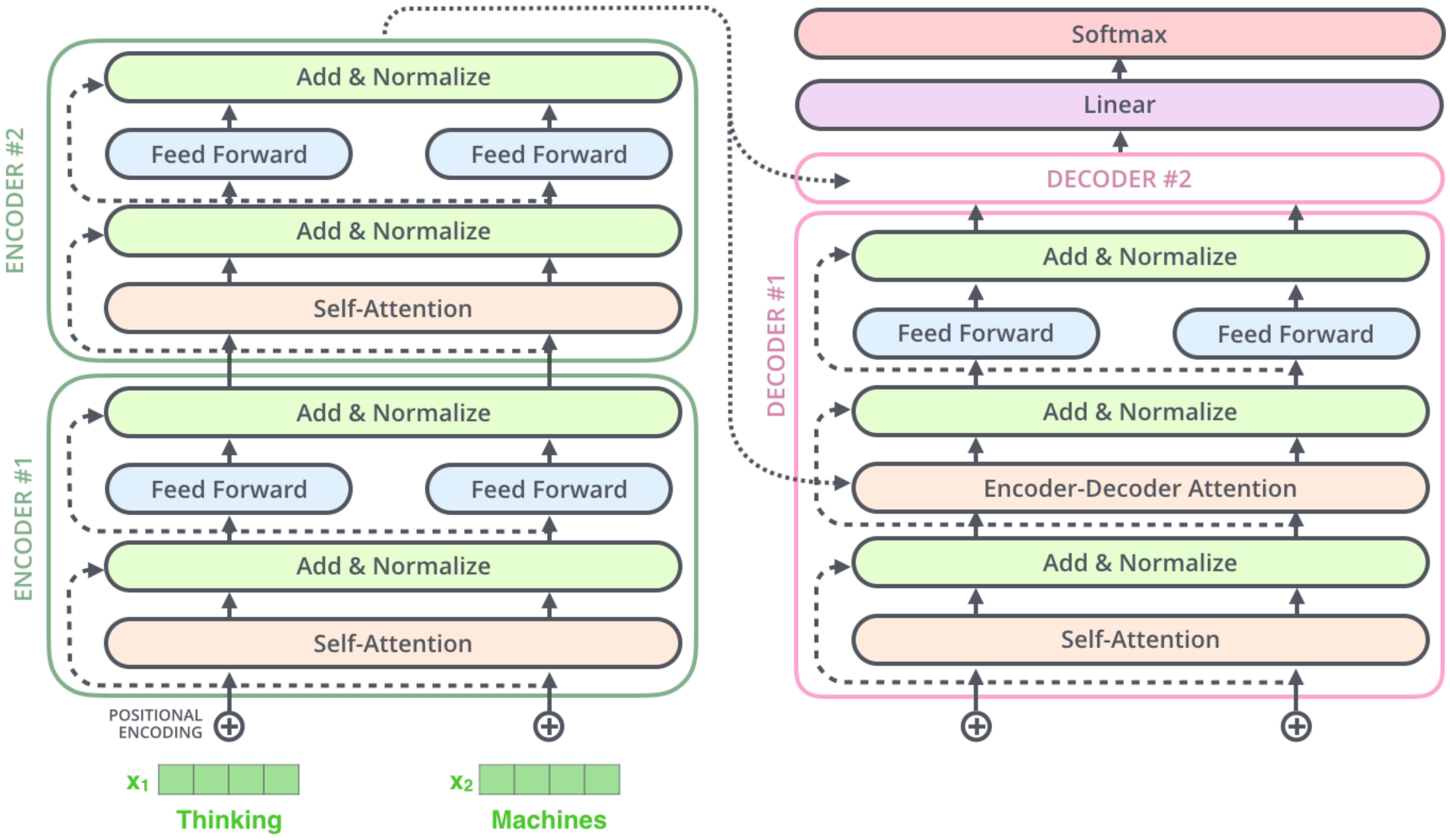
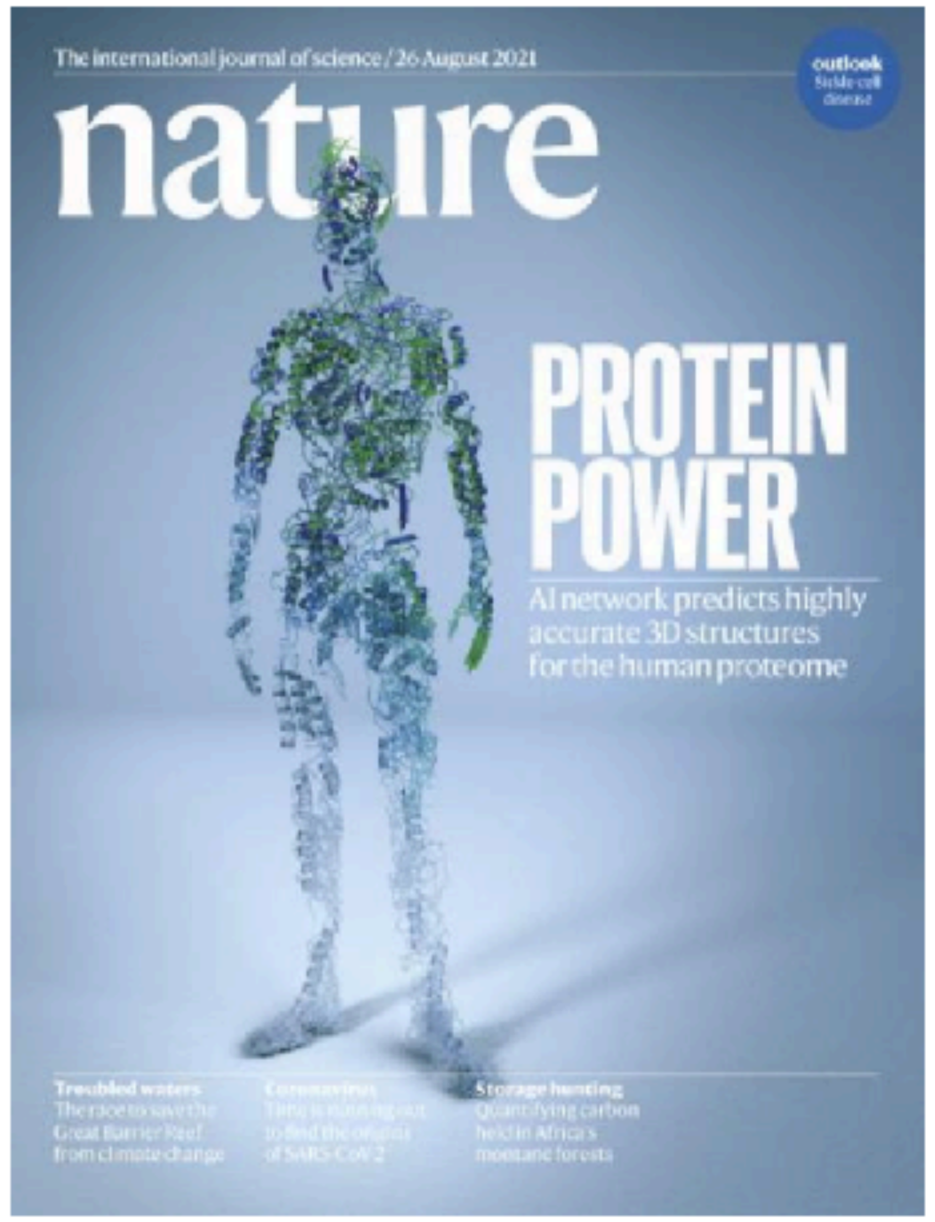


Image credit

Transformer

- BERT, GPT-1,2,3, etc. - **предобученные** трансформеры
- Трансформеры работают не только на текстах!

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

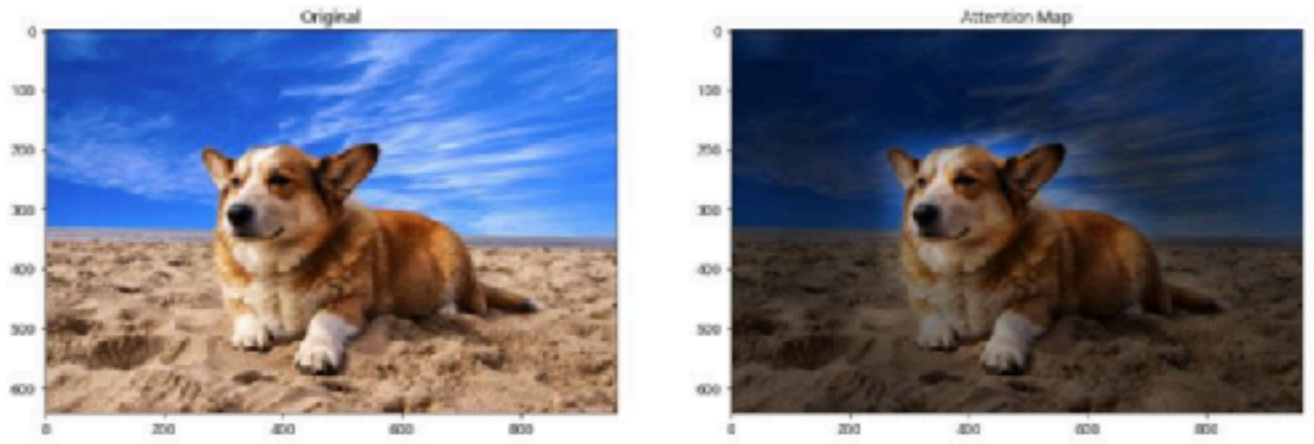
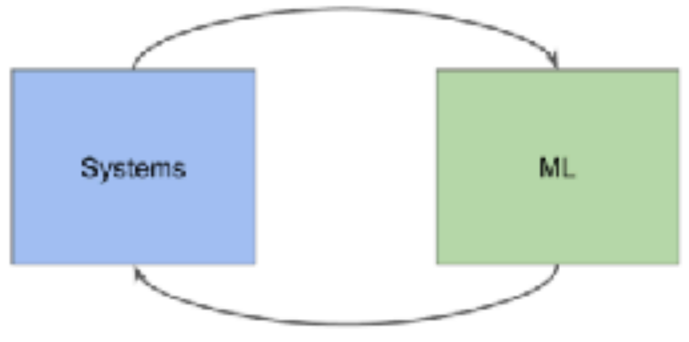
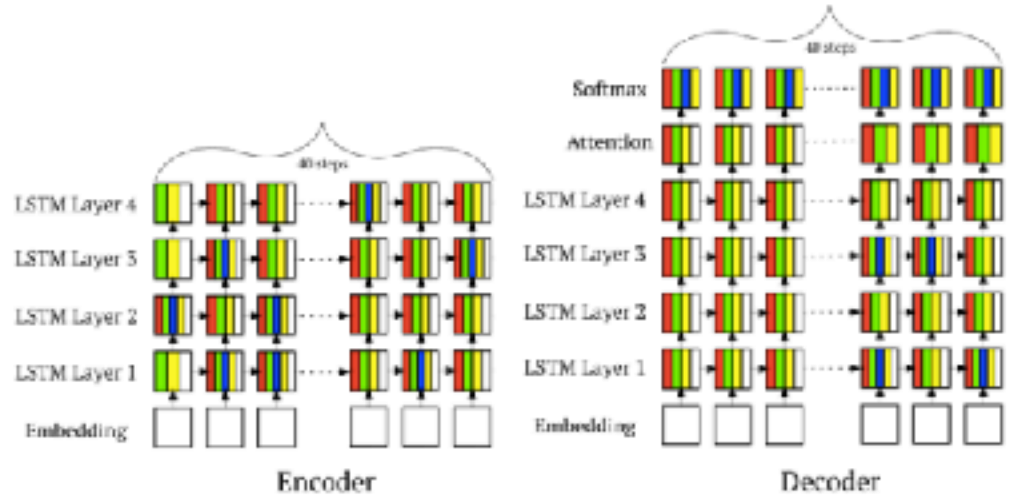


Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k





ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

Model (#devices)	GO-one (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
[illegible]	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
[illegible]	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**