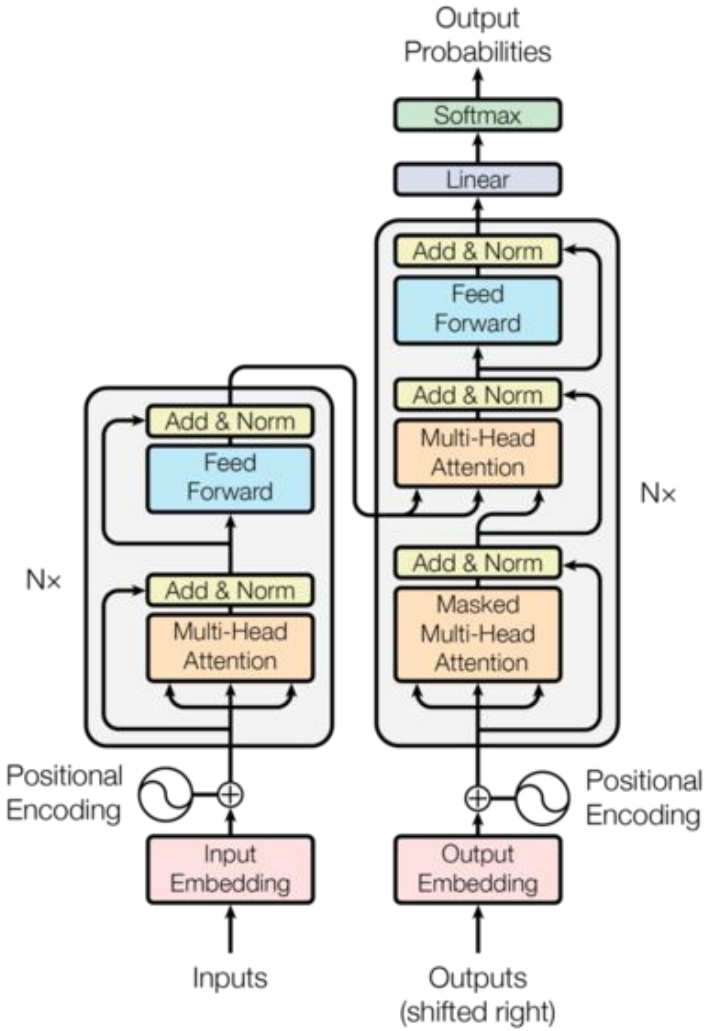
CS5841/EE5841 Machine Learning

Lecture 14: Transformers

Evan Lucas

Transformer Overview

- Built to operate on sequential data
- Uses attention without convolutional or recurrent layers
  - Mimics cognitive attention - focuses on what aspects of input are relevant
- Input to Transformer includes a positional encoding, either absolute or relative
- Can be bidirectional or unidirectional
- Often used in an encoder-decoder format, such as in the image on the right
- Introduced in 2017 by Google and affiliates [1]



Original Transformer architecture from [1]

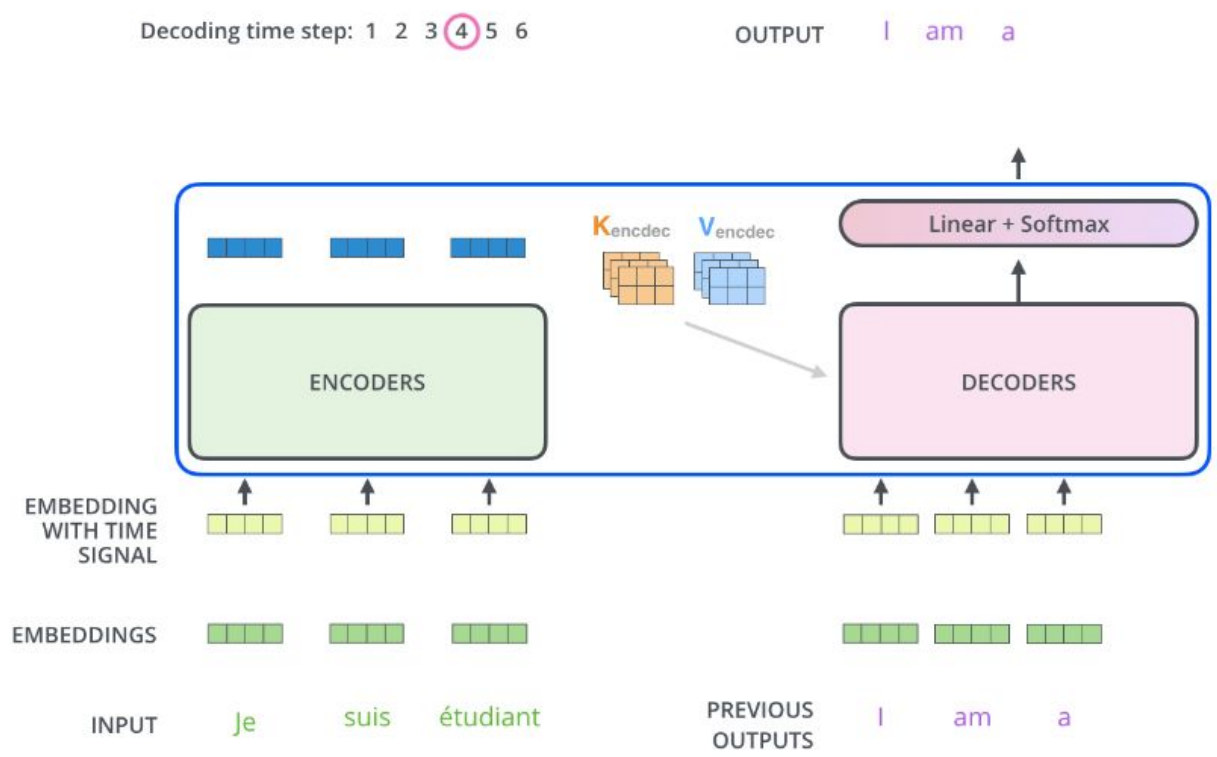
1. Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

Michigan Tech

Transformer Architecture Discussion

For summarization, we use encoder-decoder architectures

The decoder uses encoded input plus previous outputs to determine next output



https://jalammar.github.io/illustrated-transformer/

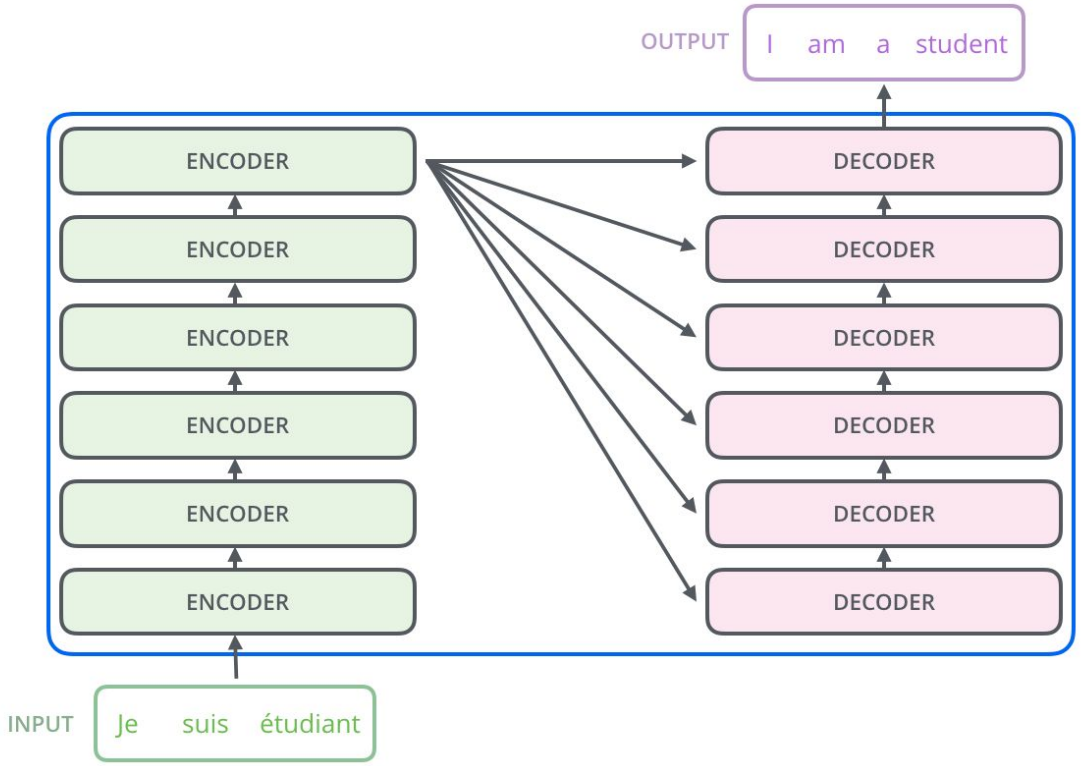
Other transformer architectures

- Encoder only (ex: BERT)
  - Trained by making model predict missing/corrupted tokens (Autoencoding pretraining)
  - Better suited for classification type tasks
- Decoder only (ex:GPT2/3)
  - Trained by having model predict next token (Autoregressive pretraining)
  - Well suited for text generation
- Encoder-Decoder (ex: Longformer Encoder-Decoder)
  - Multiple training methods, including previous as well as sequence to sequence generation
  - Used for summarization, translation, and question answering

https://huggingface.co/transformers/model_summary.html

Transformers Step-by-Step

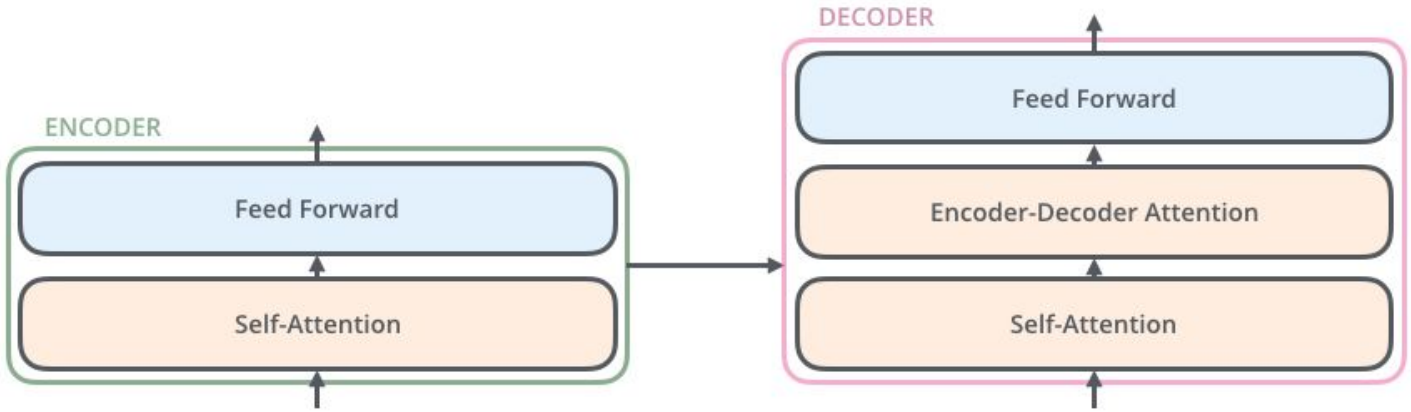
- Input is fed through several layers of encoders
- Final encoder output is fed into each decoder layer



https://jalammar.github.io/illustrated-transformer/

Transformers Step-by-Step

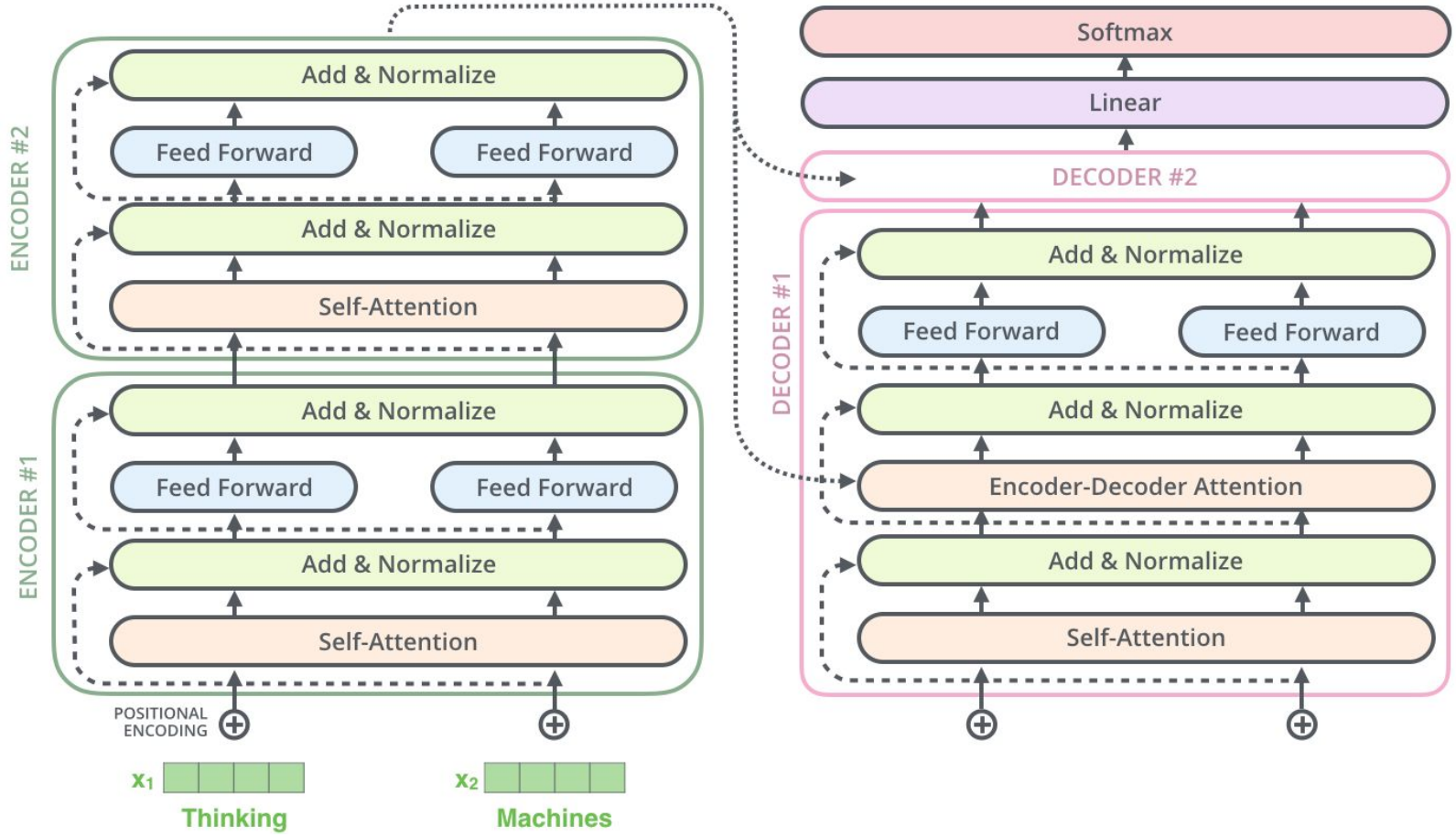
- Each encoder layer includes two main stages - multi-headed self attention on input and a feedforward layer
- Each decoder layer includes self-attention, cross-attention, and a feedforward layer
- Residual skip connections not shown for clarity



https://jalammar.github.io/illustrated-transformer/

Transformers Step-by-Step

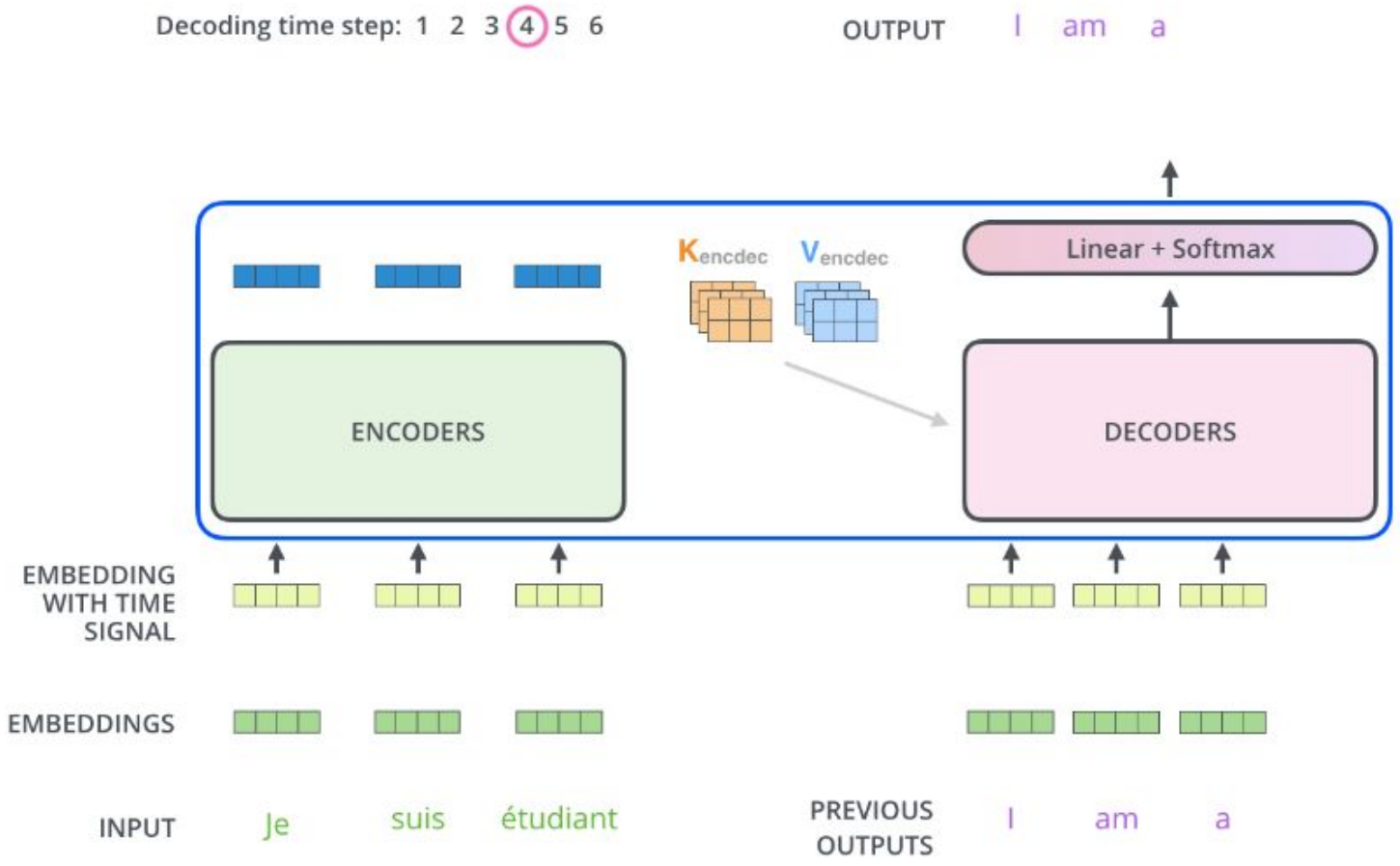
- A more detailed version of the previous slide
- Final output uses Softmax to select output token



https://jalammar.github.io/illustrated-transformer/

Transformers Step-by-Step

- Decoder outputs one token at a time
- Previous outputs are one of the inputs to the decoder

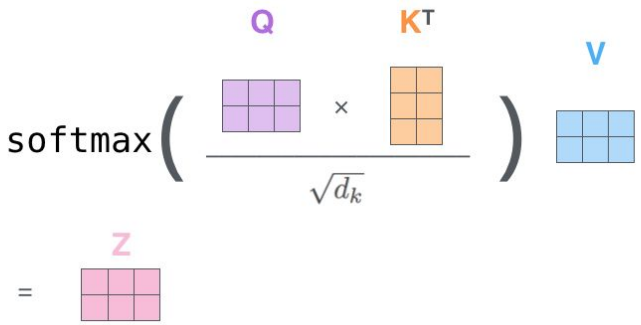


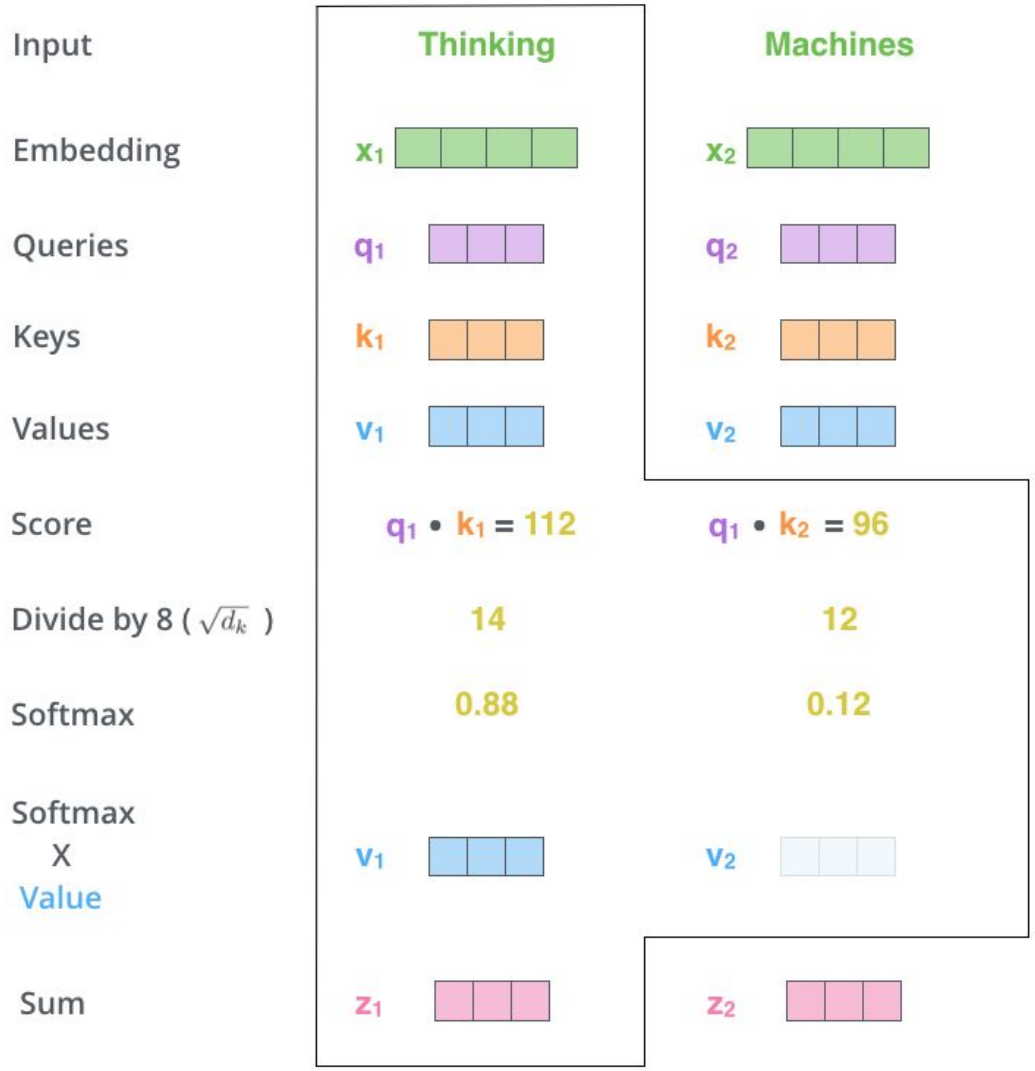
https://jalammar.github.io/illustrated-transformer/

Michigan Tech

Attention

- Each input embedding is multipled by a learned weight to create “queries”, “keys”, and “values”
  - These can be considered representations of the tokens
- Attention is calculated using multiple heads that learn different aspects

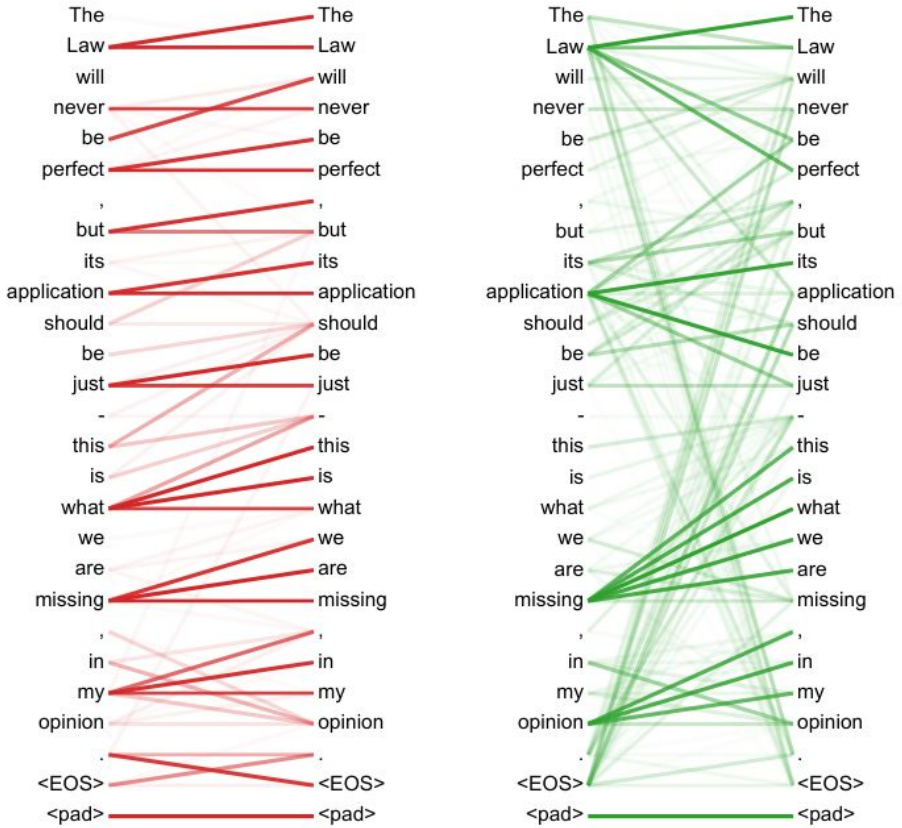




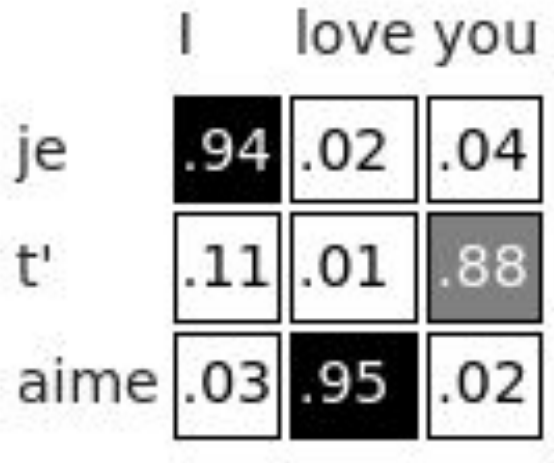
https://jalammar.github.io/illustrated-transformer/

Michigan Tech

Attention



Example of two different self-attention heads https://arxiv.org/abs/1706.03762



Example of single head cross-attention weight matrix for translation https://en.wikipedia.org/wiki/Attention_(machine_learning)

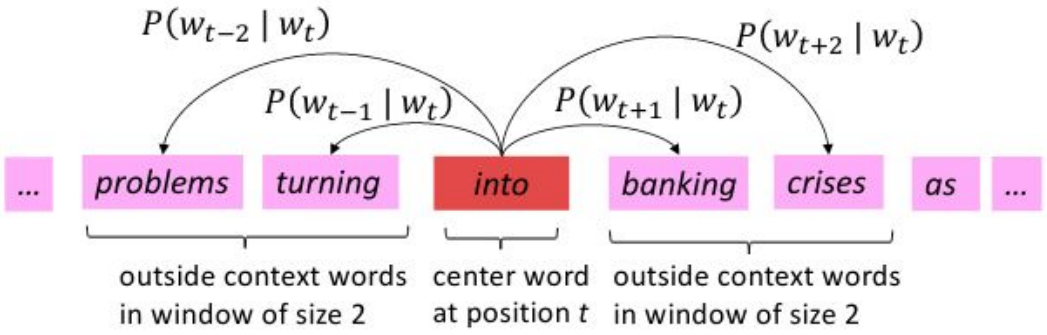
Tokenizing

- Tokenizing - preprocessing data to represent text as finite dictionary of words
- Common words get tokens, uncommon words are built from other tokens, non-word tokens are used for structure and model inputs
- Typical dictionary sizes are in the 10,000-50,000 range
- Different strategies - BytePair Encoding (BPE), WordPiece, SentencePiece

Word Embeddings

- An n-dimensional representation of the word's meaning
- Historically derived from simple probabilistic models that predict a word given context
- One example: Word2Vec (2013)
- Figures are from a low dimensional example

$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$



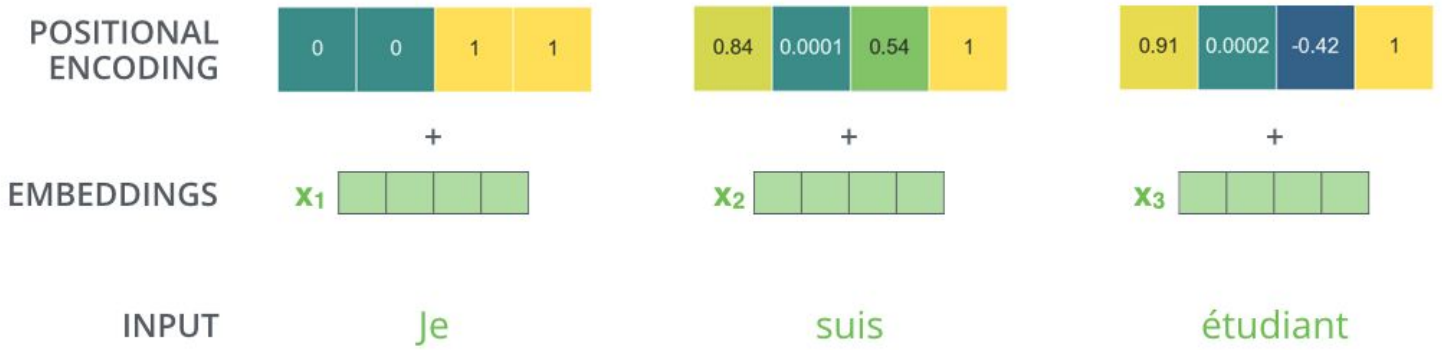
https://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture01-wordvecs1.pdf

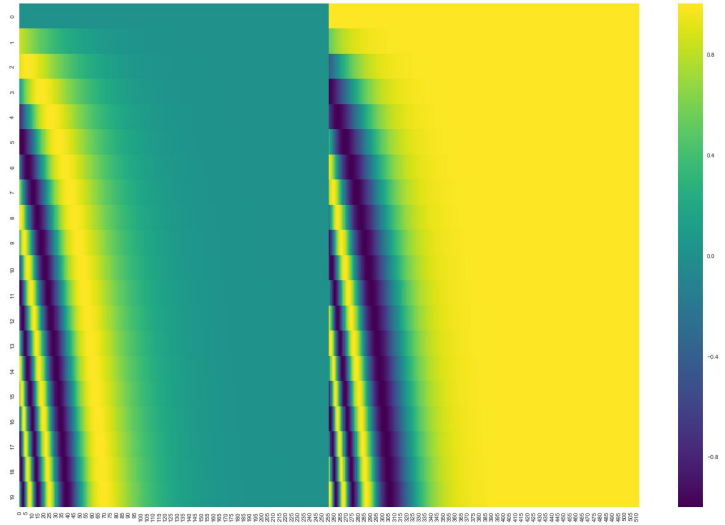
Michigan Tech

https://projector.tensorflow.org/

Positional Embeddings

- Values to include representation of location in input are added
  - Current standard is sine based
- A current topic of research.
  - Alternate positional embedding methods
  - Are they needed at all?
  - ALiBi - a modification of positional embedding





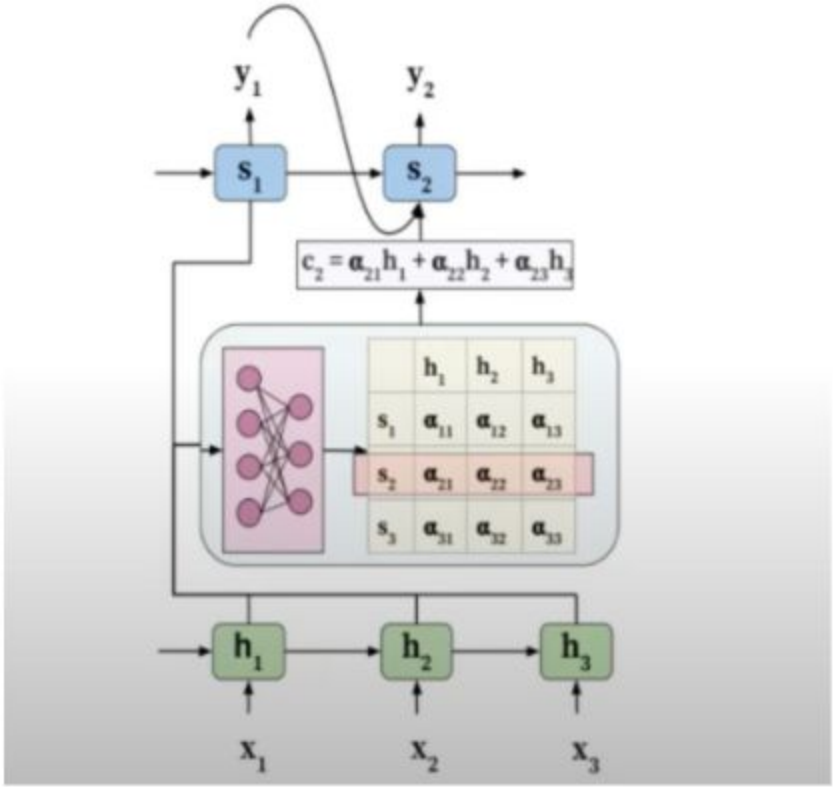
A real example for 20 tokens (rows) with a 512 embedding size (columns) using sine and cosine

https://jalammar.github.io/illustrated-transformer/

Previous Work

- Pre-2014 - RNN, LSTM, GRU
  - RNNs struggled with long sequence data
  - Could not handle long sequence lengths (forgetful)
    - Example on next slide
- Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau, 2014)
  - Landmark Paper
  - Proposed New Architecture:
    - bidirectional RNN as an encoder and an RNN decoder, with an attention mechanism in between
  - Proposed initial idea of attention
  - Model gave attention to particular hidden states when decoding each word
    - Learned which words to give attention to during translation for each word of decoder
  - Still required on RNN

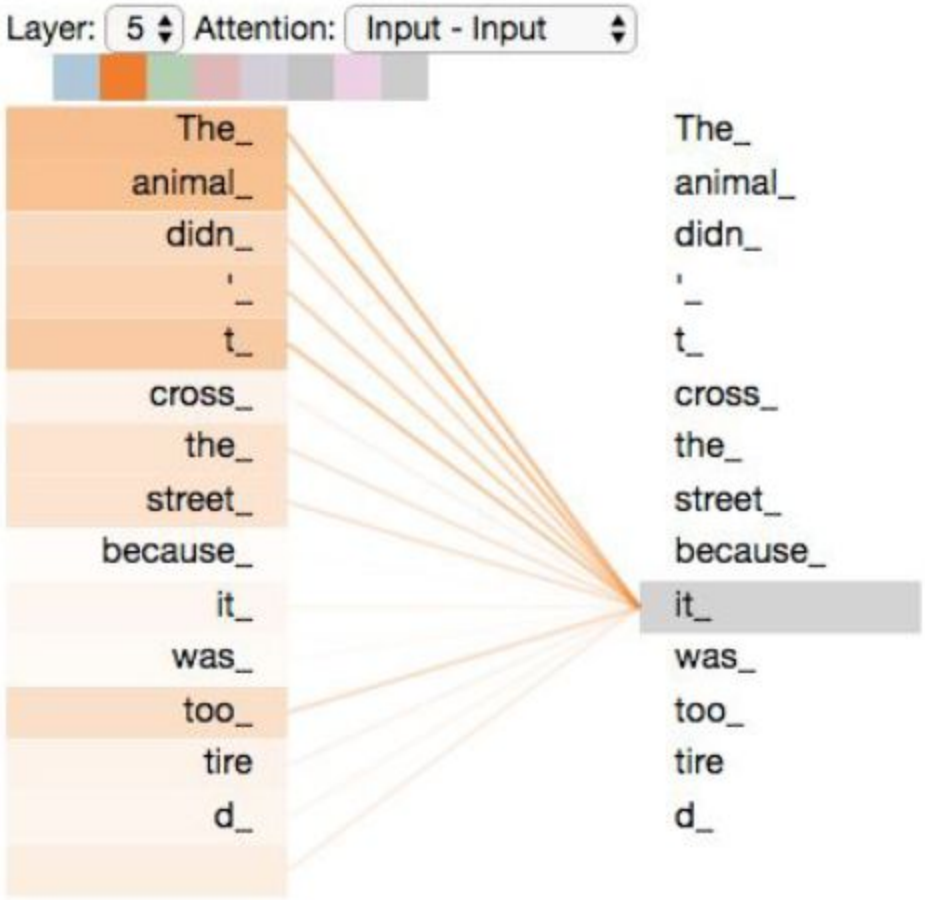
Encoder-decoder model with Attention



Michigan Tech

Self-Attention

- Each element attends to every other element
- It is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence
- Each element becomes query, key, and value from the input embeddings by multiplying by a weight matrix



https://jalammar.github.io/illustrated-transformer/

Idea Behind Attention

Goal: Learn (differentiable) how to pick relevant information from input data

1. Create three vectors from each of the encoder's input values (query, key, value)
2. Calculate a score for how much to focus on each part of the input when we encode words at specific positions
3. How?
    - Actual math on the next slide, but the idea is to select a value (referenced by a key) relevant to a query (what trying to pull from input)

- **From Paper - "A mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key" - (Vaswani, 2017)**

key	value
name	Quinn
position	quarterback
Handedness	right

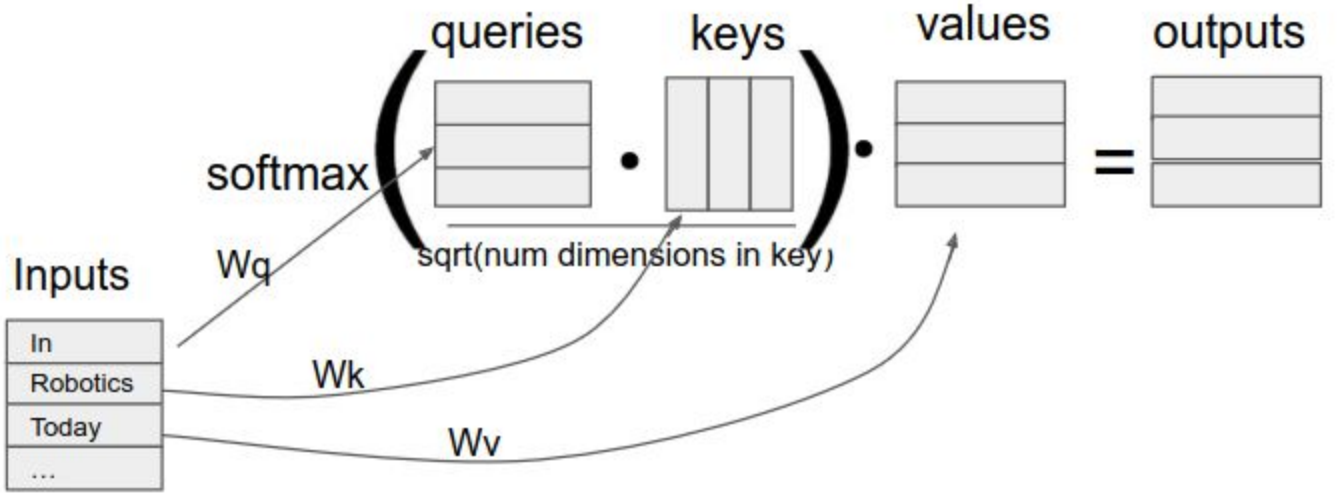
Keys and values are actually word embeddings, not words

Scaled Dot Product Attention

- Key algorithm in transformers
- Attention weights: how likely each query matches key (used for weighted sum of values)
- Dividing by d_k makes algorithm easier to train
  - prevents binary prediction of one 1 and a whole bunch of zeros
  - helps the gradients flow
- then can take loss, backpropagate, update the weights and values
- NOT stepping through a sequence like with a RNN

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

- **input consists of queries and keys of dimension dk, and values of dimension d**
- **Queries packed into matrix Q**
- **The keys and values are also packed together into matrices K and V .**



Michigan Tech

Multi-Head Attention

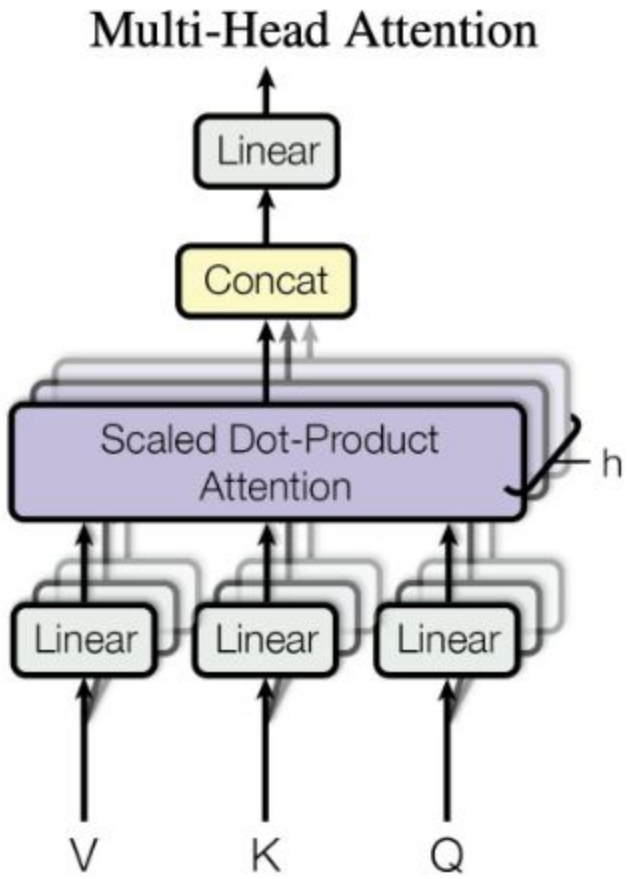
- Idea:
  a. Stack linear layers (weight matrices without biases) that are independent each for keys, queries, values.
  b. Concatenate output of attention heads to form (plus non-linearity) output layer
- Why?
  a. Allows for model to focus on different positions
  b. Gives attention layer multiple “representation subspaces”
  c. No longer need to oversaturate one attention mechanism

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$h = 8$ parallel attention layers, or heads.

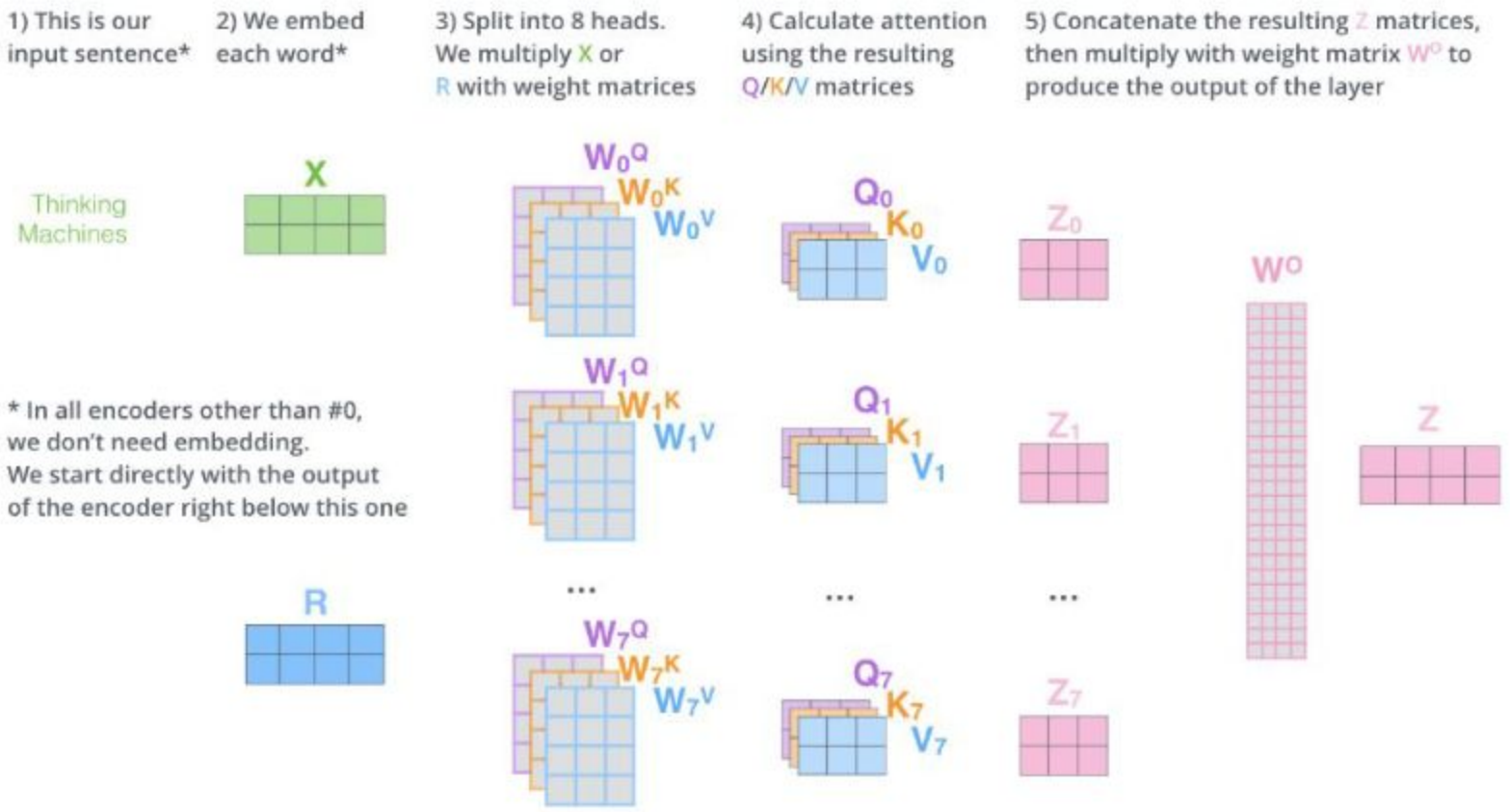
Learnable parameter matrices



Multi-Head Attention Continued

Key points:

- We calculate 8 different attention heads, but we need to combine them
- Attention heads are independent of each other



https://jalammar.github.io/illustrated-transformer/

Positional Encoding

- Need for information about the position and order of tokens in a sequence
- Positional Embedding: Vector that represents position of each token
  - Element wise addition the position embedding to the word embedding vector
  - Positional embedding be fixed or learned

"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PEpos+k can be represented as a linear function of PEpos."

For every sine-cosine pair corresponding to frequency ω_k, there is a linear transformation $M \in \mathbb{R}^{2\times 2}$ (independent of t) where the following equation holds:

=>

$$M \cdot \begin{bmatrix} \sin(\omega_k . t) \\ \cos(\omega_k . t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k . (t+\phi)) \\ \cos(\omega_k . (t+\phi)) \end{bmatrix}$$

$$M_{\phi,k} = \begin{bmatrix} \cos(\omega_k . \phi) & \sin(\omega_k . \phi) \\ -\sin(\omega_k . \phi) & \cos(\omega_k . \phi) \end{bmatrix}$$

Encoding used in paper

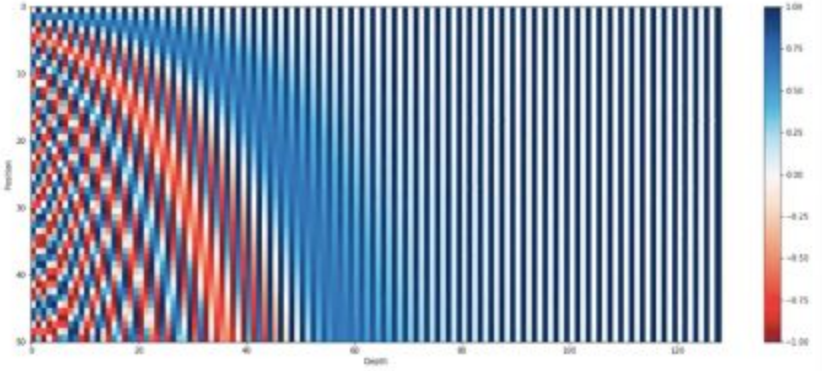
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

pos is the position and i is the dimension.

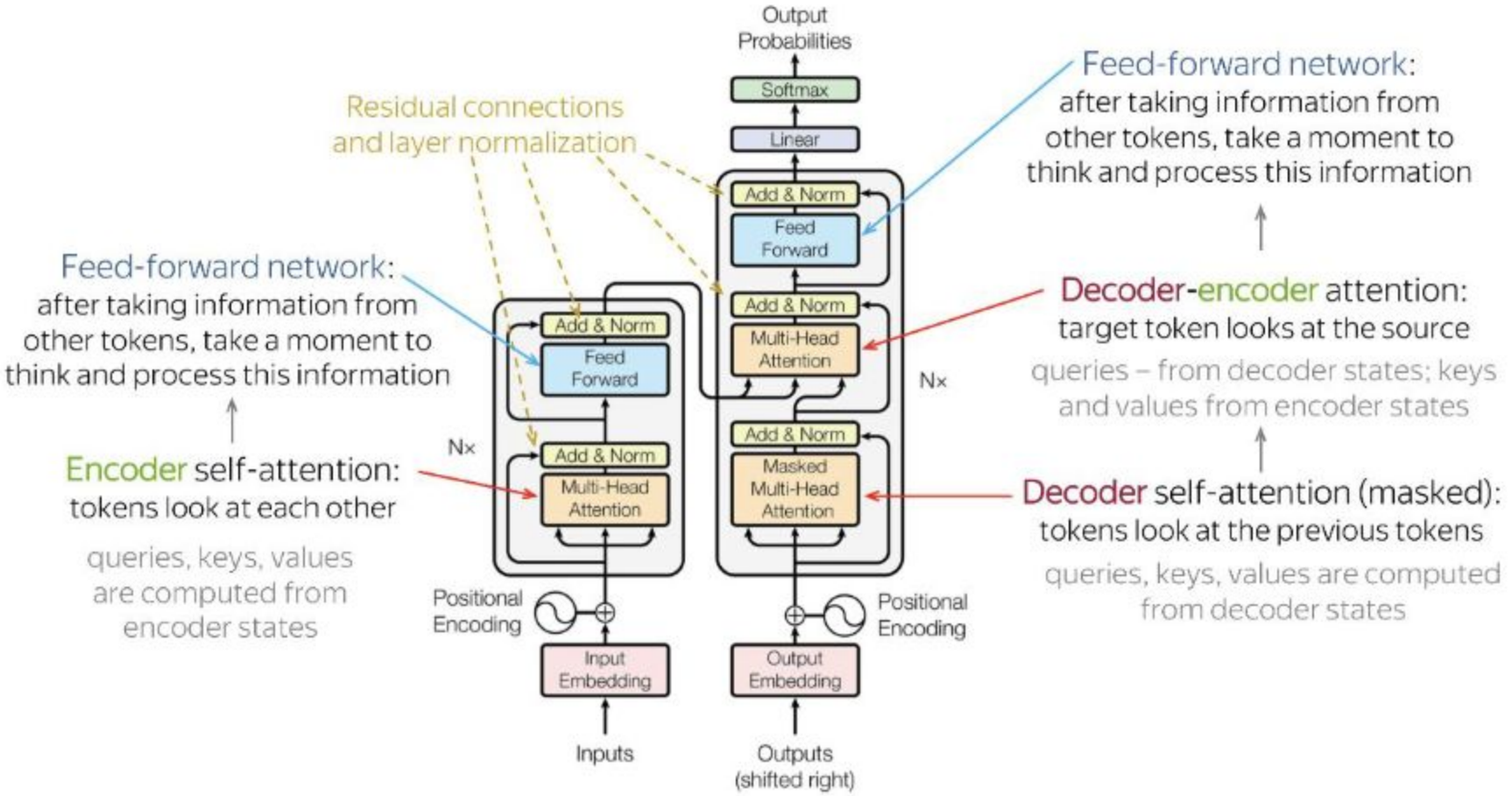
* Sine method:

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \vdots \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d\times 1}$$



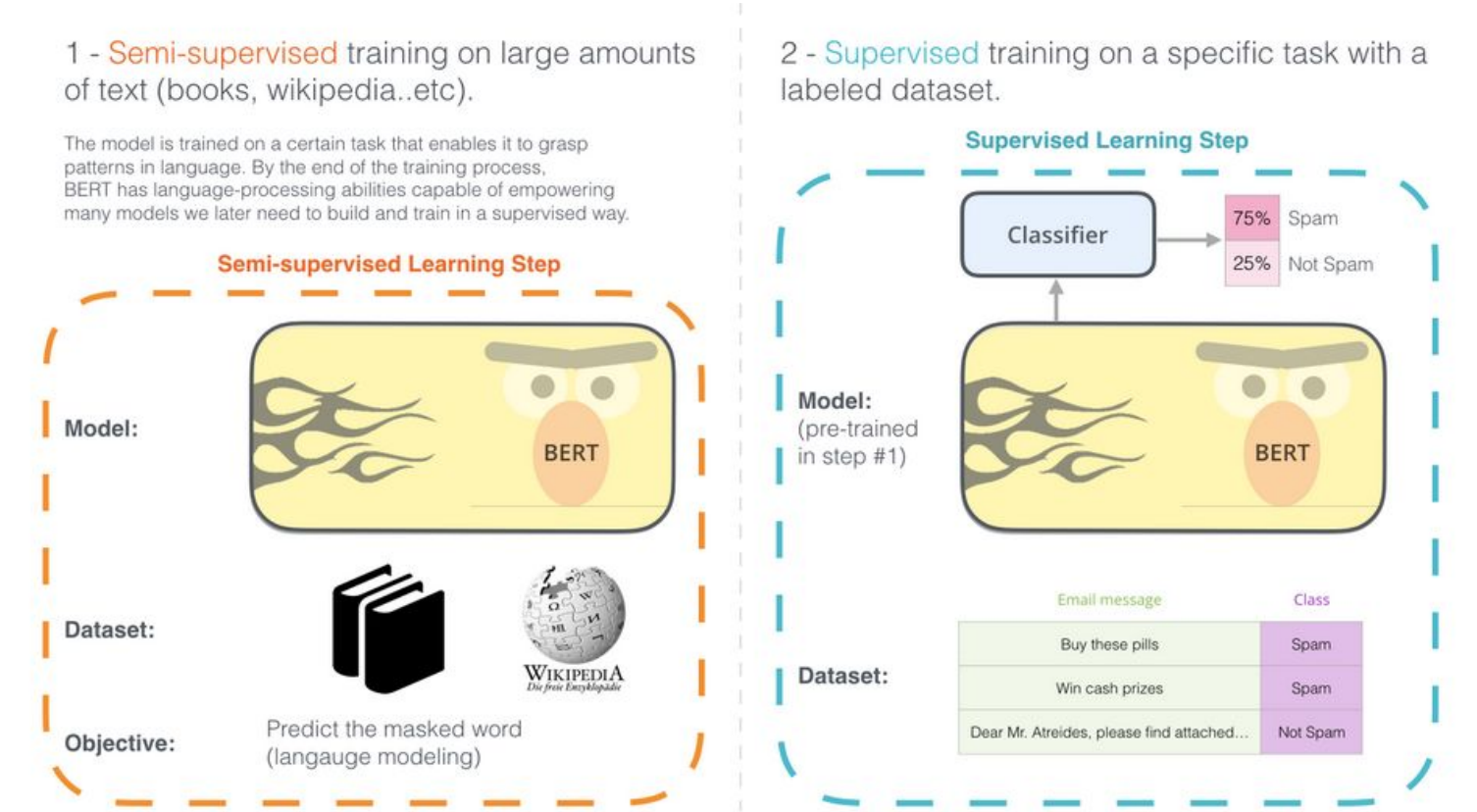
Note: positional embeddings are now learned

Full-Architecture



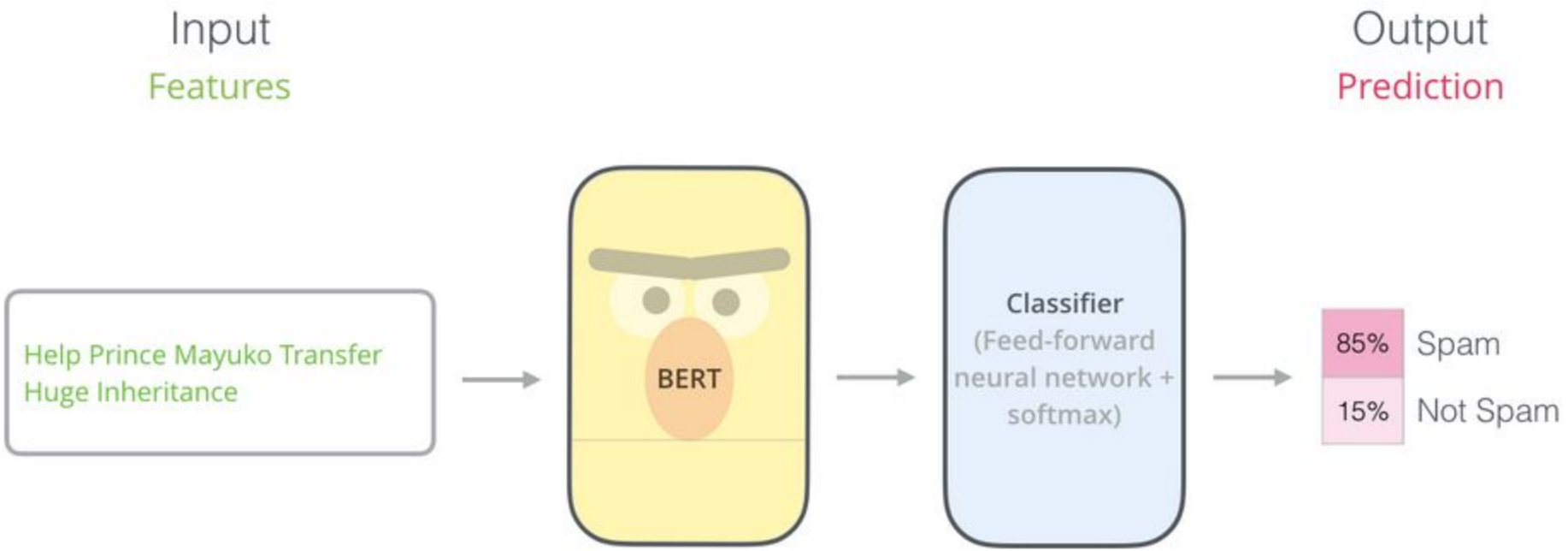
Michigan Tech

BERT

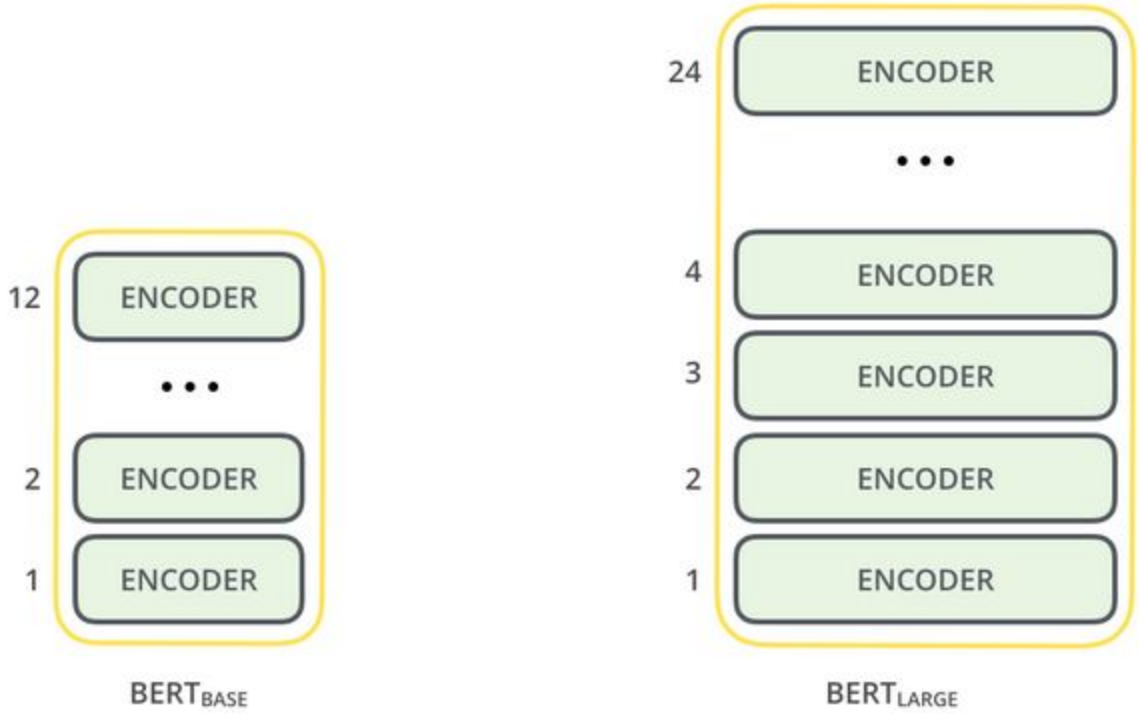


The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2. [Source for book icon].

Sentence classification example



Different model sizes

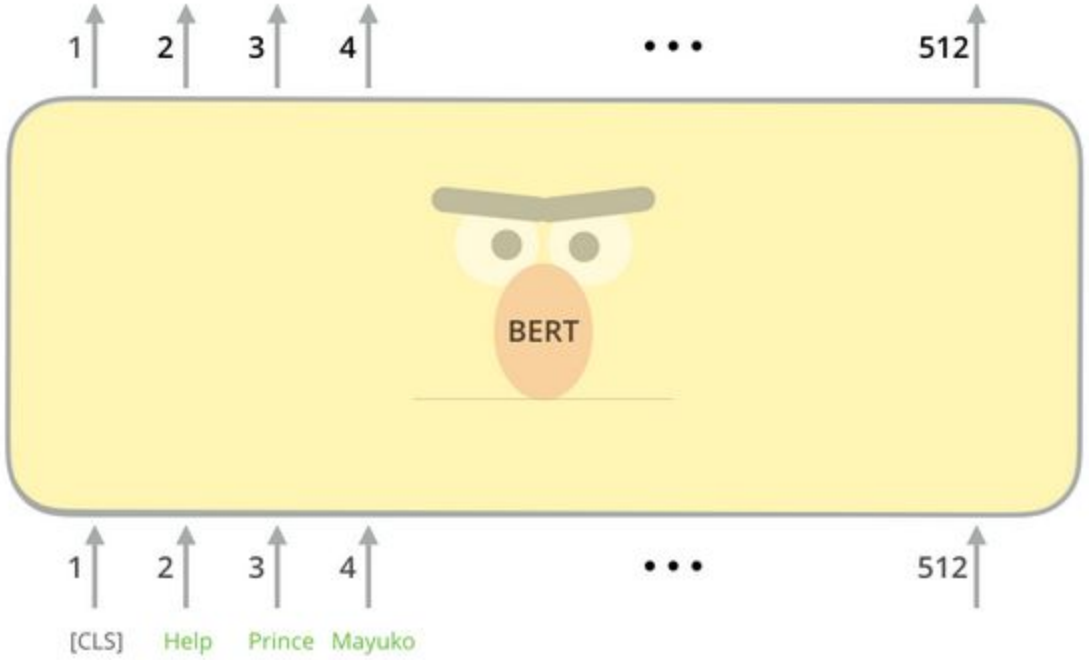


You can download all 24 from here, or individually from the table below:

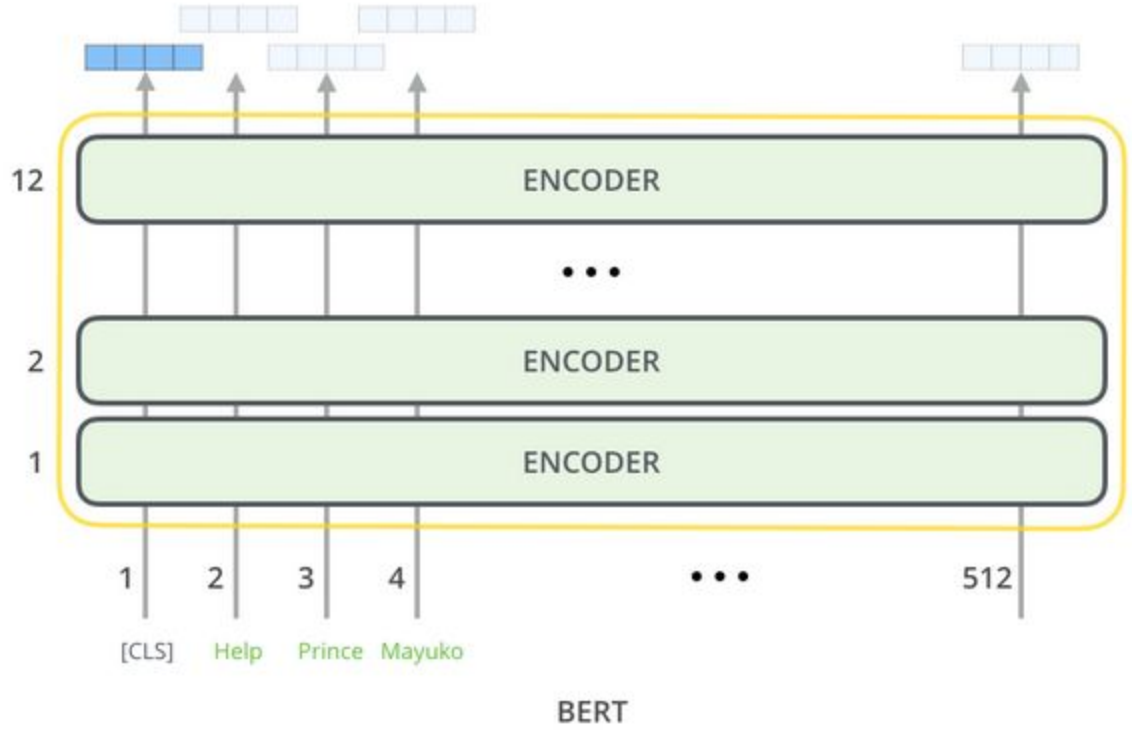
	H=128	H=256	H=512	H=768
L=2	**2/128 (BERT-Tiny)**	2/256	2/512	2/768
L=4	4/128	**4/256 (BERT-Mini)**	**4/512 (BERT-Small)**	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	**8/512 (BERT-Medium)**	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	**12/768 (BERT-Base)**

Model	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI(v2)
BERT-Tiny	64.2	0.0	83.2	81.1/71.1	74.3/73.6	62.2/83.4	70.2	70.3	81.5
BERT-Mini	65.8	0.0	85.9	81.1/71.8	75.4/73.3	66.4/86.2	74.8	74.3	84.1
BERT-Small	71.2	27.8	89.7	83.4/76.2	78.8/77.0	68.1/87.0	77.6	77.0	86.4
BERT-Medium	73.5	38.0	89.6	86.6/81.6	80.4/78.4	69.6/87.9	80.0	79.1	87.7

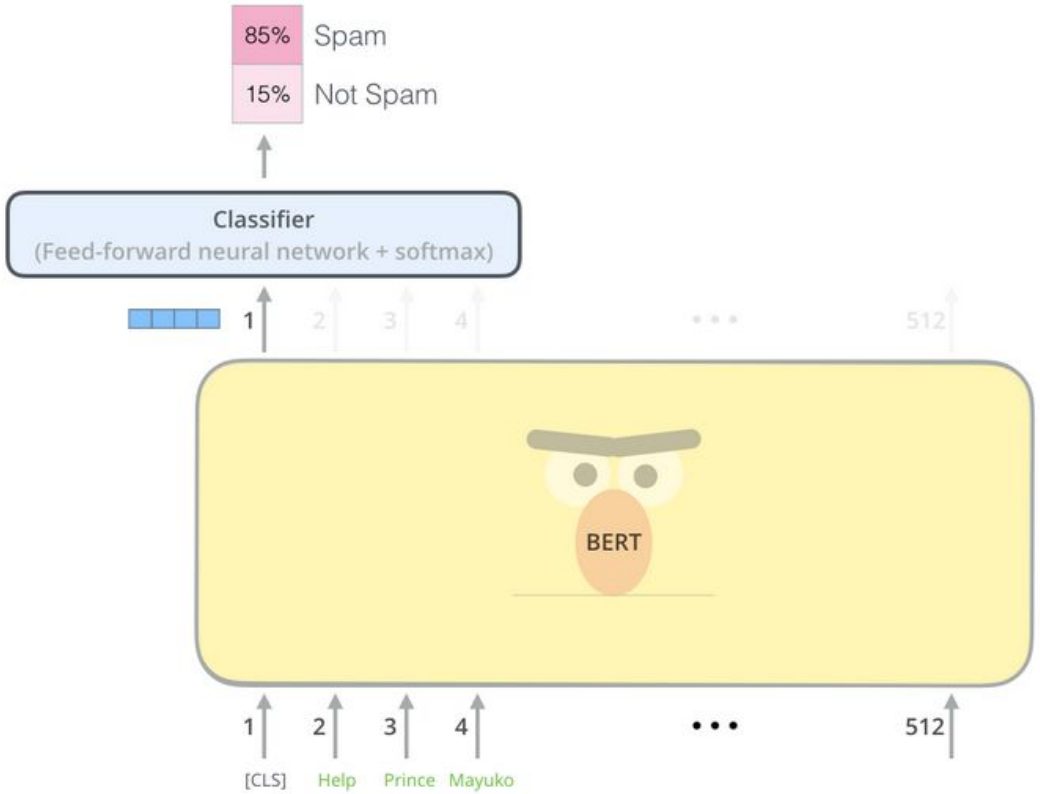
Data passthrough for BERT



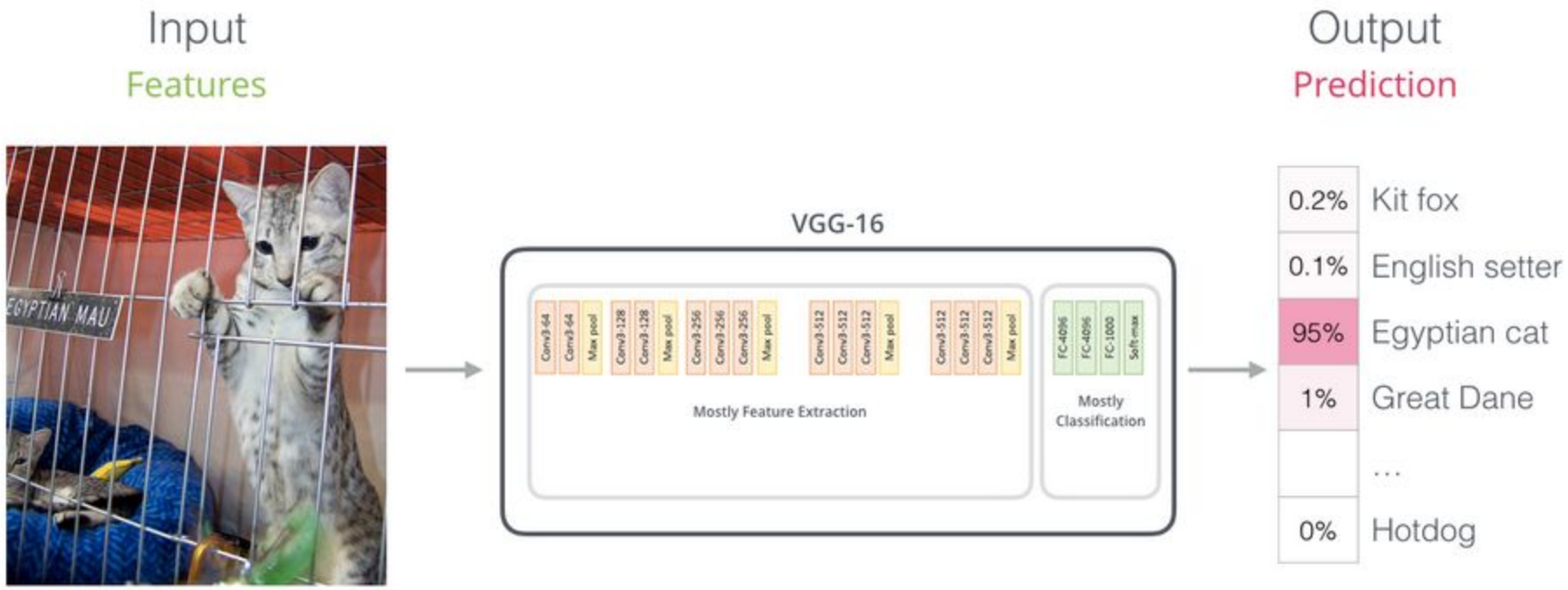
Data passthrough for BERT



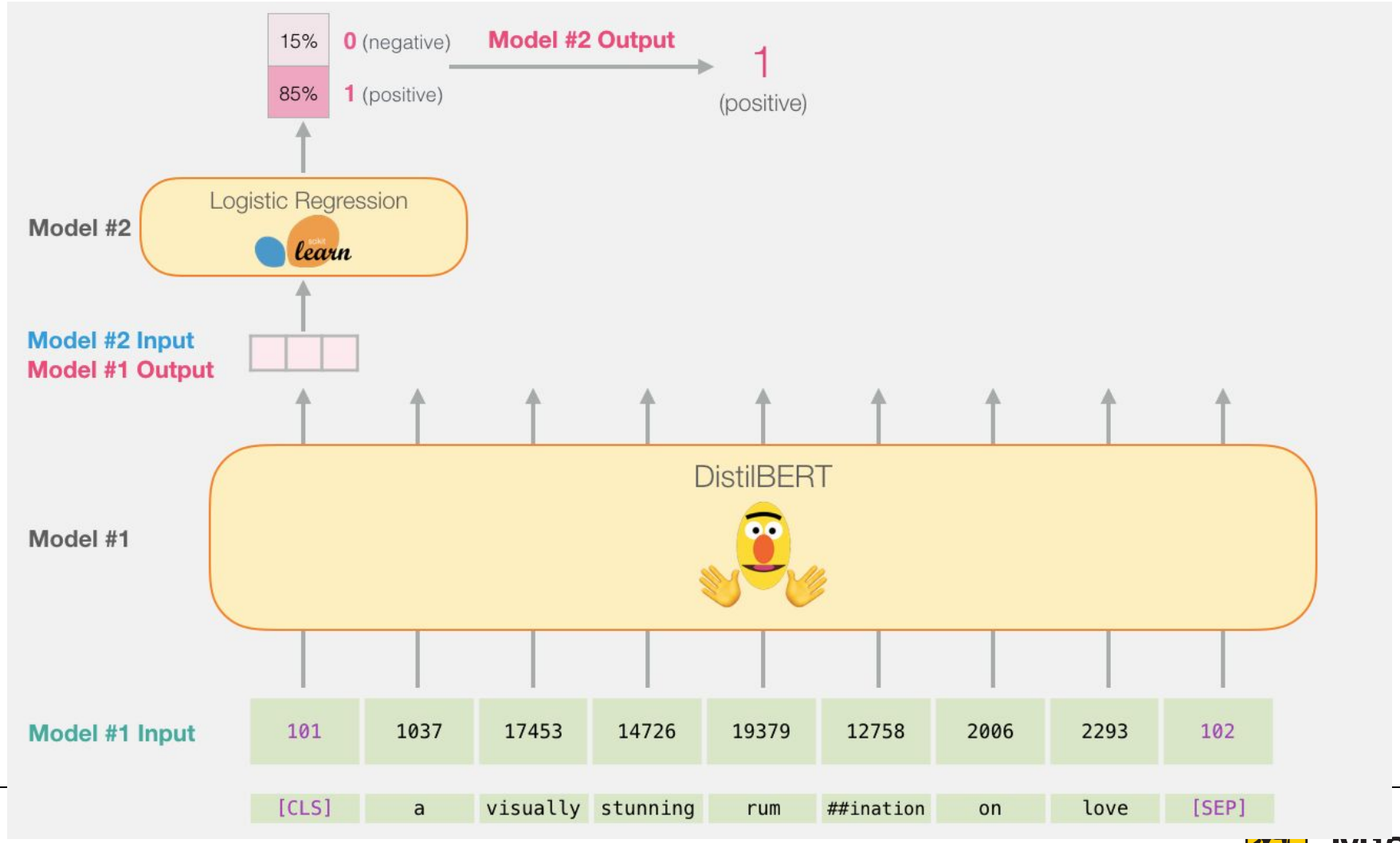
Transfer learning with BERT



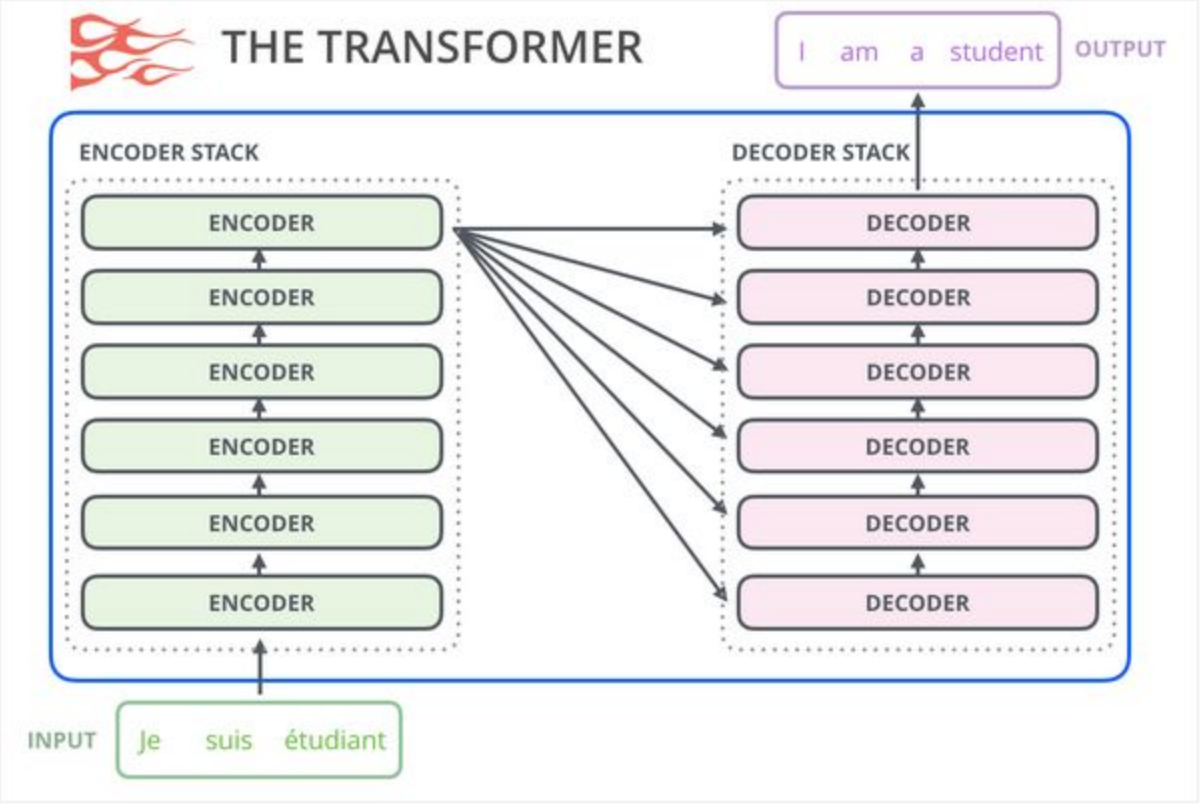
Similarity to transfer learning with CNN

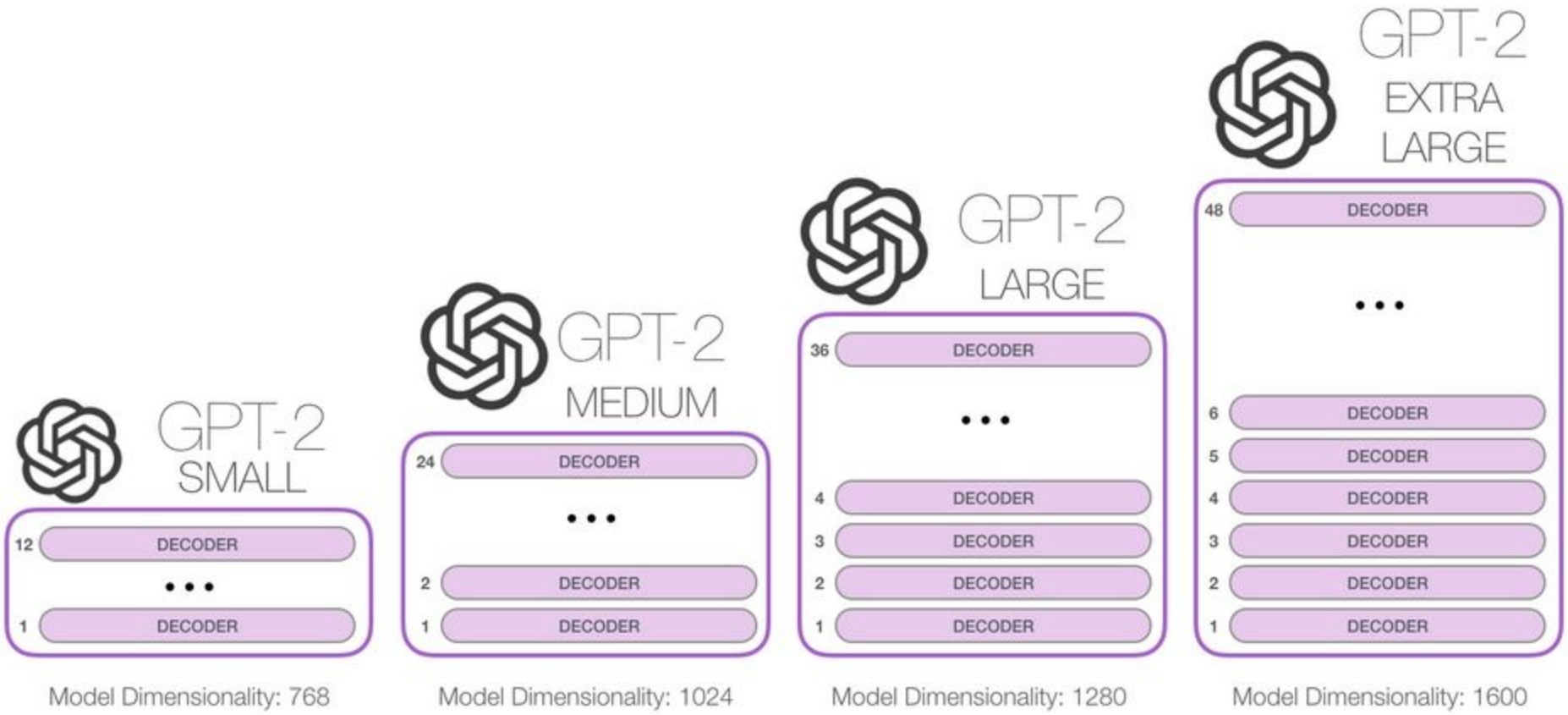


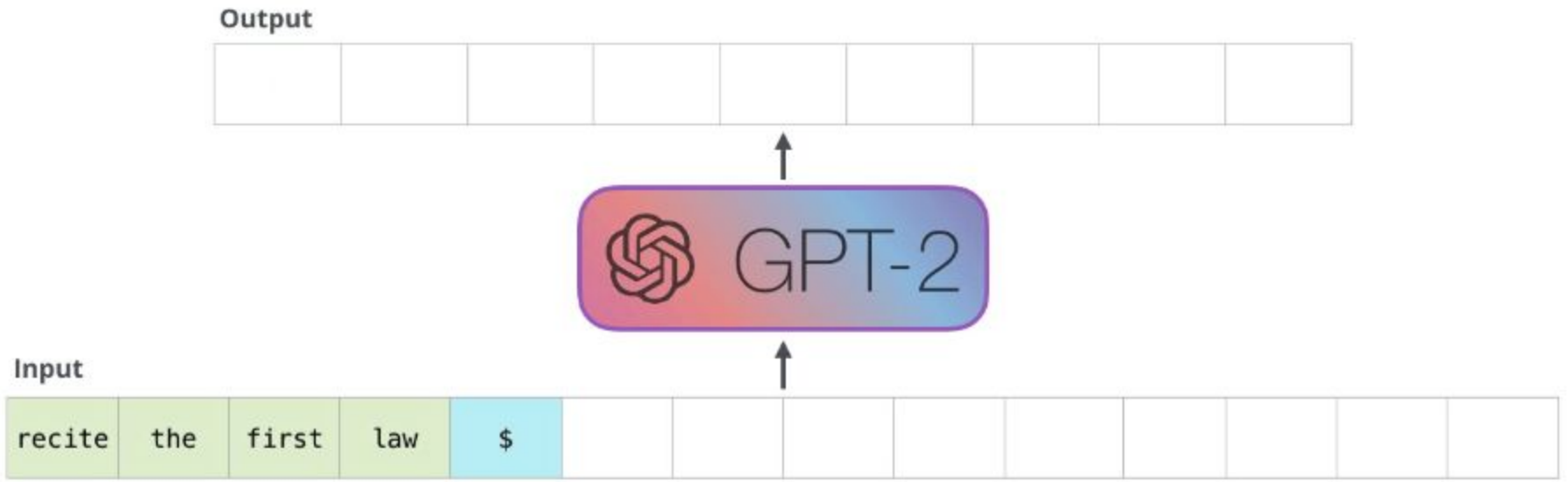
BERT as featurizer for other model

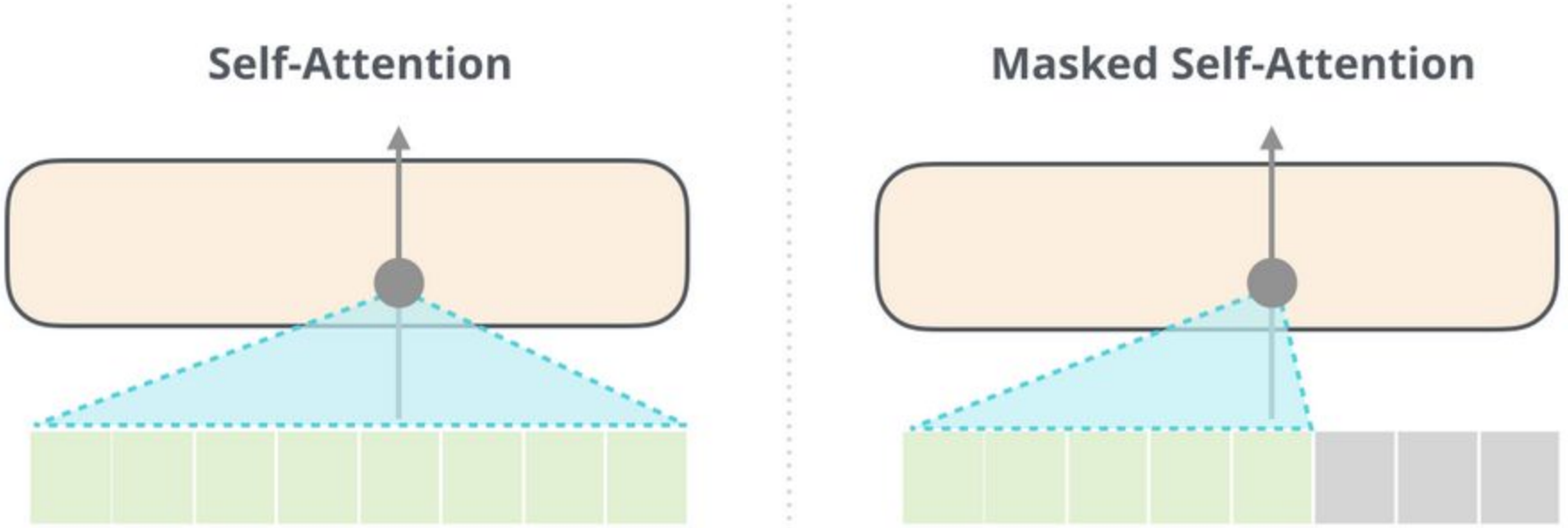


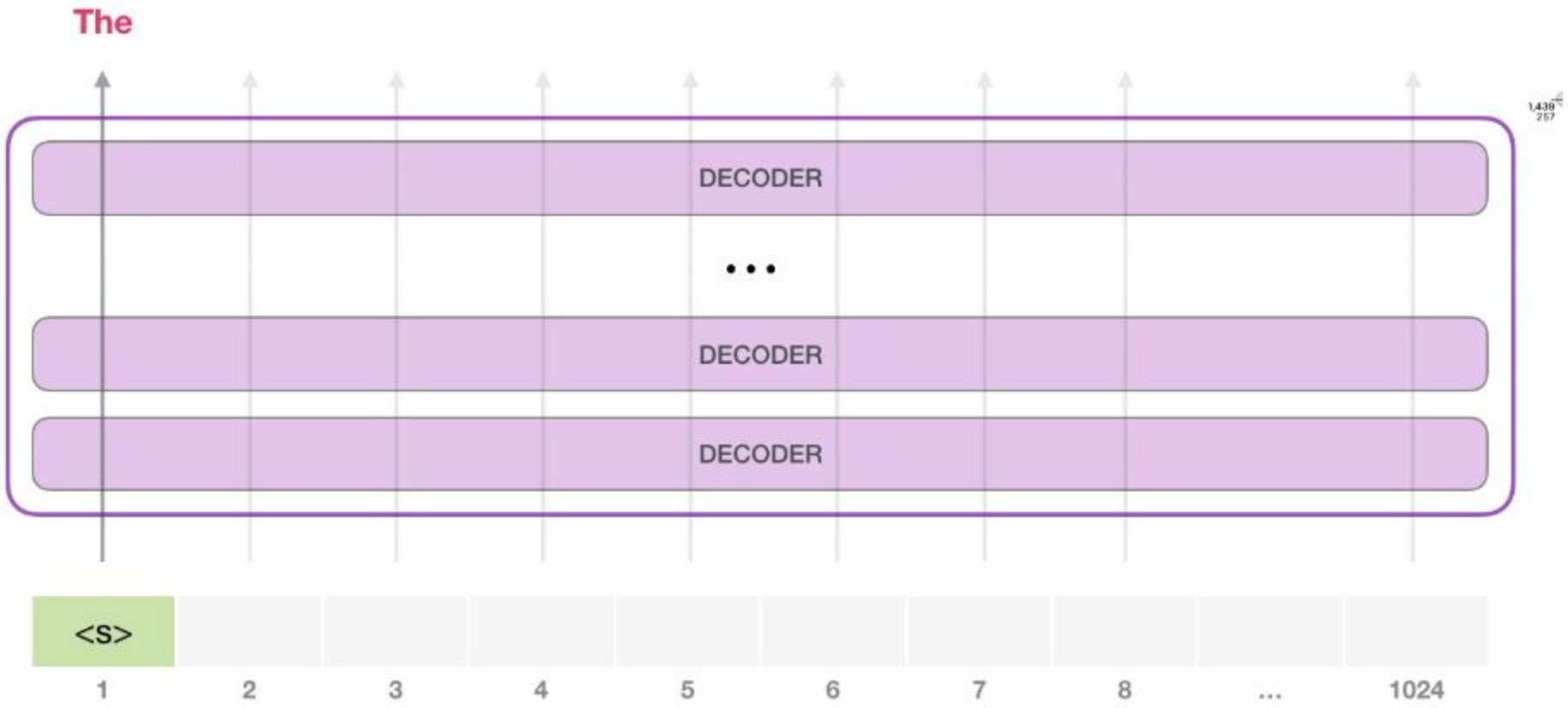
Encoder-Decoder

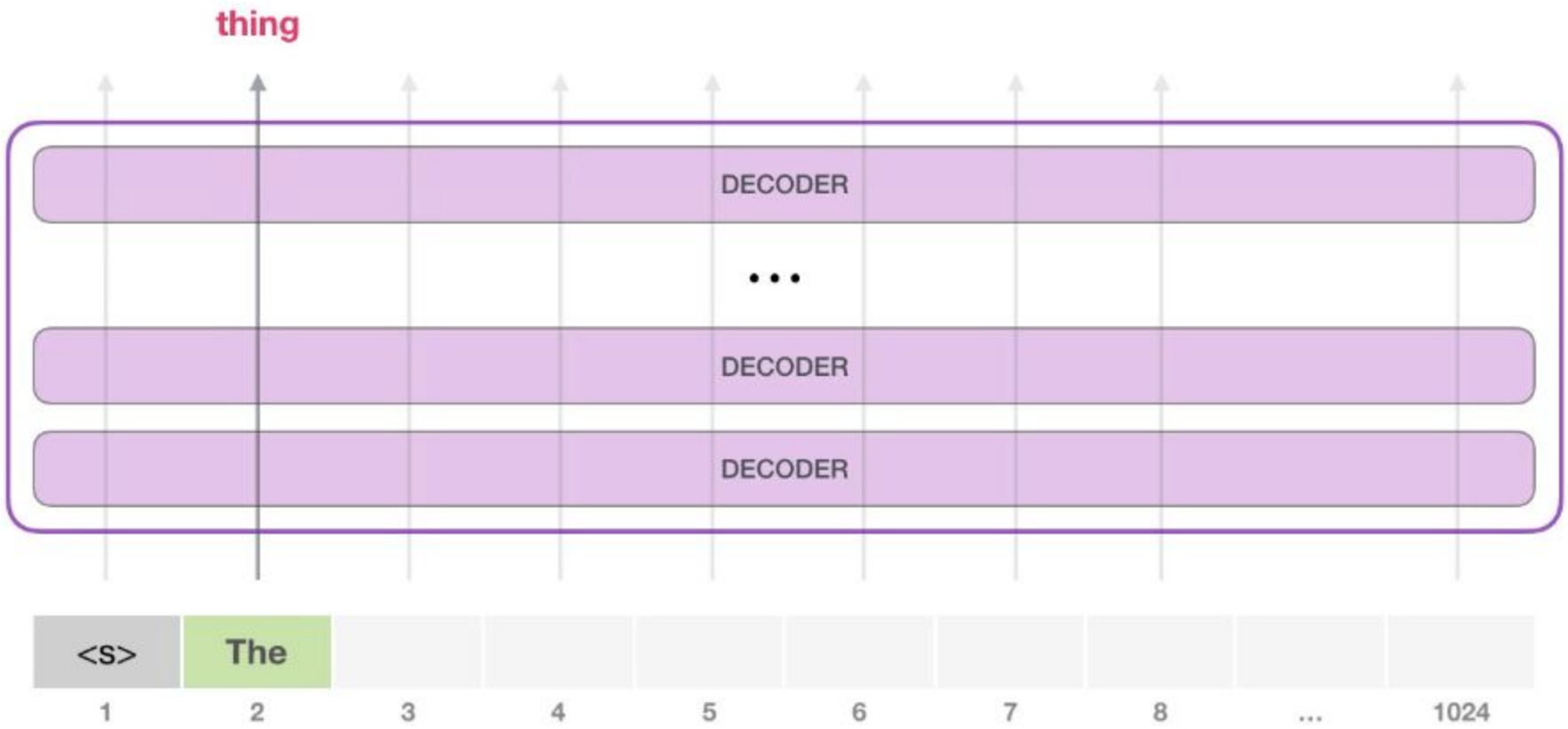


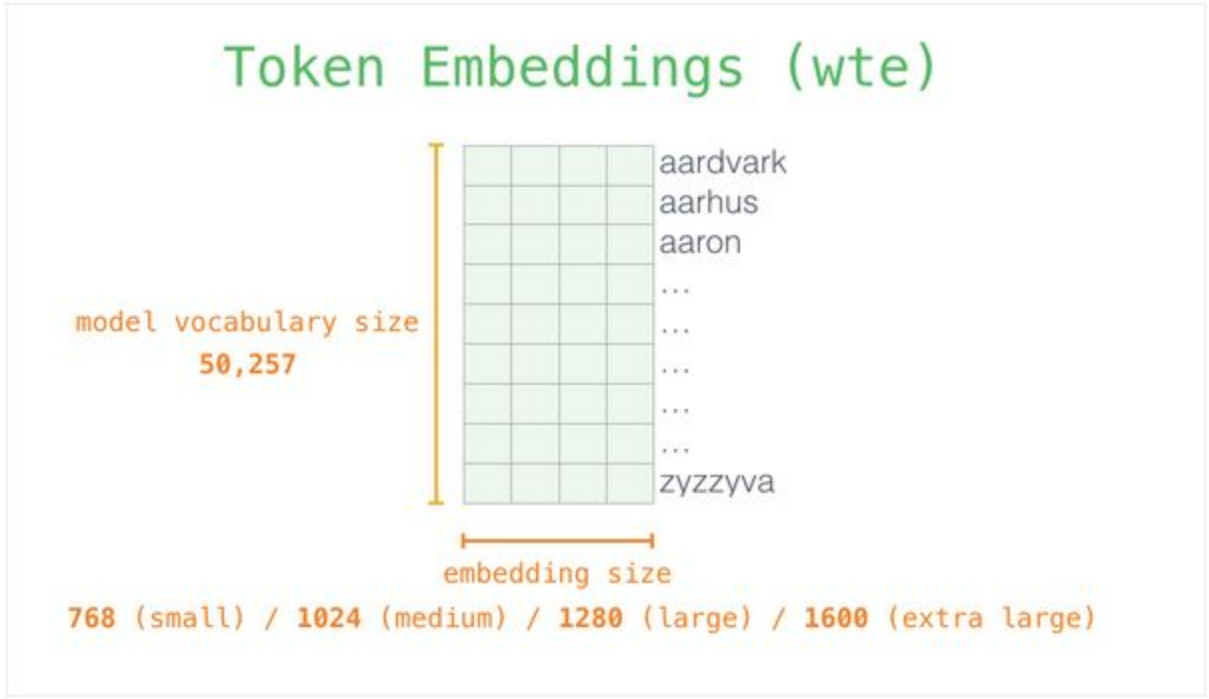

GPT-2
SMALL
12
DECODER
1
DECODER
Model Dimensionality: 768
GPT-2
MEDIUM
24
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1024
GPT-2
LARGE
36
DECODER
4
DECODER
3
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1280
GPT-2
EXTRA
LARGE
48
DECODER
6
DECODER
5
DECODER
4
DECODER
3
DECODER
2
DECODER
1
DECODER
Model Dimensionality: 1600


Output
GPT-2
Input
recite
the
first
law
$


Self-Attention
Masked Self-Attention

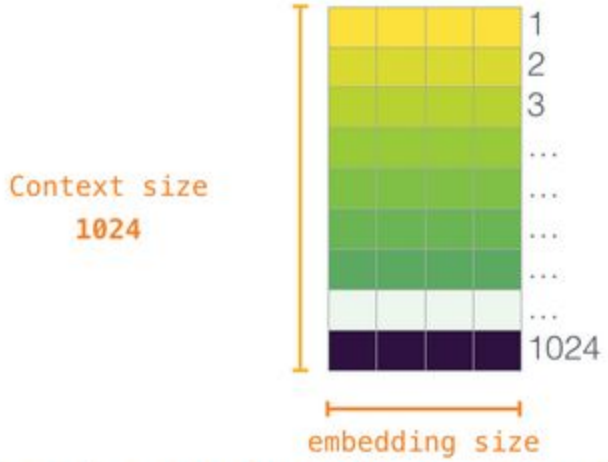

The
DECODER
DECODER
DECODER
<S>
1
2
3
4
5
6
7
8
...
1024


thing
DECODER
• • •
DECODER
DECODER
<s>
The
1
2
3
4
5
6
7
8
...
1024



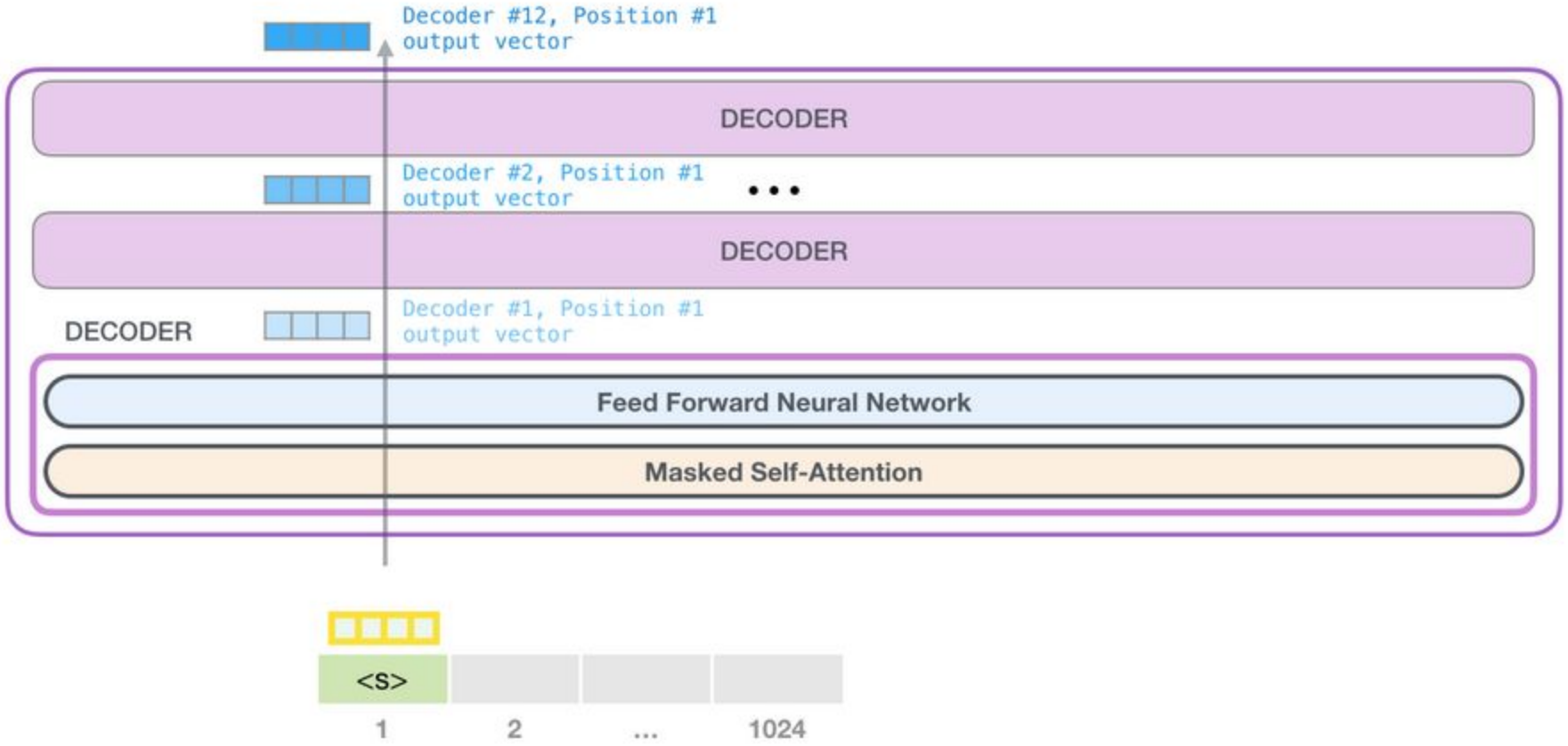
Each row is a word embedding: a list of numbers representing a word and capturing some of its meaning. The size of that list is different in different GPT2 model sizes. The smallest model uses an embedding size of 768 per word/token.

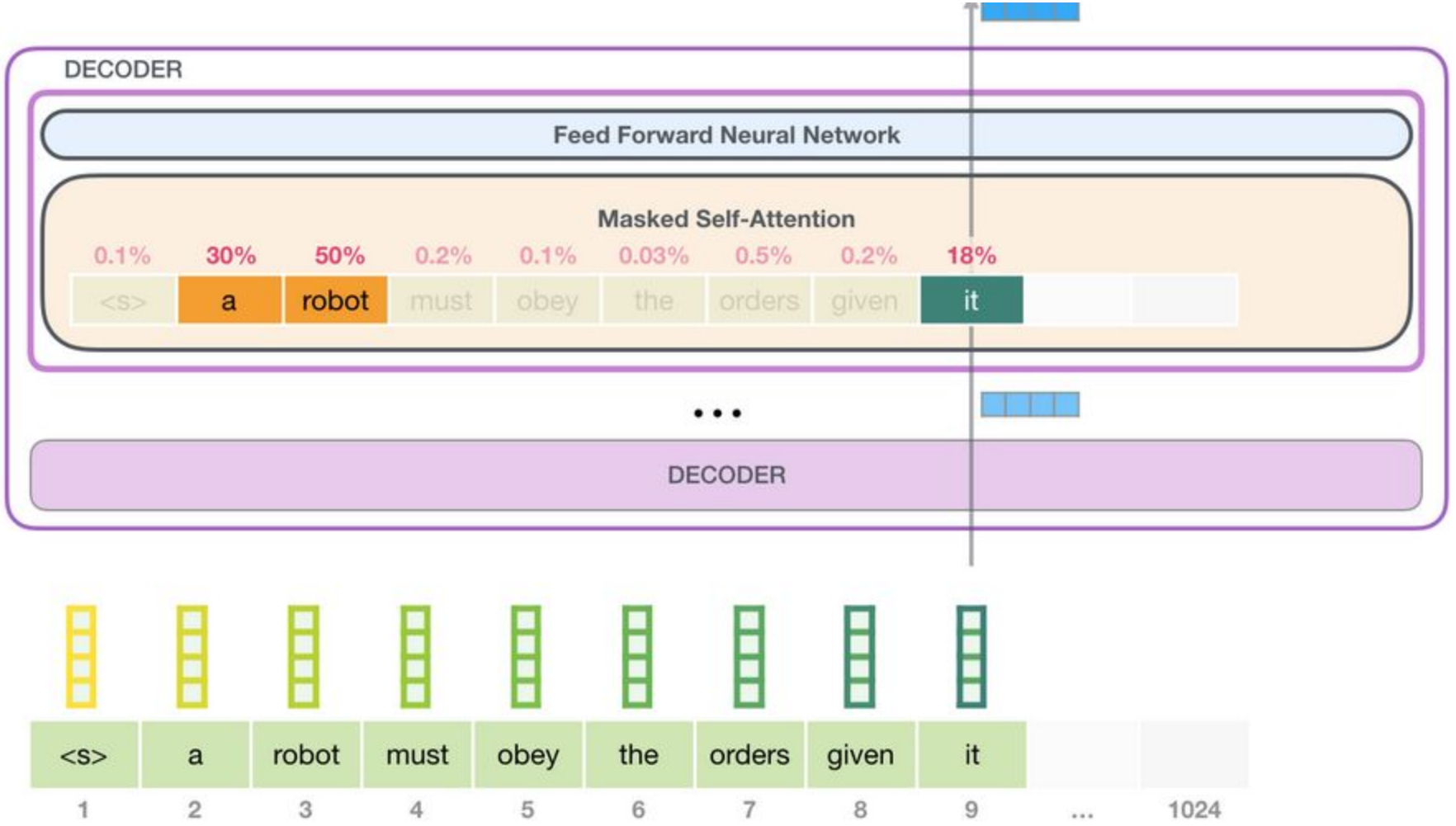
Positional Encodings (wpe)

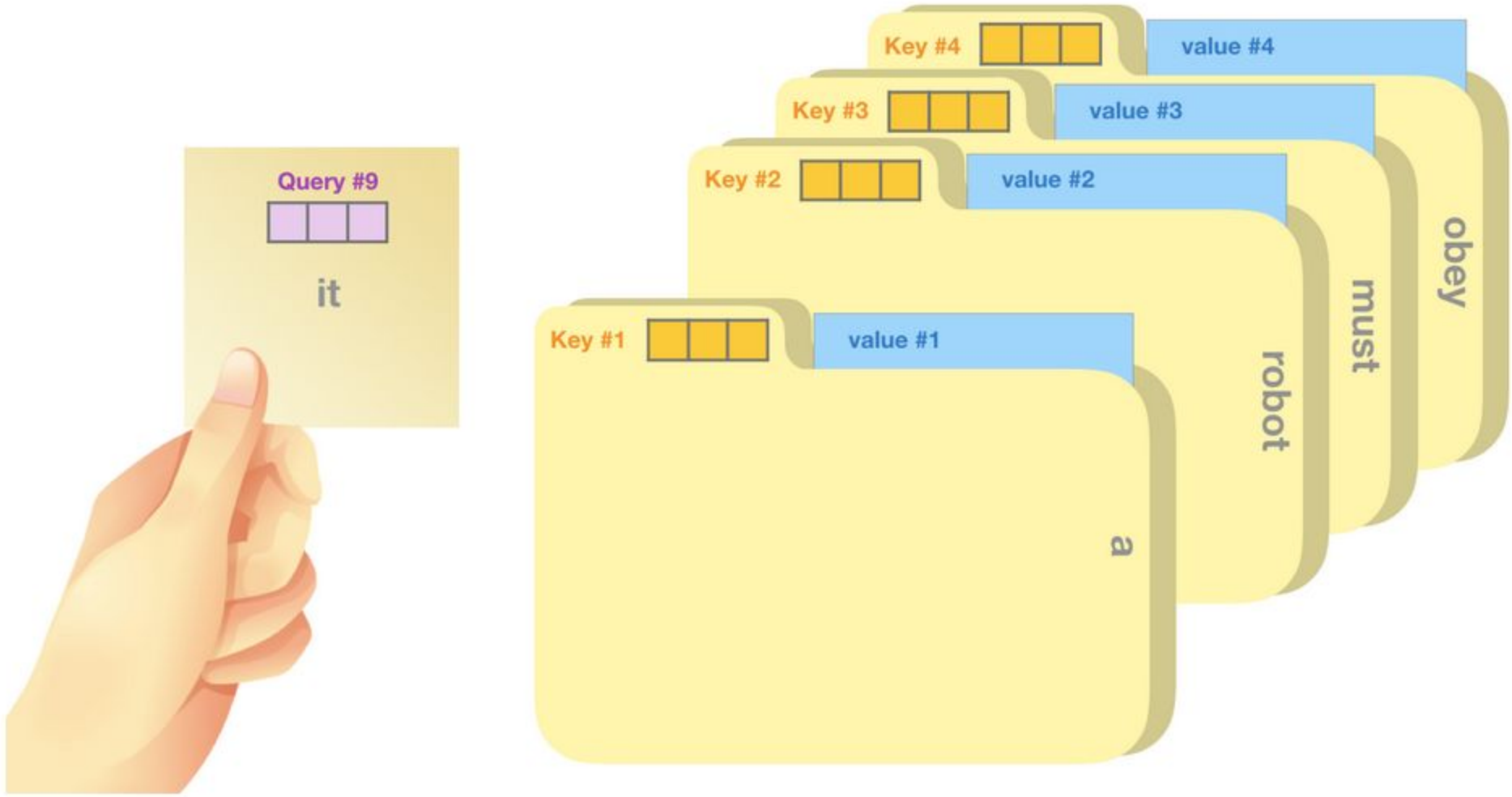


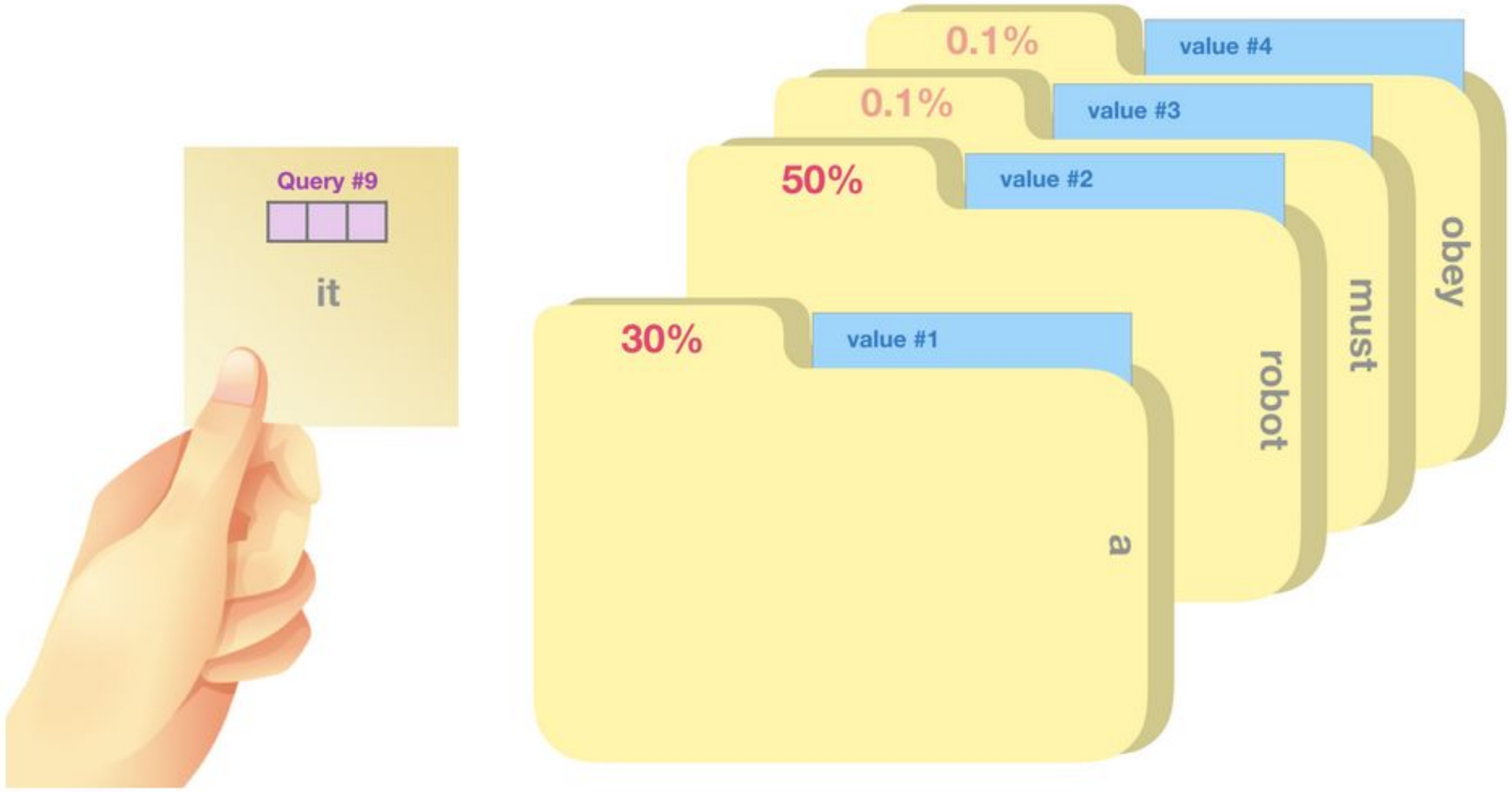
768 (small) / **1024** (medium) / **1280** (large) / **1600** (extra large)


DECODER
DECODER
=
Positional encoding for token #1
+
Token embedding of <s>
<s>
1
2
...
1024
Token Embeddings
Positional Encodings

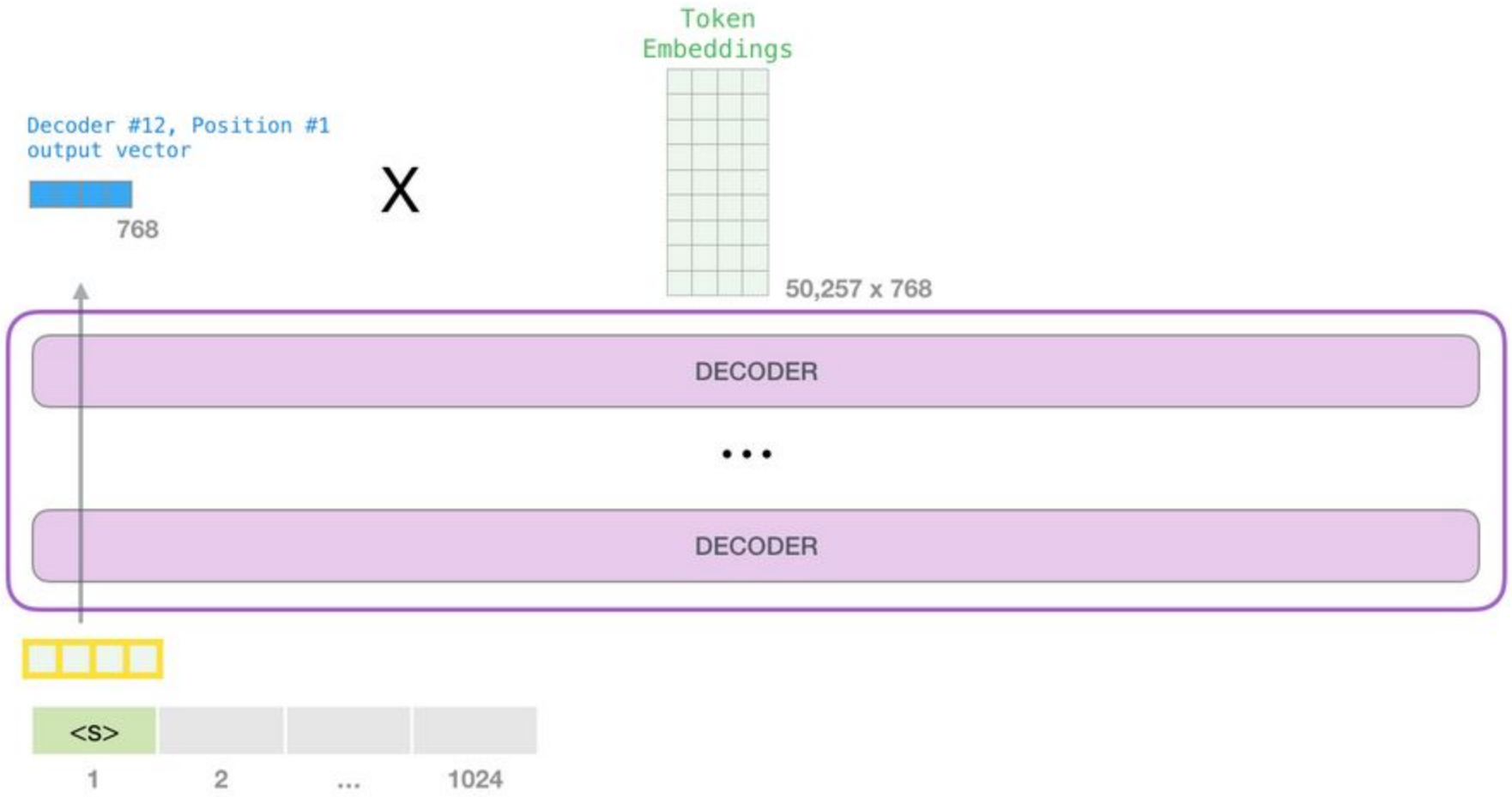

Decoder #12, Position #1
output vector
DECODER
Decoder #2, Position #1
output vector
DECODER
DECODER
Decoder #1, Position #1
output vector
Feed Forward Neural Network
Masked Self-Attention
<S>
1
2
...
1024


DECODER
Feed Forward Neural Network
Masked Self-Attention
0.1%
30%
50%
0.2%
0.1%
0.03%
0.5%
0.2%
18%
<s>
a
robot
must
obey
the
orders
given
it
DECODER
<s>
a
robot
must
obey
the
orders
given
it
1
2
3
4
5
6
7
8
9
...
1024

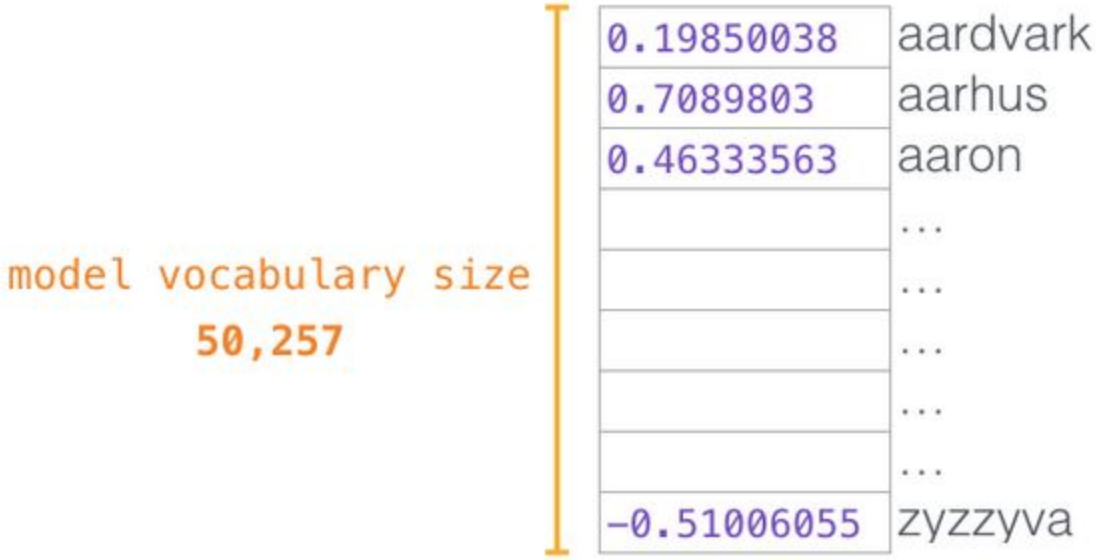

Query #9
it
Key #4
value #4
Key #3
value #3
Key #2
value #2
Key #1
value #1
a
robot
must
obey

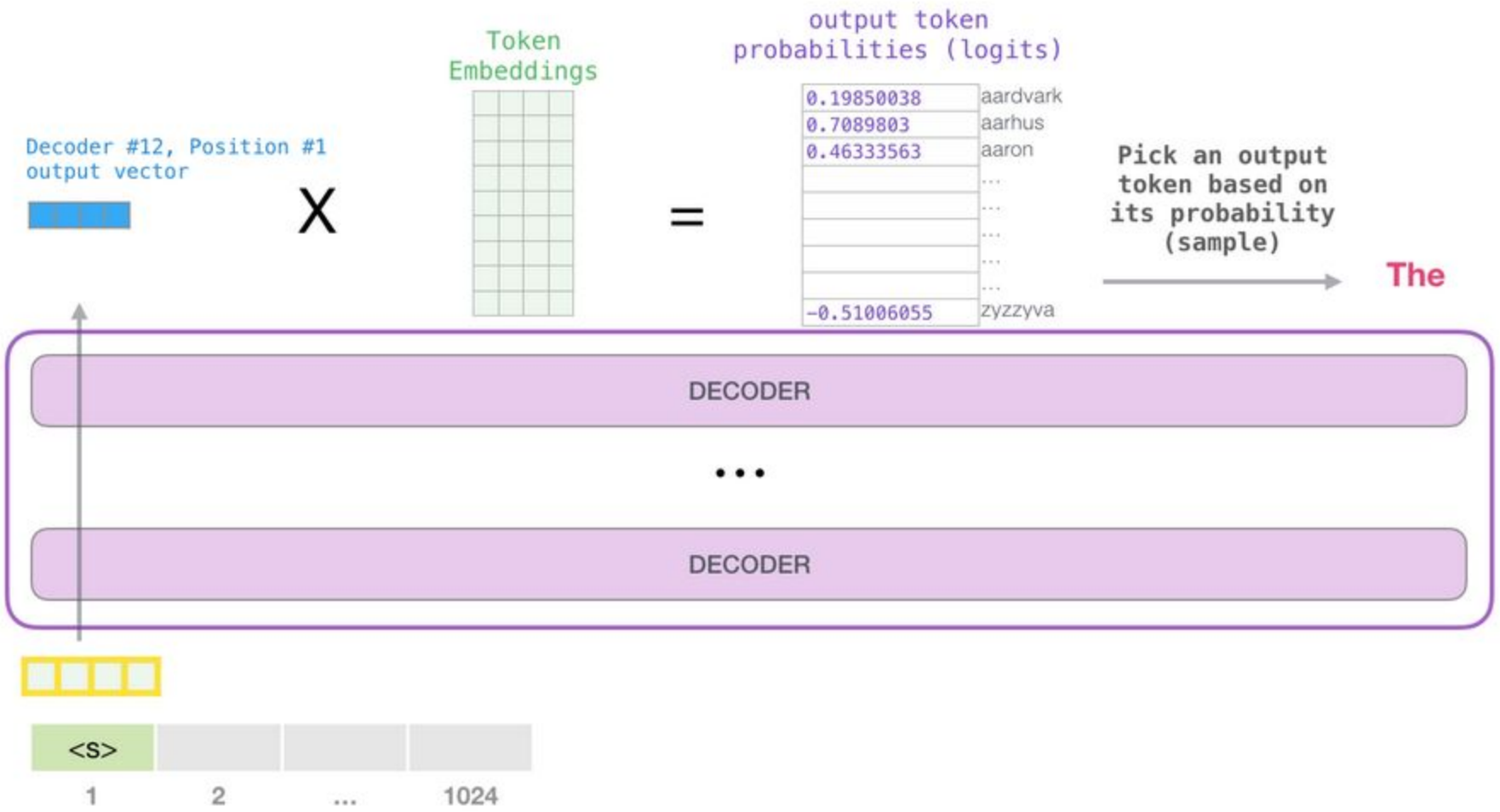

Query #9
it
0.1%
value #4
0.1%
value #3
50%
value #2
30%
value #1
a
robot
must
obey

Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	


Token
Embeddings
Decoder #12, Position #1
output vector
768
X
50,257 x 768
DECODER
DECODER
<s>
1
2
...
1024

output token probabilities (logits)



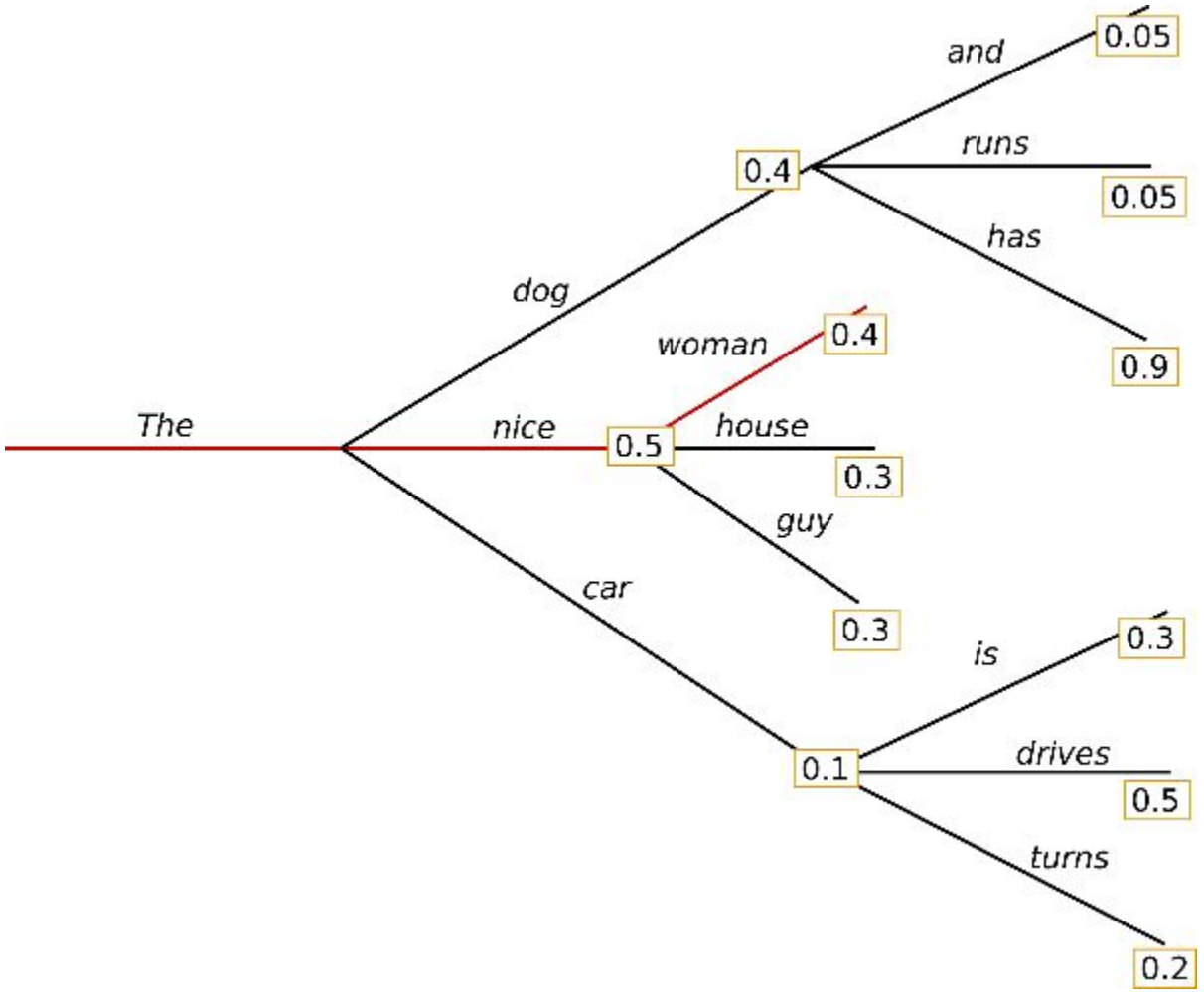

Token Embeddings
output token probabilities (logits)
Decoder #12, Position #1 output vector
X
=
0.19850038 aardvark
0.7089803 aarhus
0.46333563 aaron
-0.51006055 zyzzyva
Pick an output token based on its probability (sample)
The
DECODER
DECODER
<s>
1
2
...
1024

Let’s talk about sampling

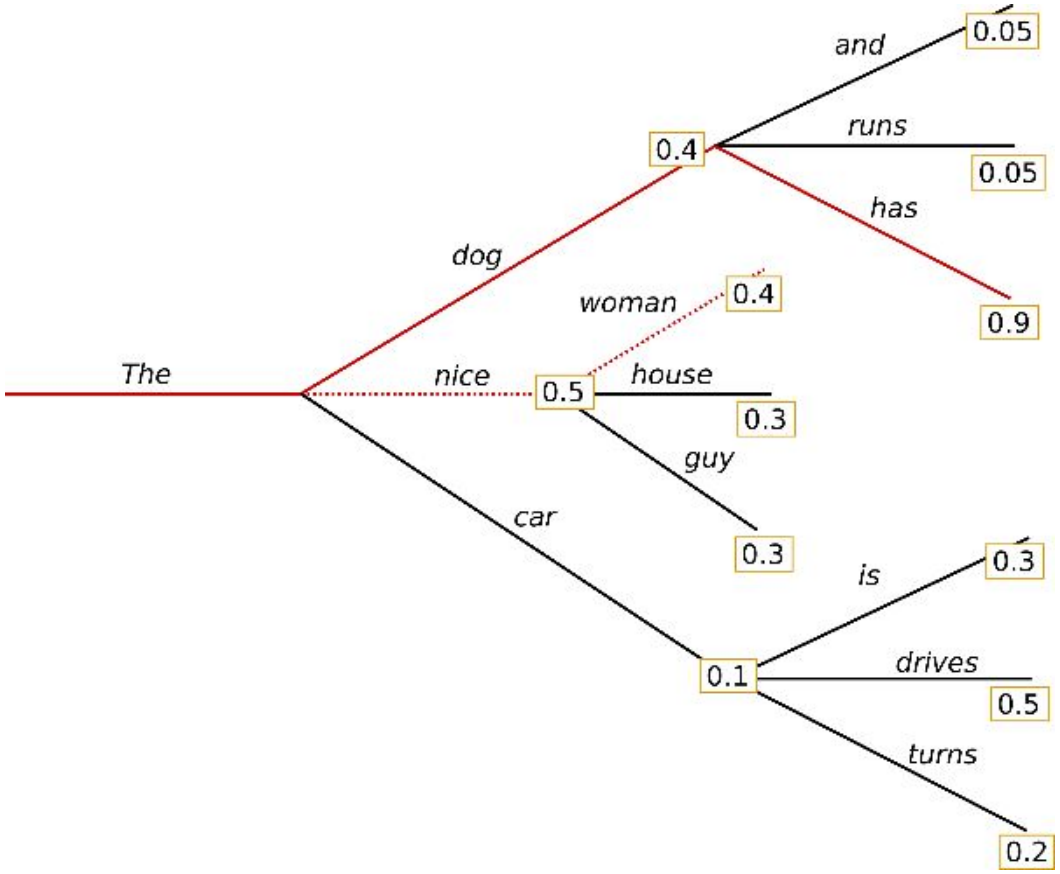
Conditional token probabilities

$$P(w_{1:T}|W_0) = \prod_{t=1}^{T} P(w_t|w_{1:t-1}, W_0) \text{ ,with } w_{1:0} = \emptyset,$$

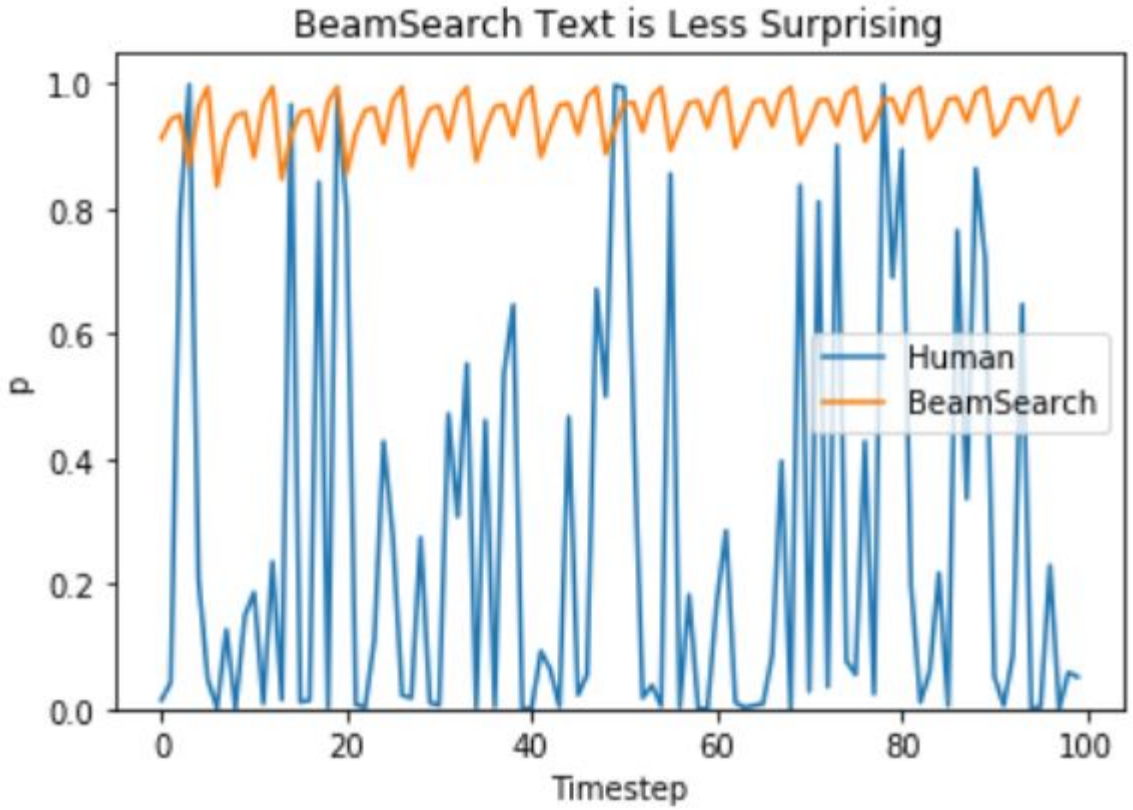
The easiest method - greedy decoding



More advanced deterministic approach - beam search

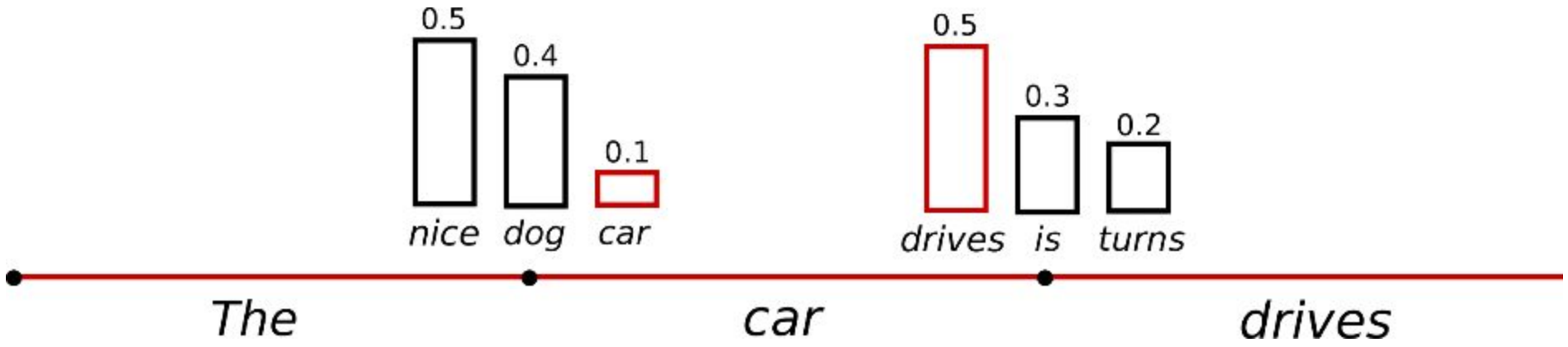


Is this good?



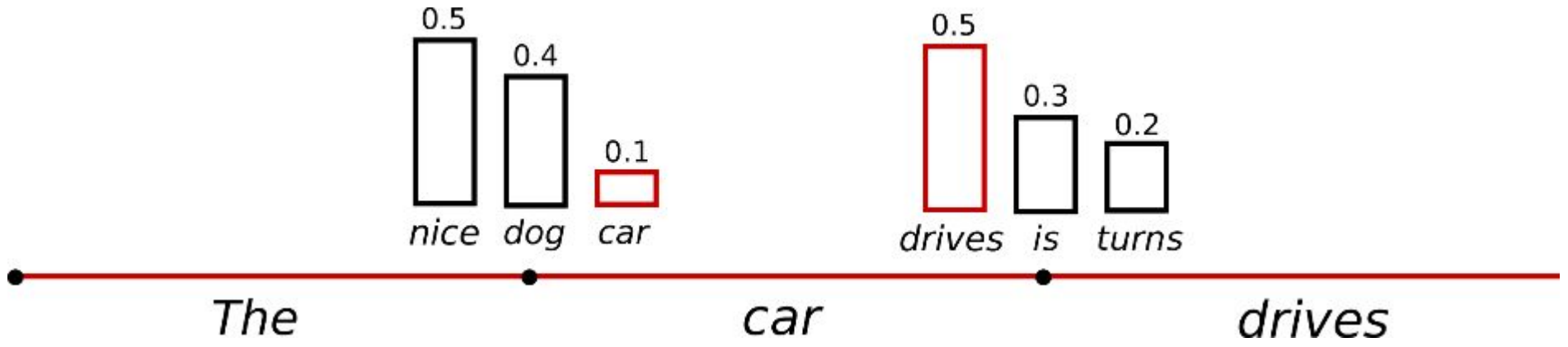
https://arxiv.org/abs/1904.09751

Sampling based approaches

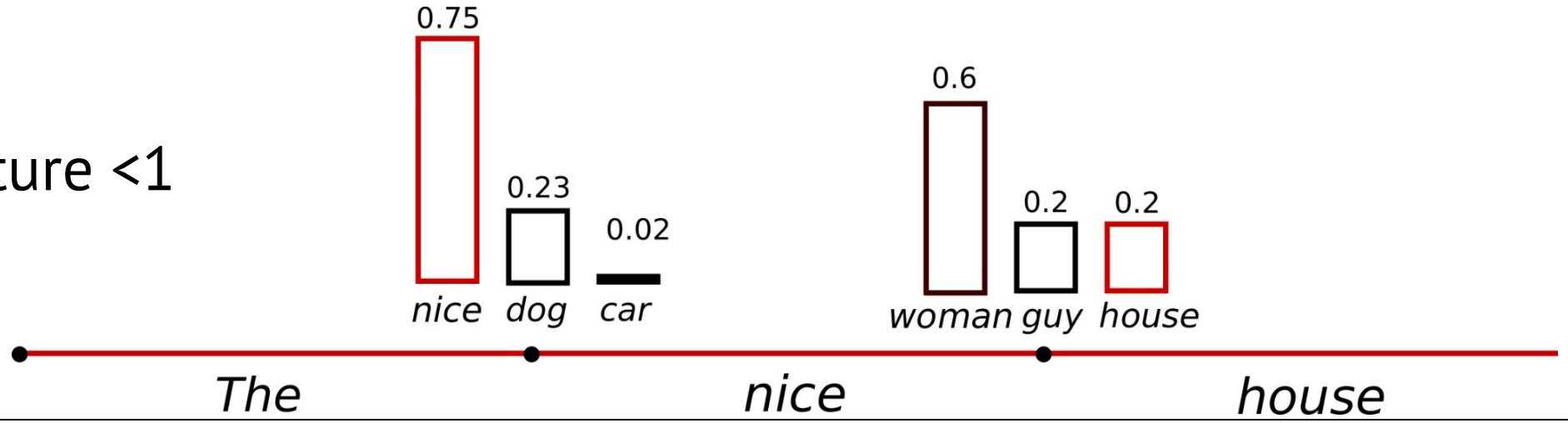


Too much randomness is bad

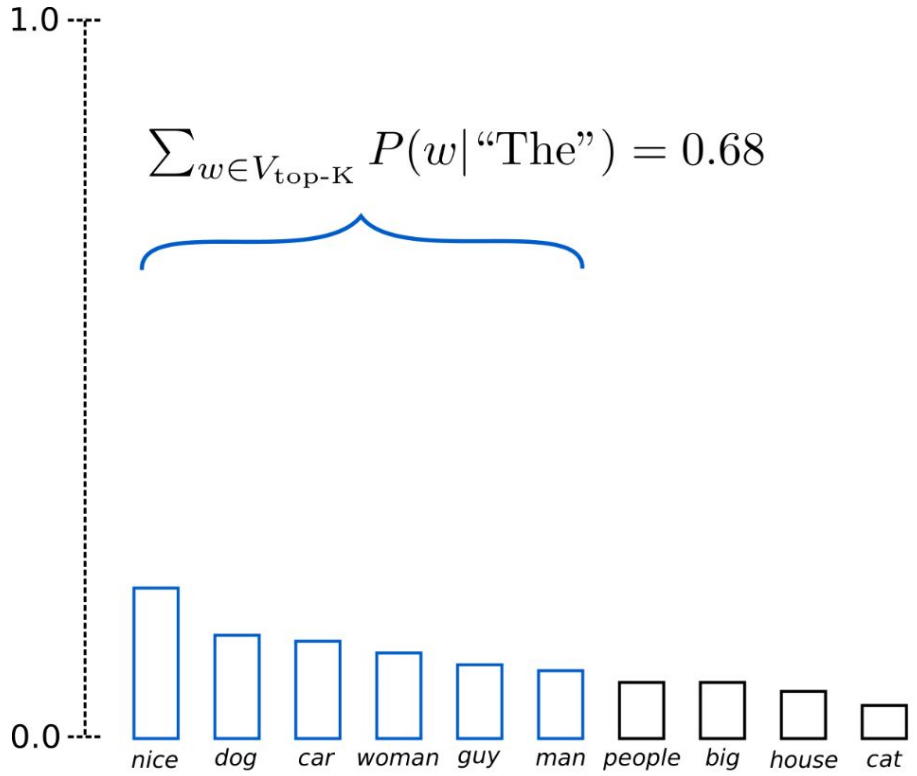
“Cooling down” distribution

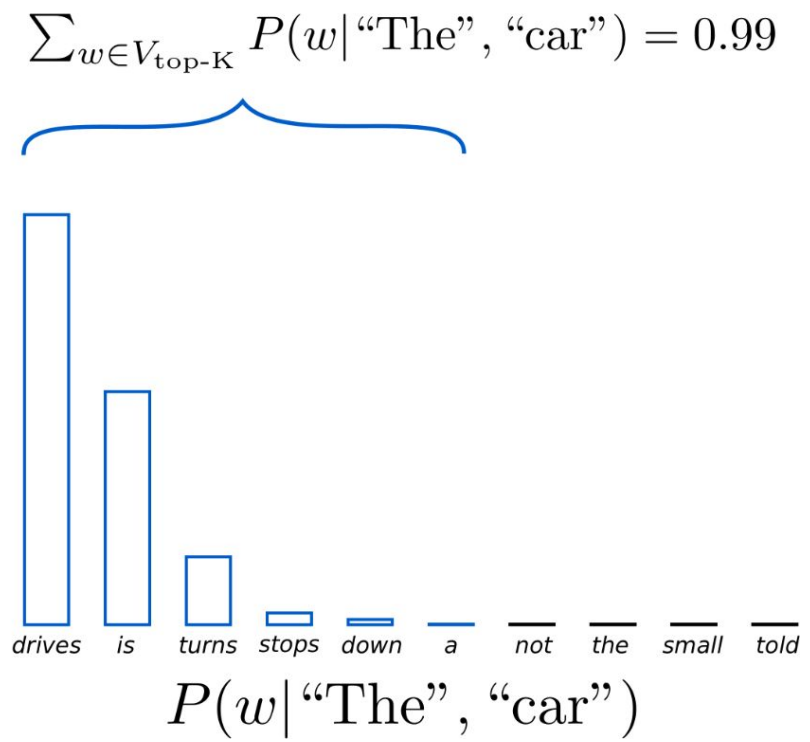


With temperature <1



Top-k sampling

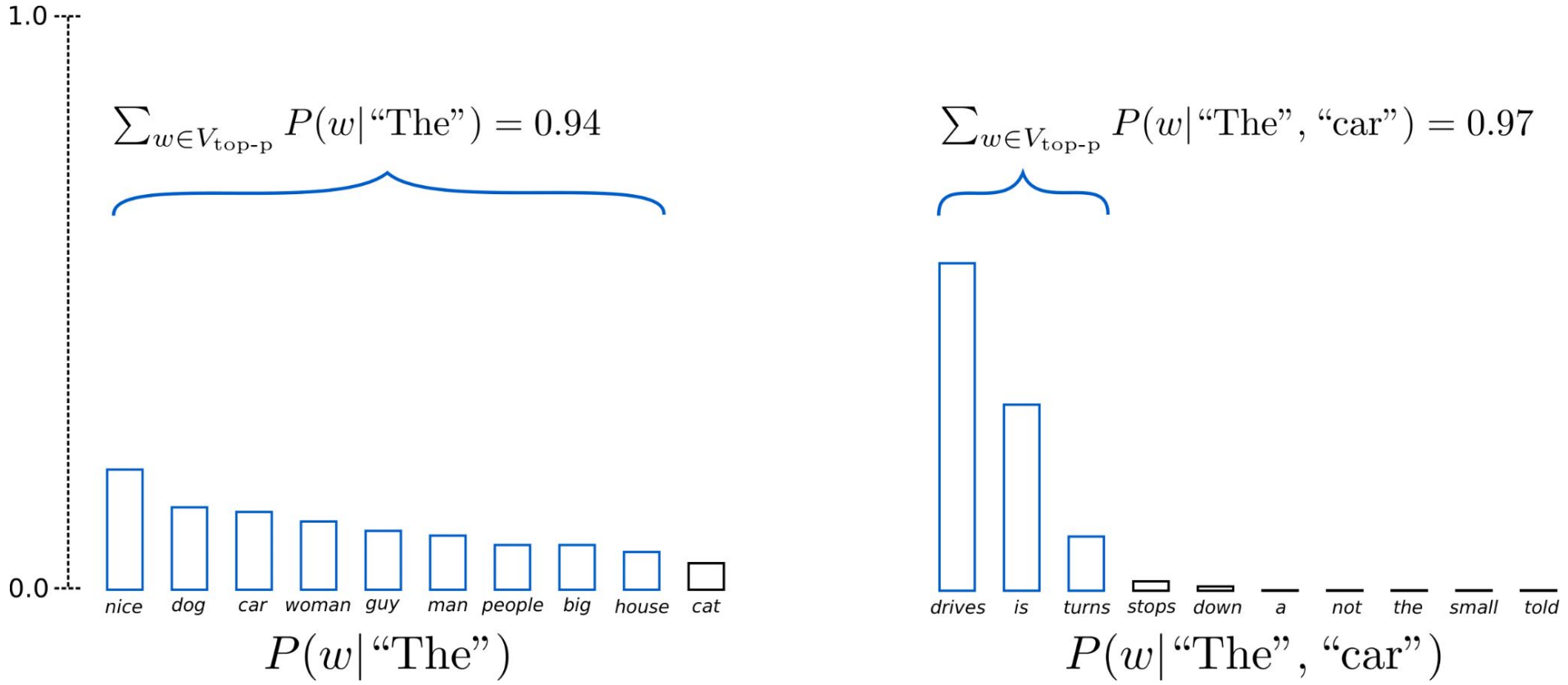




Top-k

- Better than temperature
- More human sounding than beam-search
- Doesn’t dynamically adapt to different token prediction distributions

Top-p (nucleus) sampling



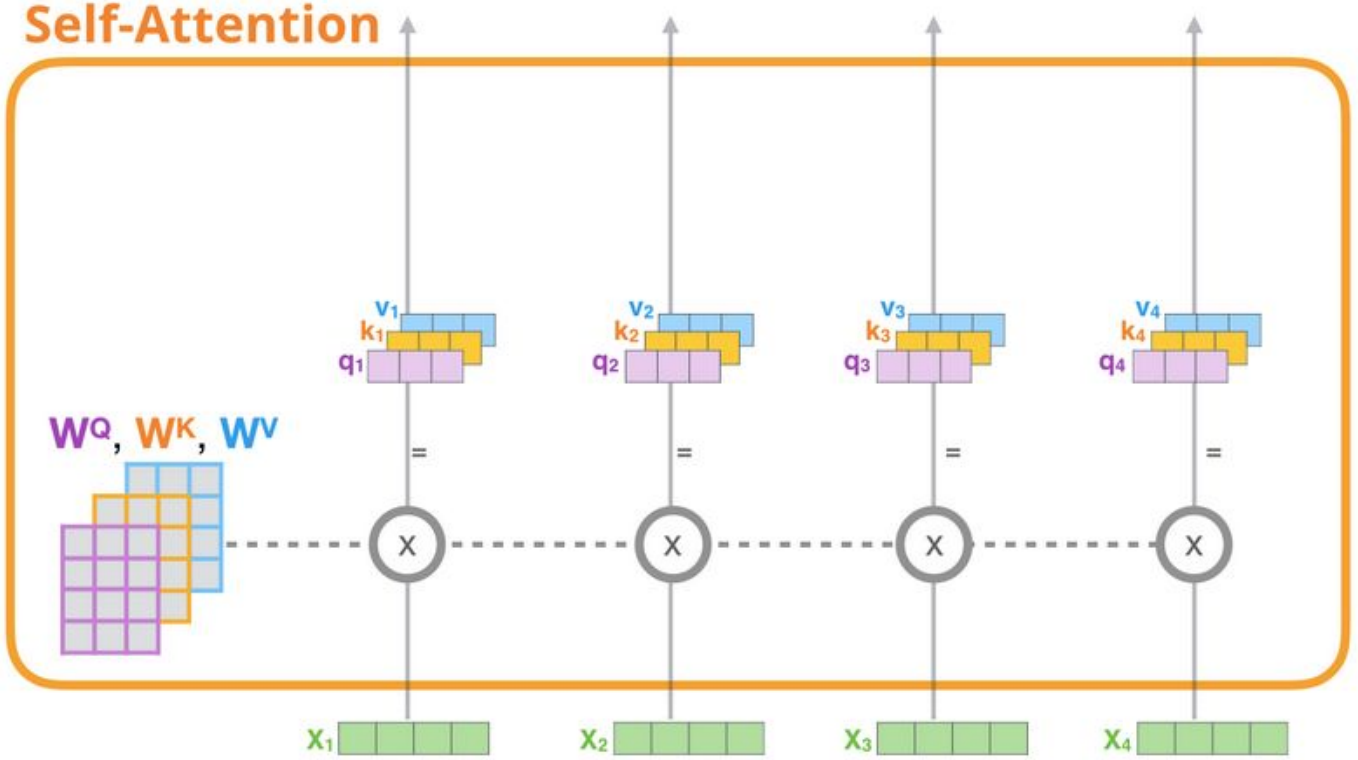
Top-p

- Generally works pretty well
- Can be used in conjunction with temperature and top-k

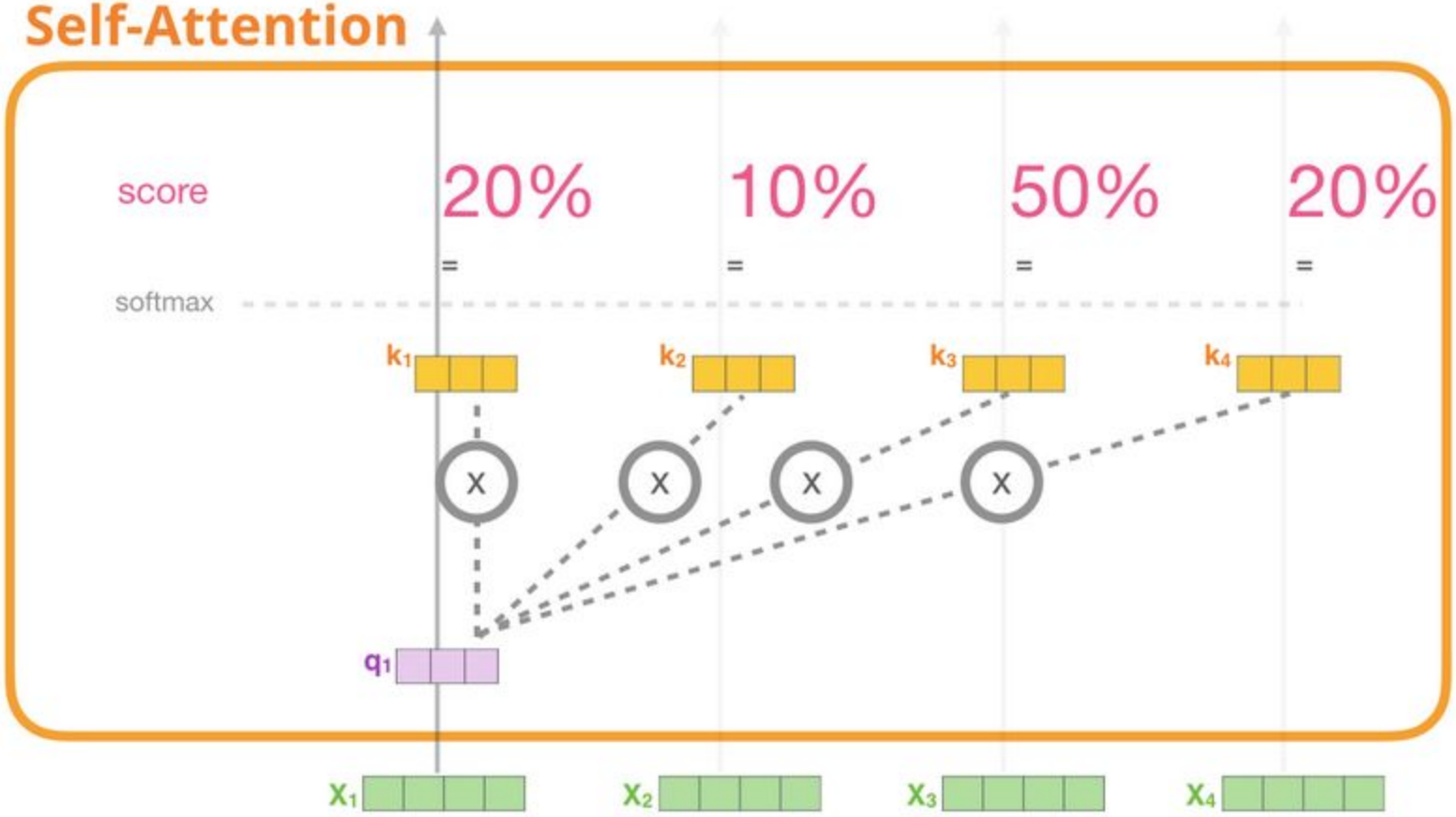
Back to self-attention

Self attention (without masking)

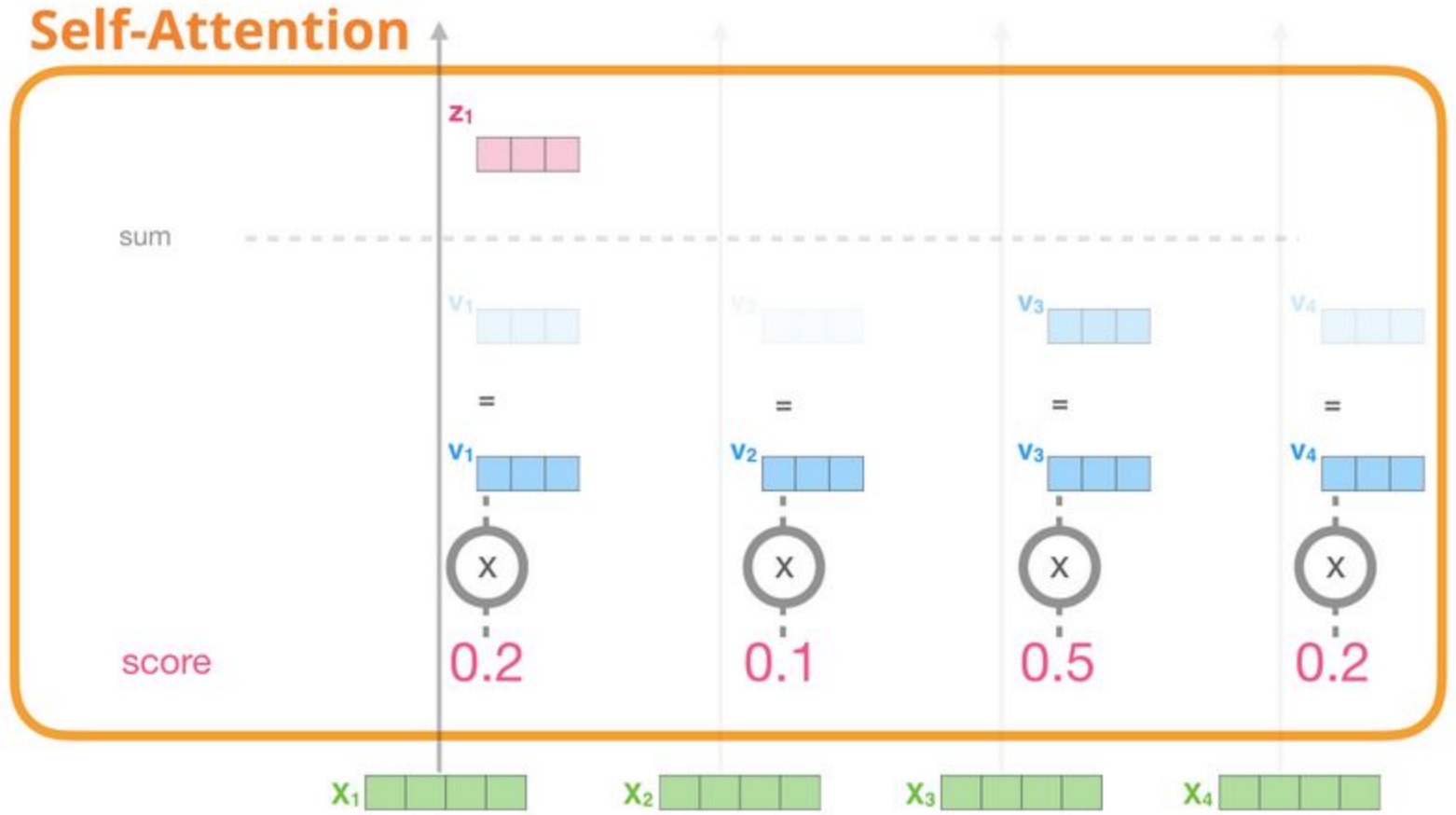
1) For each input token, create a query vector, a key vector, and a value vector by multiplying by weight Matrices W^Q, W^K, W^V



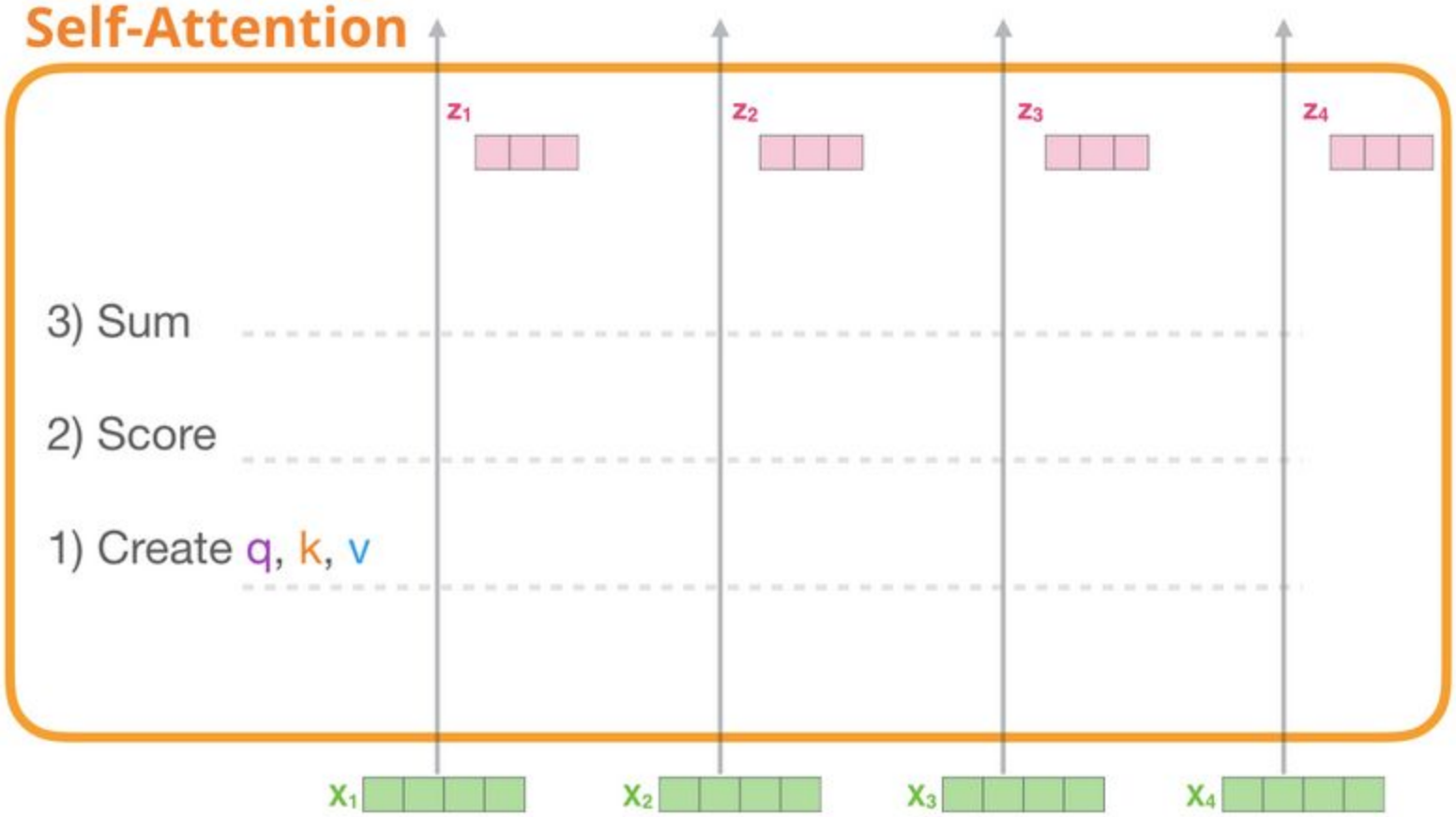
2) Multiply (dot product) the current query vector, by all the key vectors, to get a score of how well they match



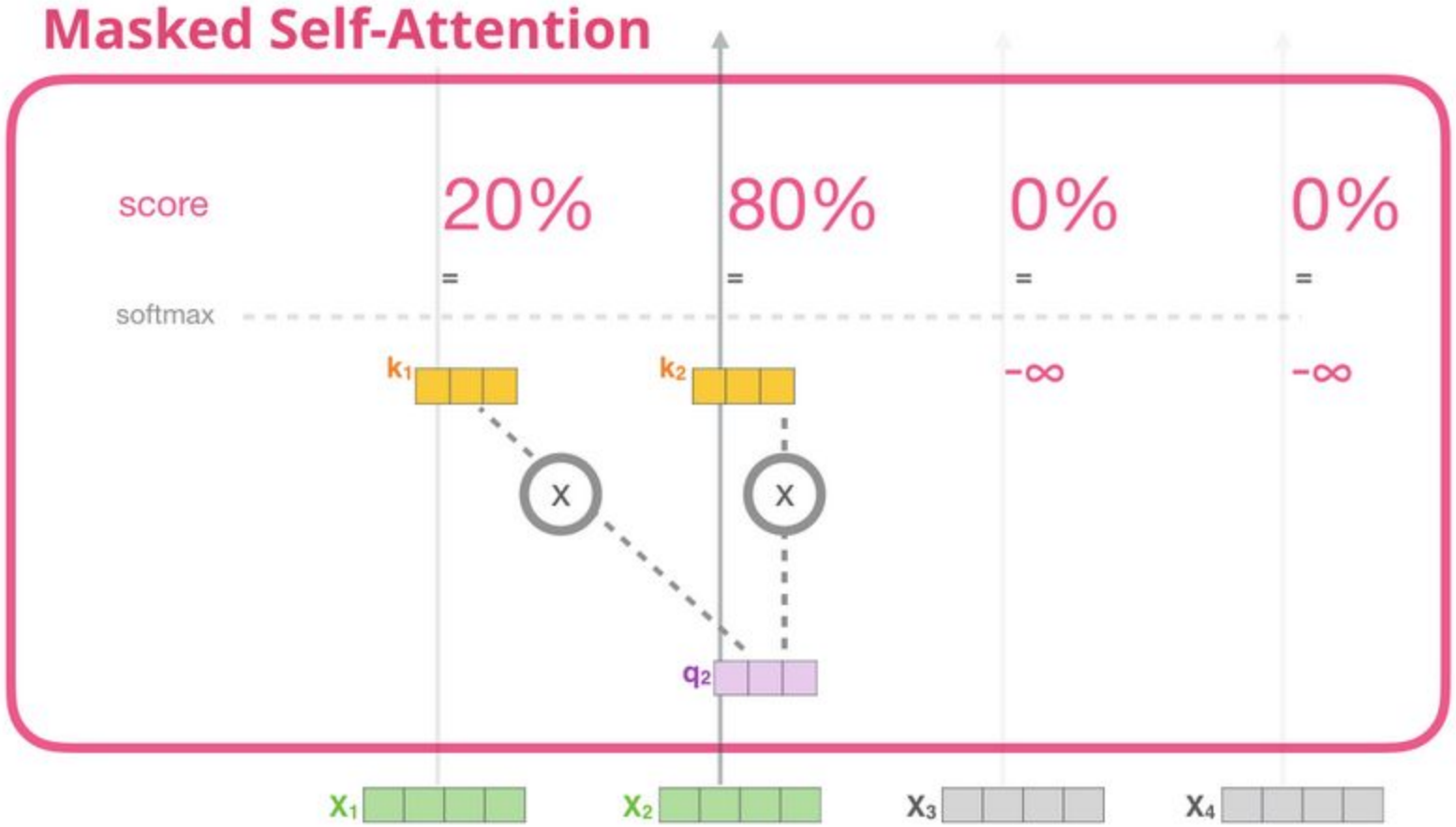
3) Multiply the value vectors by the scores, then sum up



The lower the score, the more transparent we're showing the value vector. That's to indicate how multiplying by a small number dilutes the values of the vector.

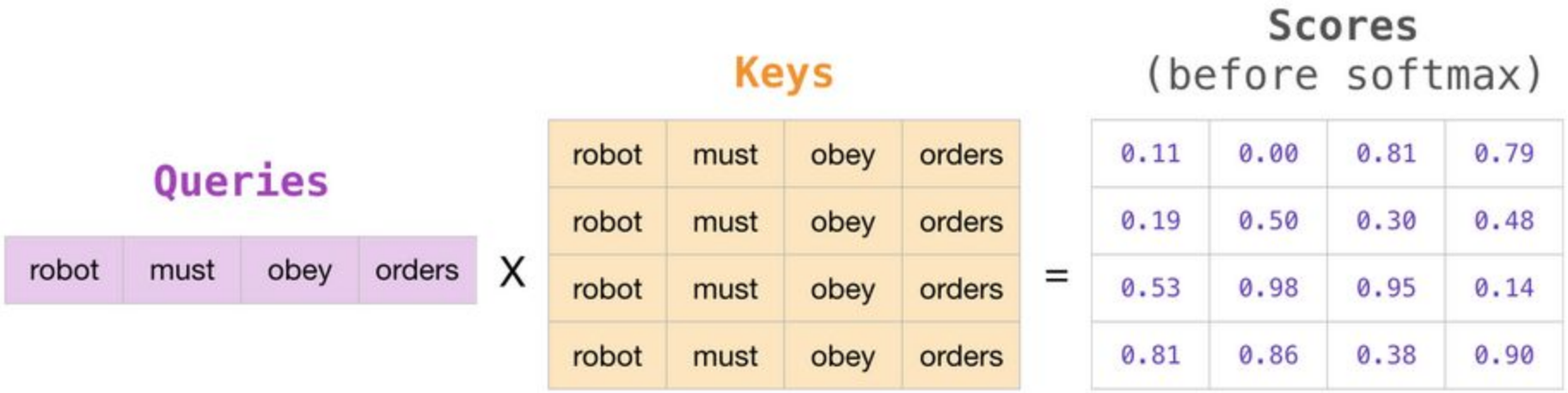

Self-Attention
z1
z2
z3
z4
3) Sum
2) Score
1) Create q, k, v
x1
x2
x3
x4

Masked self-attention



	Features				Labels
	position: 1	2	3	4	
Example: 1	robot	must	obey	orders	must
2	robot	must	obey	orders	obey
3	robot	must	obey	orders	orders
4	robot	must	obey	orders	<eos>

Linear algebra tricks



Scores
(before softmax)

0.11	0.00	0.81	0.79
0.19	0.50	0.30	0.48
0.53	0.98	0.95	0.14
0.81	0.86	0.38	0.90

Apply Attention Mask

Masked Scores
(before softmax)

0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Masked Scores
(before softmax)

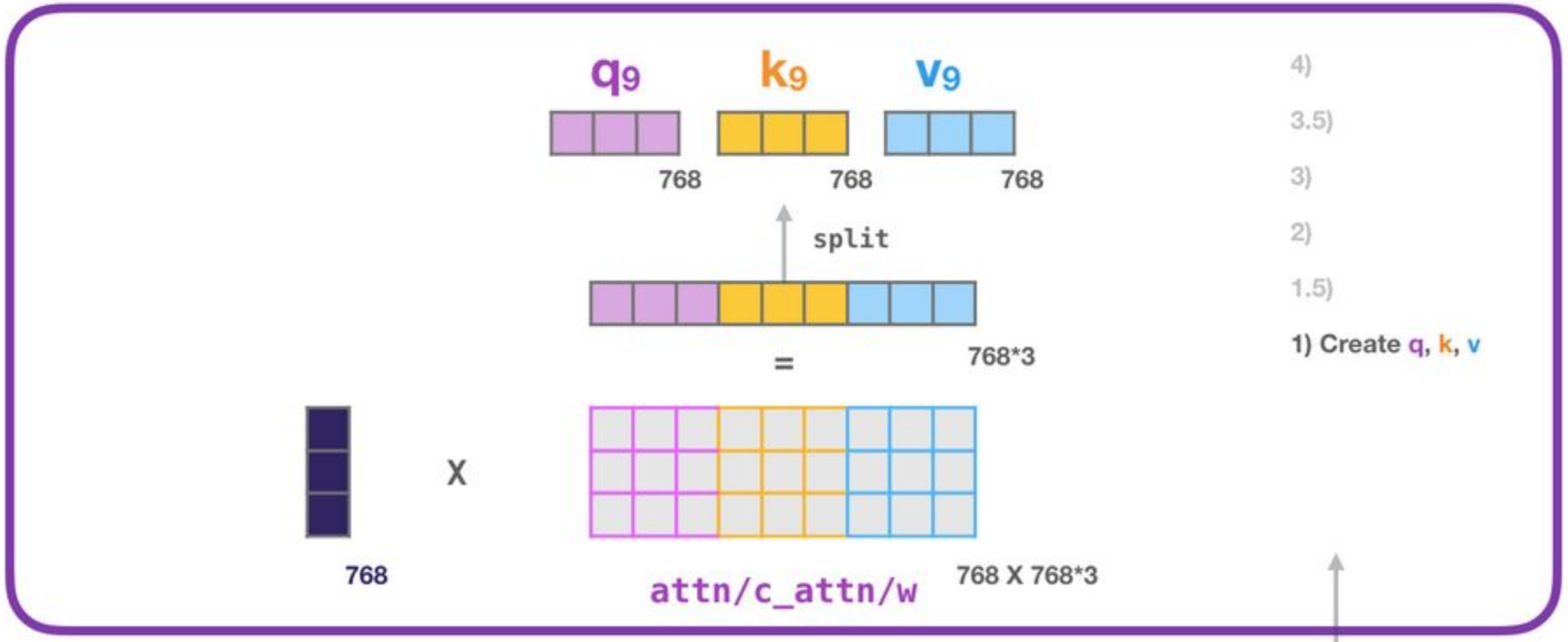
0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

Softmax
(along rows)

Scores

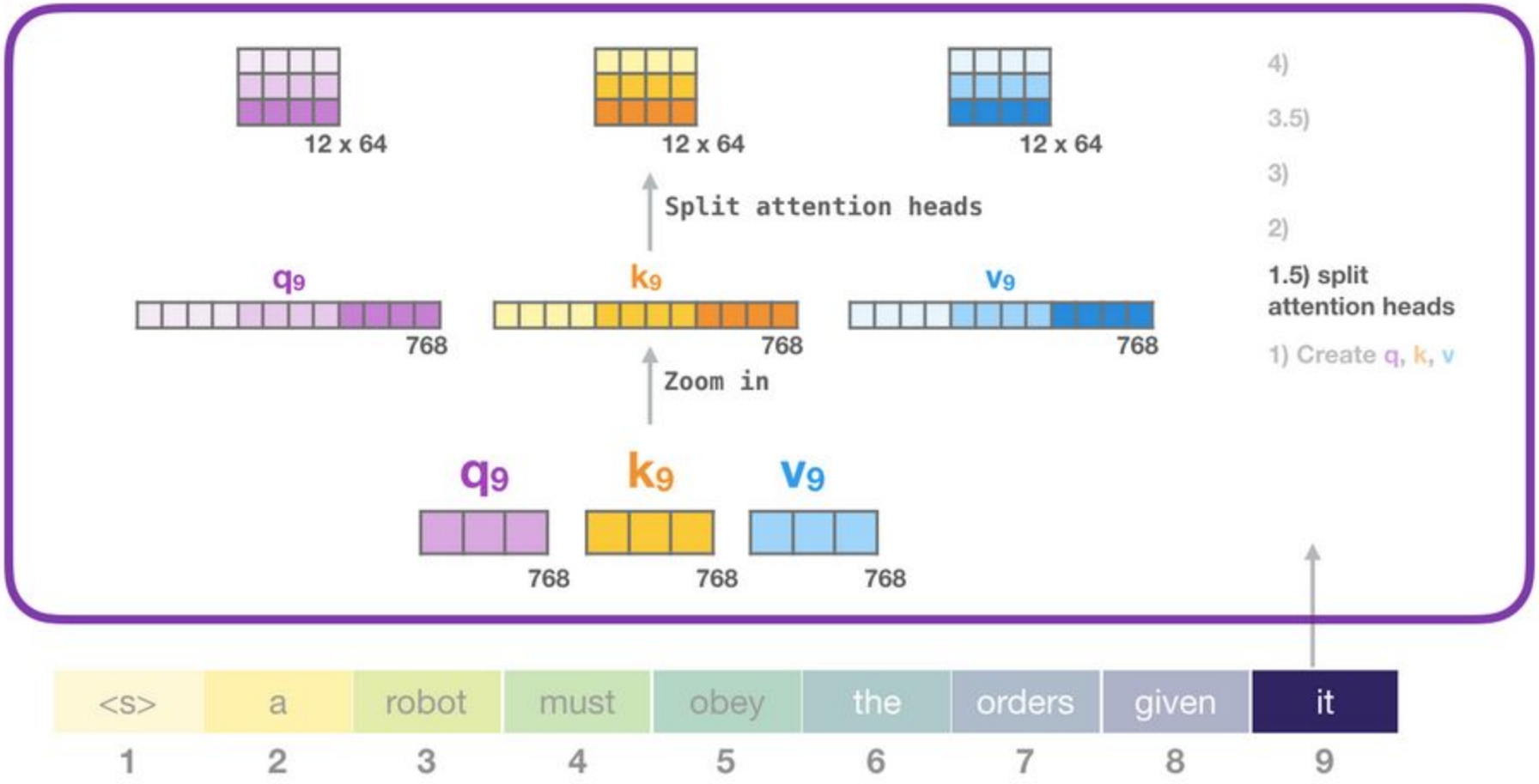
1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

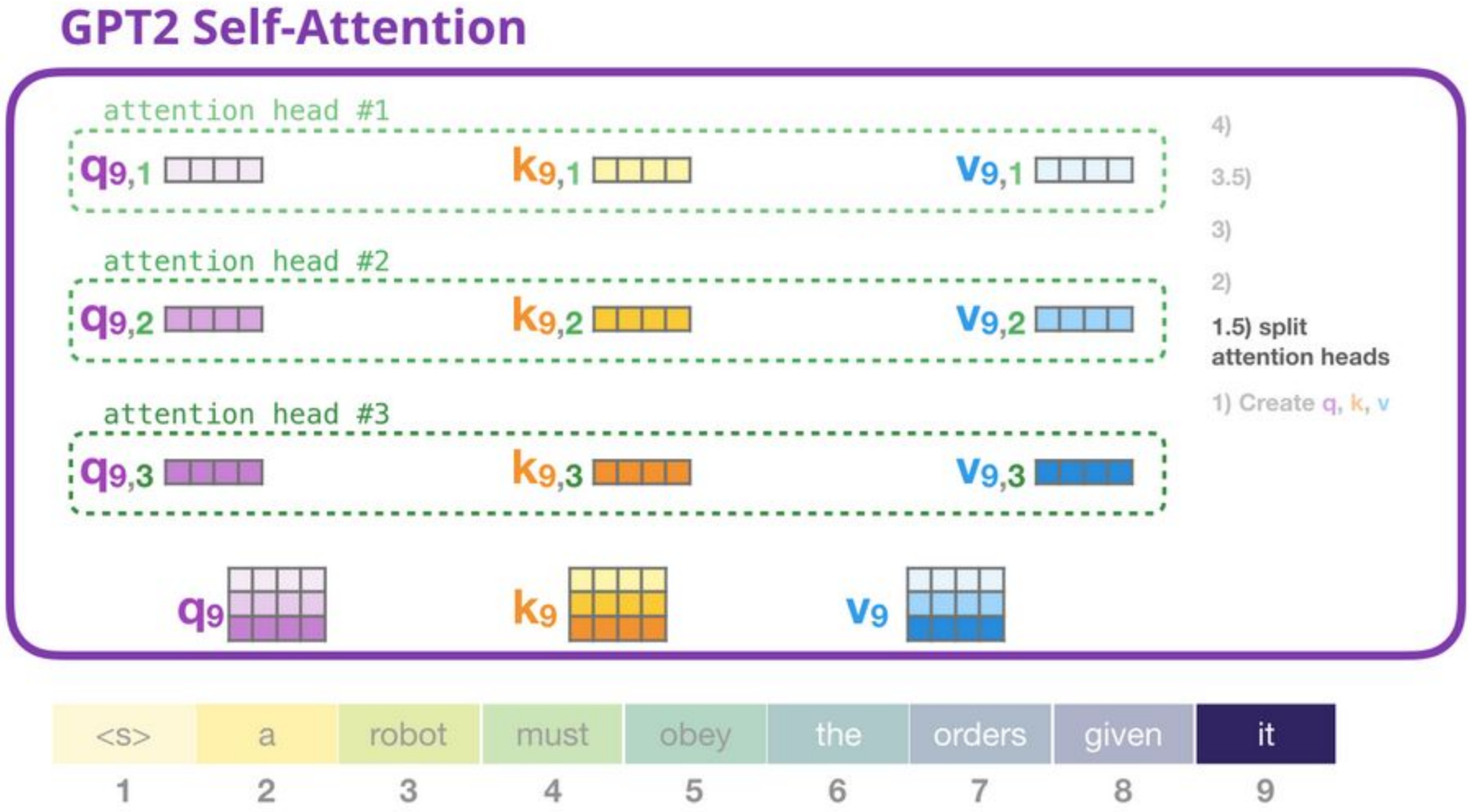
GPT2 Self-Attention



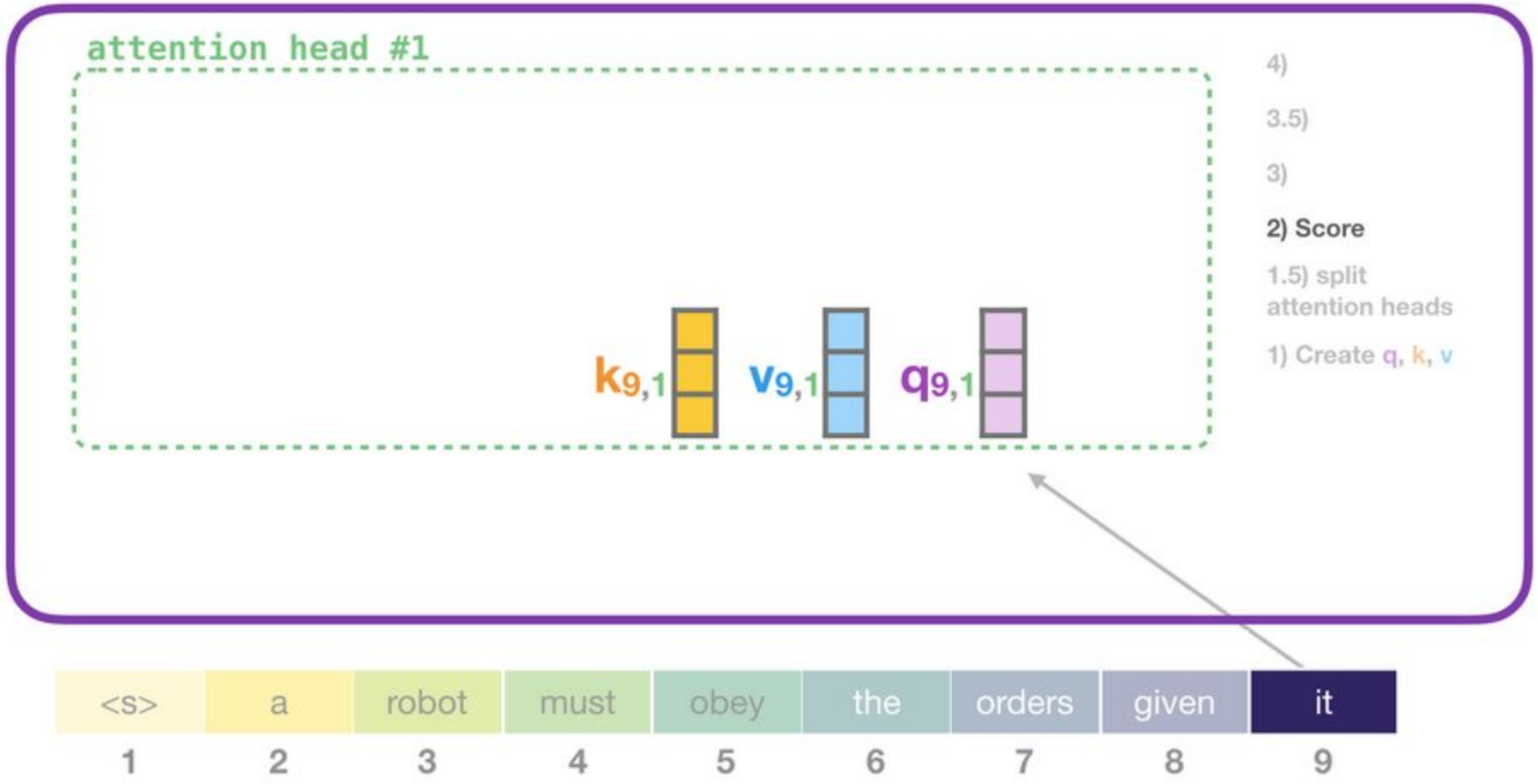
Multiplying the input vector by the attention weights vector (and adding a bias vector aftwards) results in the key, value, and query vectors for this token.

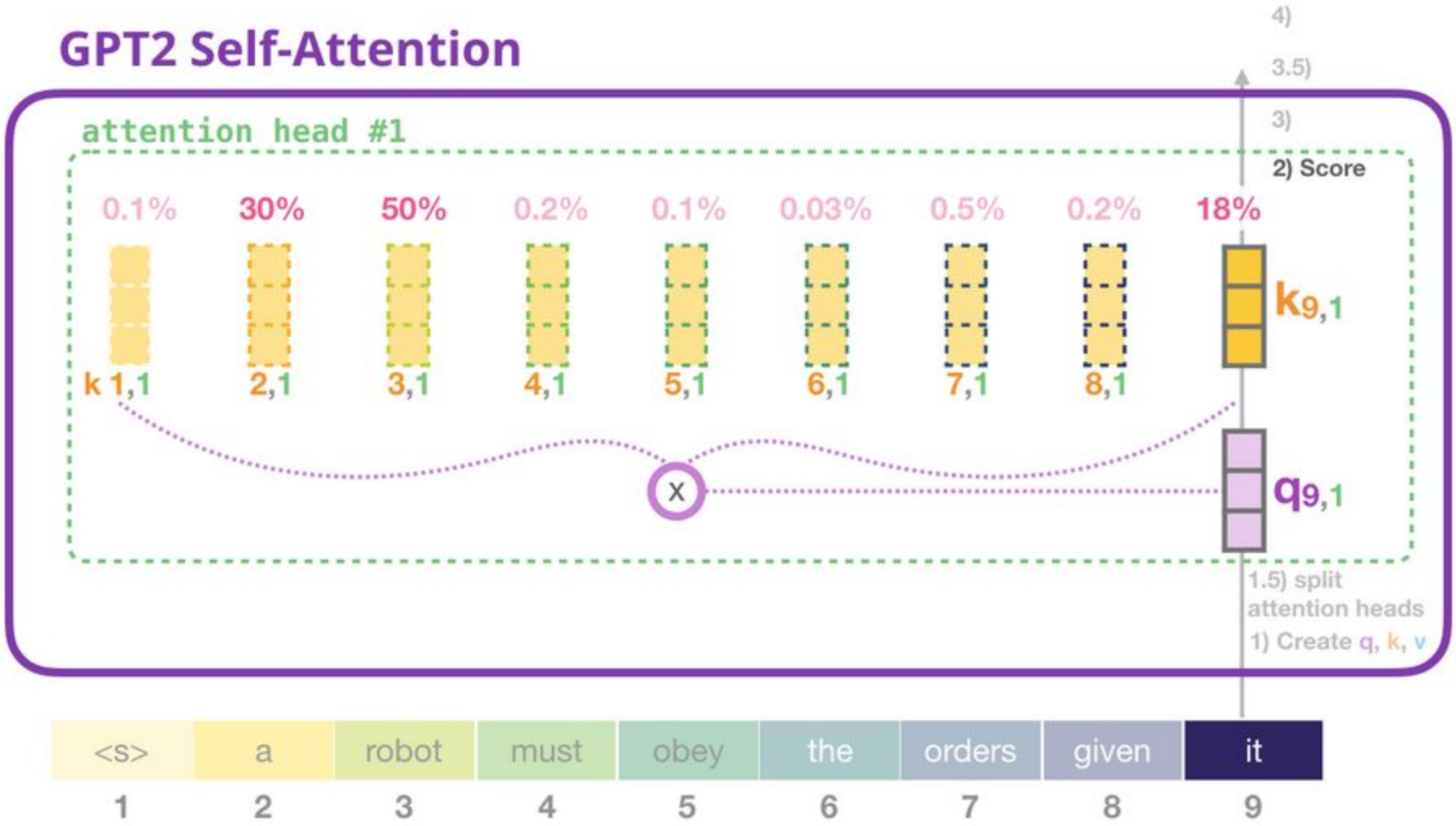
GPT2 Self-Attention



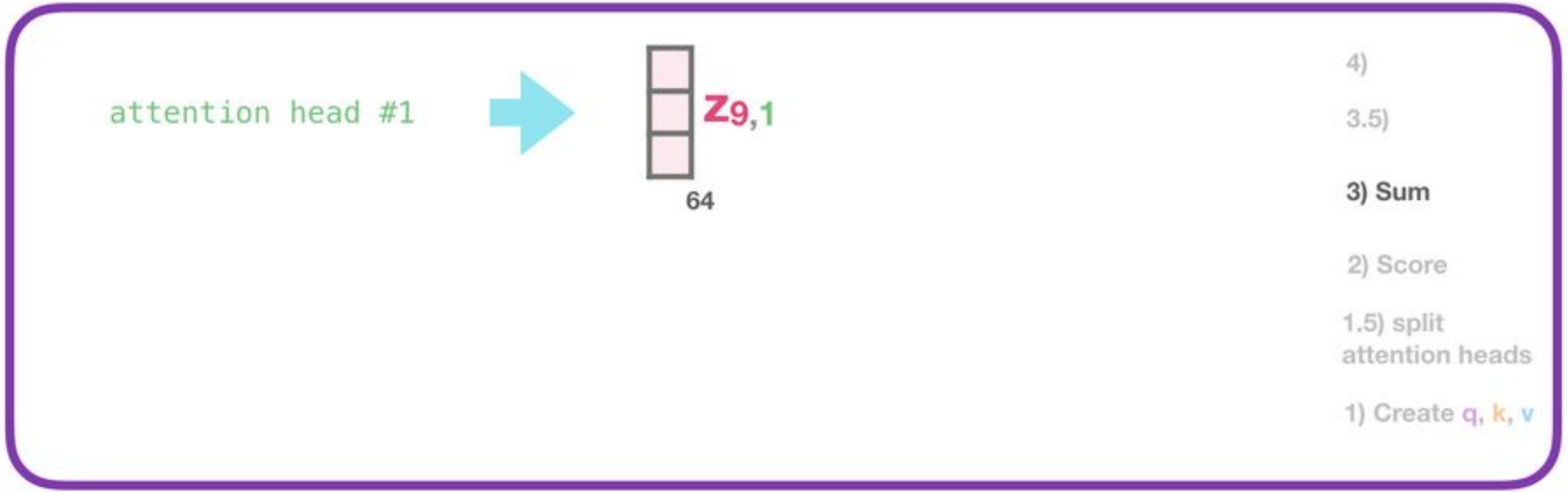

GPT2 Self-Attention
attention head #1
q9,1
k9,1
v9,1
attention head #2
q9,2
k9,2
v9,2
attention head #3
q9,3
k9,3
v9,3
q9
k9
v9
4)
3.5)
3)
2)
1.5) split attention heads
1) Create q, k, v
<s>
a
robot
must
obey
the
orders
given
it
1
2
3
4
5
6
7
8
9

GPT2 Self-Attention




GPT2 Self-Attention
attention head #1
0.1%
30%
50%
0.2%
0.1%
0.03%
0.5%
0.2%
18%
k 1,1
2,1
3,1
4,1
5,1
6,1
7,1
8,1
k9,1
q9,1
x
4)
3.5)
3)
2) Score
1.5) split attention heads
1) Create q, k, v
<s>
a
robot
must
obey
the
orders
given
it
1
2
3
4
5
6
7
8
9

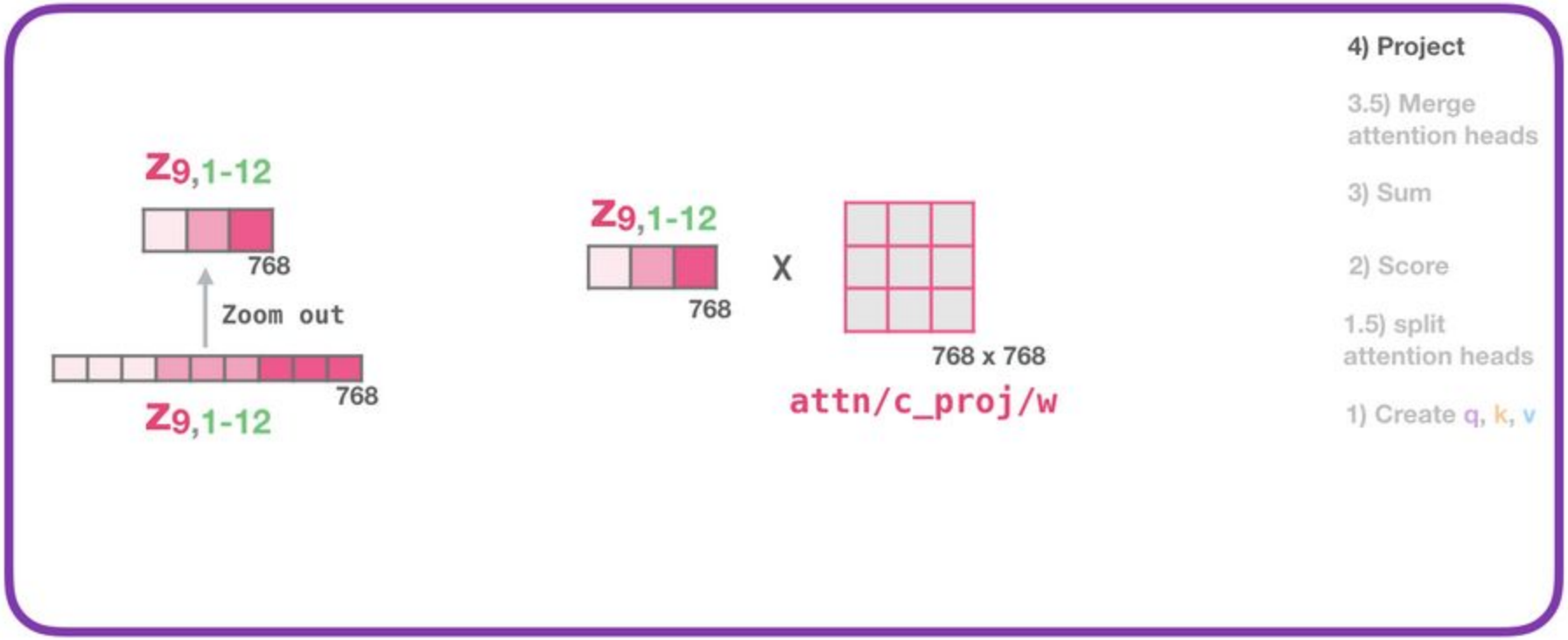
GPT2 Self-Attention

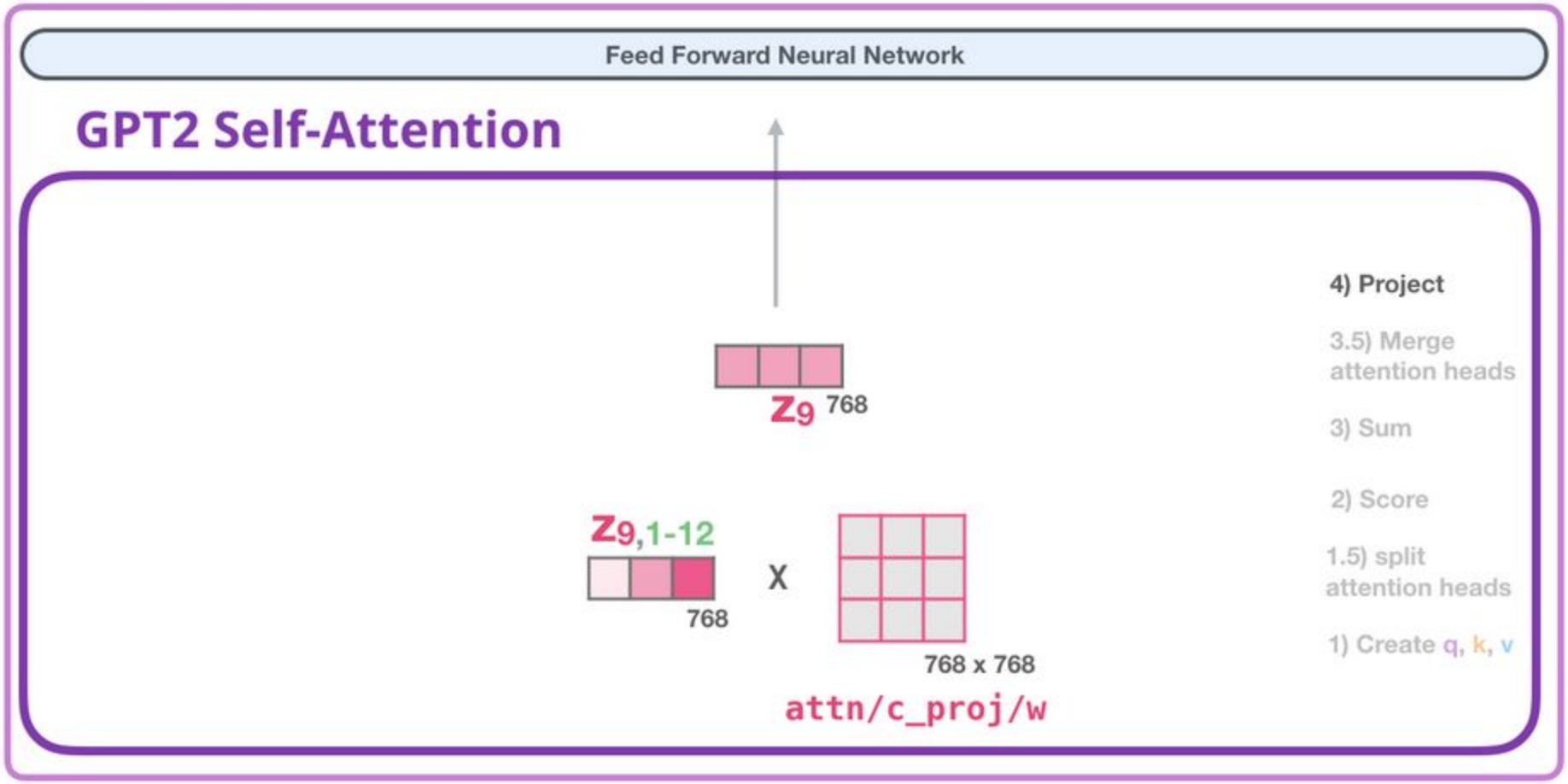


<s>	a	robot	must	obey	the	orders	given	it
1	2	3	4	5	6	7	8	9

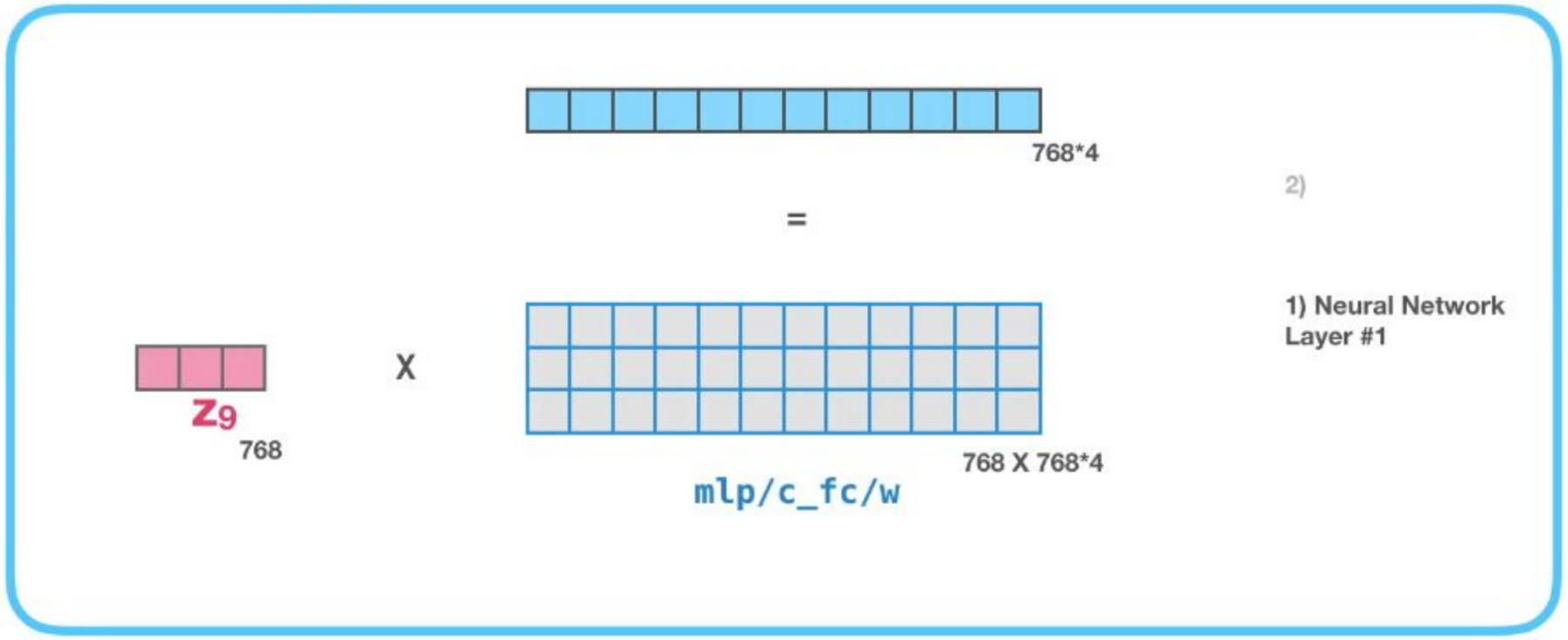
<s>	a	robot	must	obey	the	orders	given	it
1	2	3	4	5	6	7	8	9

GPT2 Self-Attention



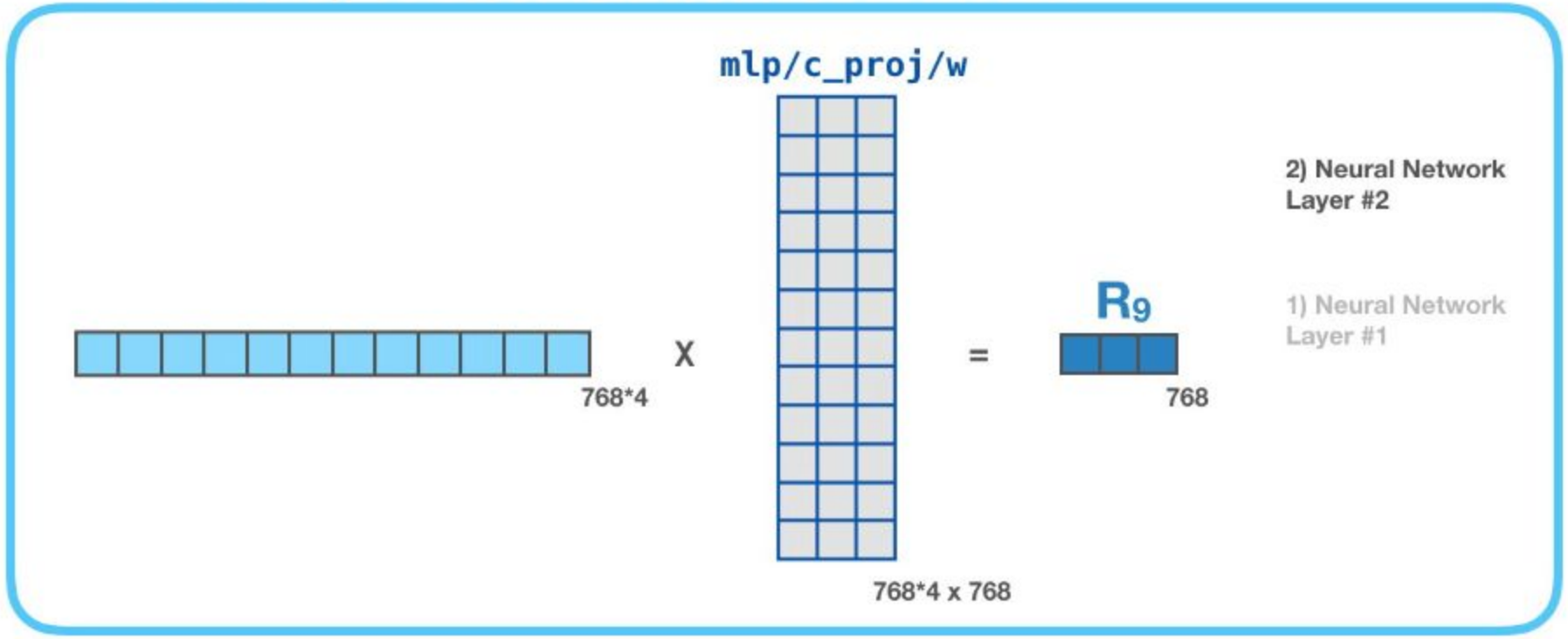

Feed Forward Neural Network
GPT2 Self-Attention
4) Project
3.5) Merge attention heads
3) Sum
2) Score
1.5) split attention heads
1) Create q, k, v
Z9 768
Z9,1-12
768
X
768 x 768
attn/c_proj/w

GPT2 Fully-Connected Neural Network

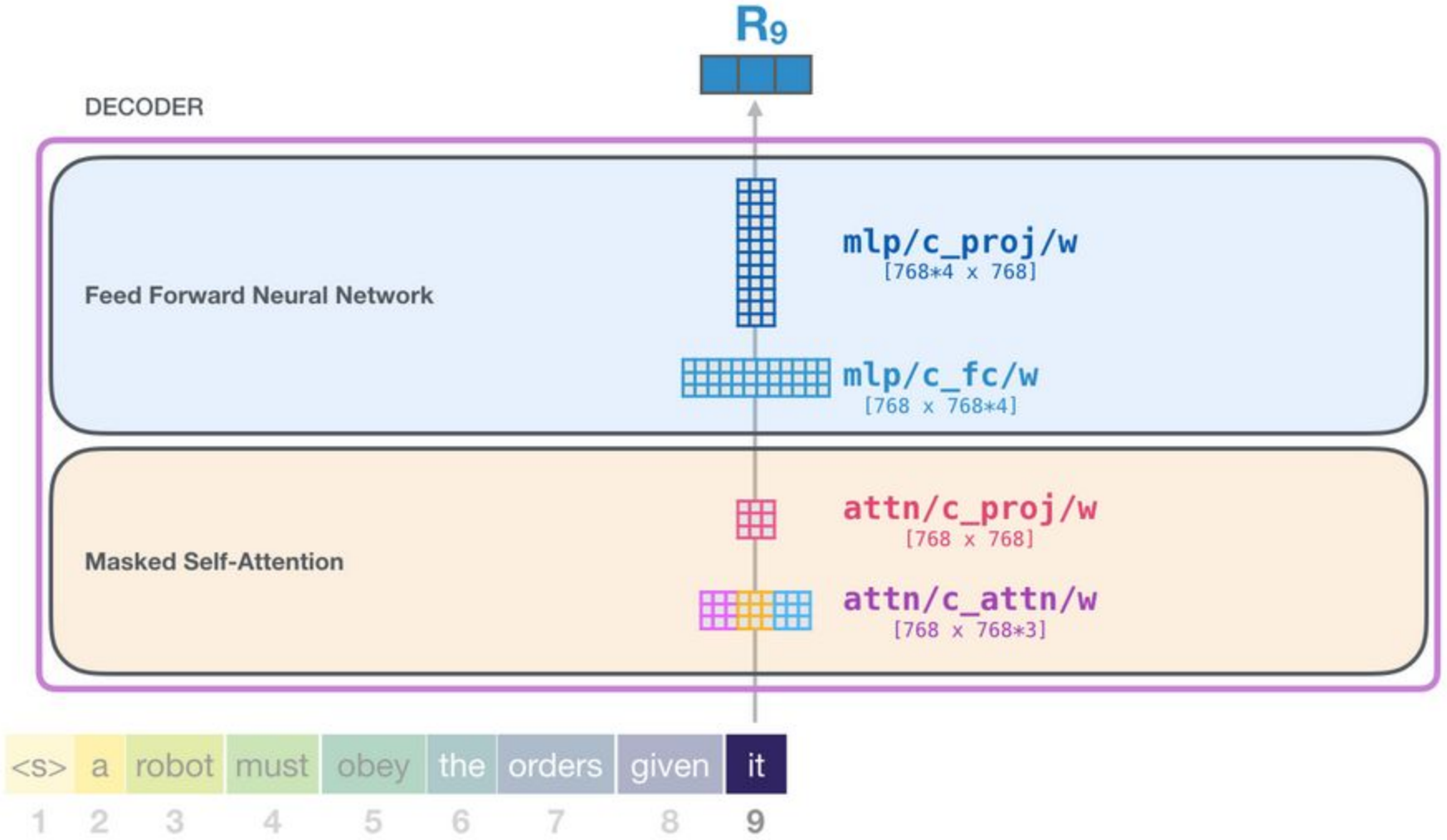


(Not shown: A bias vector)

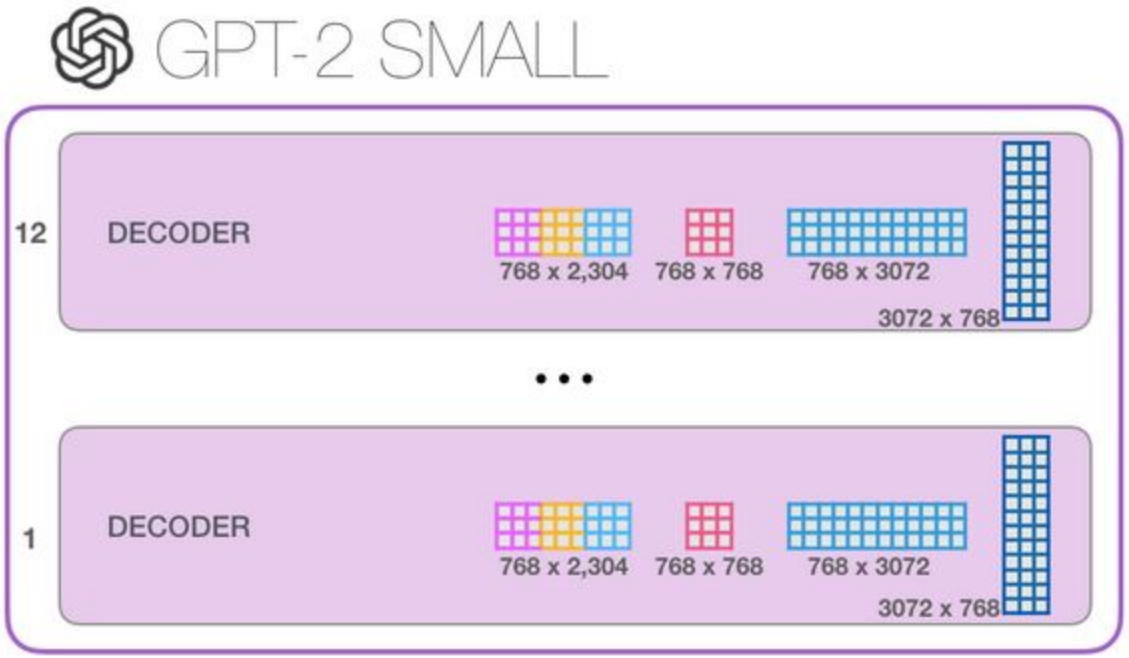
GPT2 Fully-Connected Neural Network

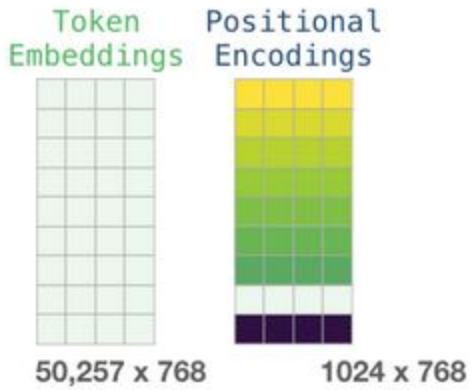


(Not shown: A bias vector)



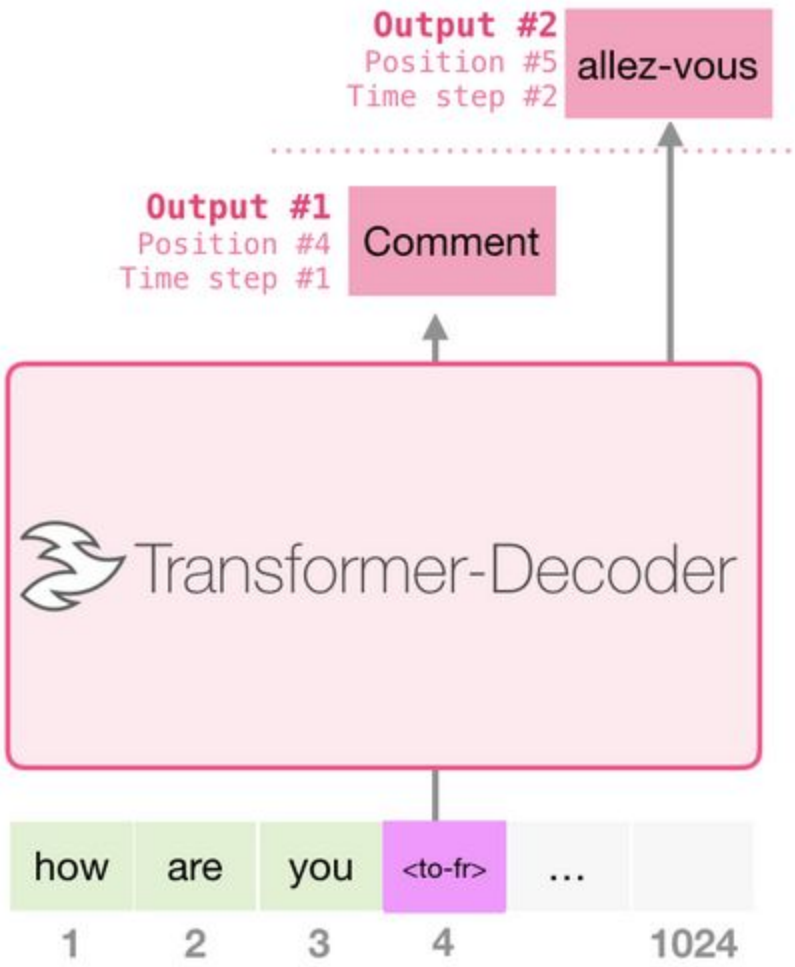
And each block has its own set of these weights. On the other hand, the model has only one token embedding matrix


GPT-2 SMALL
12
DECODER
768 x 2,304
768 x 768
768 x 3072
3072 x 768
1
DECODER
768 x 2,304
768 x 768
768 x 3072
3072 x 768


Token Embeddings
Positional Encodings
50,257 x 768
1024 x 768

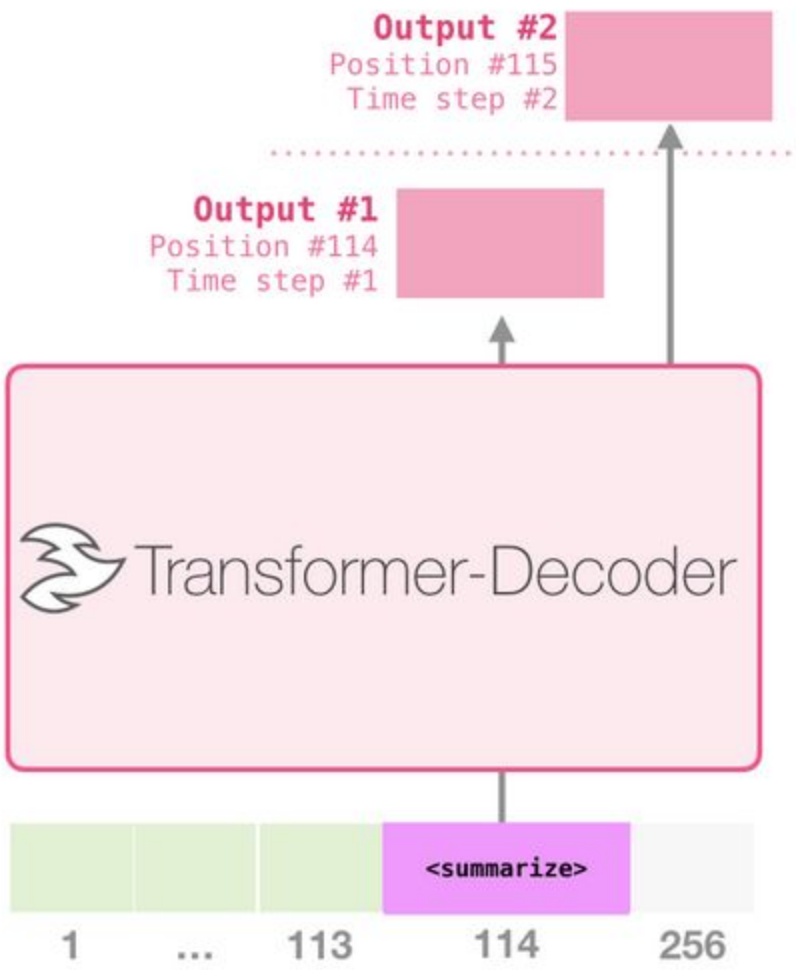
Training Dataset

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				



Training Dataset

Article #1 tokens		<summarize>	Article #1 Summary
Article #2 tokens	<summarize>	Article #2 Summary	padding
Article #3 tokens		<summarize>	Article #3 Summary



Critique / Limitations / Open Issues

- Baseline transformers use O(n^2) in memory/computation.
  - increasing size of length increases computation quadratically

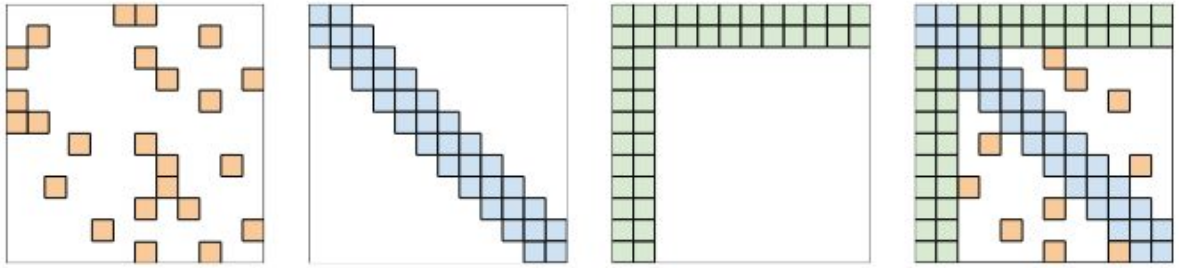
- Fixed length unlike RNN of arbitrary length
- Inability to process input sequentially (not like how the human brain works)
- Paper addressing limitation of transformers: https://arxiv.org/pdf/2002.09402.pdf
  - Limited access to long memory
  - Limited ability to update state.

The challenge of long document summarization

- Most transformer models use self-attention, which requires $O(n^2)$ time and memory complexity for n-length input
- This limits input size, historically as small as 512 tokens (2022)

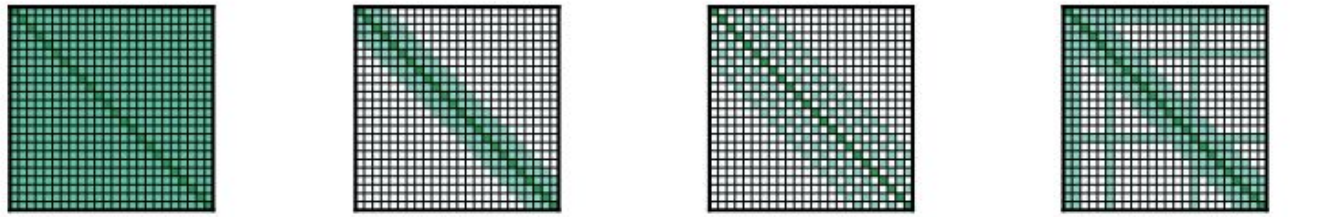
The challenge of long document summarization, modifying attention

1. Sparse Attention - Used by BigBird[1] and BigBirdPegasus



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

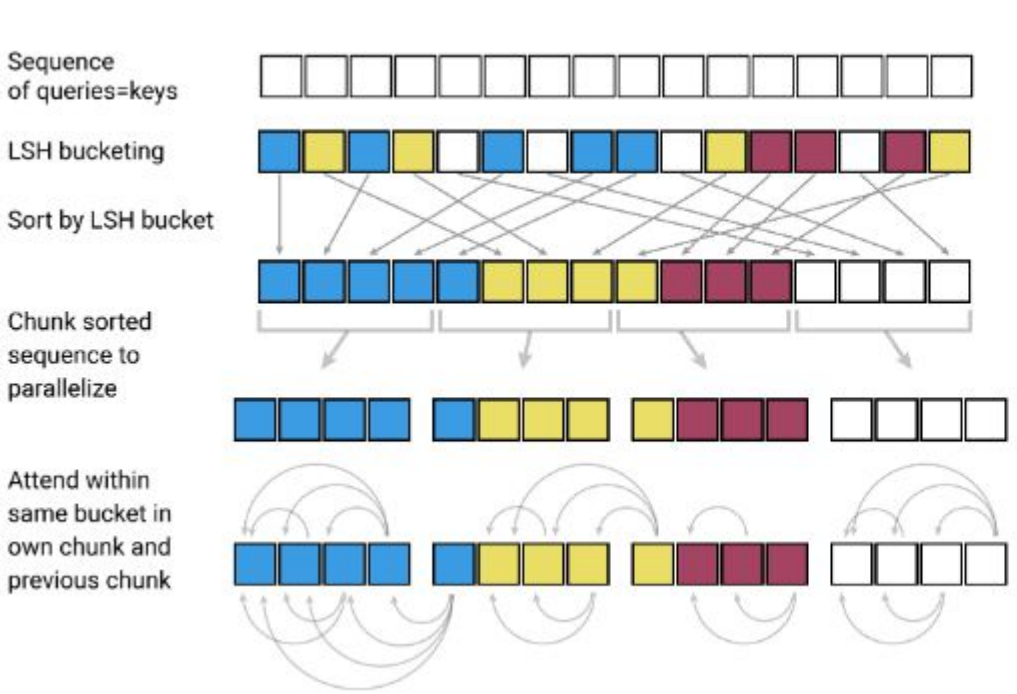
2. Sliding Dilated Window + Global Attention used by Longformer [2]

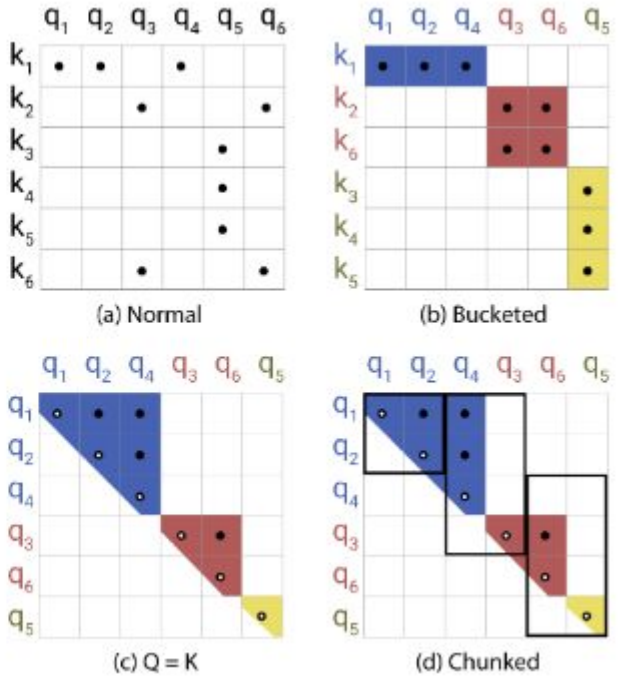


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

1. Zaheer, Manzil, et al. "Big Bird: Transformers for Longer Sequences." *NeurIPS*. 2020.
2. Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).
3. Kitaev, Nikita, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The efficient transformer." *arXiv preprint arXiv:2001.04451* (2020).

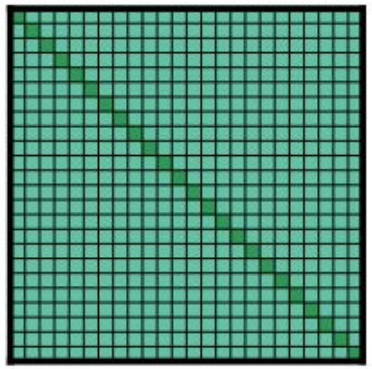
Reformer - Locality Sensitive Hashing Attention



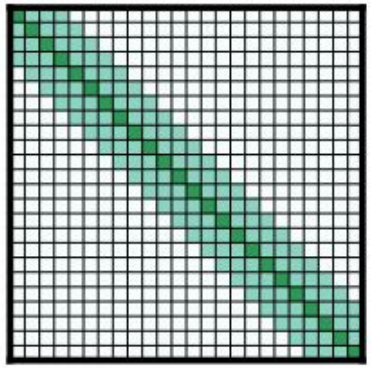


1. Zaheer, Manzil, et al. "Big Bird: Transformers for Longer Sequences." *NeurIPS*. 2020.
2. Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).
3. Kitaev, Nikita, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The efficient transformer." *arXiv preprint arXiv:2001.04451* (2020).

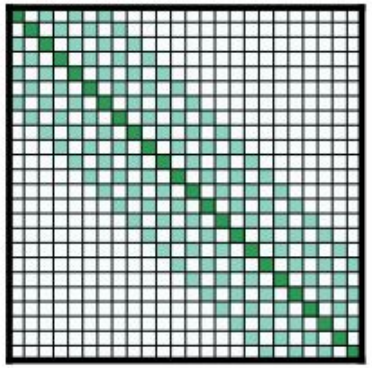
Longformer



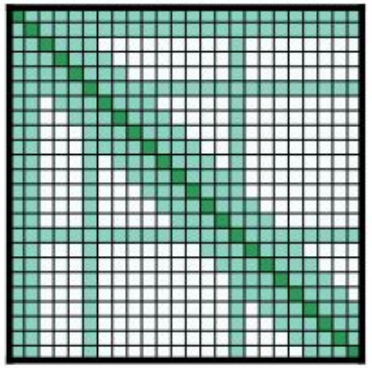
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window



(d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Longformer

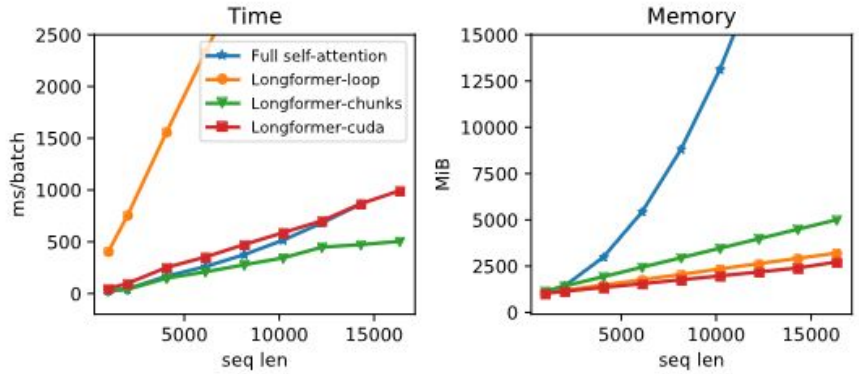
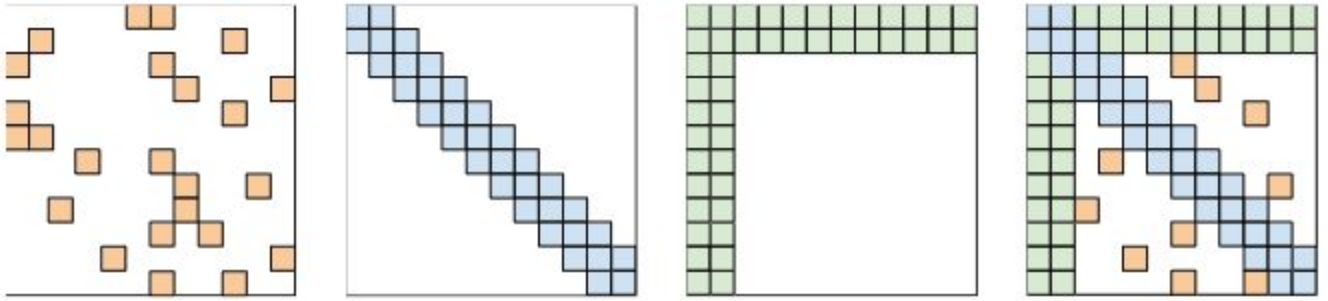


Figure 1: Runtime and memory of full self-attention and different implementations of Longformer's self-attention; `Longformer-loop` is non-vectorized, `Longformer-chunk` is vectorized, and `Longformer-cuda` is a custom cuda kernel implementations. Longformer's memory usage scales linearly with the sequence length, unlike the full self-attention mechanism that runs out of memory for long sequences on current GPUs. Different implementations vary in speed, with the vectorized `Longformer-chunk` being the fastest. More details are in section 3.2.

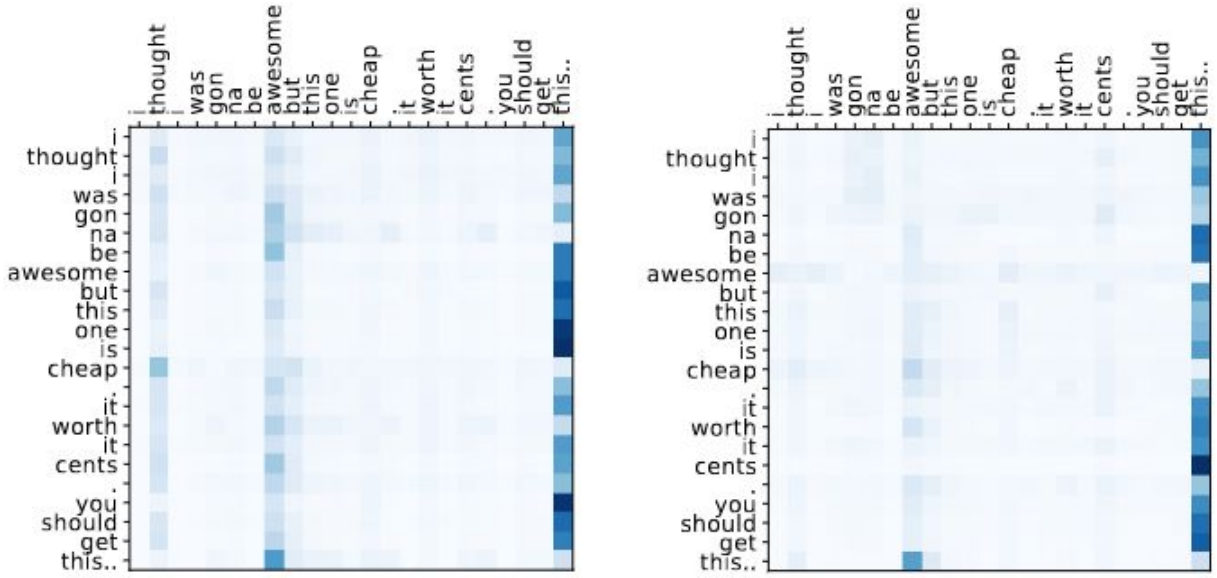
Big Bird



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

Smart Bird



(a) Attention heatmap of a 4-dim Transformer.

(b) Attention heatmap of a 256-dim Transformer.

Figure 1: The attention heatmaps learned by Transformers with 4 or 256 hidden dimensions.

Method	CNN/DM			PubMed		
	R-1	R-2	R-L	R-1	R-2	R-L
Transformer	38.51	16.08	35.90	34.43	11.84	31.76
Sparse Transformer	38.02	15.50	35.10	37.19	14.63	33.85
Longformer	37.99	15.34	35.28	37.04	14.41	33.82
BigBird	38.57	15.80	35.86	37.77	15.16	34.53
Smart Bird	**39.22**	**16.96**	**37.04**	**38.83**	**16.01**	**35.49**

Table 4: Results of text summarization.

https://arxiv.org/abs/2108.09193

Smart Bird

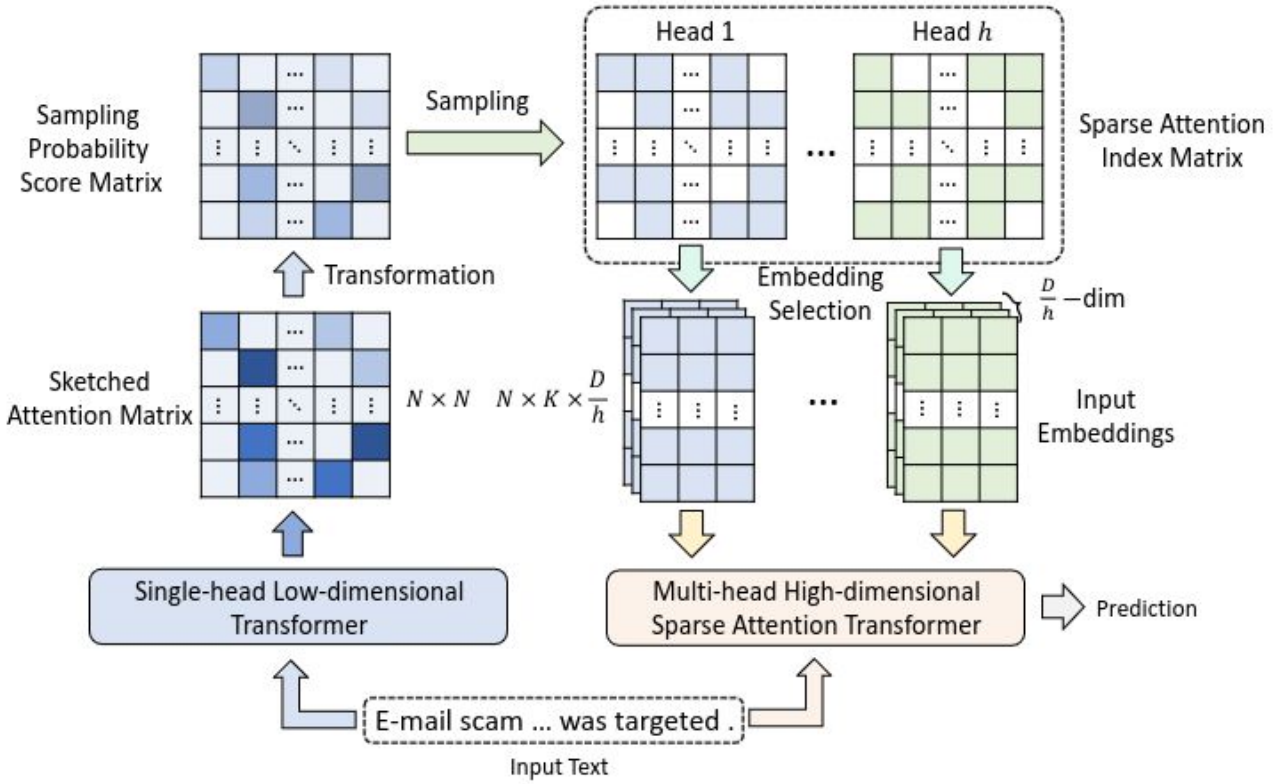
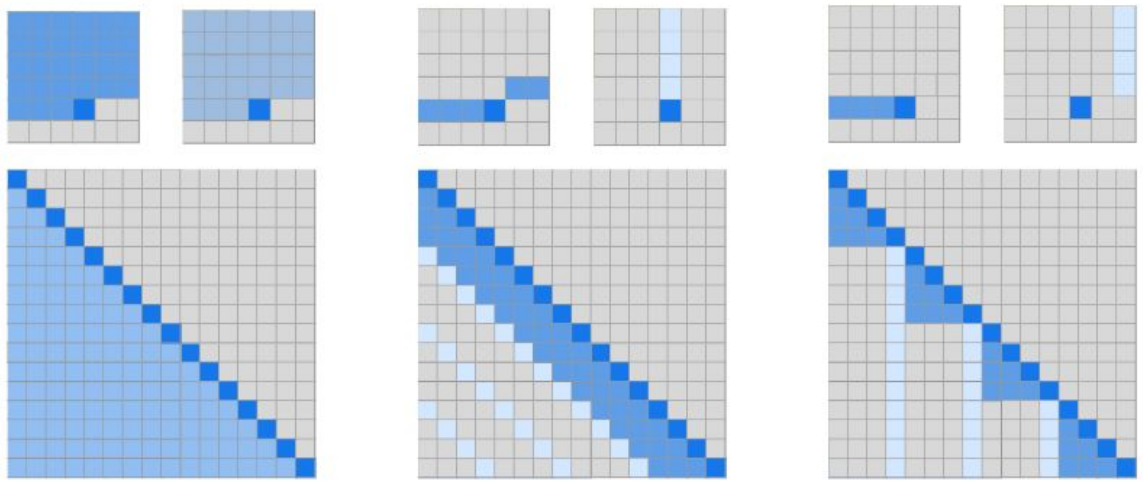


Figure 2: The overall framework of *Smart Bird*.

Sparse Transformer



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Figure 3. Two 2d factorized attention schemes we evaluated in comparison to the full attention of a standard Transformer (a). The top row indicates, for an example 6x6 image, which positions two attention heads receive as input when computing a given output. The bottom row shows the connectivity matrix (not to scale) between all such outputs (rows) and inputs (columns). Sparsity in the connectivity matrix can lead to significantly faster computation. In (b) and (c), full connectivity between elements is preserved when the two heads are computed sequentially. We tested whether such factorizations could match in performance the rich connectivity patterns of Figure 2.

Bertology

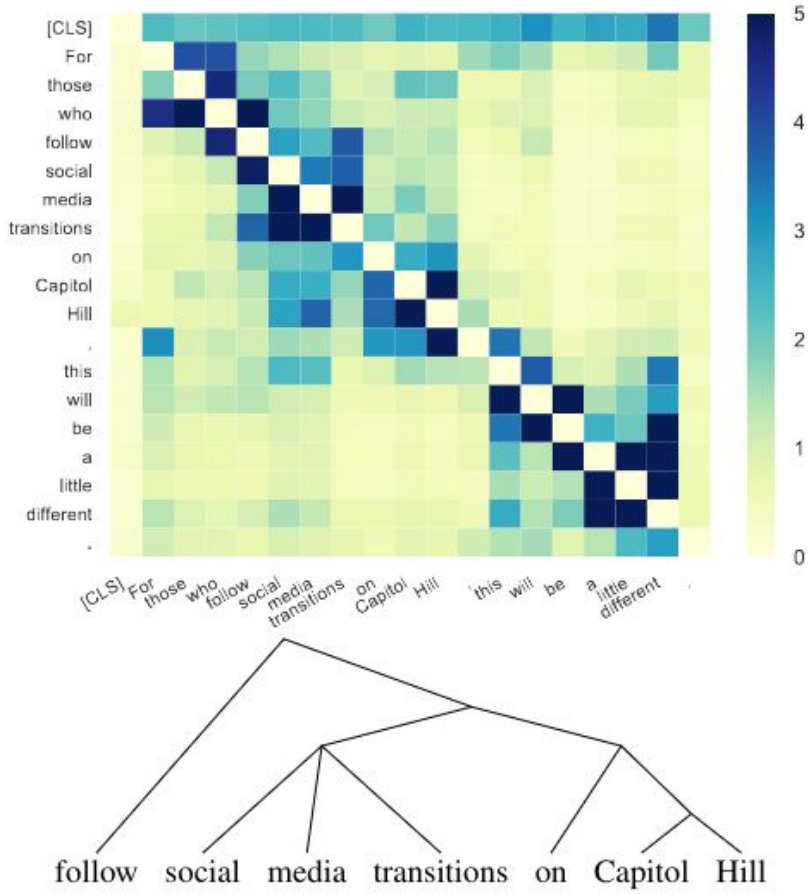


Figure 1: Parameter-free probe for syntactic knowledge: words sharing syntactic subtrees have larger impact on each other in the MLM prediction (Wu et al., 2020)

Bertology

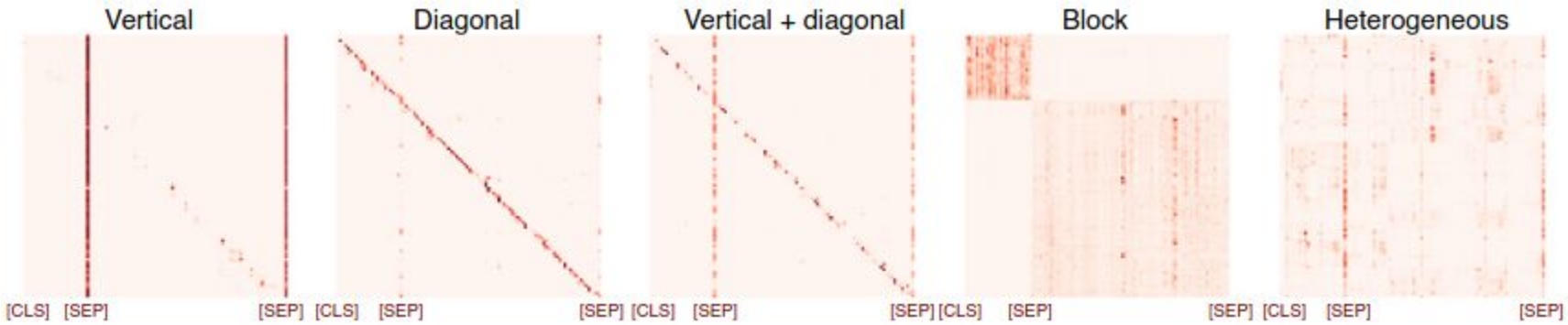


Figure 3: Attention patterns in BERT (Kovaleva et al., 2019)

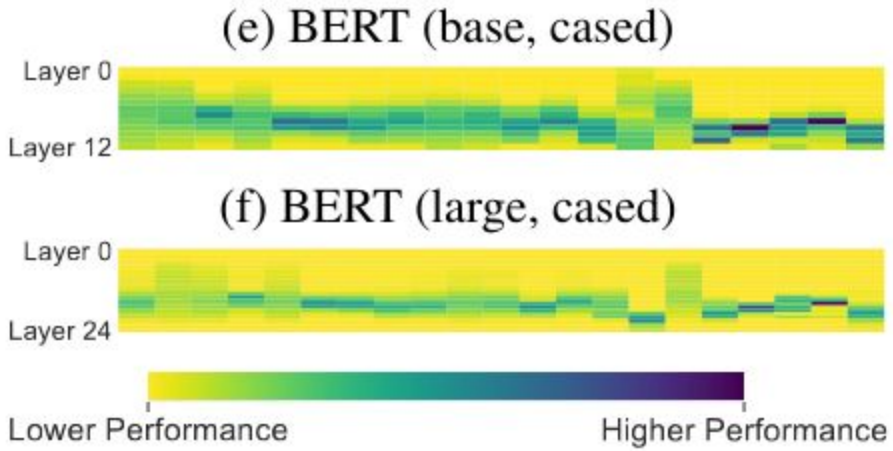
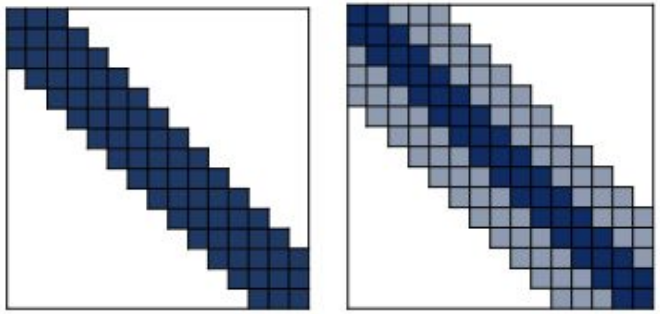


Figure 4: BERT layer transferability (columns correspond to probing tasks, Liu et al. (2019a).

Poolingformer

- Paper released in May, 2021 by Microsoft/Sichuan University
- Paper proposes a pooled sliding attention to gain context further away from token of interest



(a) Single-level local attention (b) Two-level pooling attention

Figure 1. (a): The receptive field of single-level local attention (b): The receptive field of our two-level pooling attention.

Table 4. The results on the arXiv test set. We report the results of ROUGE-1 (R-1), ROUGE-2 (R-2), and ROUGE-L (R-L).

Model	R-1	R-2	R-L
Sent-PTR (Pilault et al., 2020)	42.32	15.63	38.06
Extr-Abst-TLM (Pilault et al., 2020)	41.62	14.69	38.03
PEGASUS (Zhang et al., 2020)	44.21	16.95	38.83
Dancer (Gidiotis & Tsoumakas, 2020)	45.01	17.60	40.56
BigBird (Zaheer et al., 2020)	46.63	19.02	41.77
LED_{4k} (Beltagy et al., 2020)	44.40	17.94	39.76
LED_{16k} (Beltagy et al., 2020)	46.63	19.62	41.83
$\text{Poolingformer}_{4k}$	47.86	19.54	42.35
$\text{Poolingformer}_{16k}$	**48.47**	**20.23**	**42.69**

https://arxiv.org/abs/2105.04371

Questions + Comments?

Resources used

http://cs231n.stanford.edu/slides/2023/lecture_8.pdf

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

https://huggingface.co/blog/how-to-generate

http://jalammar.github.io/illustrated-gpt2/