Language Modeling III: Transformers

CSE 5525: Foundations of Speech and Language Processing

https://shocheen.github.io/cse-5525-fall-2025/

The Ohio State University

Sachin Kumar (kumar.1145@osu.edu)

Slide Credits: Chris Tanner

Logistics

- Homework 2 is due date next week.
  - Any thoughts, questions, concerns?

- Final project: have you formed teams already?
  - A project proposal will be due first week of October.
  - Look at last year's course website for example projects, will also post other ideas on Canvas: https://shocheen.github.io/cse-5525-fall-2025/

Recap

- Feedforward Neural Language Model
  - Need to make unreasonable assumptions and lose information from the long context
- Recurrent Neural Network (RNN)
  - Infinitely long context in theory --- hard to train (exploding/vanishing gradients), difficult to parallelize, and could be infeasible (memorize a variable length sequence in a fixed length vector).
  - Encoder-decoder architecture
- RNN + Attention
  - Solves the last issue, still hard to train efficiently (on GPUs).
- Attention is all you need [will continue today]
  - Transformer Architecture

Transformers

- Replace the linear part of RNNs with **self-attention**

- Introduce **residual connections + layernorm** to improve gradient flow (avoid gradient vanishing issues)

- Introduce **positional embeddings** to encode sequential order

Outline

- Self-Attention
- Transformer Encoder
- Transformer Decoder
- Language Modeling With Transformers

Outline

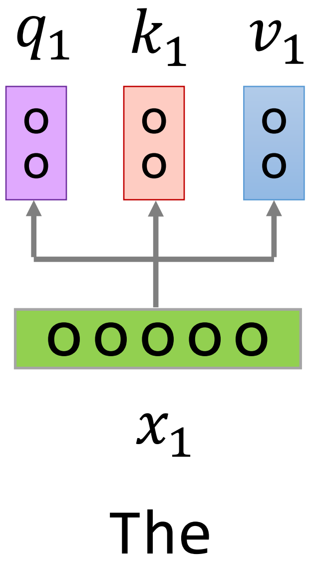
- Self-Attention
- Transformer Encoder
- Transformer Decoder
- Language Modeling With Transformers

Defining Self-Attention

- **Terminology:**
  - **Query**: to match others
  - **Key**: to be matched
  - **Value**: information to be extracted

- **Definition:** Given a set of vector **values**, and a vector **query**, *attention* is a technique to compute a weighted sum of the **value**, dependent on the **keys**.

[Vaswani et al. 2017: https://arxiv.org/abs/1706.03762]



q: query (to match others)

$$q_t = W^q x_t$$

k: key (to be matched)

$$k_t = W^k x_t$$

v: value (information to be extracted)

$$v_t = W^v x_t$$

q: query (to match others)

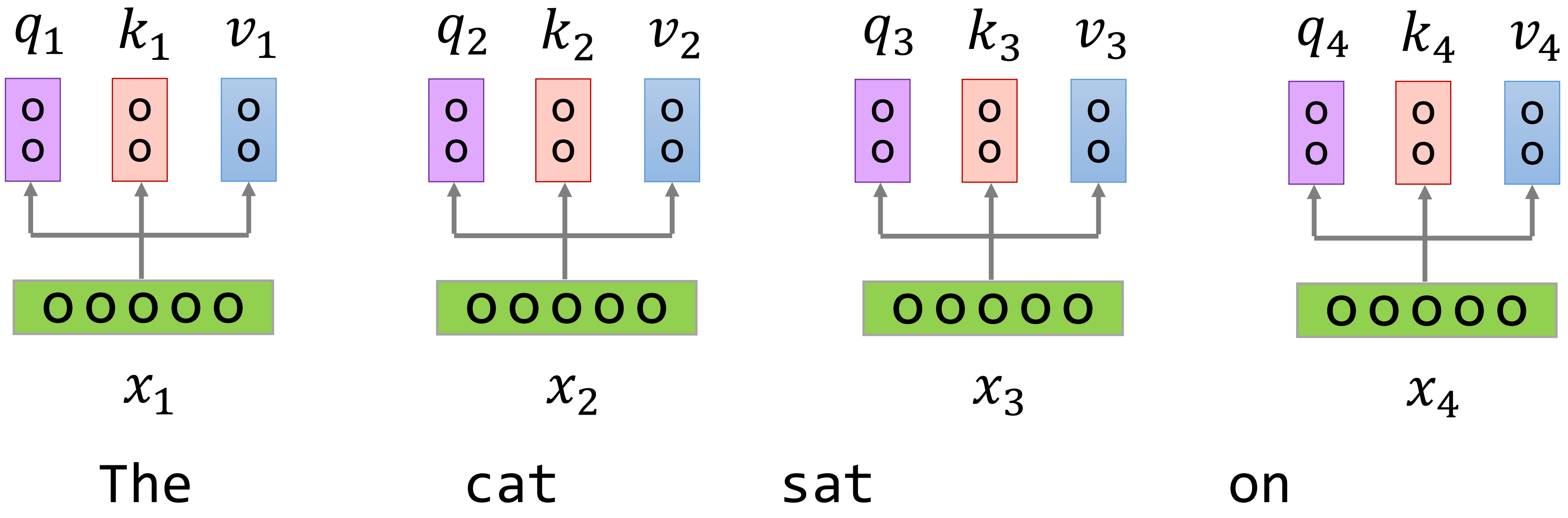
$$q_t = W^q x_t$$

k: key (to be matched)

$$k_t = W^k x_t$$

v: value (information to be extracted)

$$v_t = W^v x_t$$



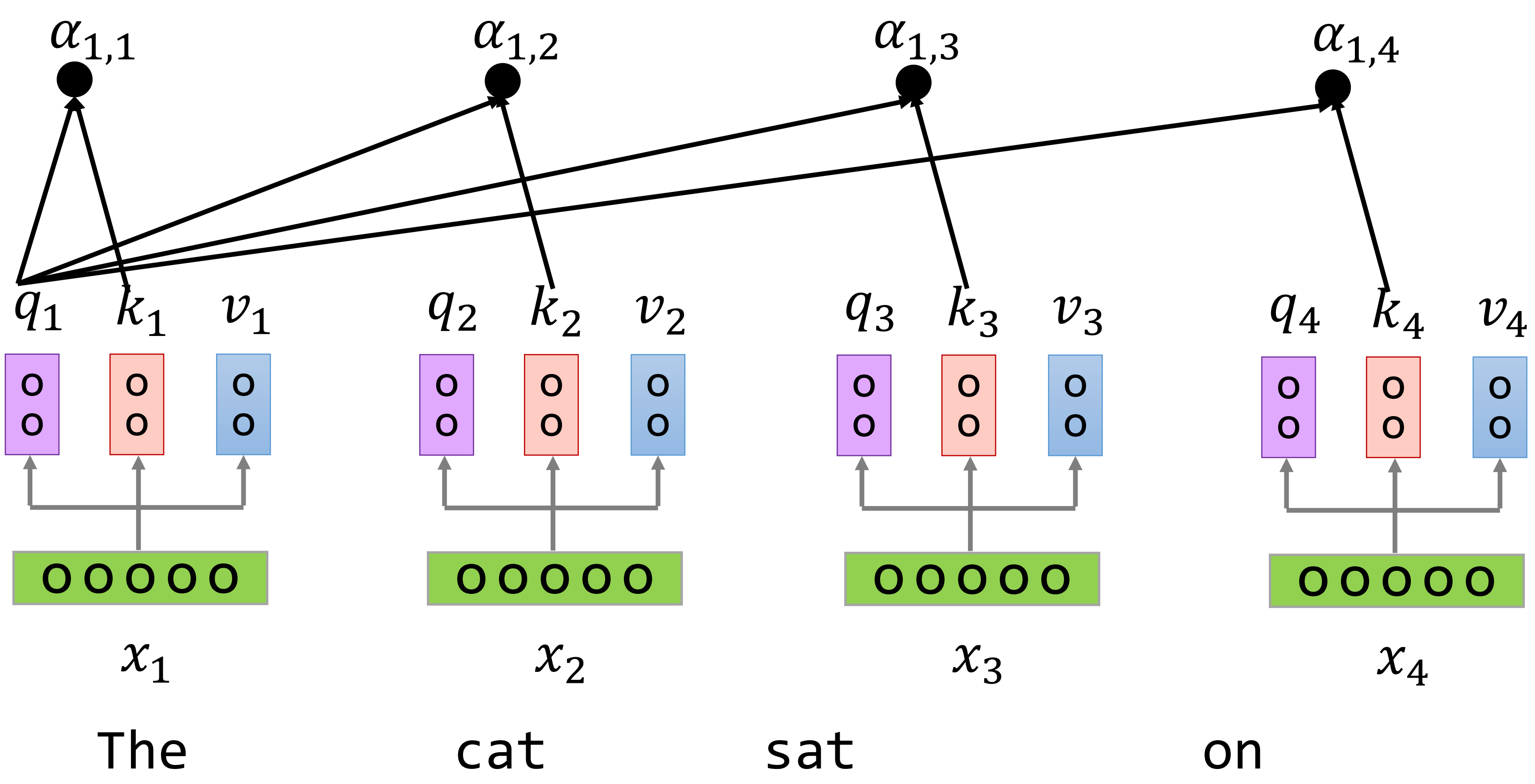
$$\alpha_{1,t} = \underbrace{q^1 \cdot k^t / \alpha}_{\text{Scaled dot product}}$$

q: query (to match others)

k: key (to be matched)

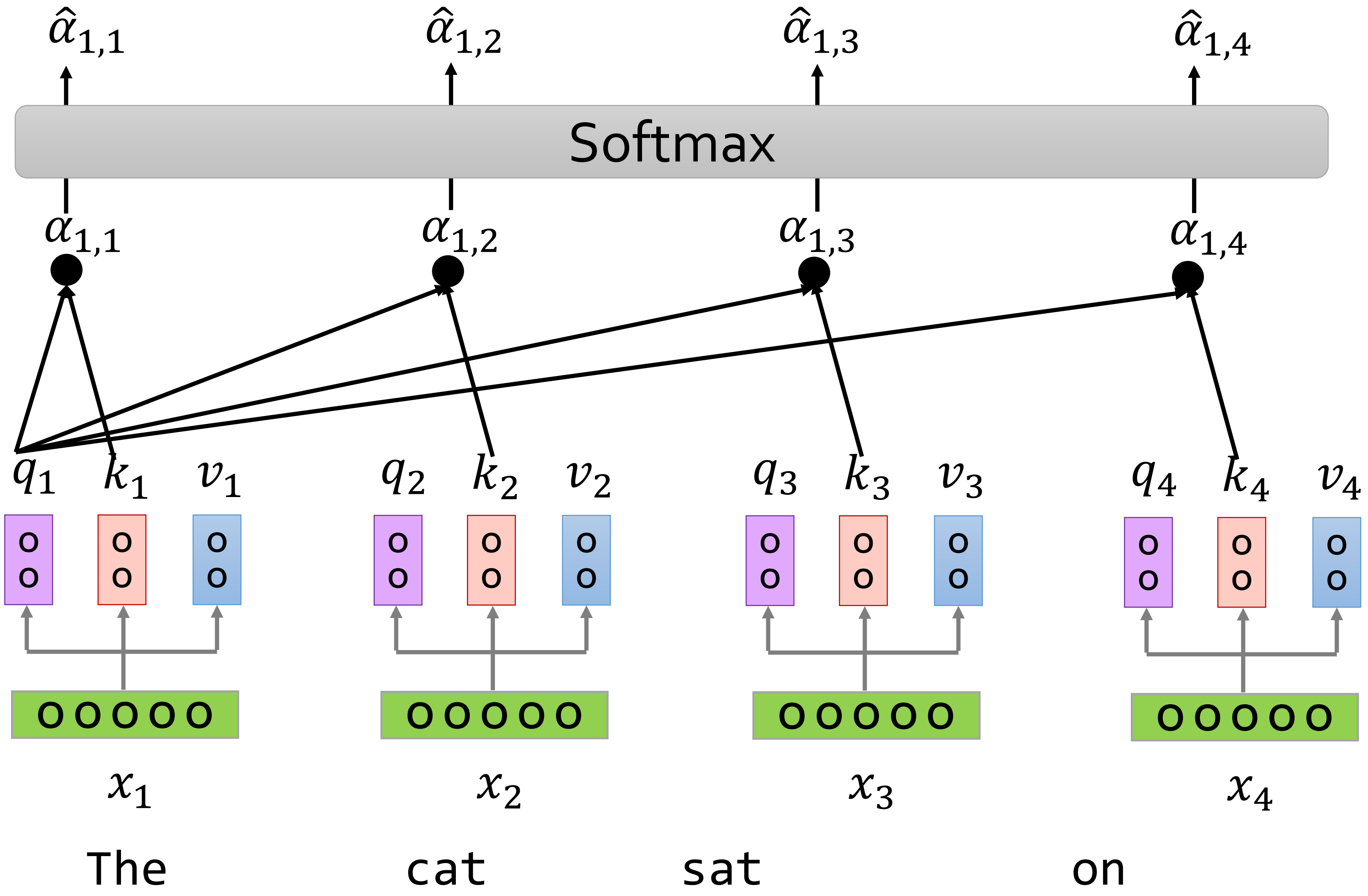
v: value (information to be extracted)

How much should "The" attend to other positions?



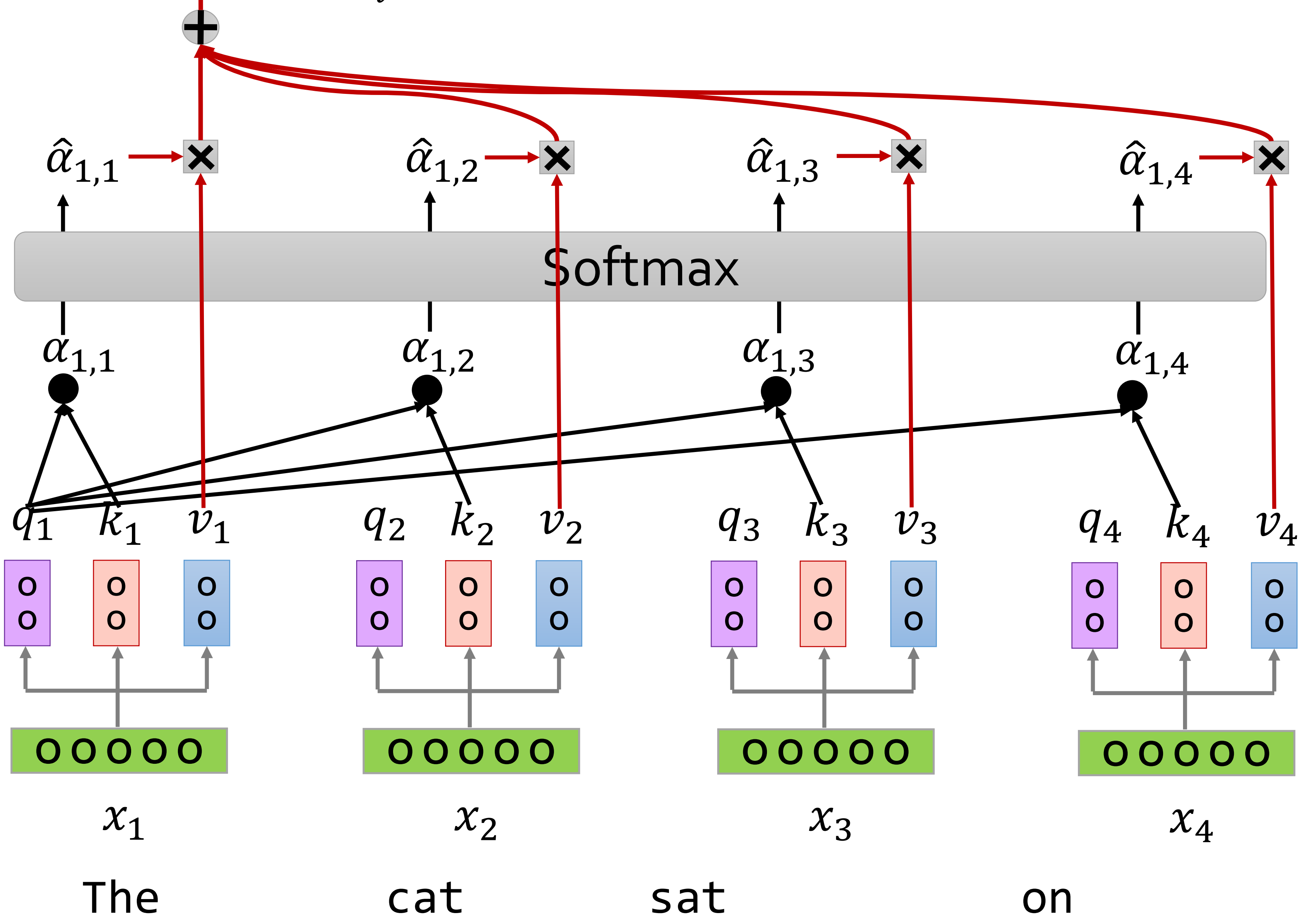
$$\sigma(z)_t = \frac{exp(z_t)}{\sum_j exp(z_j)}$$

How much should "The" attend to other positions?

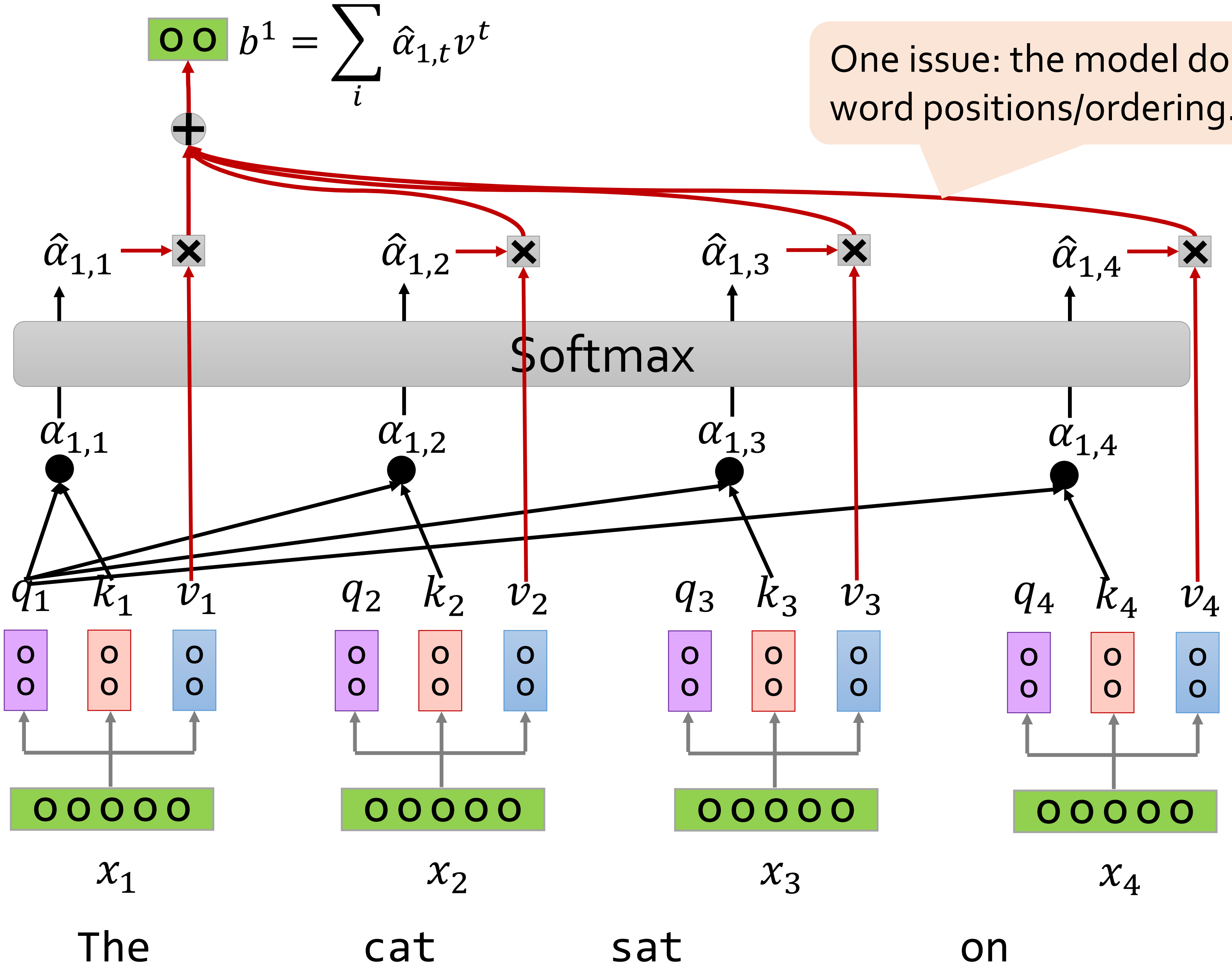


$$b^1 = \sum_i \hat{\alpha}_{1,t} v^t$$

Representation of “The” given the attention weights



One issue: the model doesn't know word positions/ordering.



Self-Attention

Step 1: Our Self-Attention Head I has just 3 weight matrices W_q, W_k, W_v in total. These same 3 weight matrices are multiplied by each x_i to create all vectors:

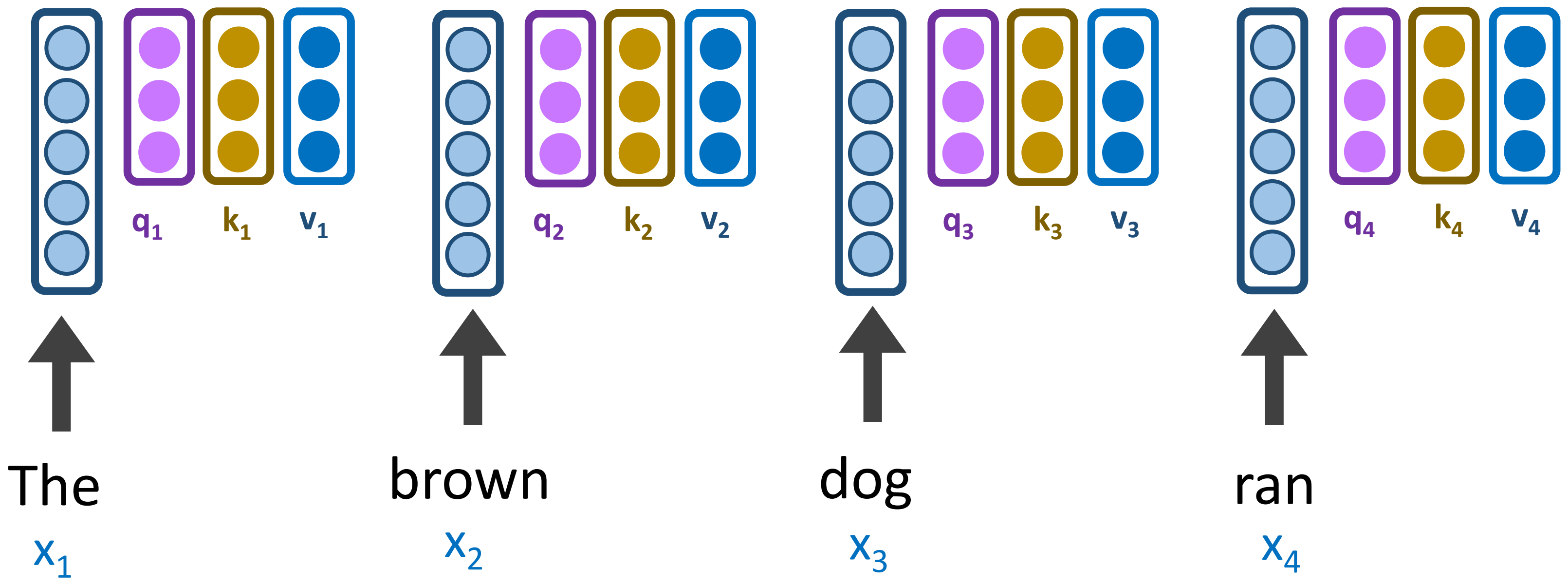
$$q_i = w_q x_i$$

$$k_i = w_k x_i$$

$$v_i = w_v x_i$$

Under the hood, each x_i has 3 small, associated vectors. For example, x_1 has:

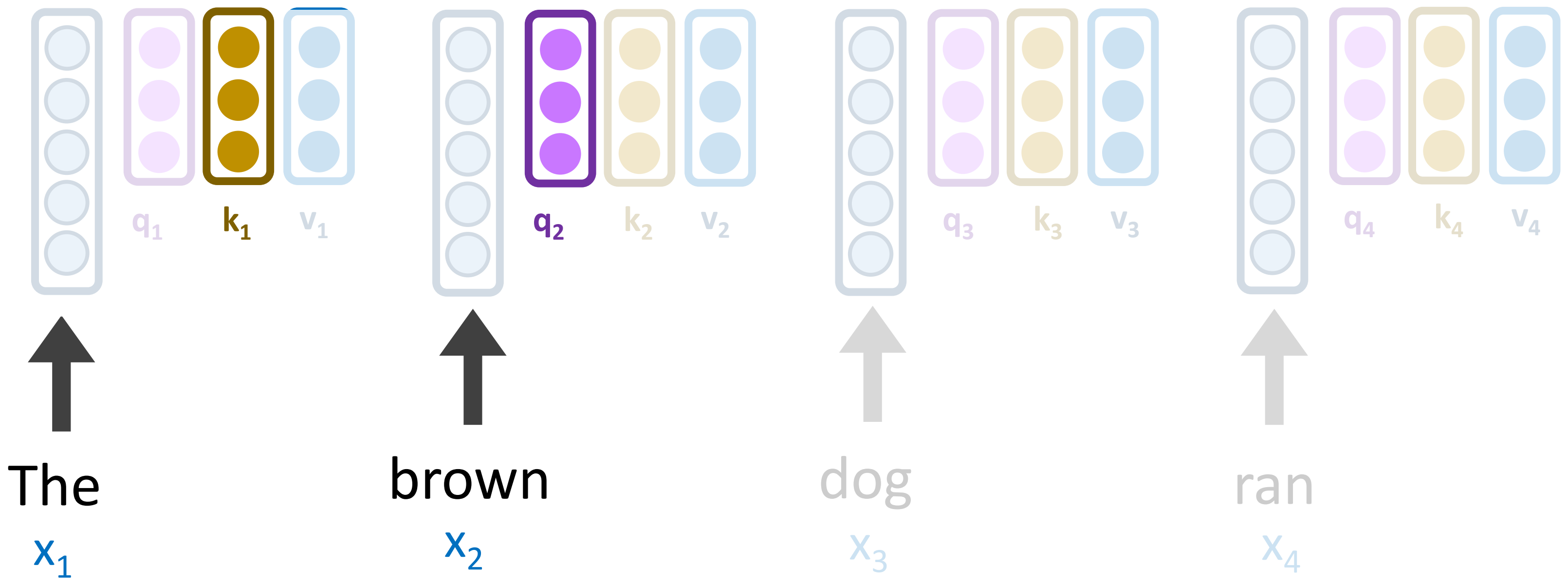
- Query $\mathbf{q_1}$
- Key $\mathbf{k_1}$
- Value $\mathbf{v_1}$



Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" $\mathbf{v}_i$

$s_1 = q_2 \cdot k_1 = 92$

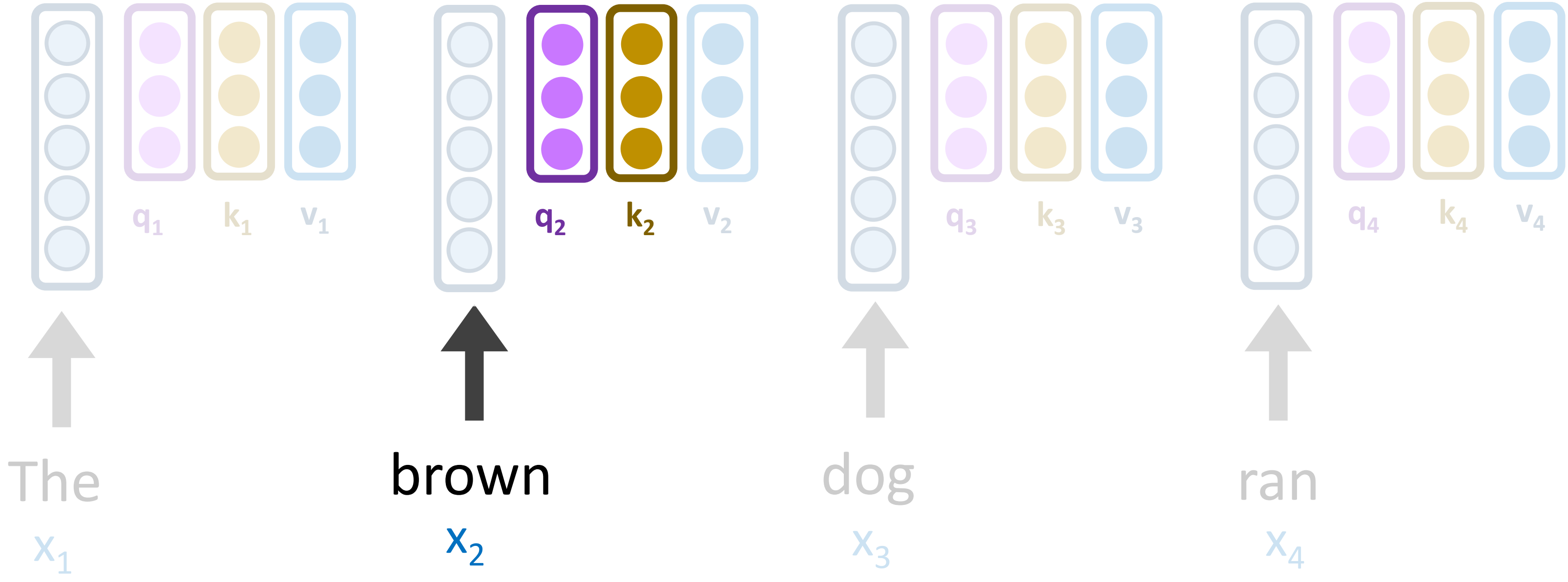


Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" $\mathbf{v_i}$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$



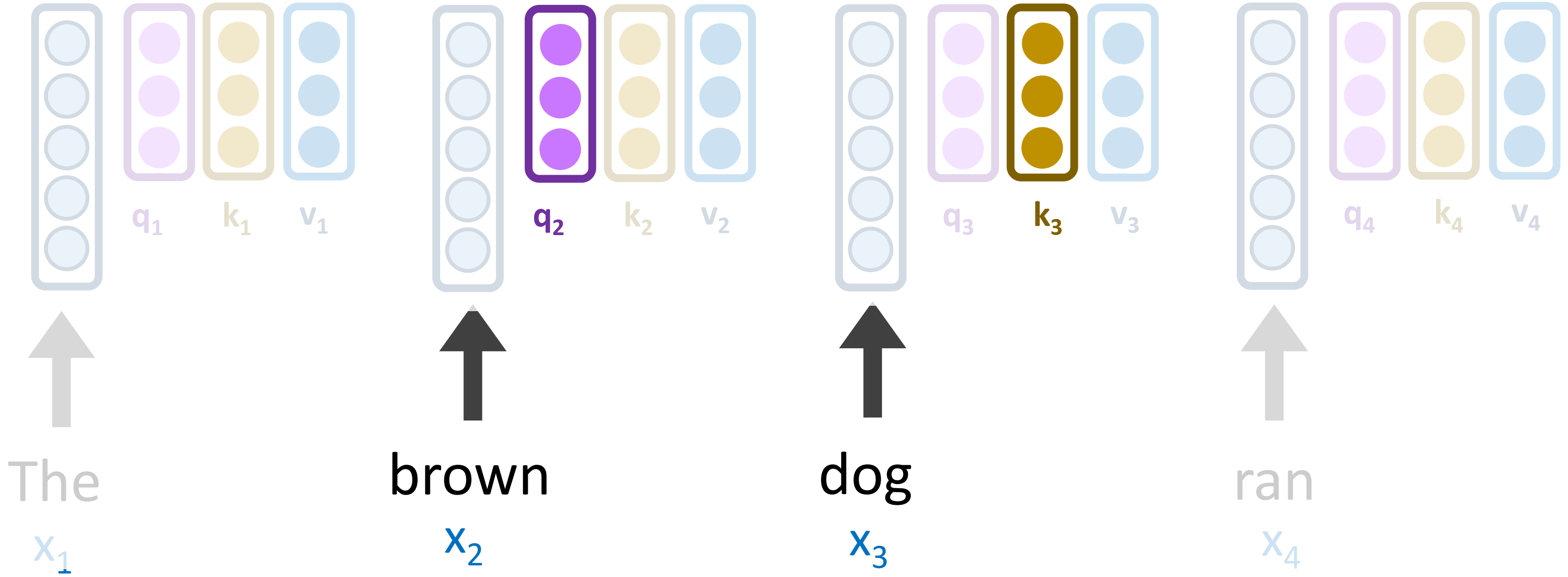
Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" $\mathbf{v}_i$

$s_3 = q_2 \cdot k_3 = 22$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$



Self-Attention

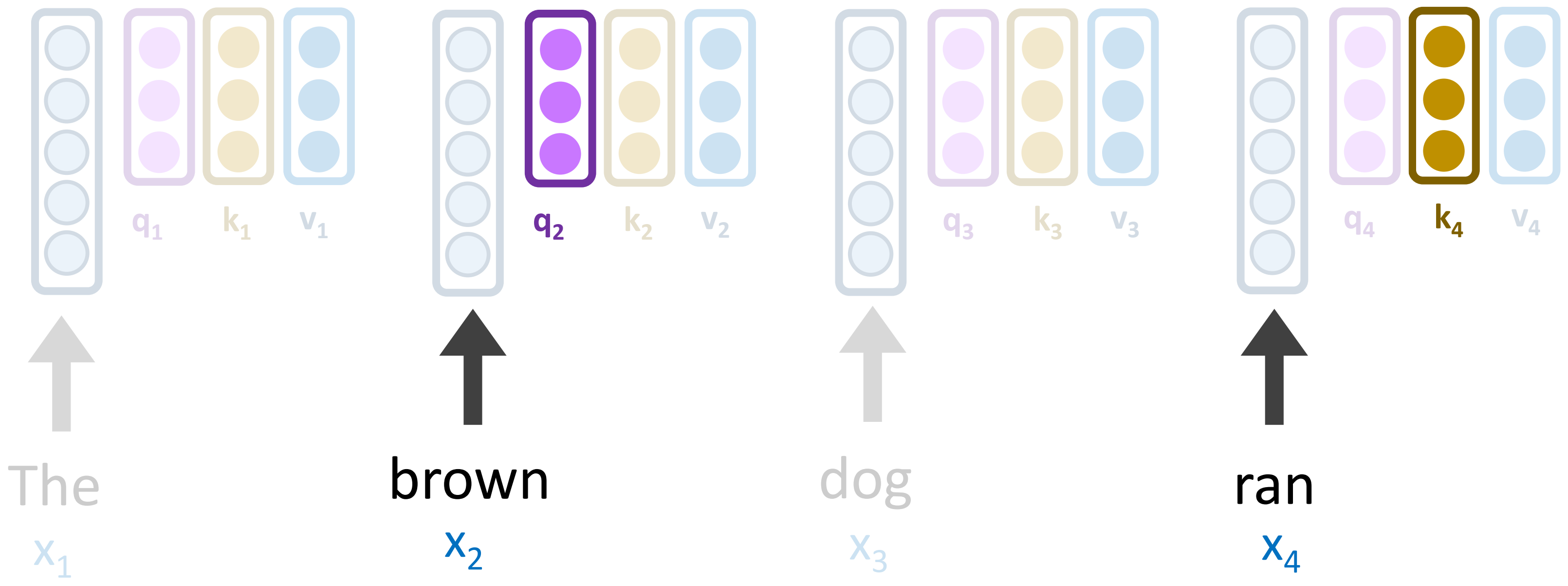
Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" $\mathbf{v}_i$

$s_4 = \mathbf{q}_2 \cdot \mathbf{k}_4 = 8$

$s_3 = \mathbf{q}_2 \cdot \mathbf{k}_3 = 22$

$s_2 = \mathbf{q}_2 \cdot \mathbf{k}_2 = 124$

$s_1 = \mathbf{q}_2 \cdot \mathbf{k}_1 = 92$



Self-Attention

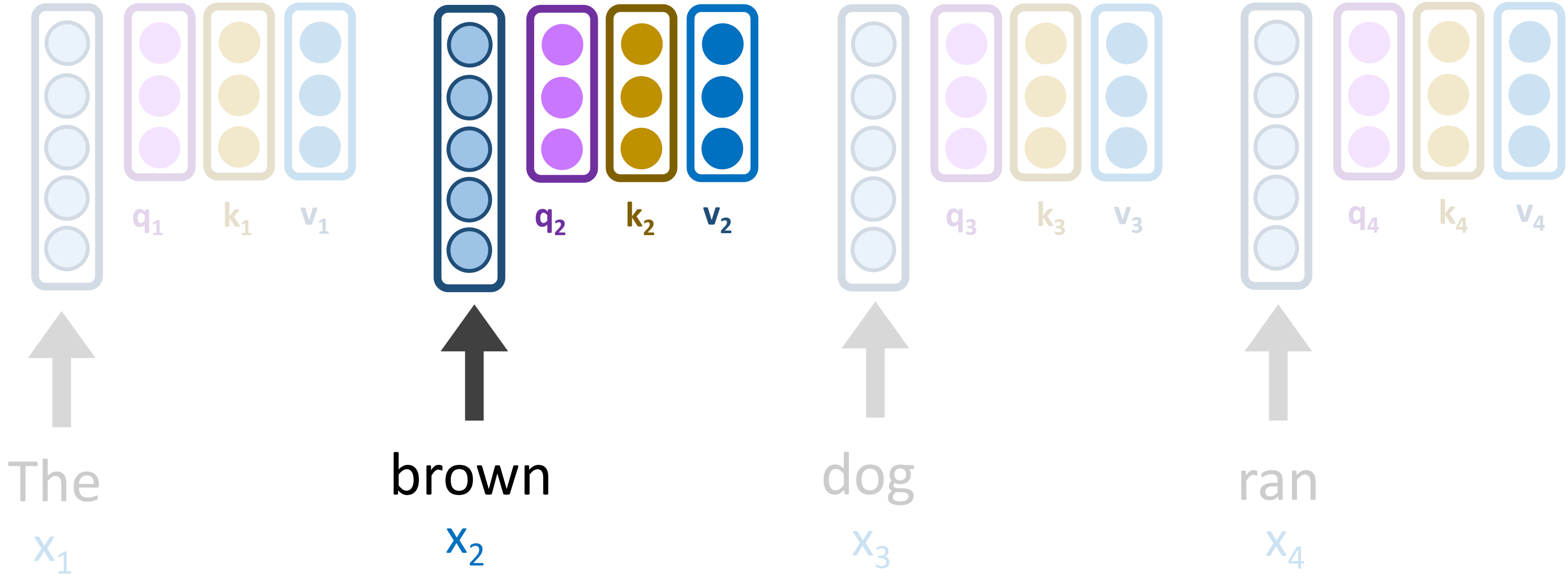
Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

$s_4 = q_2 \cdot k_4 = 8$ $\qquad$ $a_4 = \sigma(s_4/8) = 0$

$s_3 = q_2 \cdot k_3 = 22$ $\qquad$ $a_3 = \sigma(s_3/8) = .01$

$s_2 = q_2 \cdot k_2 = 124$ $\qquad$ $a_2 = \sigma(s_2/8) = .91$

$s_1 = q_2 \cdot k_1 = 92$ $\qquad$ $a_1 = \sigma(s_1/8) = .08$



Self-Attention

Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

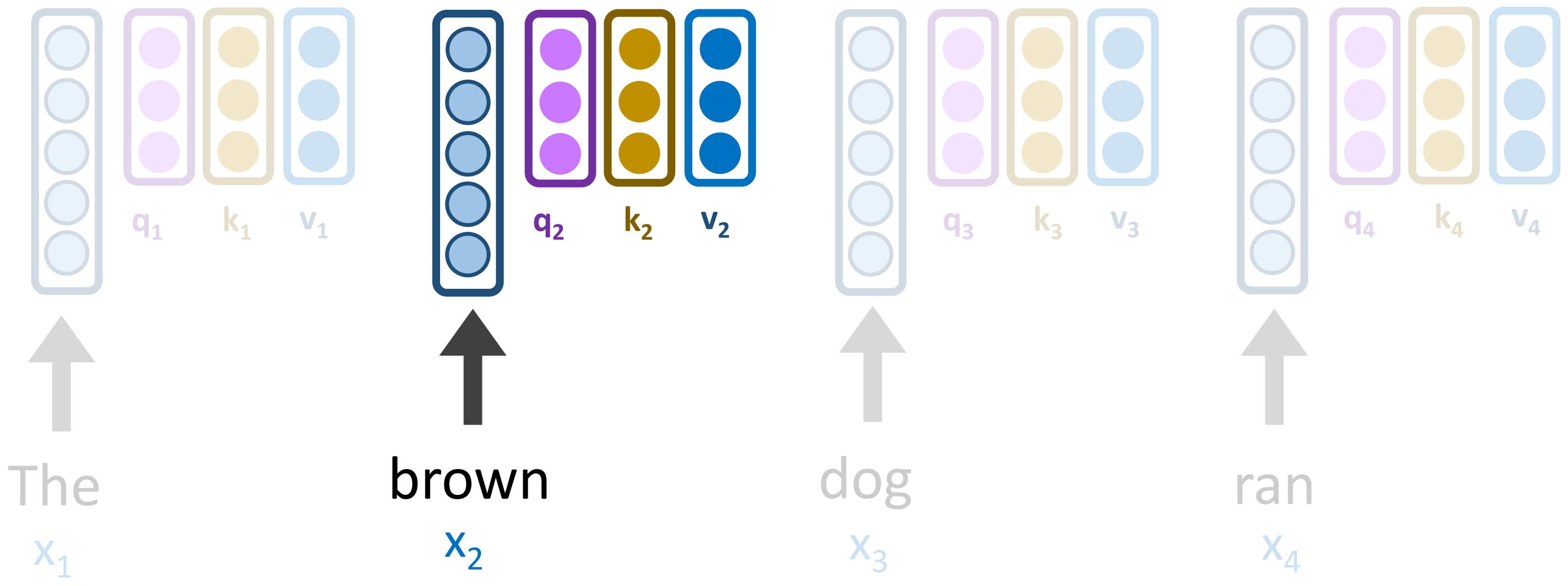
$s_4 = q_2 \cdot k_4 = 8$ $\quad$ $a_4 = \sigma(s_4/8) = 0$

$s_3 = q_2 \cdot k_3 = 22$ $\quad$ $a_3 = \sigma(s_3/8) = .01$

$s_2 = q_2 \cdot k_2 = 124$ $\quad$ $a_2 = \sigma(s_2/8) = .91$

$s_1 = q_2 \cdot k_1 = 92$ $\quad$ $a_1 = \sigma(s_1/8) = .08$

Instead of these a_i values directly weighting our original x_i word vectors, they directly weight our v_i vectors.



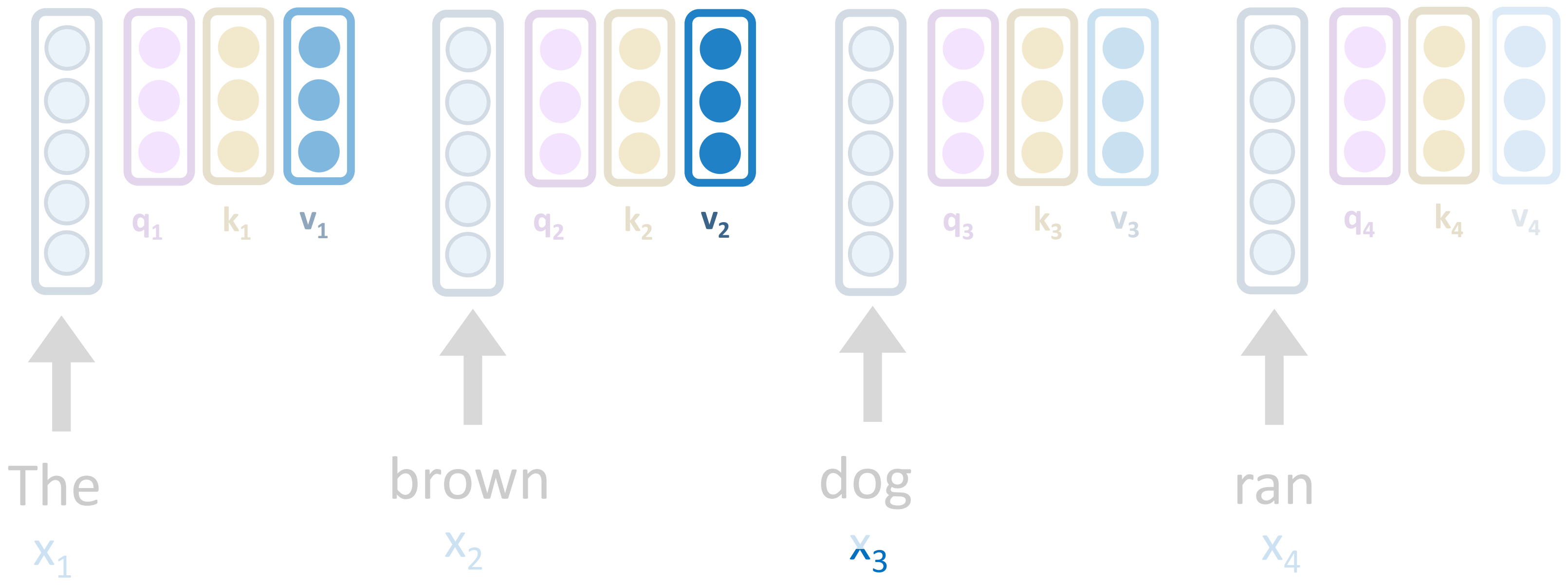
Self-Attention

Step 4: Let's weight our $\mathbf{v}_i$ vectors and simply sum them up!

z_2

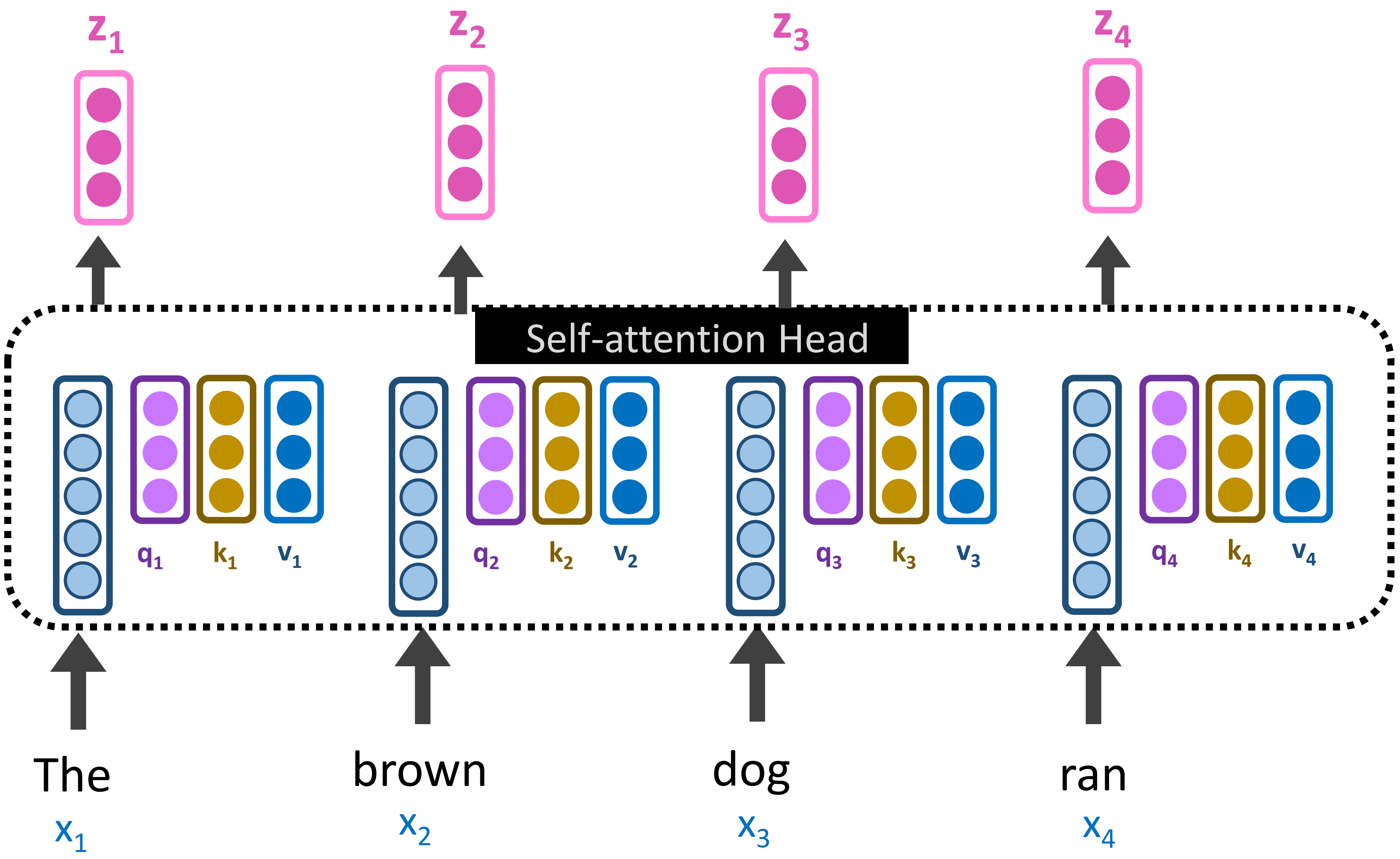
$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

$$= 0.08 \cdot v_1 + 0.91 \cdot v_2 + 0.01 \cdot v_3 + 0 \cdot v_4$$



Self-Attention

Tada! Now we have great, new representations z_i via a self-attention head



Takeaway:

Self-Attention is powerful; allows us to create great, context-aware representations

Self-Attention

$$\boldsymbol{b} = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\alpha}\right)V$$

hardmaru
@hardmaru

The most important formula in deep learning after 2018

Self-Attention

What is self-attention? Self-attention calculates a weighted average of feature representations with the weight proportional to a similarity score between pairs of representations. Formally, an input sequence of n tokens of dimensions d, $X \in \mathbf{R}^{n\times d}$, is projected using three matrices $W_Q \in \mathbf{R}^{d\times d_q}$, $W_K \in \mathbf{R}^{d\times d_k}$, and $W_V \in \mathbf{R}^{d\times d_v}$ to extract feature representations Q, K, and V, referred to as query, key, and value respectively with $d_k = d_q$. The outputs Q, K, V are computed as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \qquad (1)$$

So, self-attention can be written as,

$$S = D(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \qquad (2)$$

where softmax denotes a *row-wise* softmax normalization function. Thus, each element in S depends on all other elements in the same row.

9:08 PM · Feb 9, 2021 · Twitter Web App

553 Retweets **42** Quote Tweets **3,338** Likes

Outline

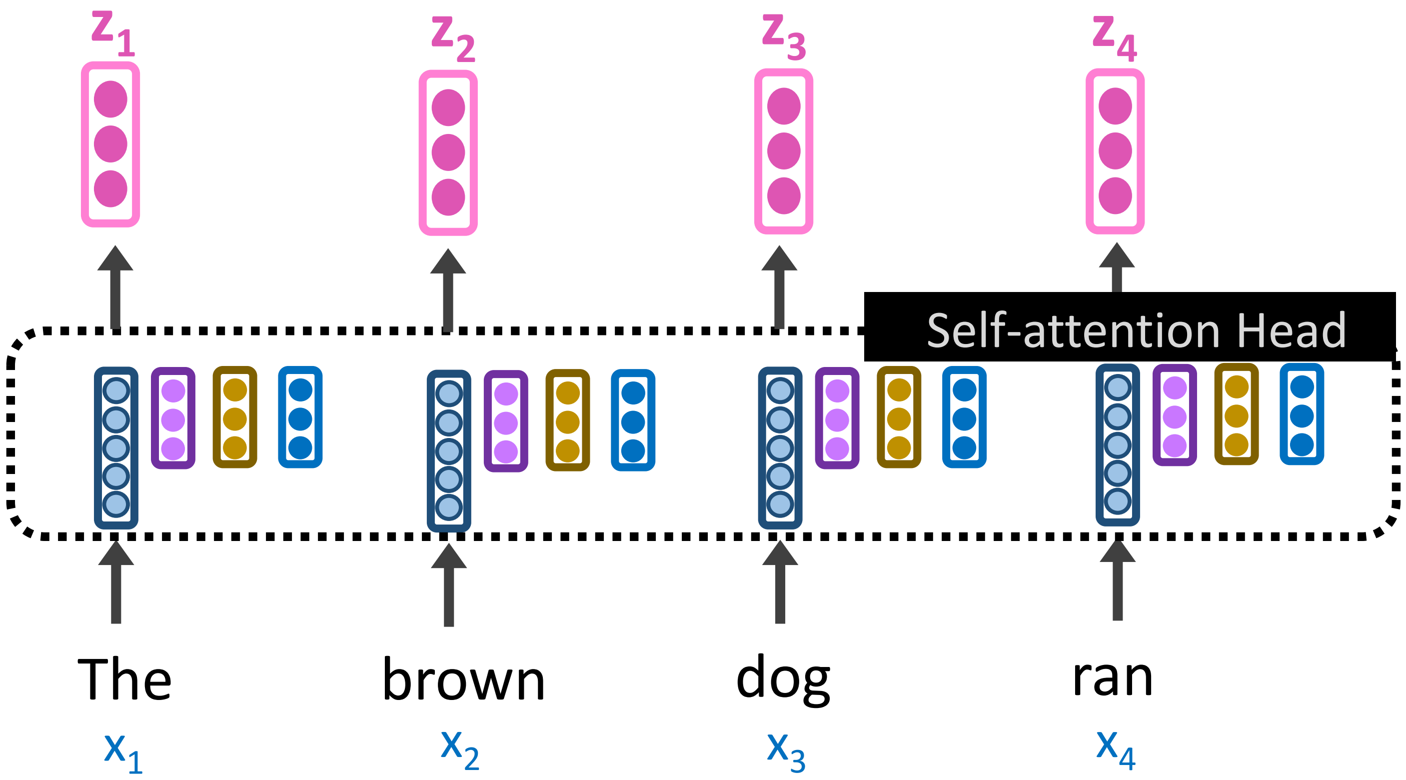
- Self-Attention
- Transformer Encoder
- Transformer Decoder
- Language Modeling With Transformers

Outline

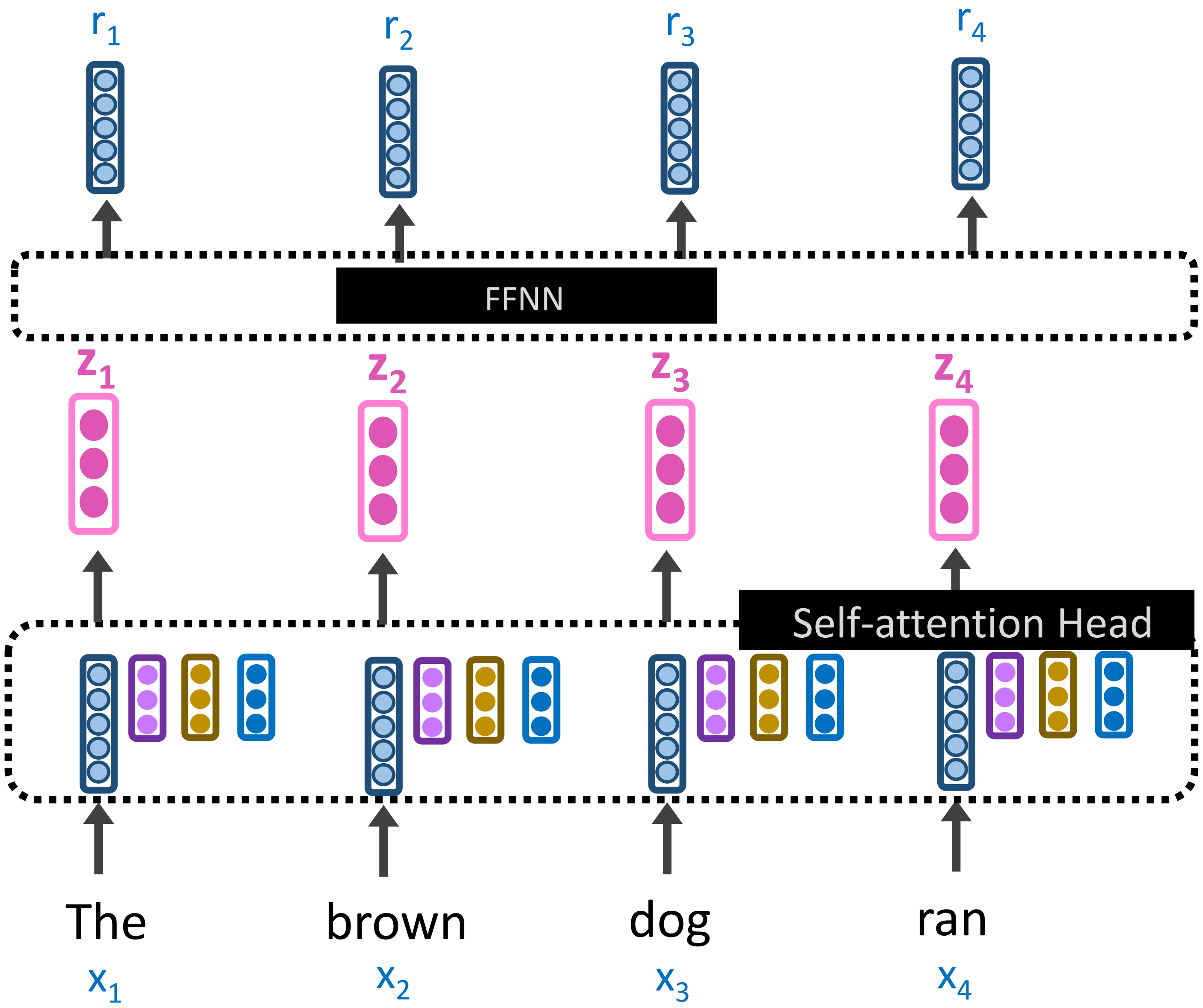
- Self-Attention
- Transformer Encoder
- Transformer Decoder
- Language Modeling With Transformers

Self-Attention

Let's further pass each z_i through a FeedForward NN

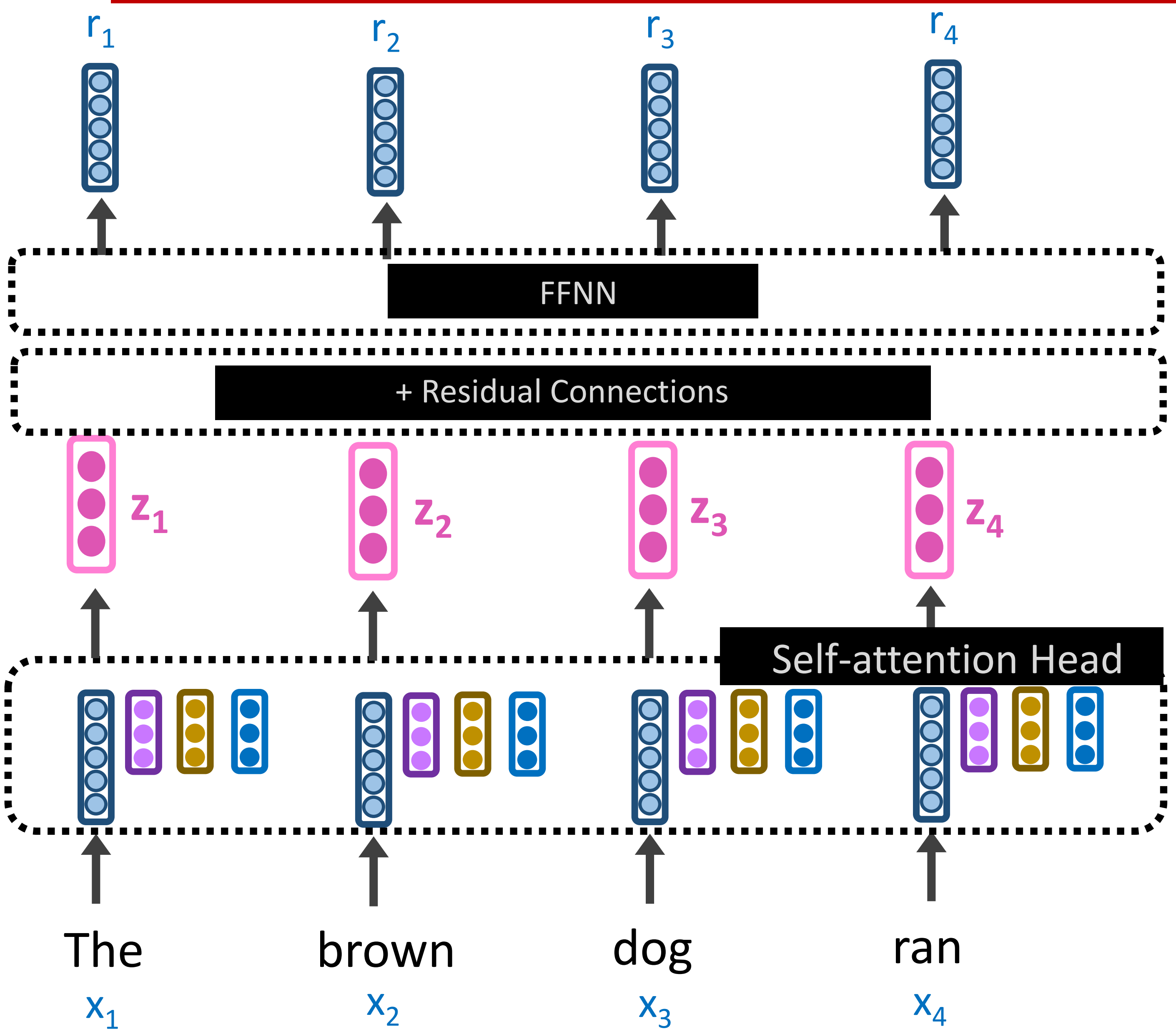


Self-Attention + FFNN



Let's further pass each z_i through a Feed Forward NN

Self-Attention + FFNN + Residual Connections

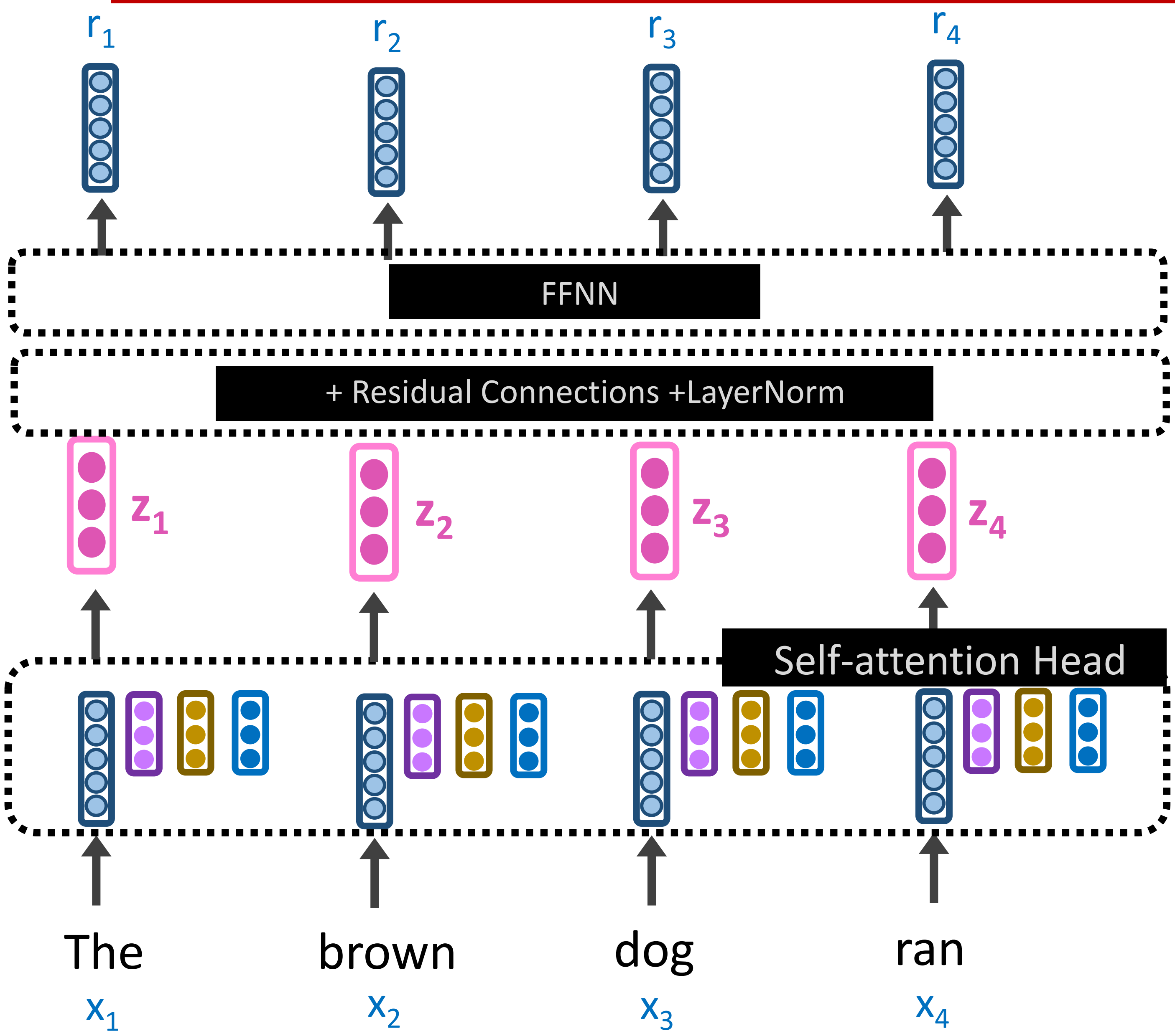


Let's further pass each z_i through a FFNN

We add a residual connection to help ensure relevant info is getting forward passed.

$$v = z + x$$

Self-Attention + FFNN + Residual Connections



Let's further pass each z_i through a FFNN

We add residual connection to help ensure relevant info is getting forward passed.

$$v = z + x$$

We perform LayerNorm to stabilize the network and allow for proper gradient flow. You should do this after the FFNN, too.

Stabilizing Gradient Flow: Residual Connection and Layernorm

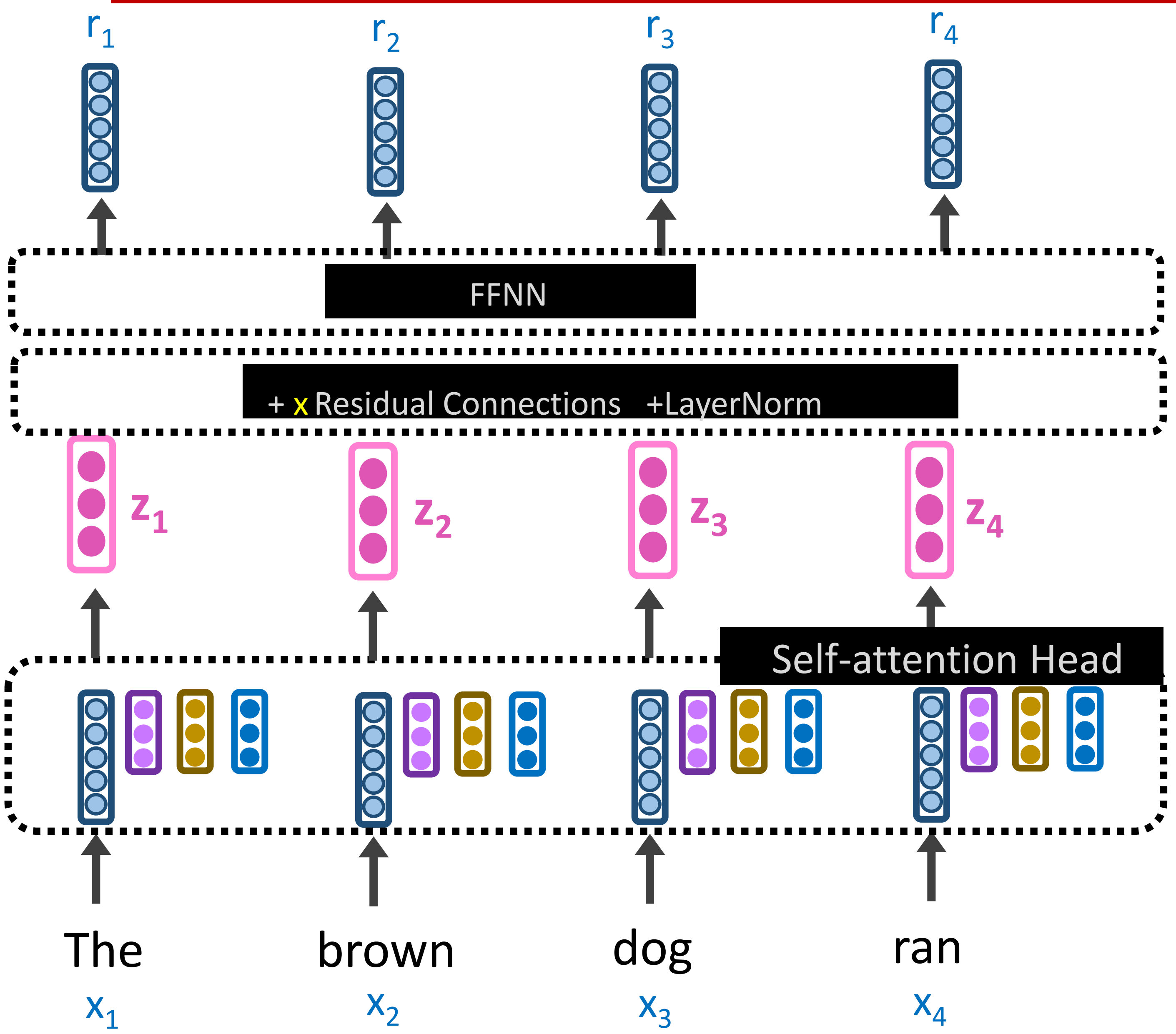
- Residual connection: y = f(x) + x
  - f might be a complex function and gives small gradients wrt x, adding x back to f(x) gives higher values of the gradient

- Layer Normalization (LayerNorm):
  - Another way to prevent vanishing gradients

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

arxiv.org/pdf/2409.12951

Self-Attention + FFNN



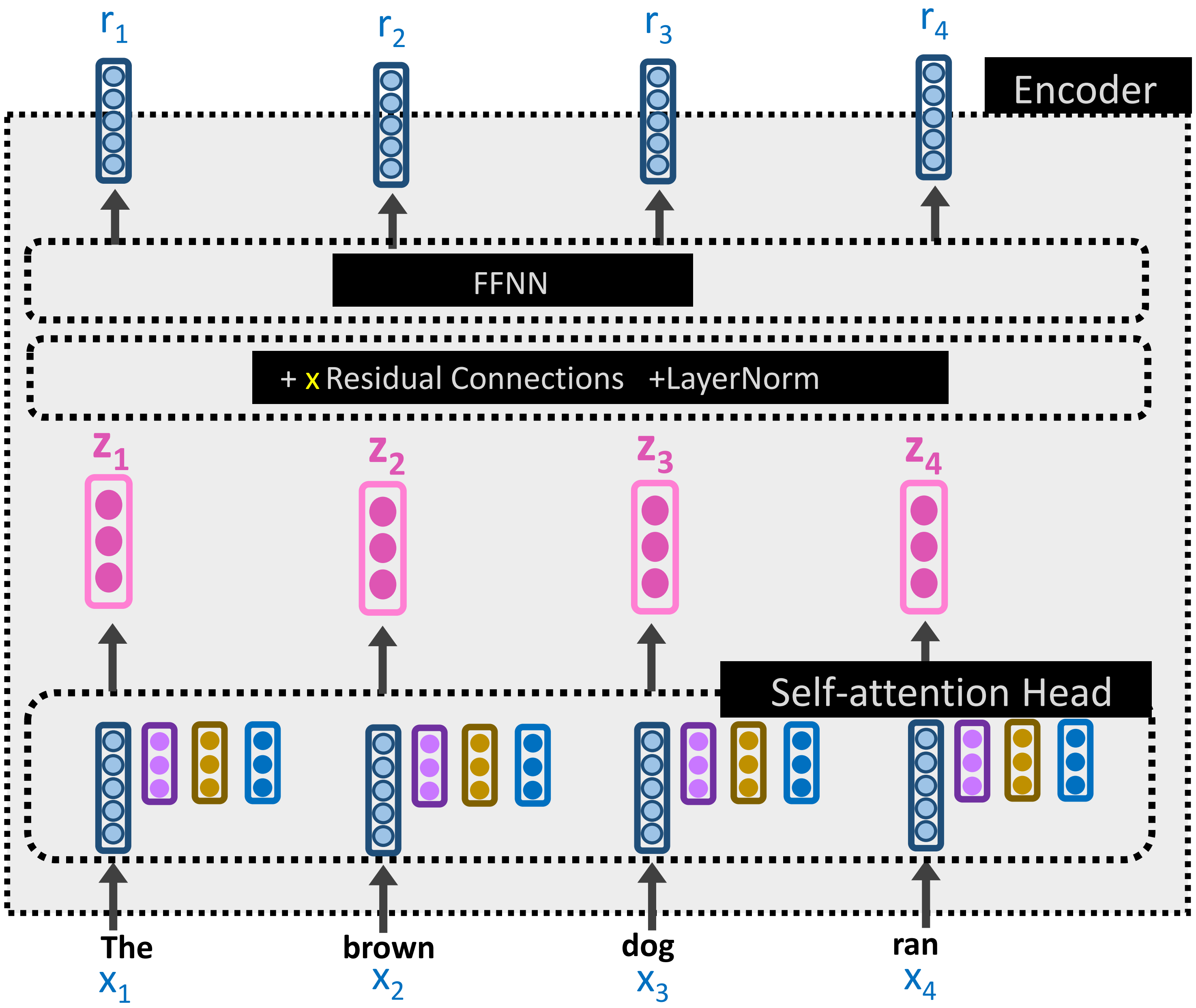
Let's further pass each z_i through a FFNN

We concat w/ a residual connection to help ensure relevant info is getting forward passed.

We perform LayerNorm to stabilize the network and allow for proper gradient flow. You should do this after the FFNN, too.

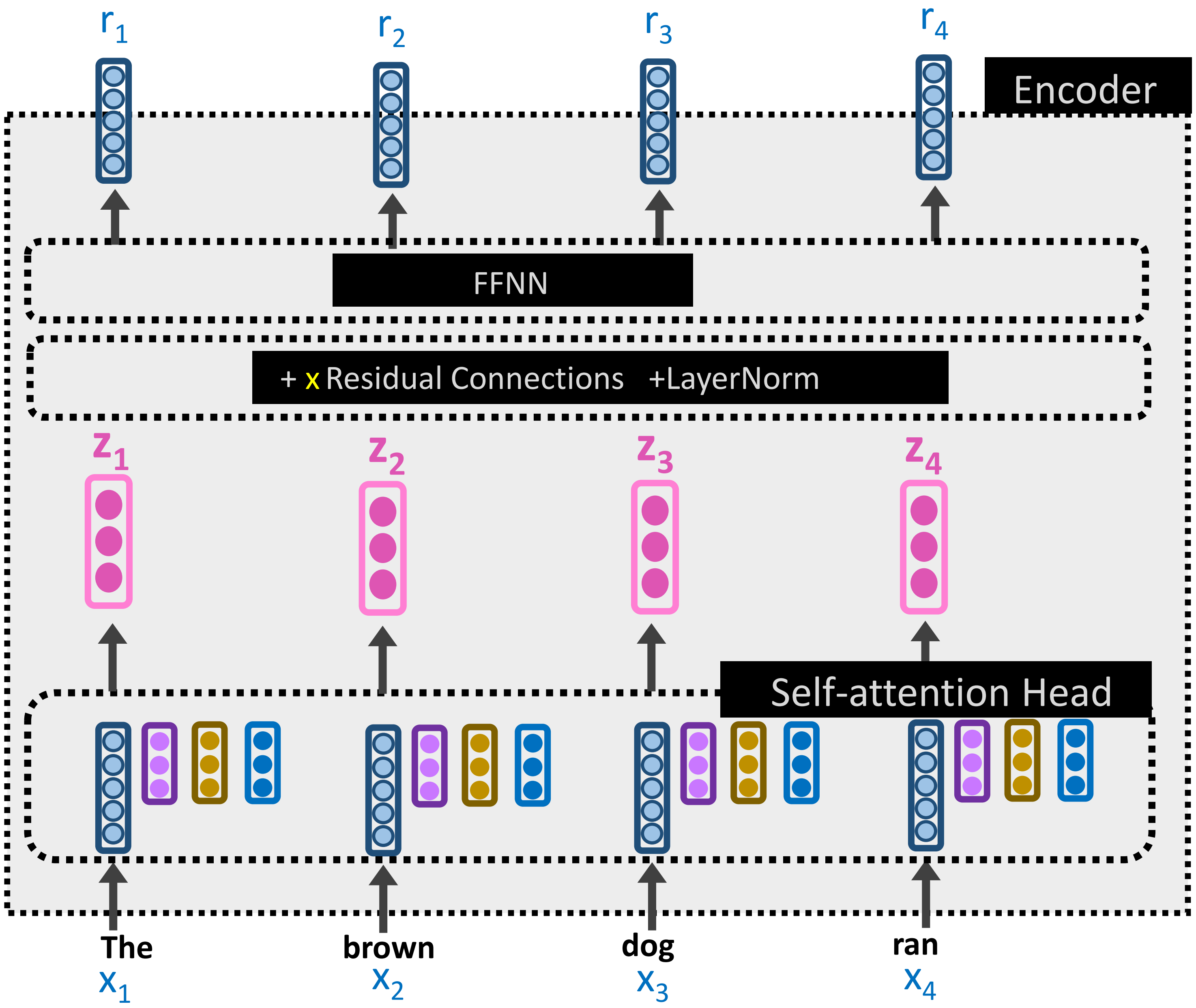
Each z_i can be computed in parallel, unlike RNNs!

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

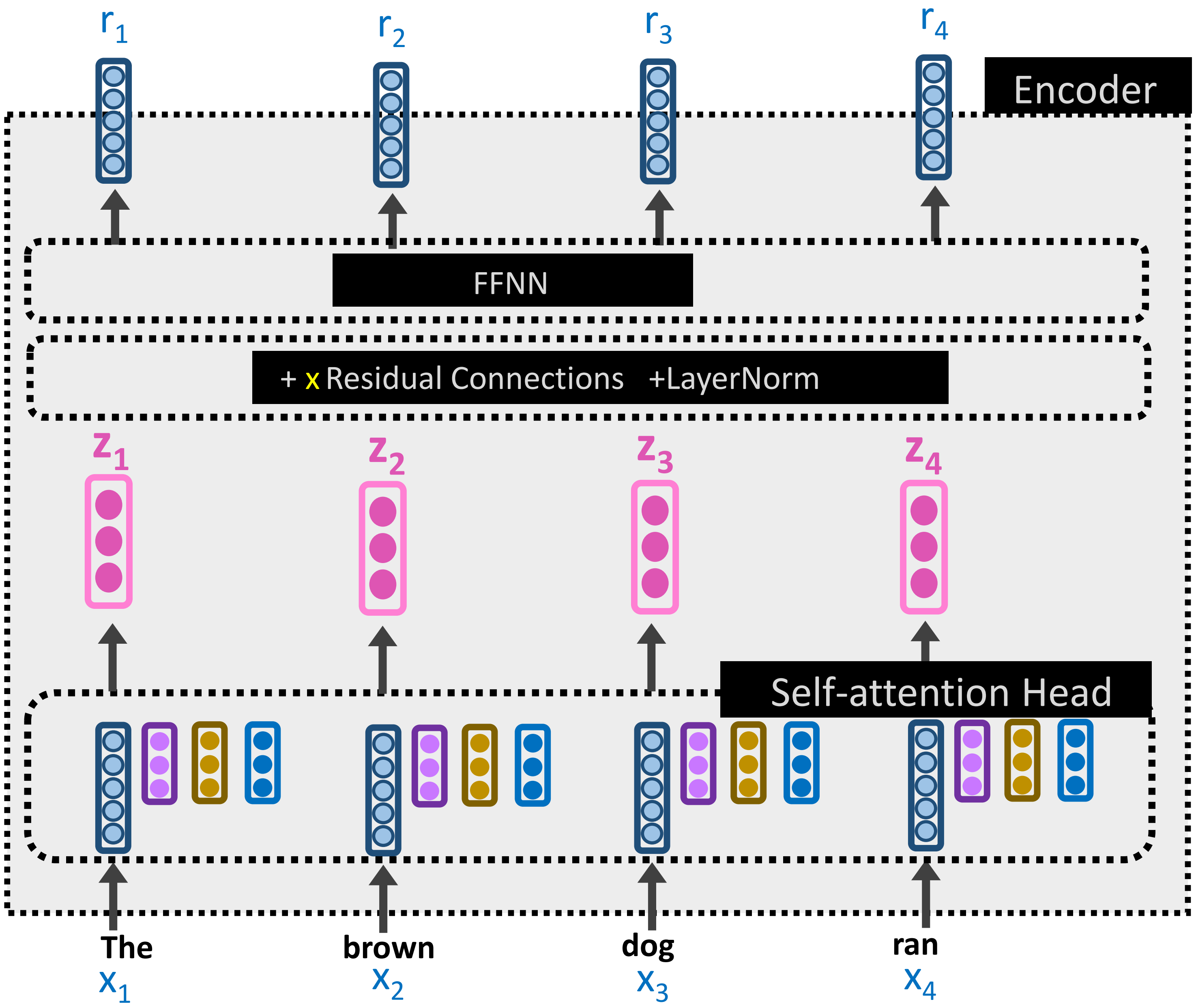
Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

Solution: add to each input word x_i a positional encoding

Input to the model is now $x_i + pos_i$

How to encode position information?

- Self attention doesn't have a way to know whether an input token comes before or after another
  - Position is important in sequence modeling in NLP

- A way to introduce position information is add individual position encodings to the input for each position in the sequence

$$x_i = x_i + pos_i$$

Where pos_{t} is a position vector

Properties of a good positional embedding

- It should output a unique encoding for each time-step (word's position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
  - The cat sat on the mat
  - The happy cat sat on the mat
- Our model should generalize to longer sentences without any efforts. Its values should be bounded.
- It must be deterministic.

Absolute position embeddings

- Define a maximum context length you model can encode: say 1000 tokens.
  - Create a separate embedding table for each position.
  - Each index 1, 2, 3, … gets an embedding.
  - Learn the embeddings with the model.

- Issues with Learned positions embeddings:
  - Maximum length that can be presented is limited (what if I get a 2000 token input)
  - Difficult to encode relative positions
    - The cat sat on the mat
    - The happy cat sat on the mat

Functional (and fixed) position embeddings

Sinusoidal embeddings

$$\vec{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

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

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

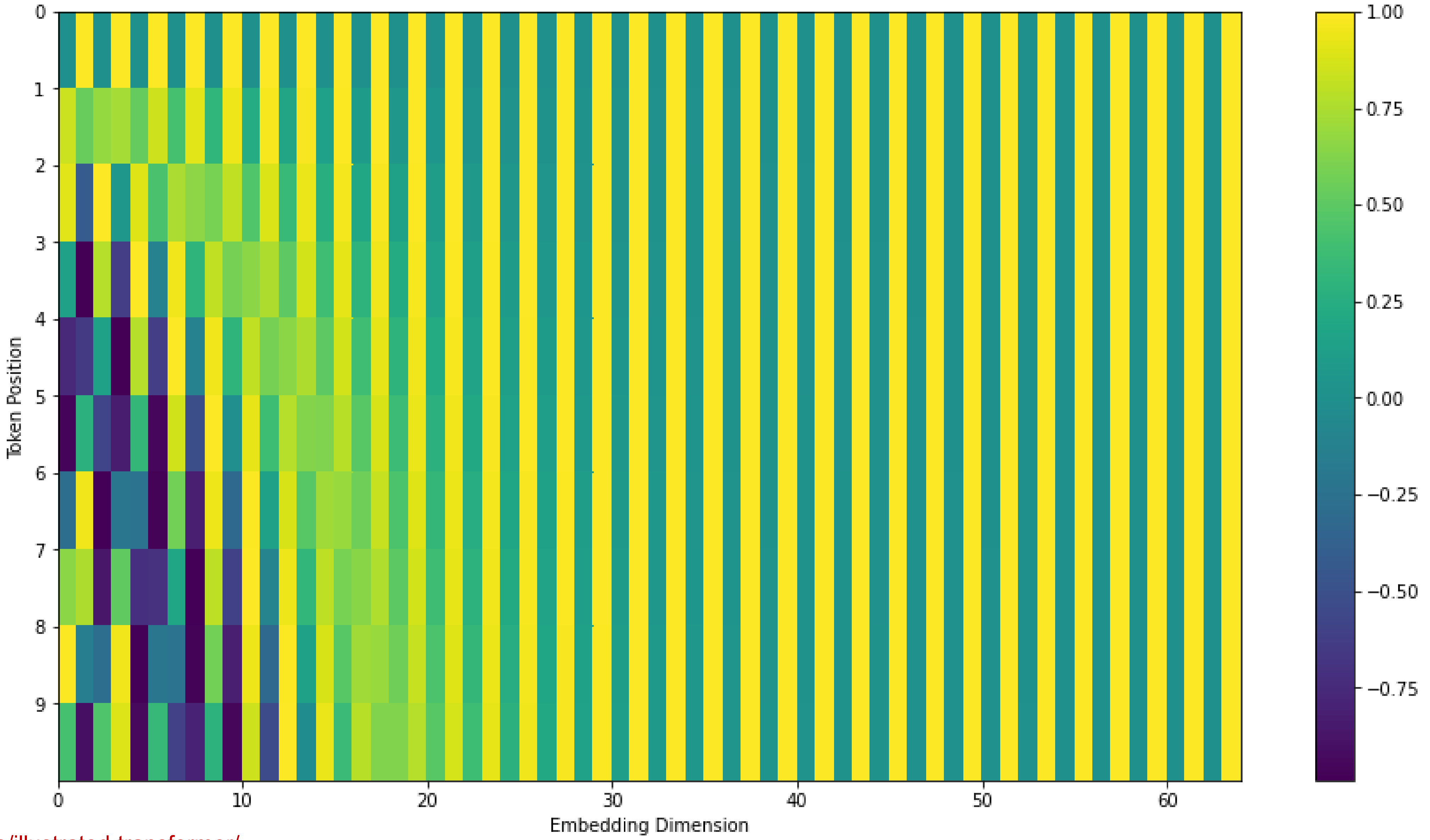
The frequencies are decreasing along the vector dimension. It forms a geometric progression on the wavelengths.

Sinusoidal Embeddings: Intuition

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

[Transformer Architecture: The Positional Encoding - Amirhossein Kazemnejad's Blog]

Position Encodings



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

Variants of Positional Embeddings

- Rotary Positional Embeddings (RoPE): [2104.09864] RoFormer: Enhanced Transformer with Rotary Position Embedding (arxiv.org)

- AliBi: [2108.12409] Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (arxiv.org)

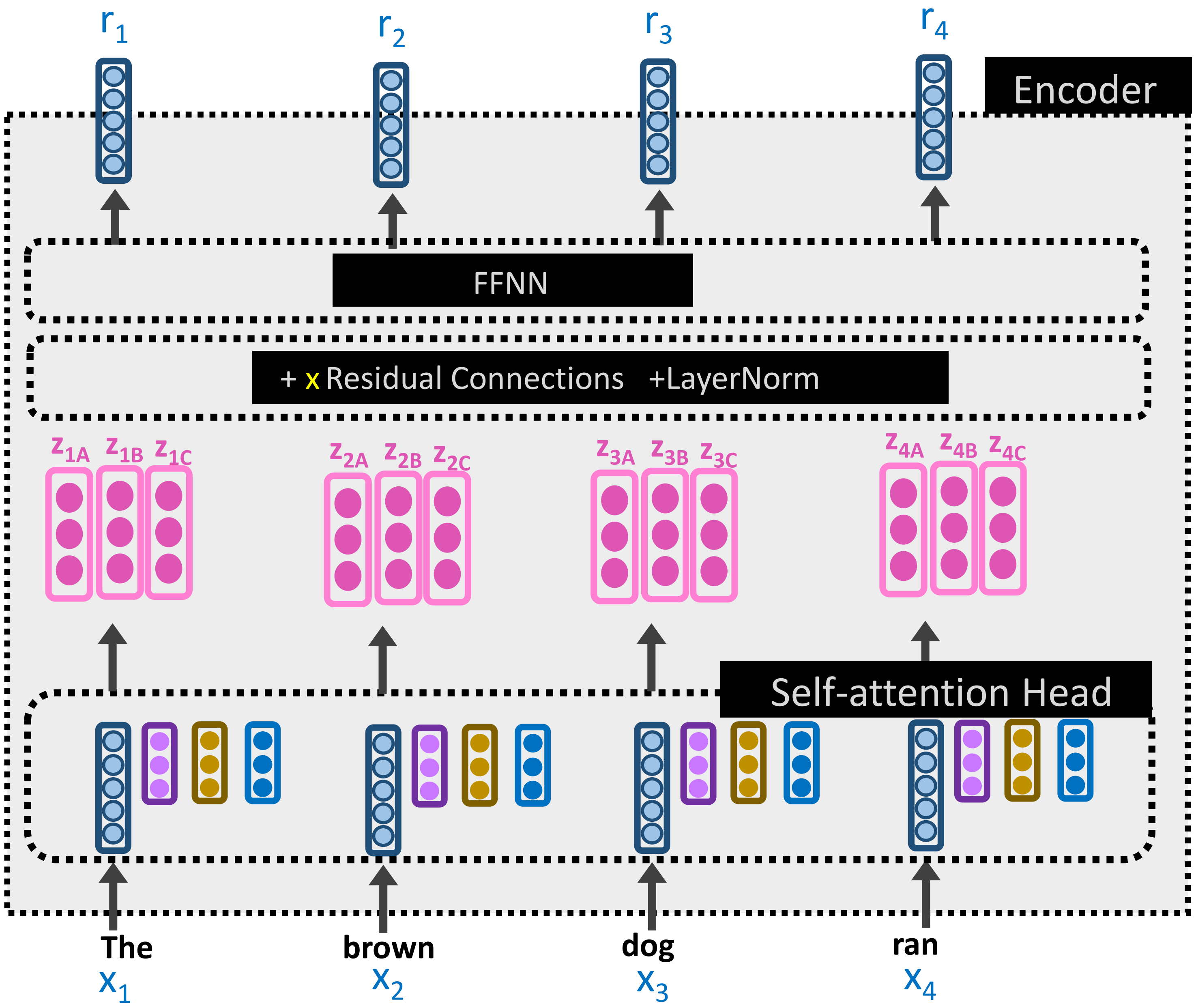
- No embeddings(!?): [2203.16634] Transformer Language Models without Positional Encodings Still Learn Positional Information (arxiv.org)

A Self-Attention Head has just one set of query/key/value weight matrices $\mathbf{w}_{q}$, $\mathbf{w}_{k}$, $\mathbf{w}_{v}$

Words can relate in many ways, so it's restrictive to rely on just one Self-Attention Head in the system.

Let's create Multi-headed Self-Attention

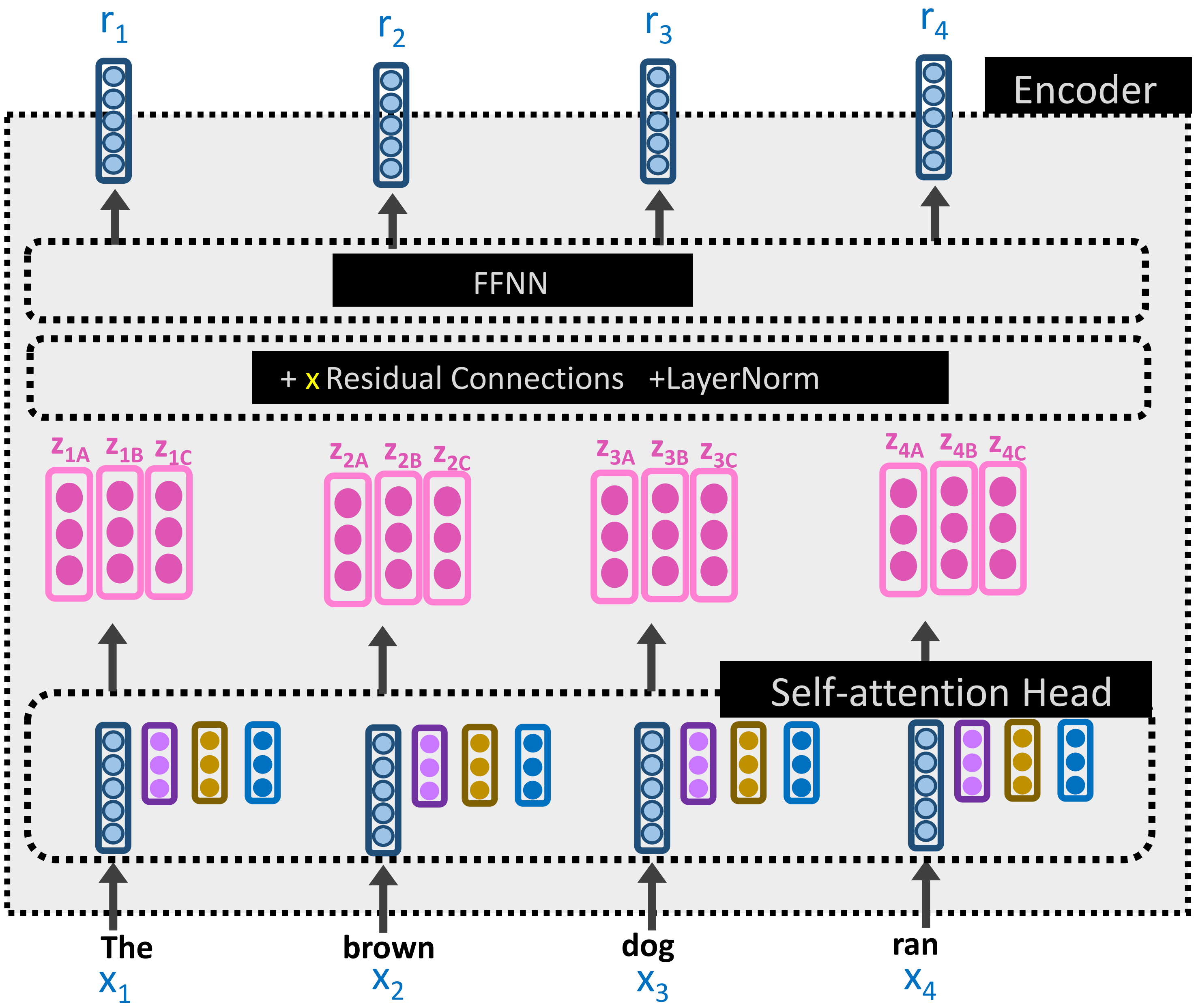
Multi-head Attention



Each Self-Attention Head produces a z_i vector using query, key, and value vectors

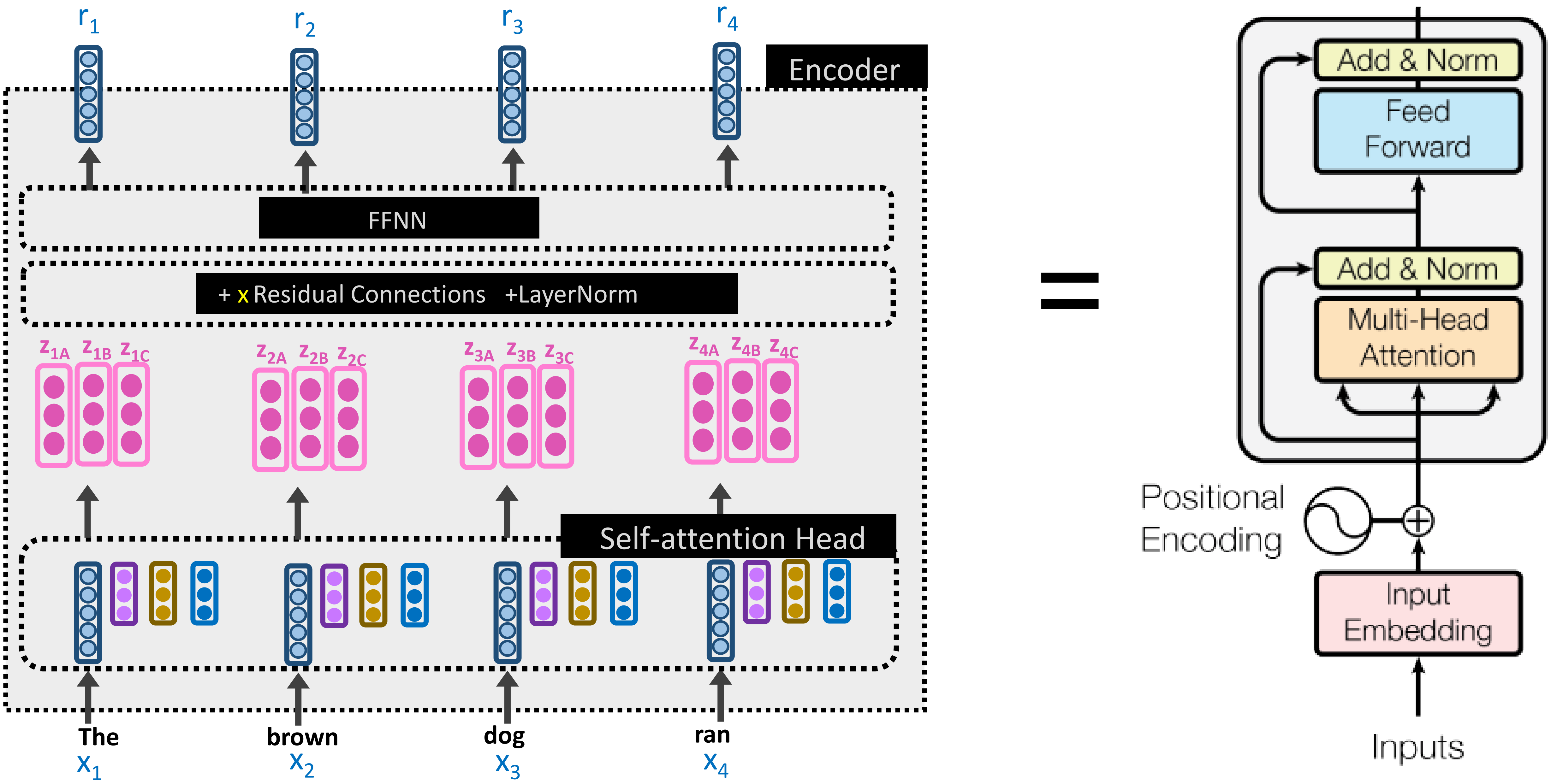
We can, in parallel, use multiple heads and concat the z_i's. For each input create multiple query, key, and value vectors

Transformer Encoder

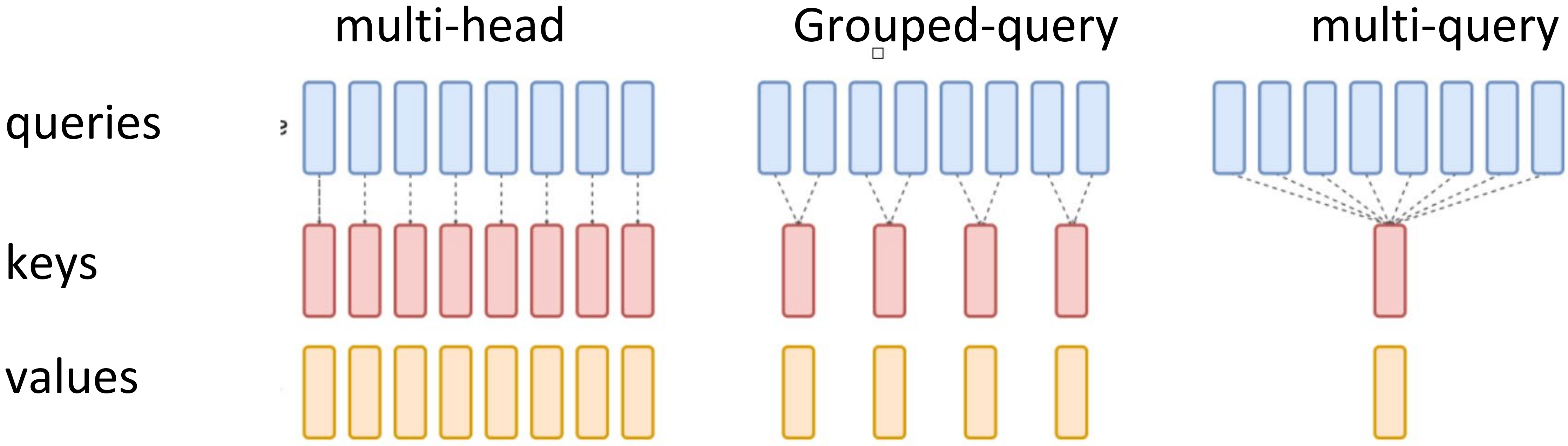


To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Transformer Encoder



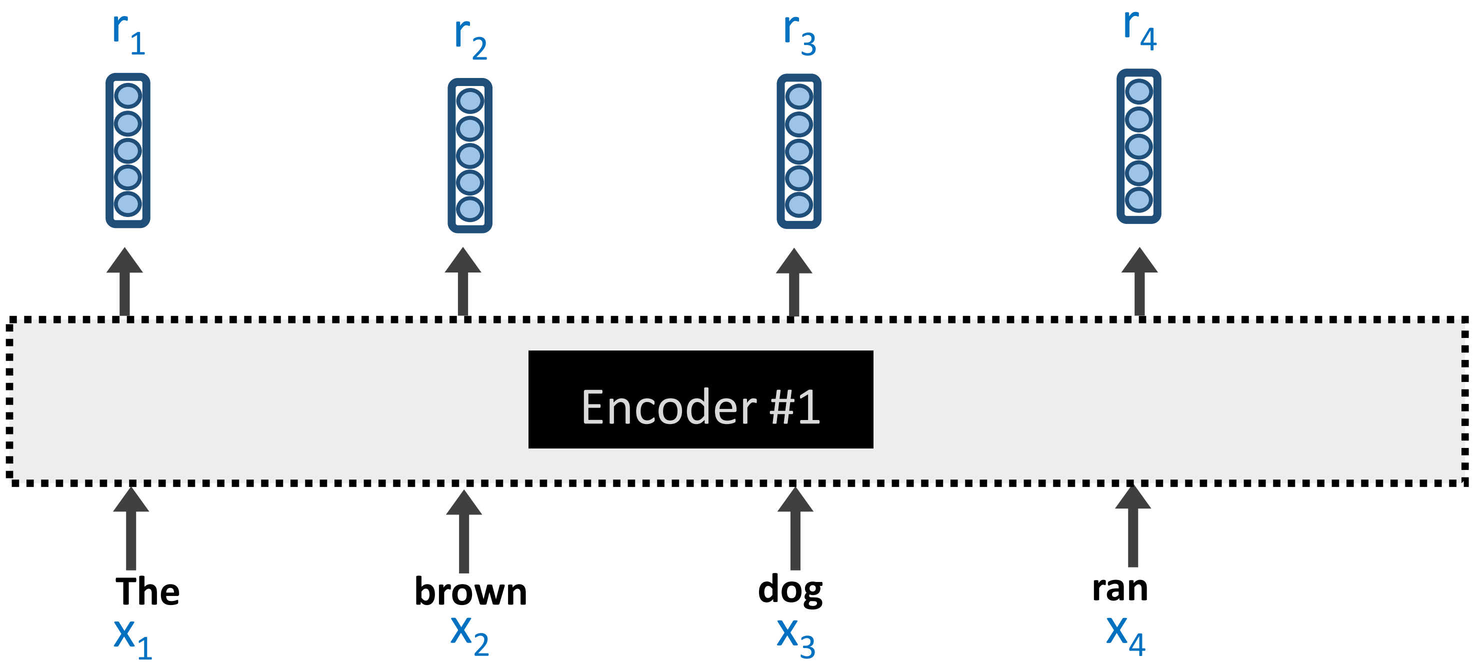
Variants of multi-head attention attention



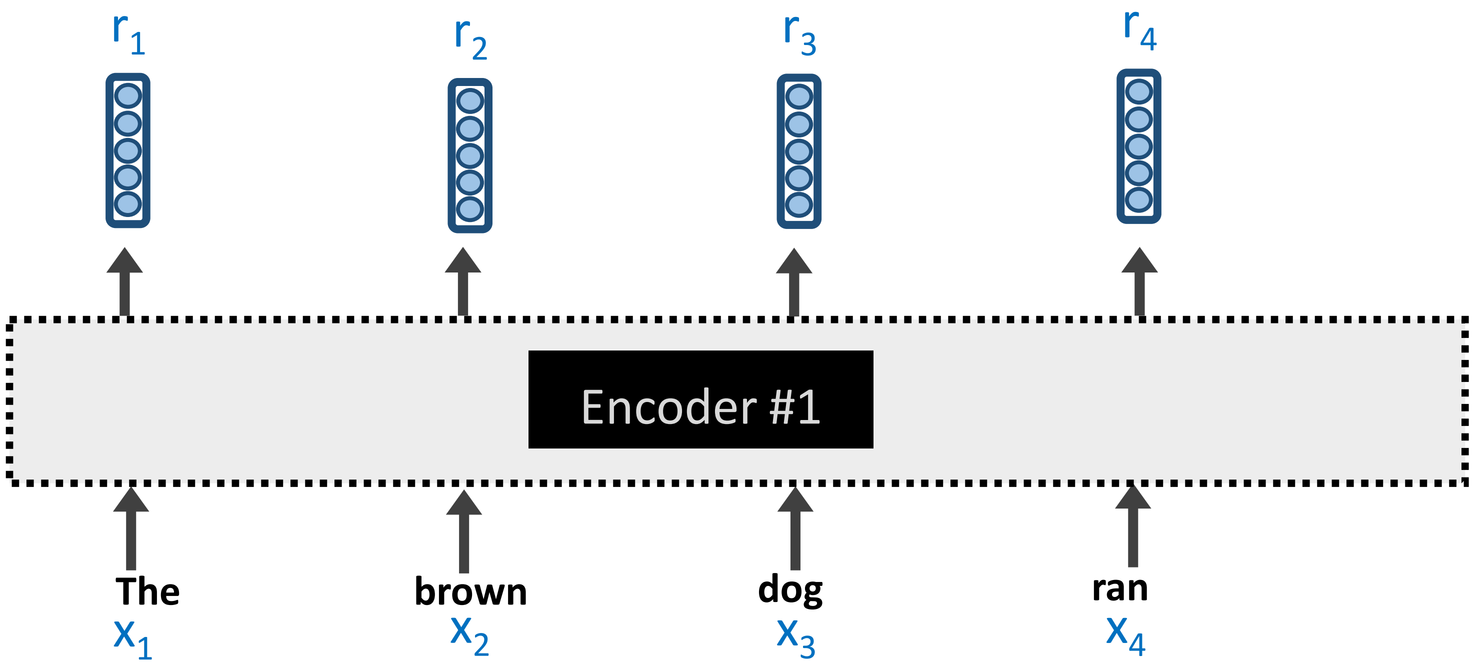
GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints (Ainslie et al., 2023)

Transformer Encoder

To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i



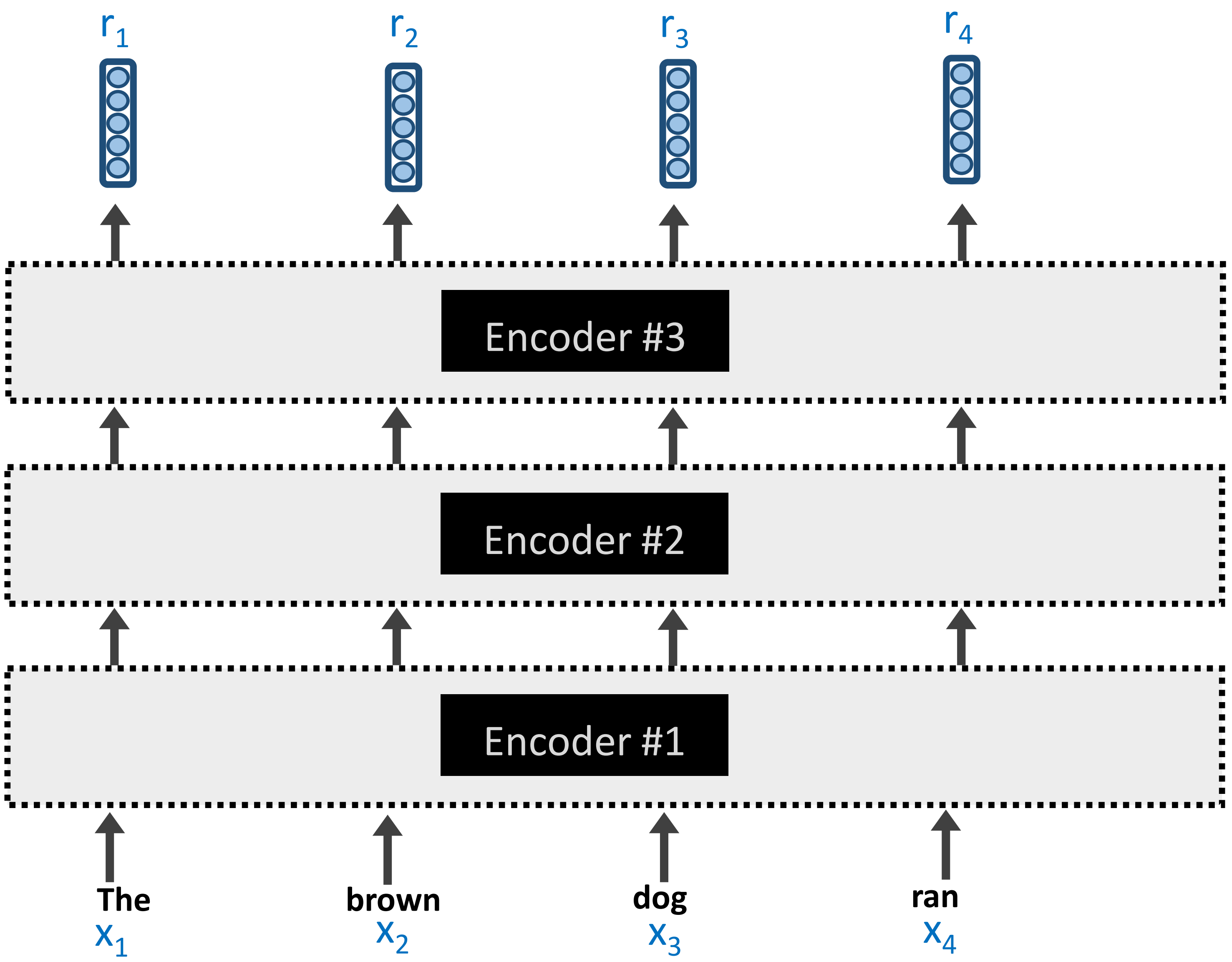
Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

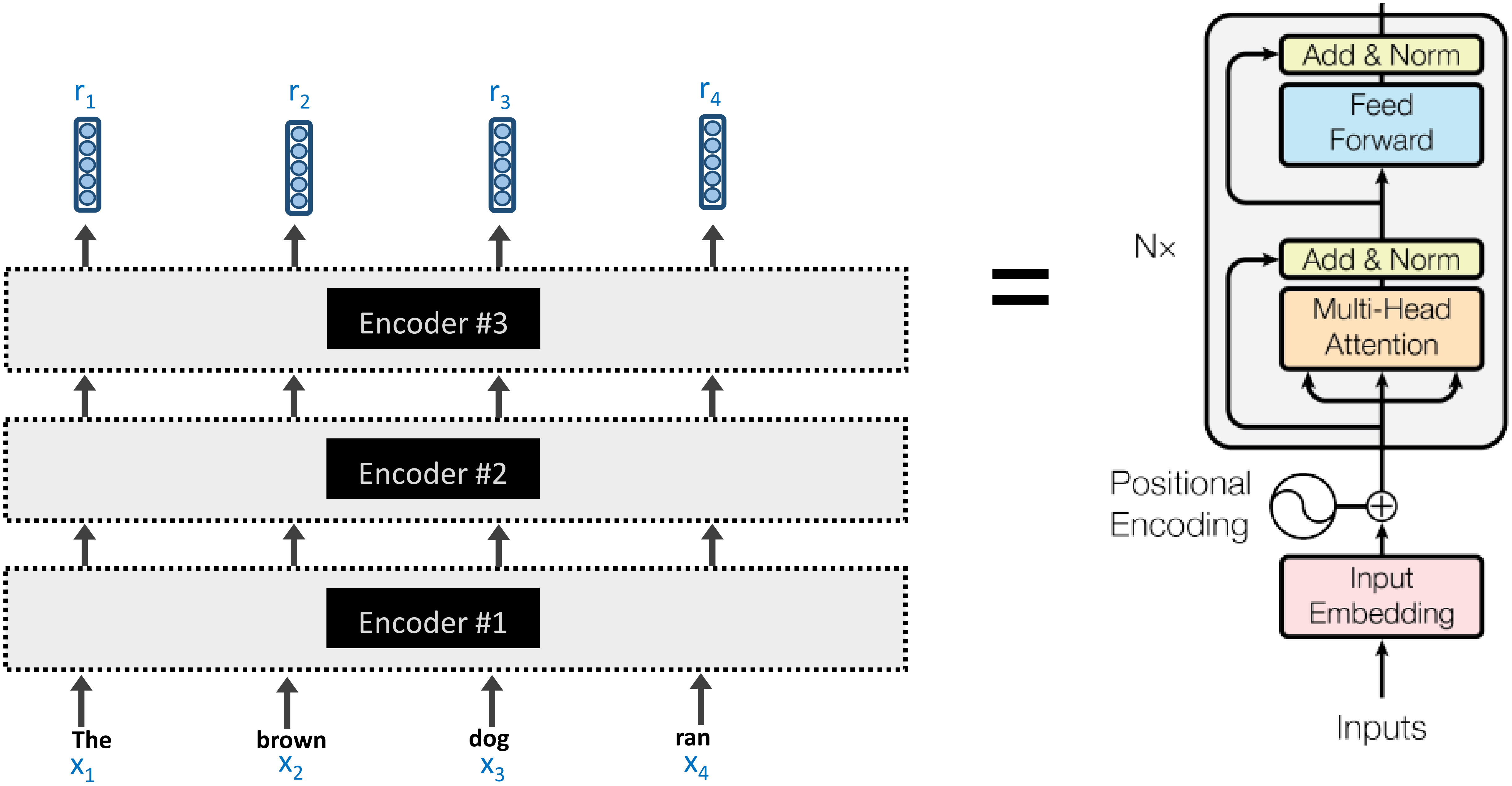
Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

Transformer Encoder



The original Transformer model was intended for Machine Translation, so it had Decoders, too

Outline

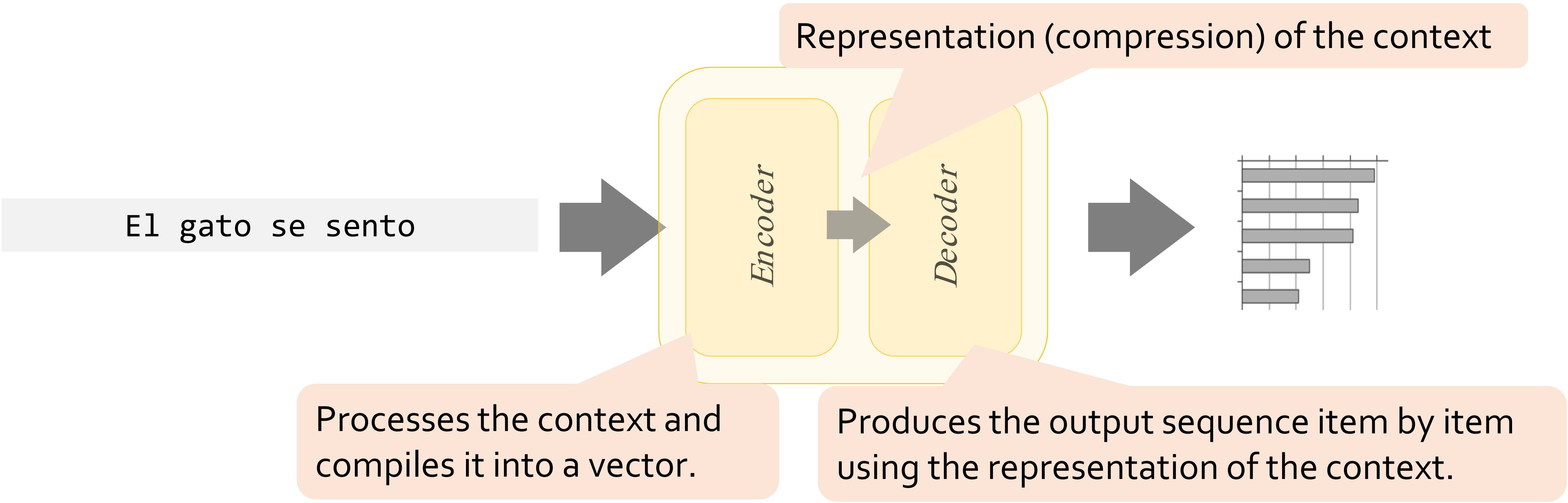
- Self-Attention
- Transformer Encoder
- Transformer Decoder
- Language Modeling With Transformers

Outline

- Self-Attention
- Transformer Encoder
- Transformer Decoder
- Language Modeling With Transformers

Encoder-Decoder Architectures

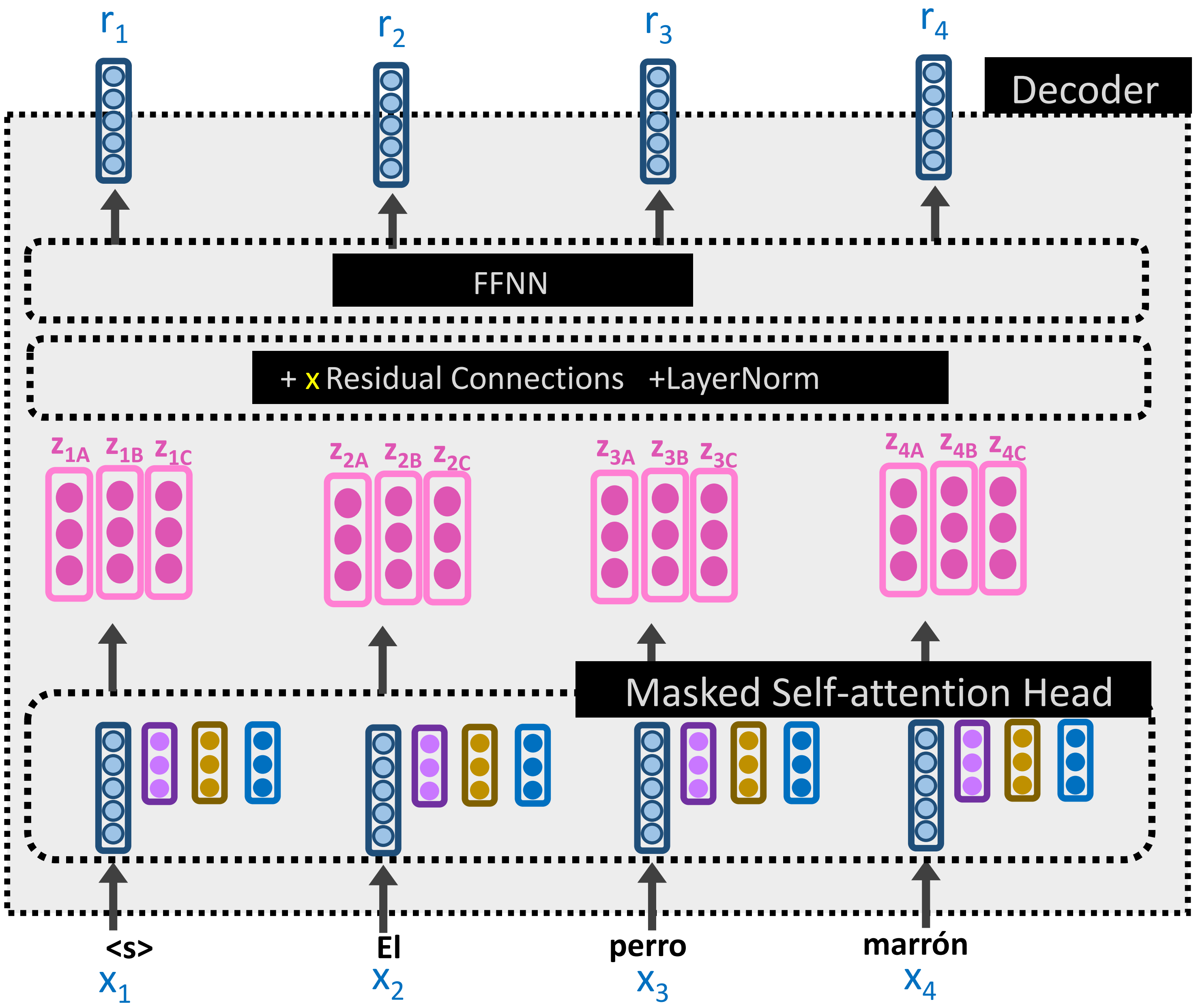
- Original transformer had two sub-models.



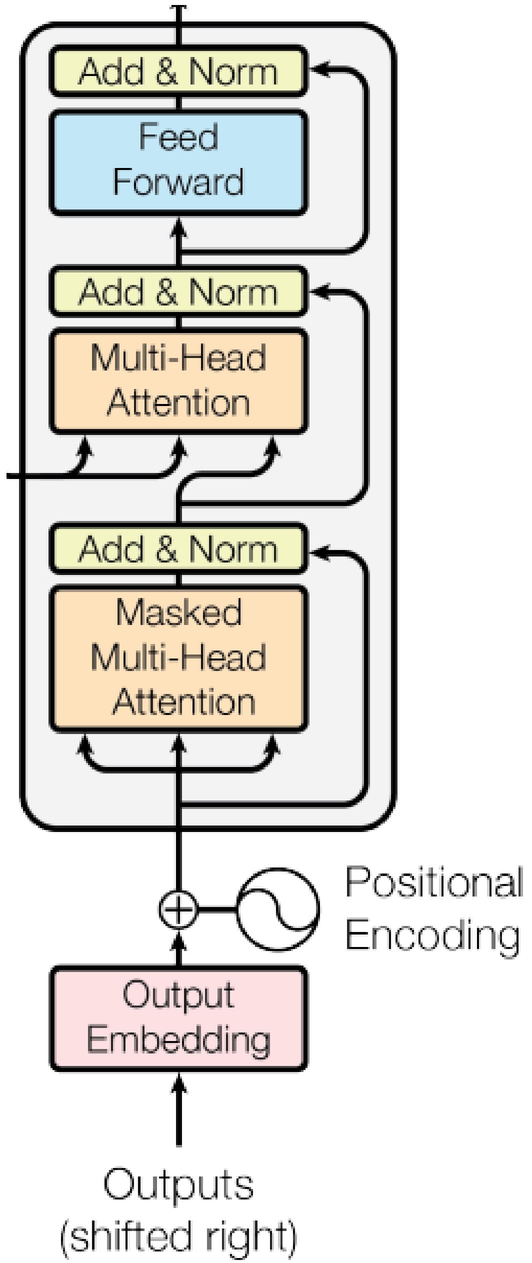
Encoder-Decoder Architectures

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

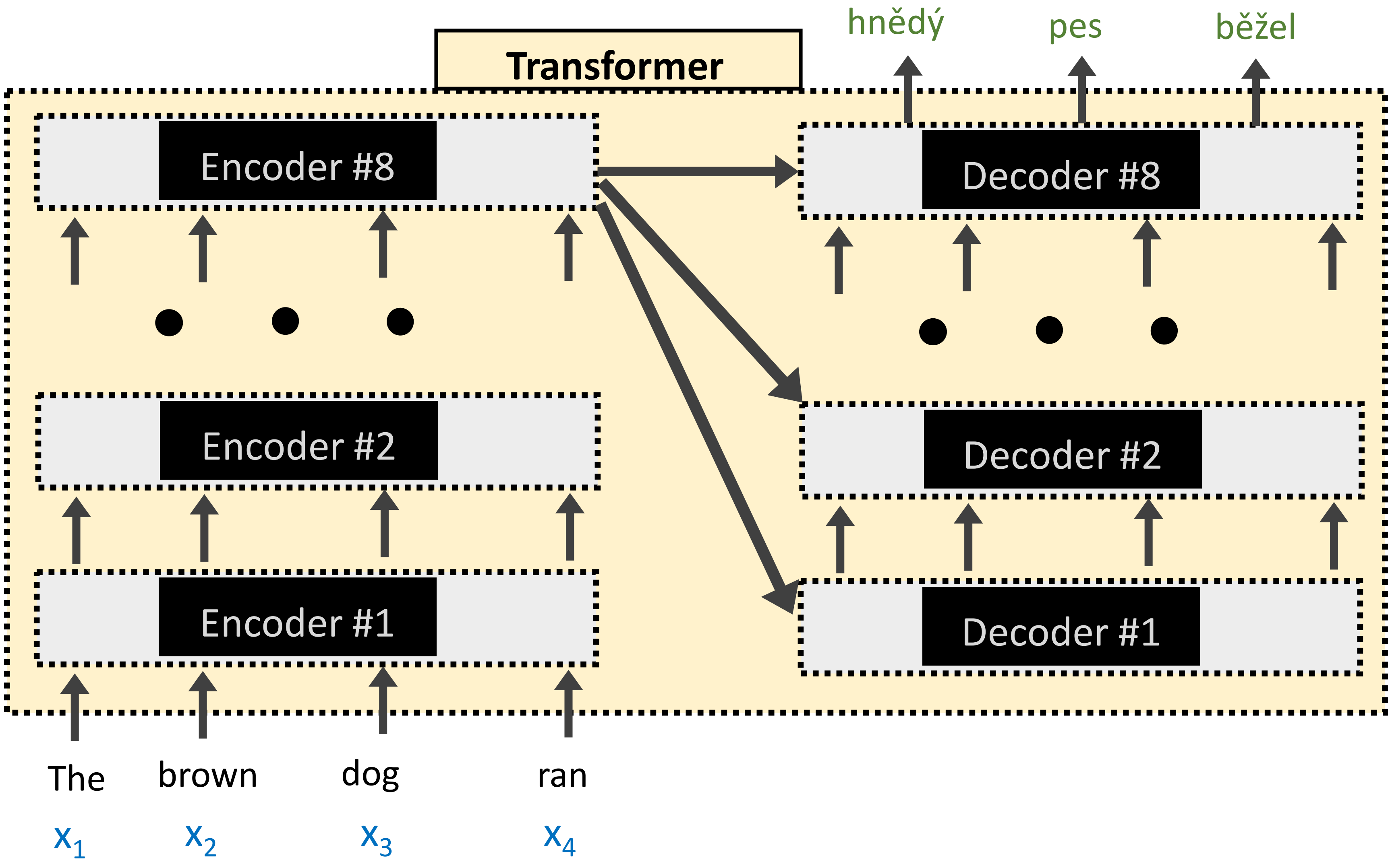
Transformer Decoder



=



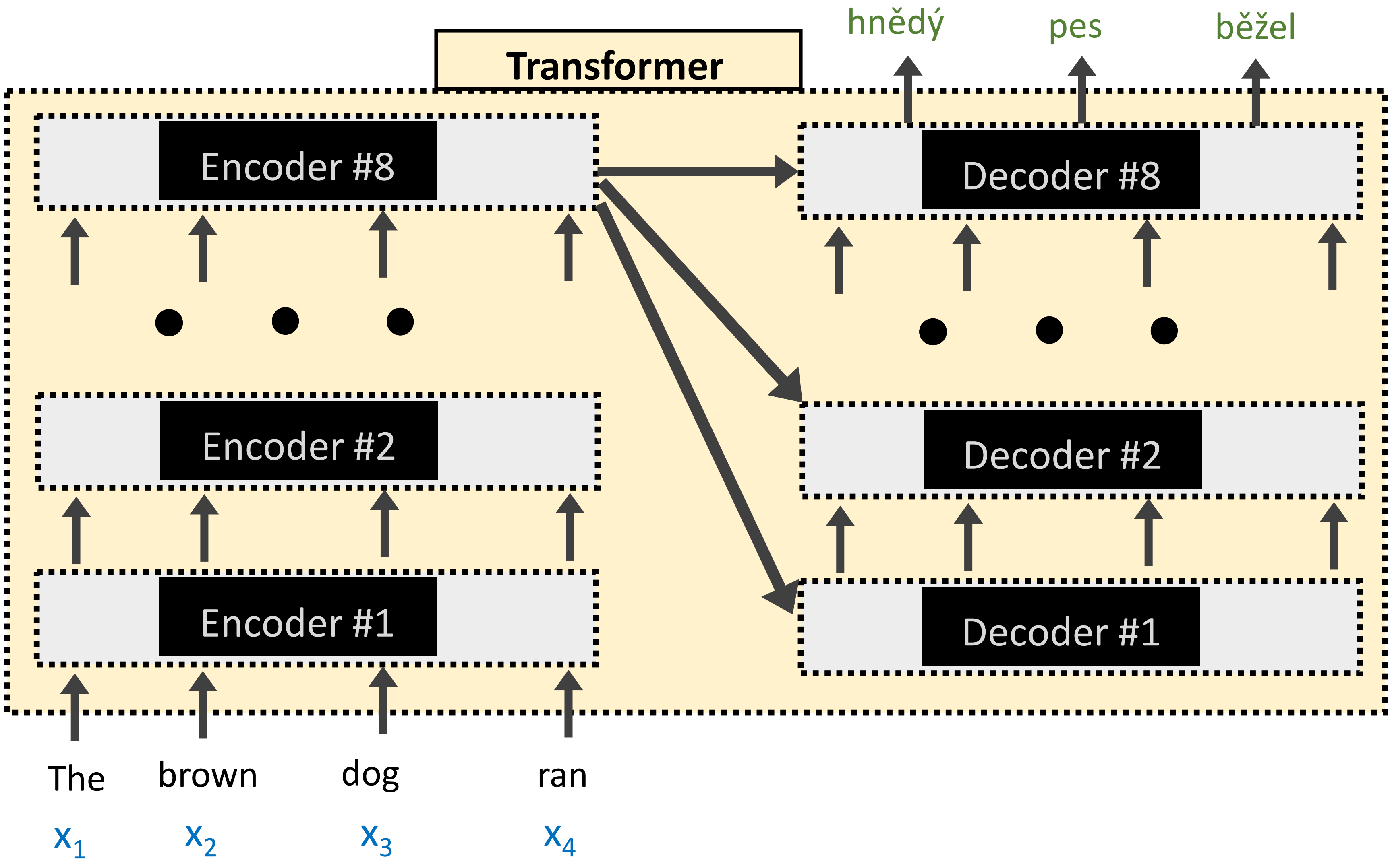
Transformer Encoders and Decoders



Transformer Encoders produce **contextualized embeddings** of each word

Transformer Decoders generate new sequences of text

Transformer Encoders and Decoders

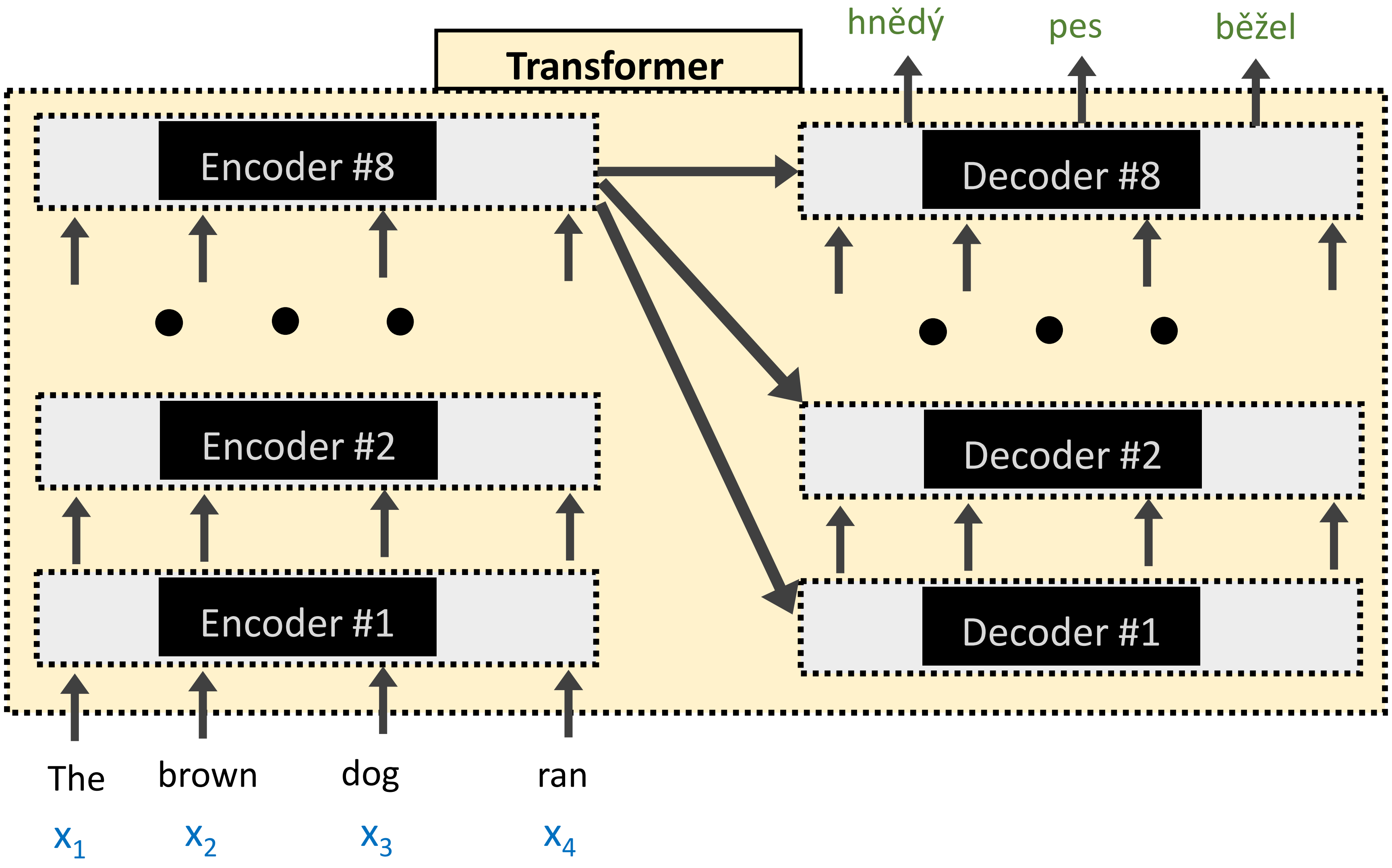


NOTE

Transformer Decoders are identical to the Encoders, except they have an additional Attention Head in between the Self-Attention and FFNN layers.

This additional Attention Head **focuses on parts of the encoder's representations.**

Transformer Encoders and Decoders

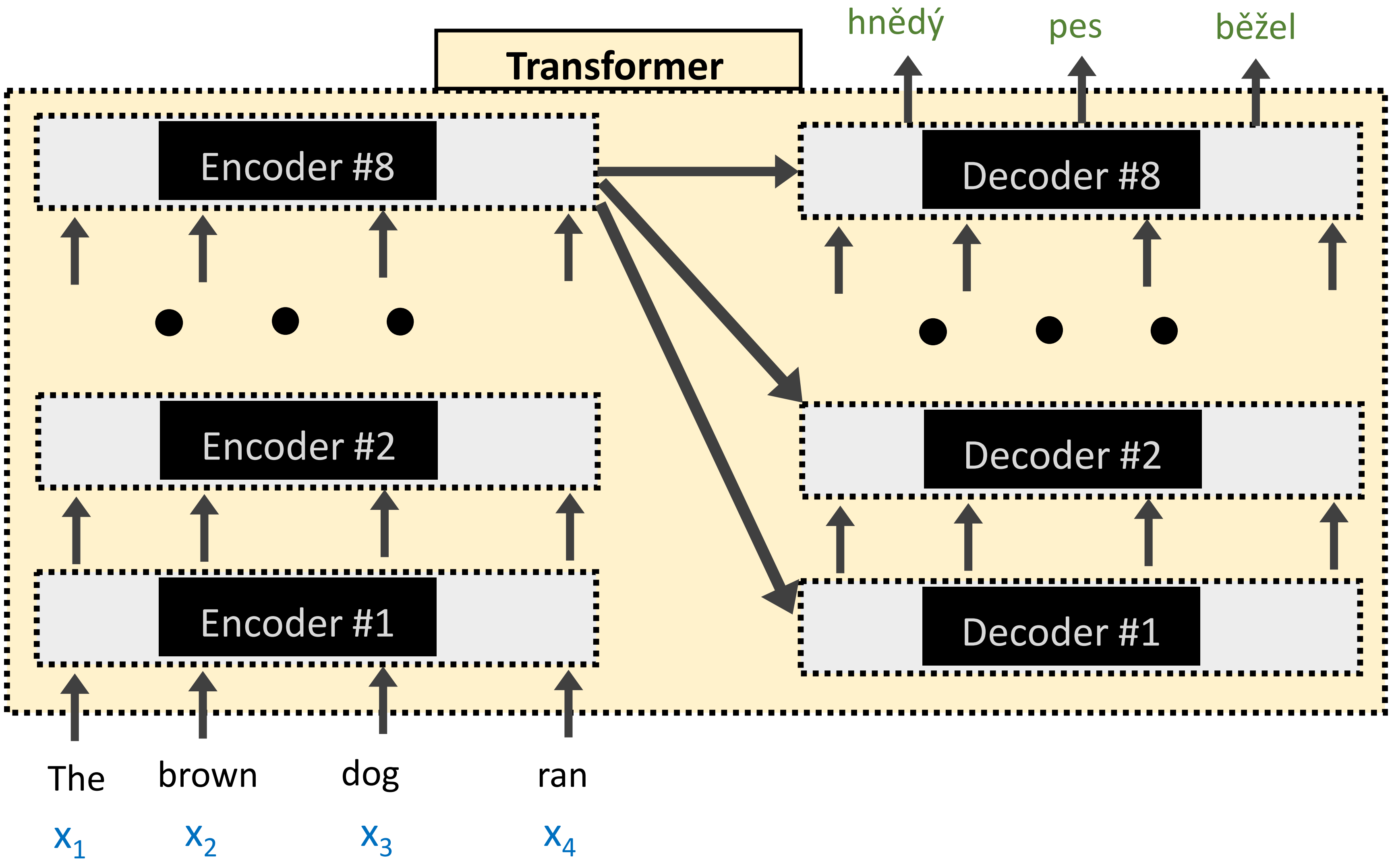


NOTE

The **query** vector for a Transformer Decoder's **Attention Head** (not Self-Attention Head) is from the output of the previous decoder layer.

However, the **key** and **value** vectors are from the Transformer Encoders' outputs.

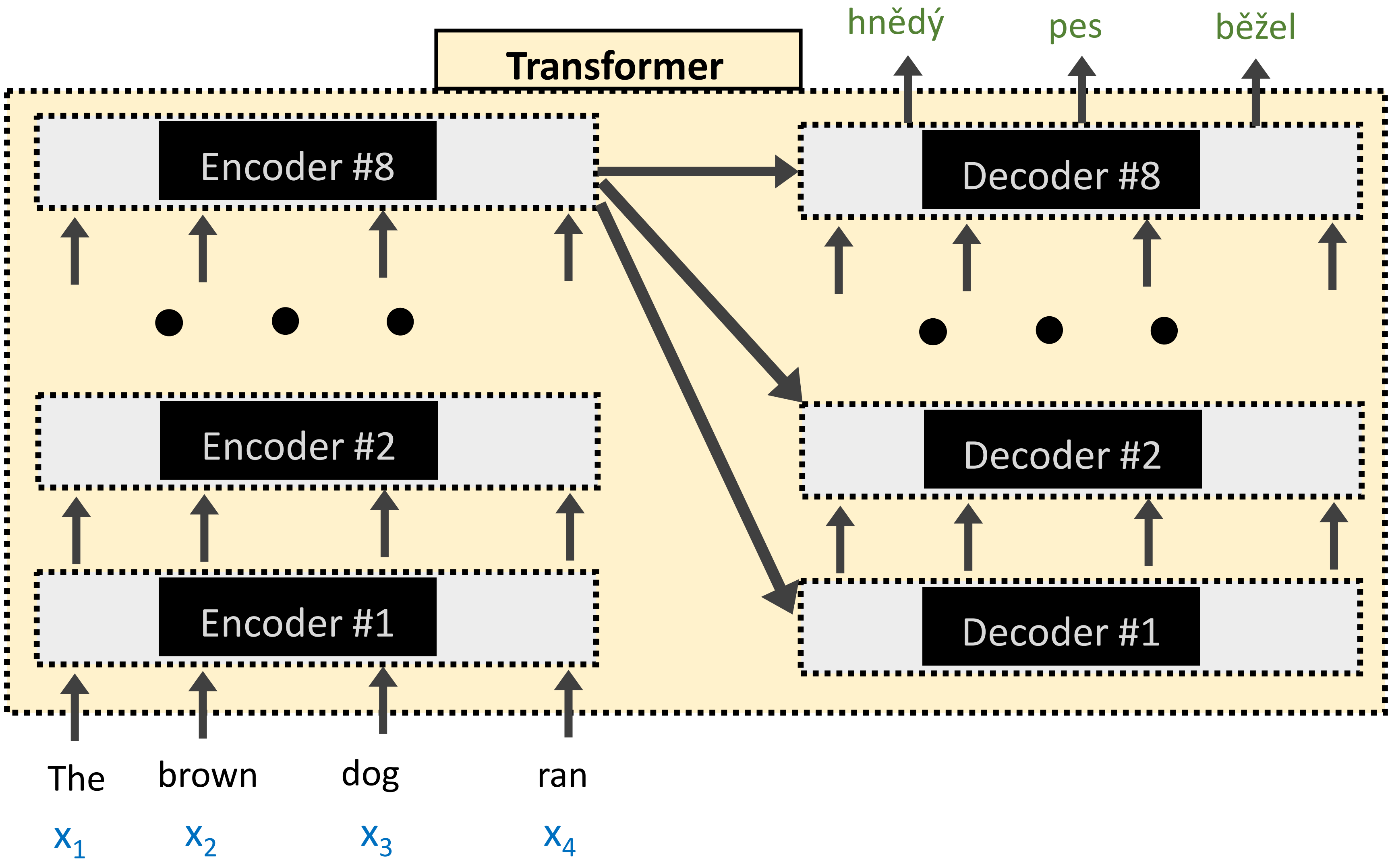
Transformer Encoders and Decoders



NOTE

The **query**, **key**, and **value** vectors for a Transformer Decoder's **Self-Attention Head** (not Attention Head) are all from the output of the previous decoder layer.

Transformer Encoders and Decoders



IMPORTANT

The Transformer Decoders have positional embeddings, too, just like the Encoders.

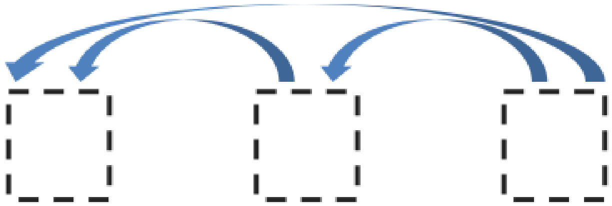
Critically, each position is only allowed to attend to the previous indices. This *masked* Attention preserves it as being an auto-regressive LM.

Transformer [Vaswani et al. 2017]

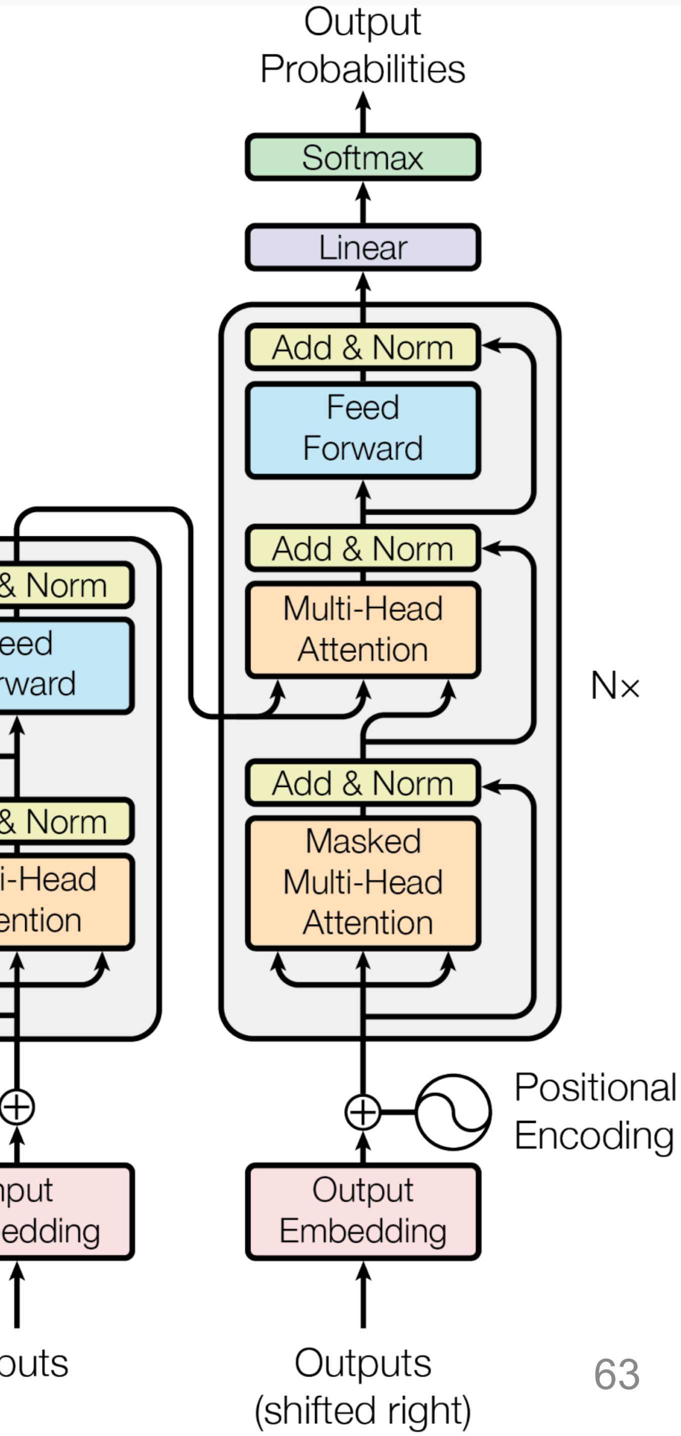
- An encoder-decoder architecture built with attention modules.
- 3 forms of attention

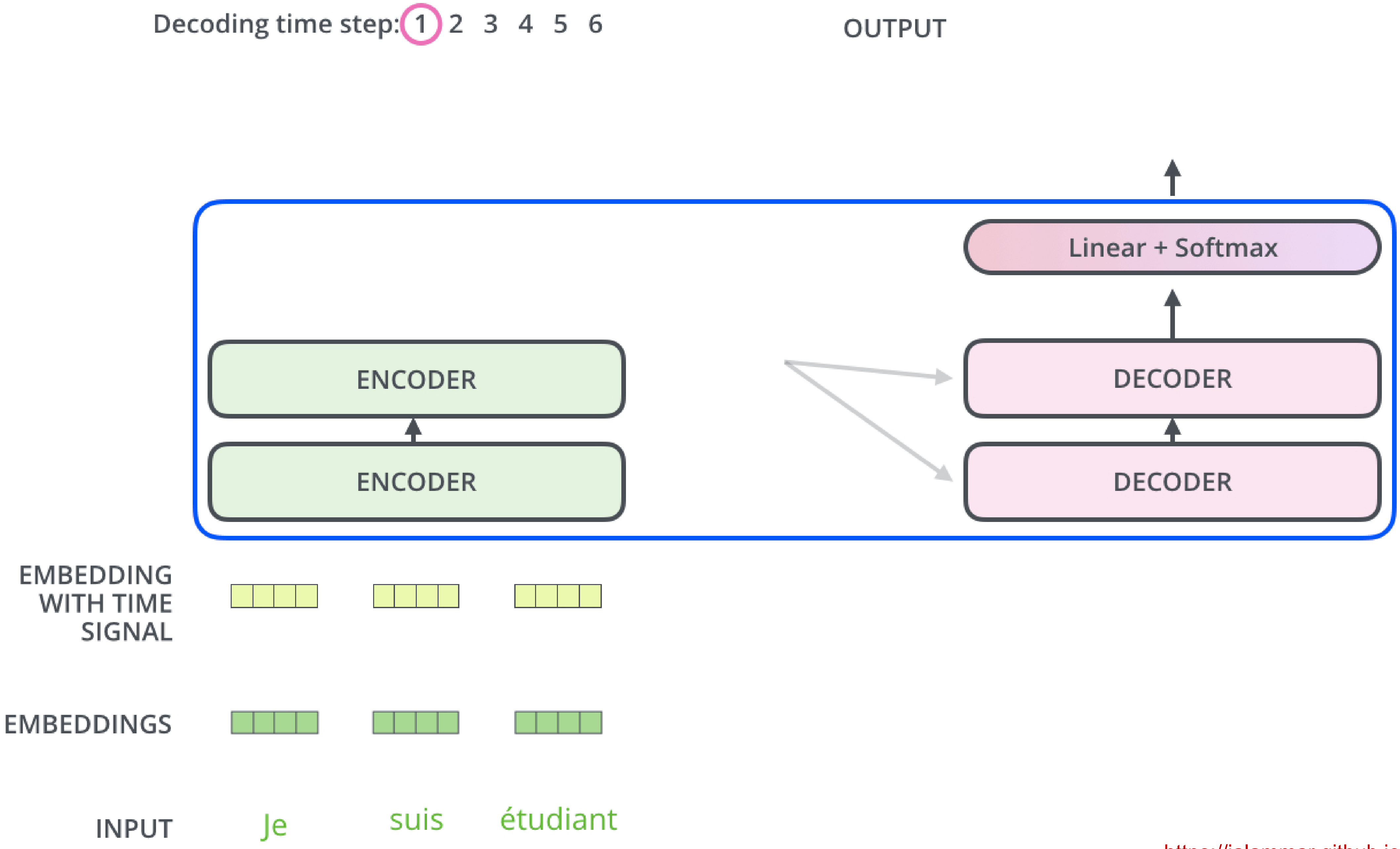
Encoder-Decoder Attention

Encoder Self-Attention

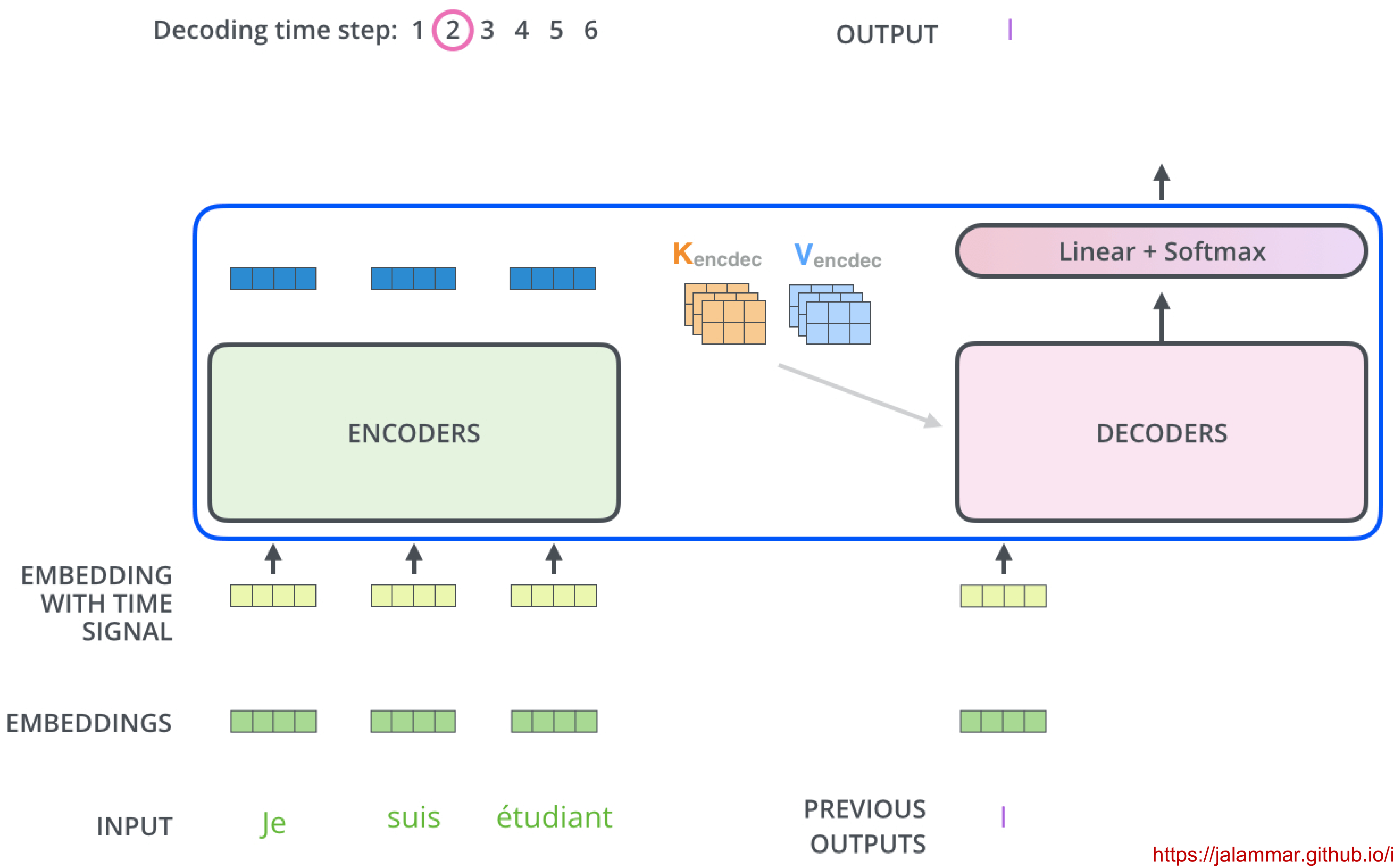


MaskedDecoder Self-Attention





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



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

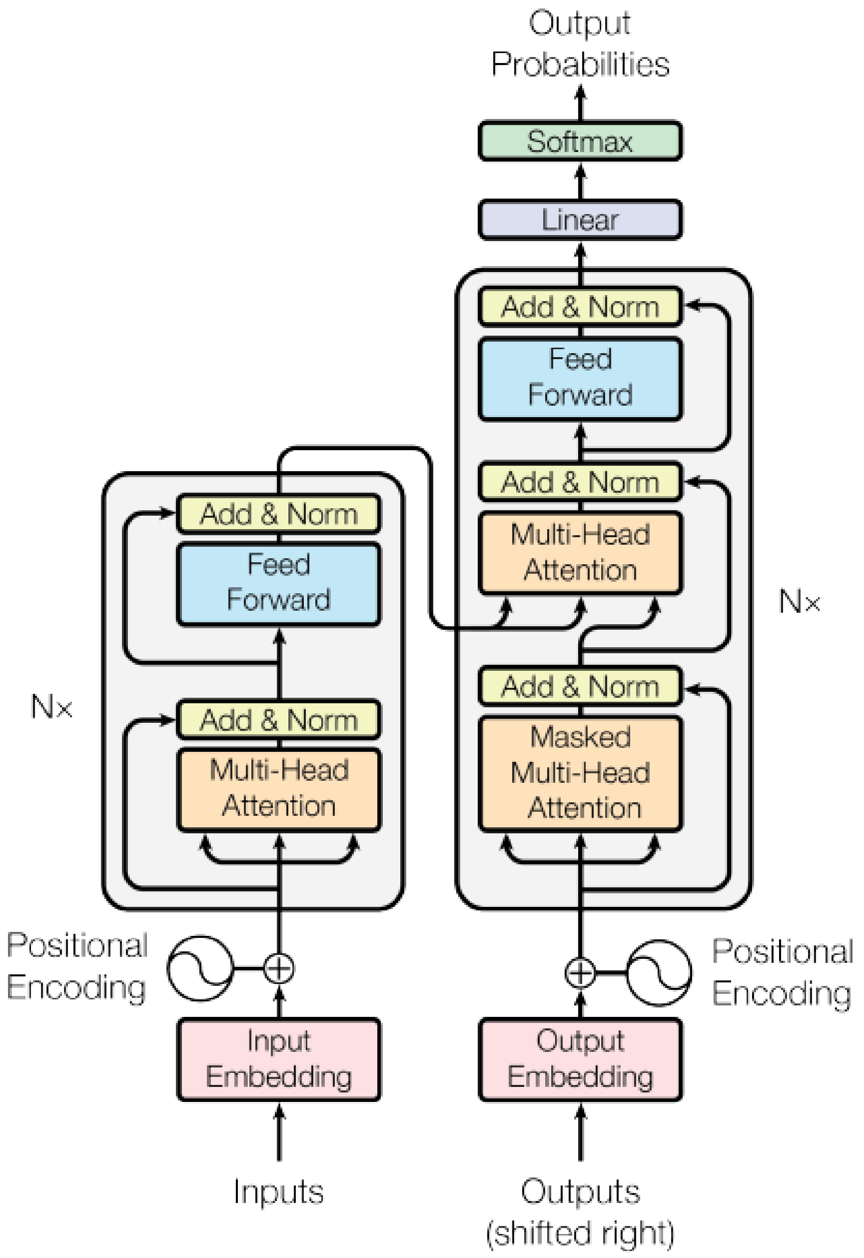


Figure 1: The Transformer - model architecture.

Attention is All you Need (2017) https://arxiv.org/pdf/1706.03762.pdf

Impact of Transformers

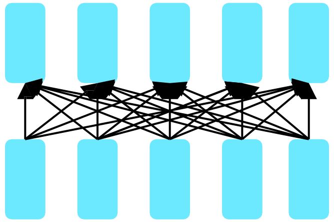
- Let to better predictive models of language ala GPTs!

Model	Layers	Heads	Perplexity
LSTMs (Grave et al., 2016)	-	-	40.8
QRNNs (Merity et al., 2018)	-	-	33.0
Transformer	16	16	19.8

["Efficient Content-Based Sparse Attention with Routing Transformers" Roy et al. 2020]

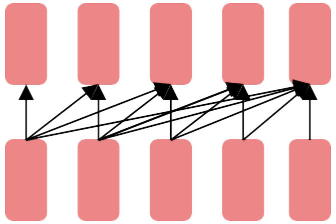
Impact of Transformers

- A building block for a variety of LMs



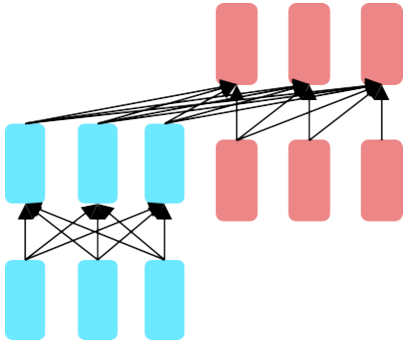
Encoders

- ❖ **Examples:** BERT, RoBERTa, SciBERT.
- ❖ Captures bidirectional context. How do we pretrain them?



Decoders

- ❖ **Examples:** GPT-2, GPT-3, Llama models, and many many more
- ❖ Other name: **causal or auto-regressive language model**
- ❖ Nice to generate from; can't condition on future words



Encoder-Decoders

- ❖ **Examples:** Transformer, T5, BART
- ❖ What's the best way to pretrain them?

Transformer LMs + Scale = LLMs

- 2 main dimensions:
- Model size, pretraining data size

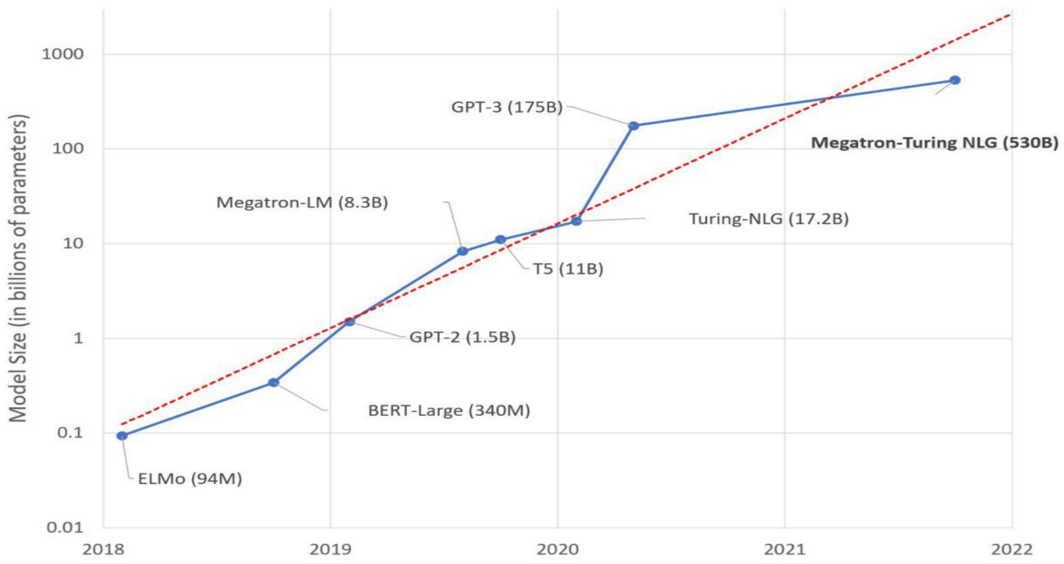
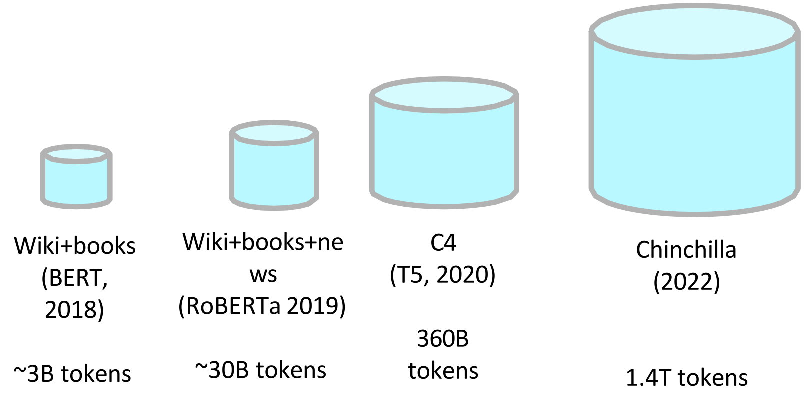
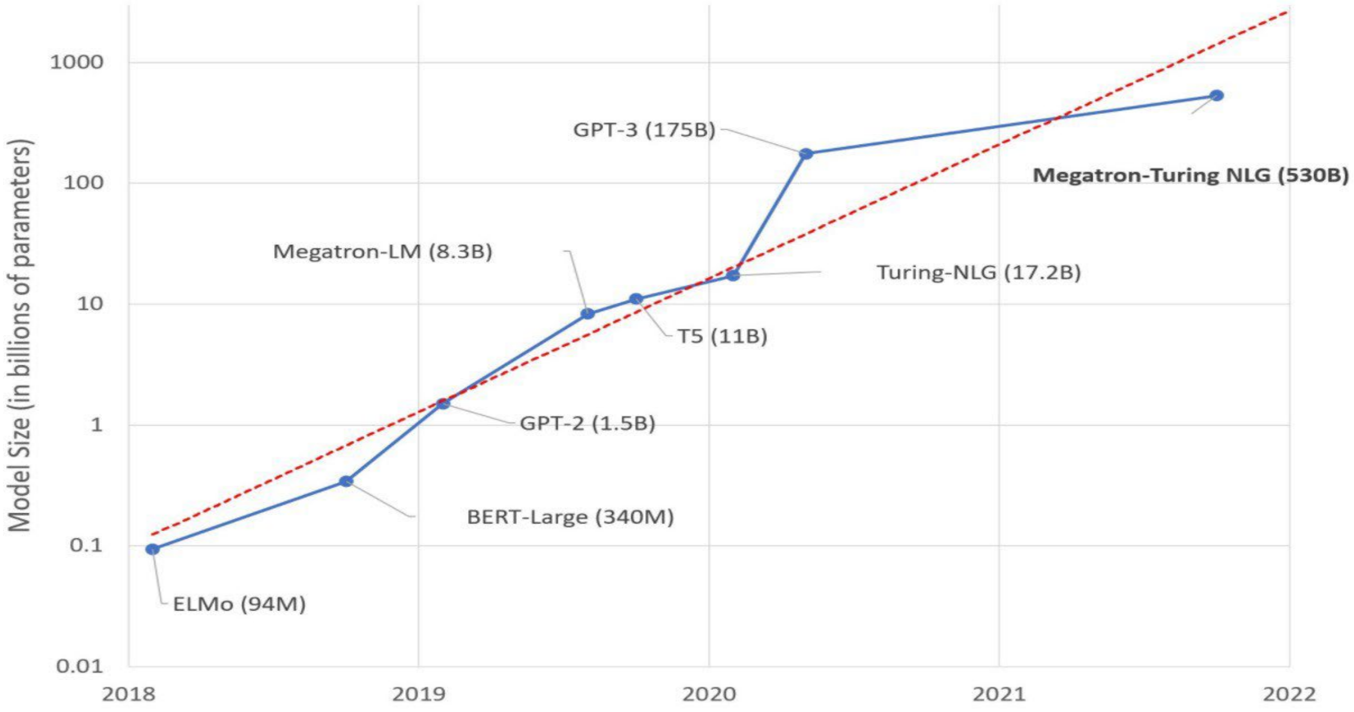


Photo credit: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/



Large Language Models

- Not only they improved performance on many NLP tasks, but exhibited new capabilities



Transformers - Summary

- Self-attention + positional embedding + others = NLP go brr
- Much faster to train than any previous architectures, much easier to scale
- Perform on par or better than previous RNN based models
  - Ease of scaling allows to extract much better performance