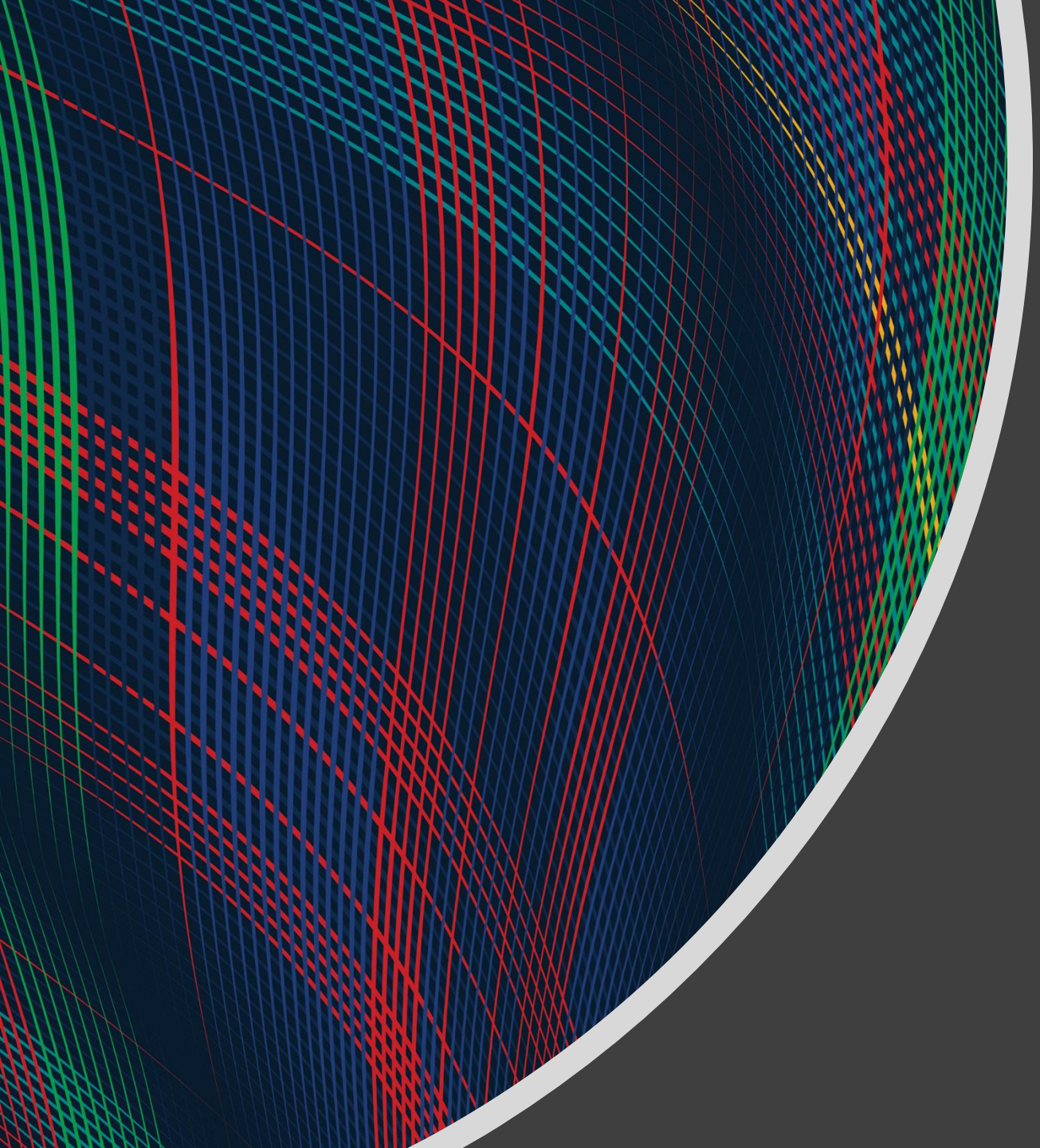


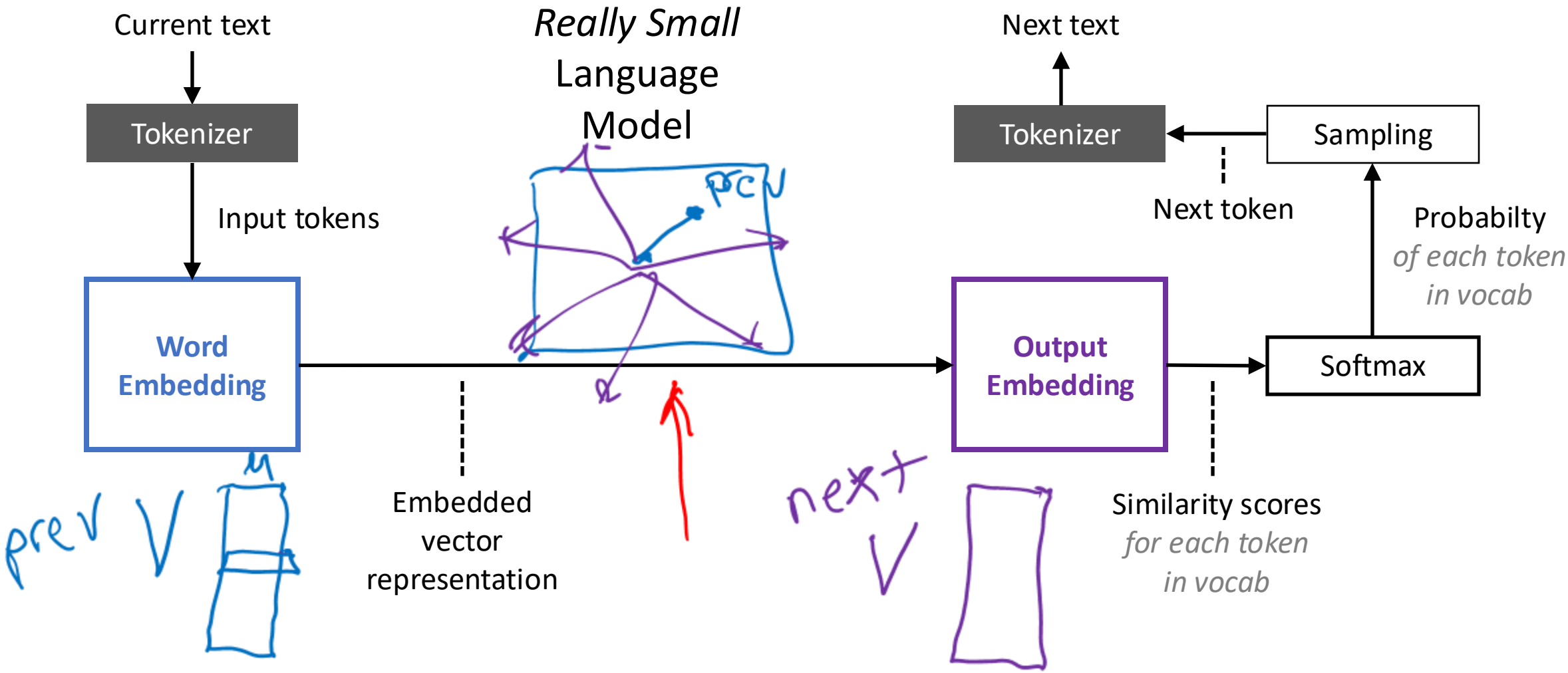
# 10-315
# Introduction to ML

# LLMs:
# Attention & Transformers

Instructor: Pat Virtue

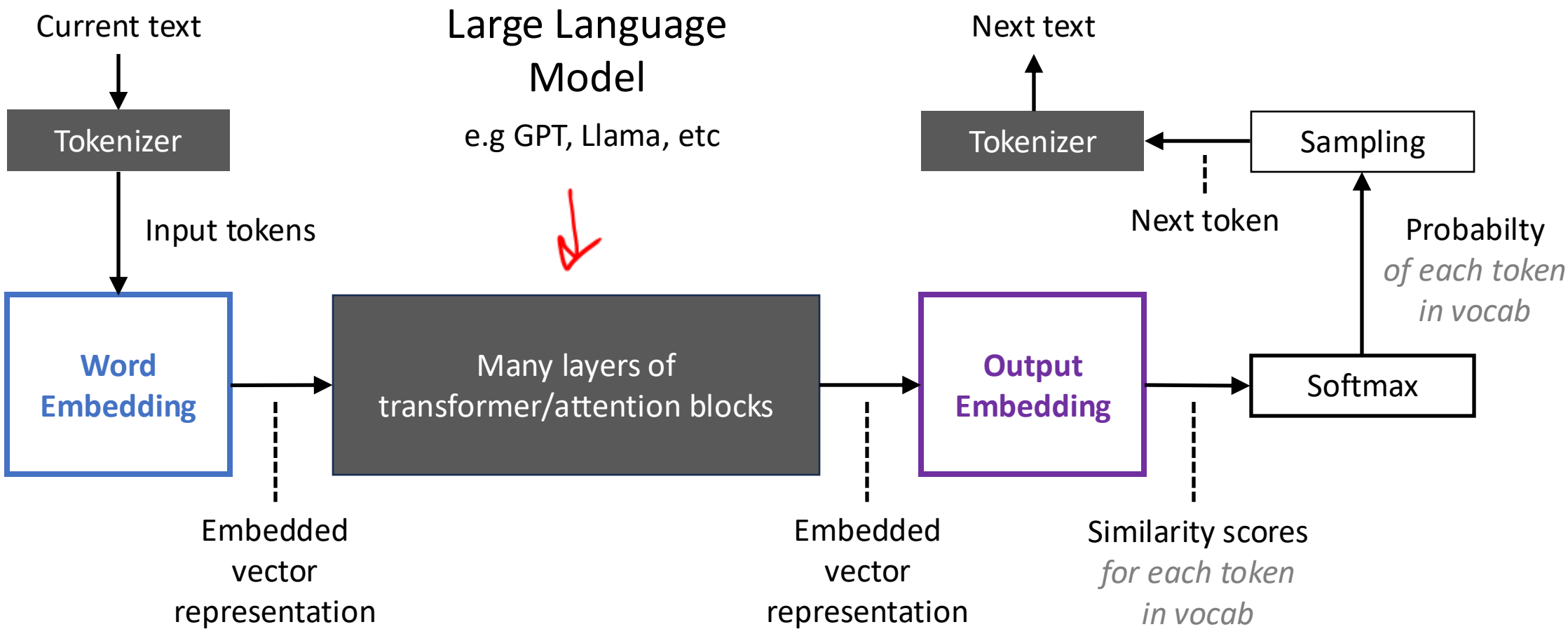
# Simple Word Embedding LM

## Building a language model with just word embedding layers ☺



# Word (Token) Embeddings

## The beginning and the end of LLM networks



# Building up to Large Language Models

## N-gram LMs

## Word Embedding LMs

- Vector representation of vocab tokens
- Sampling next token
- Learning better vectors

## Transformer LMs

- Increasing context size
- Attention
- Tranformer blocks

## More Transformers

# Transformer LMs

# Transformer Language Models

## Increasing context size

- Uniform average of context vectors
- Position encoding

## Attention

- Weighted average of context vectors
- Query Keys Values
- Expressive power of linear transforms

## Transformer blocks

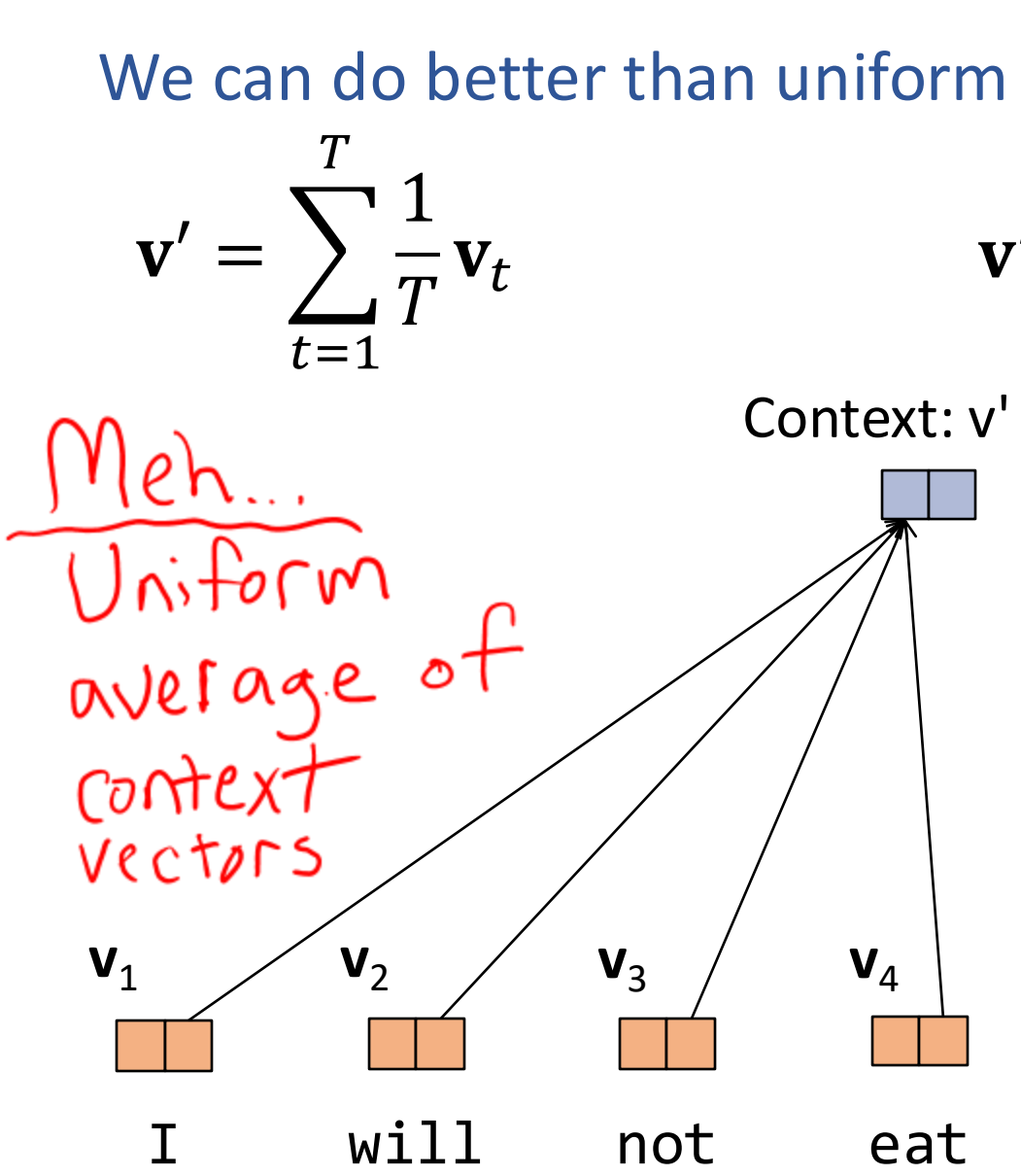
# Attention

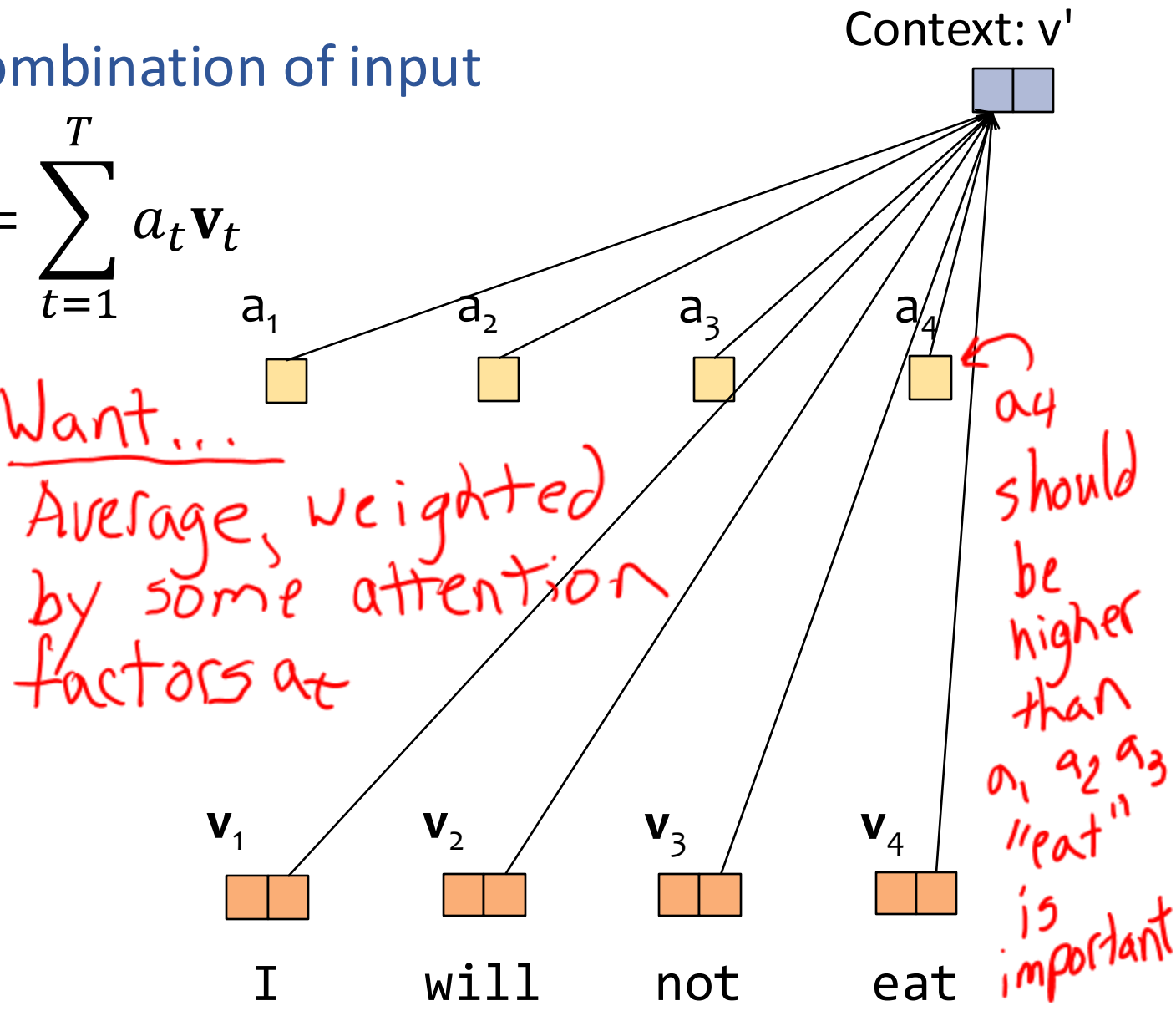
# Learn to pay attention!

## We can do better than uniform combination of input

$$\mathbf{v}' = \sum_{t=1}^{T} \frac{1}{T} \mathbf{v}_t$$

$$\mathbf{v}' = \sum_{t=1}^{T} a_t \mathbf{v}_t$$



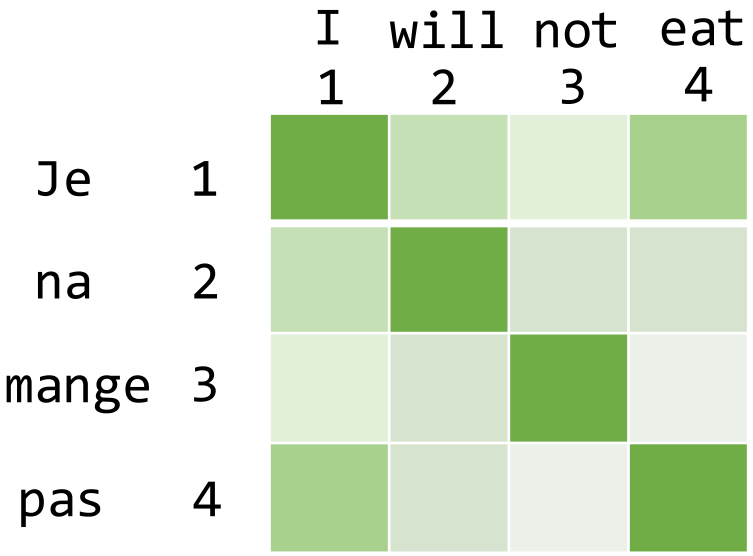
# Learn to pay attention!

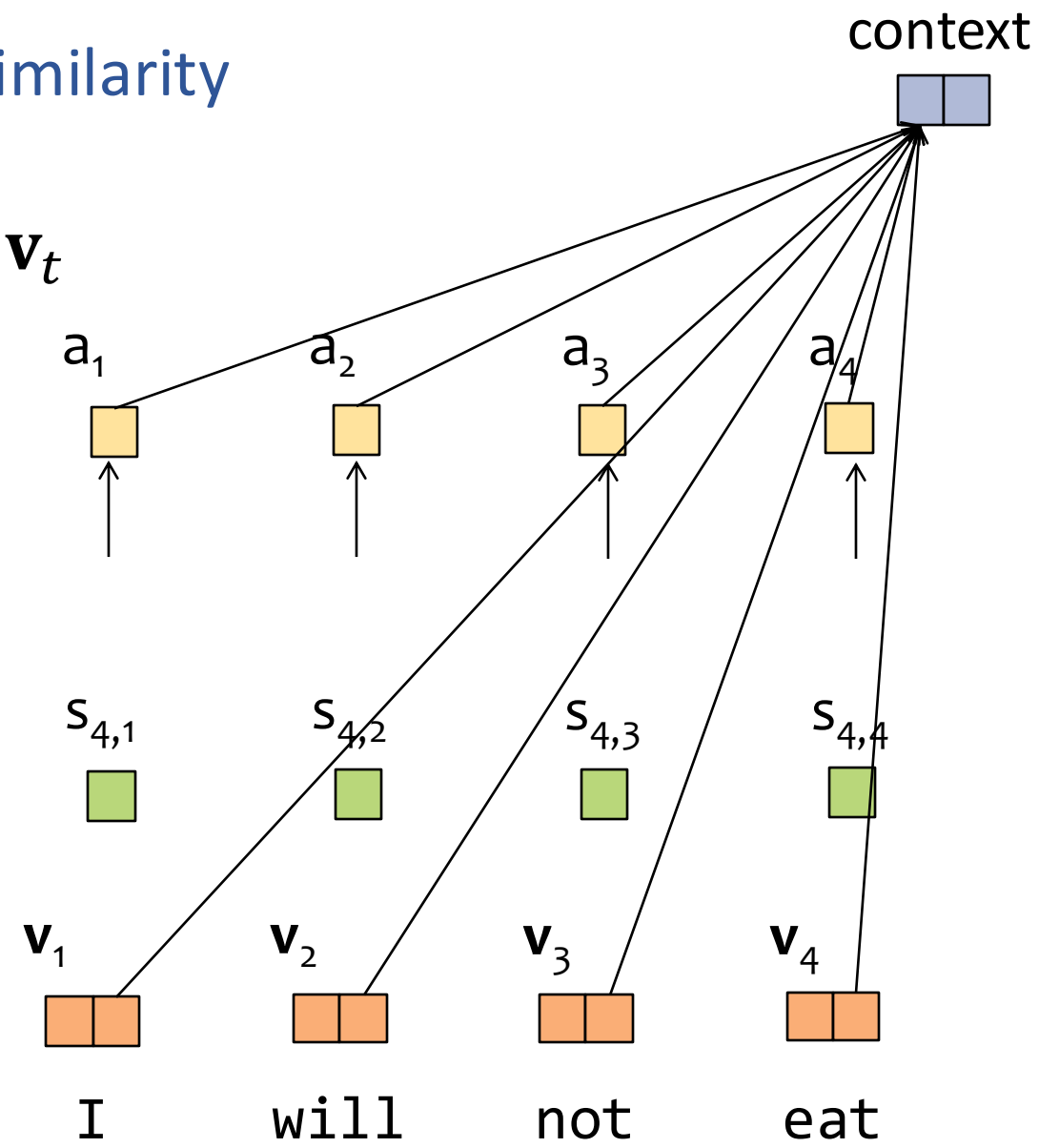
If only we had a way to measure vector similarity

$$\mathbf{v}' = \sum_{t=1}^{T} a_t \mathbf{v}_t$$

Cosine similarity matrix!

$$S = VV^T$$



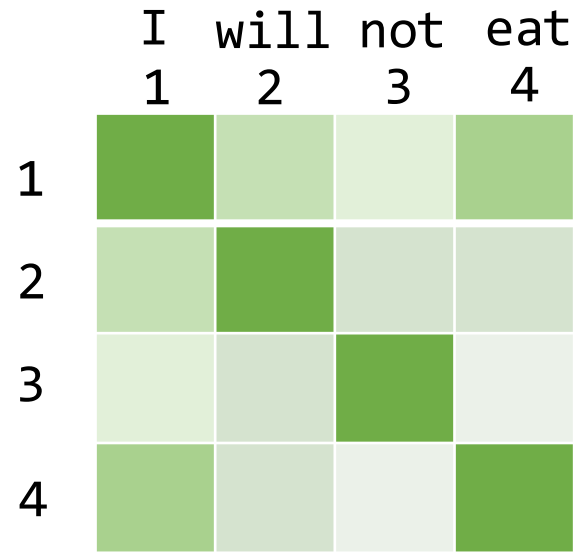
# Learn to pay attention!

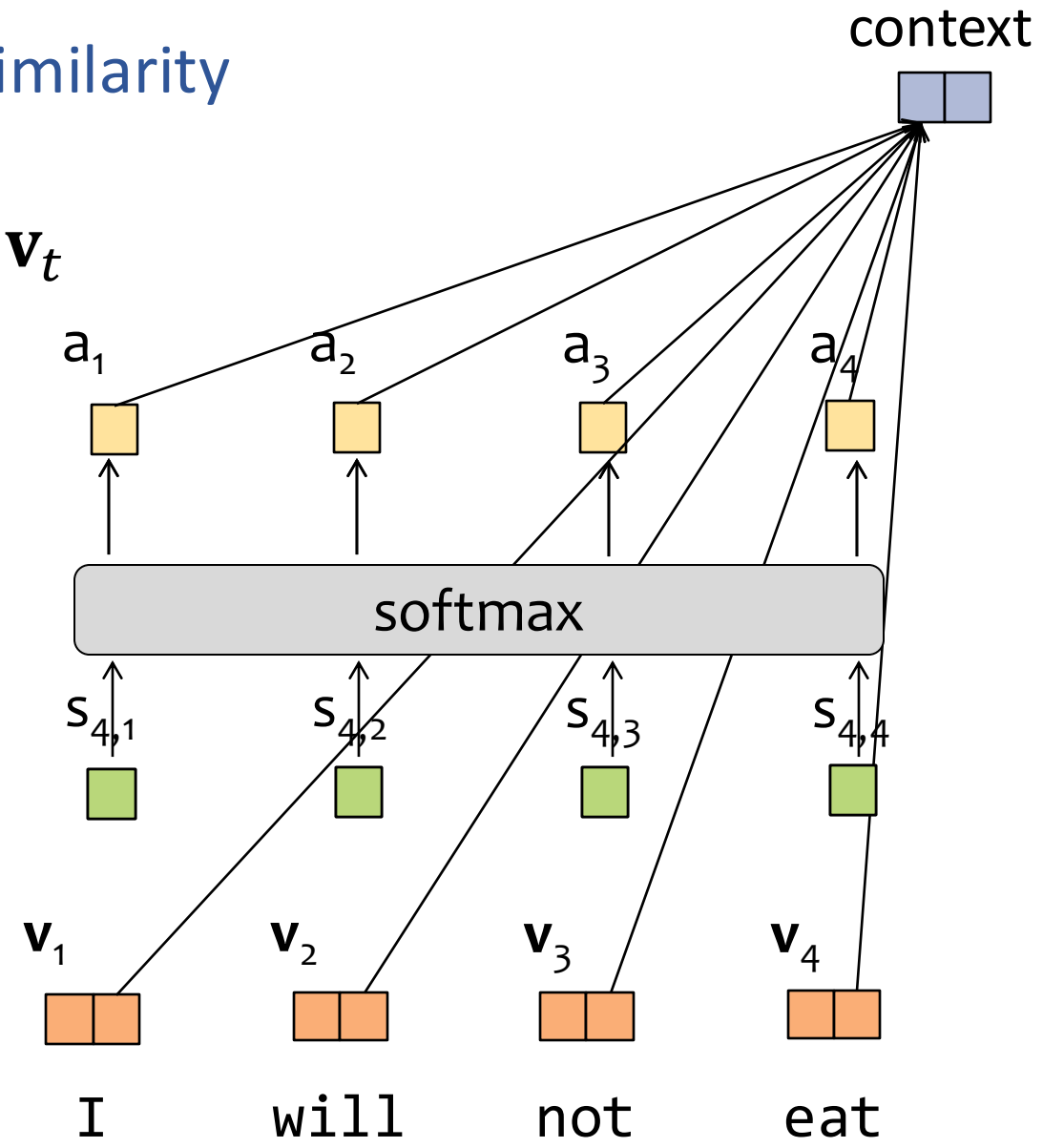
If only we had a way to measure vector similarity

$$\mathbf{v}' = \sum_{t=1}^{T} a_t \mathbf{v}_t$$

Cosine similarity matrix!

$$S = VV^T$$



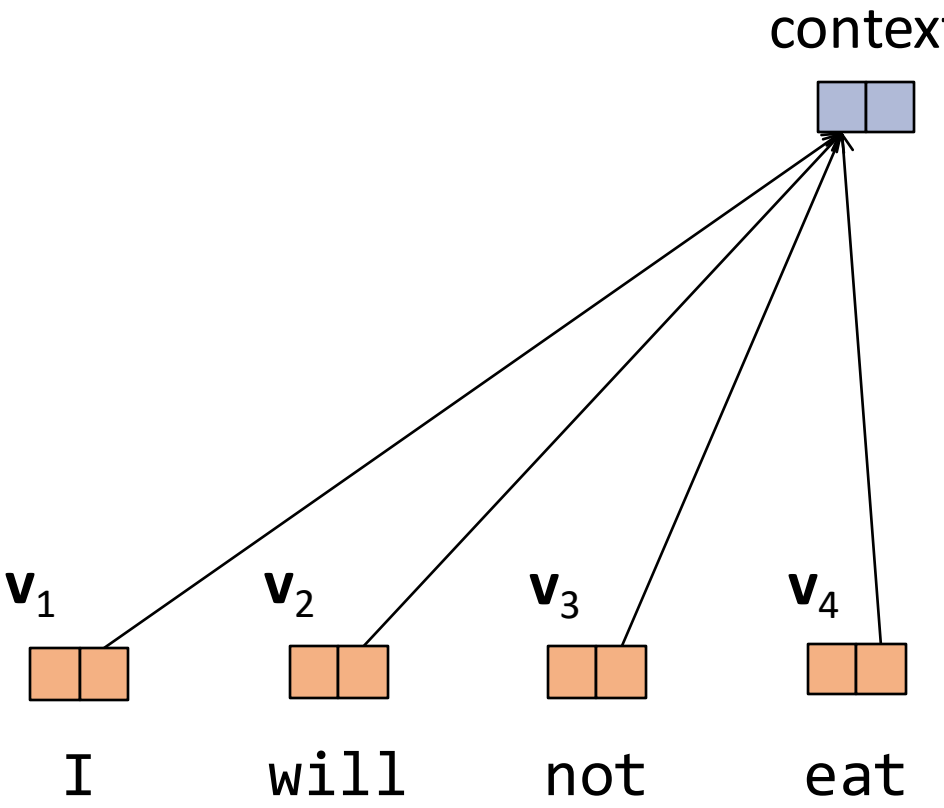



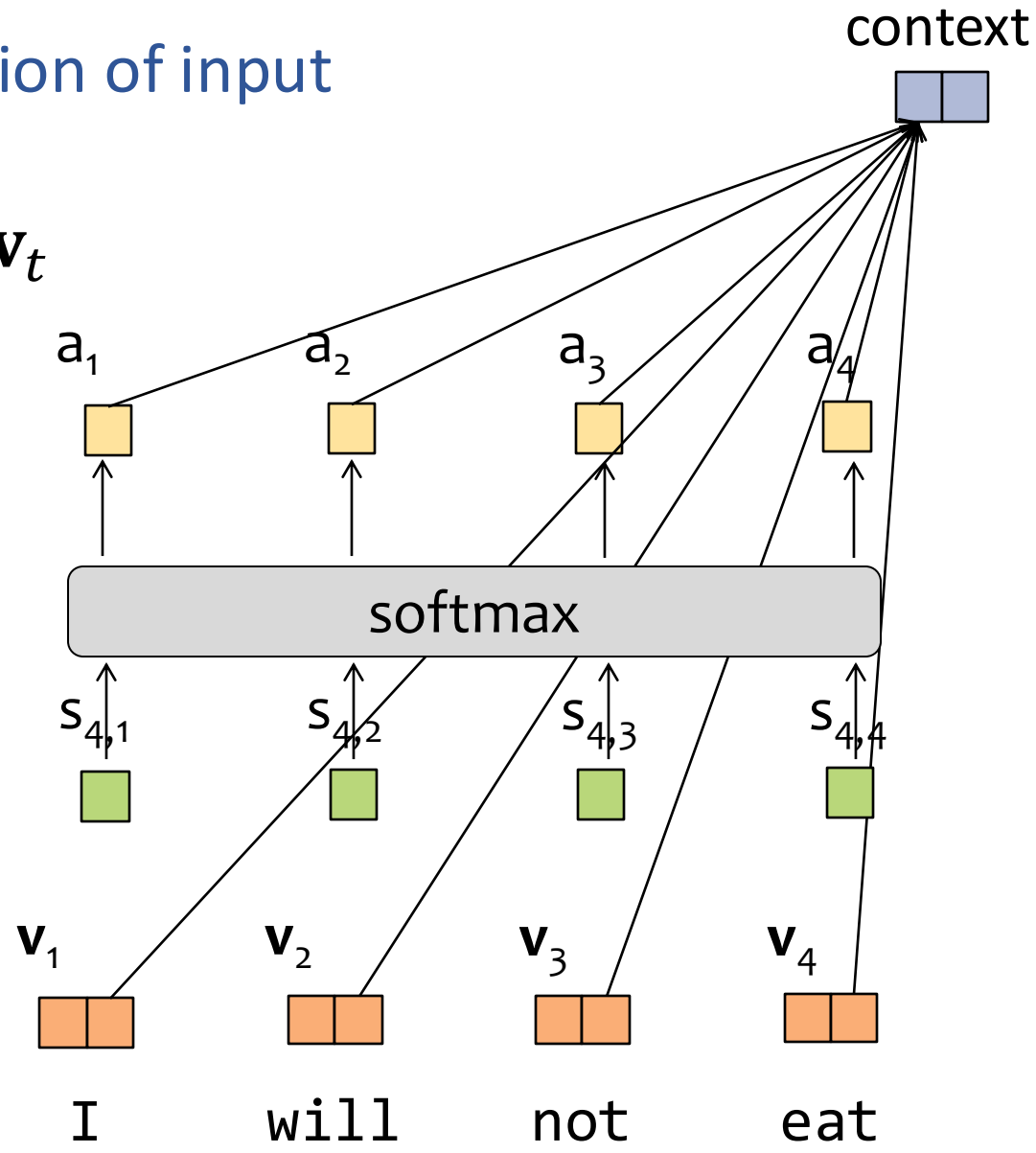
# Learn to pay attention!

## We can do better than uniform combination of input

$$\mathbf{v} = \sum_{t=1}^{T} \frac{1}{T} \mathbf{v}_t$$

$$\mathbf{v} = \sum_{t=1}^{T} a_t \mathbf{v}_t$$



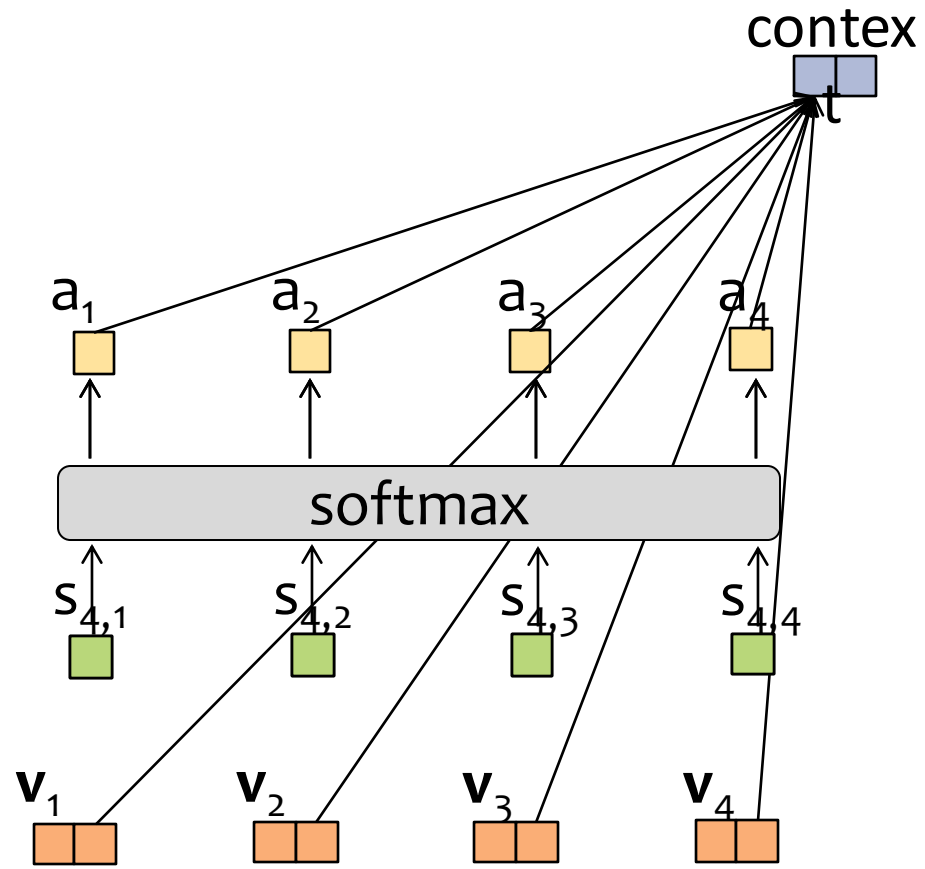
# Learn to pay attention!

But...there is an issue with just doing $VV^T$ ☹

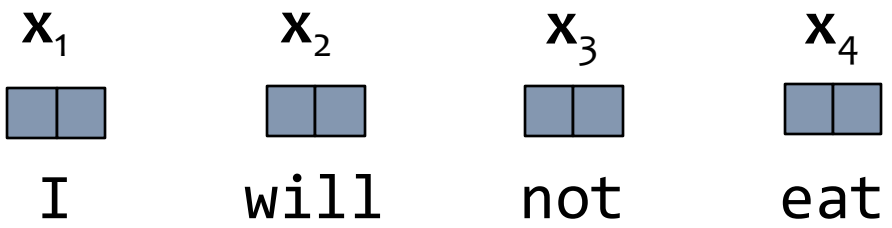
We're really just comparing input to input

→ Symmetric with strong diagonal ☹

$$S = VV^T$$

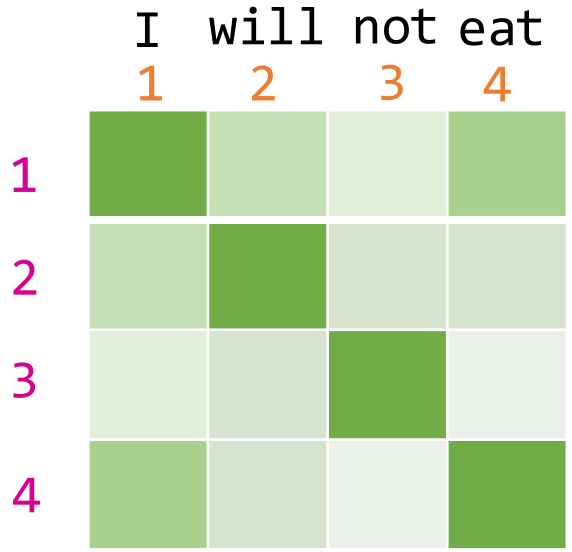

$$V = XW_V$$

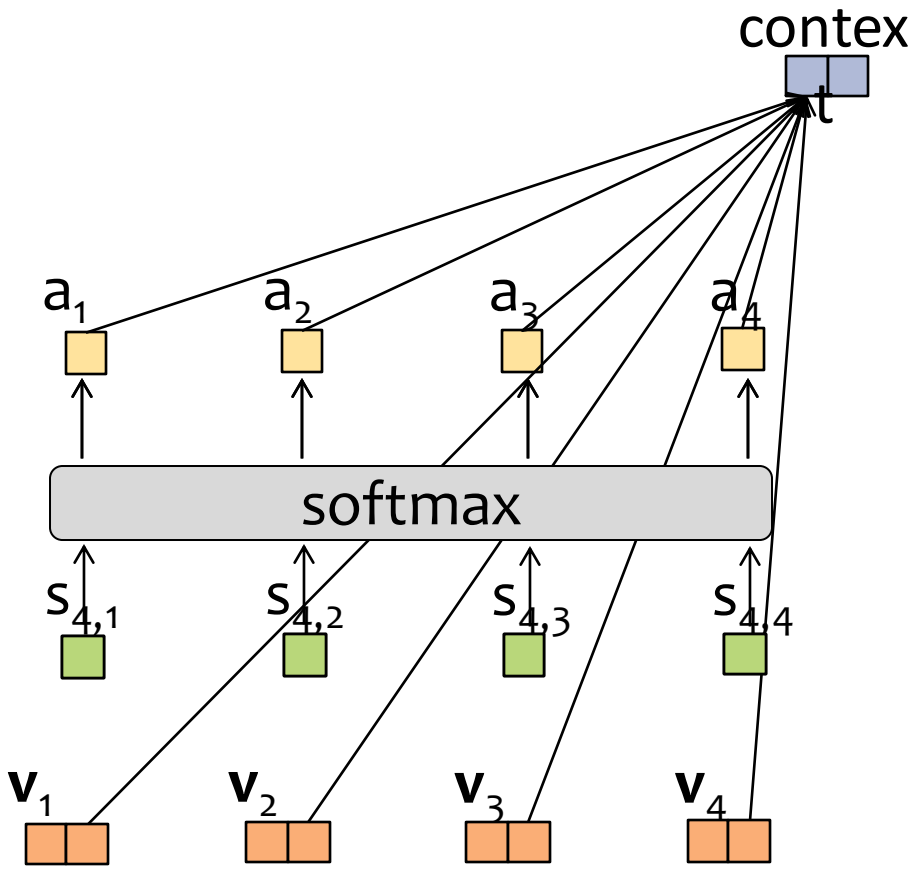


# Learn to pay attention!

Instead learn a query vectors $\mathbf{q}_t$ to represent the output

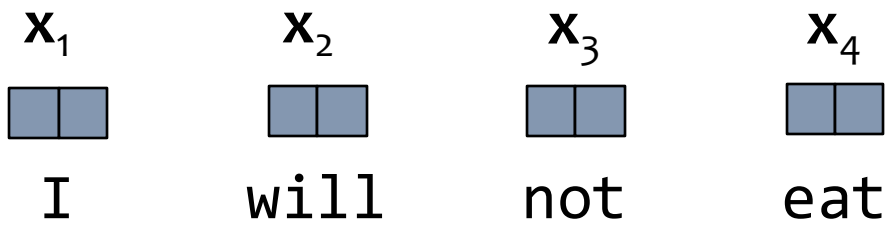
$$S = QV^T$$




$$Q = XW_Q$$

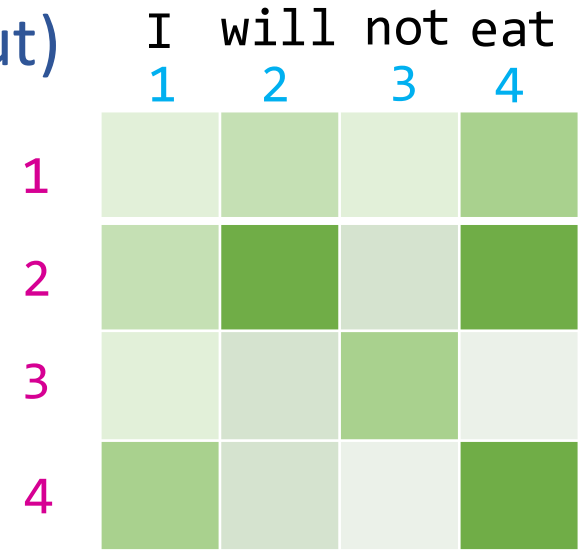
$$V = XW_V$$

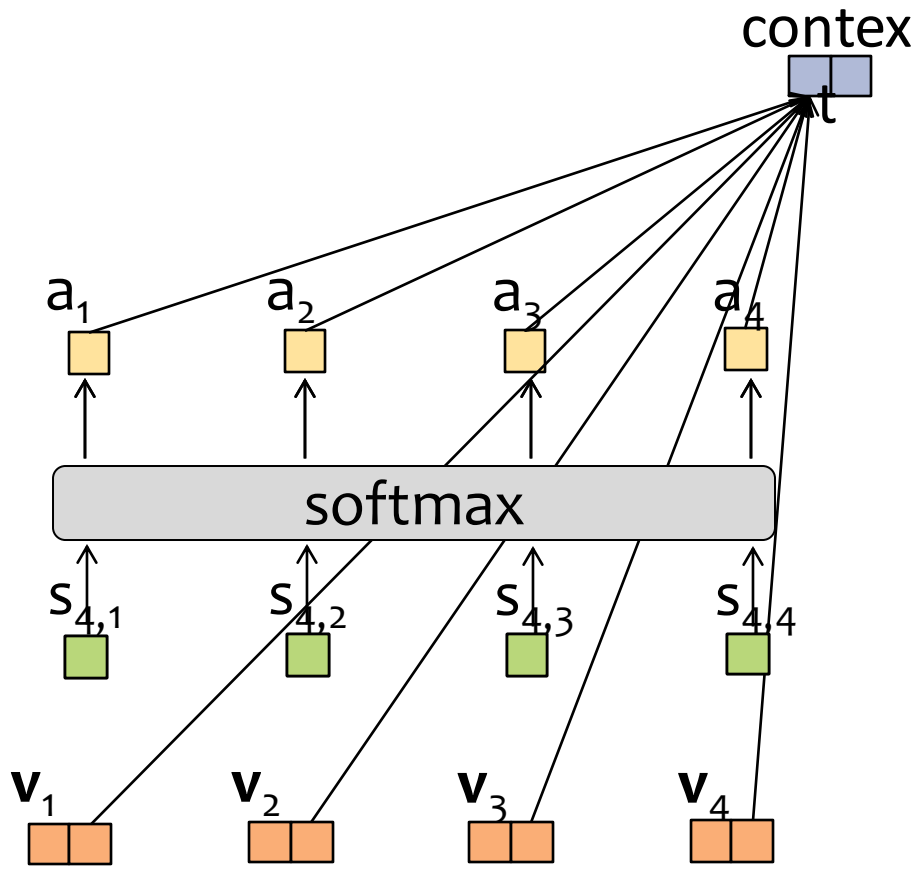


# Learn to pay attention!

Instead learn a query vectors $\mathbf{q}_t$ to represent the output

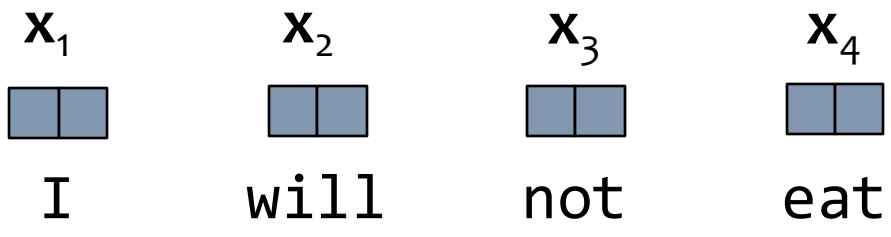
(And also $\mathbf{k}_t$ for the input)




$$S = QK^T/\sqrt{d_k}$$

$$Q = XW_Q \qquad K = XW_K \qquad V = XW_V$$



# Learn to pay attention!

Instead learn a query vectors $\mathbf{q}_t$ to represent the output

(And also $\mathbf{k}_t$ for the input)

Attention:

Query, Key, Value

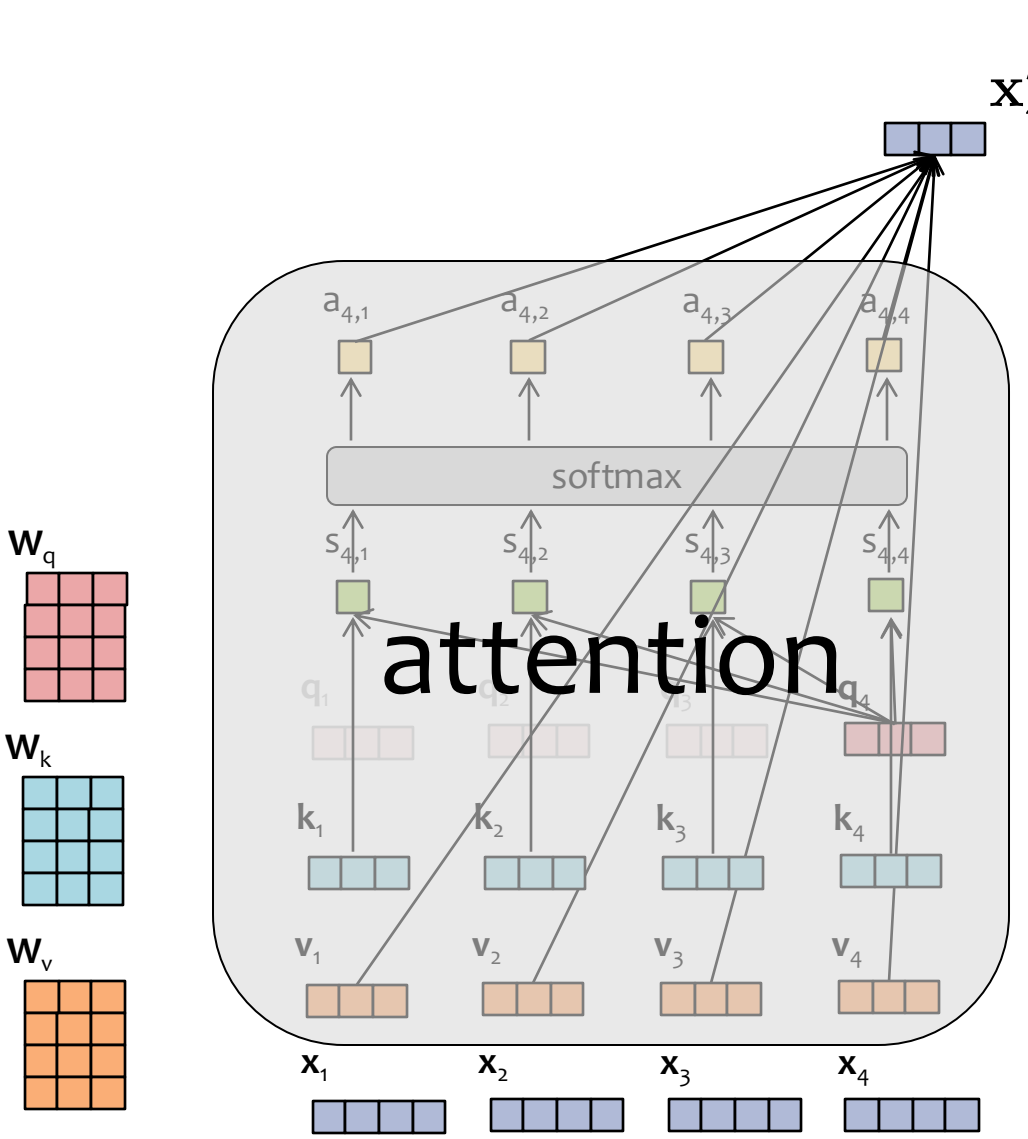
$$Q = XW_Q$$

$$K = XW_K$$

$$V = XW_V$$

$$S = QK^T/\sqrt{d_k}$$

# Scaled Dot-Product Attention




$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

# Attention

## Vector representation

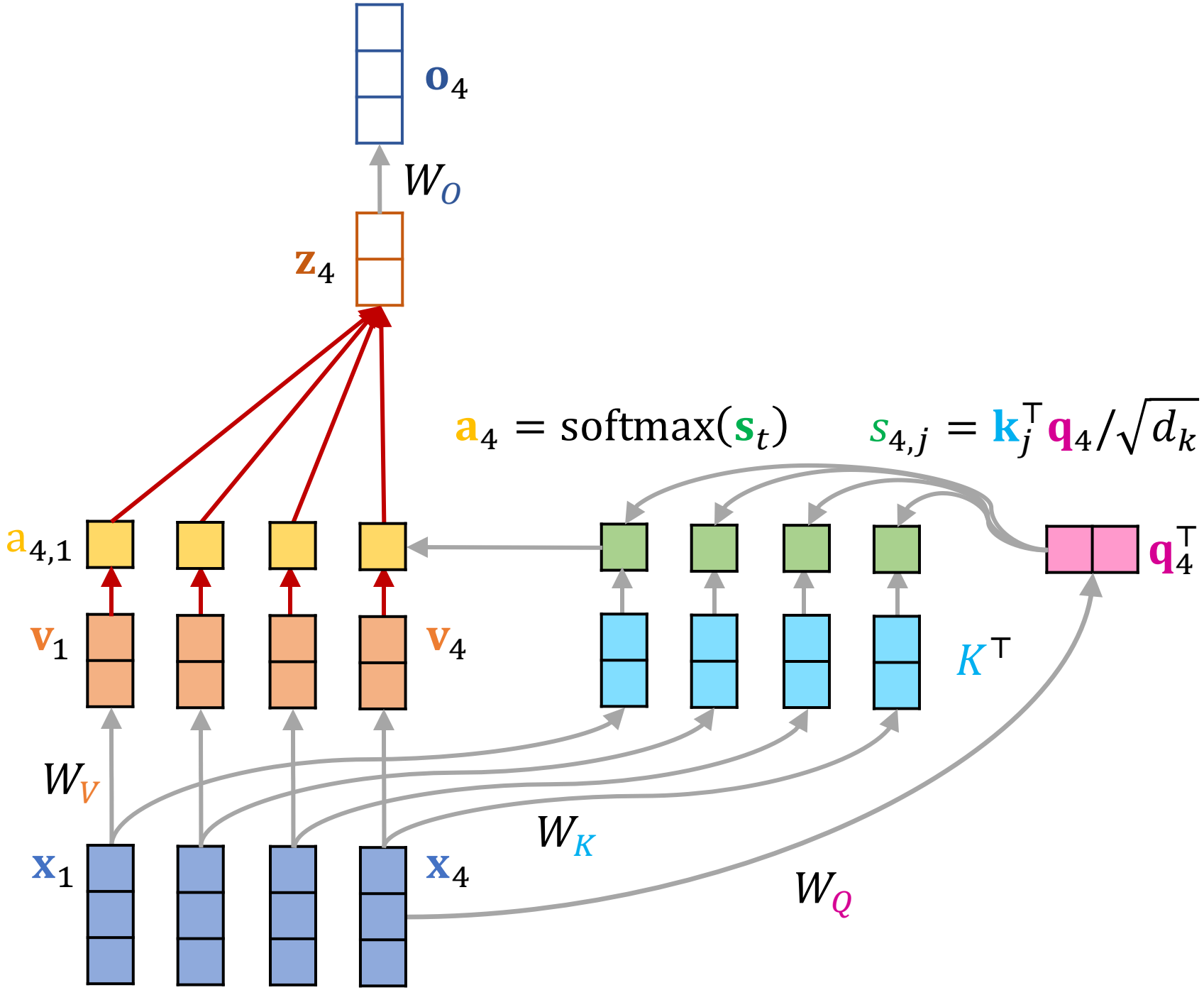
$$\mathbf{o}_t = W_O^\top \mathbf{z}_t$$

$$\mathbf{z}_t = V^\top \mathbf{a}_t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t)$$

$$\mathbf{s}_t = K\mathbf{q}_t/\sqrt{d_k}$$

$$\mathbf{q}_t = W_Q^\top \mathbf{x}_t$$
$$\mathbf{k}_t = W_K^\top \mathbf{x}_t$$
$$\mathbf{v}_t = W_V^\top \mathbf{x}_t$$



# Attention

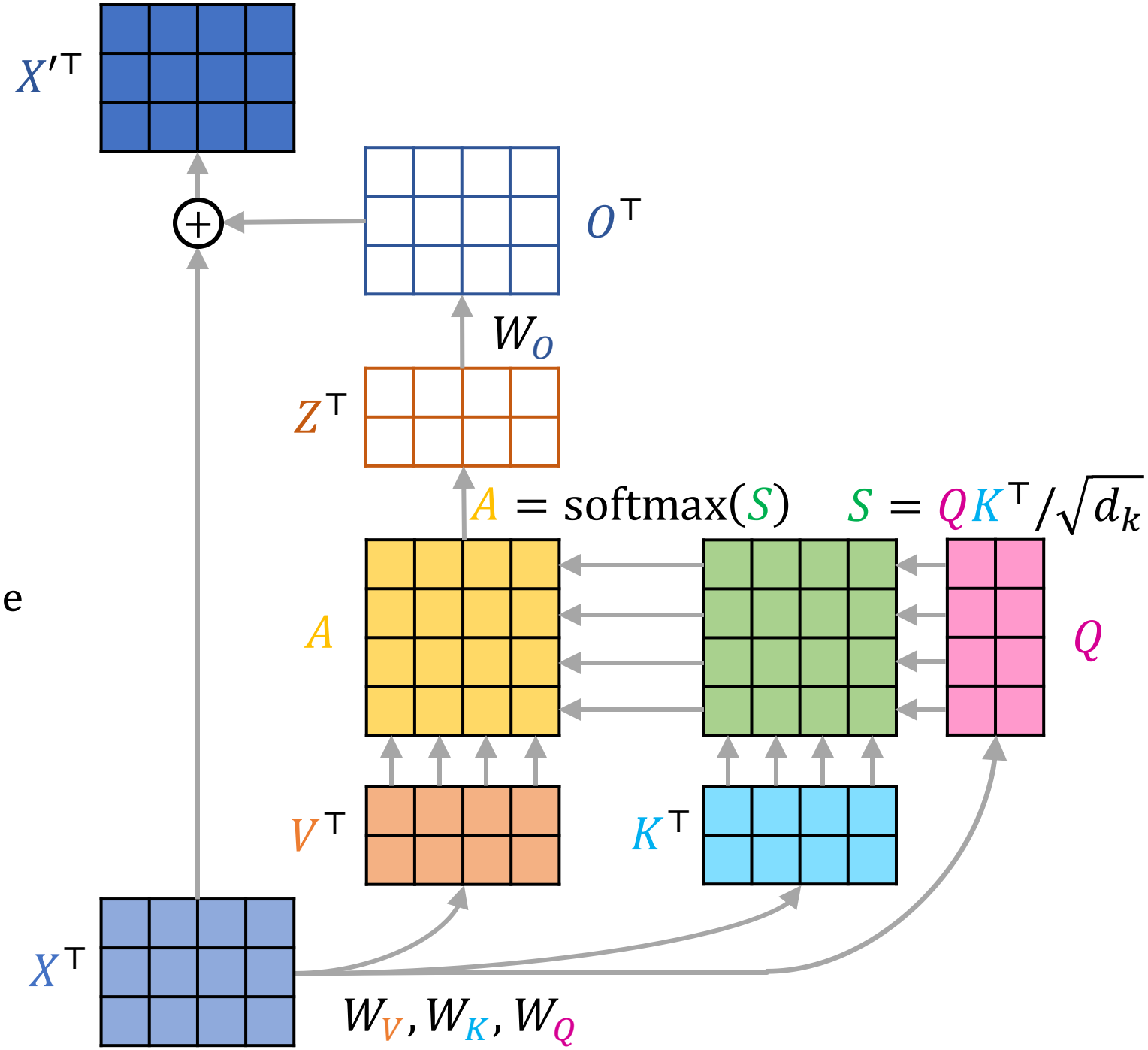
## Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \mathrm{softmax}(S)_{\mathrm{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



# Attention

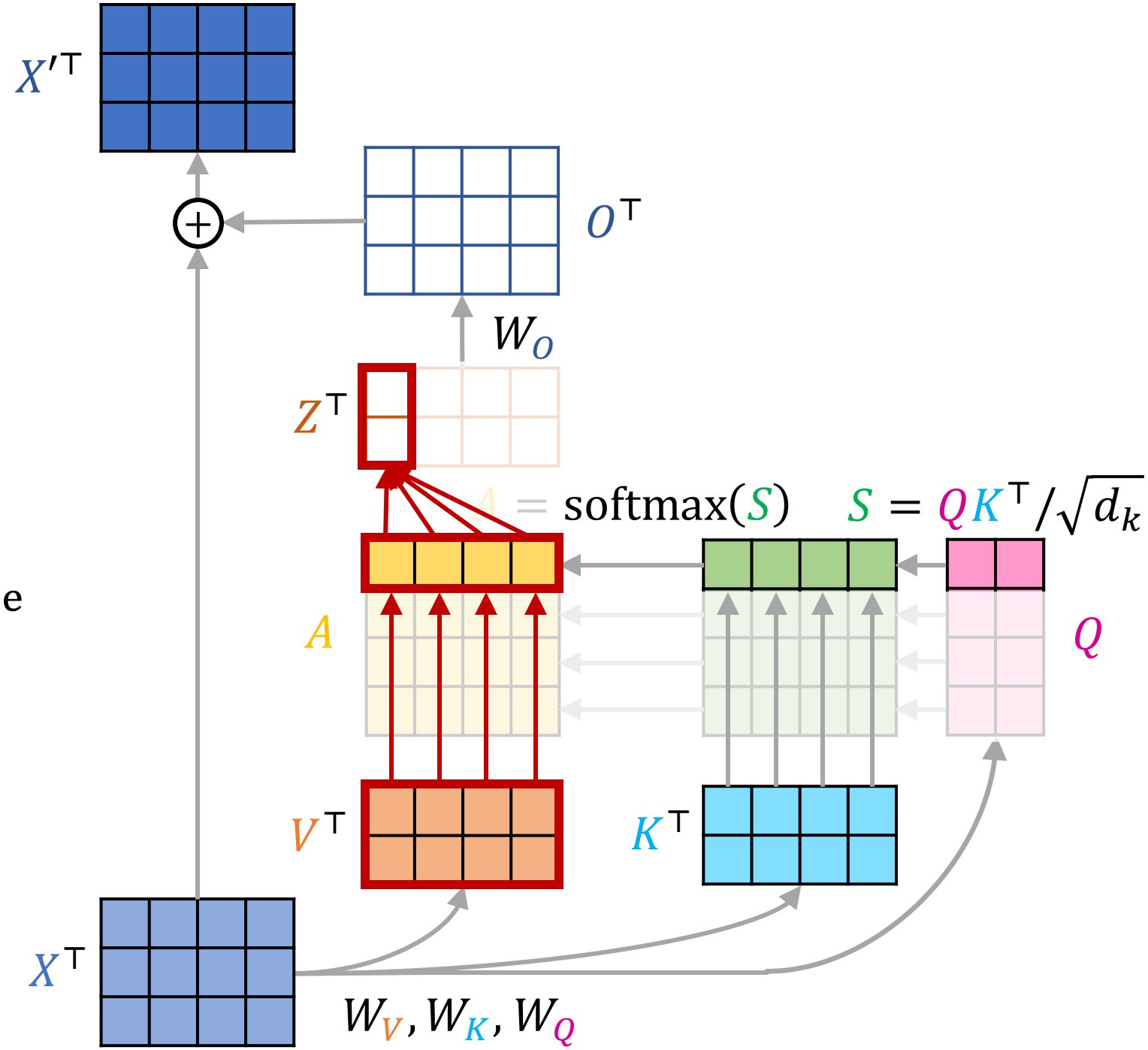
## Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \mathrm{softmax}(S)_{\mathrm{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



# Attention

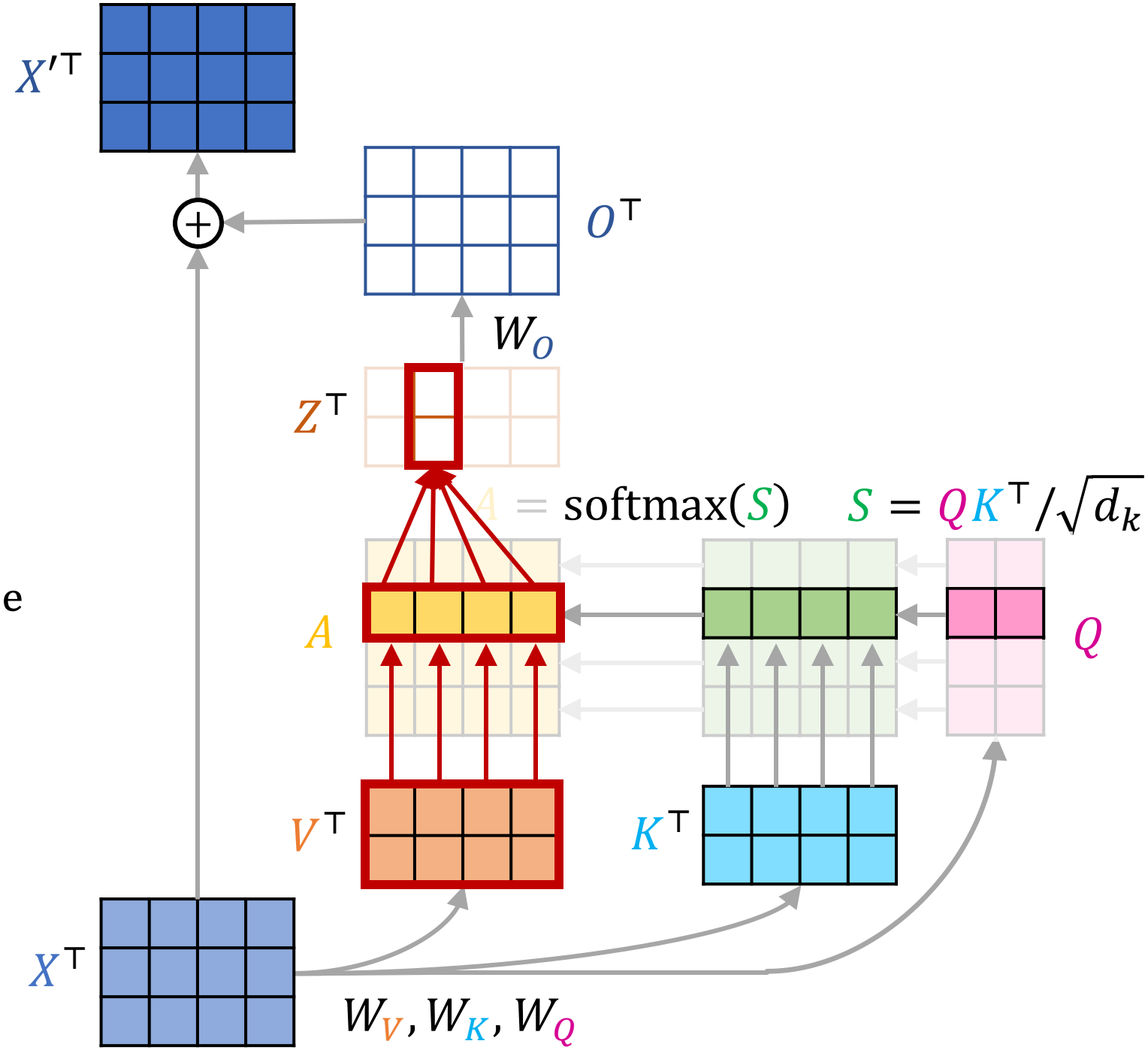
## Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \mathrm{softmax}(S)_{\mathrm{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



# Attention

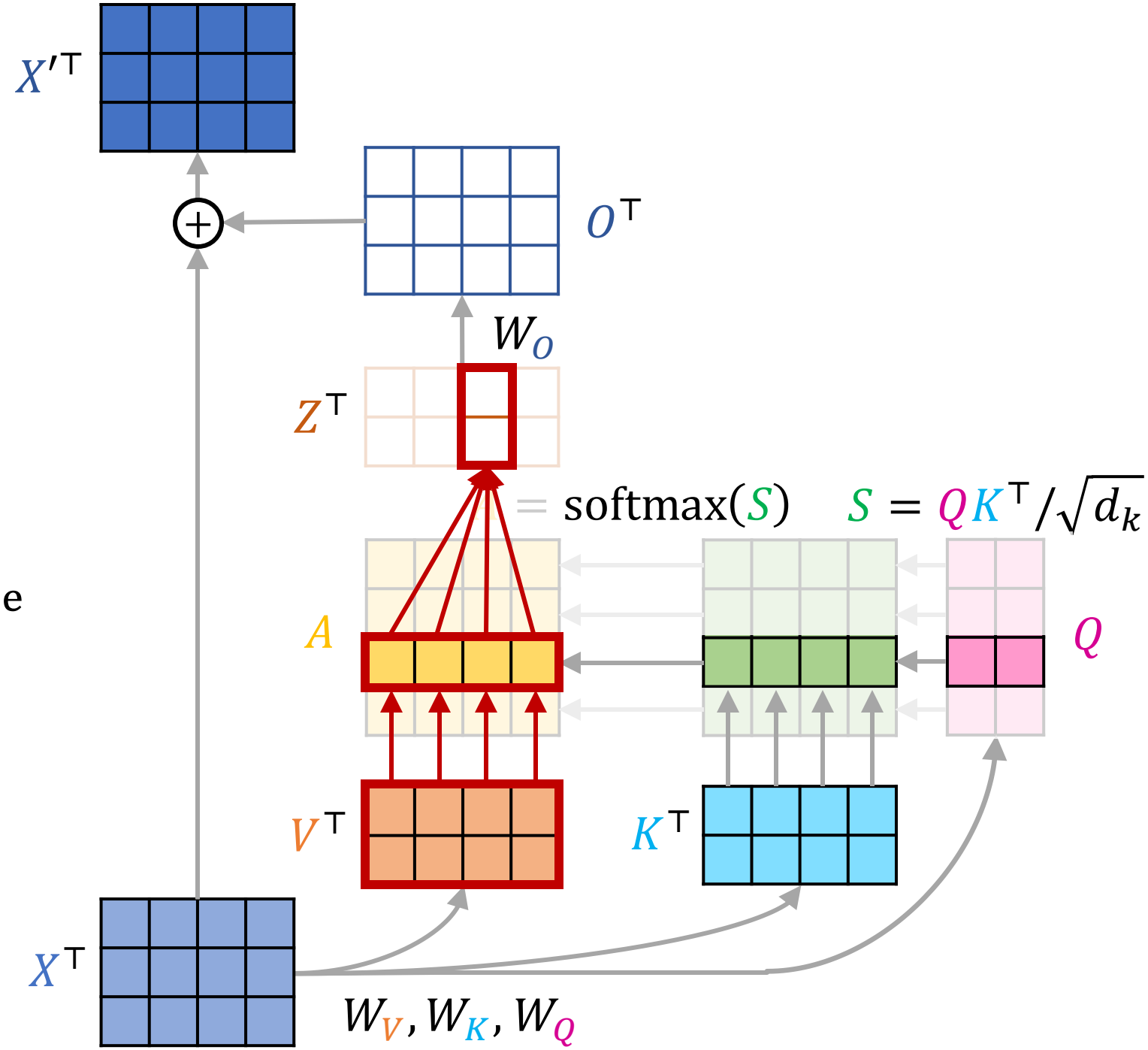
## Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \mathrm{softmax}(S)_{\text{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



# Attention

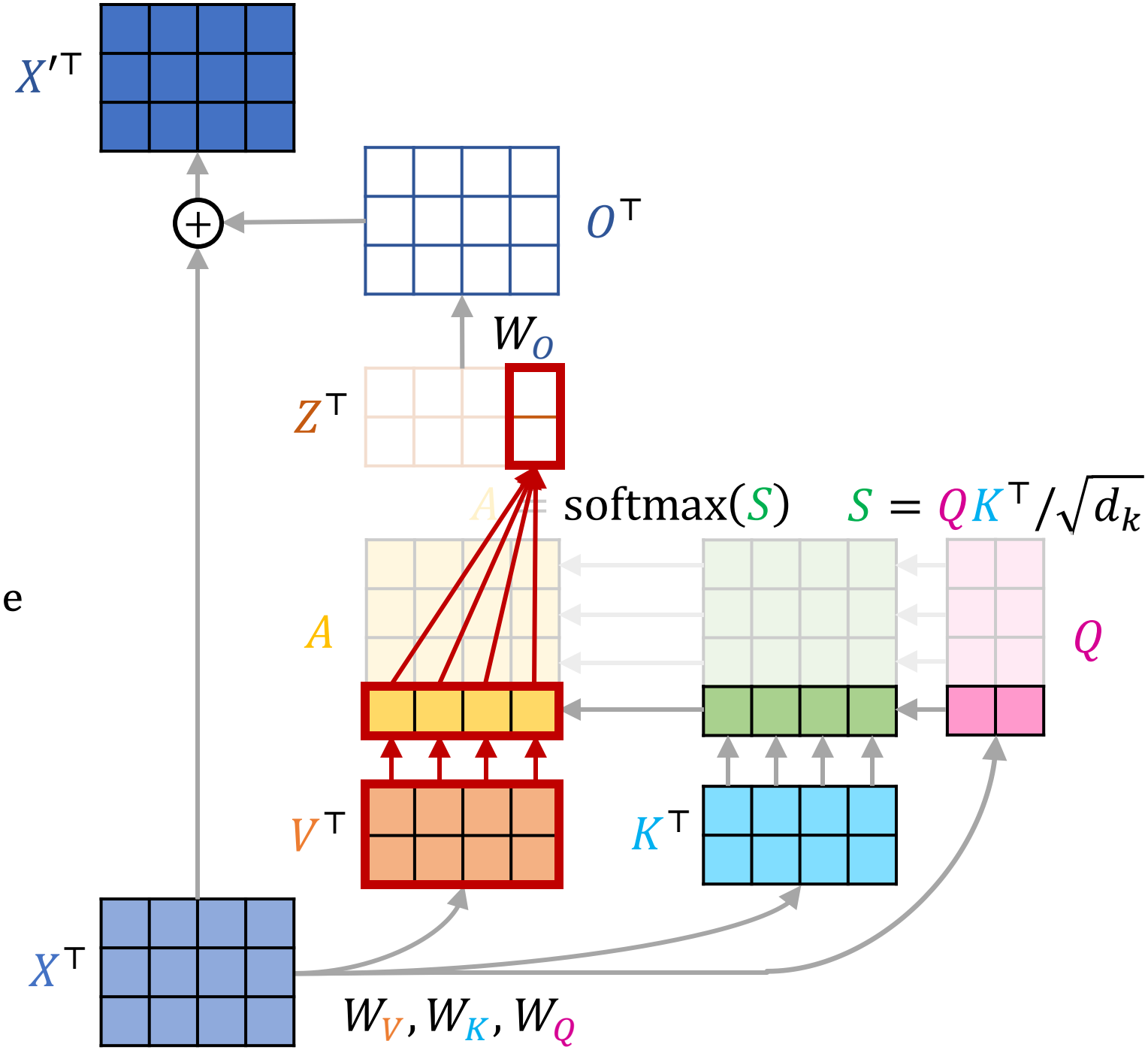
## Matrix representation

$$X' = X \oplus O$$
$$O = ZW_O$$

$$Z = AV$$

$$A = \mathrm{softmax}(S)_{\mathrm{row-wise}}$$
$$S = QK^\top / \sqrt{d_k}$$

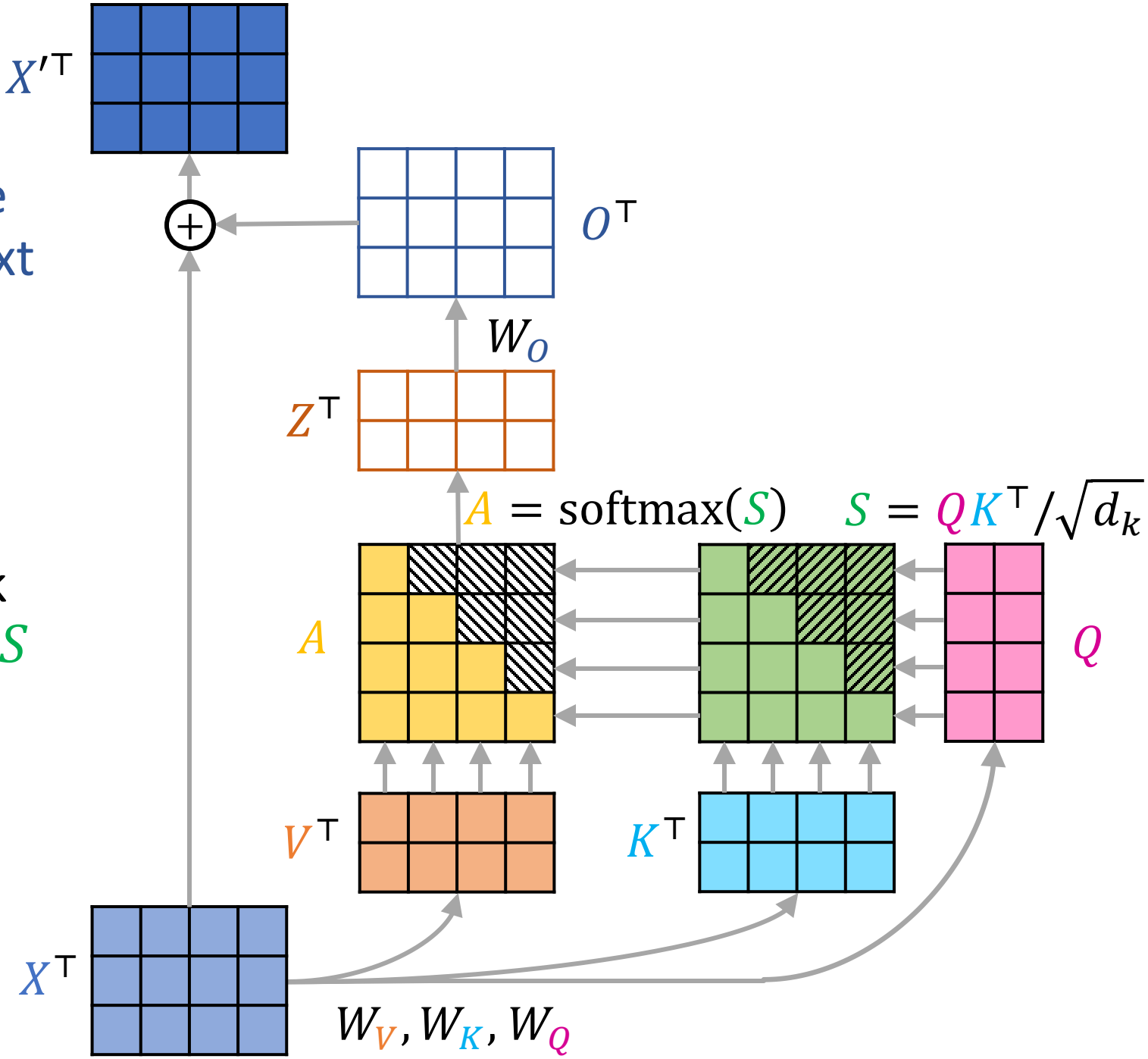
$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$



# Causal Attention

When learning to generate the next token from context $1{:}\,t$, we don't want to look ahead and use any information from $t+1$ or greater

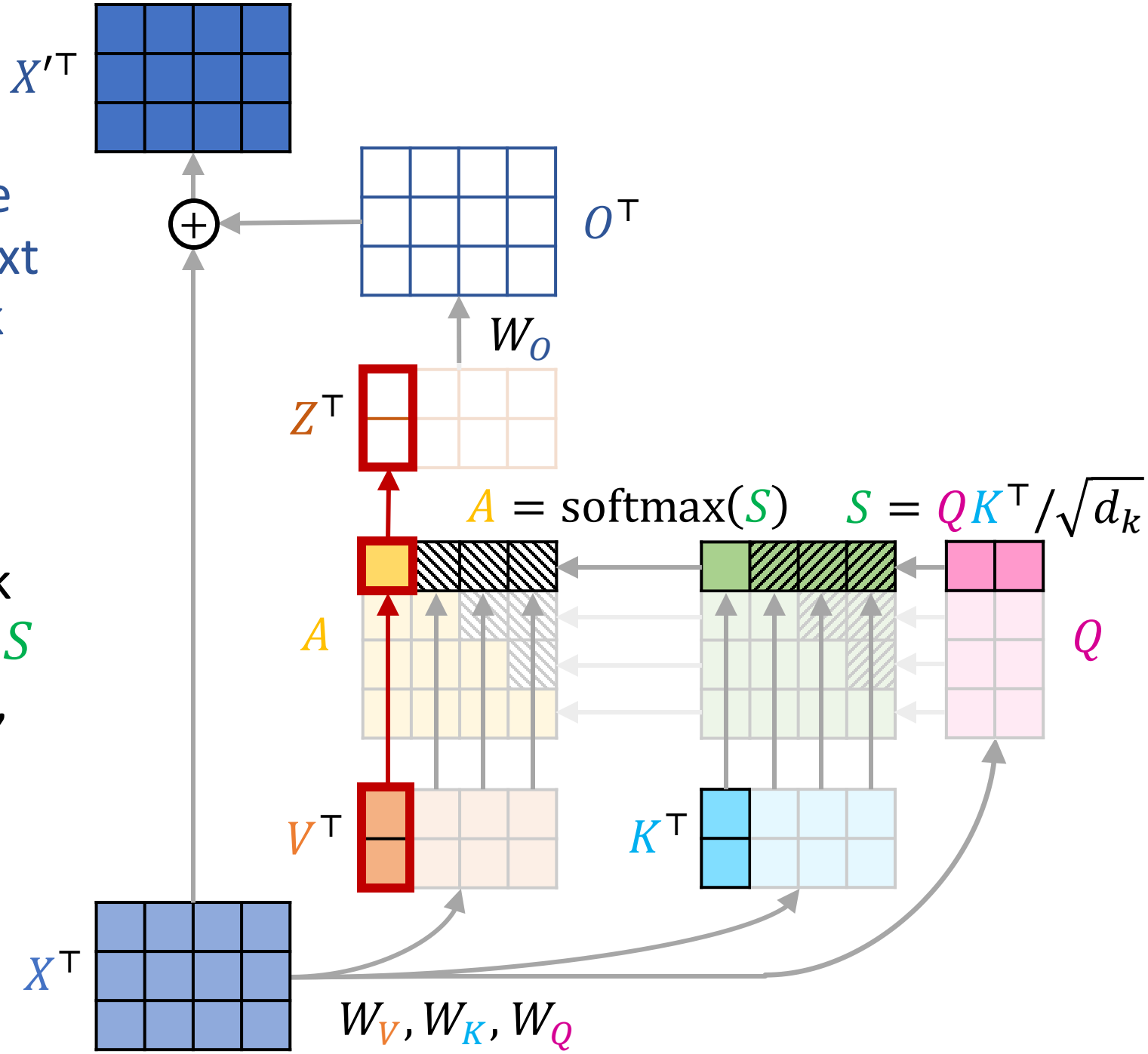
- We apply a causal mask to the attention scores $S$ (filled with $-\infty$ values), which zeros out the appropriate attention weights in $A$



# Causal Attention

When learning to generate the next token from context $1{:}\,t$, we don't want to look ahead and use any information from $t+1$ or greater

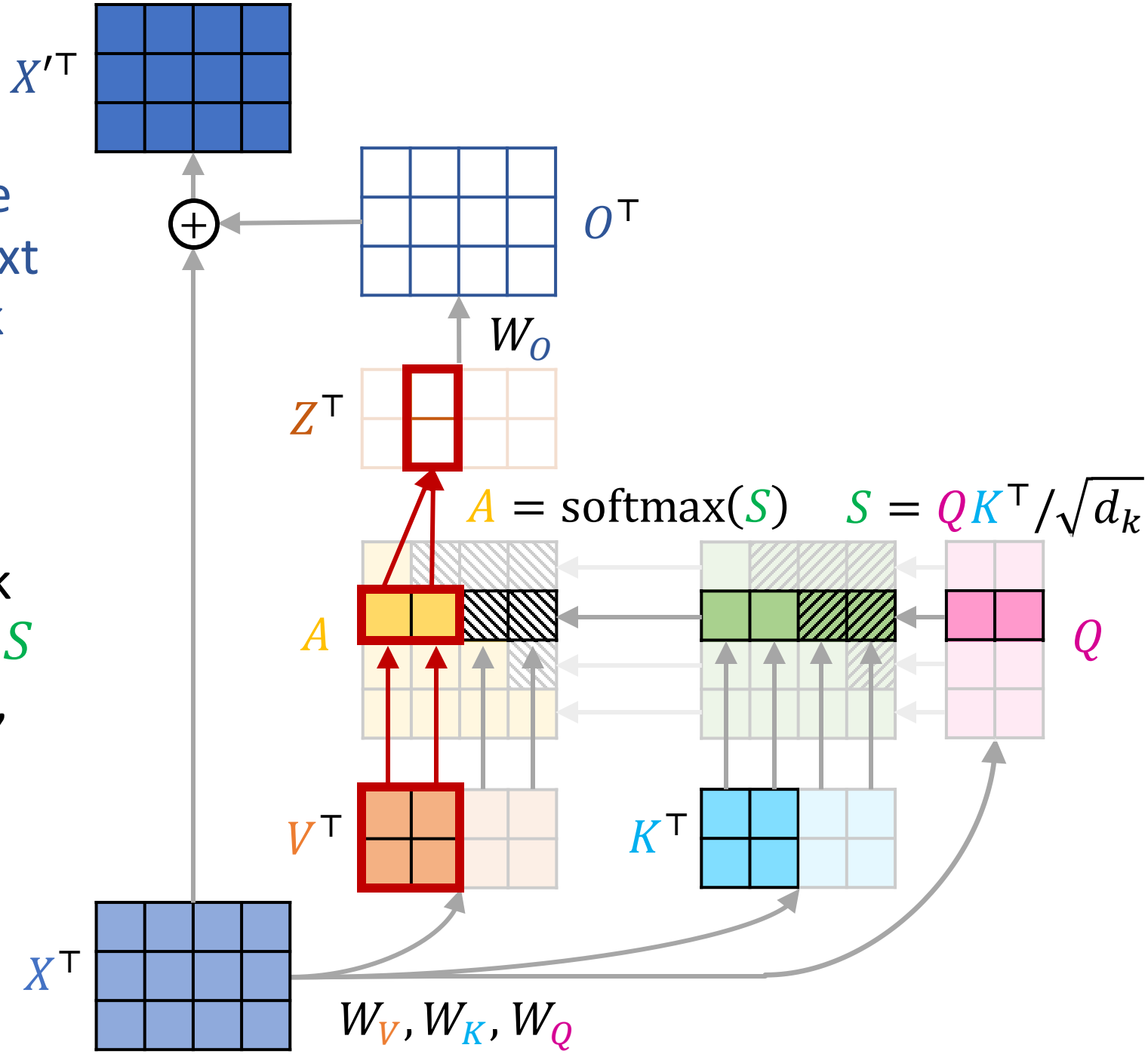
- We apply a causal mask to the attention scores $S$ (filled with $-\infty$ values), which zeros out the appropriate attention weights in $A$



# Causal Attention

When learning to generate the next token from context $1{:}\,t$, we don't want to look ahead and use any information from $t+1$ or greater

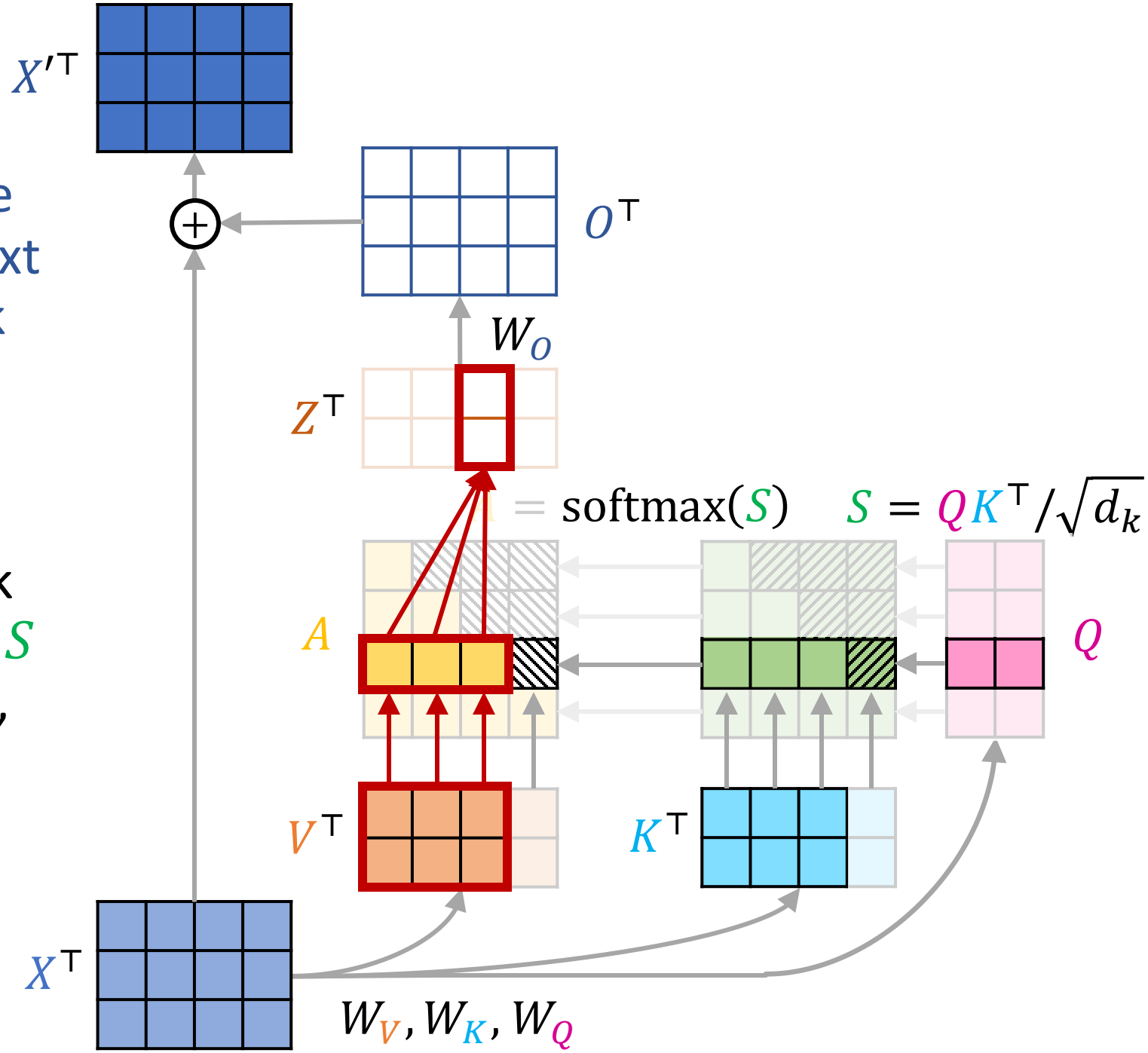
- We apply a causal mask to the attention scores $S$ (filled with $-\infty$ values), which zeros out the appropriate attention weights in $A$



# Causal Attention

When learning to generate the next token from context $1{:}\,t$, we don't want to look ahead and use any information from $t+1$ or greater

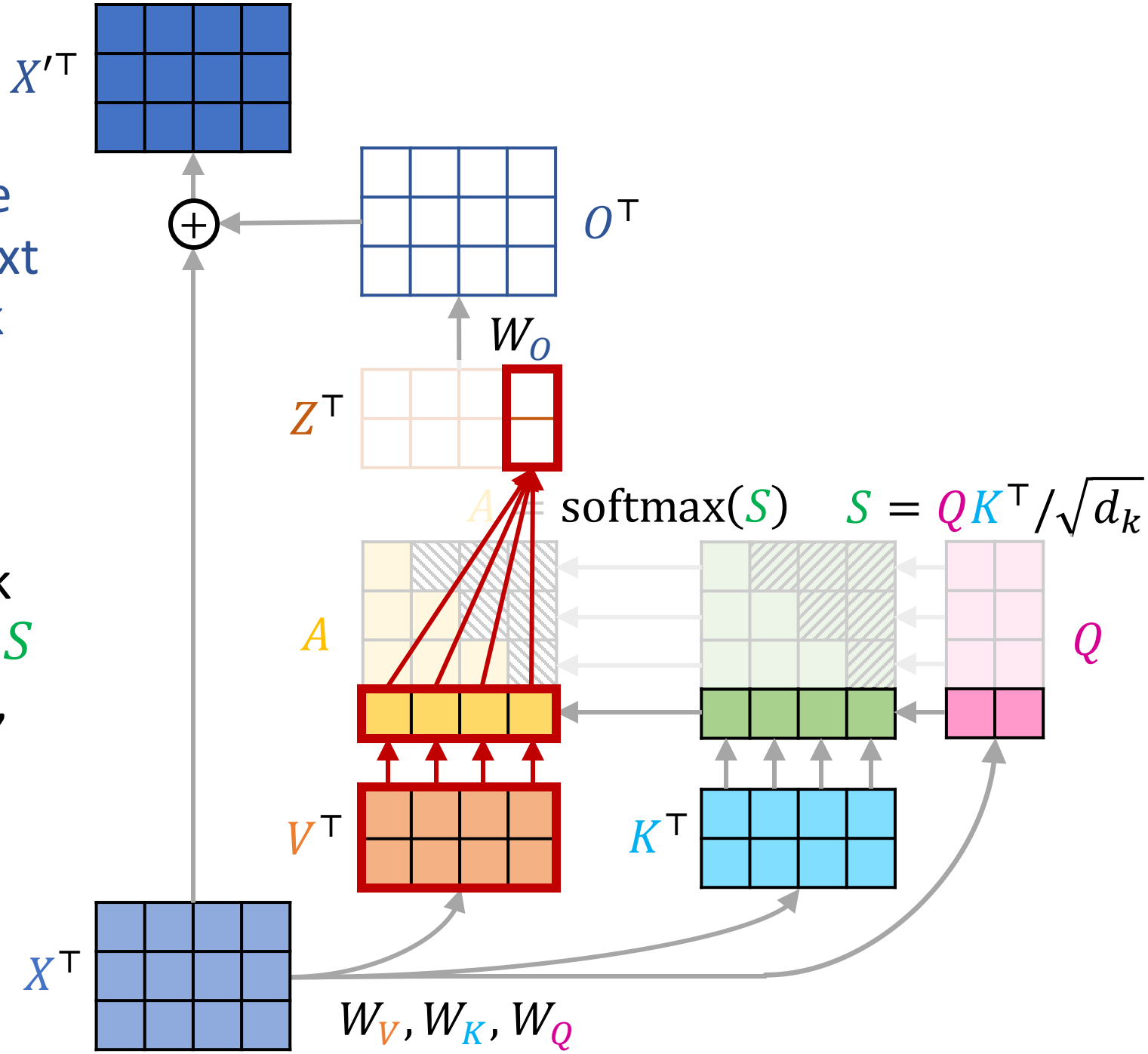
- We apply a causal mask to the attention scores $S$ (filled with $-\infty$ values), which zeros out the appropriate attention weights in $A$



# Causal Attention

When learning to generate the next token from context $1{:}\,t$, we don't want to look ahead and use any information from $t+1$ or greater

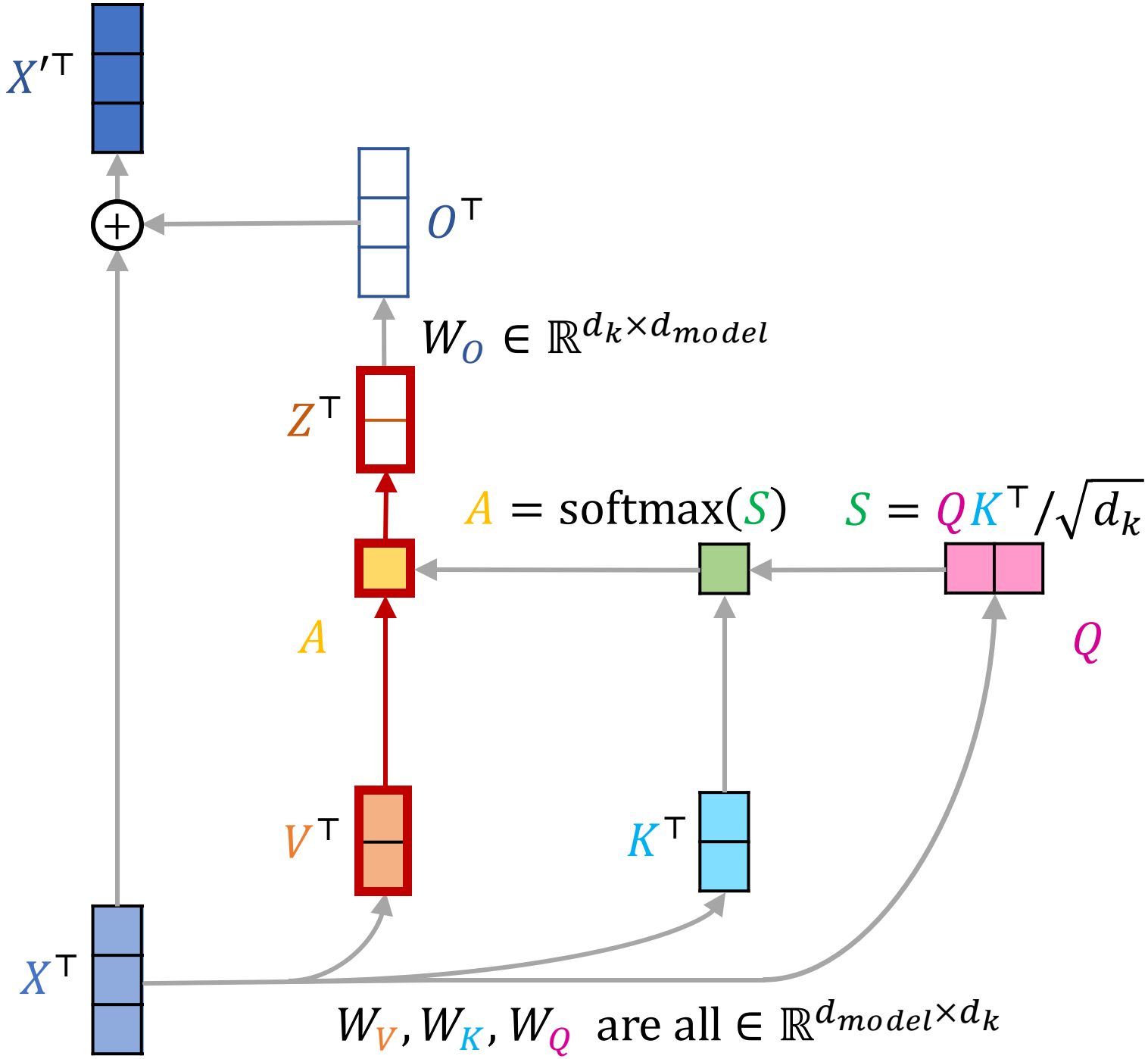
- We apply a causal mask to the attention scores $S$ (filled with $-\infty$ values), which zeros out the appropriate attention weights in $A$



# Causal Attention

## Inference time

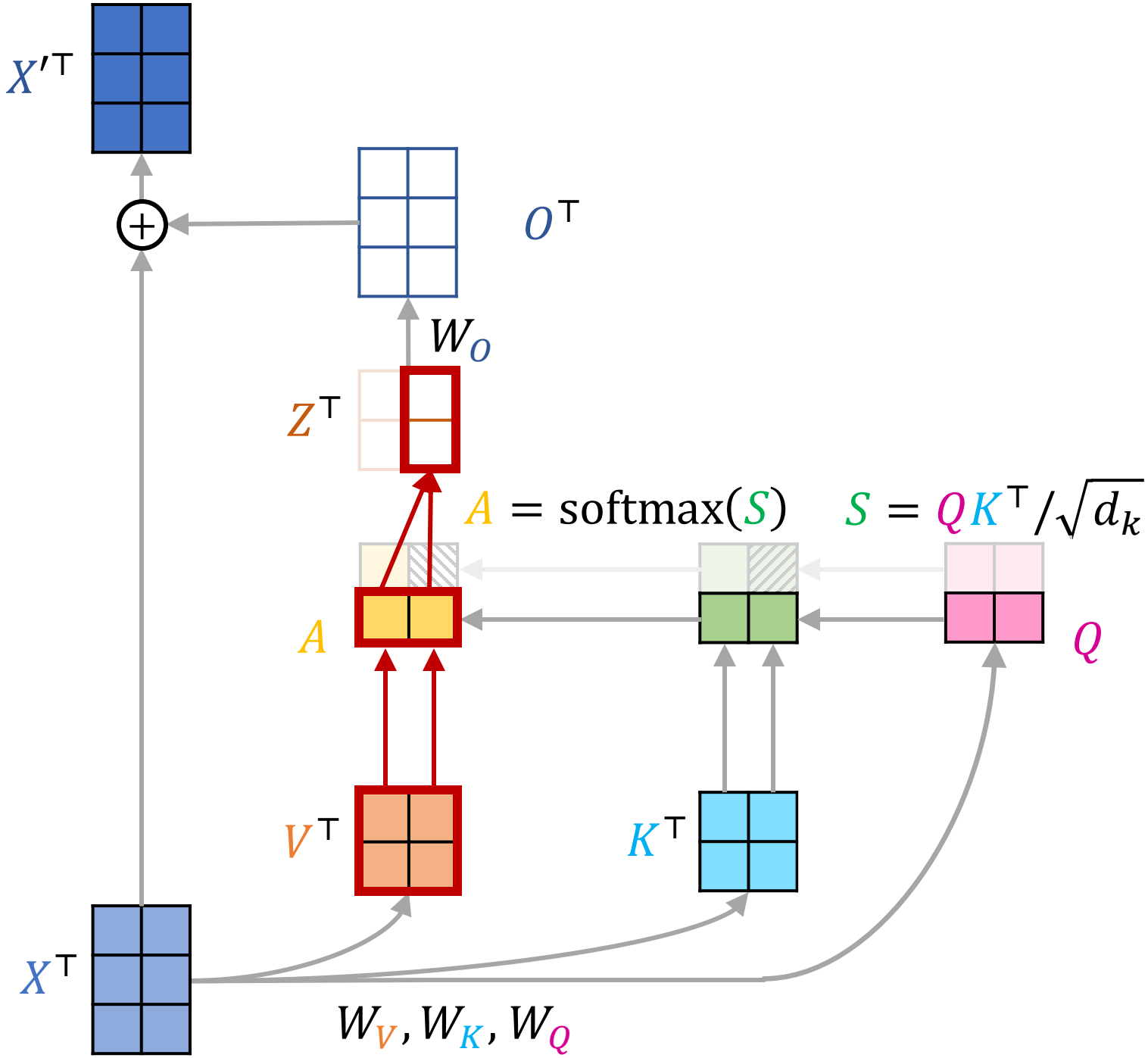
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



# Causal Attention

## Inference time

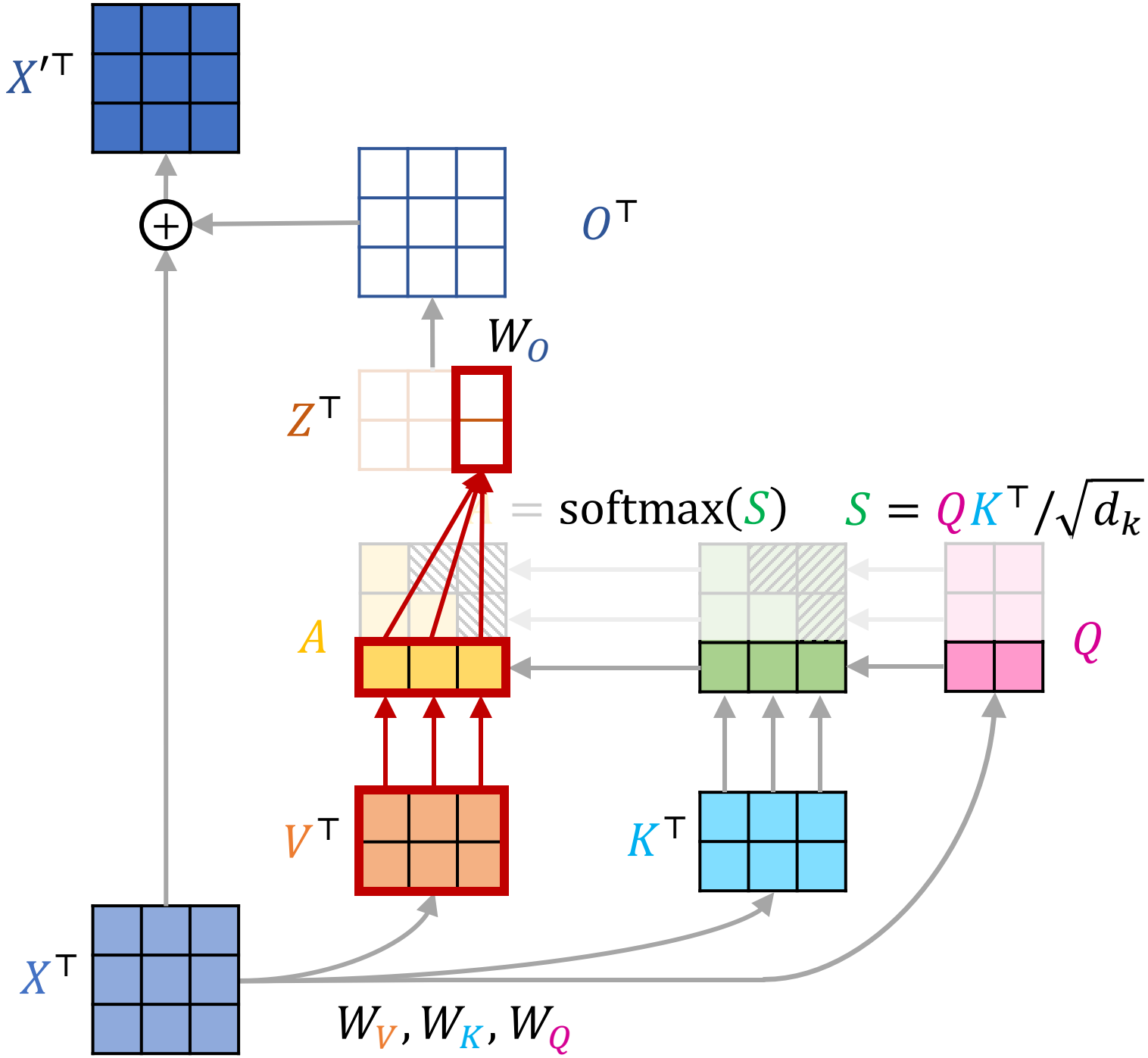
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



# Causal Attention

## Inference time

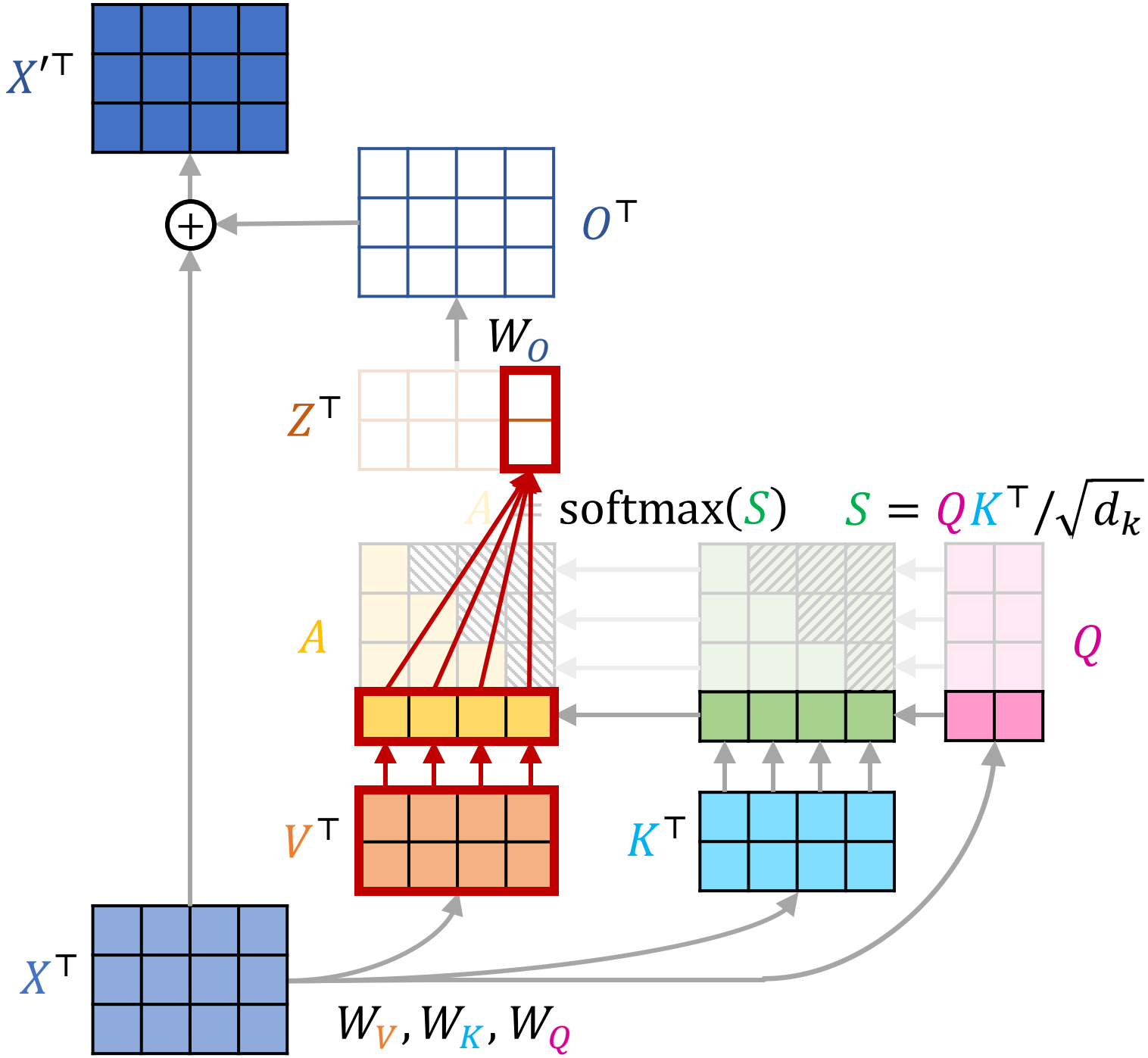
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



# Causal Attention

## Inference time

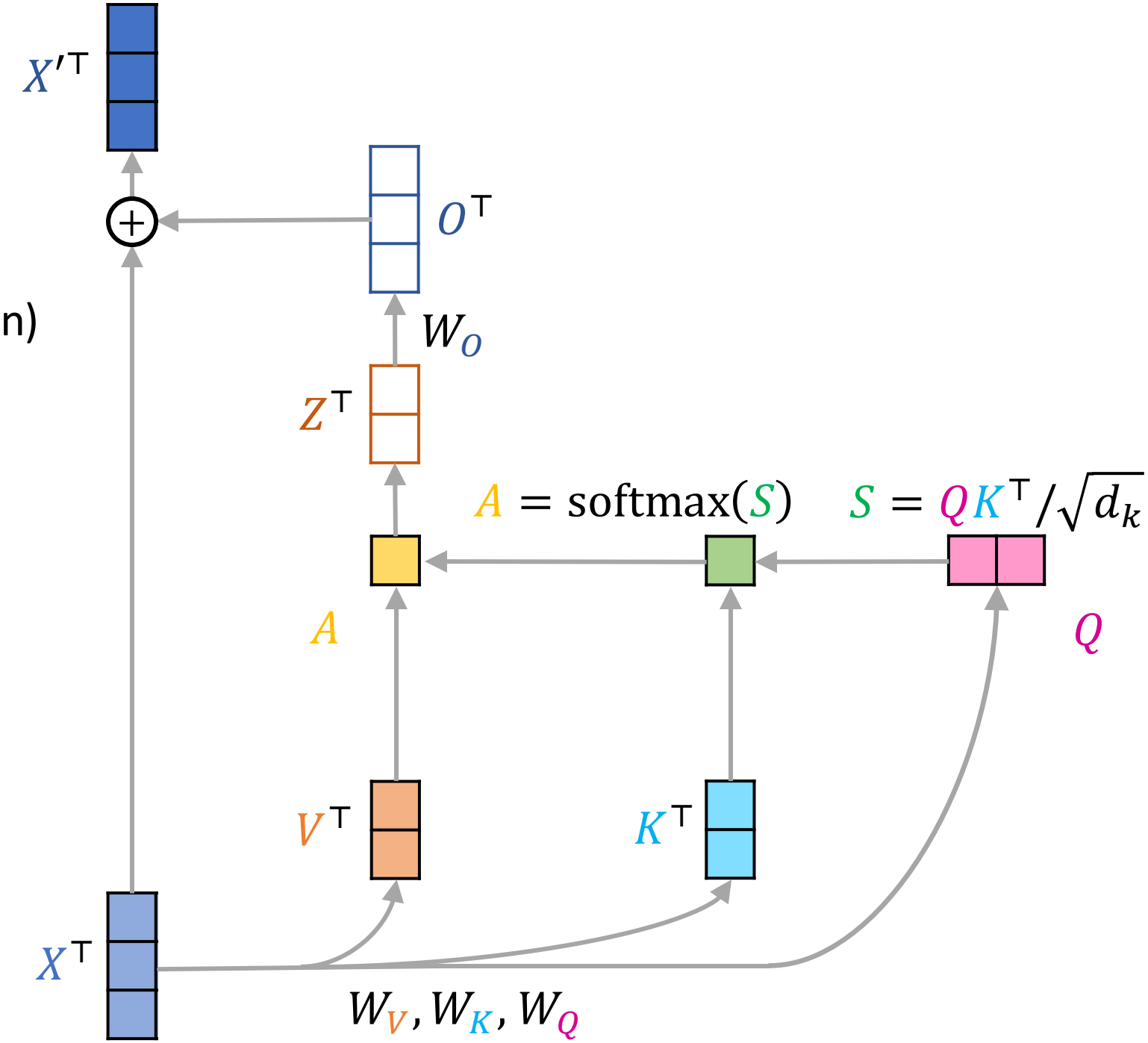
- Done training. Watch this attention block as the context size increases as we generate more and more tokens
- Note how different components build up as the context grows
- But the size of the parameters don't change!



# Causal Attention

## Inference time

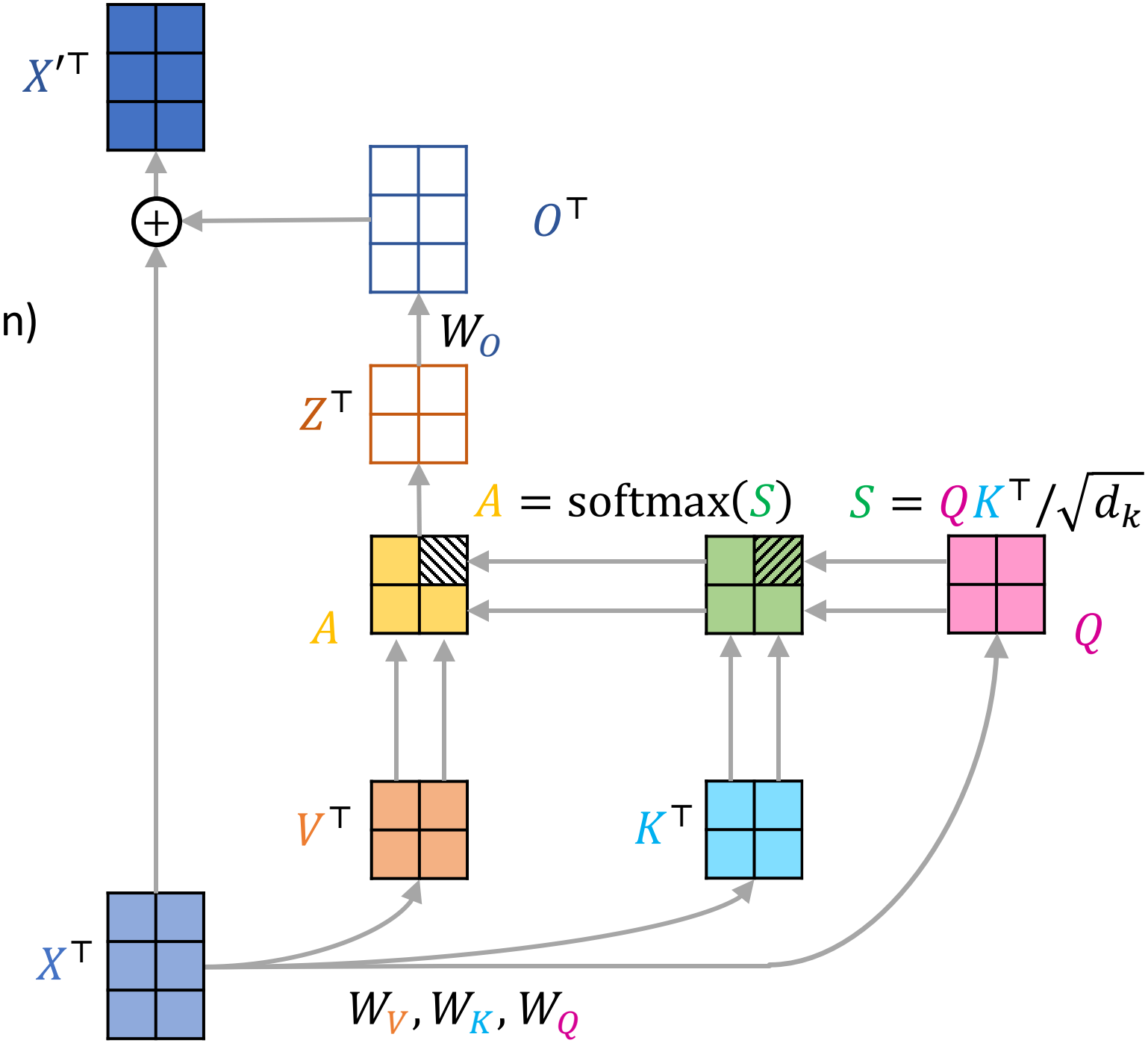
(Repeated without red arrows showing attention combination)



# Causal Attention

## Inference time

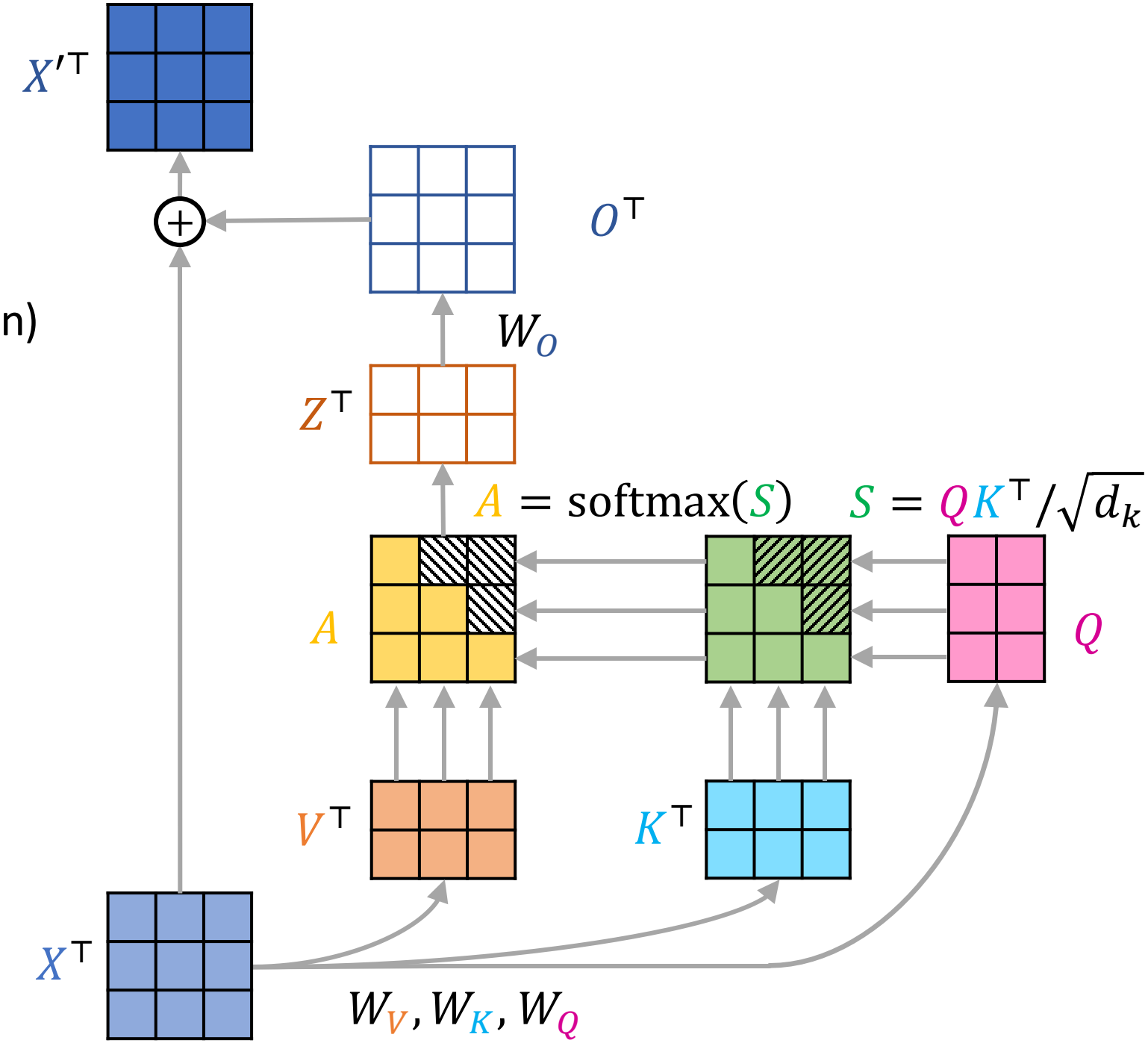
(Repeated without red arrows showing attention combination)



# Causal Attention

## Inference time

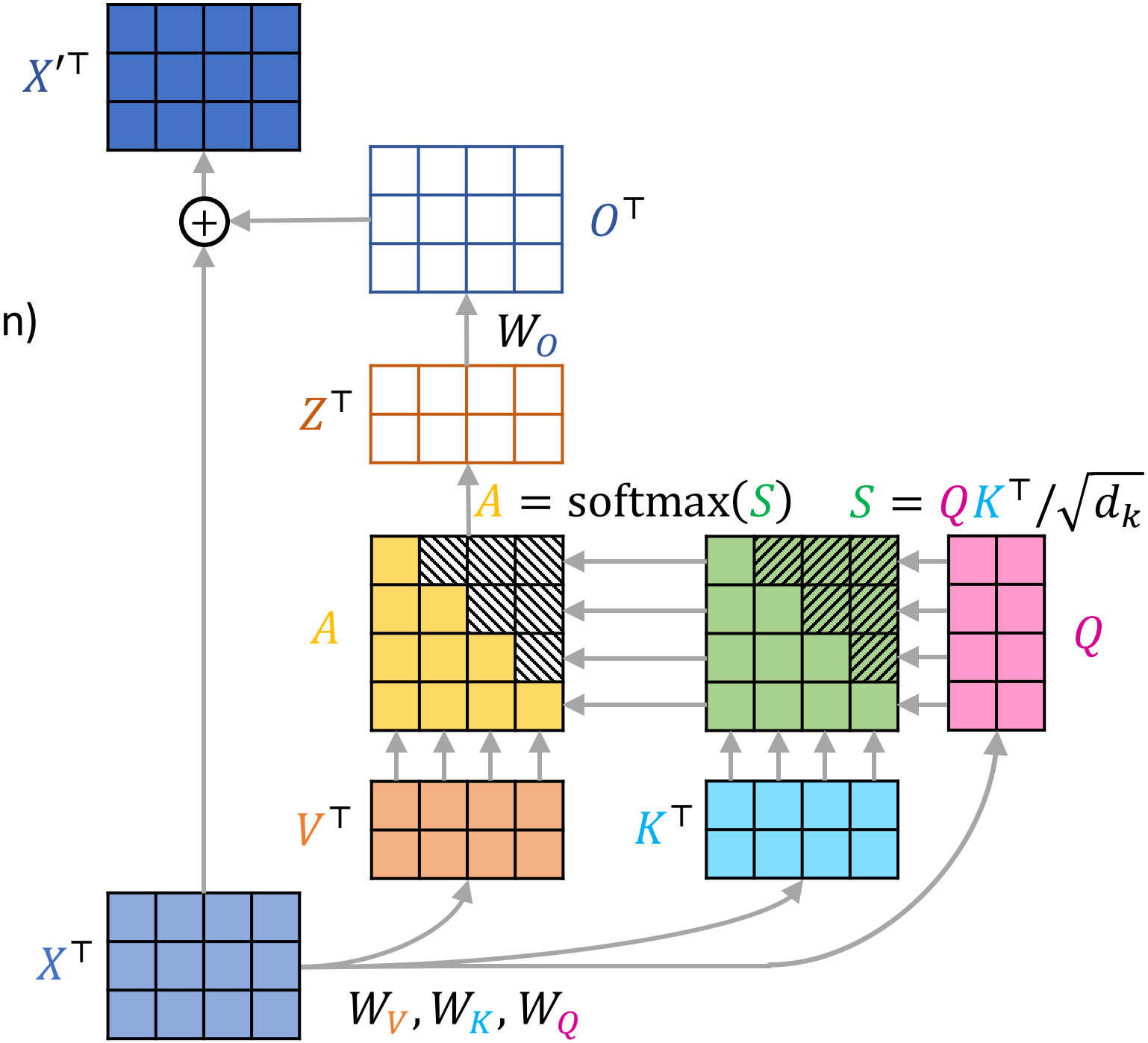
(Repeated without red arrows showing attention combination)



# Causal Attention

## Inference time

(Repeated without red arrows showing attention combination)

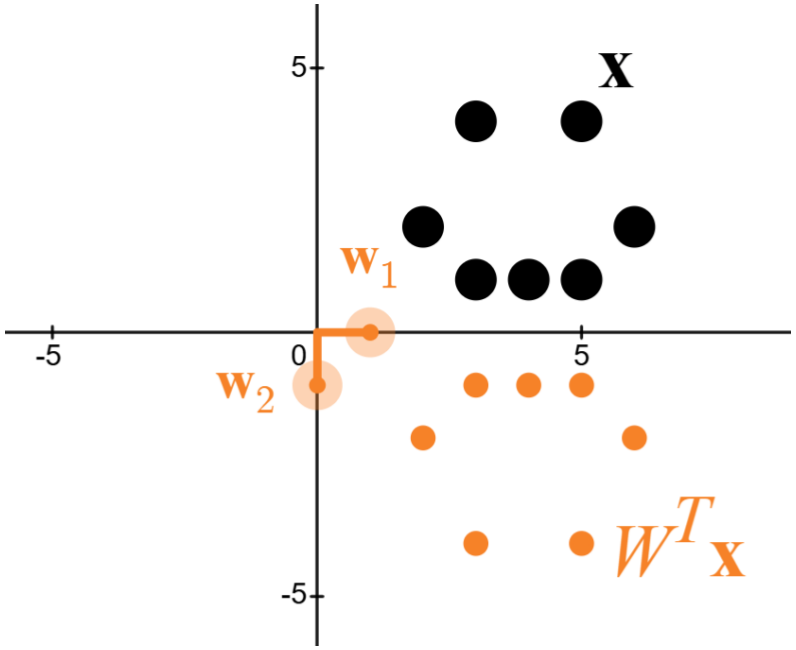


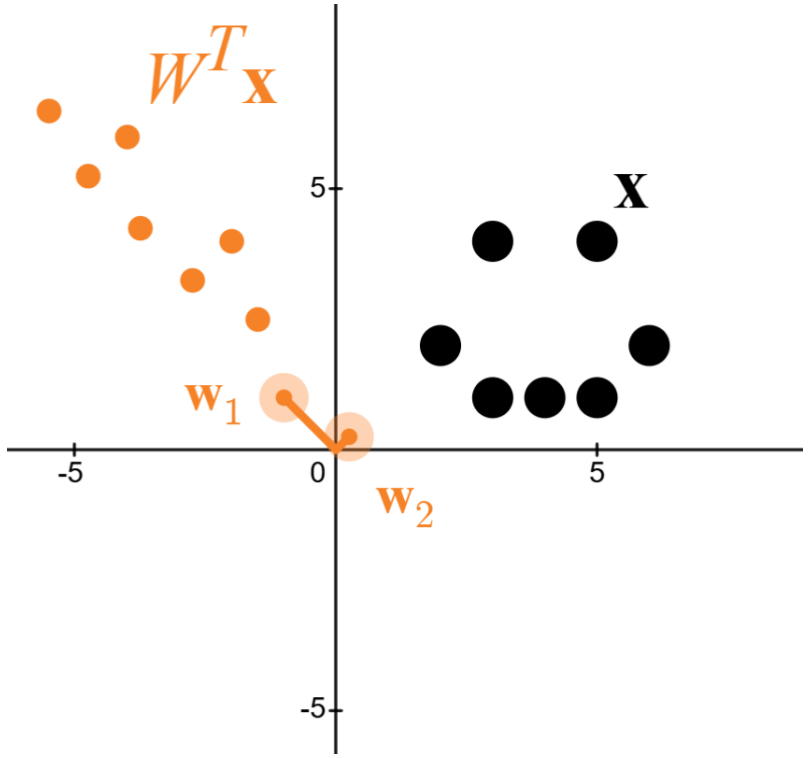
# Linear Transforms: Graphical Intuition

In Transformer models, we see quite a few linear transforms

A simple $\mathbf{z} = W^{\top}\mathbf{x}$ can move points quite a bit

Desmos example for $\boldsymbol{x}$ and $\mathbf{z}$ in $\mathbb{R}^2$
https://www.desmos.com/calculator/gl5ljvorcy

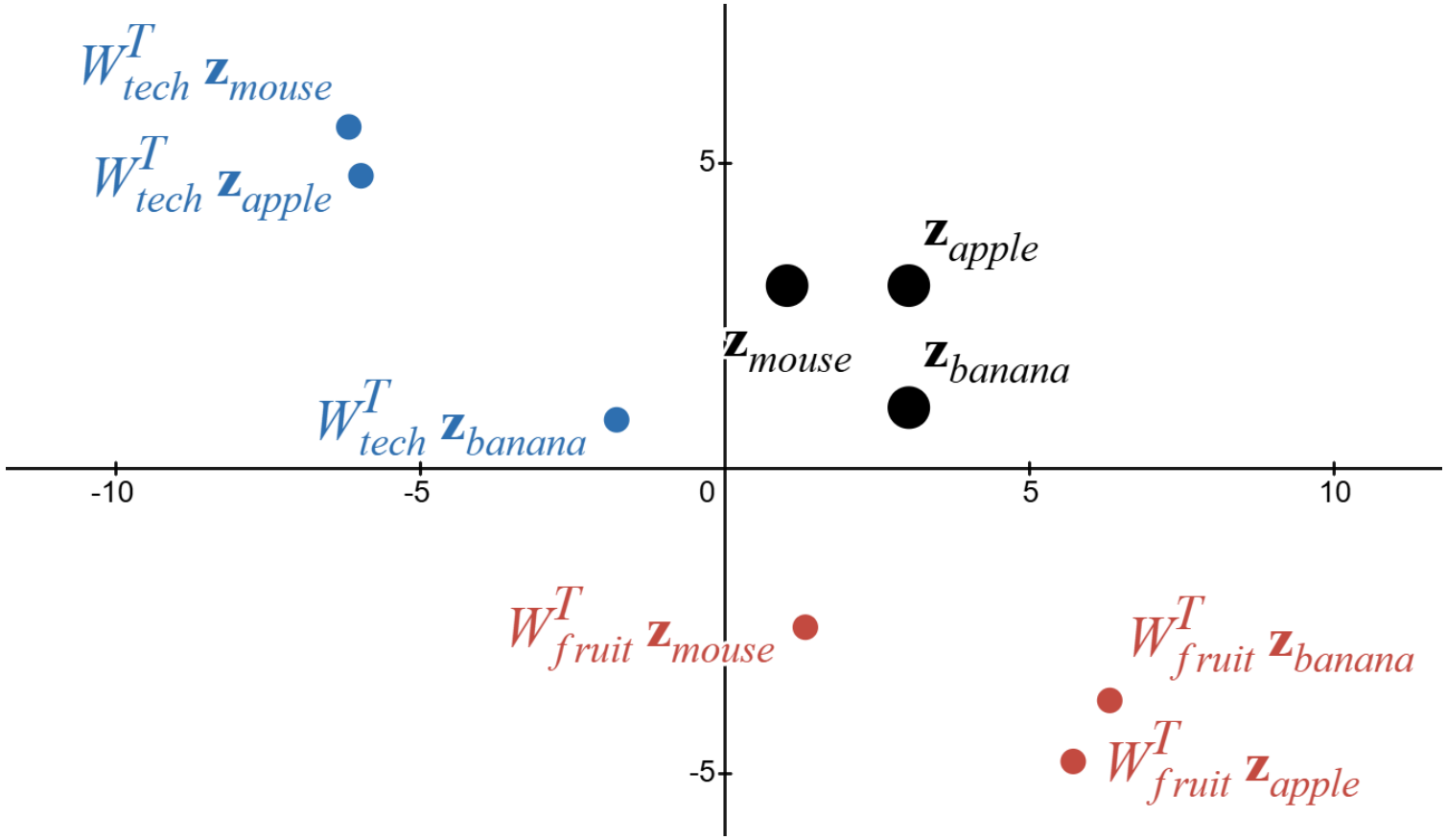


# Linear Transforms: Graphical Intuition

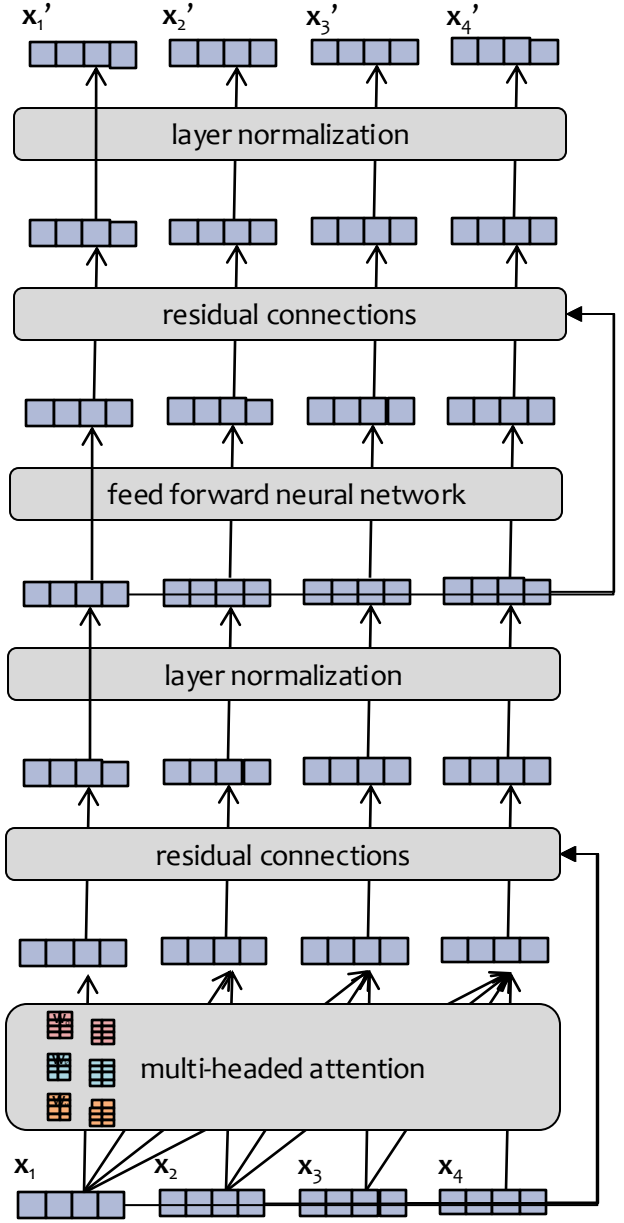
Two different transforms $W_{tech}^{\top}\mathbf{z}$ and $W_{fruit}^{\top}\mathbf{z}$ can create two different meaningful embeddings for the input vectors $\mathbf{z}$

Desmos example for $W$ in $\mathbb{R}^{2\times 2}$ https://www.desmos.com/calculator/tbeclbo83h



# Transformer Layer

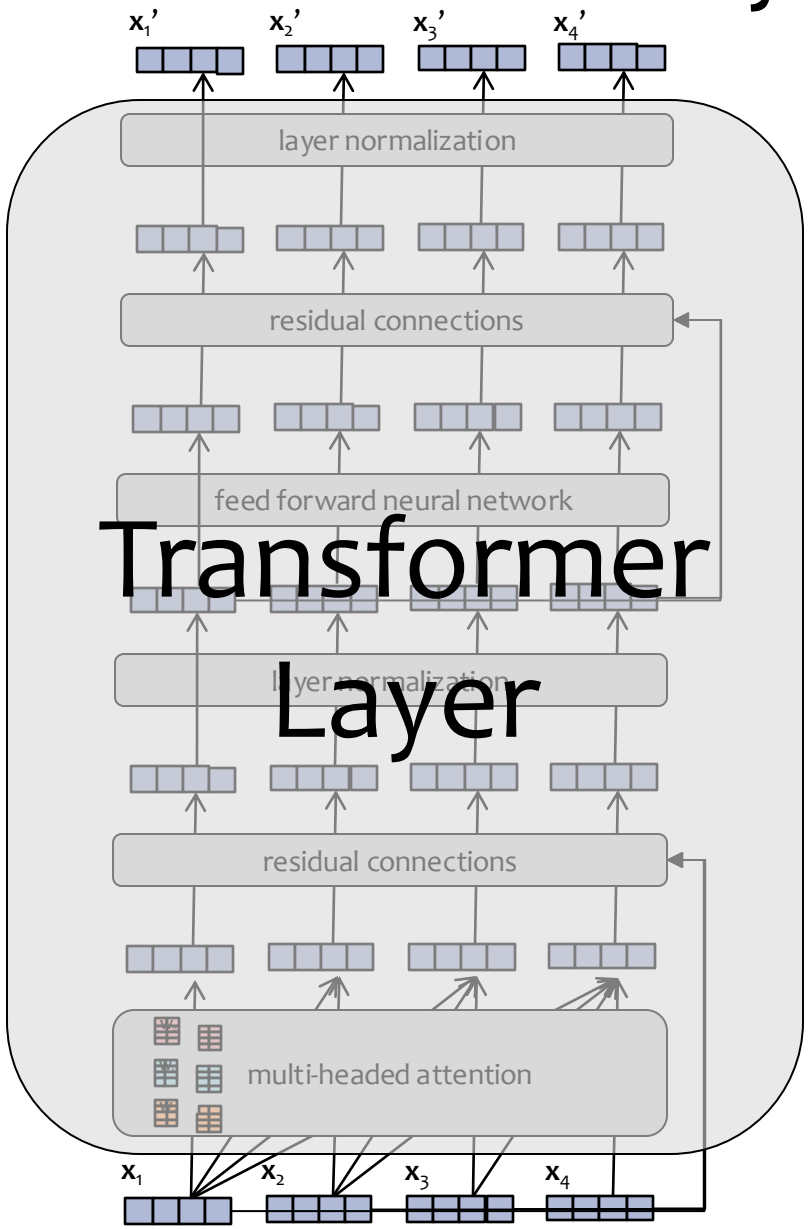
# Transformer Layer




**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

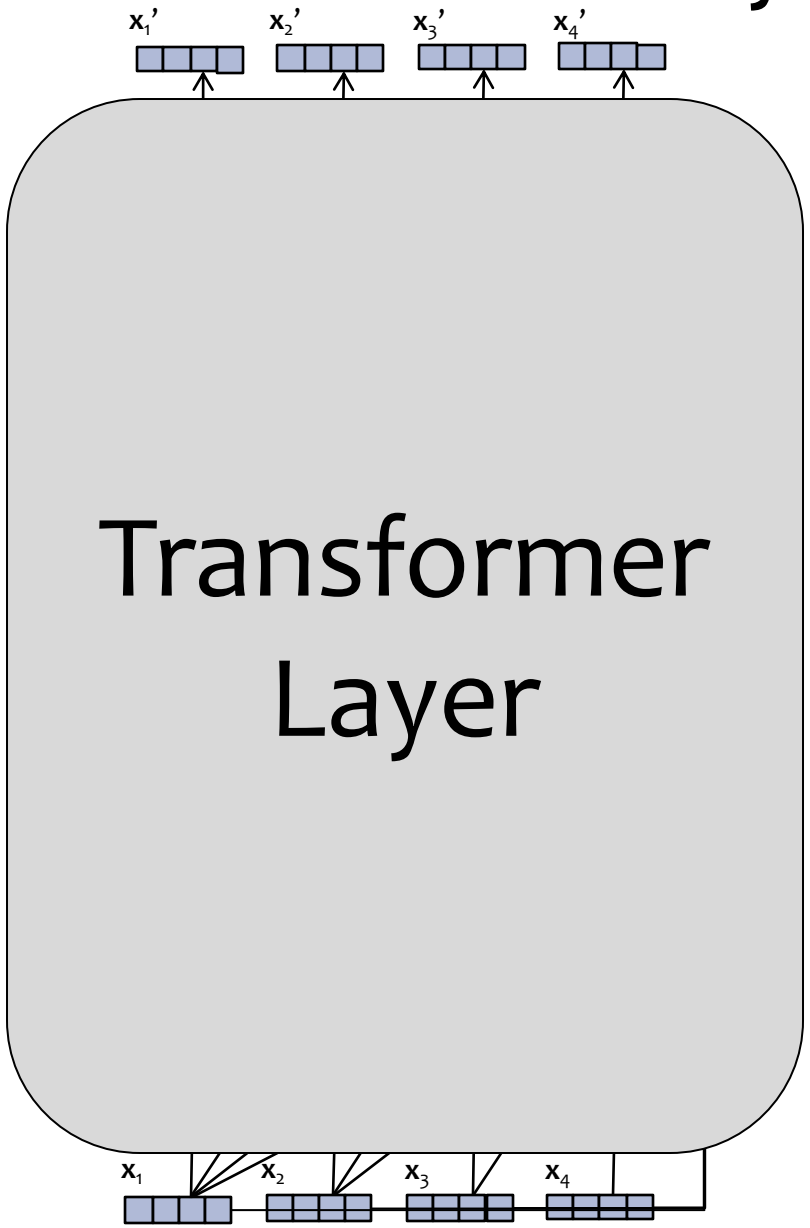
# Transformer Layer




**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

# Transformer Layer



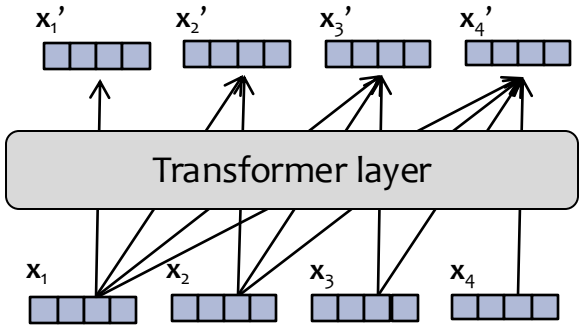

**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

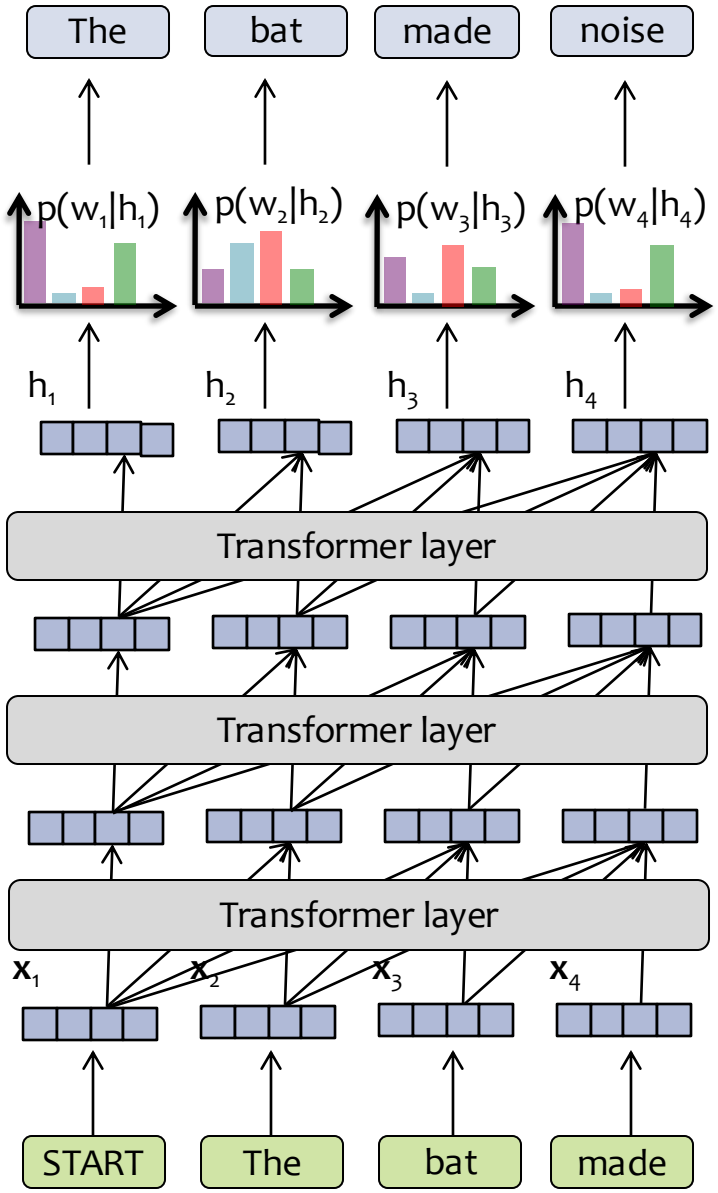
# Transformer Layer

**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections



# Transformer Language Model




...

**Each layer** of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM!

# More Transformers

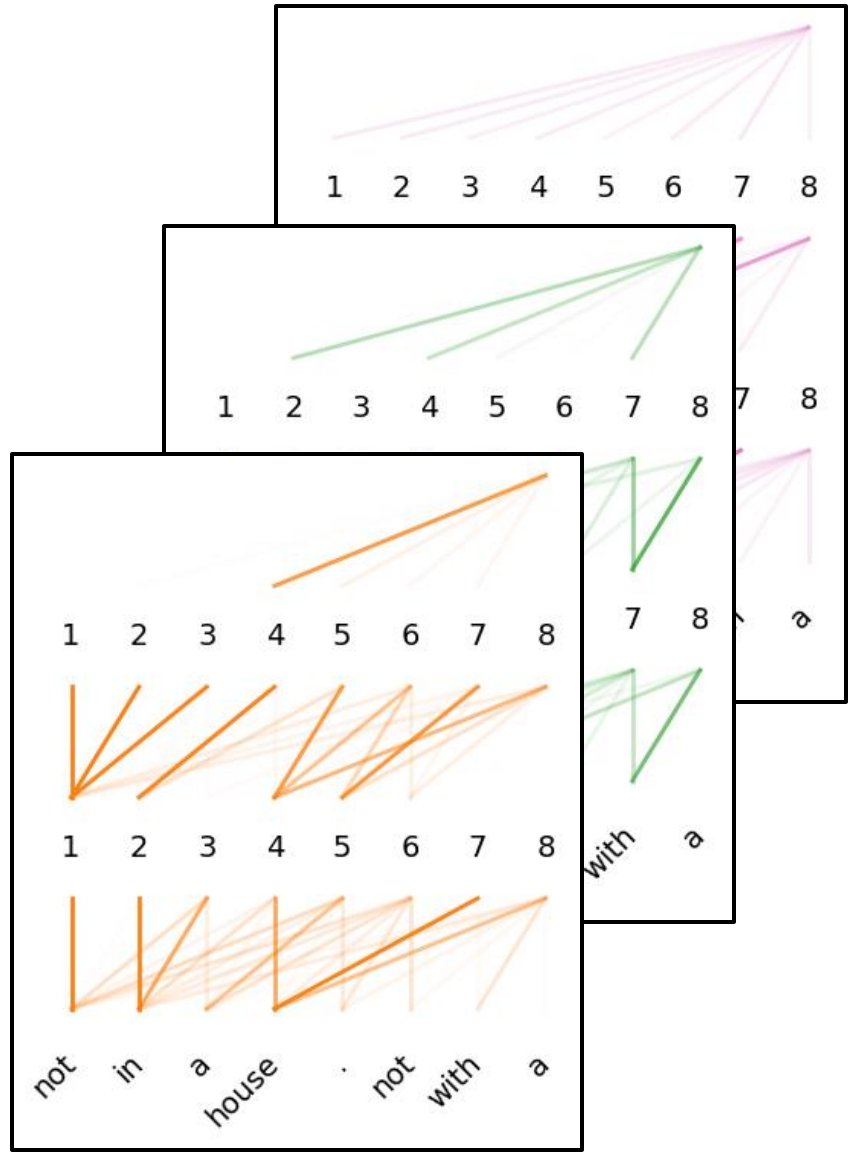
# More Transformers

## Multihead Attention

## Different types of attention

- Causal self attention
- Self attention
- Cross attention

## Vision Transformers

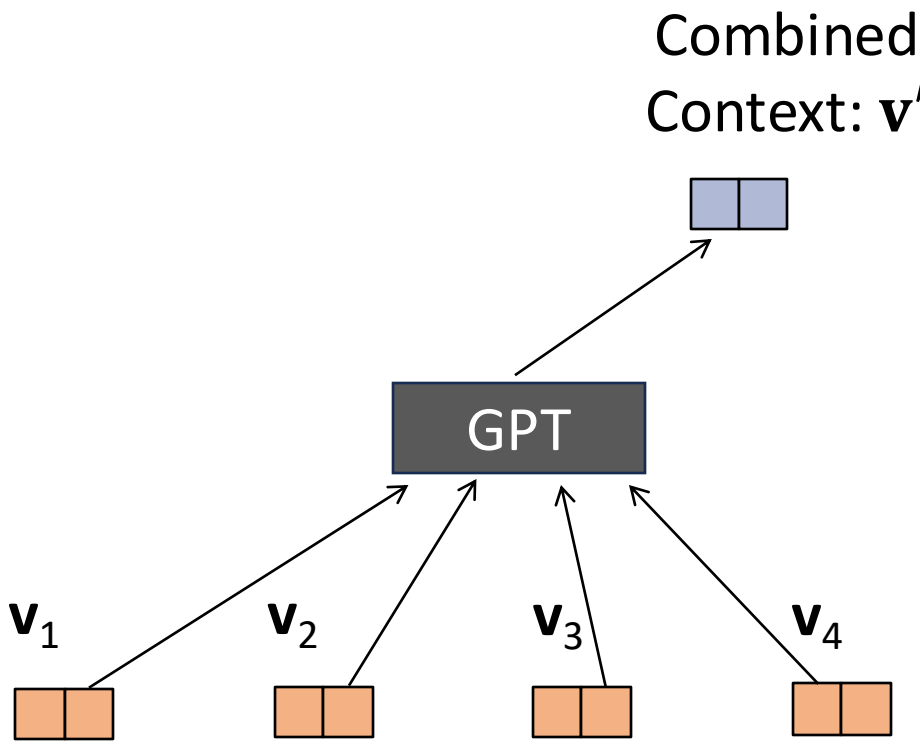


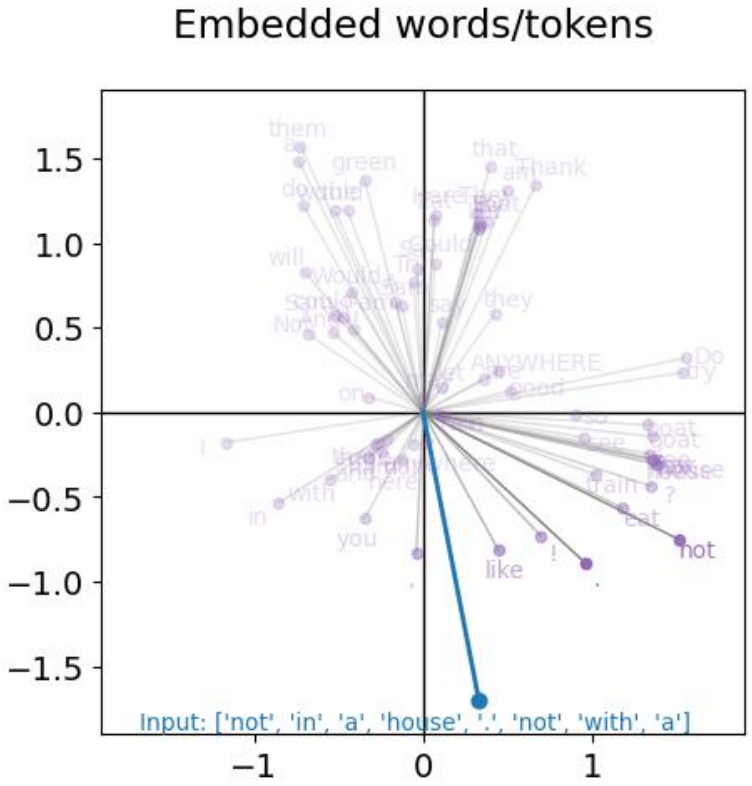
# MinGPT Femto

2-D embedded space

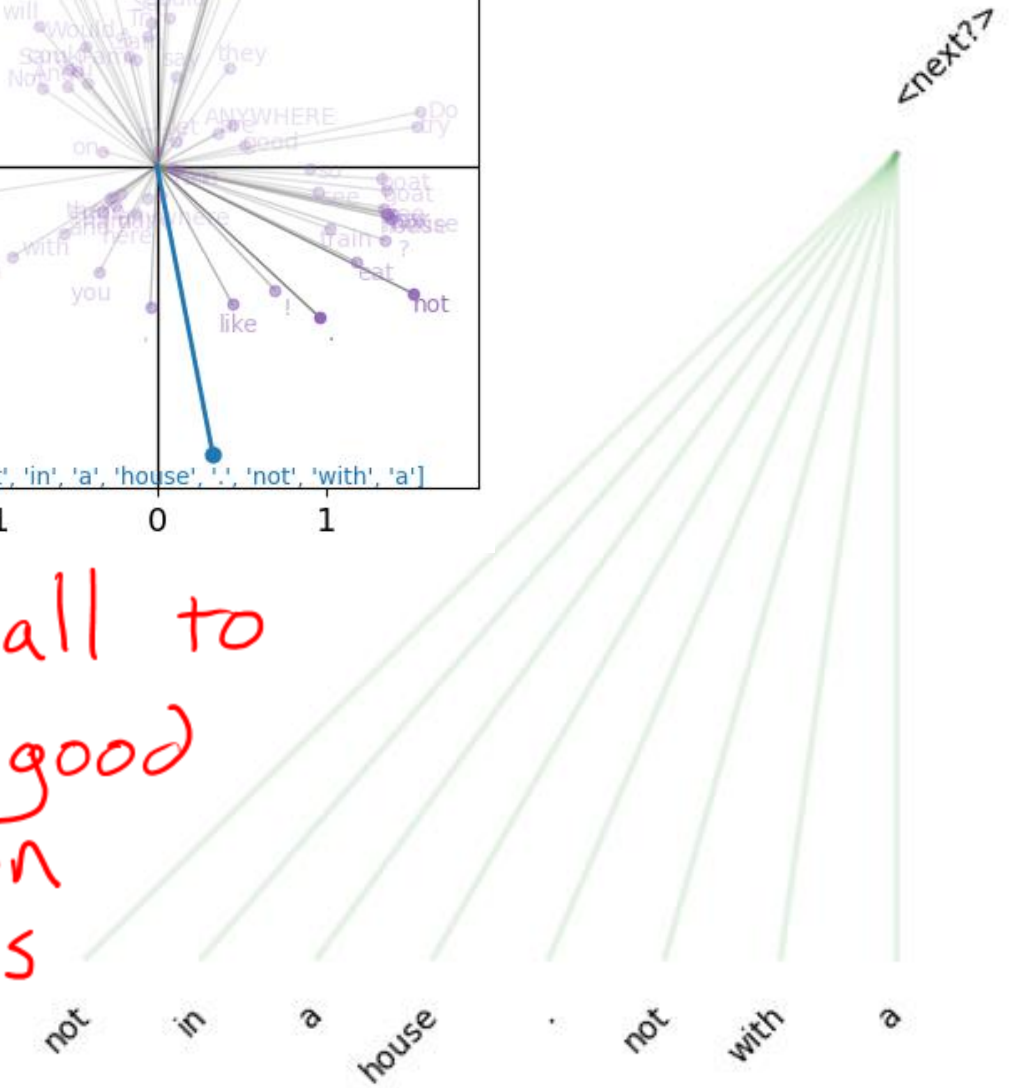
1 attention layer

1 attention head
(think channel)




Too small to learn good attention weights

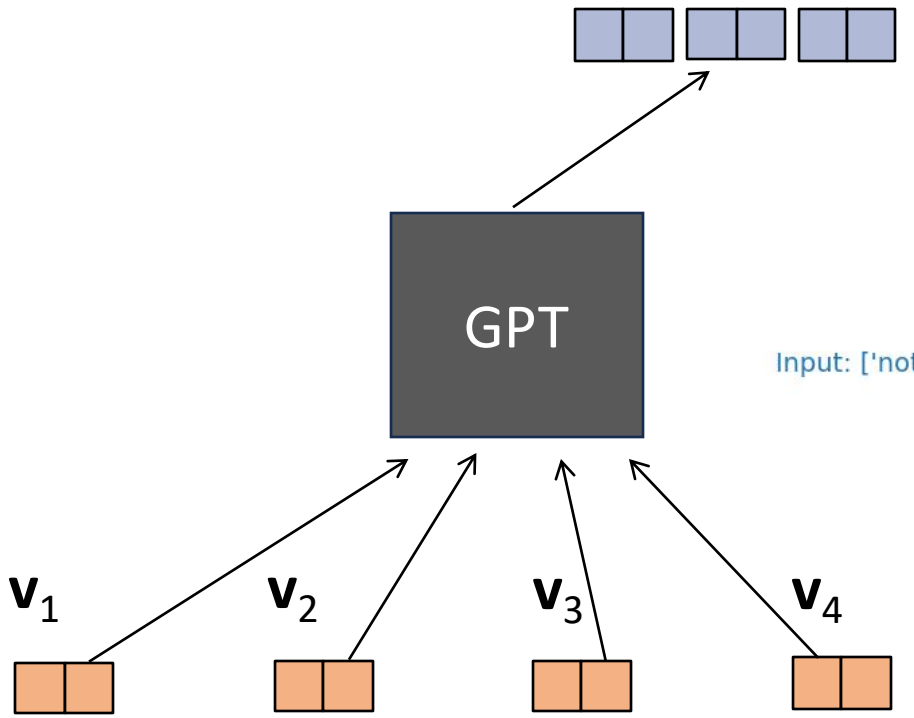


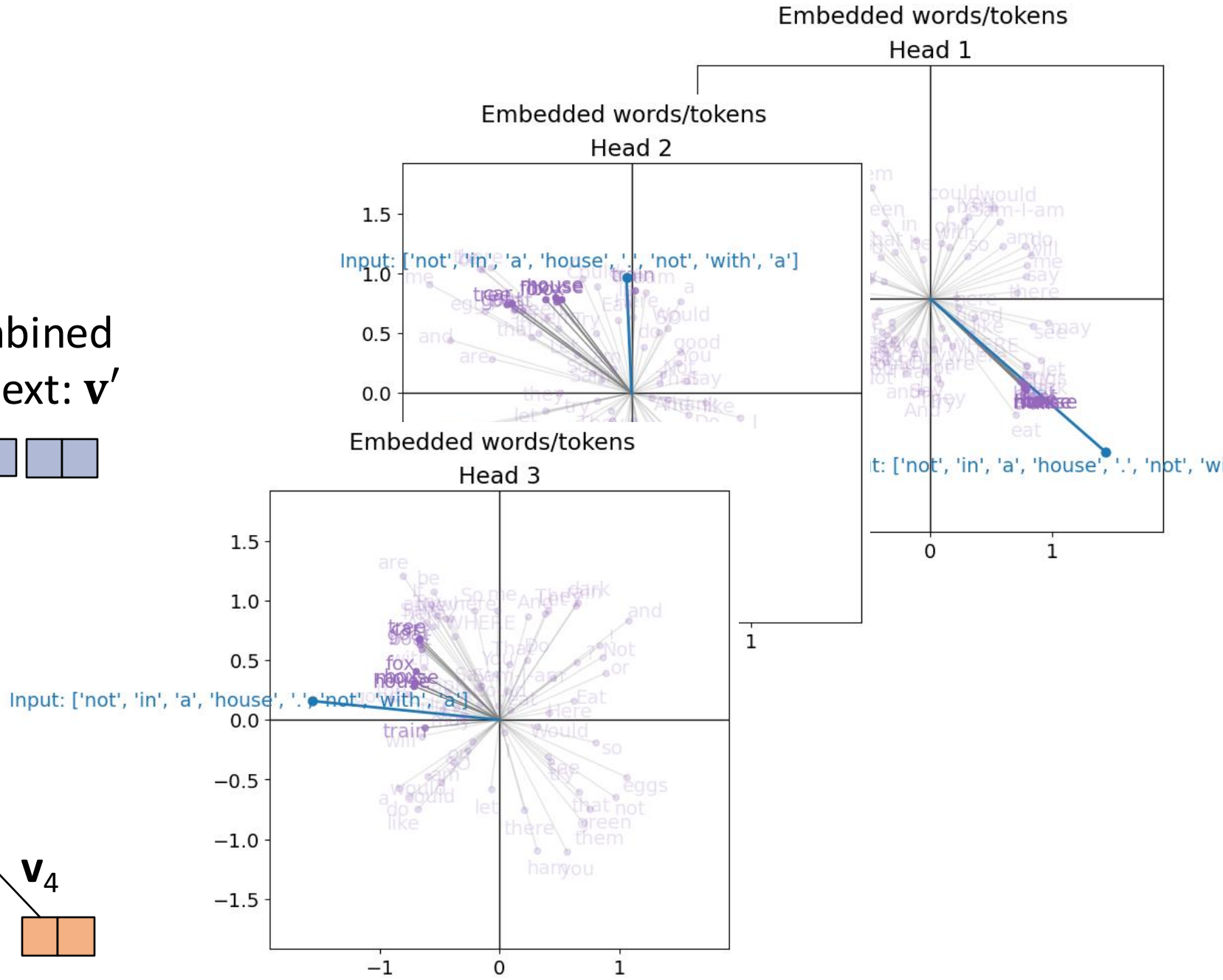
# MinGPT Pico

2-D embedded space

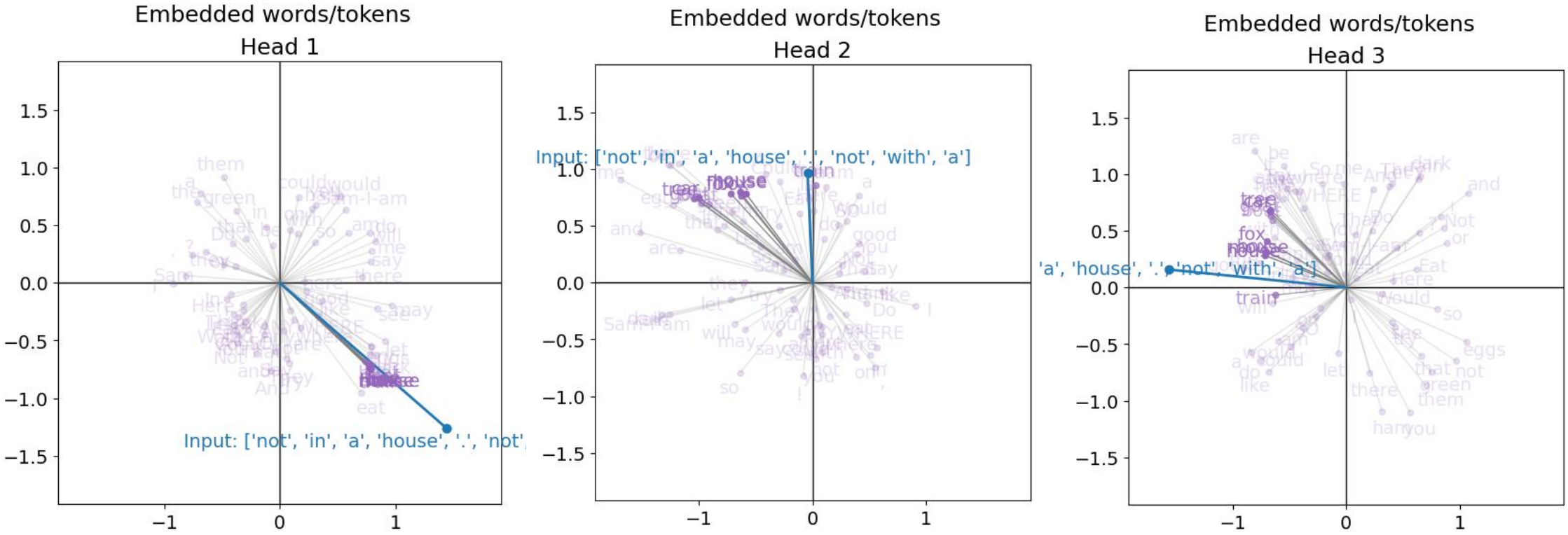
3 attention layer

3 attention heads (think channels)



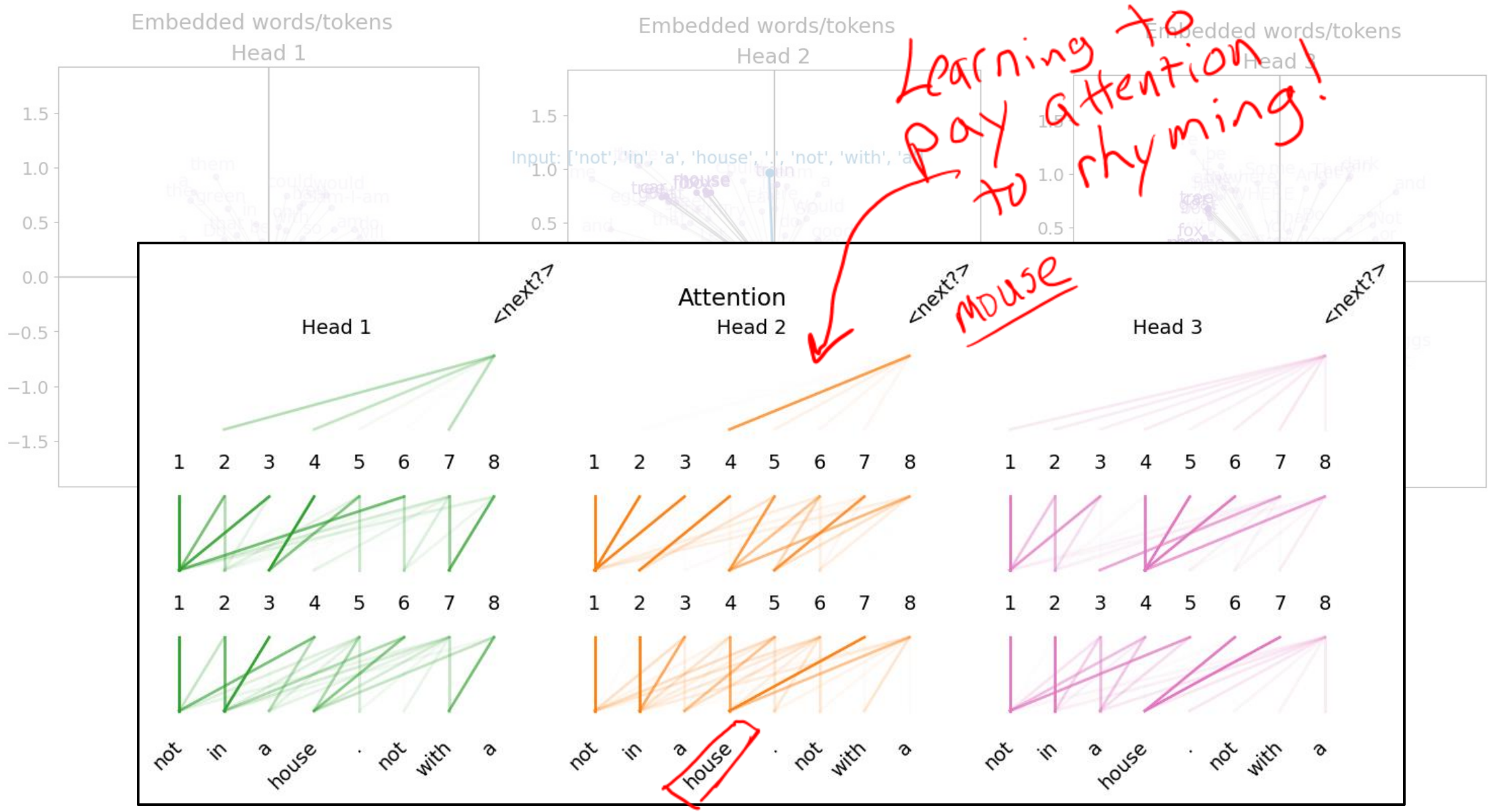



# MinGPT Pico: Output embedded space - 3 heads



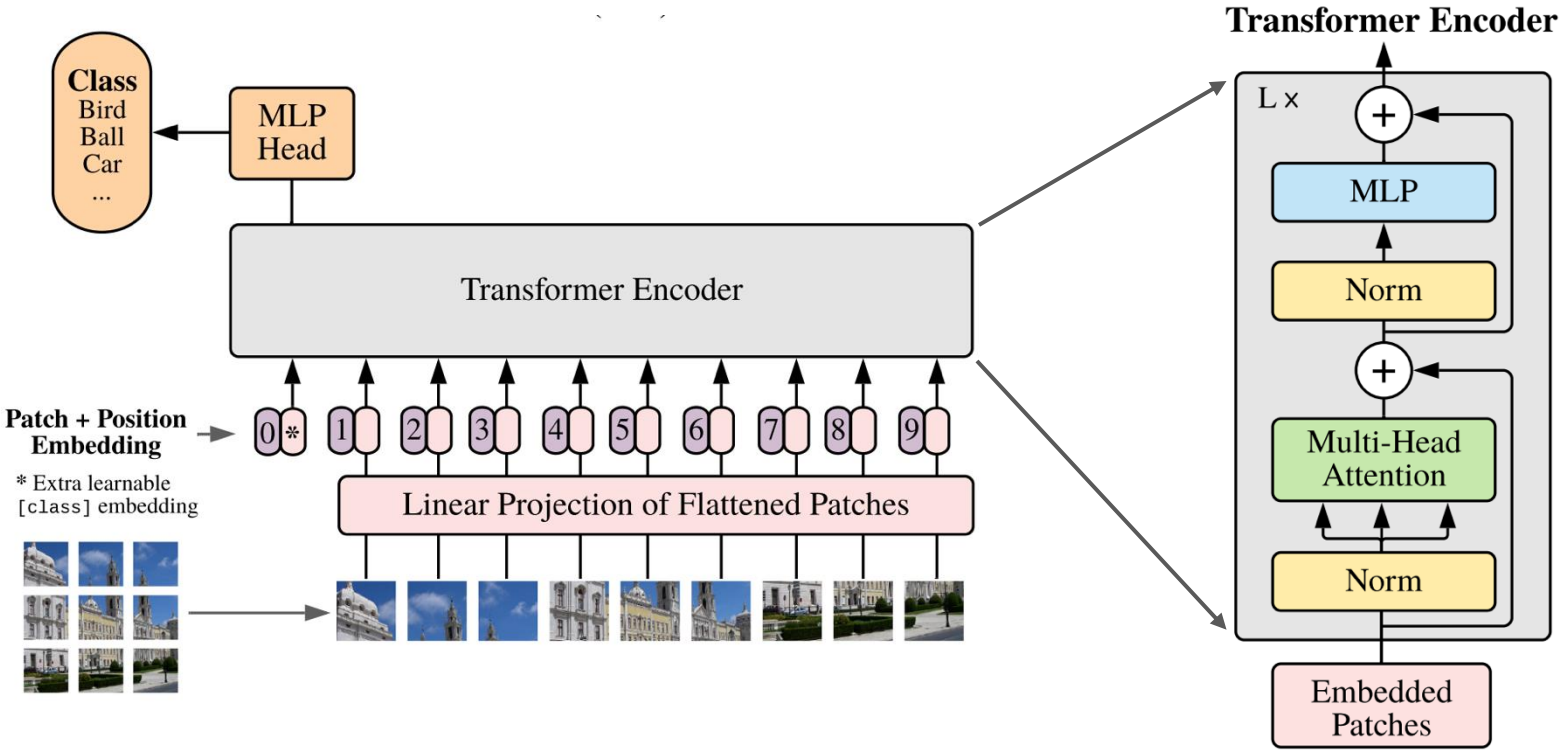

Three heads allows more room to learn diffent feature representations

# MinGPT Pico: Attention Weights – 3 layers, 3 heads



# Vision Transformers

# Vision Transformer (ViT)




- Instead of words as input, the inputs are $P \times P$ pixel *patches*
- Each patch is embedded linearly into a vector of size 1024
- Uses 1D positional embeddings
- Pre-trained on a large, supervised dataset (e.g., ImageNet 21K, JFT-300M)

Source: https://arxiv.org/pdf/2010.11929

# IN-CONTEXT LEARNING

# Few-shot Learning

- ***Definition***: in **few-shot learning** we assume that training data contains a handful (maybe two, three, or four) examples of each label

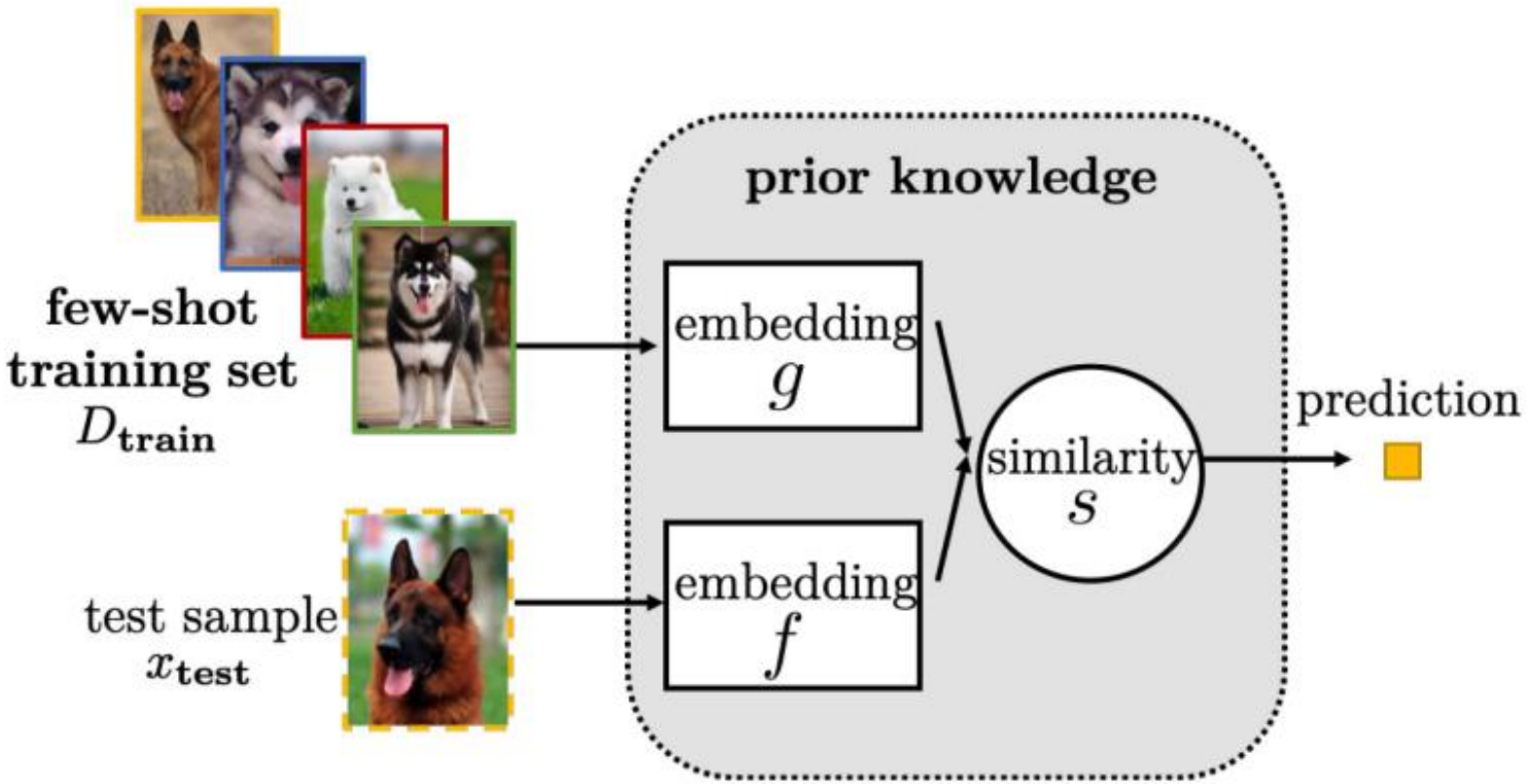



Figure from https://dl.acm.org/doi/10.1145/3386252

# Few-shot Learning with LLMs

Suppose you have…

- a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ and N is rather small (i.e. few-shot setting)
- a very large (billions of parameters) pre-trained language model

There are two ways to "learn"

This section!

**Option A: Supervised fine-tuning**

- **Definition:** fine-tune the LLM on the training data using…
  - a standard supervised objective
  - backpropagation to compute gradients
  - your favorite optimizer (e.g. Adam)
- **Pro:** fits into the standard ML recipe
- **Pro:** still works if N is large
- **Con:** backpropagation requires ~3x the memory and computation time as the forward computation
- **Con:** you might not have access to the model weights at all (e.g. because the model is proprietary)

**Option B: In-context learning**

- **Definition:**
  1. feed training examples to the LLM as a prompt
  2. allow the LLM to infer patterns in the training examples during inference (i.e. decoding)
  3. take the output of the LLM following the prompt as its prediction
- **Con:** the prompt may be very long and Transformer LMs require $O(N^2)$ time/space where N = length of context
- **Pro:** no backpropagation required and only one pass through the training data
- **Pro:** does not require model weights, only API access

# Few-shot In-context Learning

- Few-shot learning can be done via in-context learning
- Typically, a task description is presented first
- Then a sequence of input/output pairs from a training dataset are presented in sequence

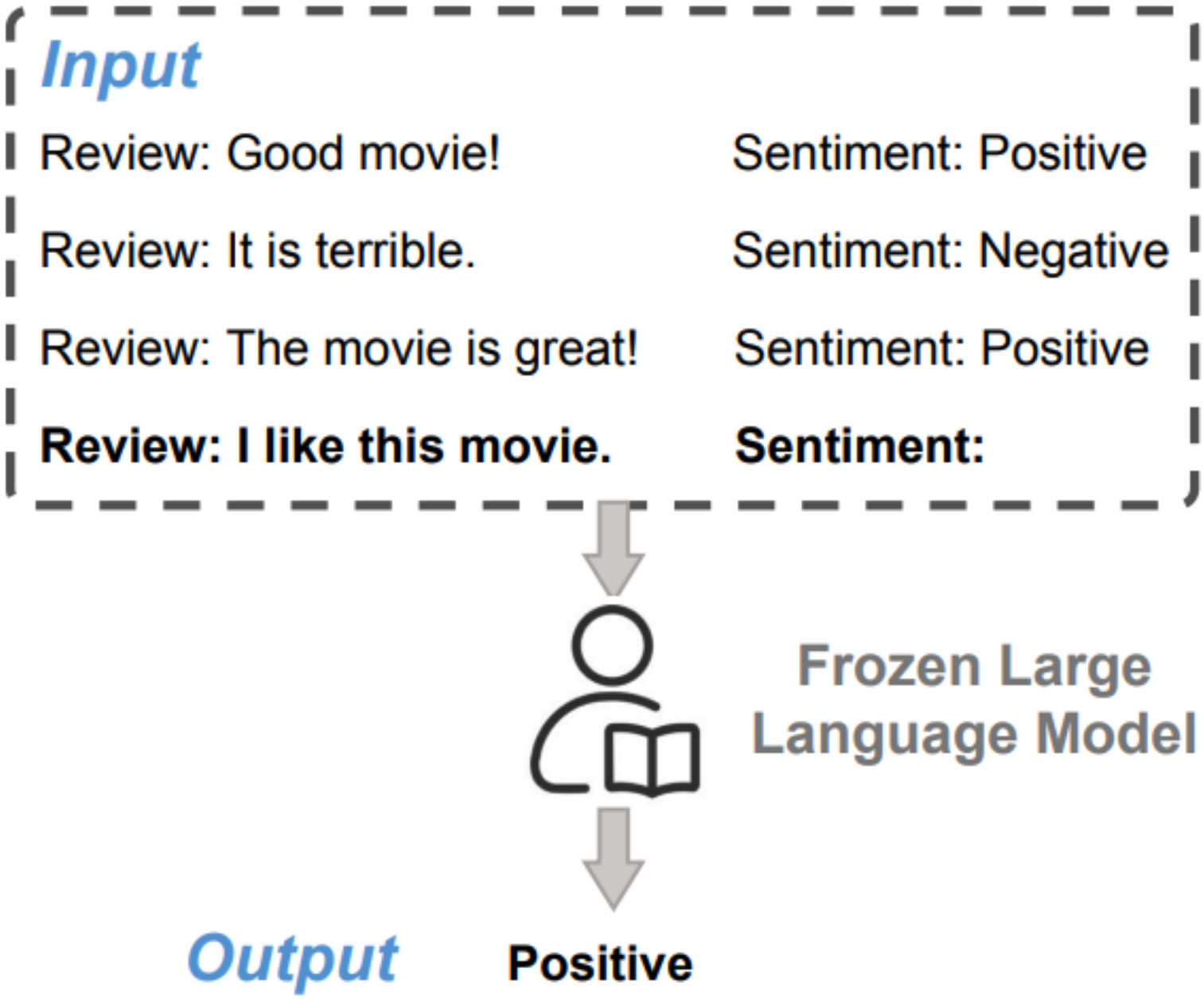



Figure from https://arxiv.org/pdf/2310.09881.pdf

# Few-shot In-context Learning

- Few-shot learning can be done via in-context learning
- Typically, a task description is presented first
- Then a sequence of input/output pairs from a training dataset are presented in sequence

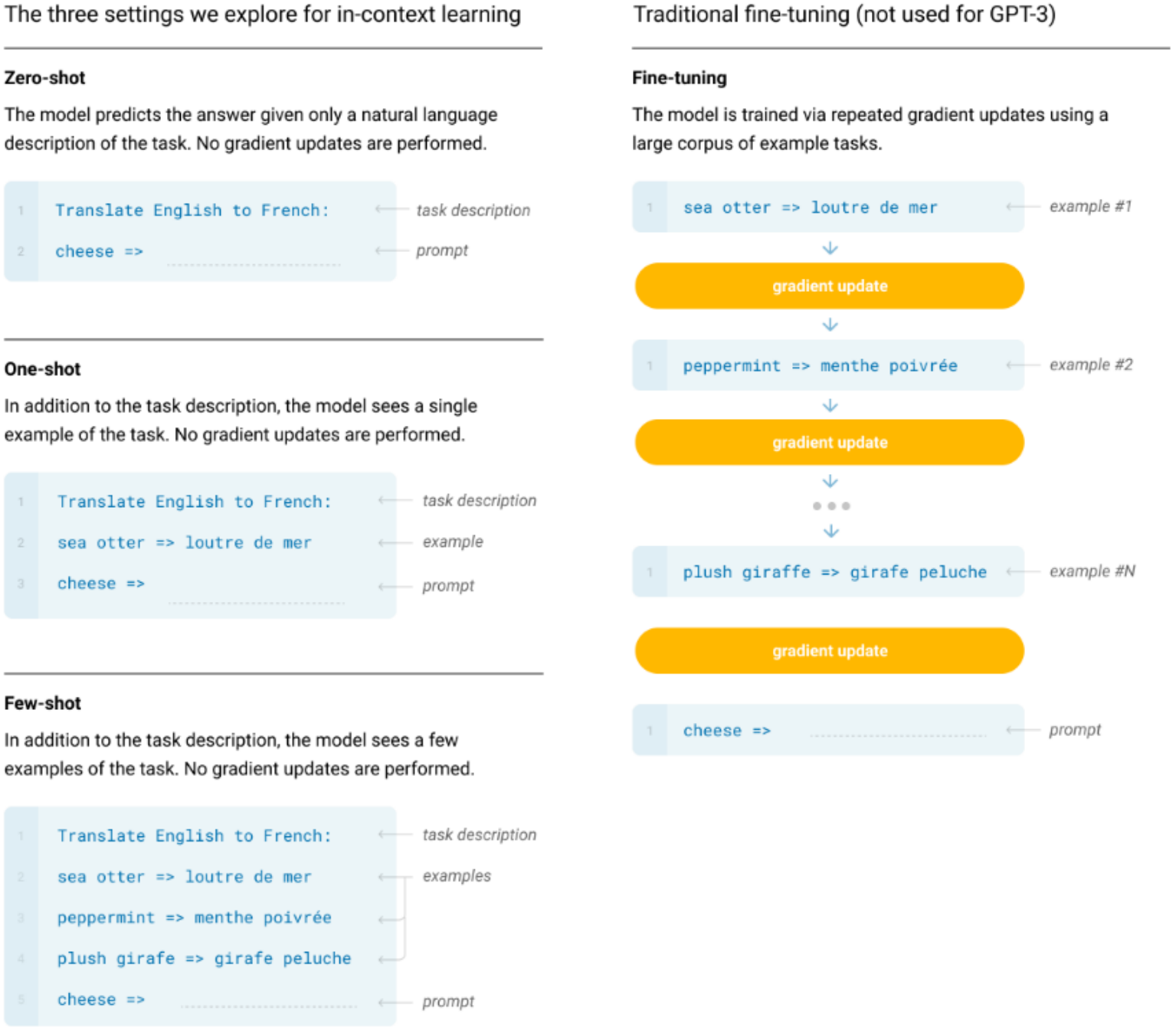



Figure from http://arxiv.org/abs/2005.14165