Part II

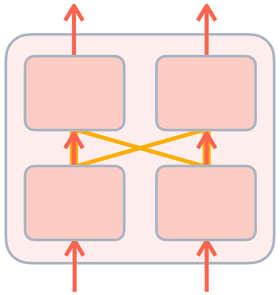
Advanced Transformers

Part I

Recurrent Networks, Seq2Seq and Attention

Part II

Transformers



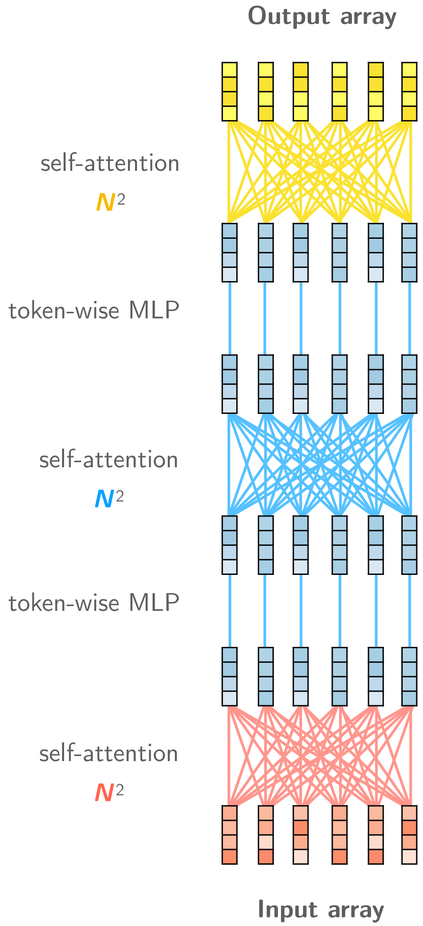
Part III

State Space Models

???

Transformer

as a kind of “token net” *



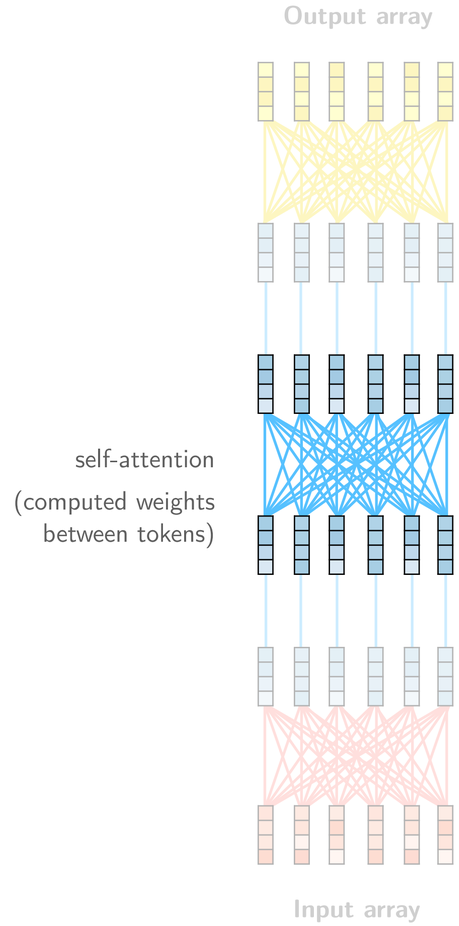
*using the word “token” a bit loosely

Transformer

as a kind of “token net”

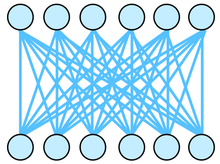
Self-attention is how we “wire up” the tokens.
The weights between neurons are learned,
but the weights between tokens are computed
in the forward pass from the tokens’ vectors.

Arguably, self-attention is the most “natural”
way to wire up tokens.



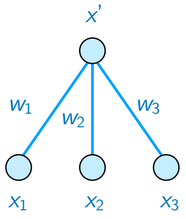
self-attention
(computed weights
between tokens)

vs

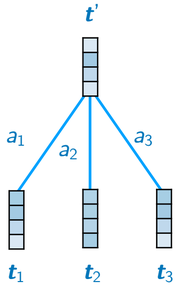


MLP
(learned weights
between neurons)

In both cases, we do a linear combination:



$\sum_i w_i x_i$

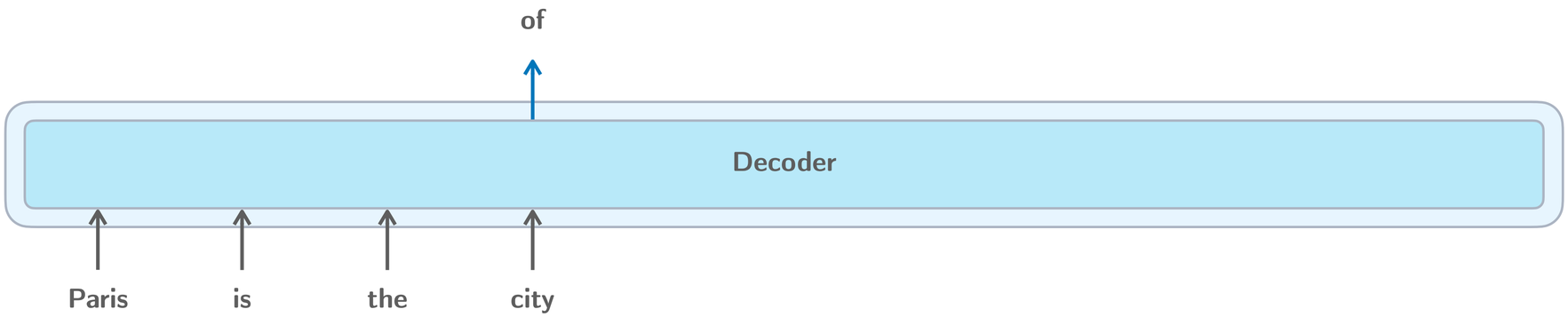


$\sum_i a_i t_i$

Decoding

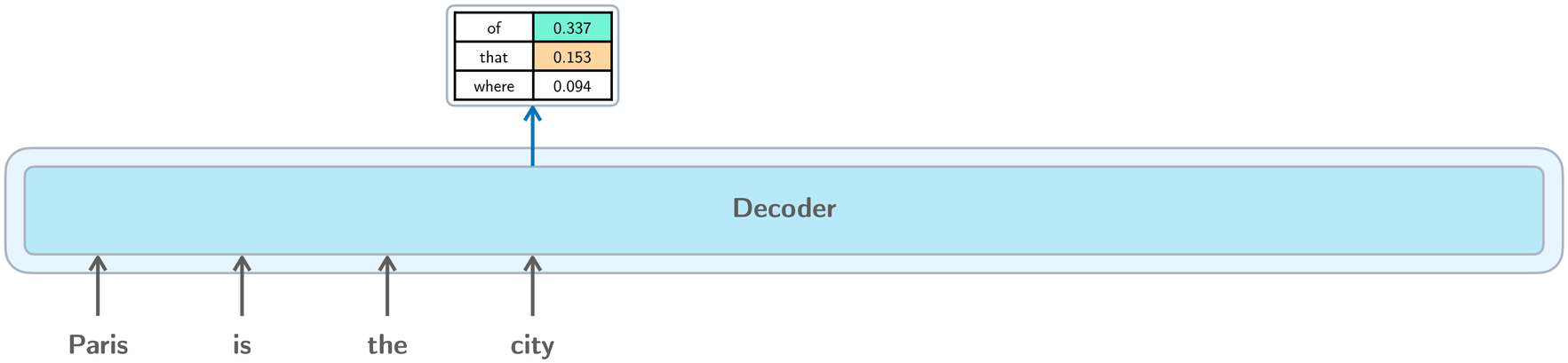
Beam search

Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217



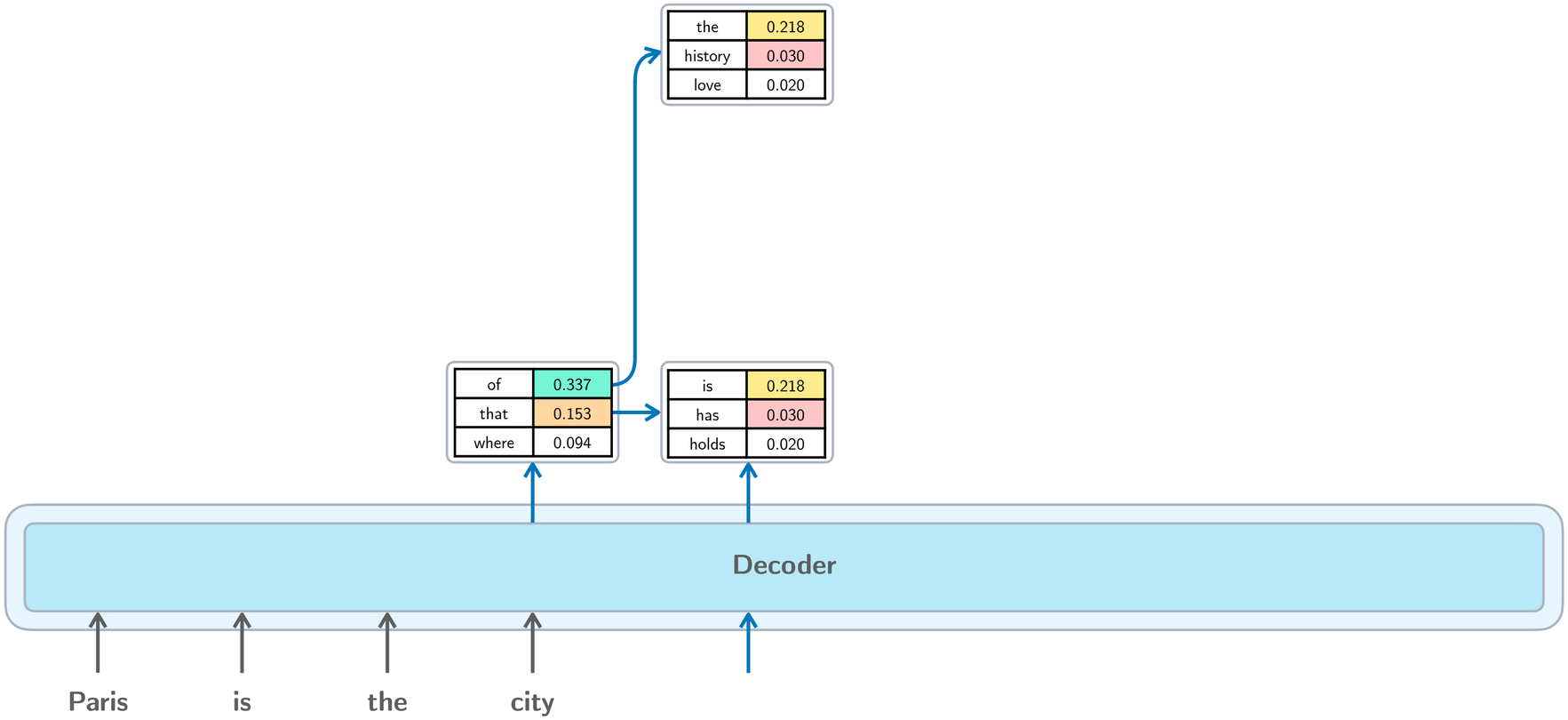
Beam search

Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217



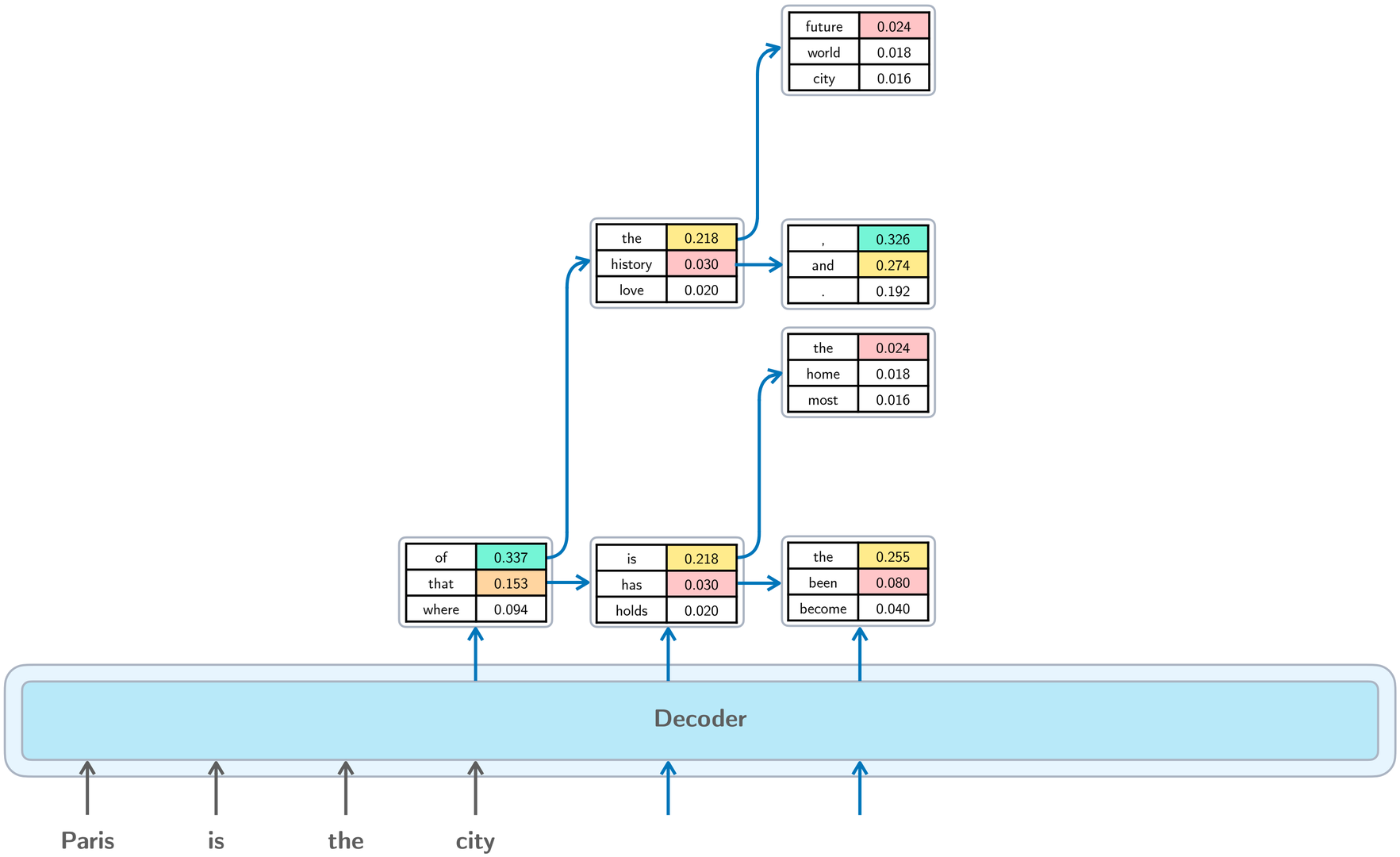
Beam search

Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217



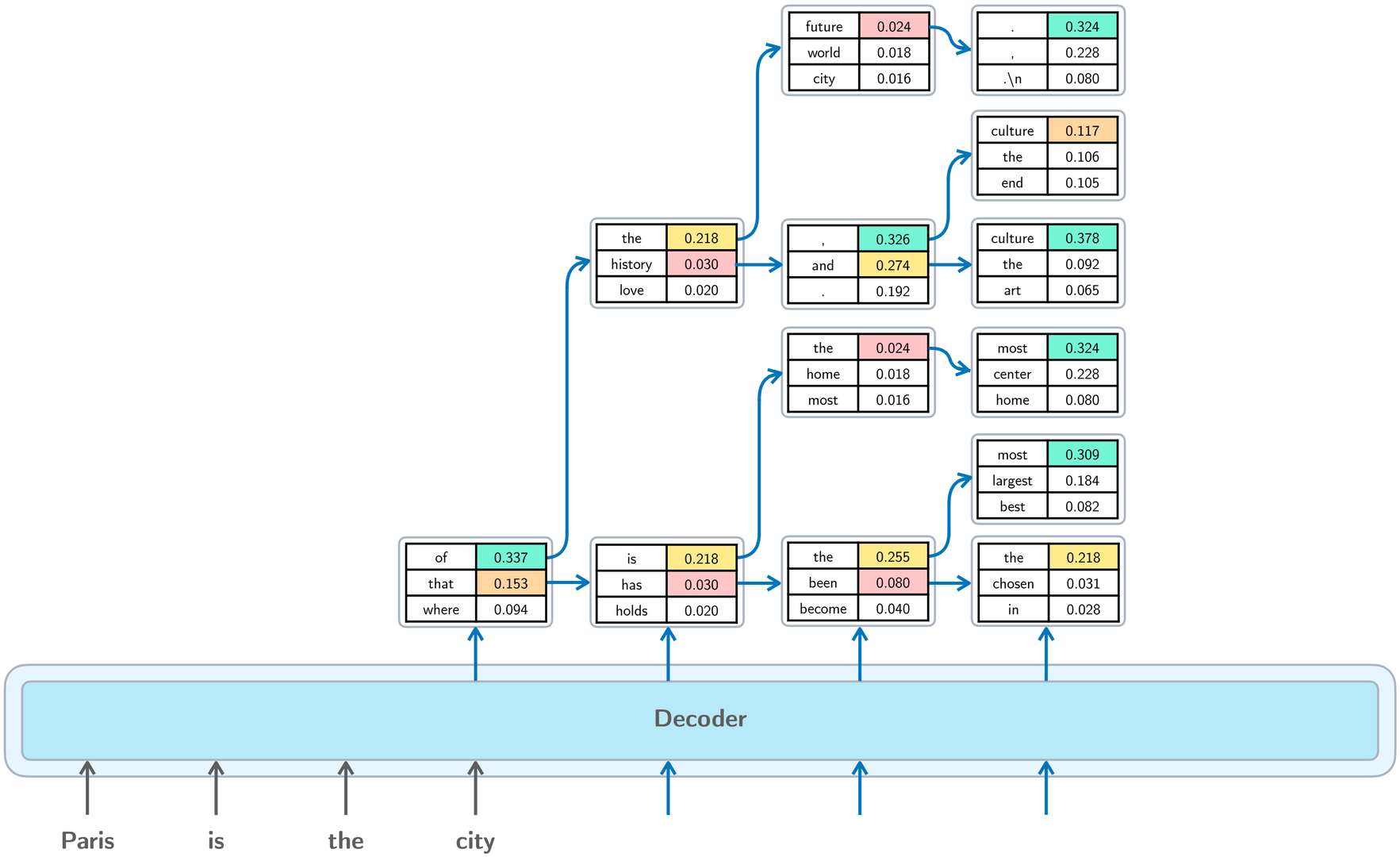
Beam search

Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217



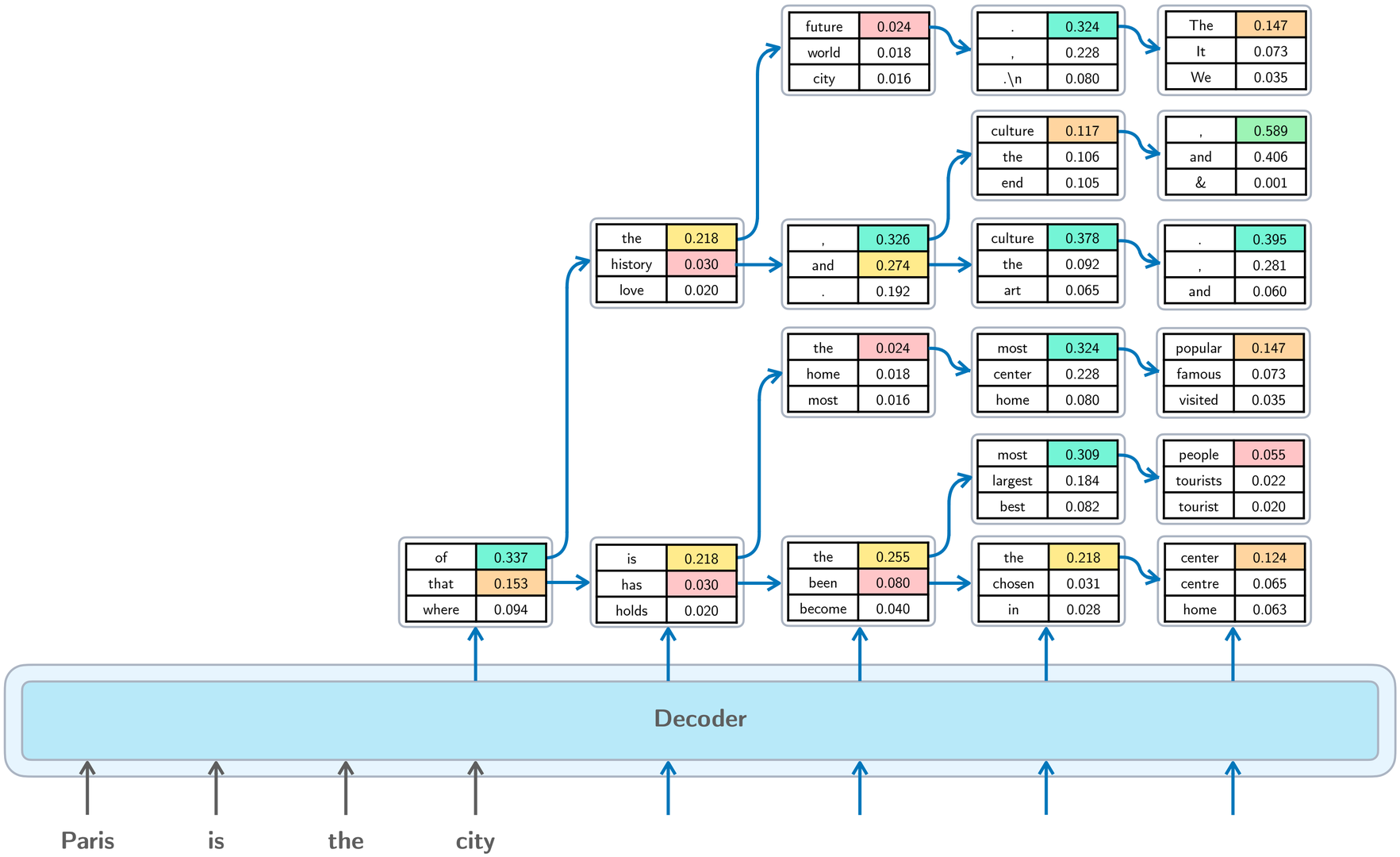
Beam search

Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217



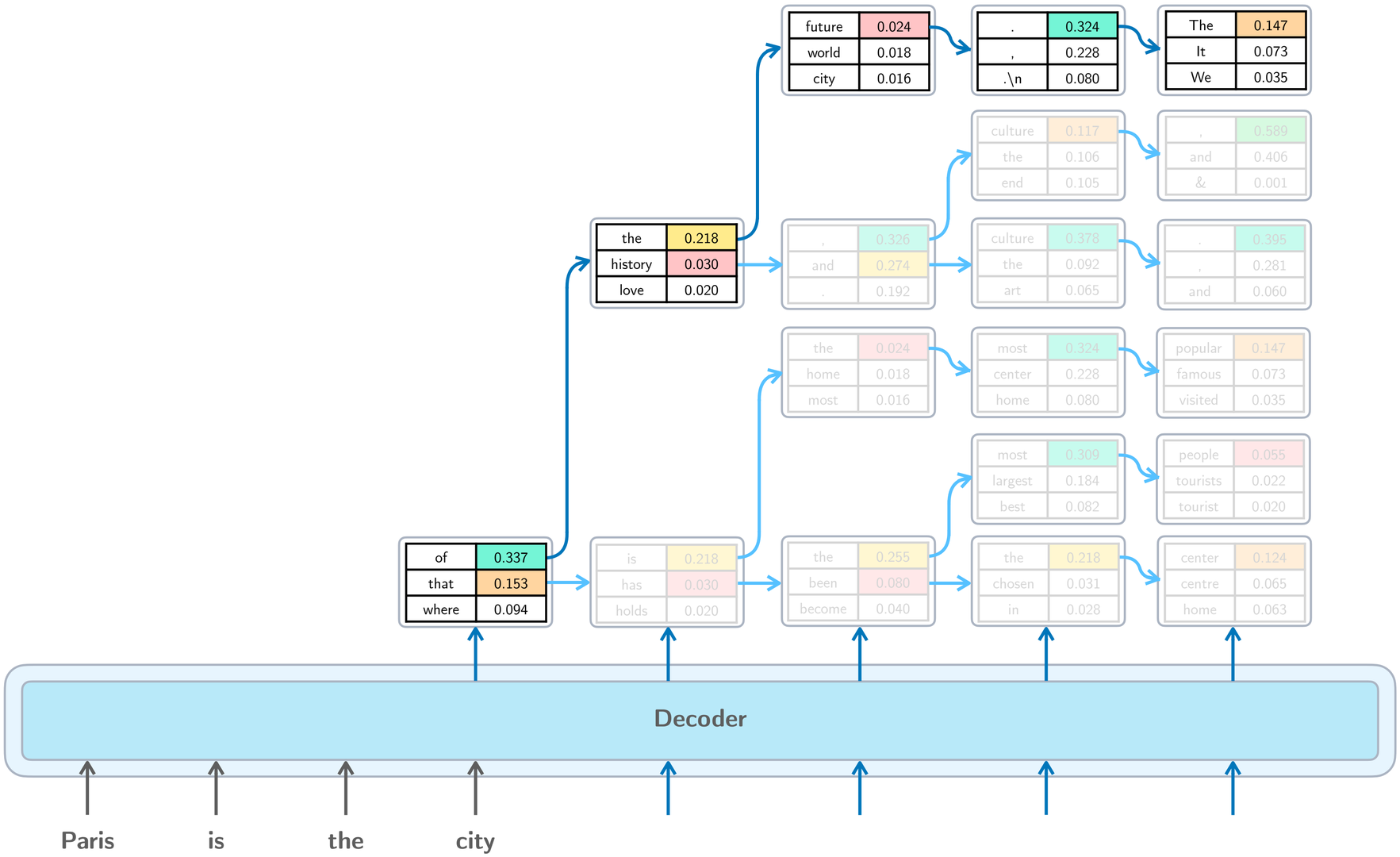
Beam search

Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217



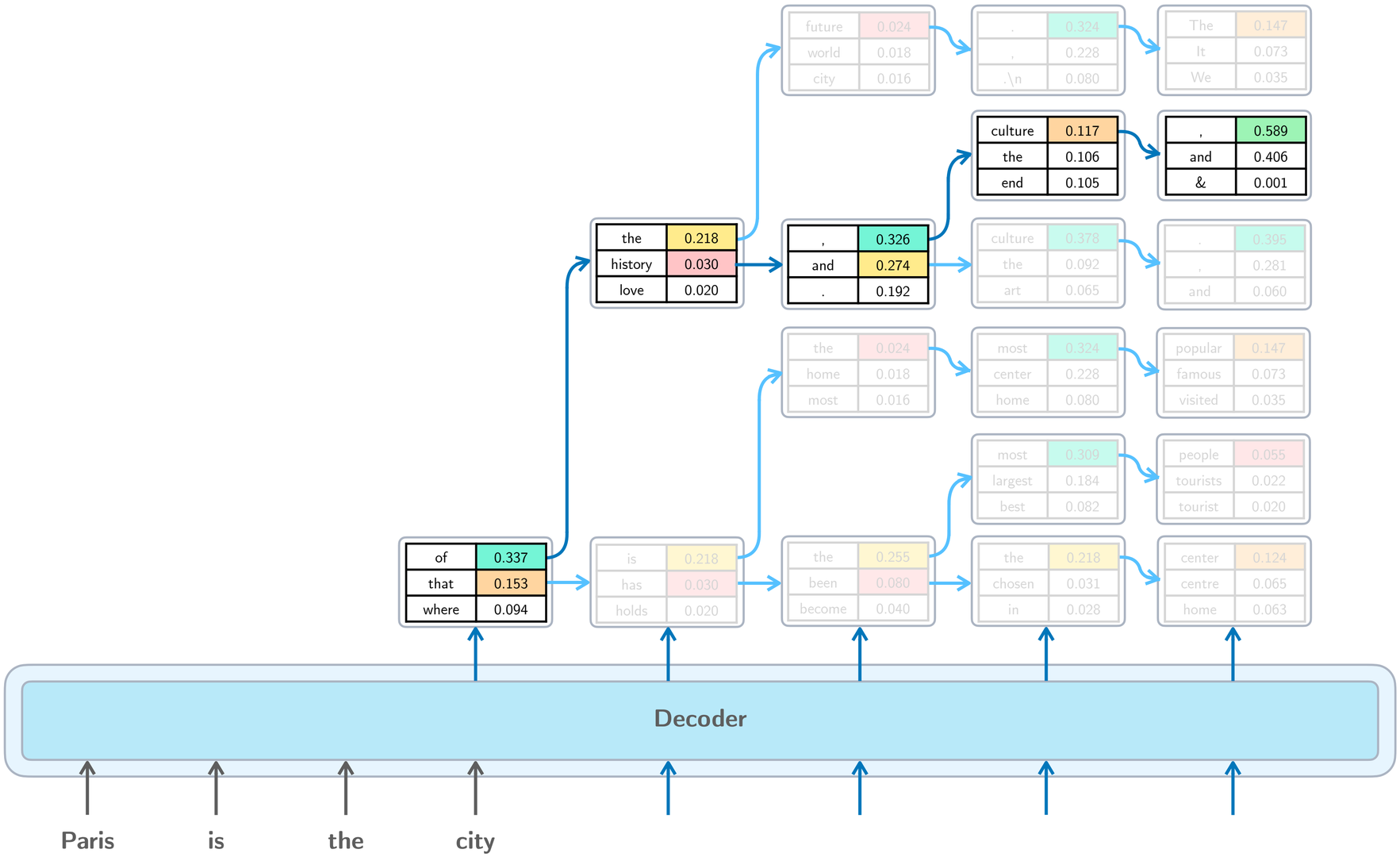
Beam search

Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217



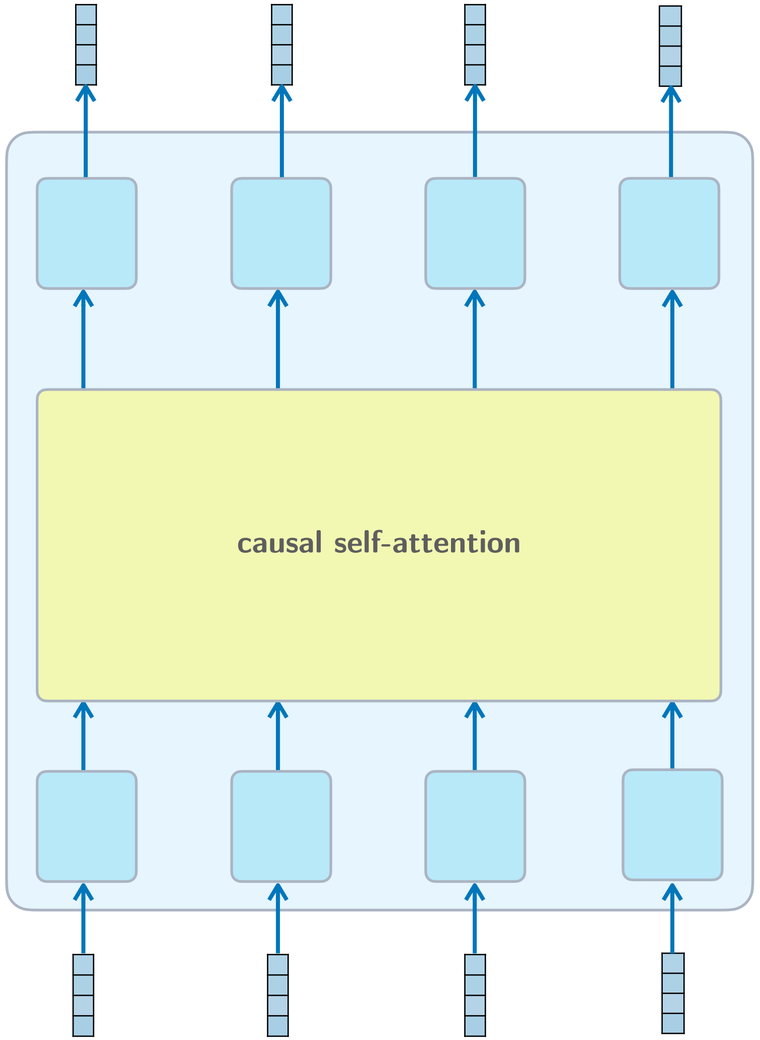
Beam search

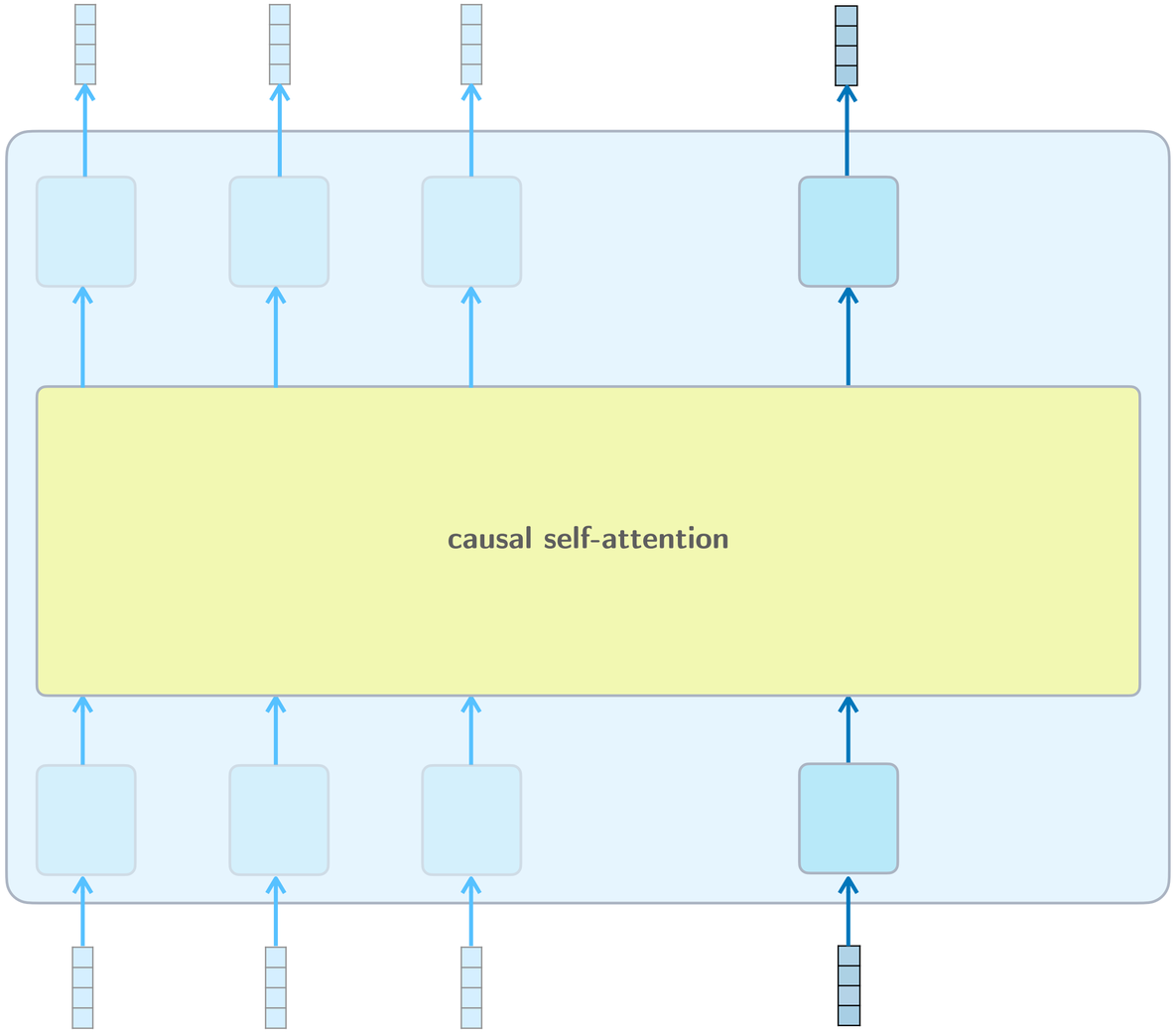
Based on https://pub.towardsai.net/how-does-an-llm-generate-text-fd9c57781217

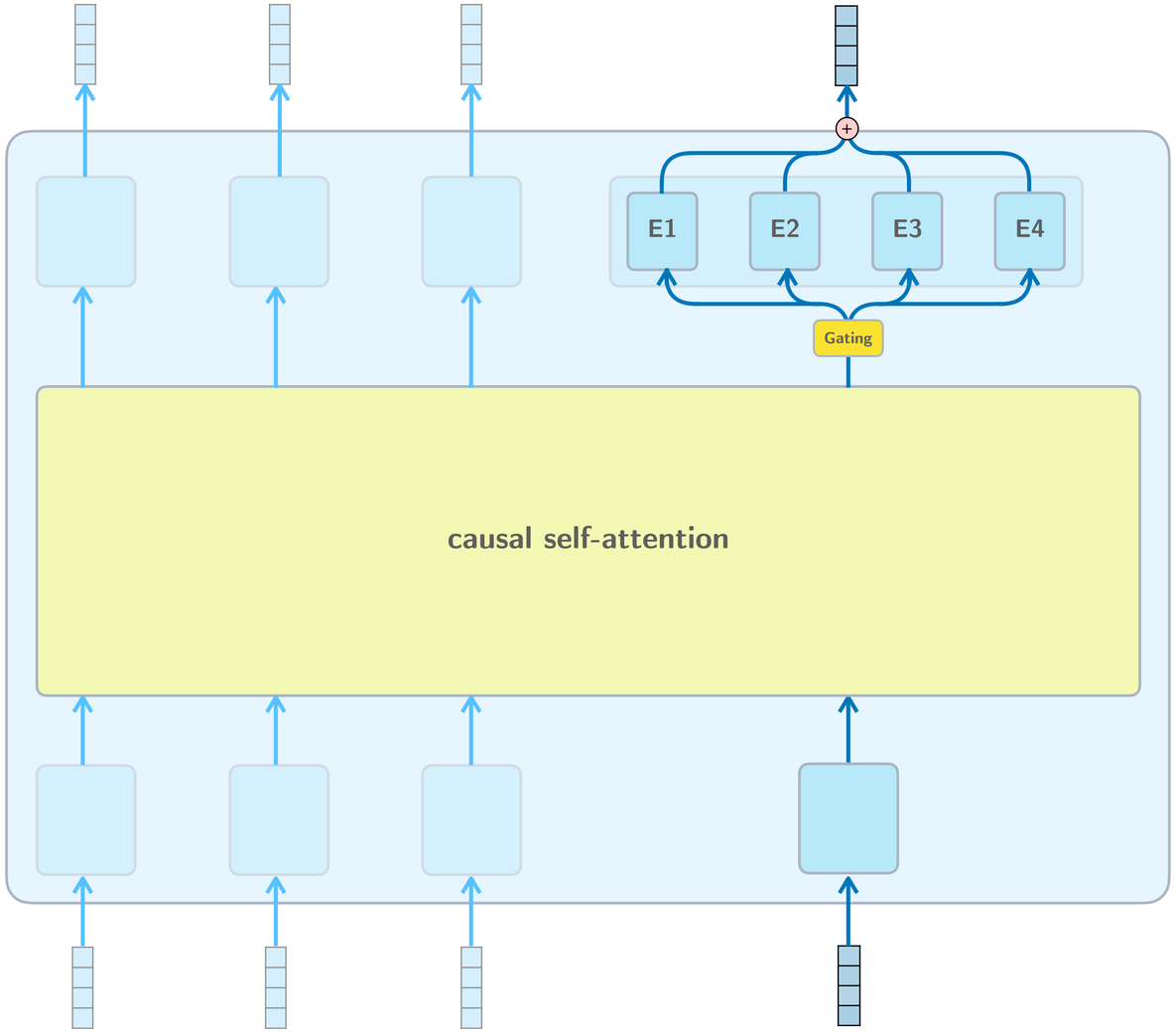


Part II

Mixture of Experts

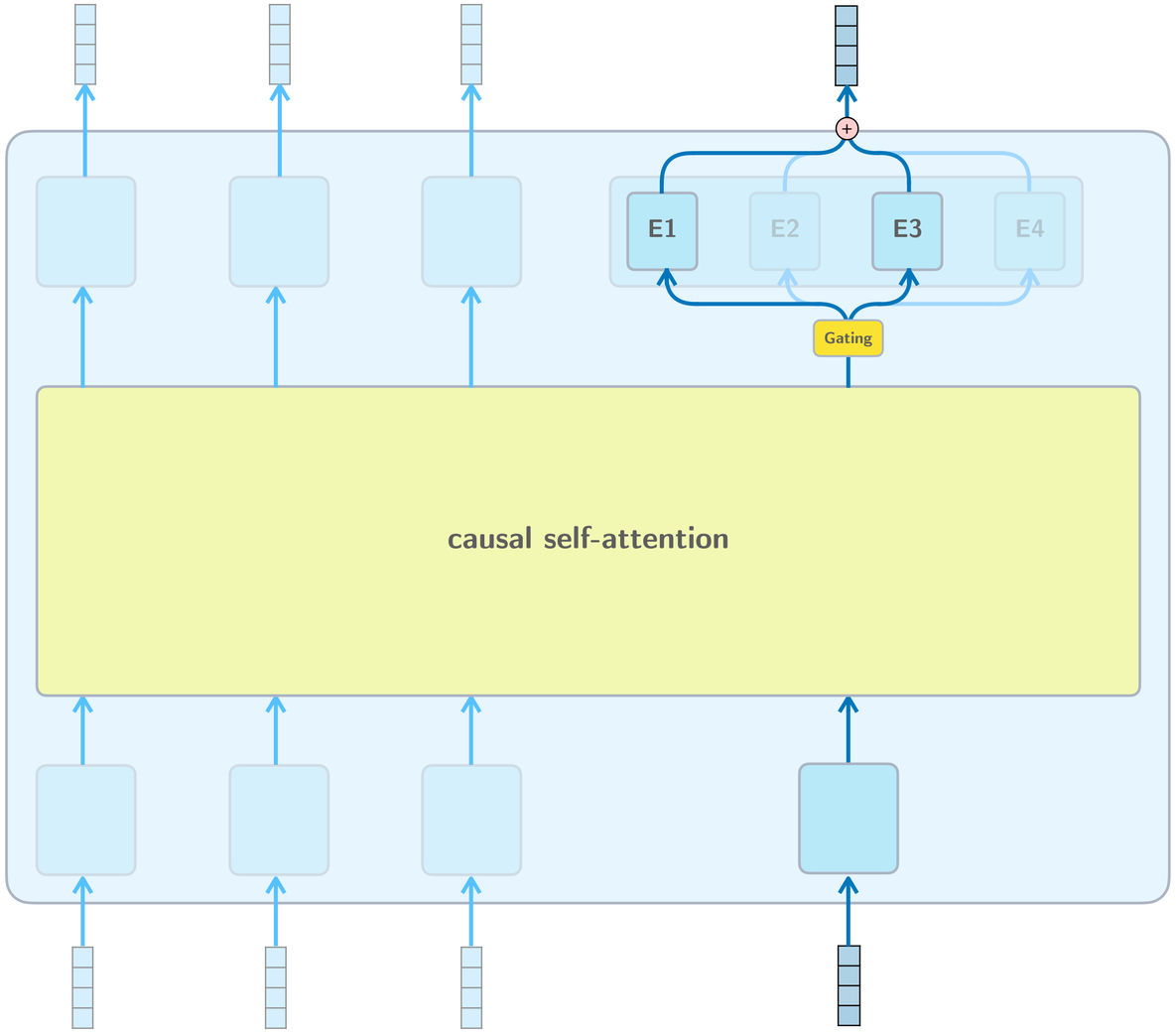

causal self-attention


causal self-attention

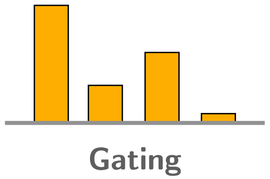


Multiple "**expert**" MLPs instead of a single MLP.

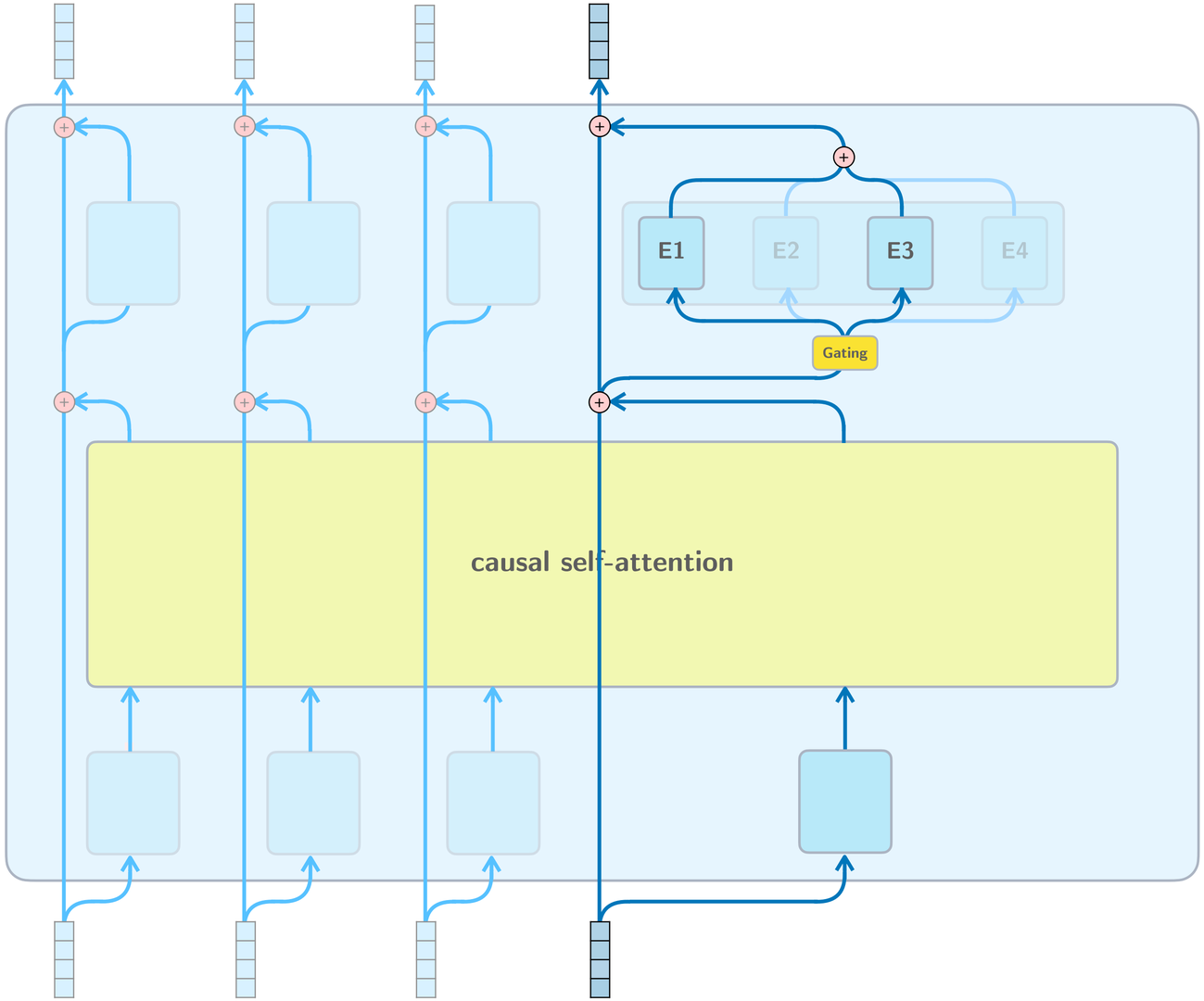
Sparse MoE



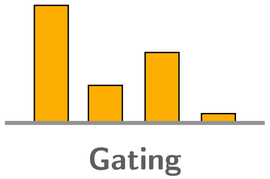
Multiple "**expert**" MLPs instead of a single MLP.



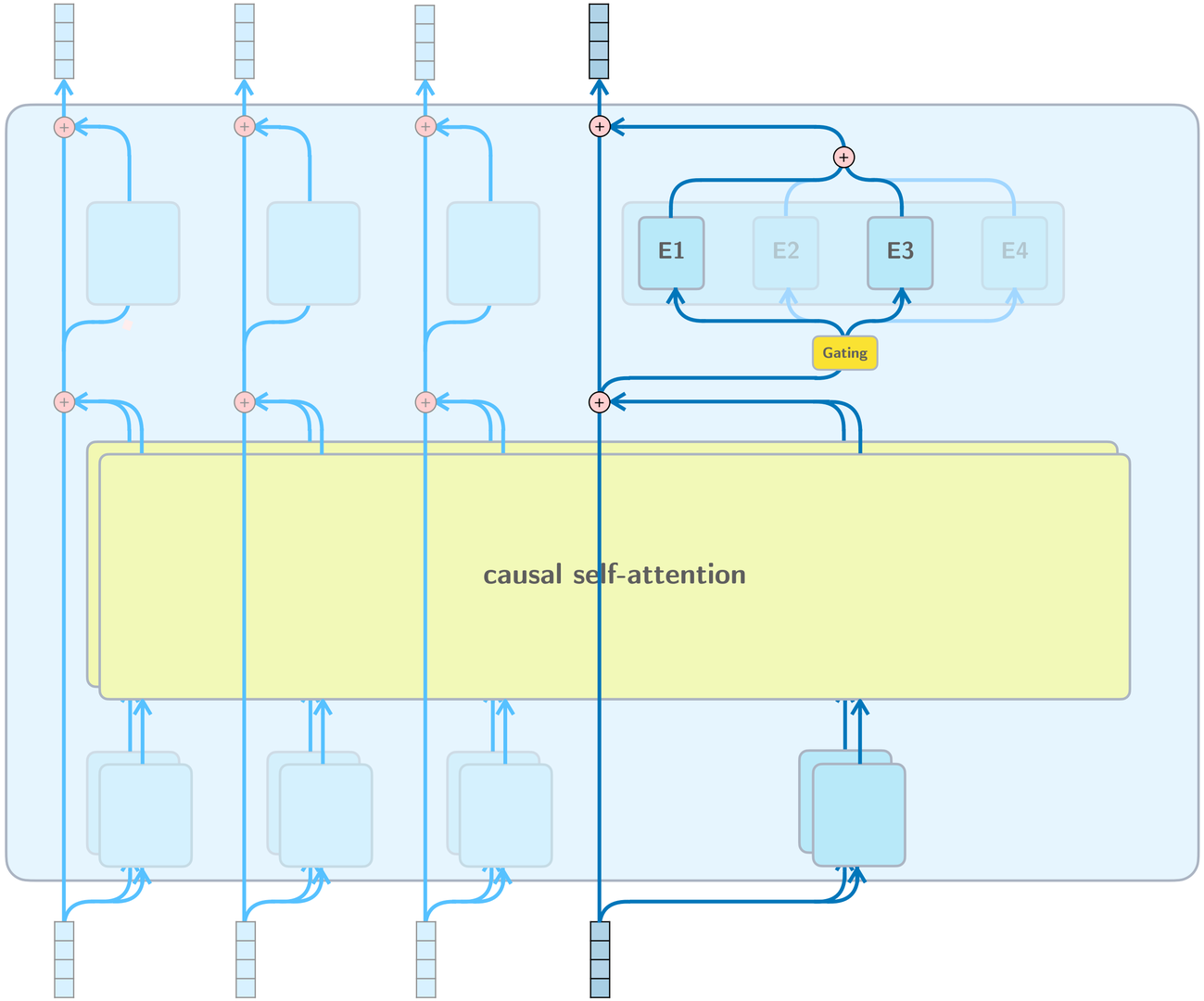
Sparse MoE



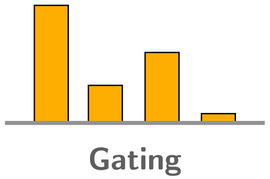
Multiple "**expert**" MLPs instead of a single MLP.



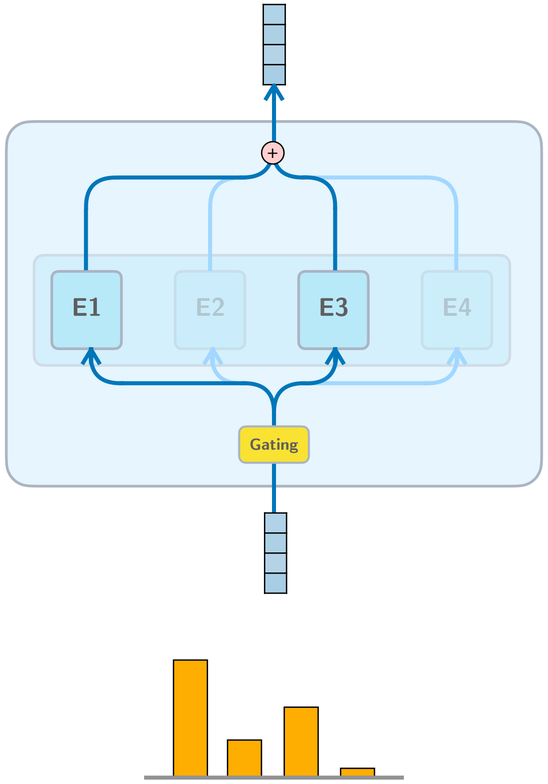
Sparse MoE



Multiple "**expert**" MLPs instead of a single MLP.

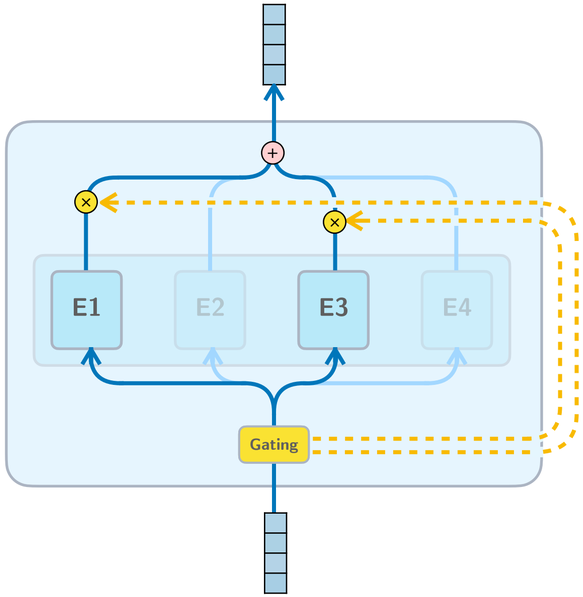


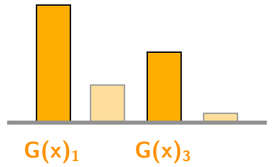
Sparse MoE



Multiple "**expert**" MLPs instead of a single MLP.

Sparse MoE



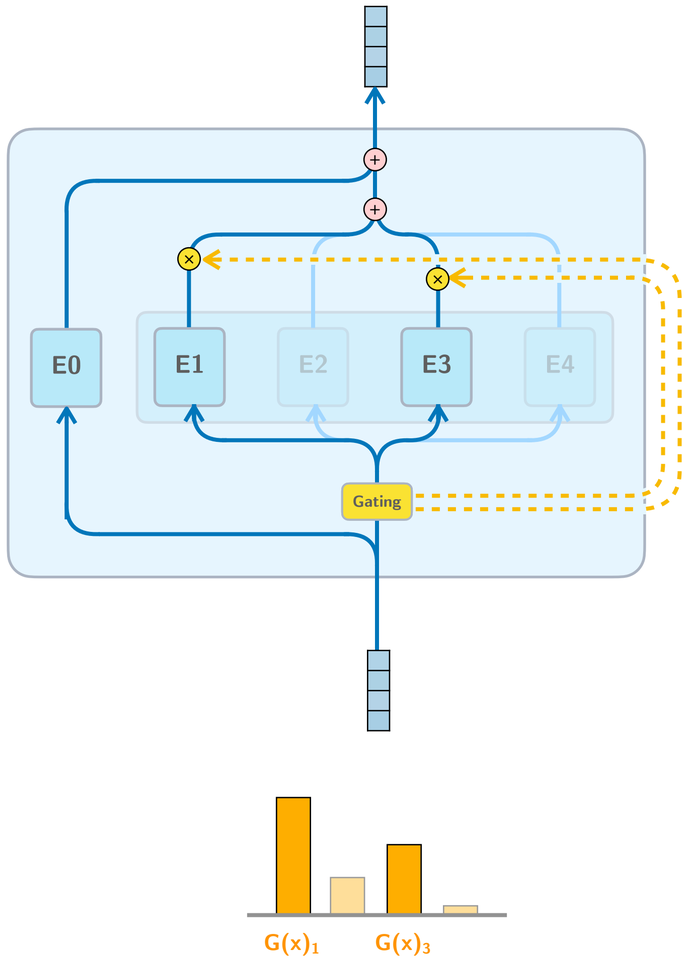


Multiple "**expert**" MLPs instead of a single MLP.

$$\text{Gating}(x)_1 \cdot \text{Expert}_1(x) + \text{Gating}(x)_3 \cdot \text{Expert}_3(x)$$

The output from each expert is **weighted** according to the gating network.

Shared experts and routed experts



Multiple "**expert**" MLPs instead of a single MLP.

$\text{Gating}(x)_1 \cdot \text{Expert}_1(x)$
$+ \text{Gating}(x)_3 \cdot \text{Expert}_3(x)$

The output from each expert is **weighted** according to the gating network.

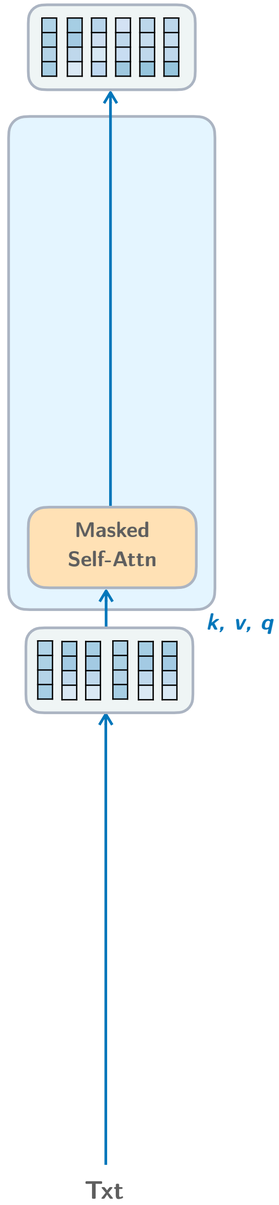
Part II

Retrieval and memory

RETRO (simplified)

RETRO = Retrieval-Enhanced Transformer

Model does not need to store all knowledge in its parameters.
Fewer parameters with same performance?

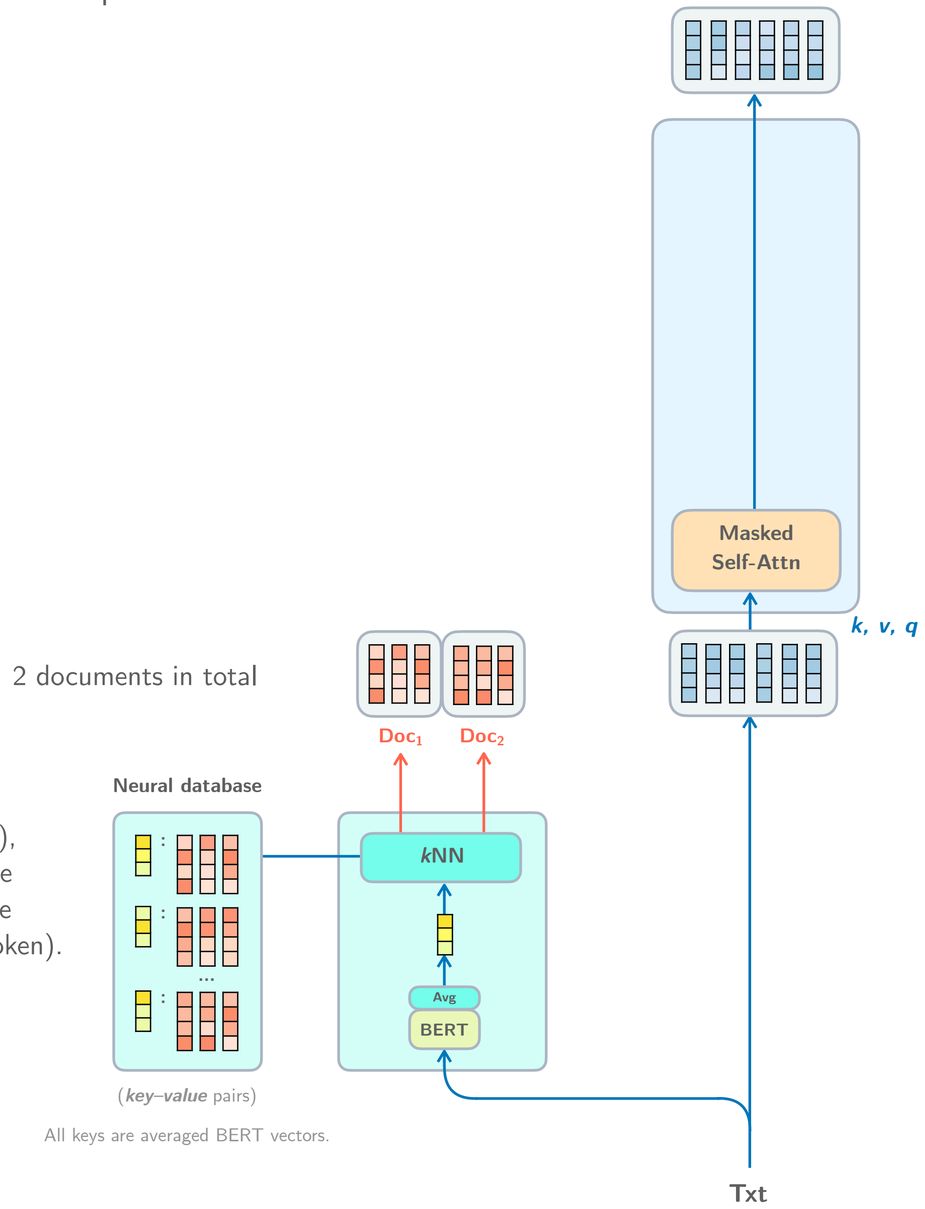


Leaving out Fwd for simplicity

RETRO (simplified)

RETRO = Retrieval-Enhanced Transformer

Model does not need to store all knowledge in its parameters.
Fewer parameters with same performance?



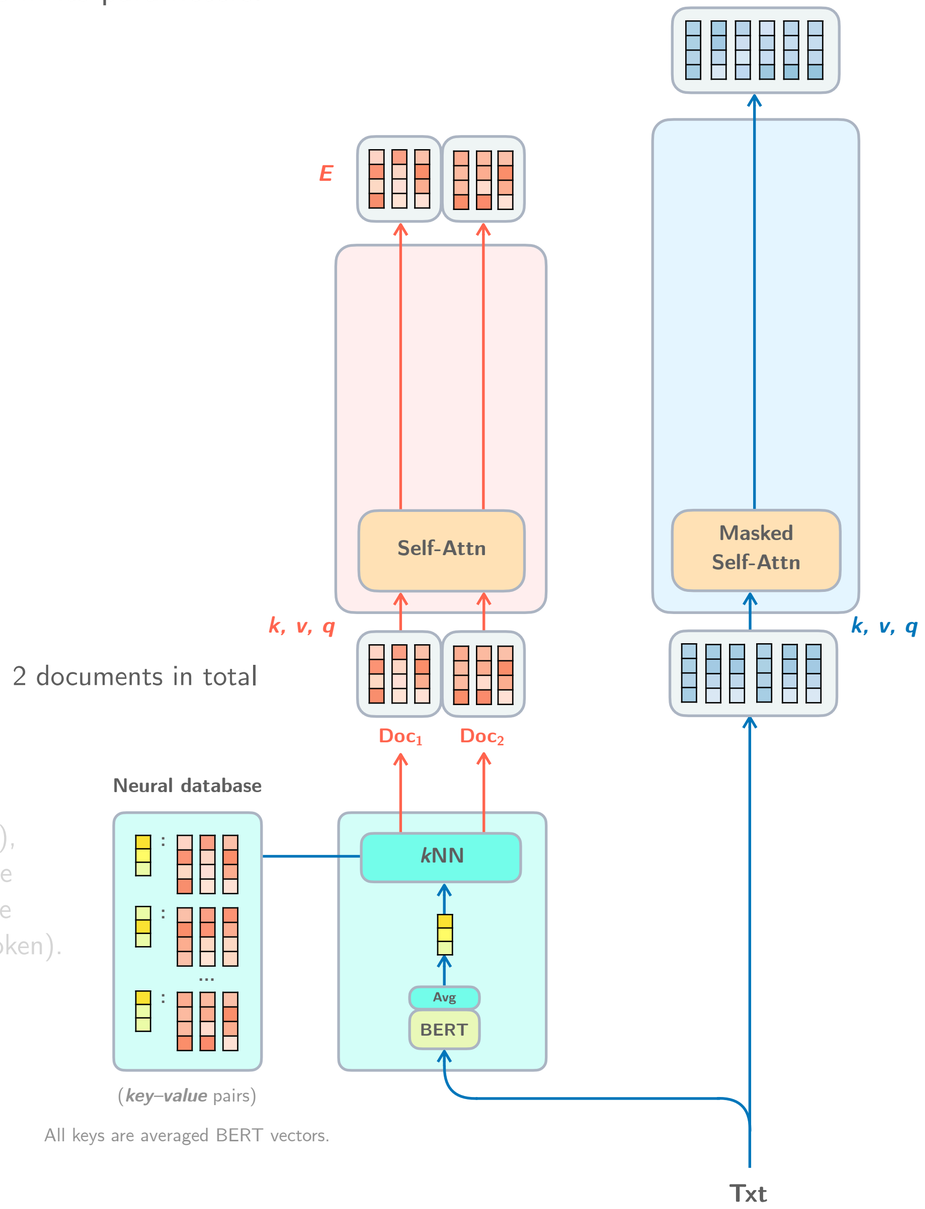
Retrieval itself is not learned (it's not differentiable), but during training, the model can expect to receive as input useful documents and it learns to make use of them (by attending over it to predict the next token).

Leaving out Fwd for simplicity

RETRO (simplified)

RETRO = Retrieval-Enhanced Transformer

Model does not need to store all knowledge in its parameters.
Fewer parameters with same performance?



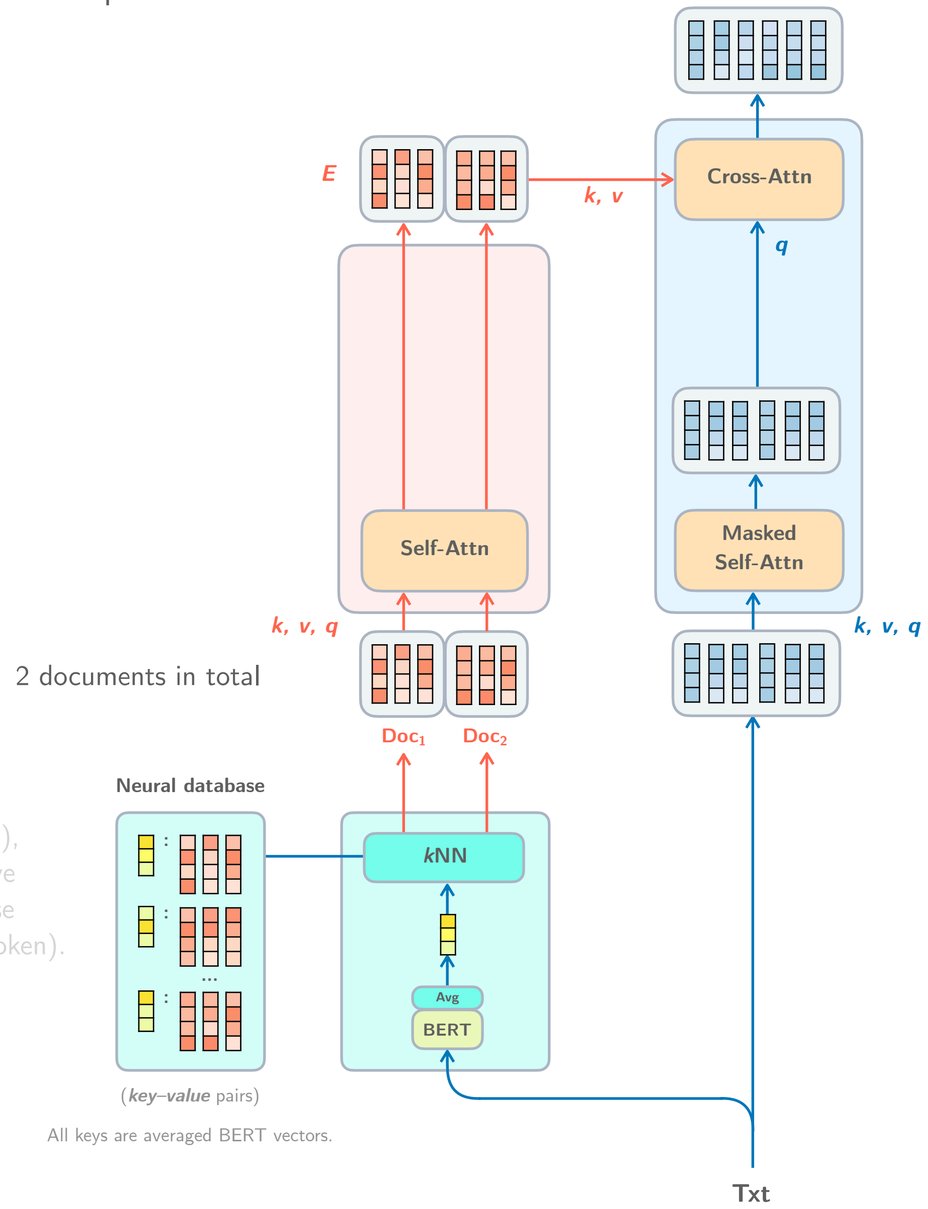
Retrieval itself is not learned (it's not differentiable), but during training, the model can expect to receive as input useful documents and it learns to make use of them (by attending over it to predict the next token).

Leaving out Fwd for simplicity

RETRO (simplified)

RETRO = Retrieval-Enhanced Transformer

Model does not need to store all knowledge in its parameters.
Fewer parameters with same performance?



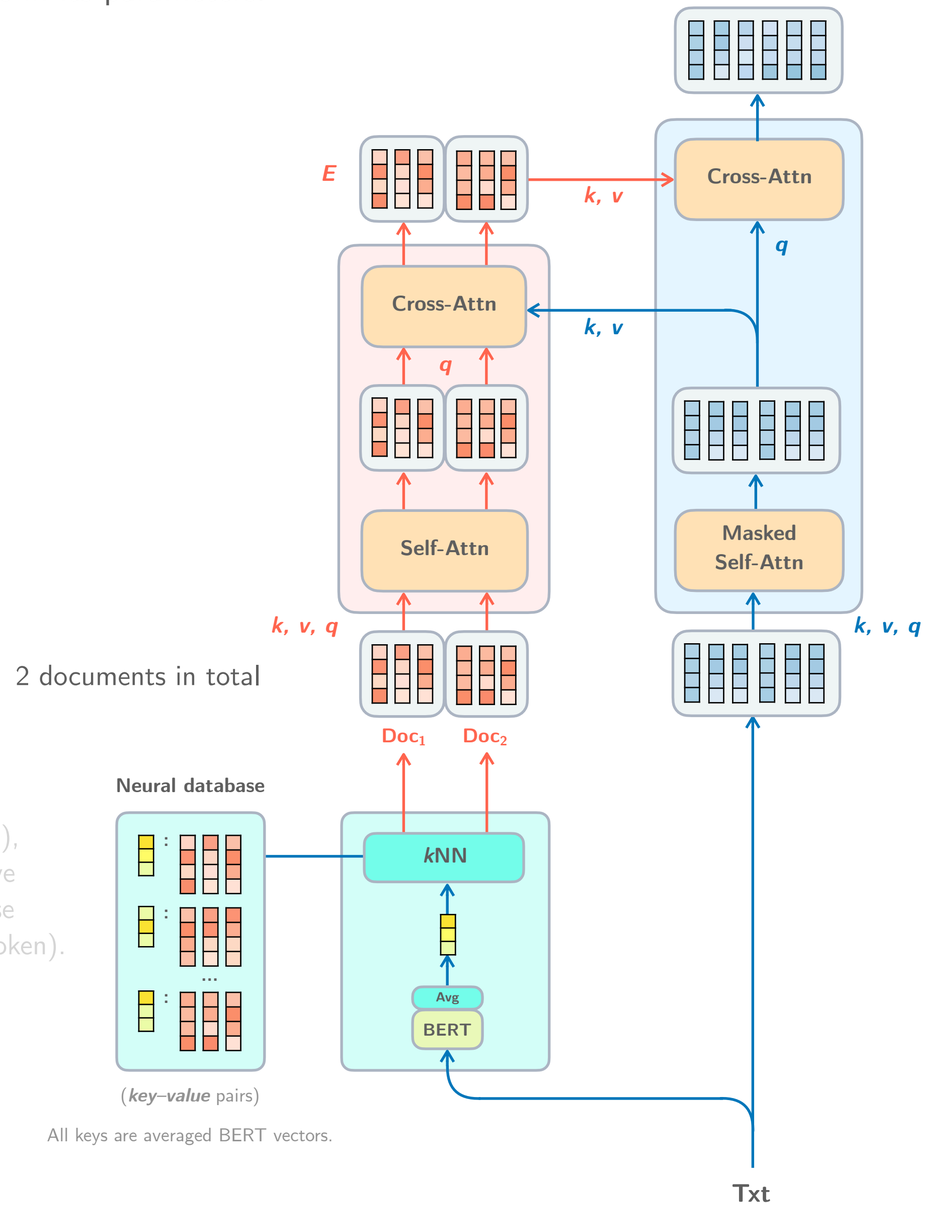
Retrieval itself is not learned (it's not differentiable), but during training, the model can expect to receive as input useful documents and it learns to make use of them (by attending over it to predict the next token).

Leaving out Fwd for simplicity

RETRO (simplified)

RETRO = Retrieval-Enhanced Transformer

Model does not need to store all knowledge in its parameters.
Fewer parameters with same performance?

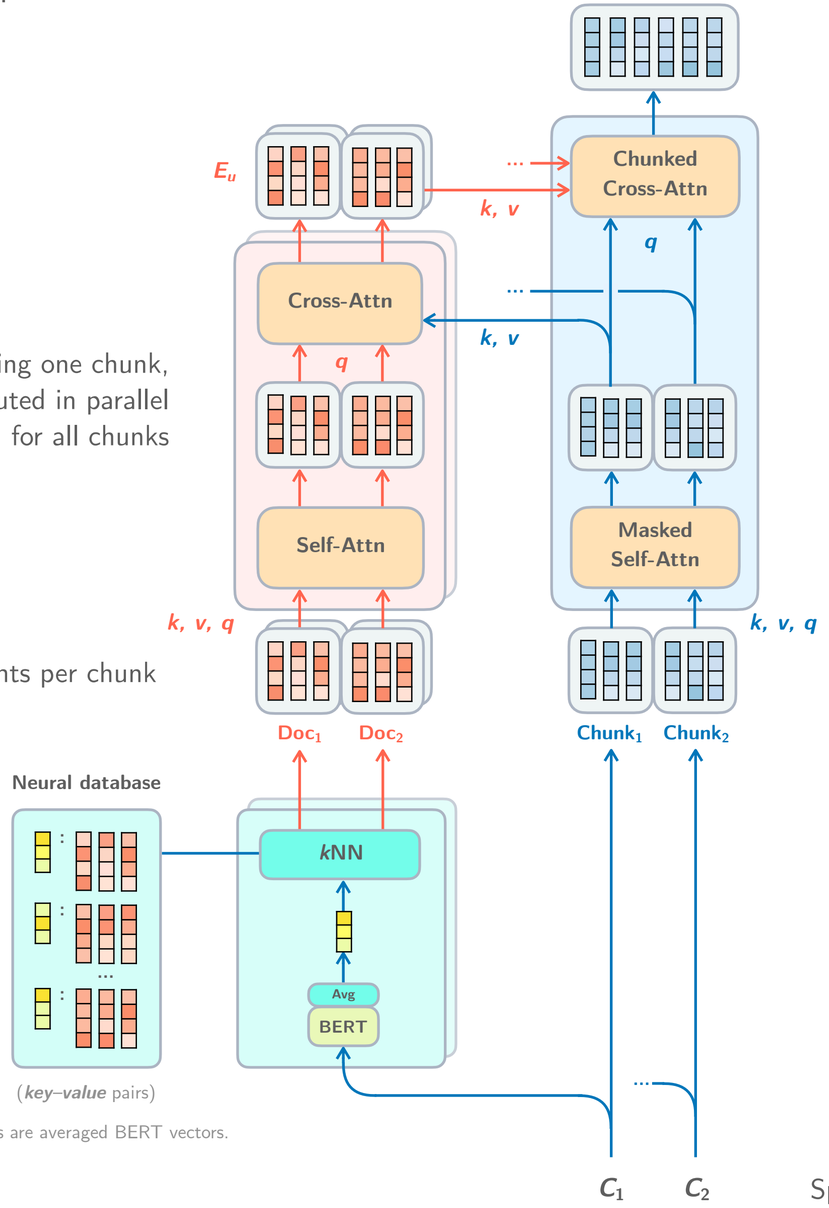


Retrieval itself is not learned (it's not differentiable), but during training, the model can expect to receive as input useful documents and it learns to make use of them (by attending over it to predict the next token).

Leaving out Fwd for simplicity

RETRO = Retrieval-Enhanced Transformer

Model does not need to store all knowledge in its parameters.
Fewer parameters with same performance?



Only showing one chunk, but this is computed in parallel for all chunks

Here, 2 documents per chunk

Retrieval itself is not learned (it's not differentiable), but during training, the model can expect to receive as input useful documents and it learns to make use of them (by attending over it to predict the next token).

Split text into fixed number of chunks l

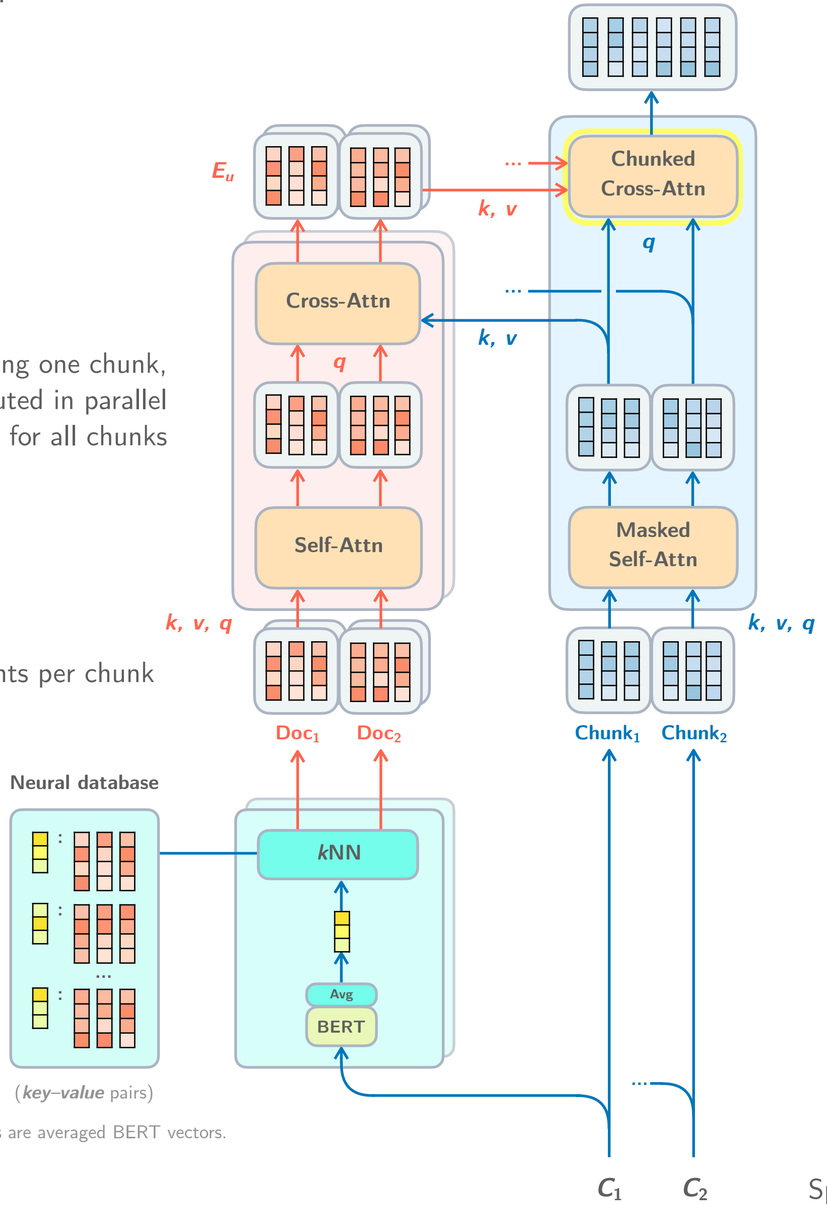
Leaving out Fwd for simplicity

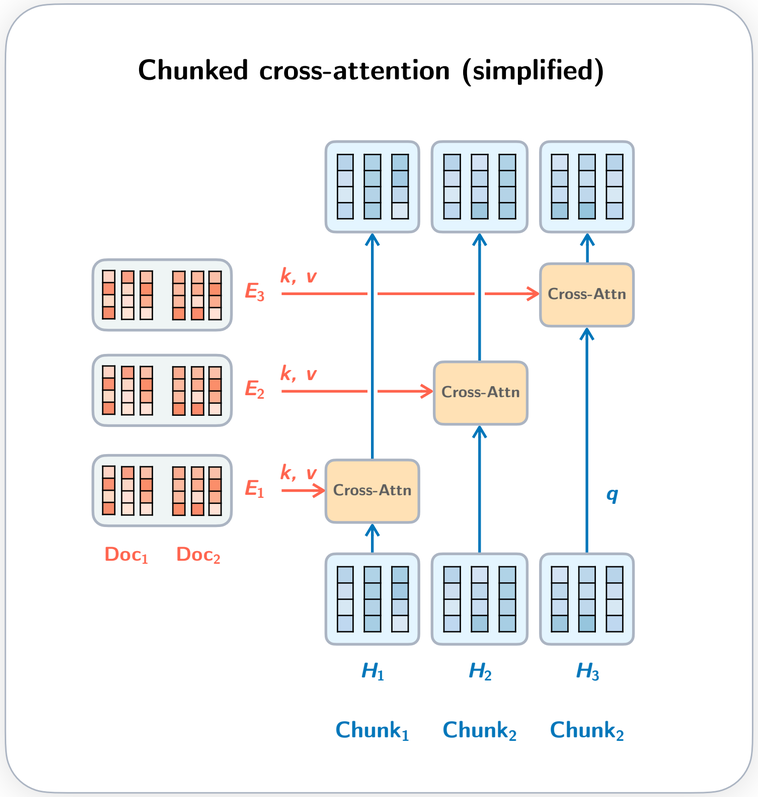
RETRO = Retrieval-Enhanced Transformer

Model does not need to store all knowledge in its parameters.
Fewer parameters with same performance?

Only showing one chunk, but this is computed in parallel for all chunks

Here, 2 documents per chunk





E_1 are the encoded documents corresponding to chunk 1 (C_1, and representation H_1)

Retrieval itself is not learned (it's not differentiable), but during training, the model can expect to receive as input useful documents and it learns to make use of them (by attending over it to predict the next token).

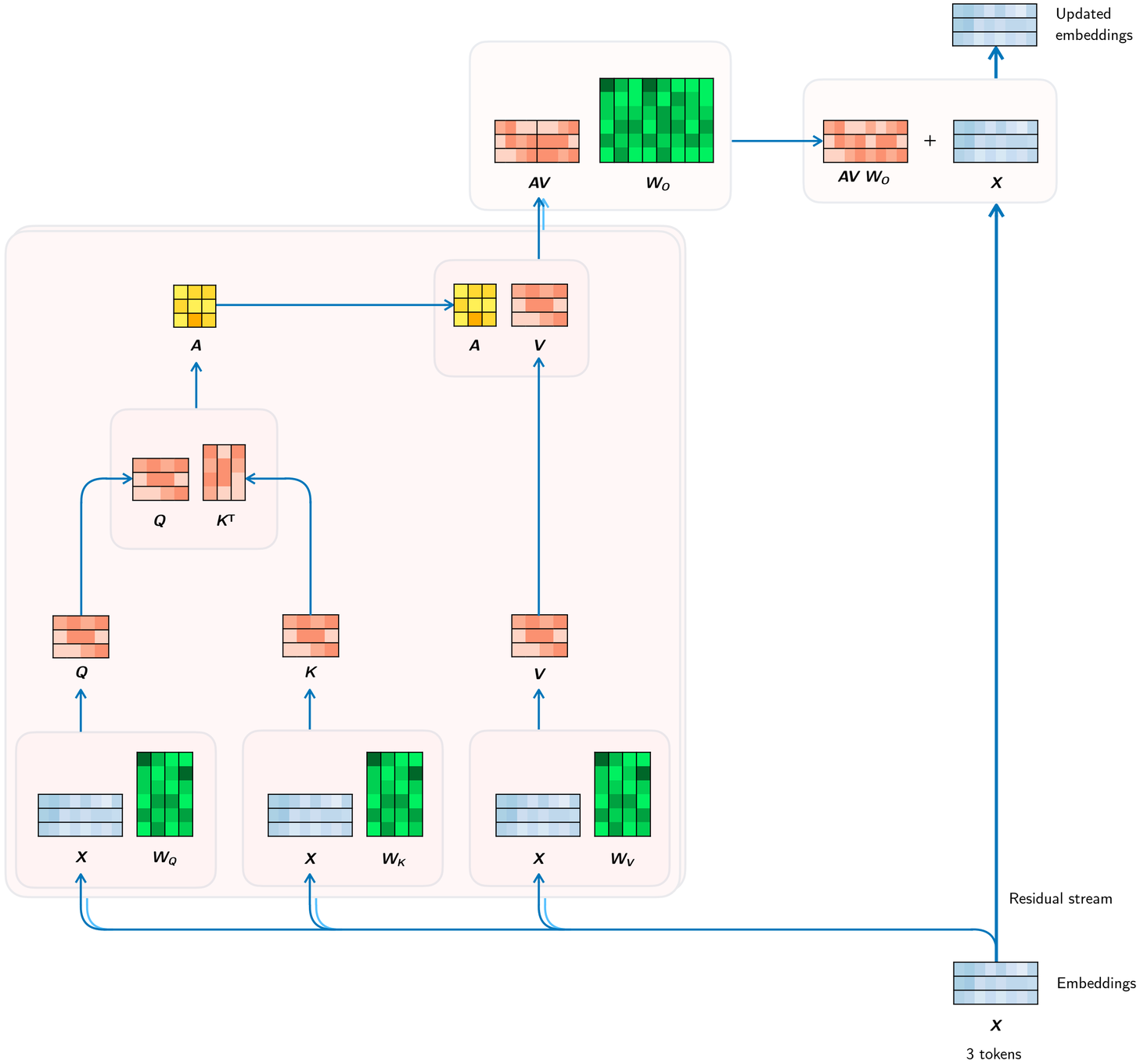
Split text into fixed number of chunks l

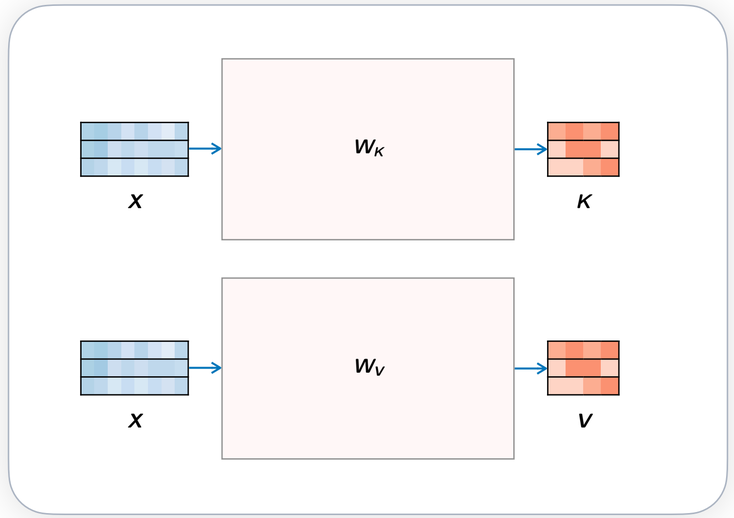
Leaving out Fwd for simplicity

Part II

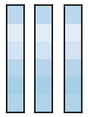
Alternative attention designs

Attention

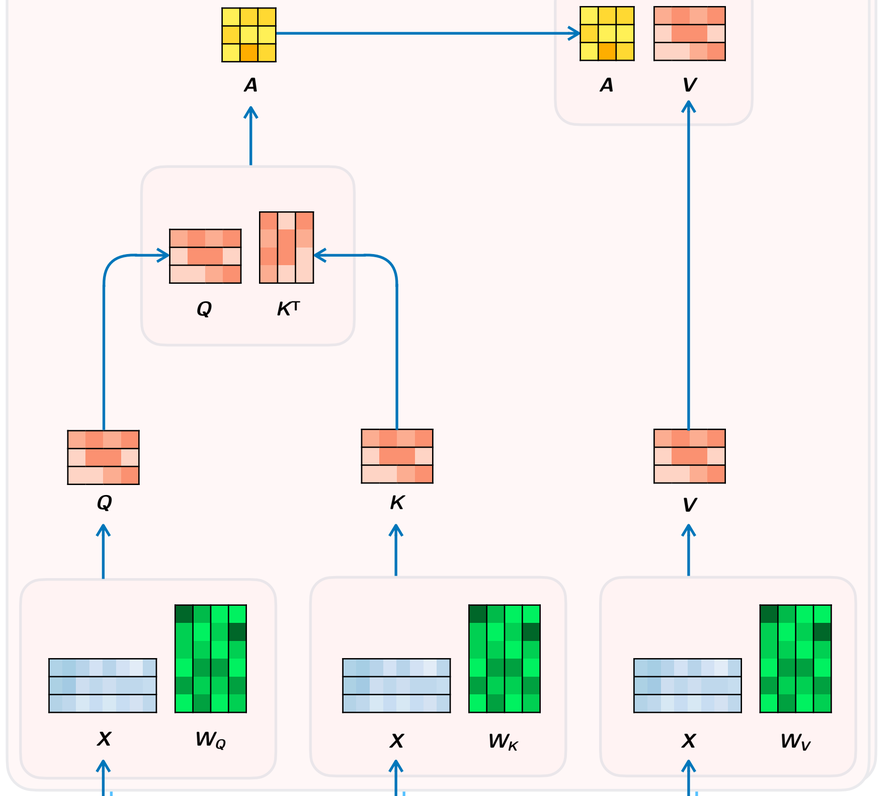


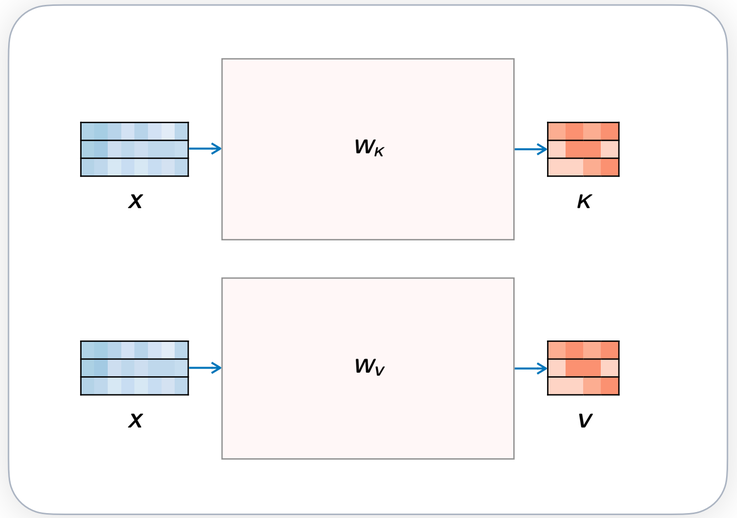


X has shape (3, *d*)
Here: d=8
$\boldsymbol{W_Q}$ has shape (*d*, *h*)
We do $\boldsymbol{Q} = \boldsymbol{X}\,\boldsymbol{W_Q}$, and then ***Q*** has shape (3, *h*)
(same with ***K***, ***V***)
h is chosen such that $h = d/n_{heads}$
so that we can cleanly concatenate the weighed-value matrices together again to get (3, n_{heads}*h) = (3, *d*)
Here: n_{heads}=2, *h*=4



Attention





X has shape $(3, d)$

Here: d=8

W_Q has shape (d, h)

We do $Q = X W_Q$, and then Q has shape $(3, h)$

(same with K, V)

h is chosen such that $h = d/n_{heads}$

so that we can cleanly concatenate the weighed-value matrices together again to get $(3, n_{heads}*h) = (3, d)$

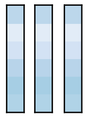
Here: n_{heads}=2, h=4

Residual stream

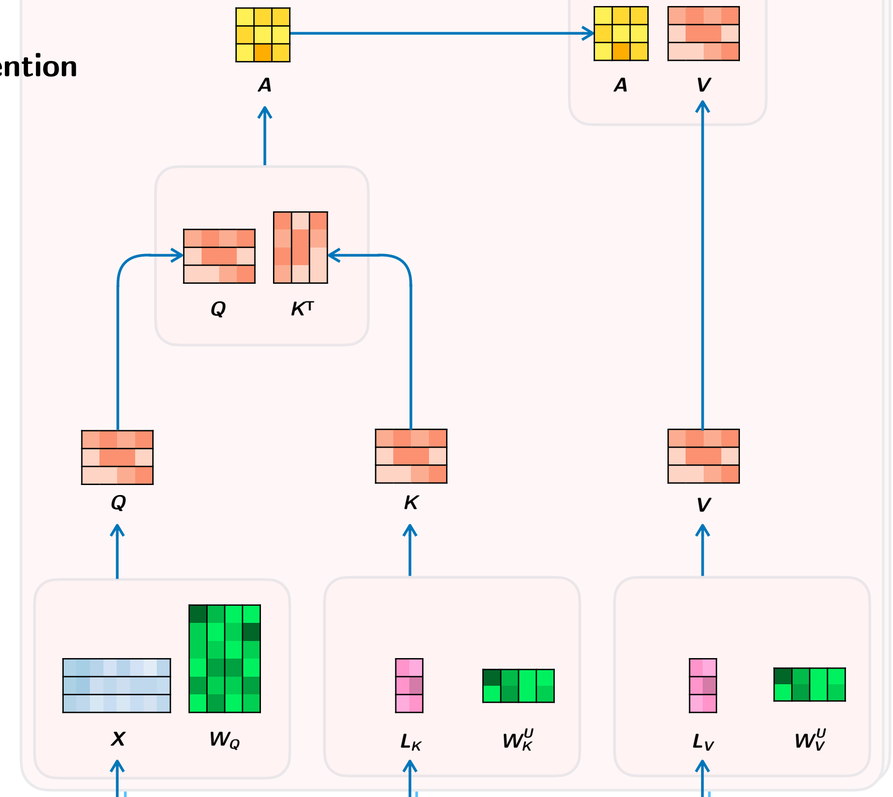
Embeddings

X

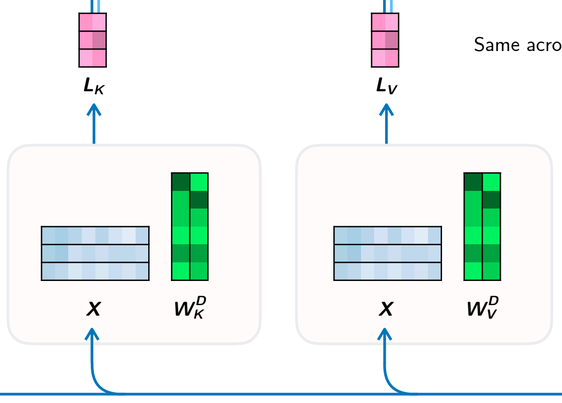
3 tokens



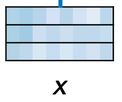
Latent attention



Same across all heads!

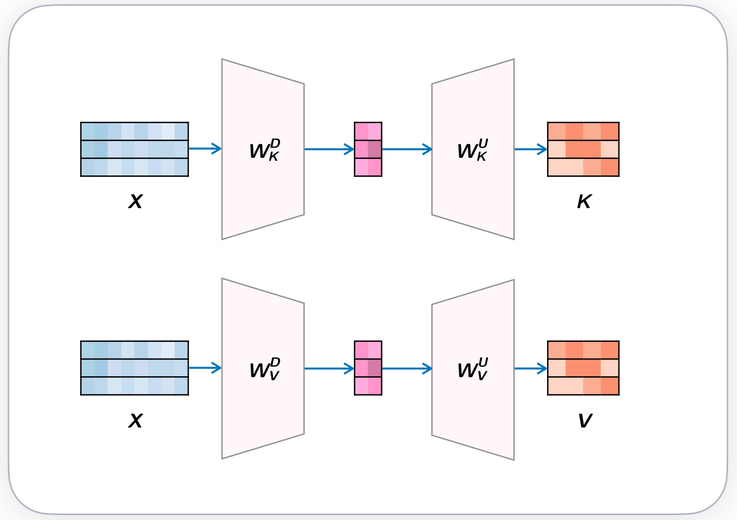


Residual stream



Embeddings

3 tokens



X has shape $(3, d)$

Here: d=8

W_Q has shape (d, h)

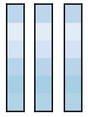
We do $Q = X W_Q$, and then Q has shape $(3, h)$

(same with K, V)

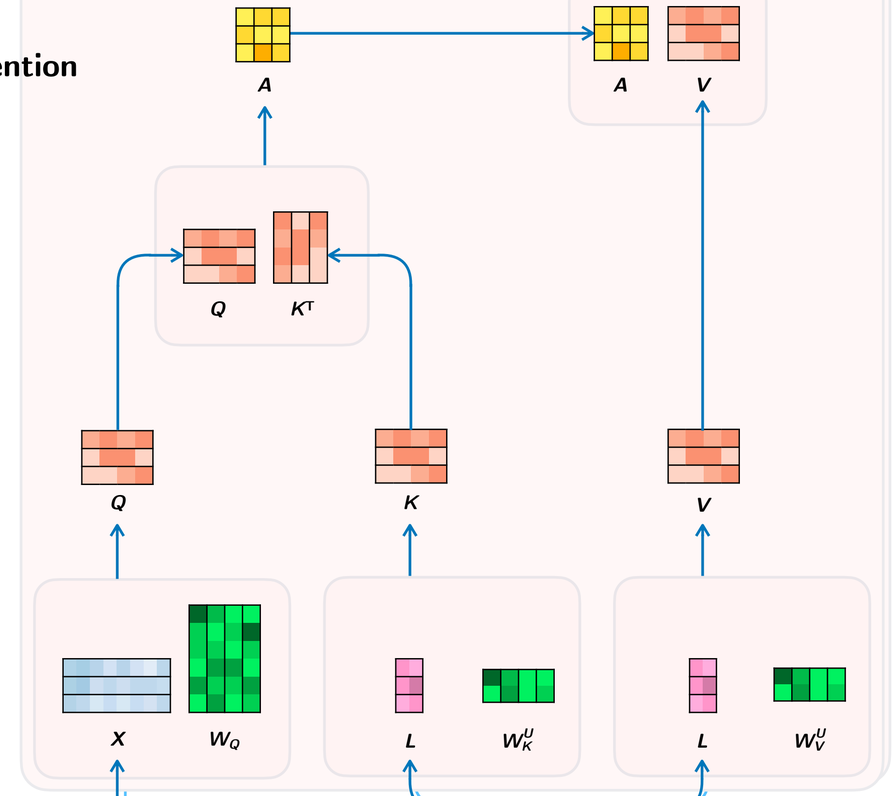
h is chosen such that $h = d/n_{heads}$

so that we can cleanly concatenate the weighed-value matrices together again to get $(3, n_{heads}*h) = (3, d)$

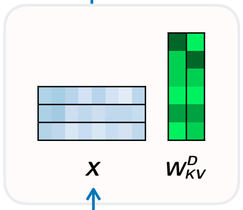
Here: n_{heads}=2, h=4

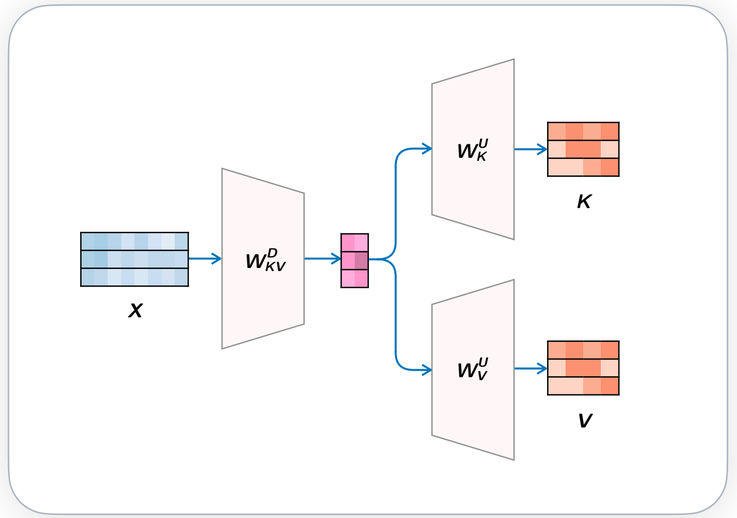


Latent attention



Same across all heads!





X has shape $(3, d)$

Here: d=8

W_Q has shape (d, h)

We do $Q = X W_Q$, and then Q has shape $(3, h)$

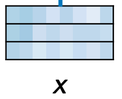
(same with K, V)

h is chosen such that $h = d/n_{heads}$

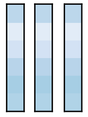
so that we can cleanly concatenate the weighed-value matrices together again to get $(3, n_{heads}*h) = (3, d)$

Here: n_{heads}=2, h=4

Residual stream



Embeddings

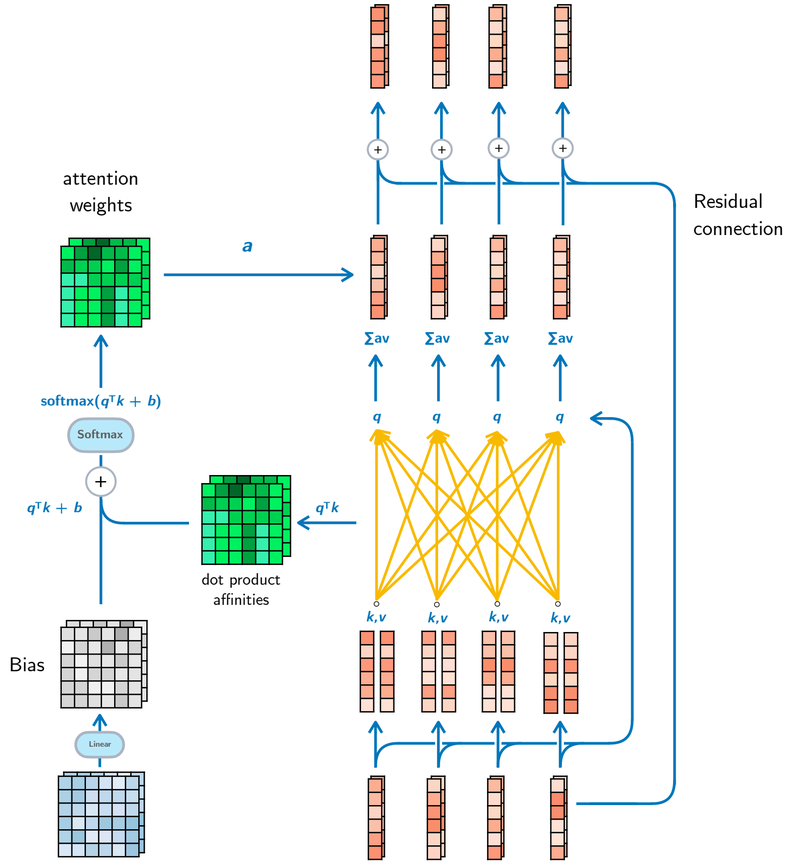


3 tokens

Biased attention

e.g. used in AlphaFold 2

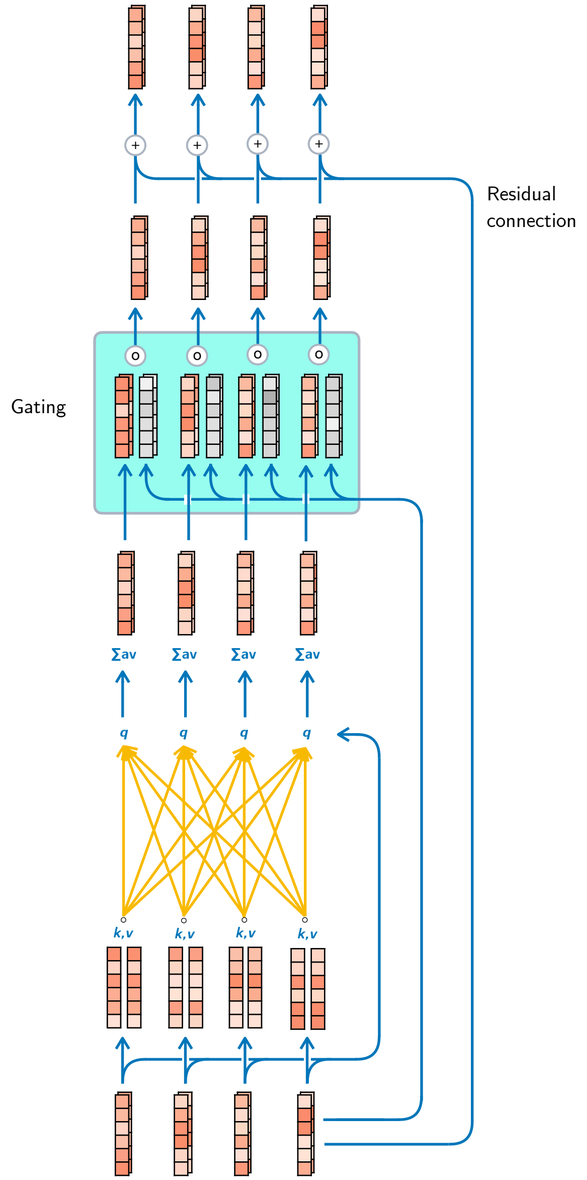
two-lane transformer, where one lane biases the other
by adding a bias tensor to the dot-product similarity scores
alternatively, think of encoder biasing the decoder
or decoder biasing the encoder



Linear layers to compute keys, values, queries not shown.

Gated attention

e.g. used in AlphaFold 2



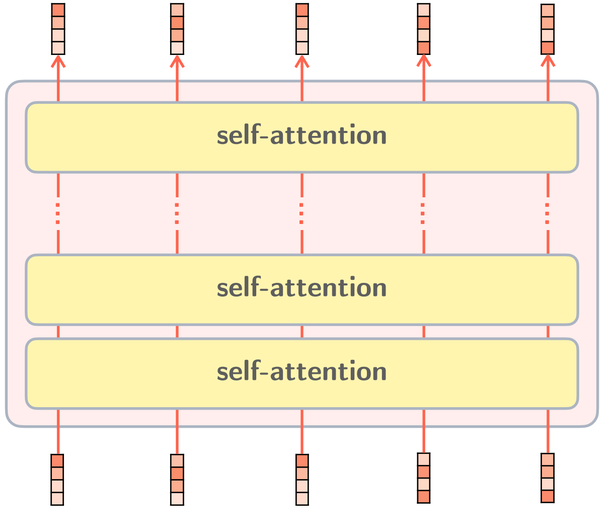
Linear layers to compute keys, values, queries not shown.

Part II

Non-attention transformers

What is a transformer?

Not showing non-mixing layers for simplicity

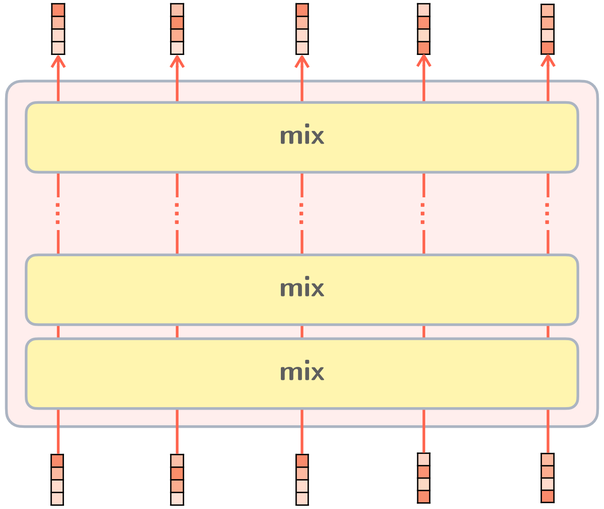


Transformer

Any non-recurrent sequence-processing neural network where the sequence representation gets refined with each layer by letting individual time steps learn about all the other time steps?

What is a transformer?

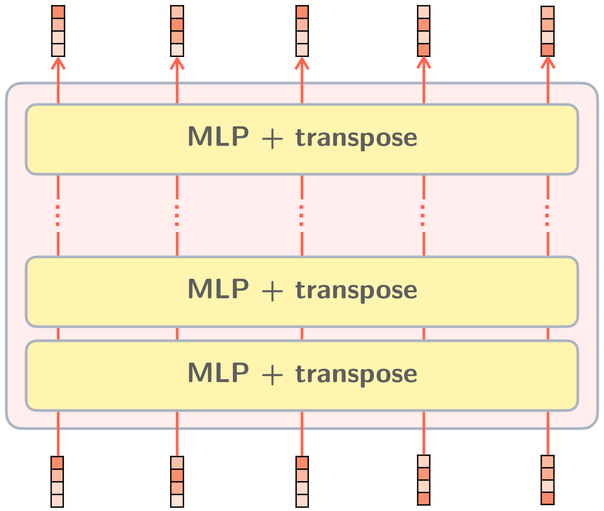
Not showing non-mixing layers for simplicity



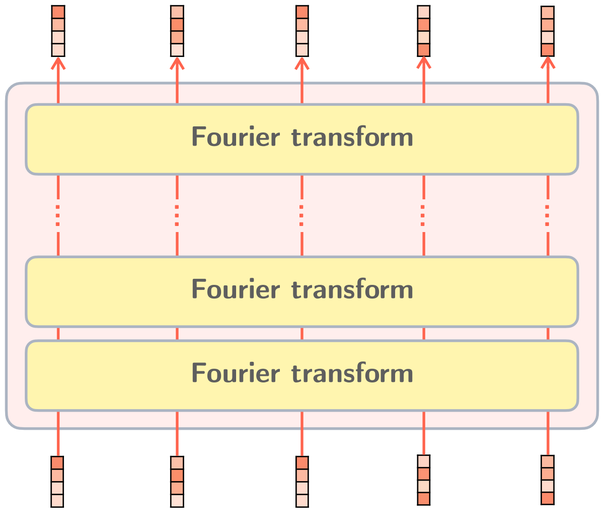
“Token network”

Any non-recurrent sequence-processing neural network where the sequence representation gets refined with each layer by letting individual time steps learn about all the other time steps?

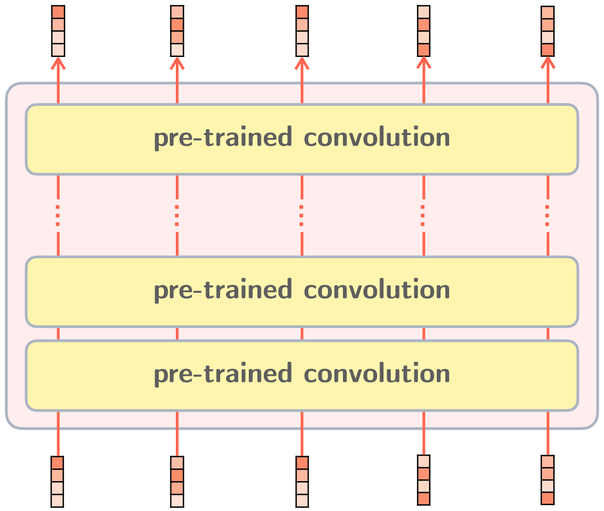
What is a transformer?



MLP Mixer



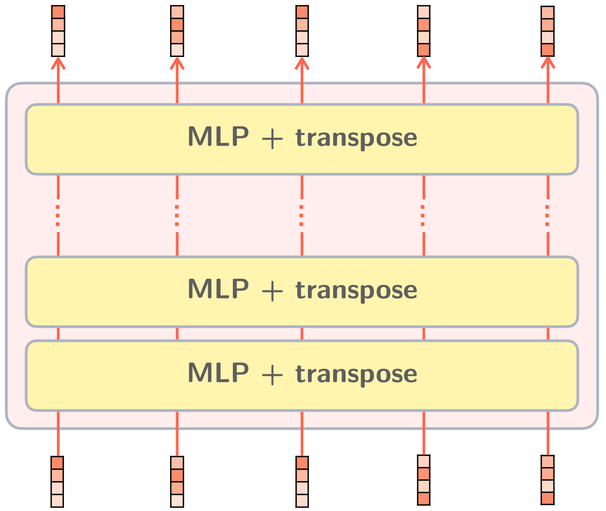
FNet



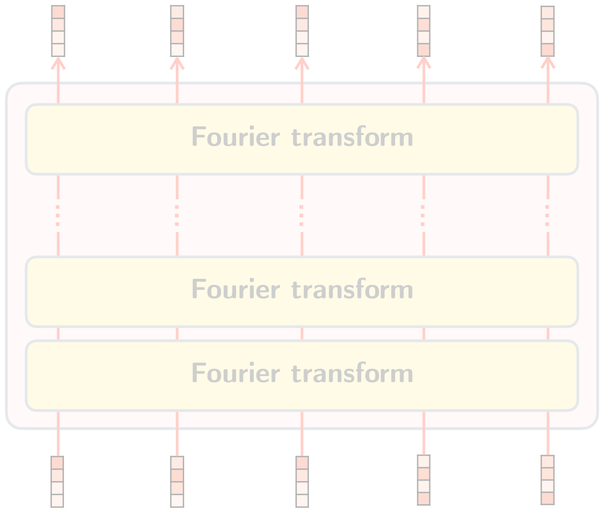
pre-trained convolutions

Any non-recurrent sequence-processing neural network where the sequence representation gets refined with each layer by letting individual time steps learn about all the other time steps?

What is a transformer?



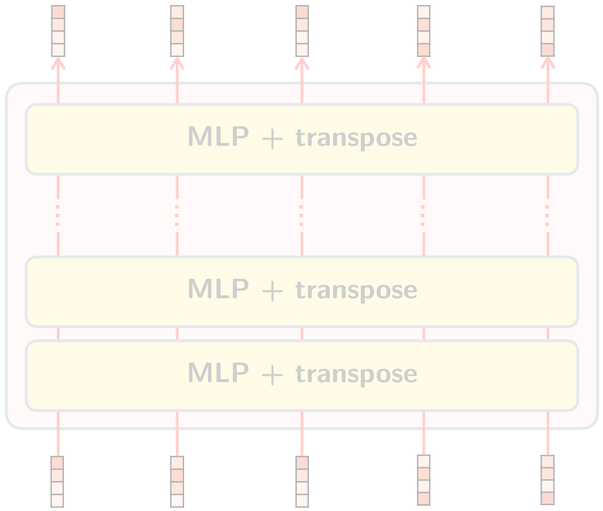
MLP Mixer



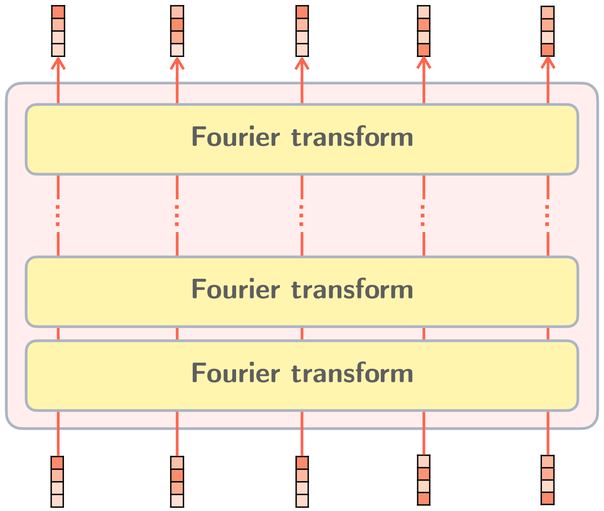
FNet

pre-trained convolutions

What is a transformer?



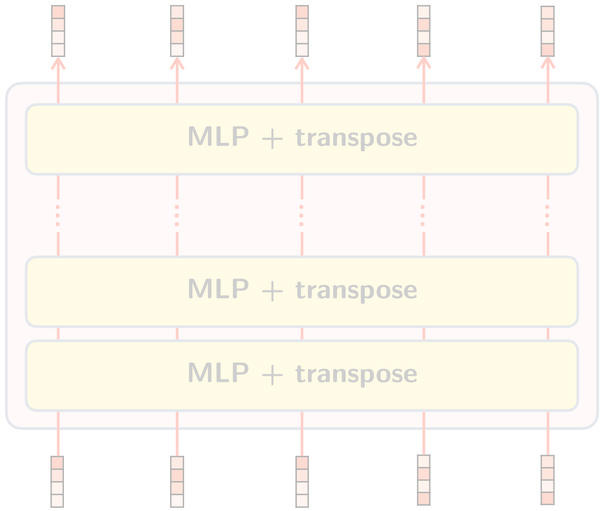
MLP Mixer



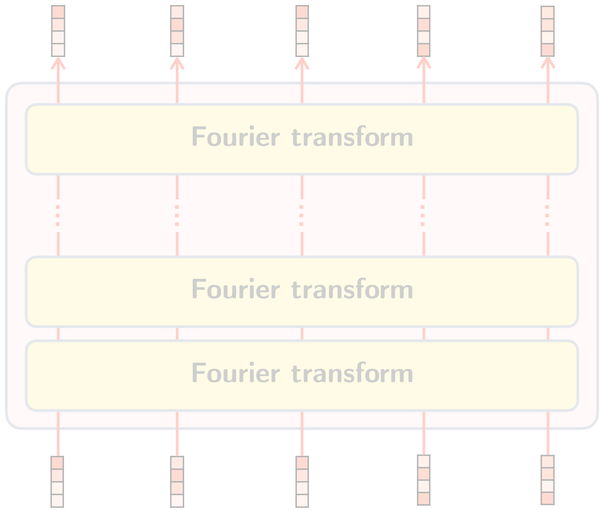
FNet

pre-trained convolutions

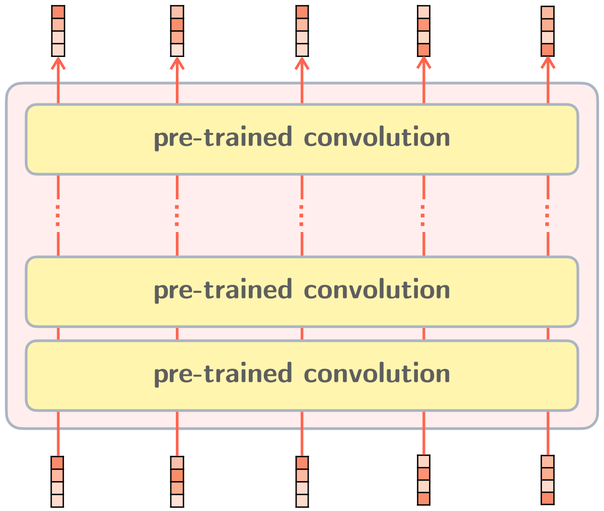
What is a transformer?



MLP Mixer

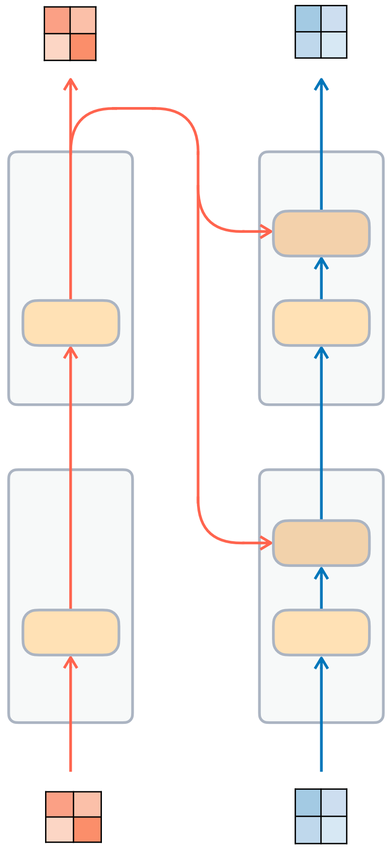


FNet



pre-trained convolutions

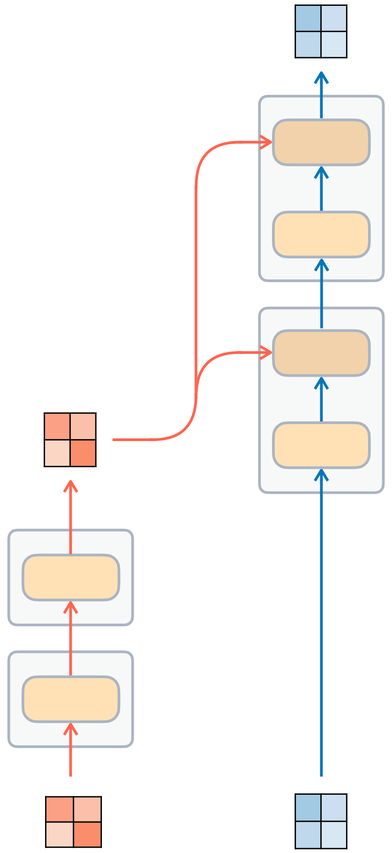
Multi-track transformers



Encoder–Decoder
is not 2-tracked

First, the encoder fully encodes the input into a representation,
then, the decoder decodes the output by attending to this representation.

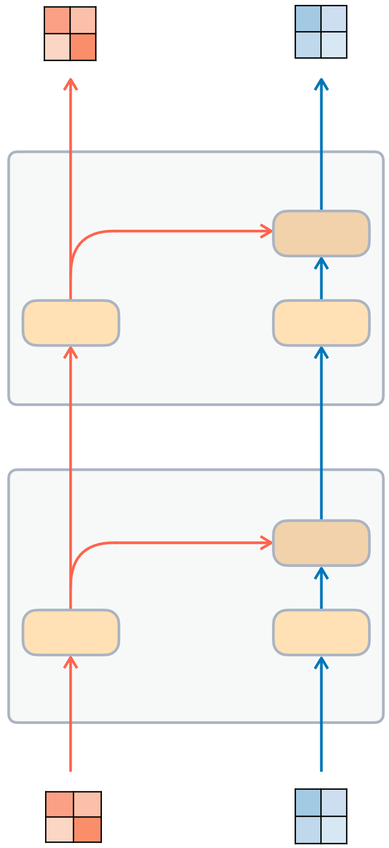
In other words, first, fully run the encoder, *then*, fully run the decoder.



Encoder–Decoder
is not 2-tracked

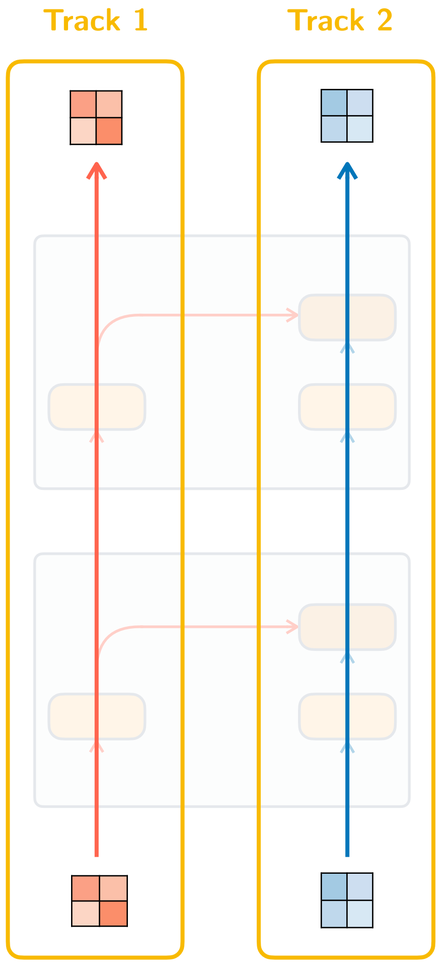
First, the encoder fully encodes the input into a representation,
then, the decoder decodes the output by attending to this representation.

In other words, first, fully run the encoder, *then*, fully run the decoder.



modified
“Encoder–Decoder”
to be 2-tracked

In a transformer with multiple tracks,
the tracks communicate in each “block”,
e.g. via cross-attention or biased self-attention.



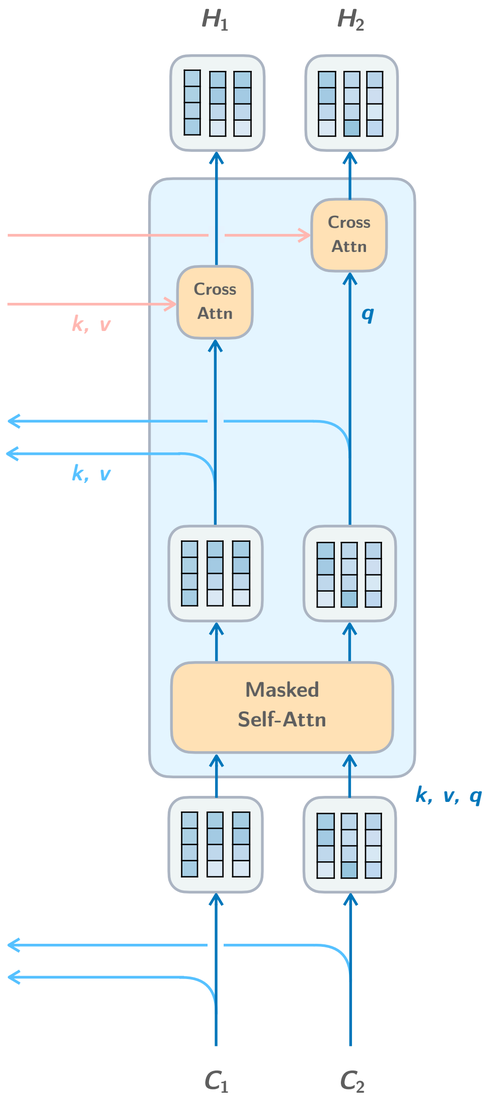
modified
“Encoder–Decoder”
to be 2-tracked

In a transformer with multiple tracks,
the tracks communicate in each “block”,
e.g. via cross-attention or biased self-attention.

Interpreted as a N-track transformer

Each track maintains and refines a representation of one chunk.

Each track works the same.

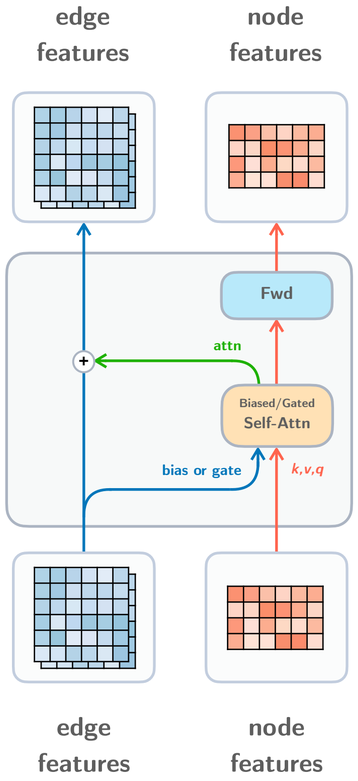


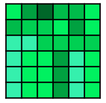
Graph Transformer

Interpreted as a 2-track transformer

to maintain and refine *node* and *edge* representations

Different tracks operate on different representations.





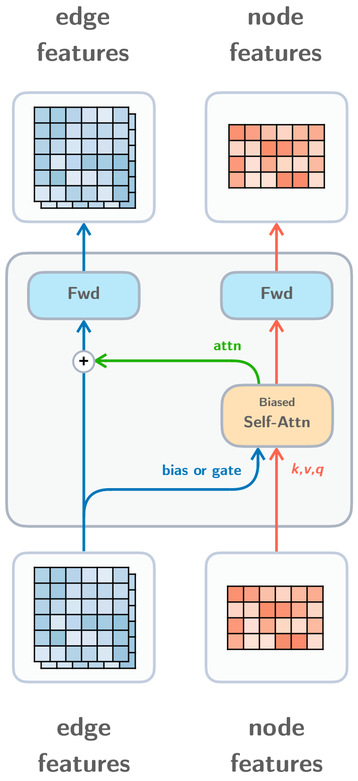
Attention matrix has the same shape as the edge feature matrix.
Each cell refers to a node–node relationship, i.e. an edge.

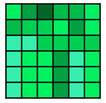
Graph Transformer

Interpreted as a 2-track transformer

to maintain and refine *node* and *edge* representations

Different tracks operate on different representations.





Attention matrix has the same shape as the edge feature matrix.
Each cell refers to a node–node relationship, i.e. an edge.

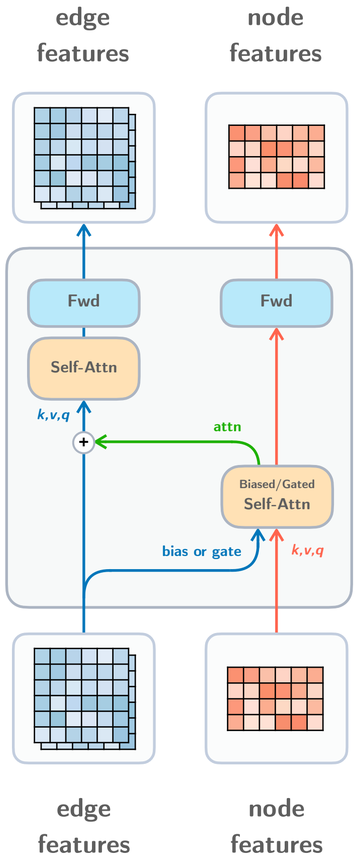
Graph Transformer

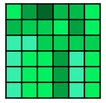
Interpreted as a 2-track transformer

to maintain and refine *node* and *edge* representations

Different tracks operate on different representations.

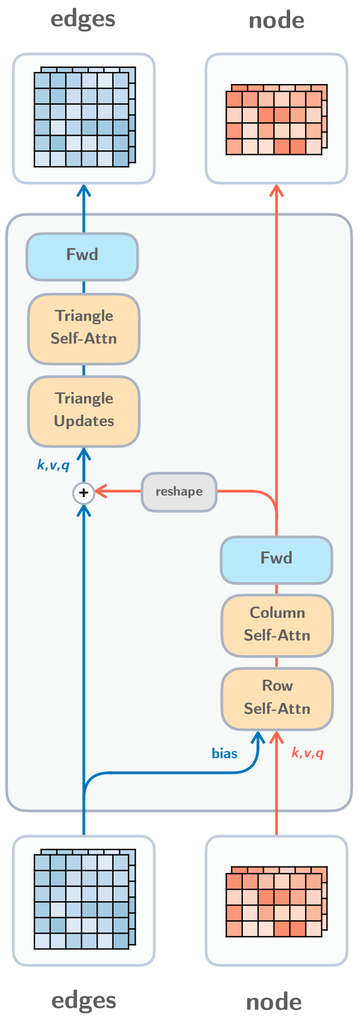
You could also do self-attention to the edge features directly.



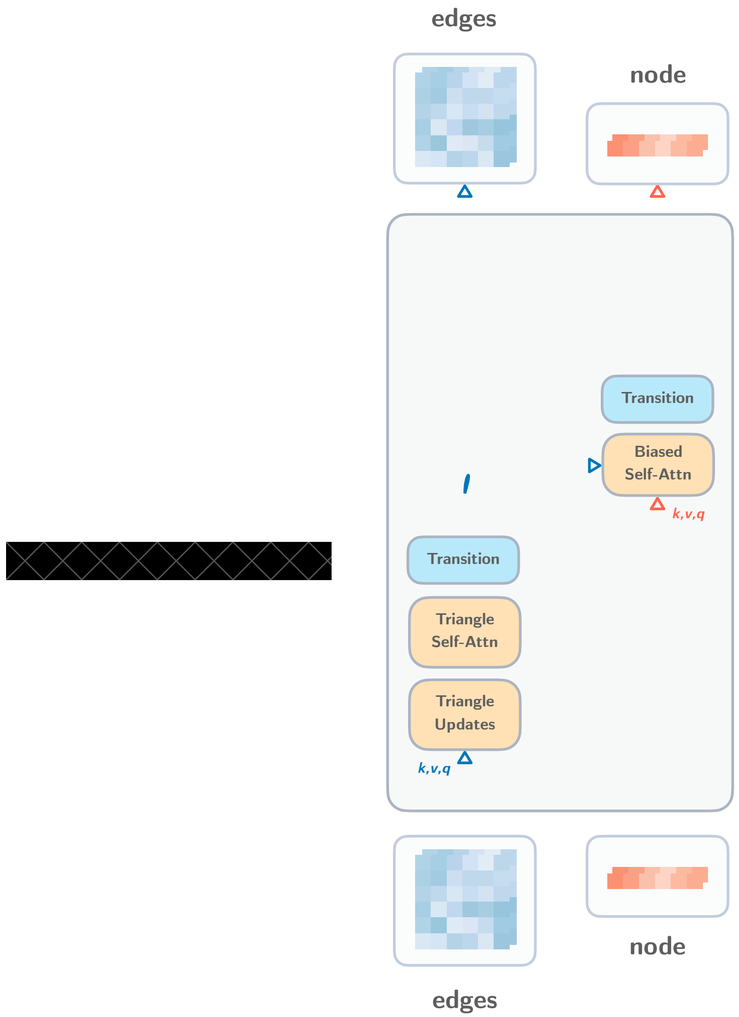


Attention matrix has the same shape as the edge feature matrix.
Each cell refers to a node–node relationship, i.e. an edge.

AlphaFold2 (Evoformer)

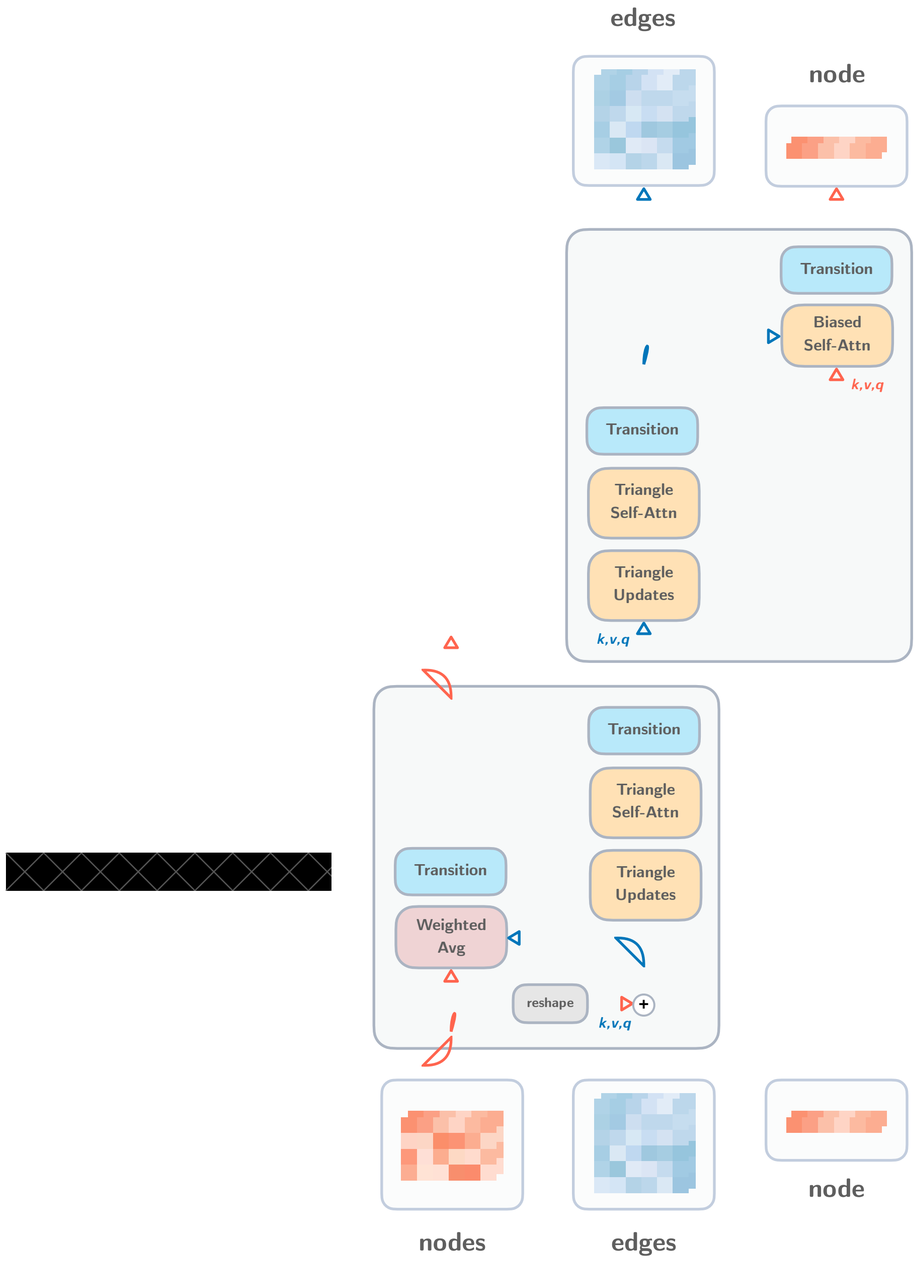


AlphaFold3 (Pairformer)

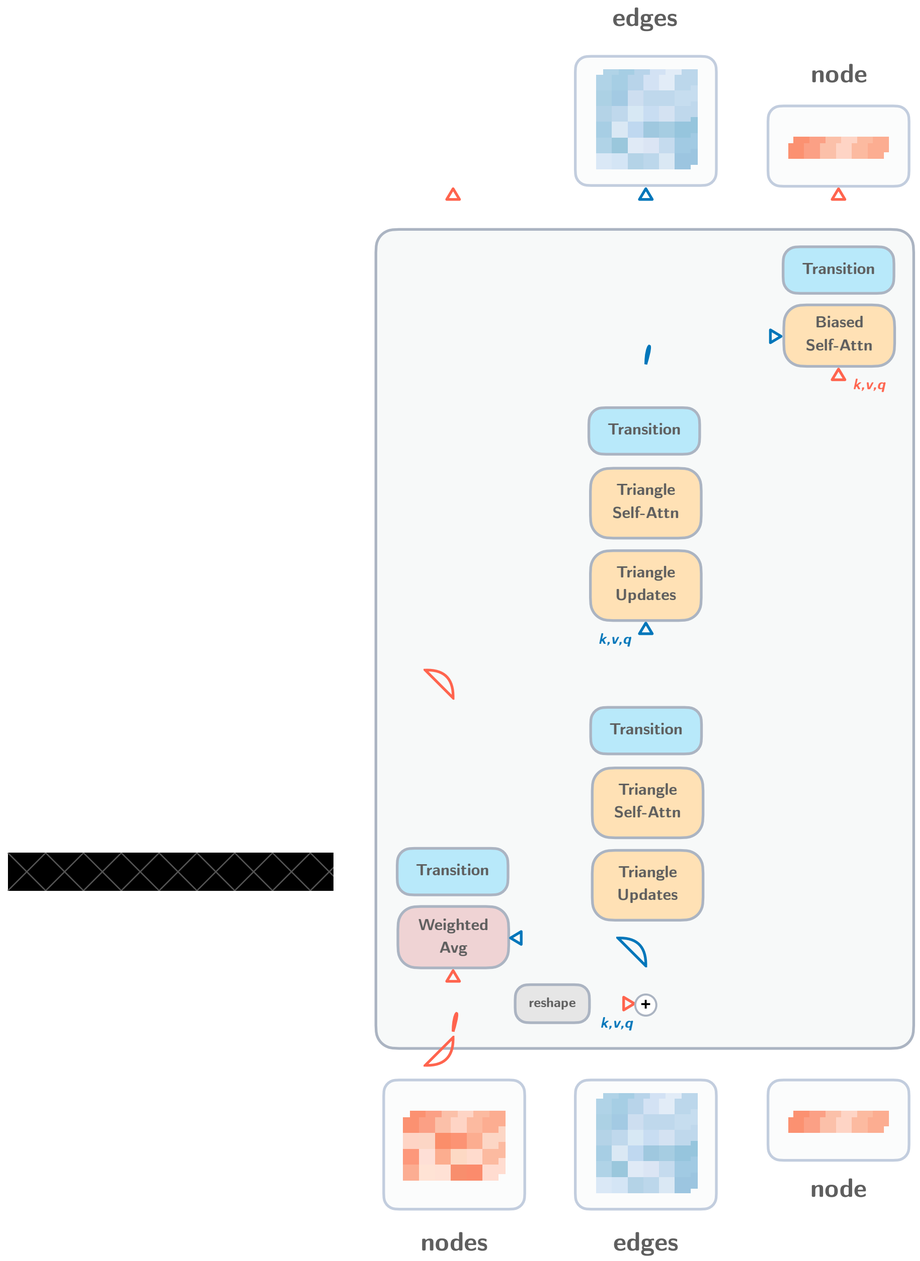


Communication only from left to right.

AlphaFold3 (MSA Module + Pairformer)

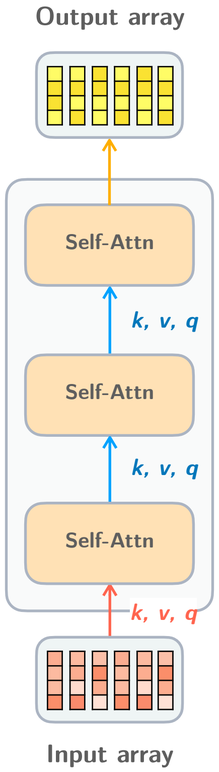


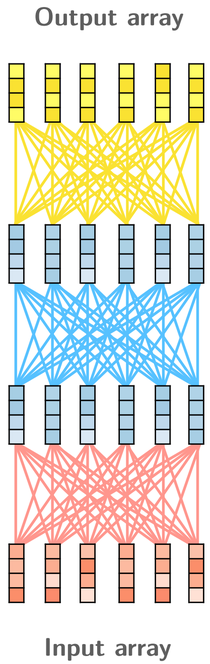
AlphaFold3 (MSA Module + Pairformer)



Perceiver

(Latent Transformer)


Output array
Self-Attn
k, v, q
Self-Attn
k, v, q
Self-Attn
k, v, q
Input array

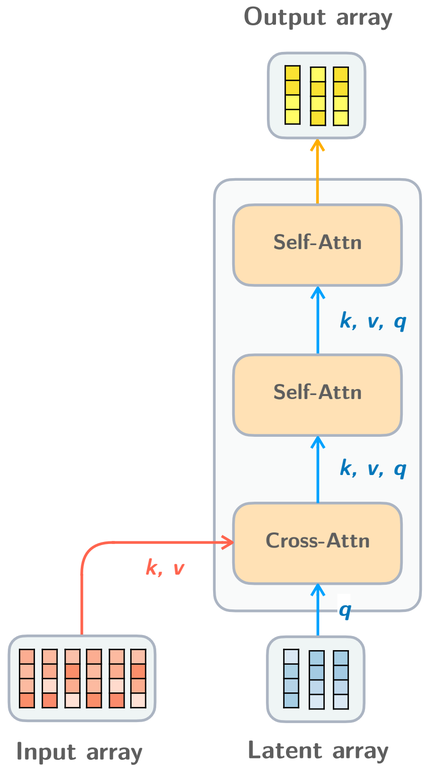

Output array
Input array

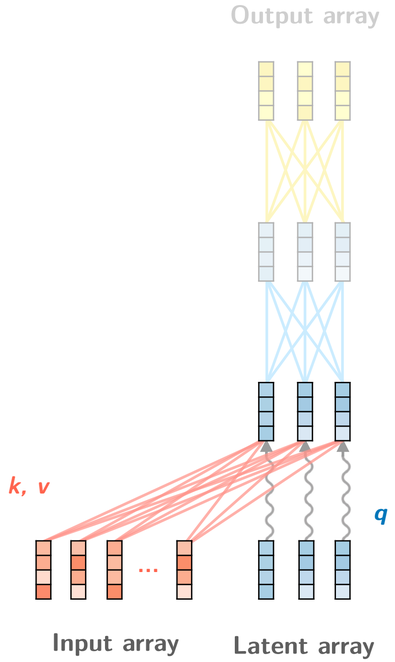
Perceiver

"Latent transformer"

It's a very general-purpose architecture.

Weights are optionally shared across layers.
If so, it's basically a *vertical RNN*,
i.e. recurrence in depth rather than time.



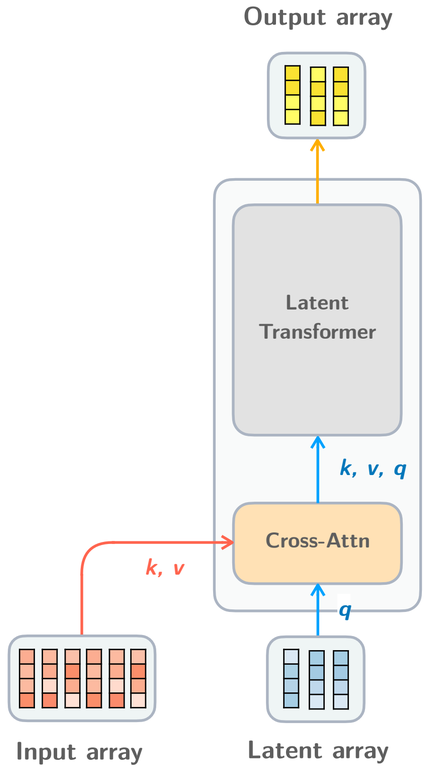


Perceiver

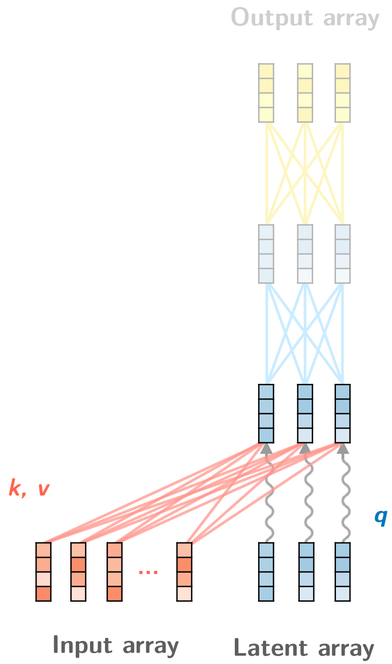
"Latent transformer"

It's a very general-purpose architecture.

Weights are optionally shared across layers.
If so, it's basically a *vertical RNN*,
i.e. recurrence in depth rather than time.



Self-attention operating only on latents

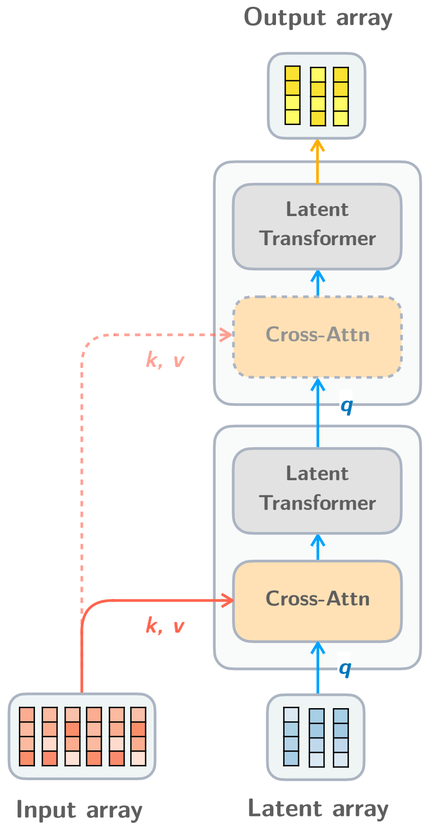


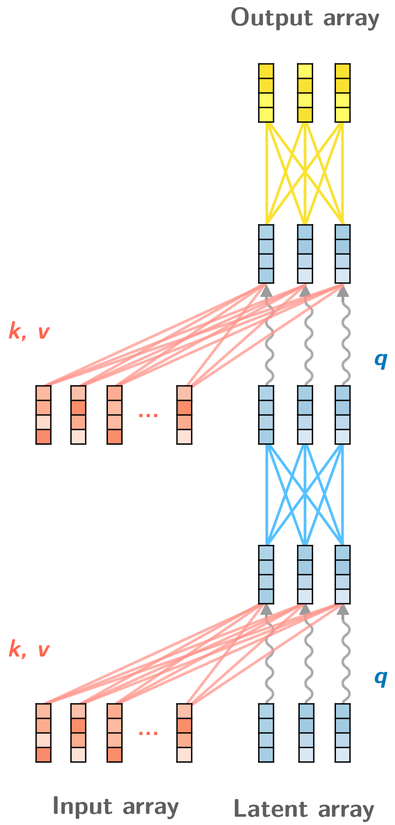
Perceiver

"Latent transformer"

It's a very general-purpose architecture.

Weights are optionally shared across layers.
If so, it's basically a *vertical RNN*,
i.e. recurrence in depth rather than time.



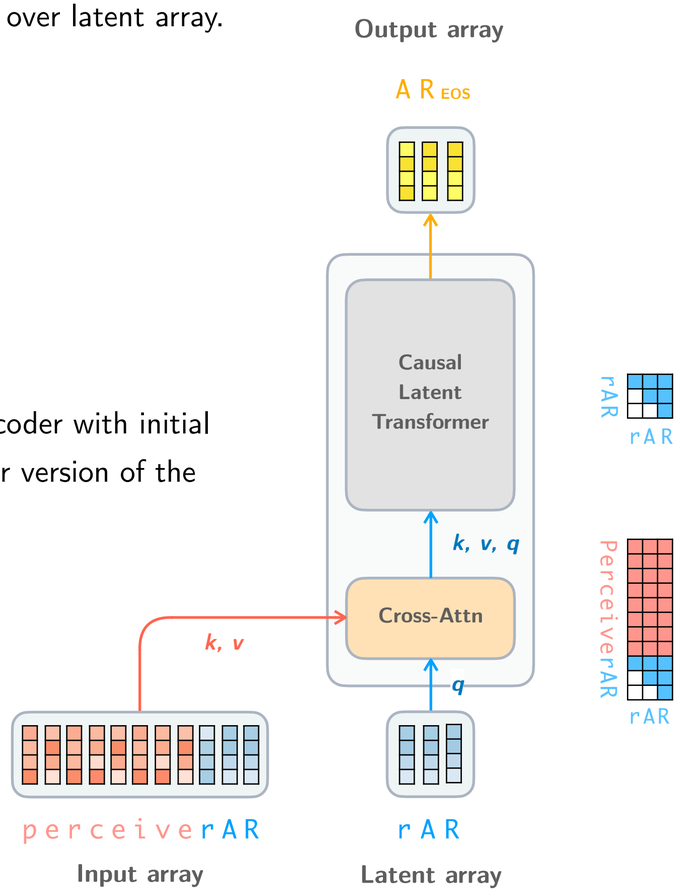


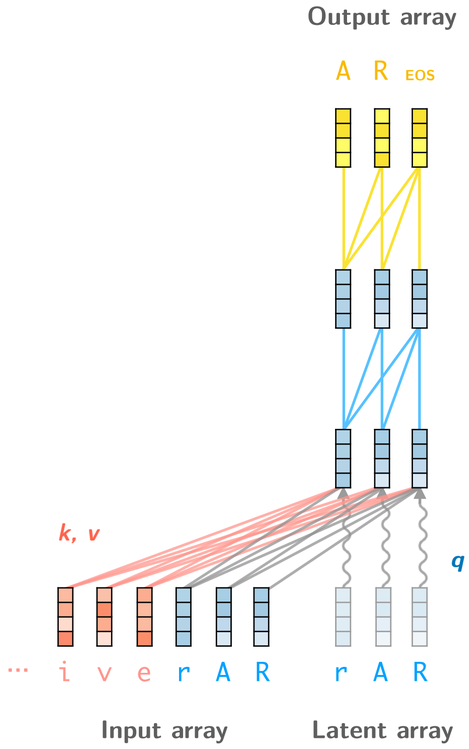
Perceiver AR

Applied to autoregressive modeling.

Each latent is associated with an output.
We can still vary the number of latents as before.
Latent self-attention is also causal now.
Input array can be much larger than the latent array.
We only do deep self-attention over latent array.

Basically just a transformer decoder with initial cross-attention layer to a longer version of the autoregressive array?



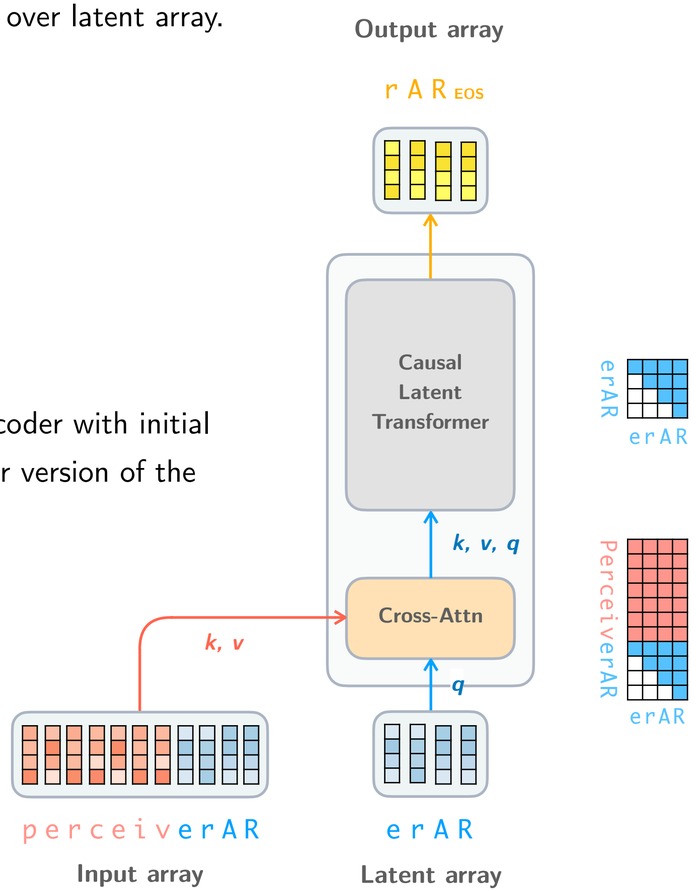


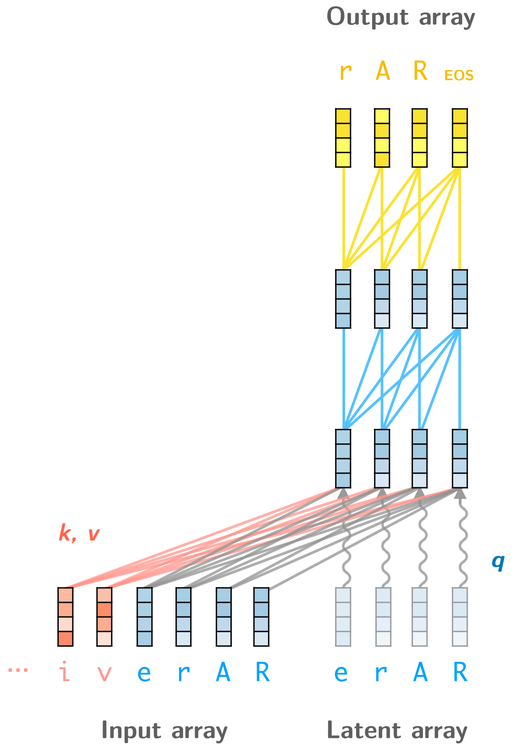
Perceiver AR

Applied to autoregressive modeling.

Each latent is associated with an output.
We can still vary the number of latents as before.
Latent self-attention is also causal now.
Input array can be much larger than the latent array.
We only do deep self-attention over latent array.

Basically just a transformer decoder with initial cross-attention layer to a longer version of the autoregressive array?



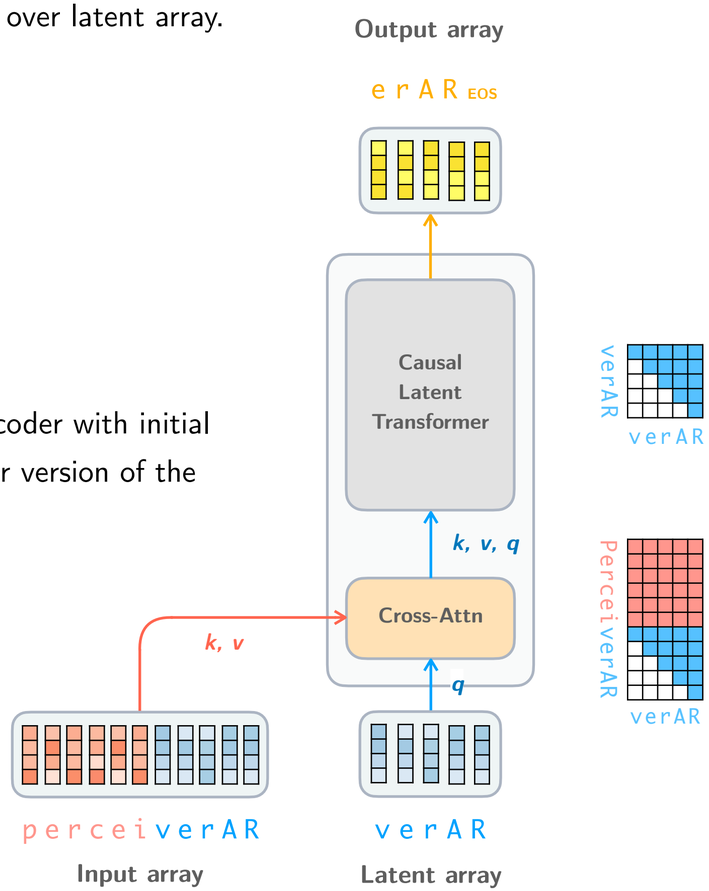


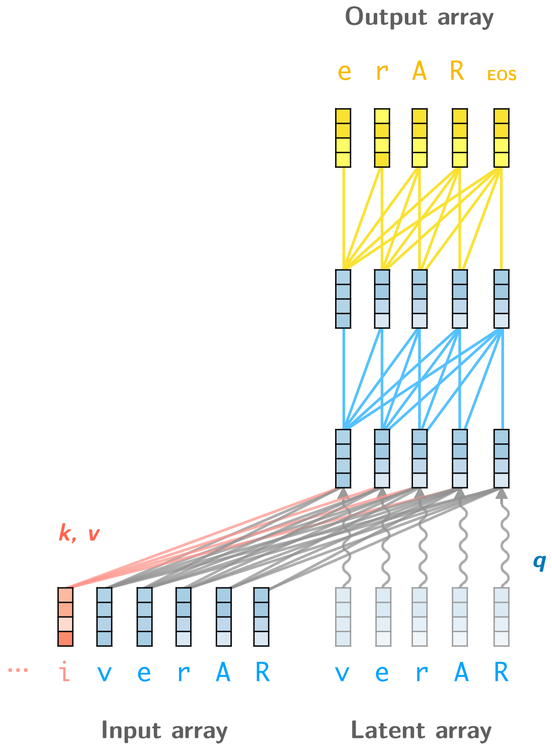
Perceiver AR

Applied to autoregressive modeling.

Each latent is associated with an output.
We can still vary the number of latents as before.
Latent self-attention is also causal now.
Input array can be much larger than the latent array.
We only do deep self-attention over latent array.

Basically just a transformer decoder with initial cross-attention layer to a longer version of the autoregressive array?

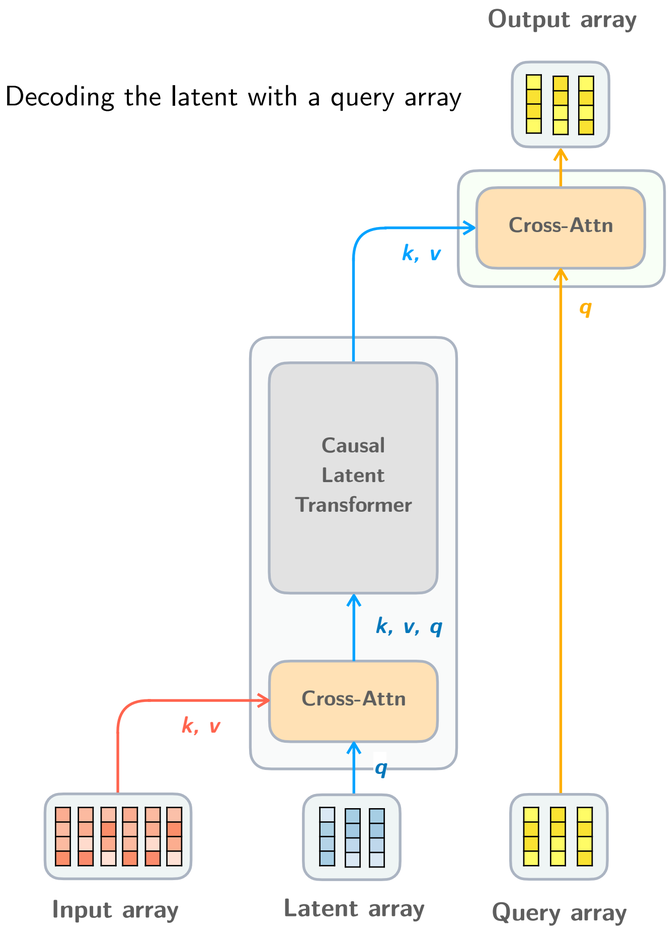


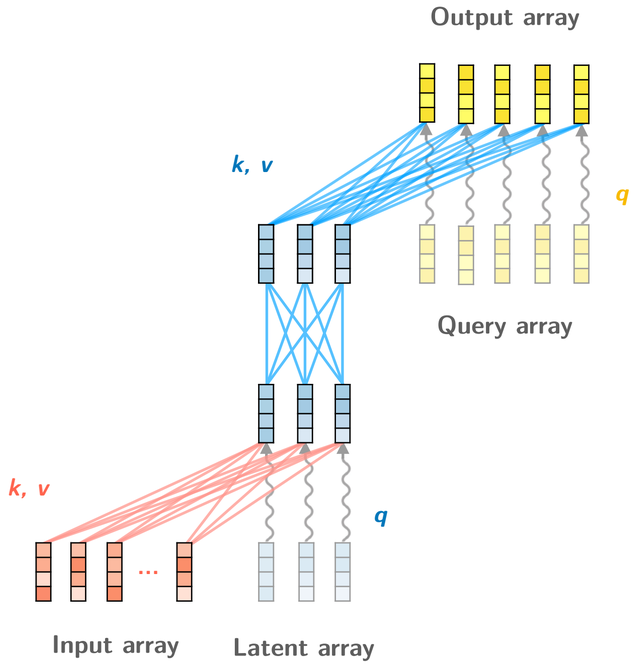


Perceiver IO

Even more general-purpose.

Better for outputs more complicated than, say,
simple classification or regression.

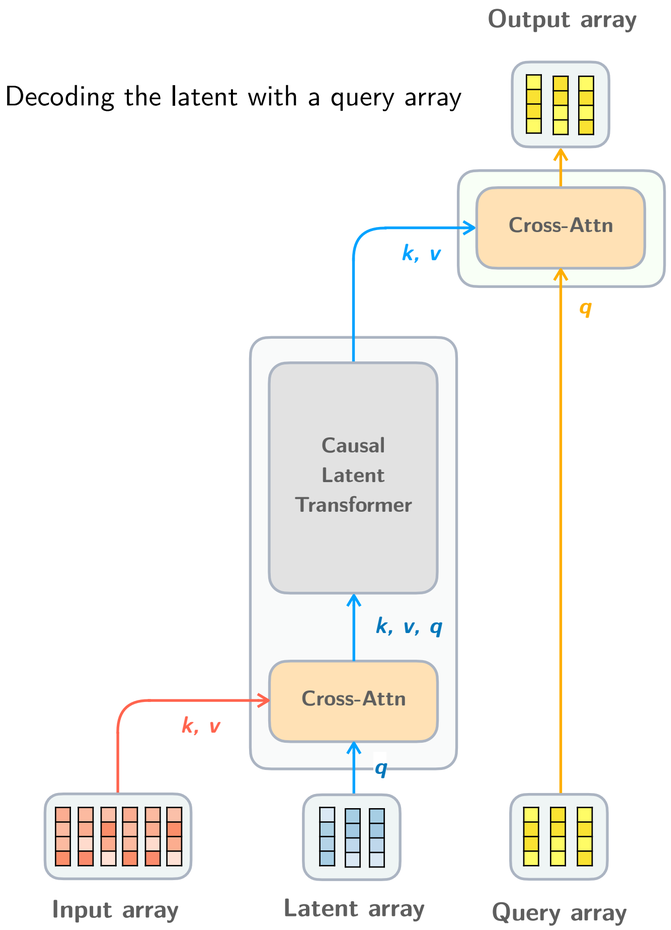


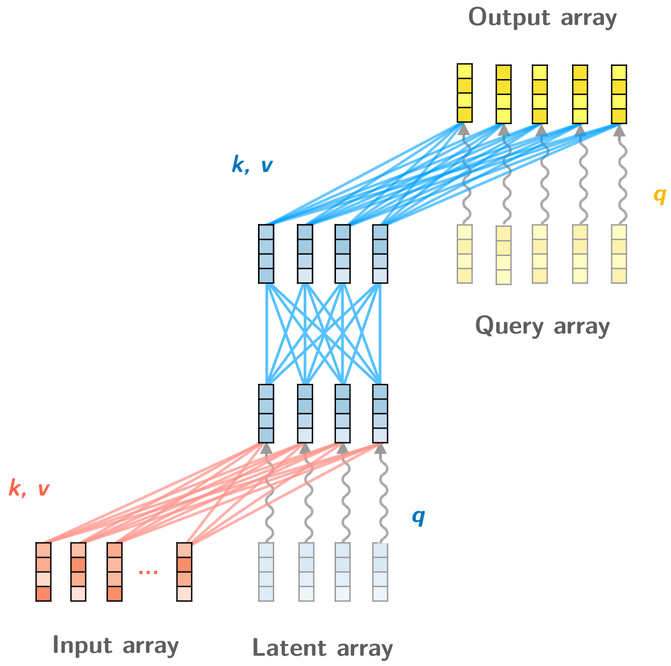


Perceiver IO

Even more general-purpose.

Better for outputs more complicated than, say,
simple classification or regression.



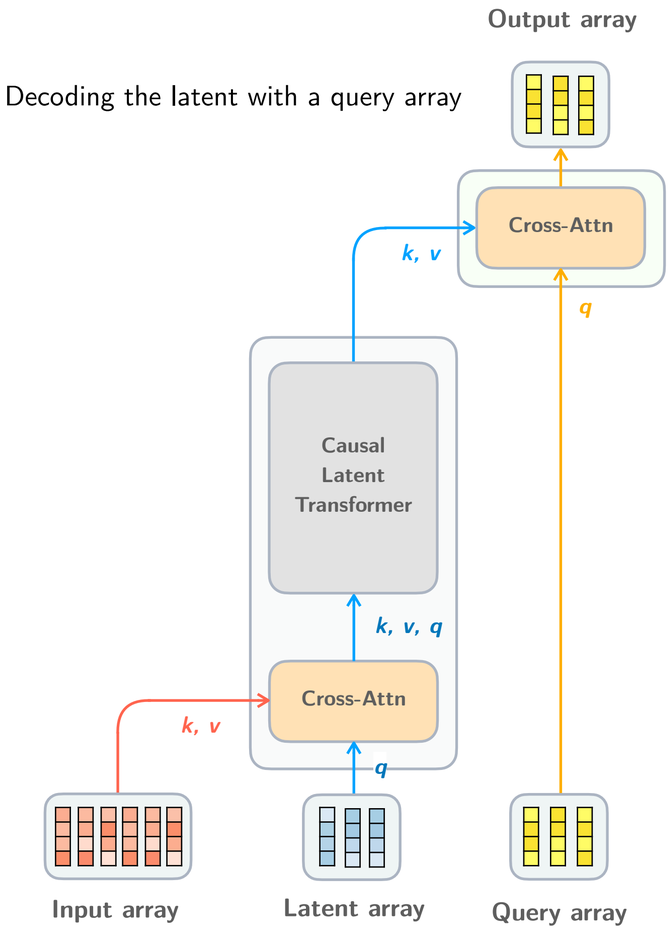


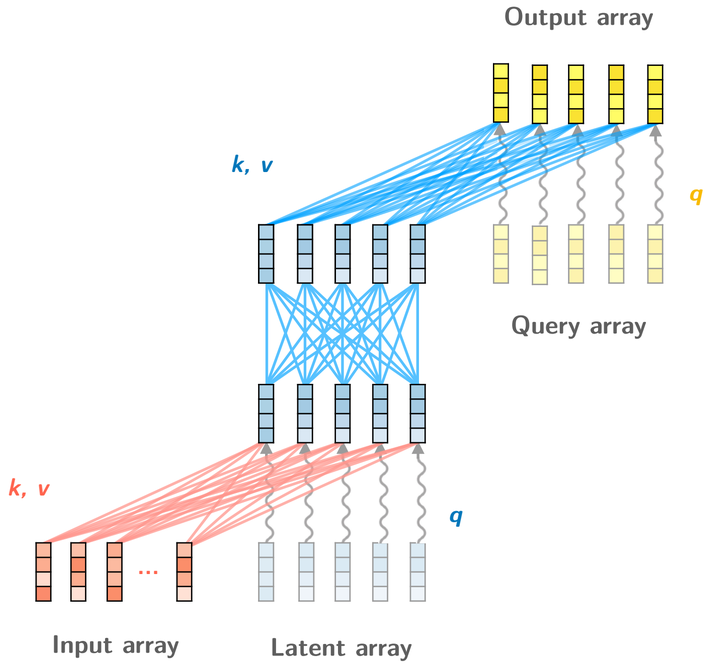
Increasing the "latent-array bandwidth"
(Not the same as increasing the bandwidth of each token,
which is the dimensionality of the latents)

Perceiver IO

Even more general-purpose.

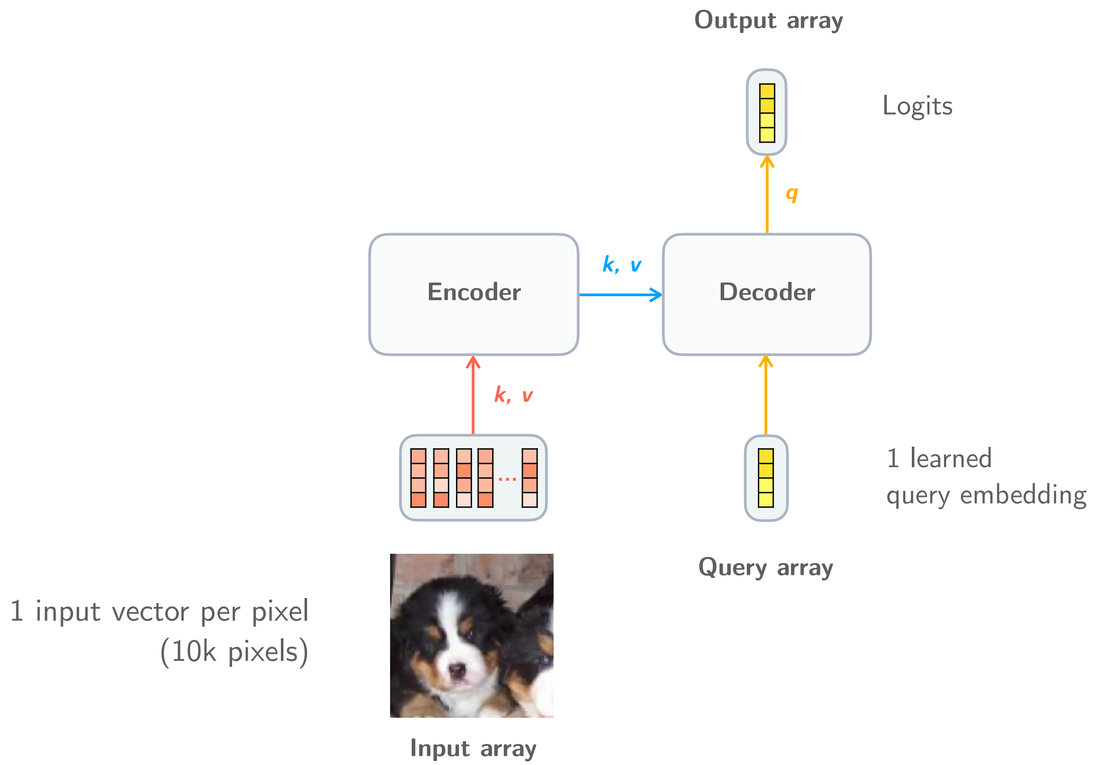
Better for outputs more complicated than, say,
simple classification or regression.





Increasing the "latent-array bandwidth"
(Not the same as increasing the bandwidth of each token,
which is the dimensionality of the latents)

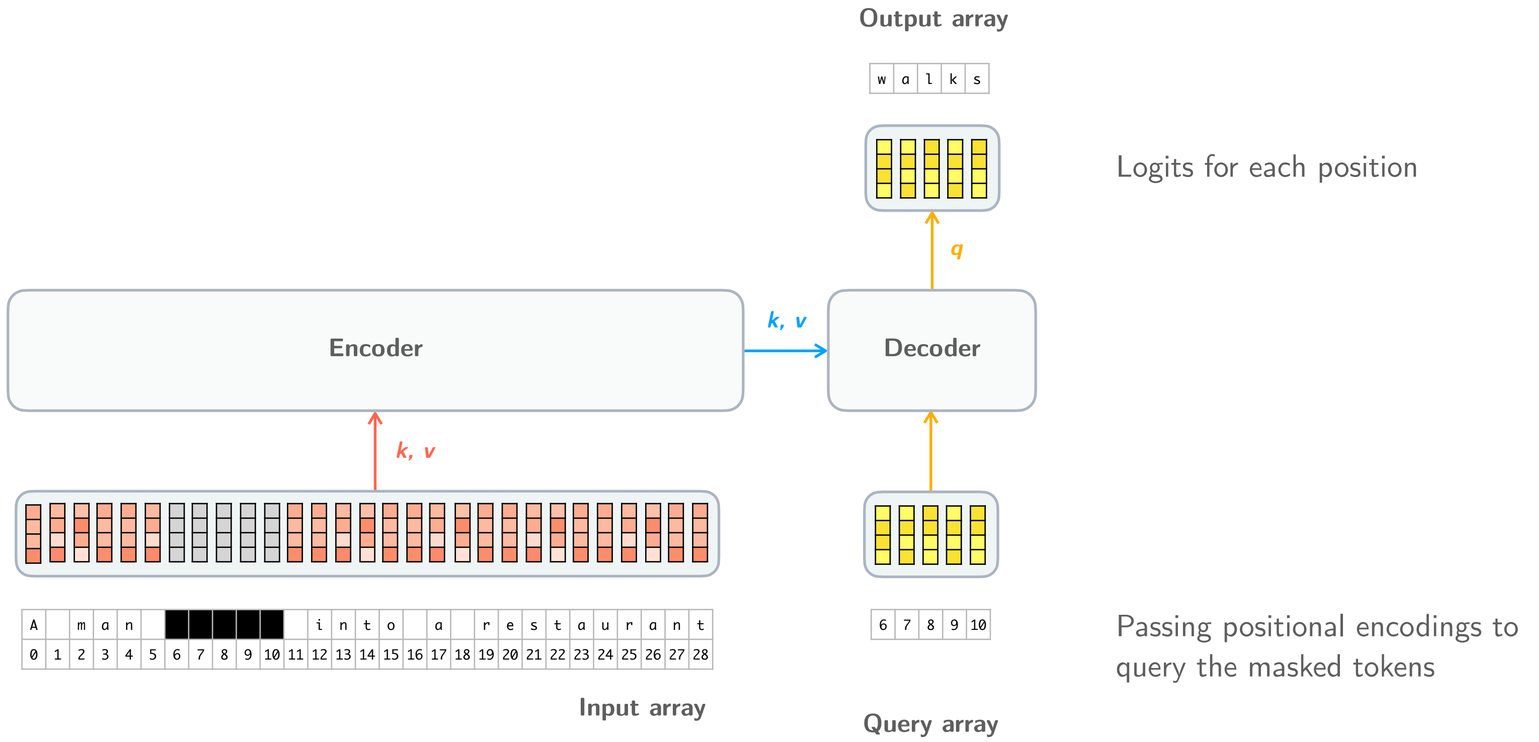
Example: Classification



Perceiver IO

Example: Masked Language Modeling

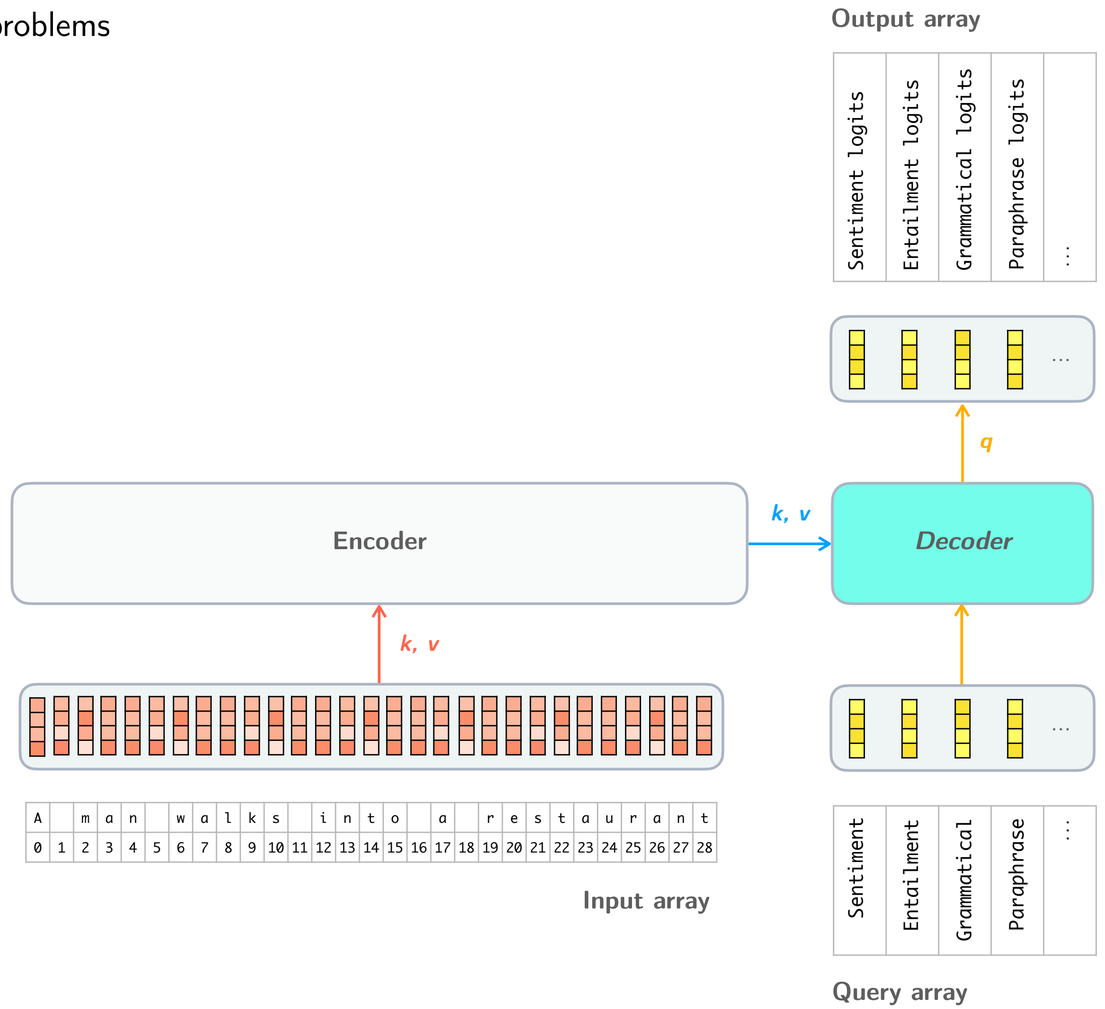
Similar for other spatial or sequence-structured arrays.



Perceiver IO

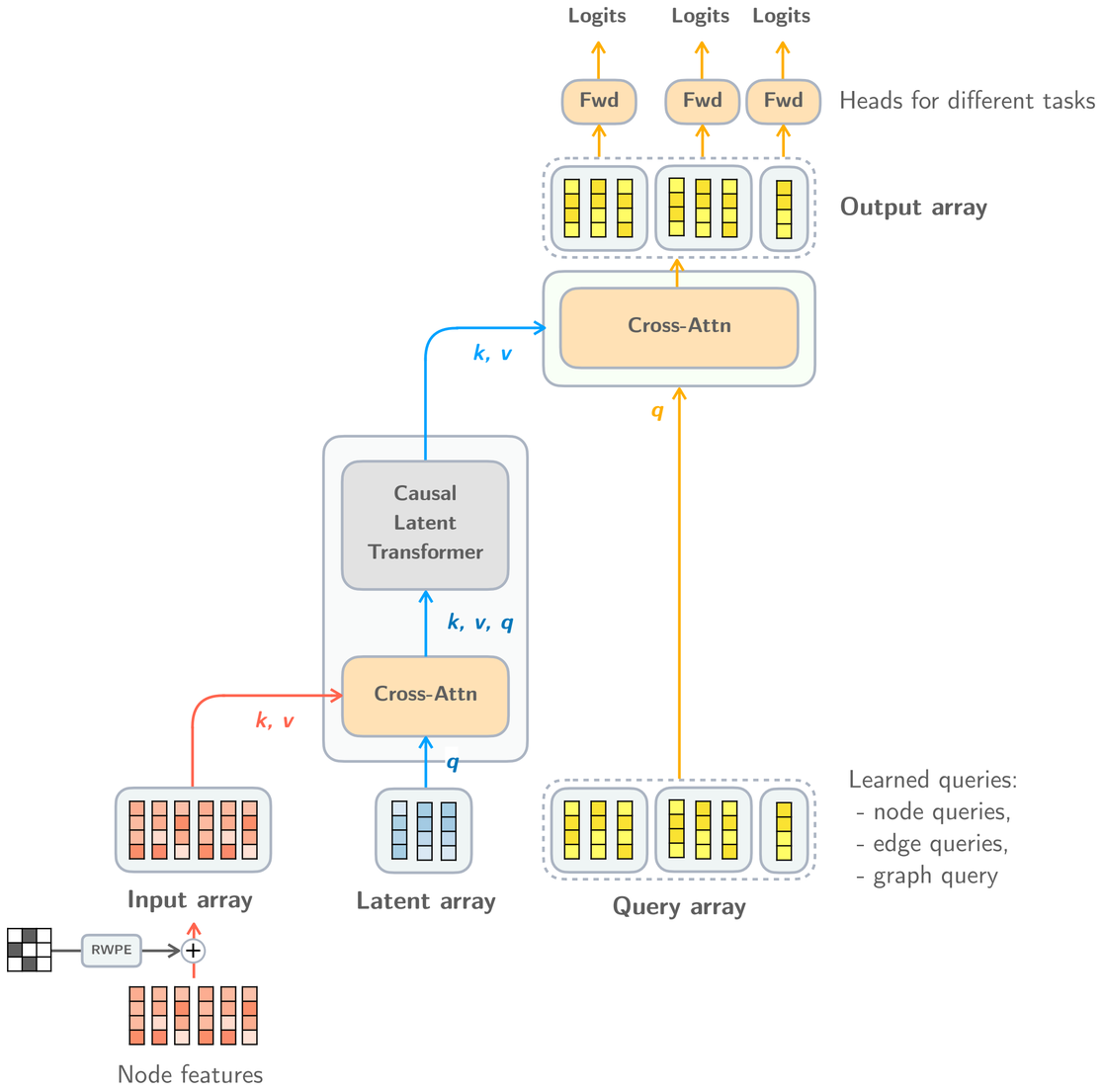
Example: Masked Language Modeling → GLUE fine-tuning

Similar for other multi-task problems



For multi-task in general: each task has its own token, and queries the latent array to get the output.

Graph Perceiver IO



Thanks for your *attention*.