C. Attention and Transformers

C.1 Origins of attention

C.2 Attention as a standalone operator

C.3 Transformer architectures

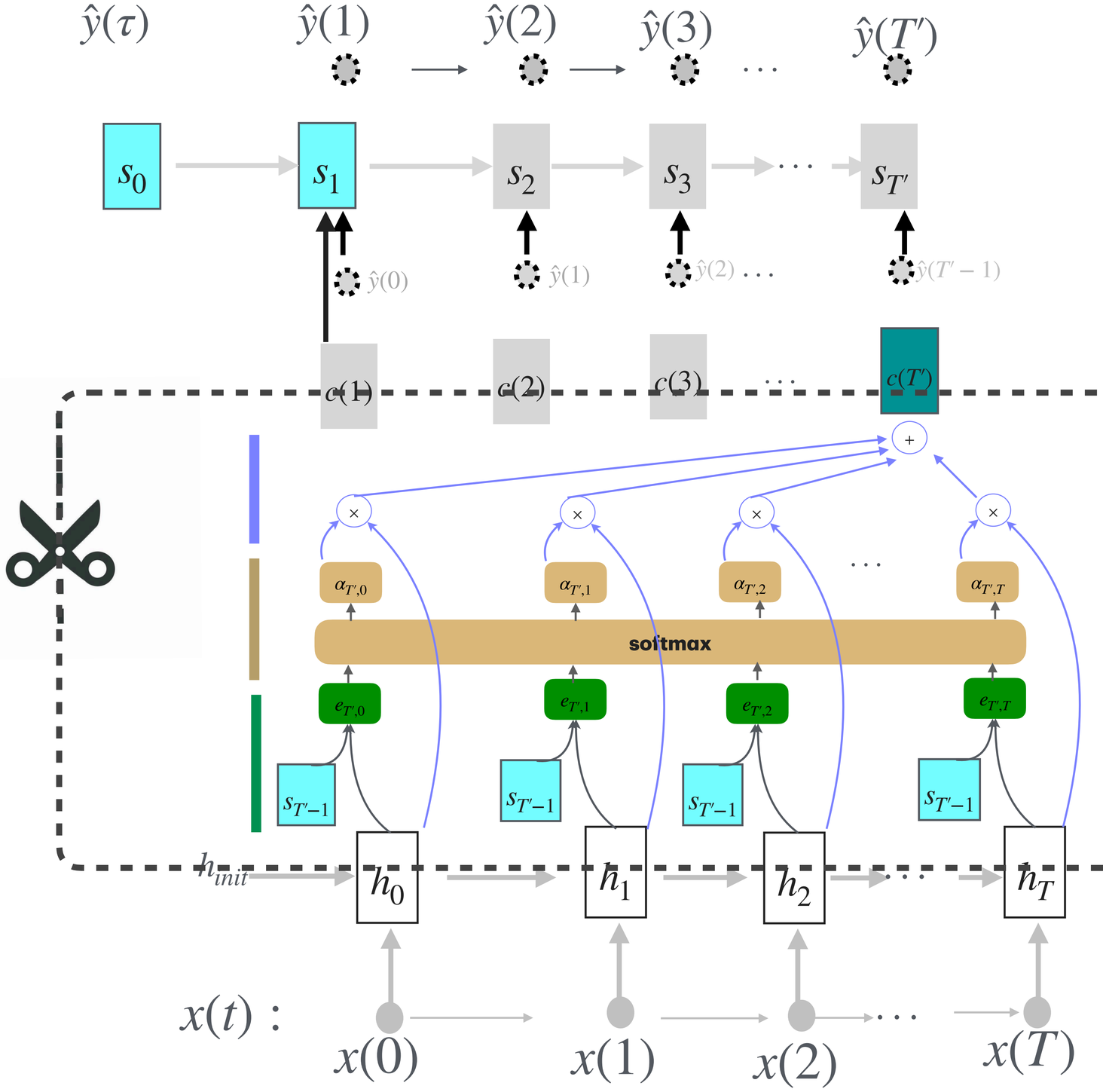
C.4 Transformers across different data structures

C.5 Training transformers

We will see how attention evolved beyond its original sequence-to-sequence setting into a more general building block.

Tiam Heydari, PhD
Postdoctoral Fellow, UBC

- In seq2seq, attention was introduced to help the decoder look back at the encoder states
- People realized this "attention block" can be separated from the recurrent architecture
- the core mechanism of attention itself is more general: **score**, **softmax**, **weighted sum**,(selective aggregation weighted by relevance).
- This suggests attention can be detached from the RNN and reused as its own building block
- In this part (Part II), we reformulate attention in that general standalone form

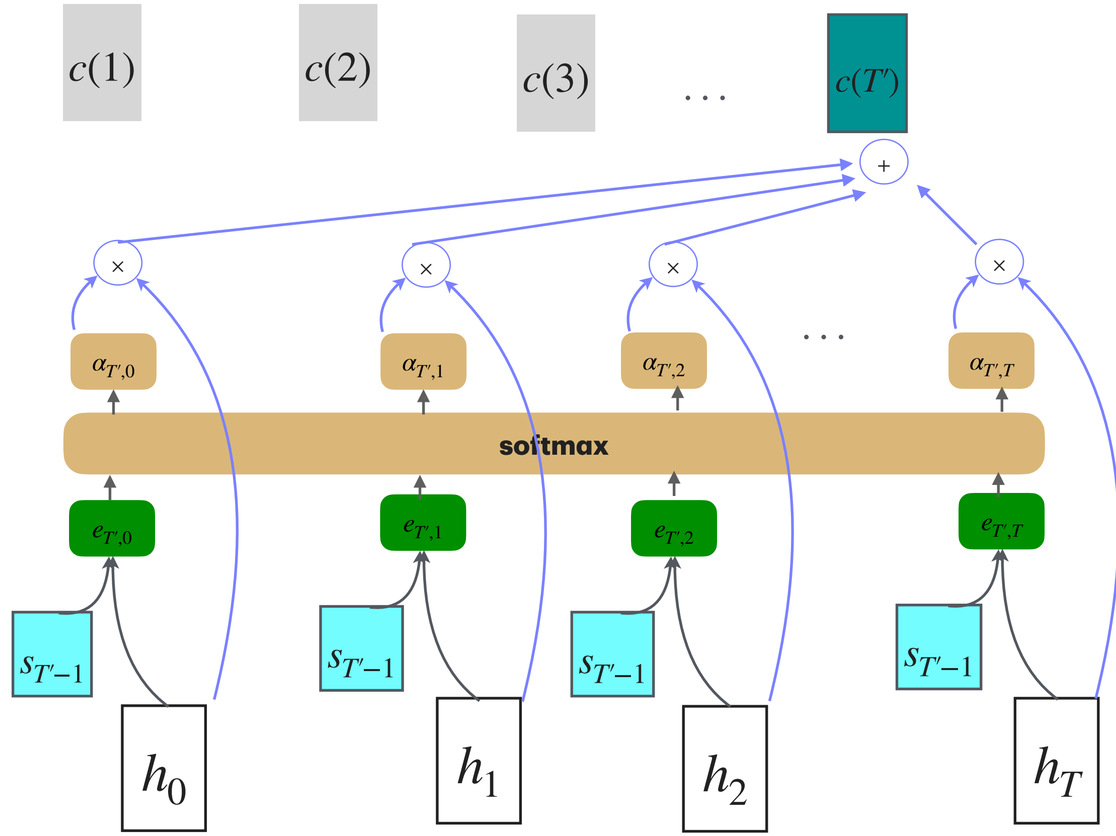


- Let's take this operation, and look at it carefully step by step:

$$c_i = \sum_j \alpha_{i,j} h_j$$

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_k \exp(e_{i,k})} = softmax_j(e_{i,j})$$

$$e_{i,j} = F(s_{i-1}, h_j)$$

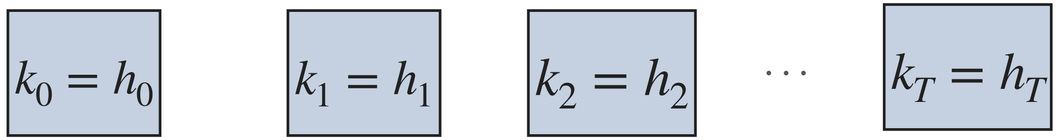


Step 1

We have a set of stored representations, call them **Keys (K)**

$k_j \in \{k_0, k_1, \ldots, k_T\}$ that encode the information we want to retrieve from.

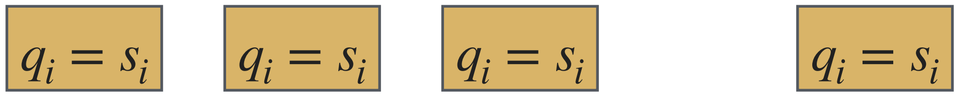
In seq2seq setting, they are encoder states $k_j = h_j$



Step 2

We also have a **Query (Q)**, q_i a vector representing what we are looking for.

In seq2seq setting, query is the decoder hidden $q_i = s_i$

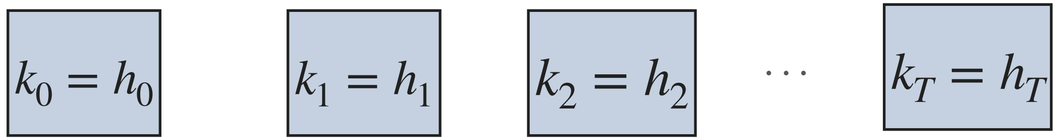


Step 1

We have a set of stored representations, call them **Keys (K)**

$k_j \in \{k_0, k_1, \ldots, k_T\}$ that encode the information we want to retrieve from.

In seq2seq setting, they are encoder states $k_j = h_j$



Step 3

Now compare the current Query q_i with each Keys $k_j \in \{k_0, k_1, \ldots, k_T\}$.

gives raw compatibility scores: $e_{i,j} = F(q_i, k_j)$

Step 2

We also have a **Query (Q)**, q_i a vector representing what we are looking for.

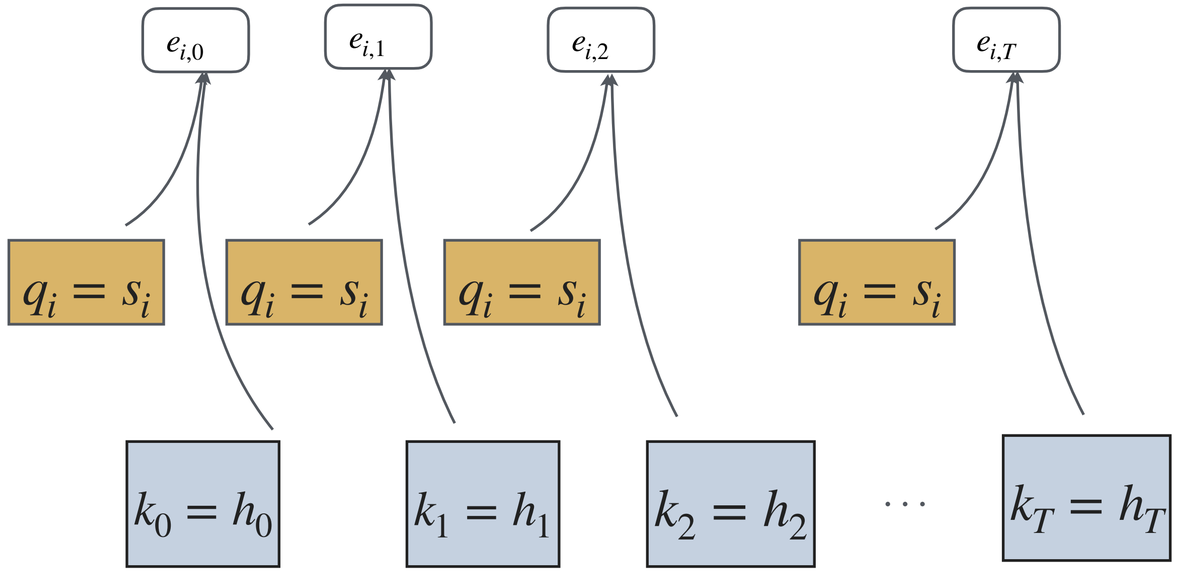
In seq2seq setting, query is the decoder hidden $q_i = s_i$

Step 1

We have a set of stored representations, call them **Keys (K)**

$k_j \in \{k_0, k_1, \ldots, k_T\}$ that encode the information we want to retrieve from.

In seq2seq setting, they are encoder states $k_j = h_j$



Step 4

Now apply softmax to the raw scores and get attention weights:

$\alpha_{i,j} = \text{softmax}_j(e_{i,j})$

Step 3

Now compare the current Query q_i with each Keys $k_j \in \{k_0, k_1, \ldots, k_T\}$.

gives raw compatibility scores: $e_{i,j} = F(q_i, k_j)$

Step 2

We also have a **Query (Q)**, q_i a vector representing what we are looking for.

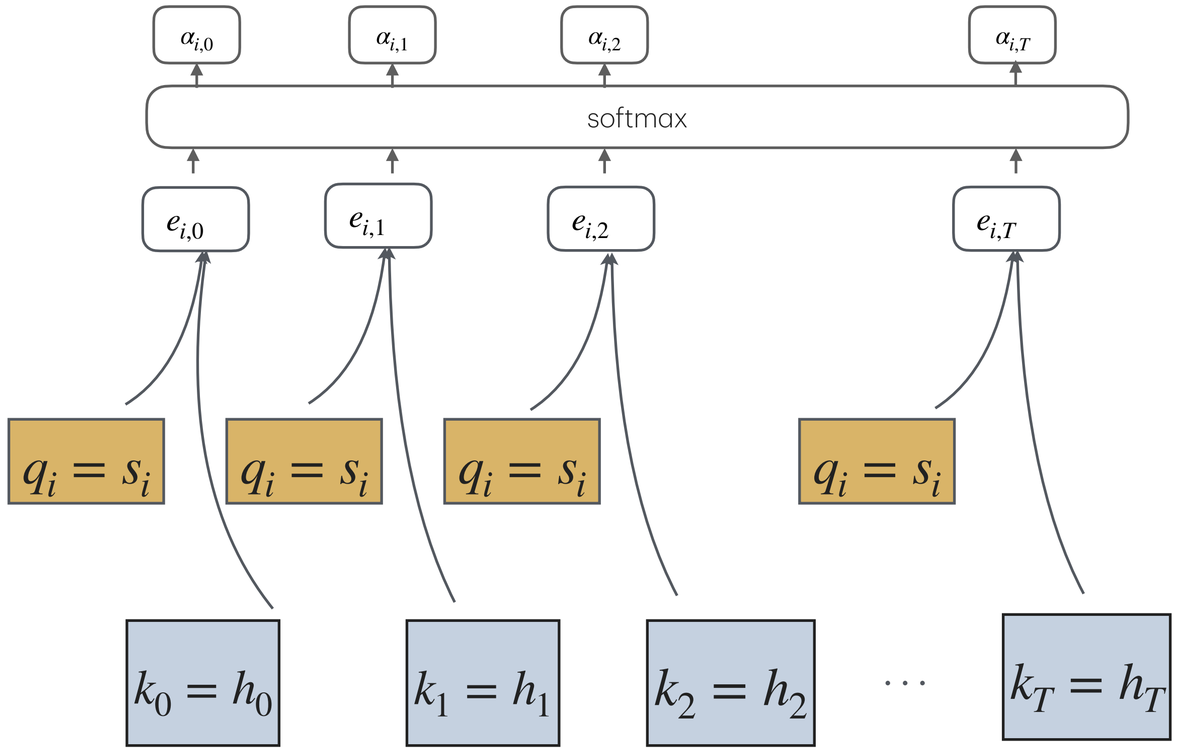
In seq2seq setting, query is the decoder hidden $q_i = s_i$

Step 1

We have a set of stored representations, call them **Keys (K)**

$k_j \in \{k_0, k_1, \ldots, k_T\}$ that encode the information we want to retrieve from.

In seq2seq setting, they are encoder states $k_j = h_j$



Step5

Now use the attention weights $\alpha_{i,j}$ to combine the corresponding **Values (V)** and form the final output for the current Query:

$$c_i = \sum_j \alpha_{i,j} v_j$$

In this original seq2seq setting, the same stored vectors play both roles:

$$v_j = h_j$$

Step 4

Now apply softmax to the raw scores and get attention weights:

$$\alpha_{i,j} = \text{softmax}_j(e_{i,j})$$

Step 3

Now compare the current Query q_i with each Keys $k_j \in \{k_0, k_1, \ldots, k_T\}$.

gives raw compatibility scores: $e_{i,j} = F(q_i, k_j)$

Step 2

We also have a **Query (Q),** q_i a vector representing what we are looking for.

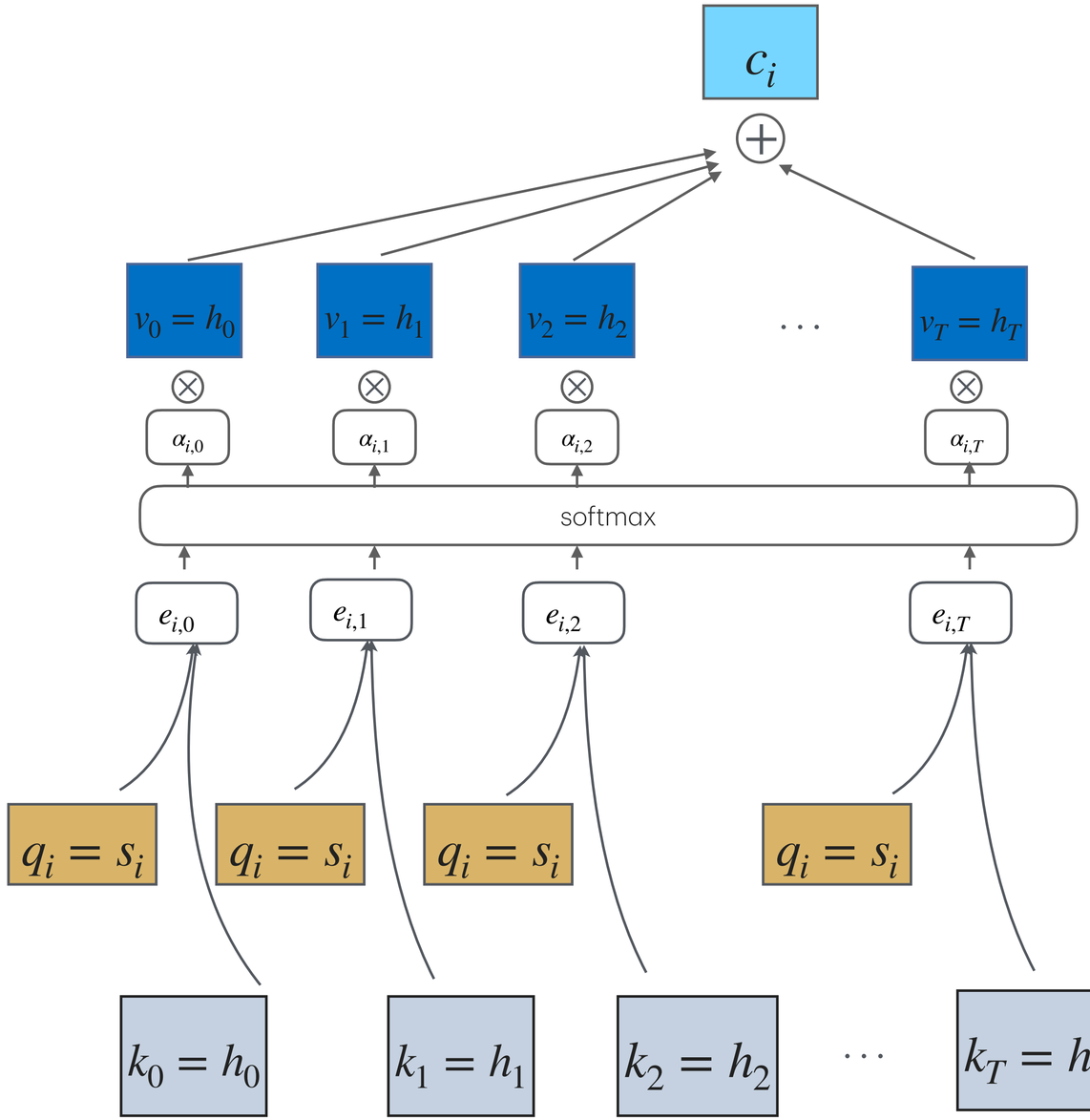
In seq2seq setting, query is the decoder hidden $q_i = s_i$

Step 1

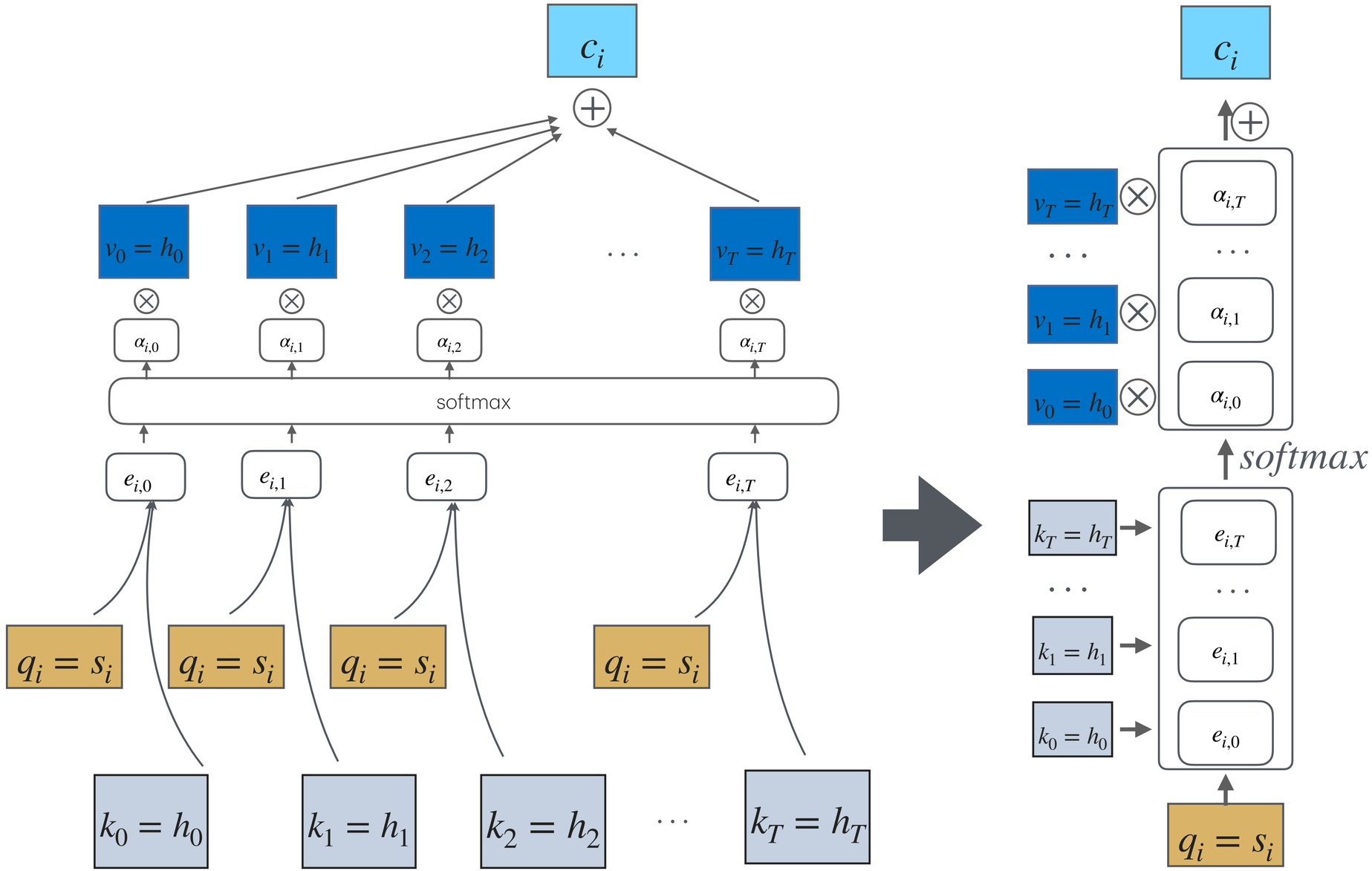
We have a set of stored representations, call them **Keys (K)**

$k_j \in \{k_0, k_1, \ldots, k_T\}$ that encode the information we want to retrieve from.

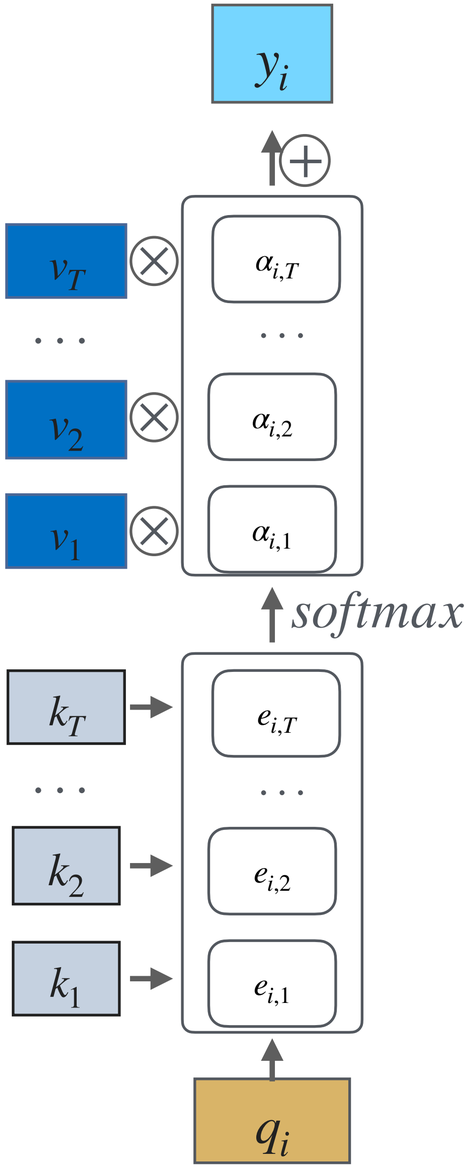
In seq2seq setting, they are encoder states $k_j = h_j$



We can now simplify the diagram, since we understand the roles of its components:



- Now let's rewrite of each step with all seq2seq references removed



- ***Note that I re-index everything to start from index 1, and not zero anymore.

- Let us now remove the seq2seq-specific interpretation
- Recall that in the original seq2seq setting, the same vector played both roles:

$$k_j = v_j = h_j.$$

The same h_j was used to match against the query (as a Key) and to retrieve information (as a Value). There is no reason these two roles should be tied together — matching and retrieving are different operations

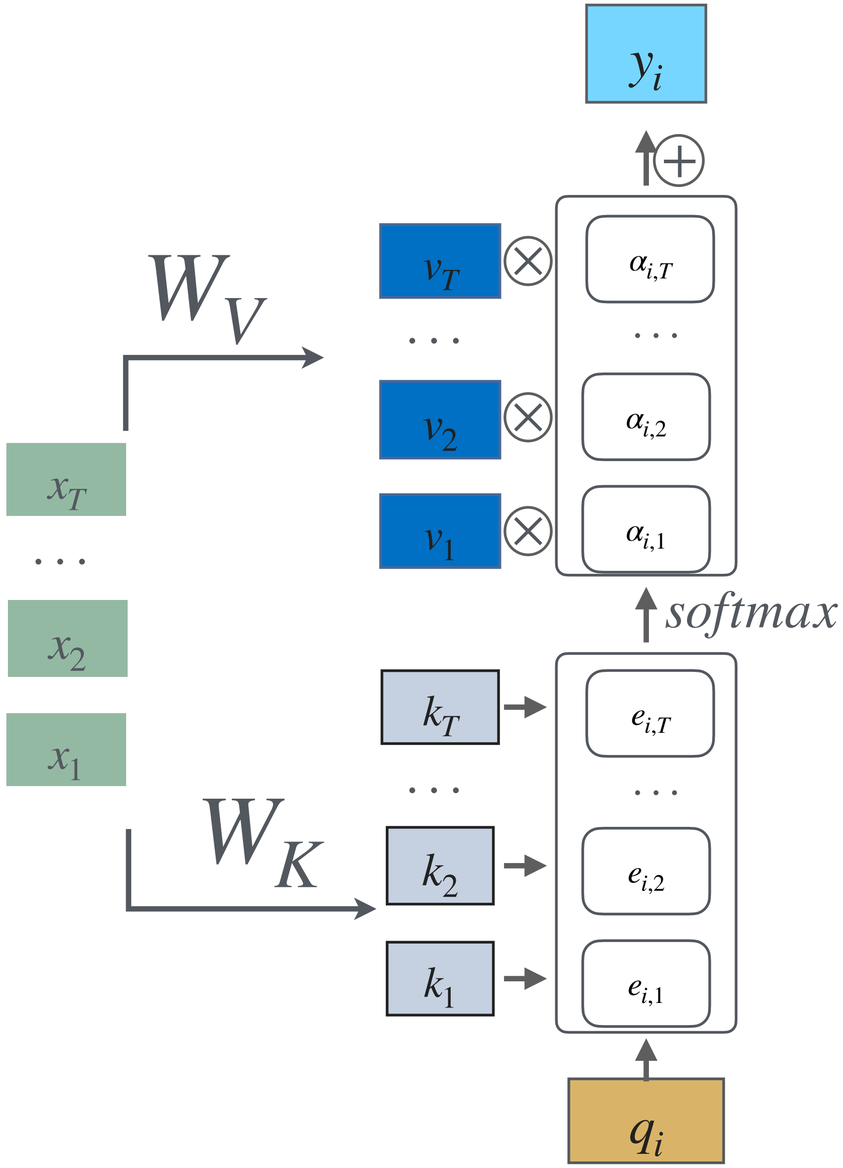
More generally, both Keys and Values can be learned linear projections of any set of vectors $\{x_0, x_1, \ldots, x_T\}$ that we want to query about. This set is called input representations.

$$k_j = W_K x_j$$

$$v_j = W_V x_j$$

where W_K and W_V are matrices that are learned independently

**"we will generalize the Query in the same way next

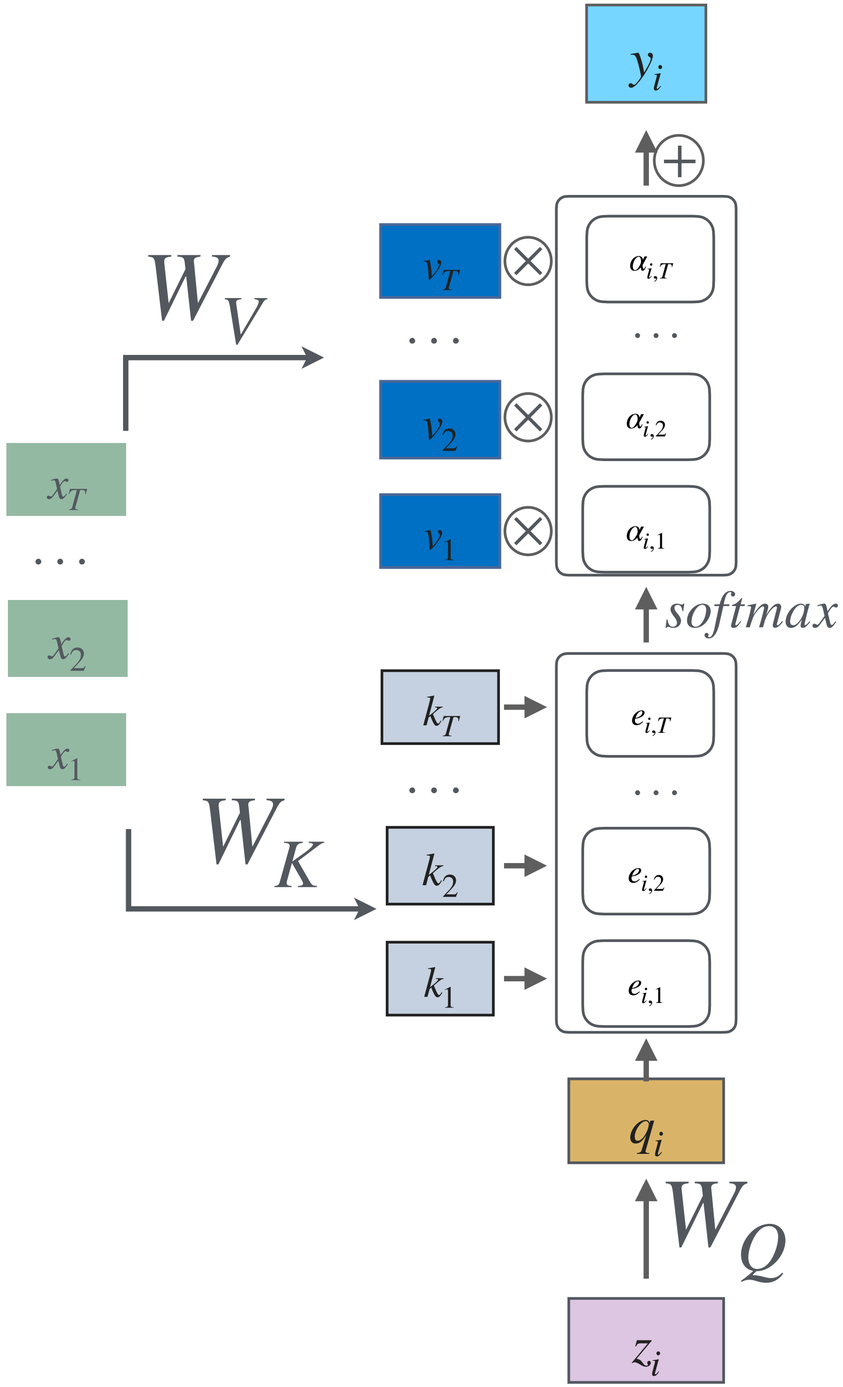


- We can apply the same logic to the Query. In the original seq2seq setting, the query was used raw ($q_i = s_i$)

- There is no reason the query should not also be a learned projection. More generally, the Query can be a learned linear projection of any vector z_i that represents what we are currently looking for:

$$q_i = W_Q z_i$$

where W_Q is a matrix learned independently.



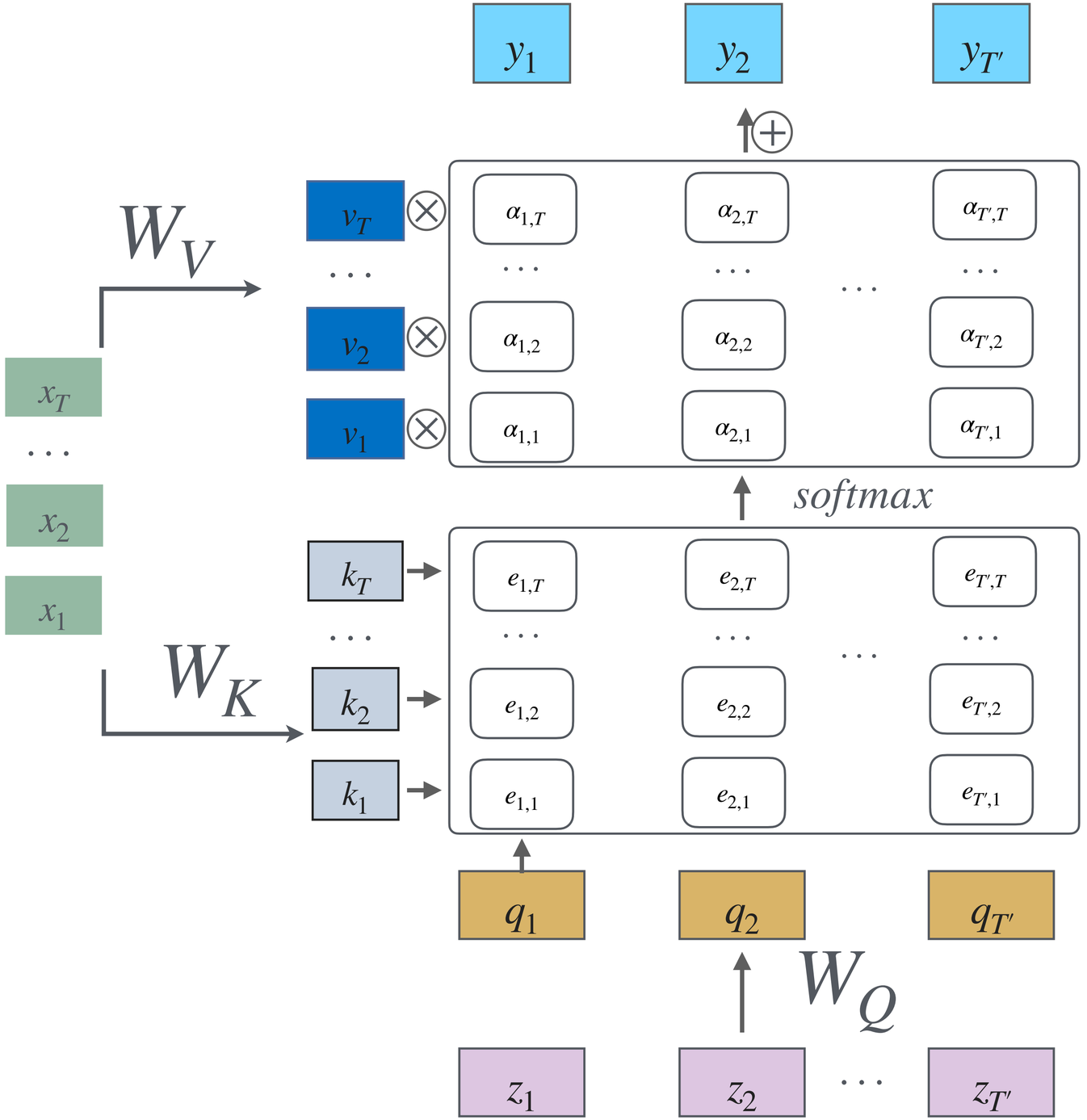
- So far, we illustrated the computation for one Query z_i

- But in the original setting (remember seq2seq example), we actually had many source vectors from which queries are built from:

Source vectors: $\{z_1, z_2, \ldots, z_{T'}\}$

Queries: $q_i = W_Q z_i$

- ***Note that I re-index everything to start from index 1, and not zero anymore.



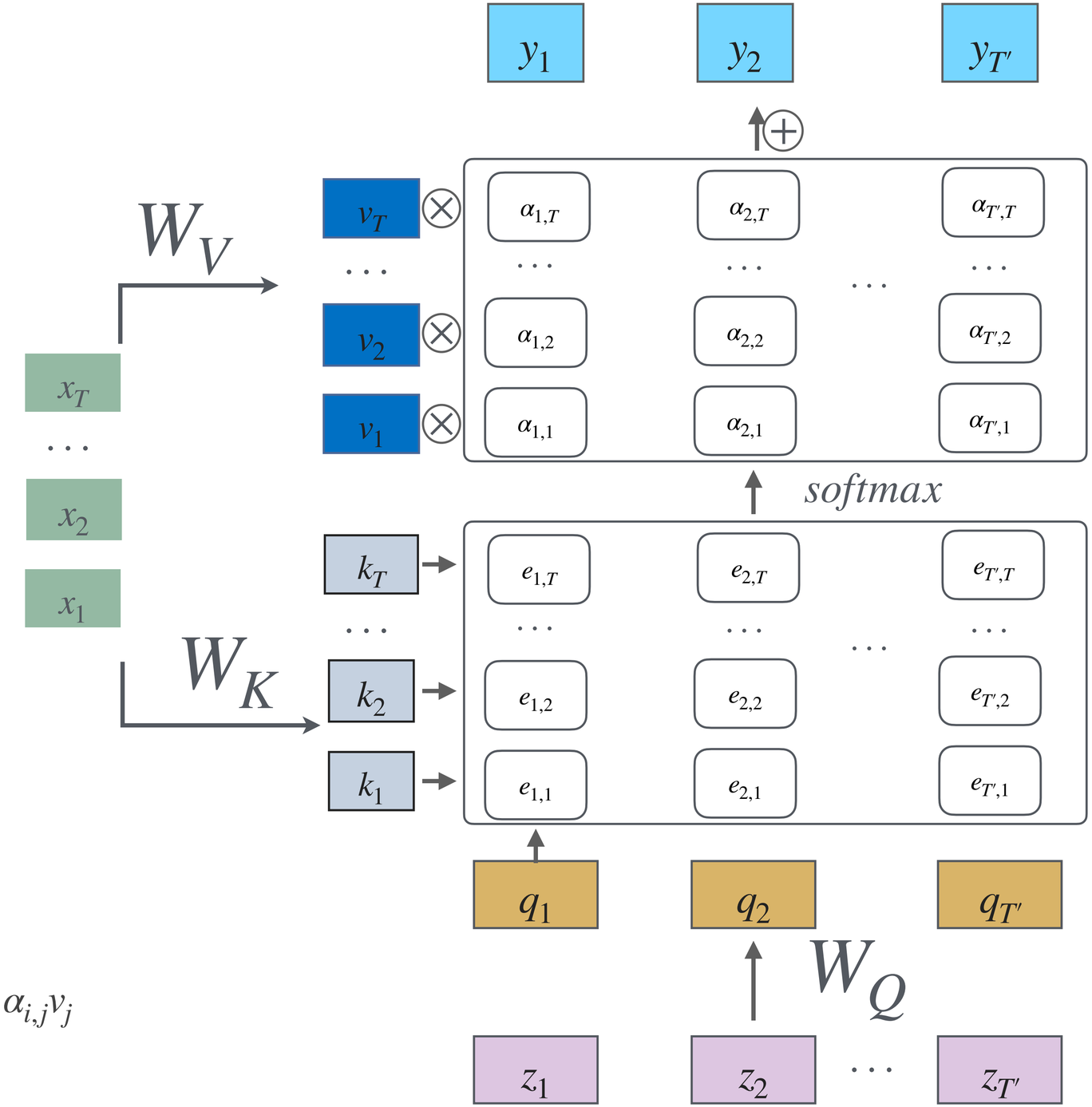
Attention in its general form:

- We have separated the three roles: Queries, Keys, and Values
- We also saw that attention can act on many Queries at once So we can now write attention as a general operator:

Attention: compare Queries to Keys, normalize the scores, and combine the Values

- To calcite attention scores , a common choice is scaled dot product of two vectors q_i and k_j, scaled by dimension of queries and keys, d:

$$e_{i,j} = F(q_i, k_j) = \frac{q_i^\top k_j}{\sqrt{d_k}}$$

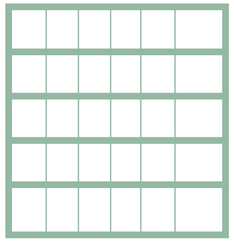


- The whole **Attention operation** Mathematically is:

$$e_{i,j} = \frac{q_i^\top k_j}{\sqrt{d_k}} \quad \rightarrow \quad \alpha_{i,j} = \text{softmax}_j(e_{i,j}) \quad \rightarrow \quad y_i = \sum_j \alpha_{i,j} v_j$$

Rewriting Attention in matrix form

$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix}$$

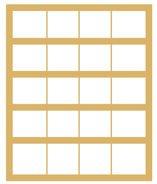


$T \times d_x$

$$Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_{T'}^\top \end{bmatrix}$$

$T' \times d_z$

$$Q = ZW_Q$$

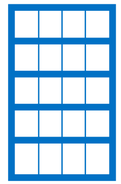


$T' \times d_k$

$$K = XW_K$$

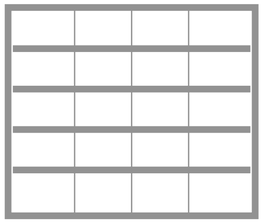
$T \times d_k$

$$V = XW_V$$



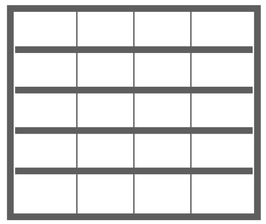
$T \times d_v$

$$E = \frac{QK^\top}{\sqrt{d_k}}$$



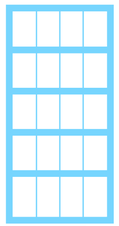
$T' \times T$

$$A = \text{softmax}_{rowwise}(E)$$



$T' \times T$

$$Y = AV$$



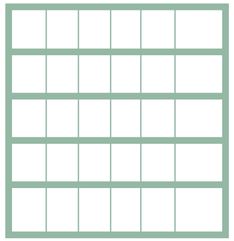
$T' \times d_v$

Note:

$$W_Q \in \mathbb{R}^{d_z \times d_k}$$
$$W_K \in \mathbb{R}^{d_x \times d_k}$$
$$W_V \in \mathbb{R}^{d_x \times d_v}$$

Rewriting Attention in matrix form

$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix}$$

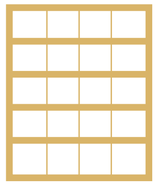


$T \times d_x$

$$Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_{T'}^\top \end{bmatrix}$$

$T' \times d_z$

$$Q = ZW_Q$$

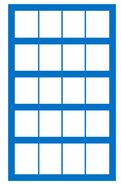


$T' \times d_k$

$$K = XW_K$$

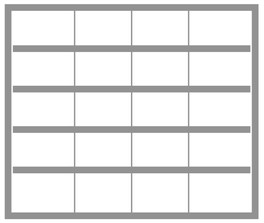
$T \times d_k$

$$V = XW_V$$



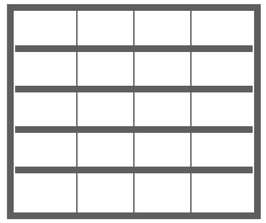
$T \times d_v$

$$E = \frac{QK^\top}{\sqrt{d_k}}$$



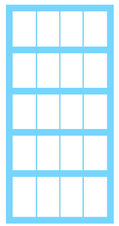
$T' \times T$

$$A = \text{softmax}_{rowwise}(E)$$



$T' \times T$

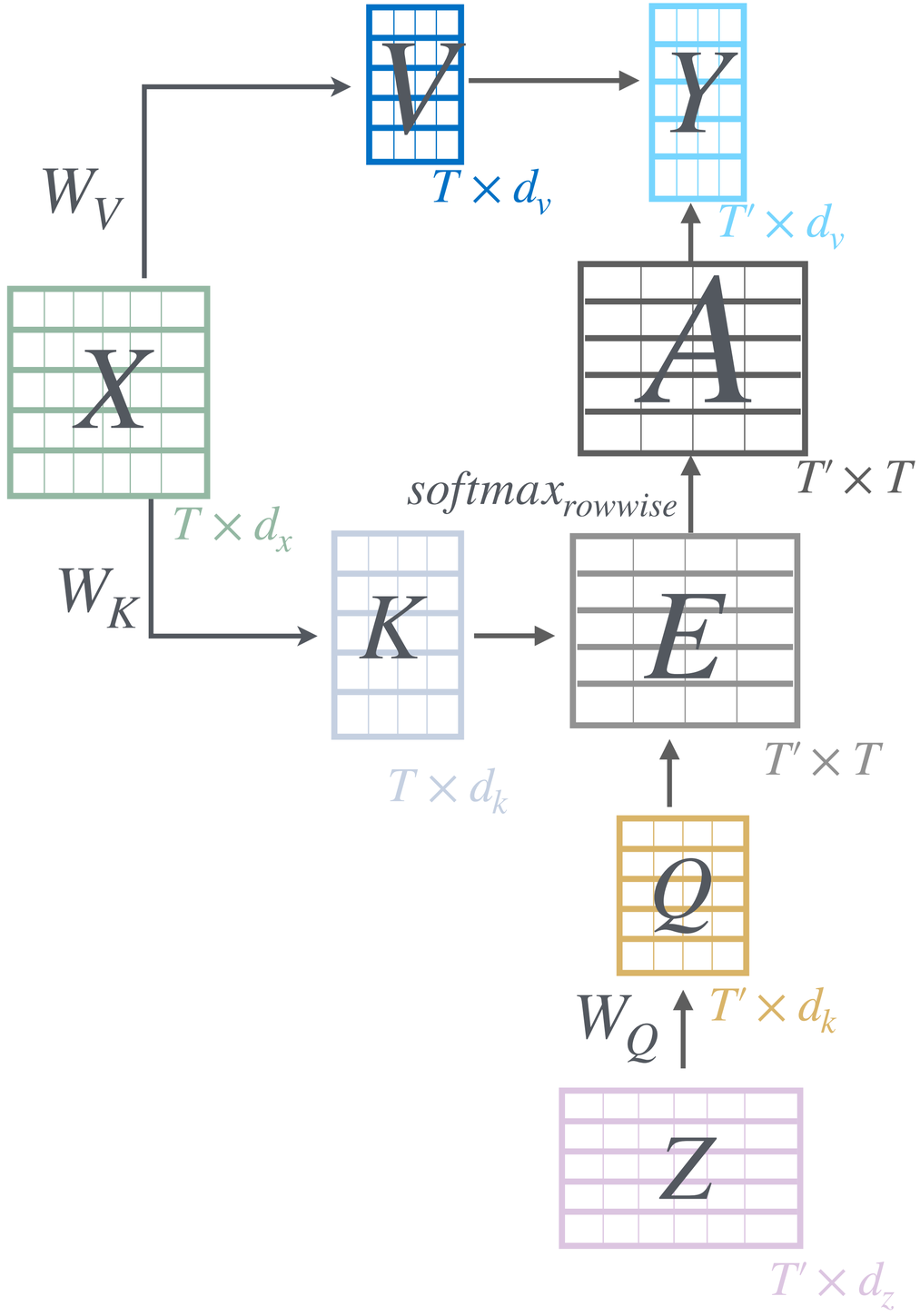
$$Y = AV$$



$T' \times d_v$

Note:

$$W_Q \in \mathbb{R}^{d_z \times d_k}$$
$$W_K \in \mathbb{R}^{d_x \times d_k}$$
$$W_V \in \mathbb{R}^{d_x \times d_v}$$



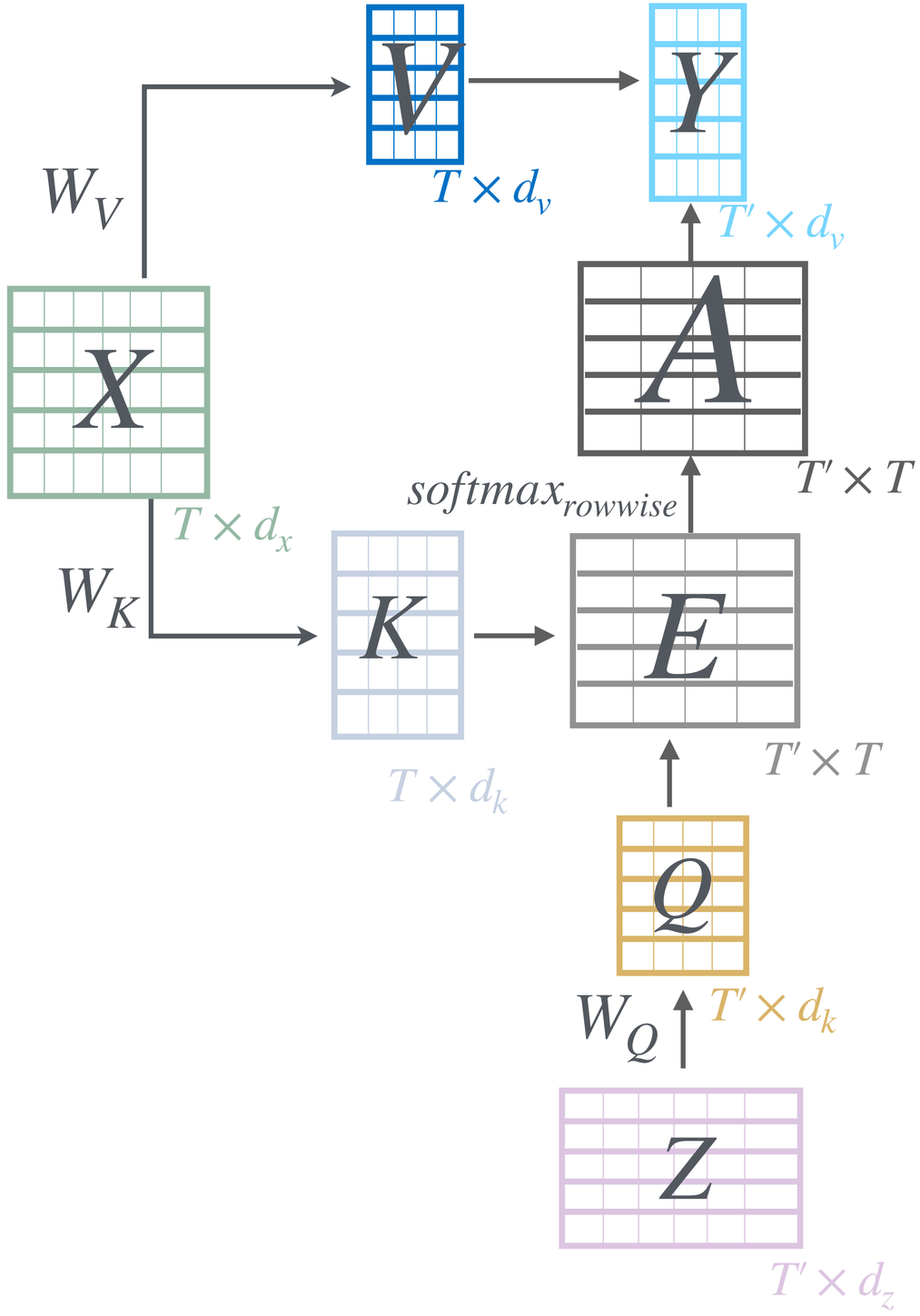
We can write it even more compactly:

$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix} \quad T \times d_x \qquad Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_{T'}^\top \end{bmatrix} \quad T' \times d_z$$

$$Q = ZW_Q \quad T' \times d_k \qquad K = XW_K \quad T \times d_k \qquad V = XW_V \quad T \times d_v$$

$$Y = \text{Attention}(Q, K, V) = \text{softmax}_{rowwise}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Note: $W_Q \in \mathbb{R}^{d_z \times d_k}$
$W_K \in \mathbb{R}^{d_x \times d_k}$
$W_V \in \mathbb{R}^{d_x \times d_v}$



1) Attention is **permutation equivariant** in the Queries Z

Let P_Z permute the rows of Z. Then:

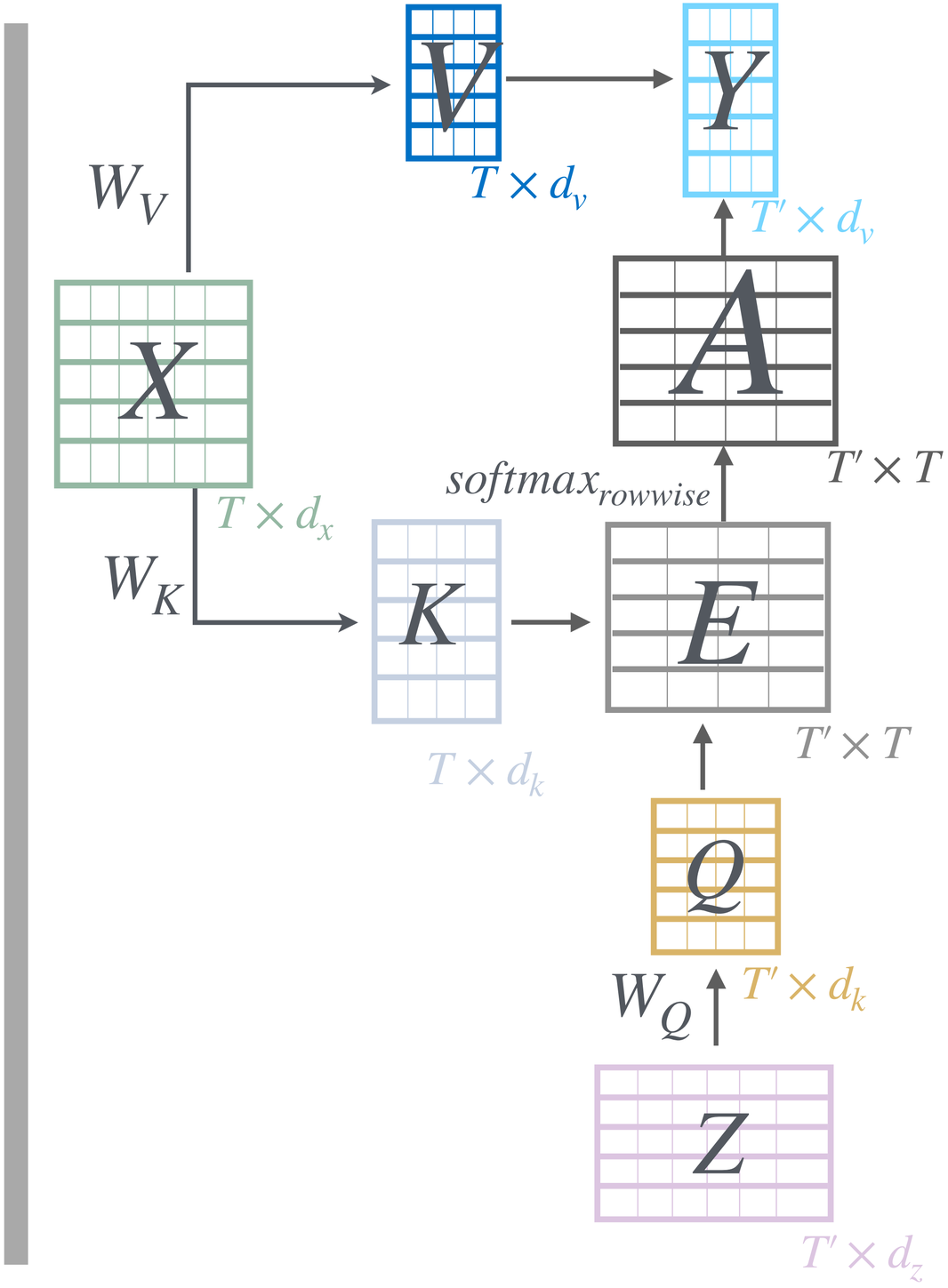
$$Z' = P_Z Z \quad \Rightarrow \quad Q' = P_Z Q \qquad E' = \frac{Q'K^\top}{\sqrt{d_k}} = \frac{(P_Z Q)K^\top}{\sqrt{d_k}} = P_Z E$$

$$A' = \text{softmax}_{roweise}(E') = \text{softmax}_{roweise}(P_Z E) = P_Z A$$

$$Y' = A'V = (P_Z A)V = P_Z(AV) = P_Z Y$$

$$Y' = \text{Attention}(P_Z Q, K, V) = P_Z Y$$

Interpretation: Reordering the Queries reorders the outputs in the same way. That makes sense because each Query asks its own question, so changing their order should only change the order of the answers.



2) Attention is **permutation invariant** in the Key/Value set X

Let P_X permute the rows of X. Then:

$$X' = P_X X \quad \Rightarrow \quad K' = P_X K, \qquad V' = P_X V$$

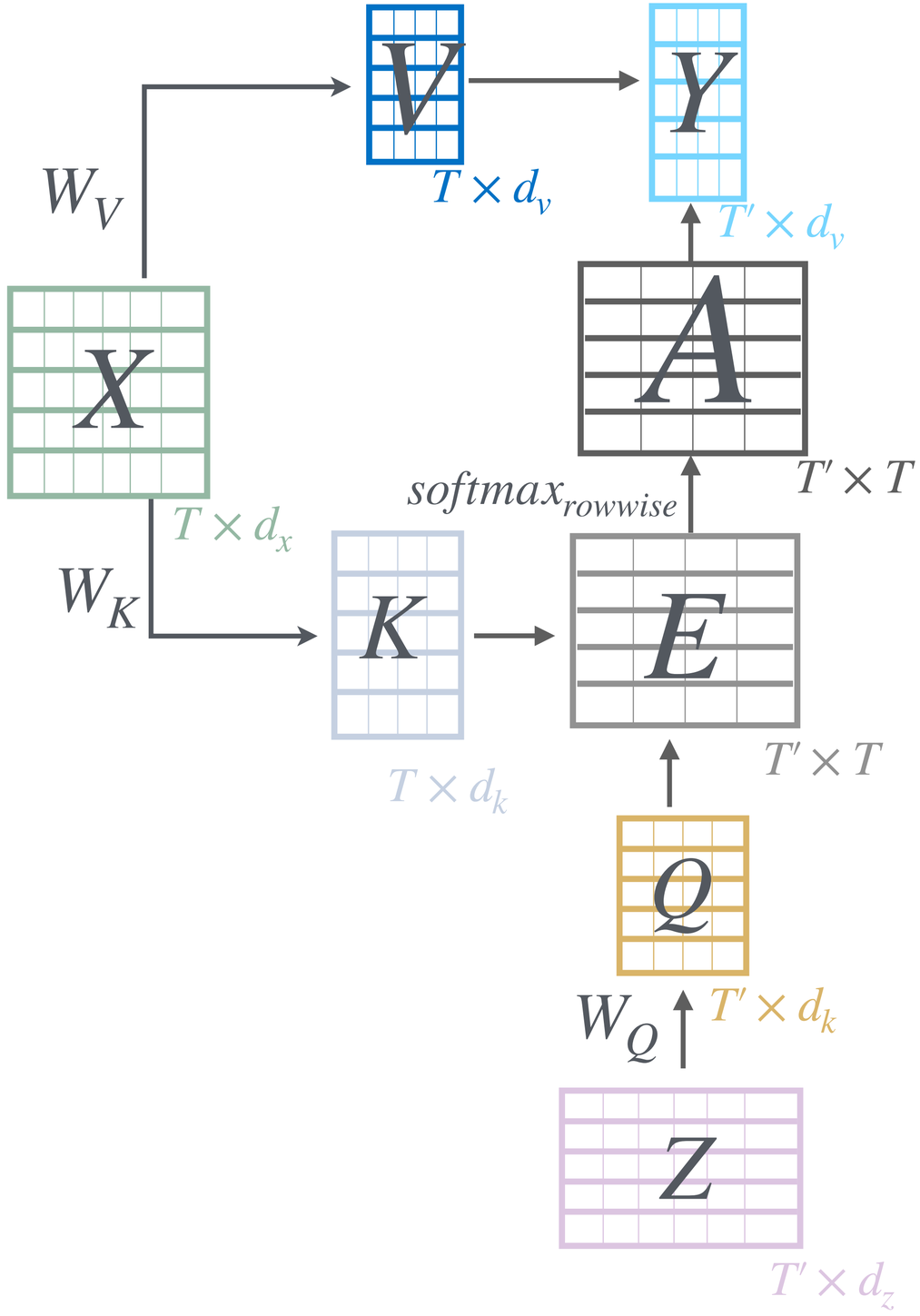
$$E' = \frac{Q(K')^\top}{\sqrt{d_k}} = \frac{Q(P_X K)^\top}{\sqrt{d_k}} = \frac{QK^\top P_X^\top}{\sqrt{d_k}} = EP_X^\top$$

$$A' = \text{softmax}_{rowwise}(E') = \text{softmax}_{rowwise}(EP_X^\top) = AP_X^\top$$

$$Y' = A'V' = (AP_X^\top)(P_X V) = A(P_X^\top P_X)V = AV = Y$$

$$Y' = \text{Attention}(Q, P_X K, P_X V) = Y$$

Interpretation: Reordering the Key/Value set does not change the output, as long as Keys and Values are permuted together. That makes sense because they act like a memory bank: changing the order of stored items should not change what is retrieved.



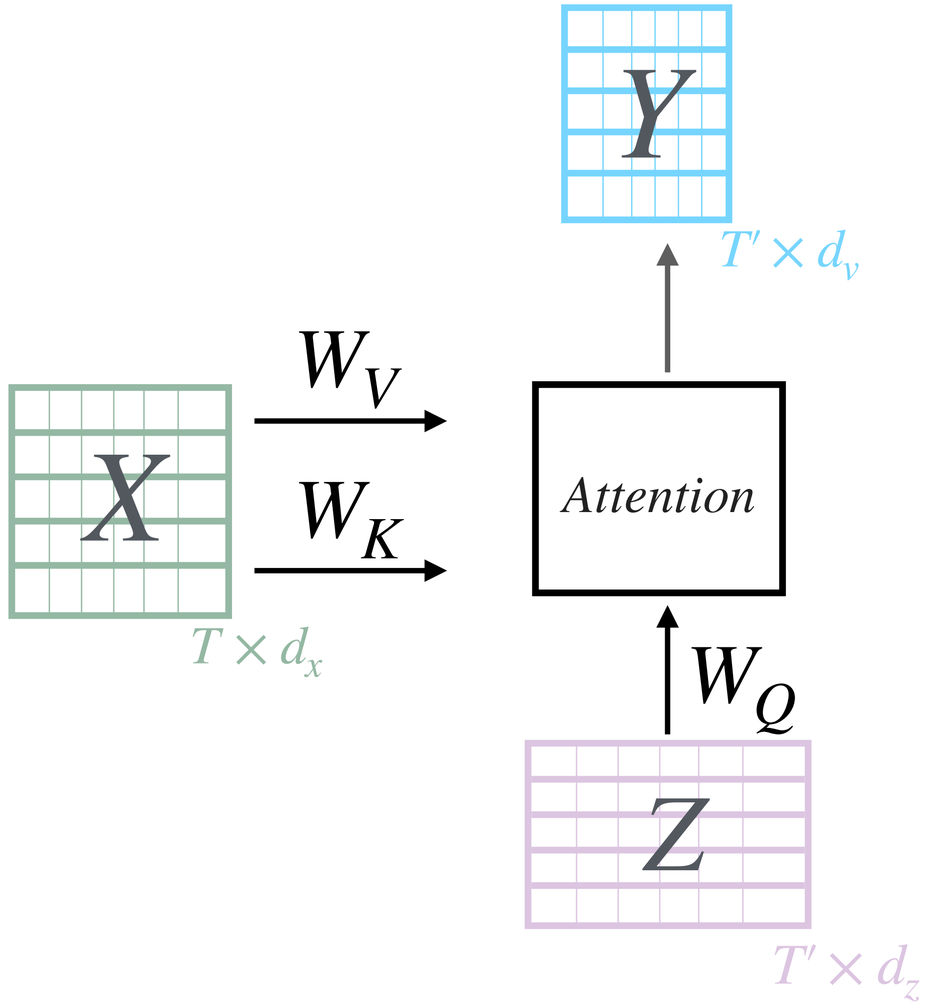
Attention Operator updates each element of Z by letting it softly read from all elements of X

We start with two sets of representations:
Z : containing T' query items (each has dim d_z).
X : containing T source items (each has dim d_x).

The idea of Attention is that each query in Z can retrieve information from the whole set X, not by selecting one item hard, but by forming a weighted combination of all T source items.

Doing this for all T' queries produces a new set of T'output representations Y.

The Attention operator implements this through learned projections W_Q, W_K, W_V followed by score computation, softmax normalization, and weighted mixing.



ingredient: **Attention Operator block (PyTorch implementation)**

Input/output

1- Input tensors

Query source: $Z \in [B, T', d_z]$

Key/Value source: $X \in [B, T, d_x]$

2- Output tensor:

Attention output: $Y \in [B, T', d_v]$

variable

1-Learnable projections: (Use nn.Linear(...)):

$W_Q : d_z \to d_k,\ W_K : d_x \to d_k,\ W_V : d_x \to d_v$

In PyTorch, where B is batch size:

$$Q = W_Q(Z) \in [B, T', d_k]$$

$$K = W_K(X) \in [B, T, d_k]$$

$$V = W_V(X) \in [B, T, d_v]$$

Computation

1-Attention computation:

$$E = \frac{QK^\top}{\sqrt{d_k}} \in [B, T', T]$$

In PyTorch transpose Keys on the last two dims use batched matmul:

$$A = \text{softmax}(E, \text{ dim} = -1) \in [B, T', T]$$

2- $Y = AV \in [B, T', d_v]$