

KABUL UNIVERSITY

FACULTY OF COMPUTER SCIENCE

Deep Learning

Introduction to Attention and Transformers

PhD Course

Lecturer: Ziaullah Momand, PhD

Learning objectives

- Explain why attention helps a model use relevant information across a sequence.
- Distinguish queries, keys, values, self-attention, and cross-attention.
- Calculate one attention output and explain position information and causal masking.
- Trace information through a Transformer and identify its main applications.

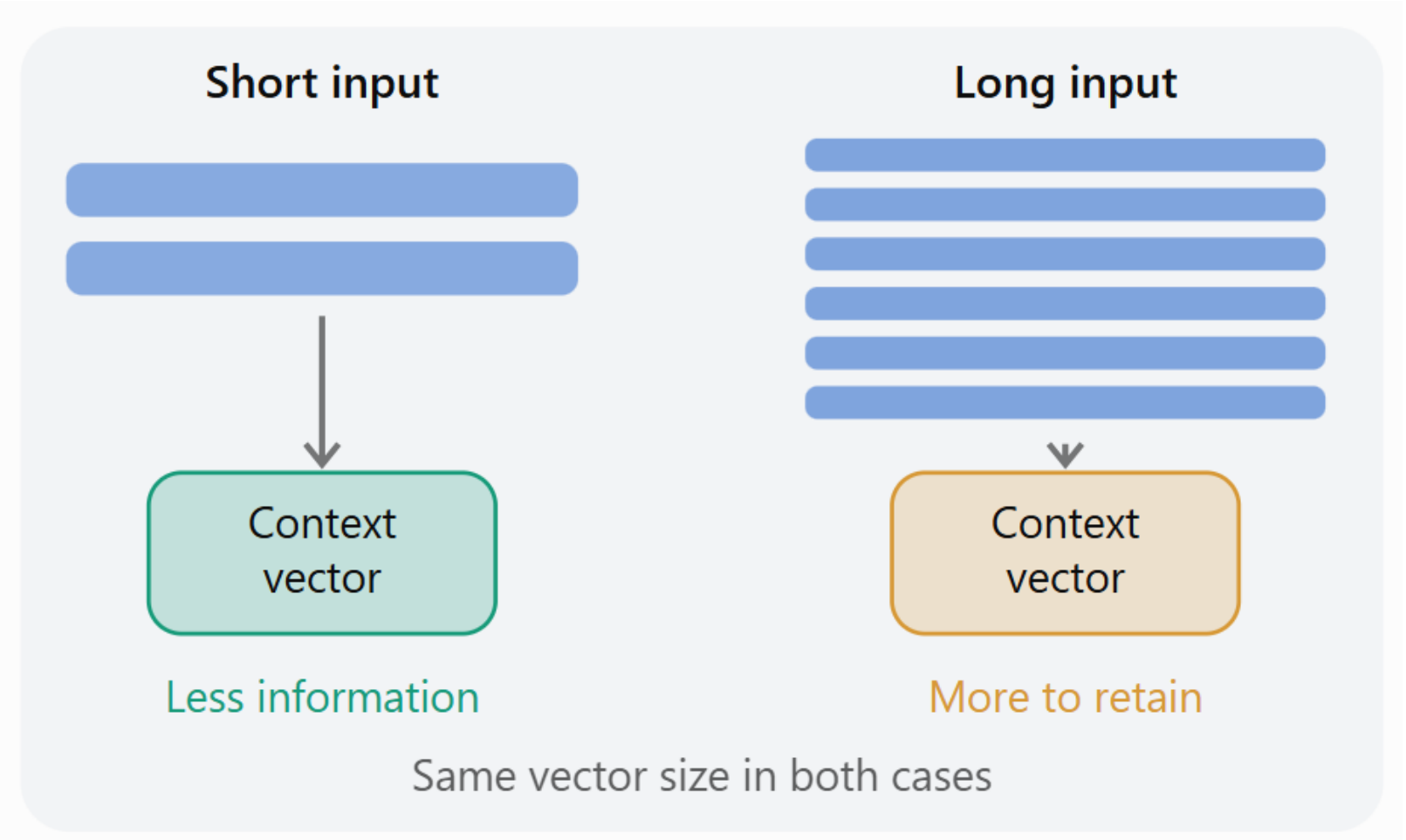
Word meaning depends on context

- “The student read the book because it was interesting.” Here, “it” refers to the book.
- An initial token embedding represents the token. A contextual representation also uses information from the sentence.
- Attention lets a token gather information from other relevant positions.
- Let’s treat words as tokens. Practical tokenizers may split words into smaller units.

Why attention helps sequence models

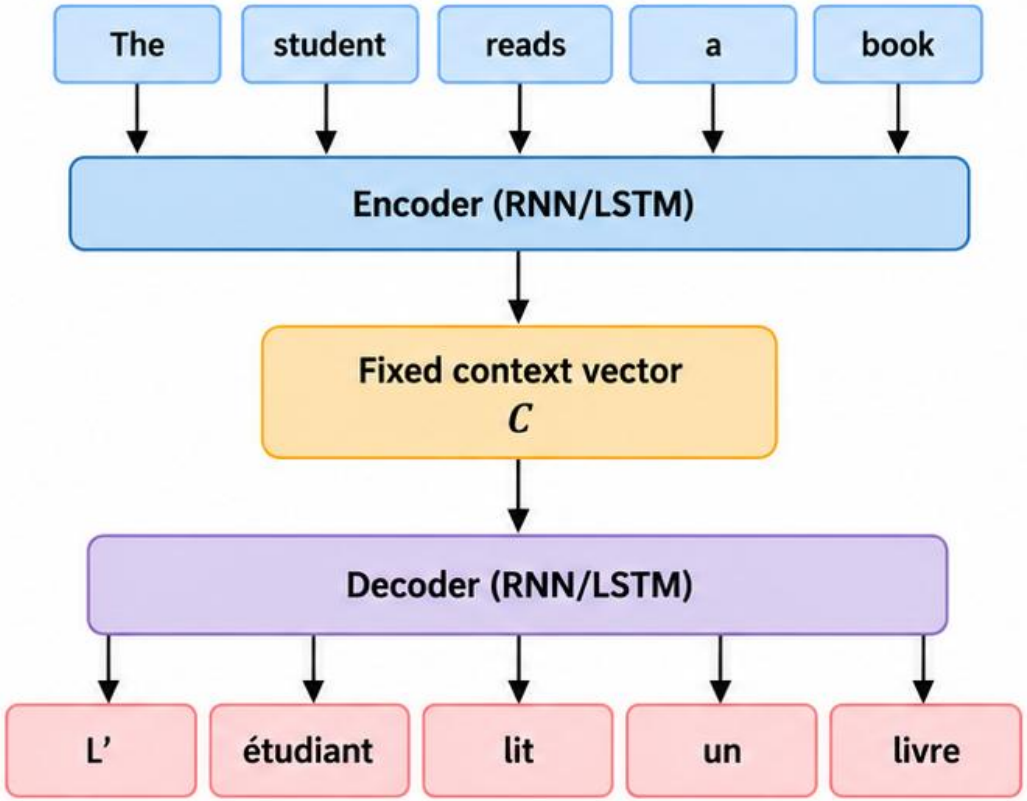
- An RNN carries information through successive hidden states, which can make distant relationships difficult to learn.
- Recurrent states depend on earlier states, limiting parallel computation across positions.
- Early encoder–decoder models often compressed the source into one fixed-size context vector.
- Attention gives the model direct access to a collection of representations when building an output.

Why attention helps sequence models



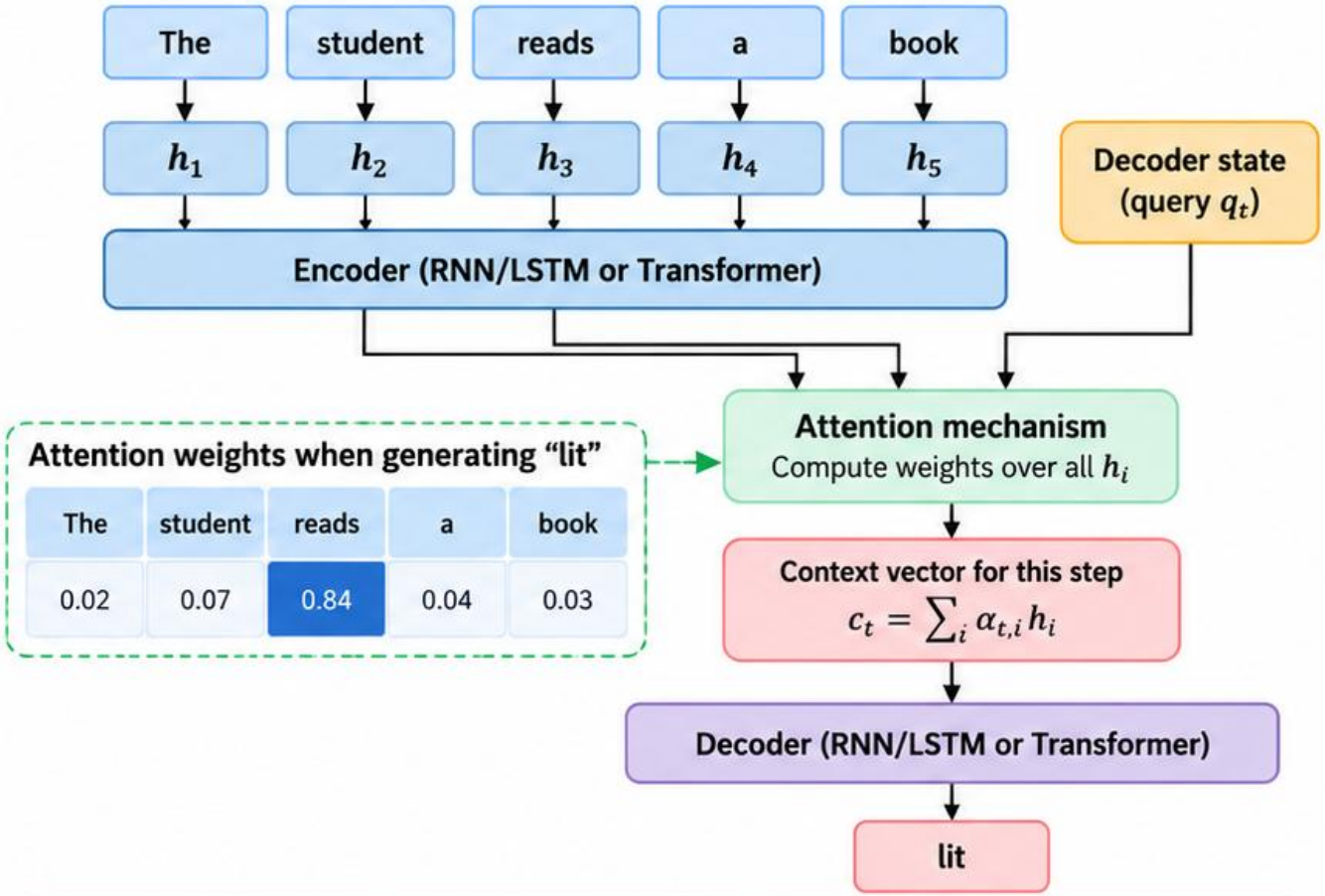
The problem: Whether the input sentence contains 5 words or 100 words, the context vector has the same number of dimensions.

Without Attention
(Early Encoder–Decoder)



- The entire source sentence is compressed into one vector.
- The same vector C is used at every decoding step.
- Long sentences may lose important details.

With Attention
(RNN with Attention / Transformer)



- The decoder computes attention weights over all encoder states.
- It creates a different context vector c_t at each decoding step.
- This allows the decoder to focus on the relevant source words.

Attention as a weighted combination

$$z = \sum_{j=1}^{n} \alpha_j v_j, \qquad \alpha_j \geq 0, \qquad \sum_{j=1}^{n} \alpha_j = 1$$

- Attention assigns a relevance weight to each available value vector.
- Softmax attention gives nonnegative weights whose sum is one.
- The output combines the values according to their weights, so several positions can contribute.

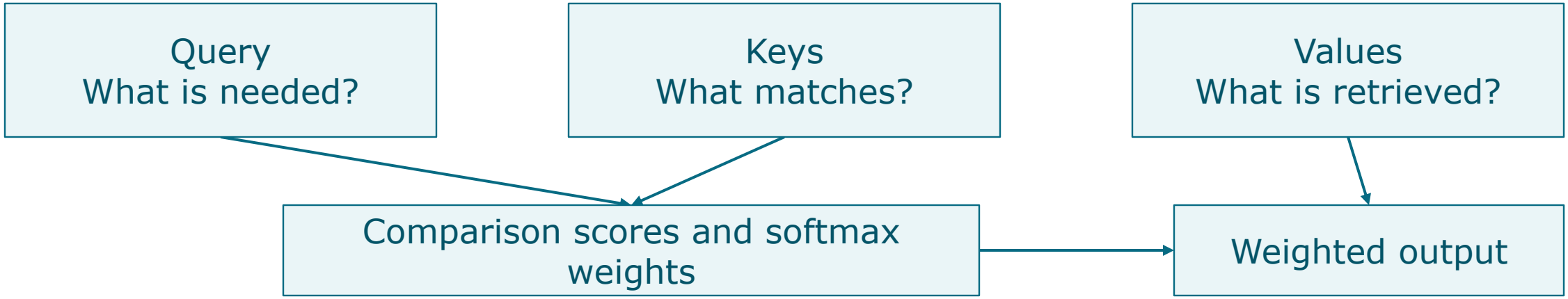
The Central Learning Goal

- **How token gathers information from other tokens and how transformer blocks use this process.**

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

- **Compare queries with keys** to calculate relevance scores.
- **Scale the scores** to help keep their magnitude manageable.
- **Apply softmax** to turn the scores into attention weights.
- **Combine the values** using those weights to produce contextual representations.

Queries, keys, and values



- The query describes what the current position is looking for.
- Each key provides features used to assess relevance to the query.
- Each value supplies the information that the attention output combines.

Learning the query, key, and value three roles in one operation

You want material about neural networks:

- Your search request describes what you need.
- Each book has searchable descriptors.
- Each book contains information you can retrieve.

Component	Role	Library analogy
Query q	Describes what the current position is looking for	Search request
Key$k$	Provides features against which the query is compared	Searchable descriptors
Value v	Provides information to be combined into the output	Retrieved content

Where do the Vectors come from?

For a token representation x_i:

$$q_i = x_i W_Q$$

$$k_i = x_i W_K$$

$$v_i = x_i W_V$$

The matrices W_Q, W_K, and W_V are learned during training.

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

X: token representations W: learned projection matrices

Scaled dot-product attention

$$Z = \text{softmax}\left(\frac{QK^{\mathsf{T}}}{\sqrt{d_k}}\right)V$$

Q: queries K: keys V: values d_k: query/key dimension

- Compare every query with the keys using dot products, then scale the scores.
- Apply softmax across the key positions separately for each query.
- Multiply the attention weights by the values to obtain contextual output vectors.

Example: the input vectors

Token	Key vector	Value vector
I	[0, 1]	[1, 0]
love	[1, 0]	[0, 1]
learning	[2, 0]	[1, 1]

- Input sequence: “I love learning”. We calculate only the output for “love”.
- The query for “love” is [1, 0]. Each key and value shown has two components.
- These invented vectors isolate the attention calculation. A trained model learns the projections that produce them.

Worked example: scores and scaling

$$s_j = \frac{q k_j^\top}{\sqrt{2}}, \qquad [s_1, s_2, s_3] \approx [0,\ 0.7071,\ 1.4142]$$

- Dot products for “I”, “love”, and “learning” are 0, 1, and 2.
- Each key has two components, so divide every score by the square root of two.
- The scaled scores express relative compatibility. They do not yet sum to one.

Example: softmax weights

$$\alpha_j = \frac{e^{s_j}}{\sum_{\ell=1}^{3} e^{s_\ell}}$$

Token	Exponential	Weight
I	1.0000	0.1400
love	2.0281	0.2840
learning	4.1133	0.5760

- Exponentiate the scaled scores, then divide by the sum of those exponentials.
- The weights are approximately 0.1400, 0.2840, and 0.5760.
- “Learning” contributes most in this example. All three positions still contribute.

Example: the attention output

$$z_{\text{love}} = 0.1400[1, 0] + 0.2840[0, 1] + 0.5760[1, 1]$$
$$\approx [0.7160,\ 0.8600]$$

- Multiply each value vector by its attention weight and add coordinate by coordinate.
- The resulting vector for “love” combines information from the entire three-token sequence.
- This vector goes to later model components. It is not yet a predicted word.

Self-attention across the sequence

Query / source	I	love	learning
I	0.50	0.30	0.20
love	0.140	0.284	0.576
learning	0.10	0.25	0.65

Illustrative matrix: only the middle row comes from the worked example.

- Every token supplies a query and receives its own weighted combination of values.
- Rows identify query positions. Columns identify the positions supplying information.
- Each row sums to one. Attention from one token to another need not be symmetric.

Multi-head attention

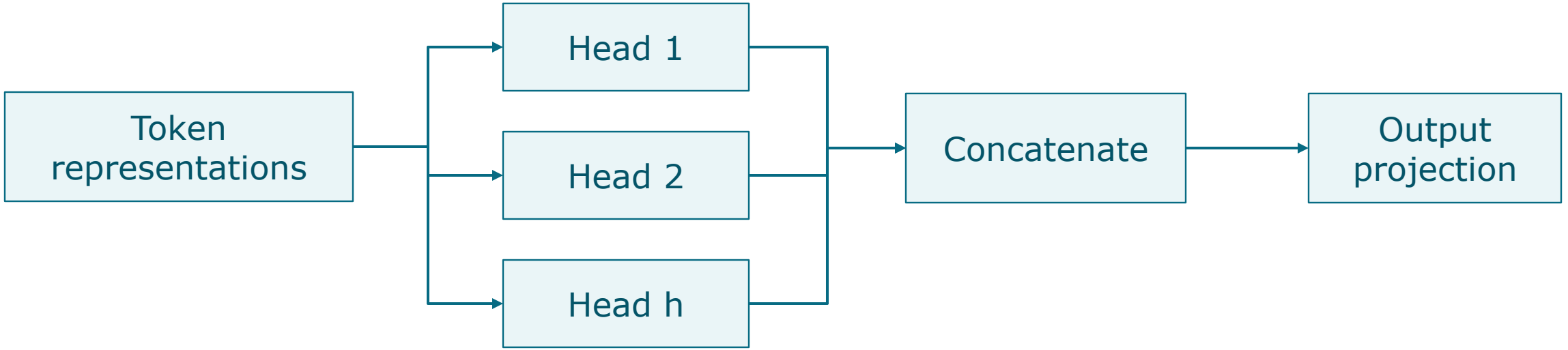
One attention head produces one pattern of weighted information gathering.
However, a sentence contains many potentially useful relationships:

- A pronoun and a possible referent.
- A verb and its subject.
- A verb and its object.
- Nearby modifiers.
- Relationships between distant phrases.

Multi-head attention creates several attention operations with different learned projections.

Each head learns a different way to compare and combine token information. The model then combines the heads' results.

Multi-head attention



Example: 8 heads, each with 64 output features, combine into 512 features.

- Each head uses its own learned projections to compare and combine token information.
- Concatenating the head outputs and applying an output projection combines their results.
- Heads can learn different patterns, but no head has a guaranteed grammatical role.

Positional information represents order

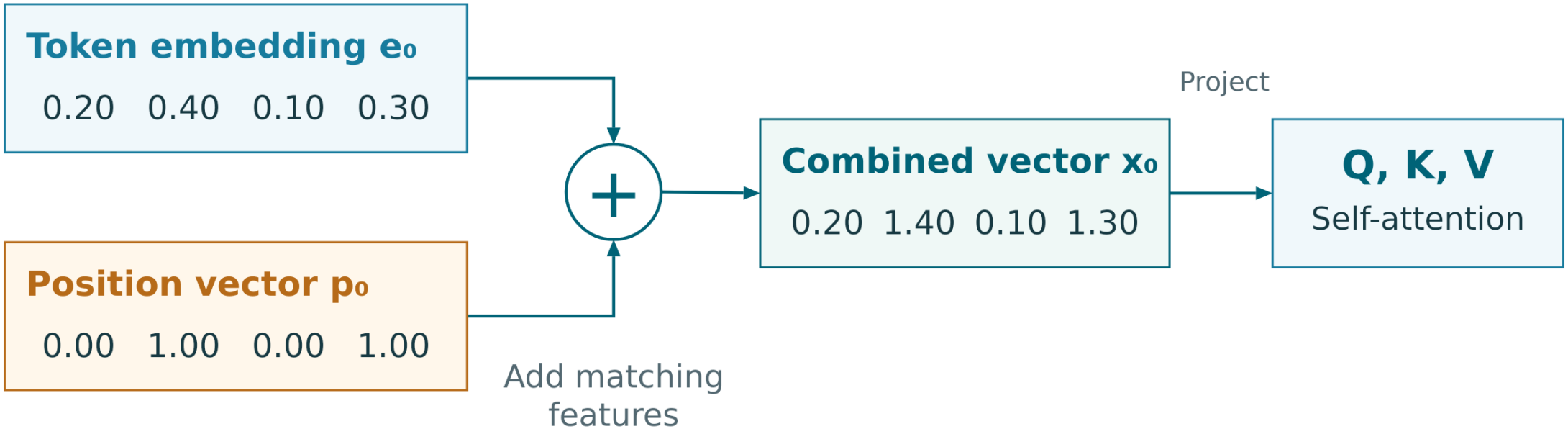
$$x_i = e_i + p_i$$

- "The dog chased the cat" and "The cat chased the dog" contain the same words but have different meanings.
- The basic Transformer adds a positional vector to each token embedding.
- Positions may use fixed sine/cosine patterns or learned embeddings.

Adding positional information

$$x_i = e_i + p_i$$

Example token: Students at position 0



Each token carries information about its content and its position.

Illustrative 4-feature embedding and sinusoidal position encoding

?