Evolution of Sequence-to-Sequence Models

1. Sequence-to-Sequence Tasks
Some problems require mapping an **input sequence** to an **output sequence** that may be of different length and structure — for example:
•**Machine translation** (“I am happy” → “Je suis heureux”)
•**Summarization**
•**Speech-to-text**
These are called **sequence-to-sequence (Seq2Seq)** tasks. They require models that can handle variable lengths, preserve order, and understand dependencies between elements across the sequence.

2. Early Sequence Models: RNNs
Recurrent Neural Networks (RNNs) were the first widely used models for sequence data. They process tokens step-by-step, passing a **hidden state** forward that summarizes what has been seen so far.
•**Strength**: Naturally handles sequences and token order.
•**Weakness**:
- **Vanishing gradients**: The influence of earlier tokens fades as the sequence grows.
- **Short memory**: Struggles to capture long-range dependencies.

3. LSTMs and GRUs: Fixing RNN Weaknesses
To address these issues:
•**LSTMs (Long Short-Term Memory)** introduced:
- A **cell state** to carry long-term information.
- **Gates** (forget, input, output) to control what is remembered, updated, or output.
- Better at preserving information over longer sequences.

•**GRUs (Gated Recurrent Units)**:
- A simpler variant with only two gates (update, reset).
- Merged hidden and cell states into one.
- Fewer parameters, faster training, similar performance to LSTMs in many cases.

These gated RNNs made sequence modeling more effective, but **they still processed tokens sequentially** and struggled with tasks where input and output lengths differed significantly.

4. The Need for Encoder–Decoder Architectures
While LSTMs/GRUs improved memory, they still worked best when the input and output sequences were aligned in length and timing (e.g., labeling tasks).
For translation, summarization, and other Seq2Seq problems:
•The input must be **fully understood** before output starts.
•The output may be longer, shorter, or rearranged compared to the input.
The solution: split the process into two parts:
•**Encoder**: Reads the entire input sequence, step-by-step, and produces a **context vector** summarizing it.
•**Decoder**: Starts from the context vector and generates the output sequence, step-by-step.
Benefits:
1.Handles **different-length input/output** naturally.
2.Gives the encoder time to fully process the input before output generation.
3.Creates a clean separation between understanding (encoder) and producing (decoder).

5. The Context Vector Bottleneck
In the original encoder–decoder design:
•The **final hidden state** of the encoder (the context vector) was the *only* information passed to the decoder.
•This worked for short sequences, but in long or complex inputs, compressing all information into a single fixed-length vector caused an **information bottleneck** — some details were inevitably lost.

6. Attention Mechanism: Easing the Bottleneck
Attention was introduced to address the limitations of the single context vector.
•Instead of using only the final encoder state, attention lets the decoder **look back at all encoder hidden states** when generating each output token.
•At each step, the decoder calculates **attention weights** over the encoder states, focusing on the most relevant parts of the input.
•**Impact**:
- Greatly improves performance on long sequences.
- Particularly effective for translation, where different output tokens align with different parts of the input.

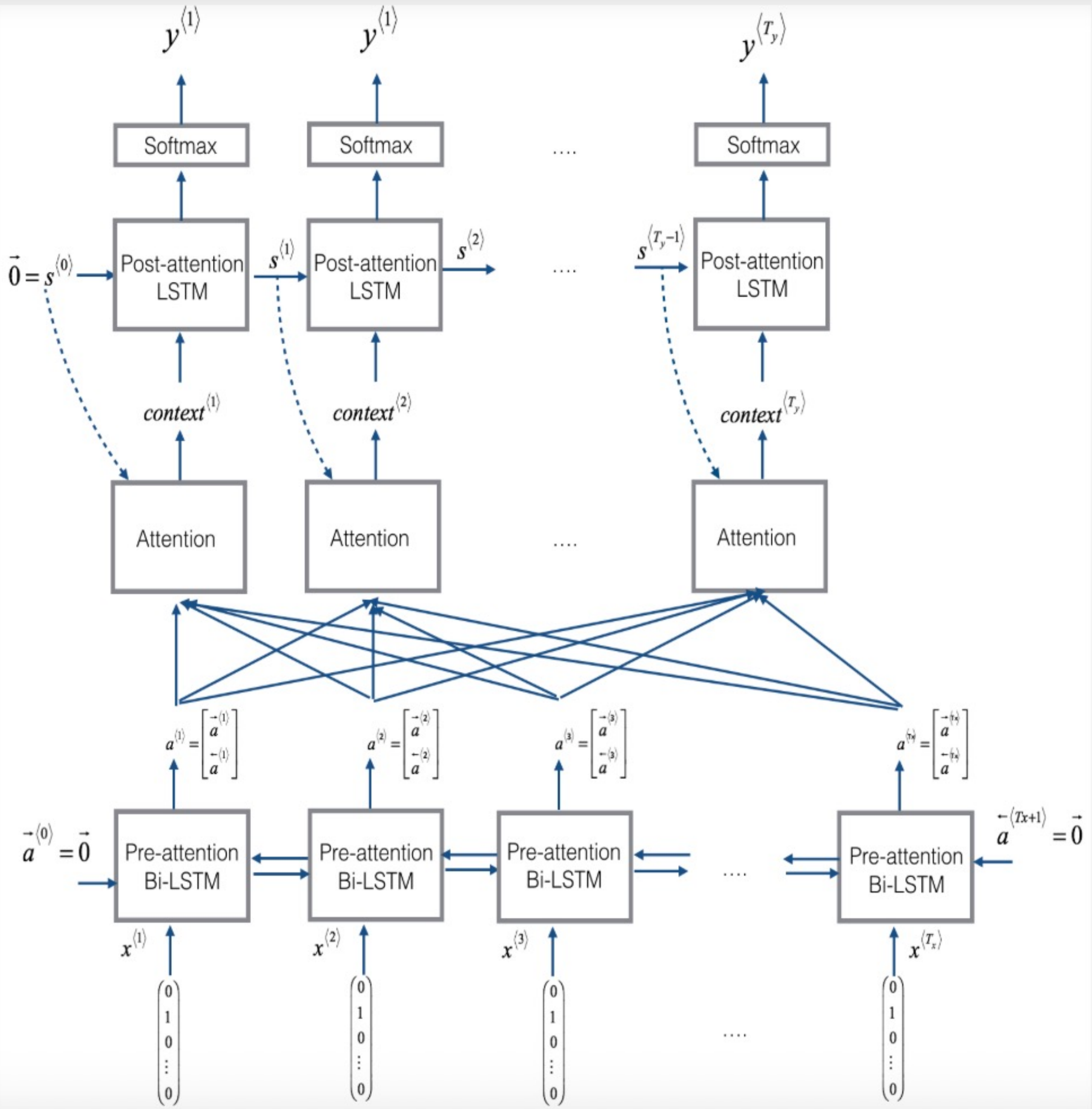
Initially, attention was an **add-on** to RNN/LSTM/GRU encoder–decoders.

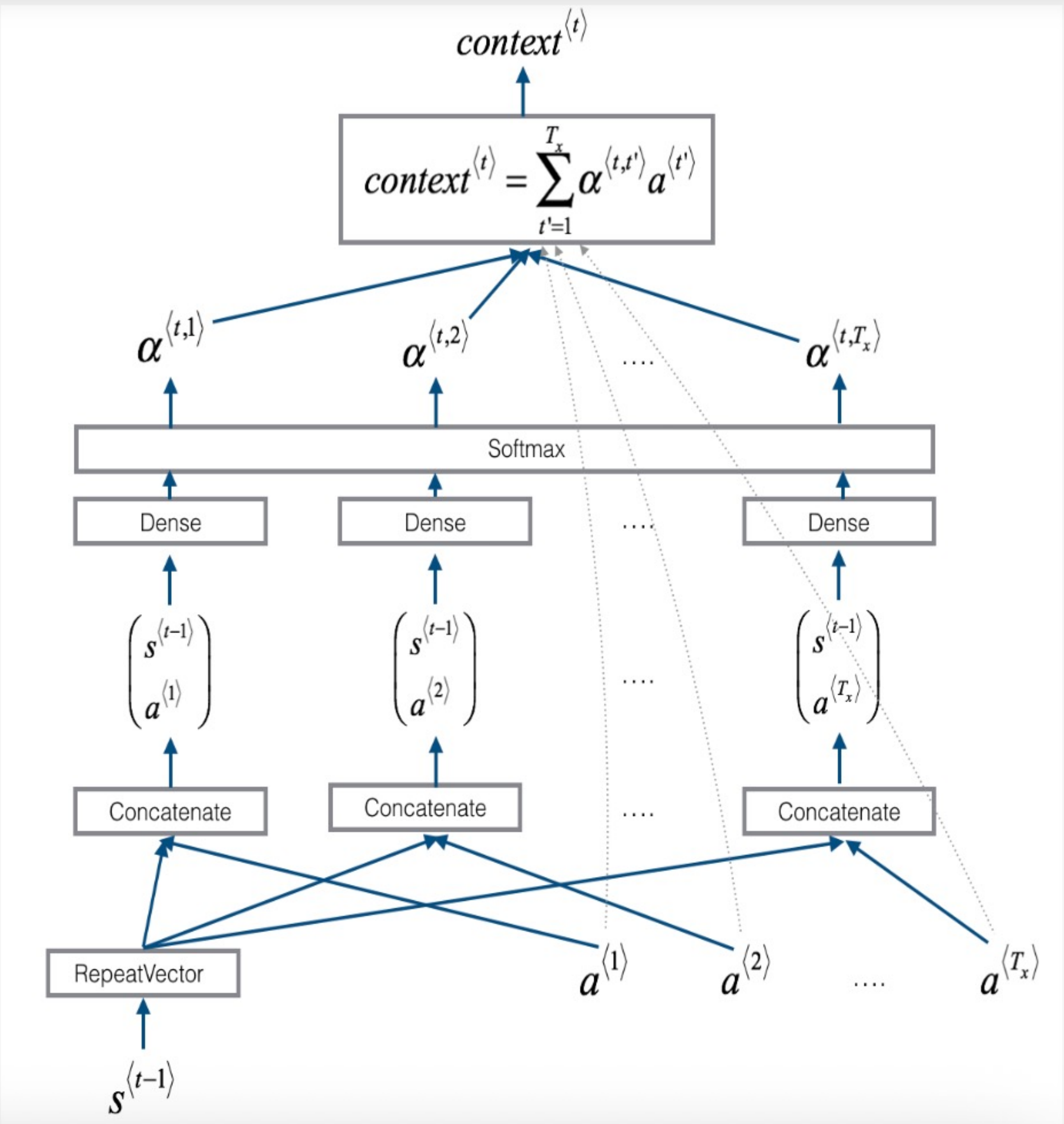
7. Transformers: Attention as the Foundation
The **Transformer** architecture took the attention idea further:
•Removed recurrence entirely.
•Built both encoder and decoder from **stacked self-attention layers** + feed-forward networks.
•Added **positional encodings** to preserve order (since self-attention alone doesn’t know token positions).
•**Benefits**:
- Processes all tokens in parallel → much faster training.
- Handles long-range dependencies without vanishing gradients.
- Scales well to large datasets and deep networks.

At a high level, Transformers still follow the **encoder–decoder** pattern, but internally they rely entirely on **self-attention and cross-attention** instead of recurrent processing.

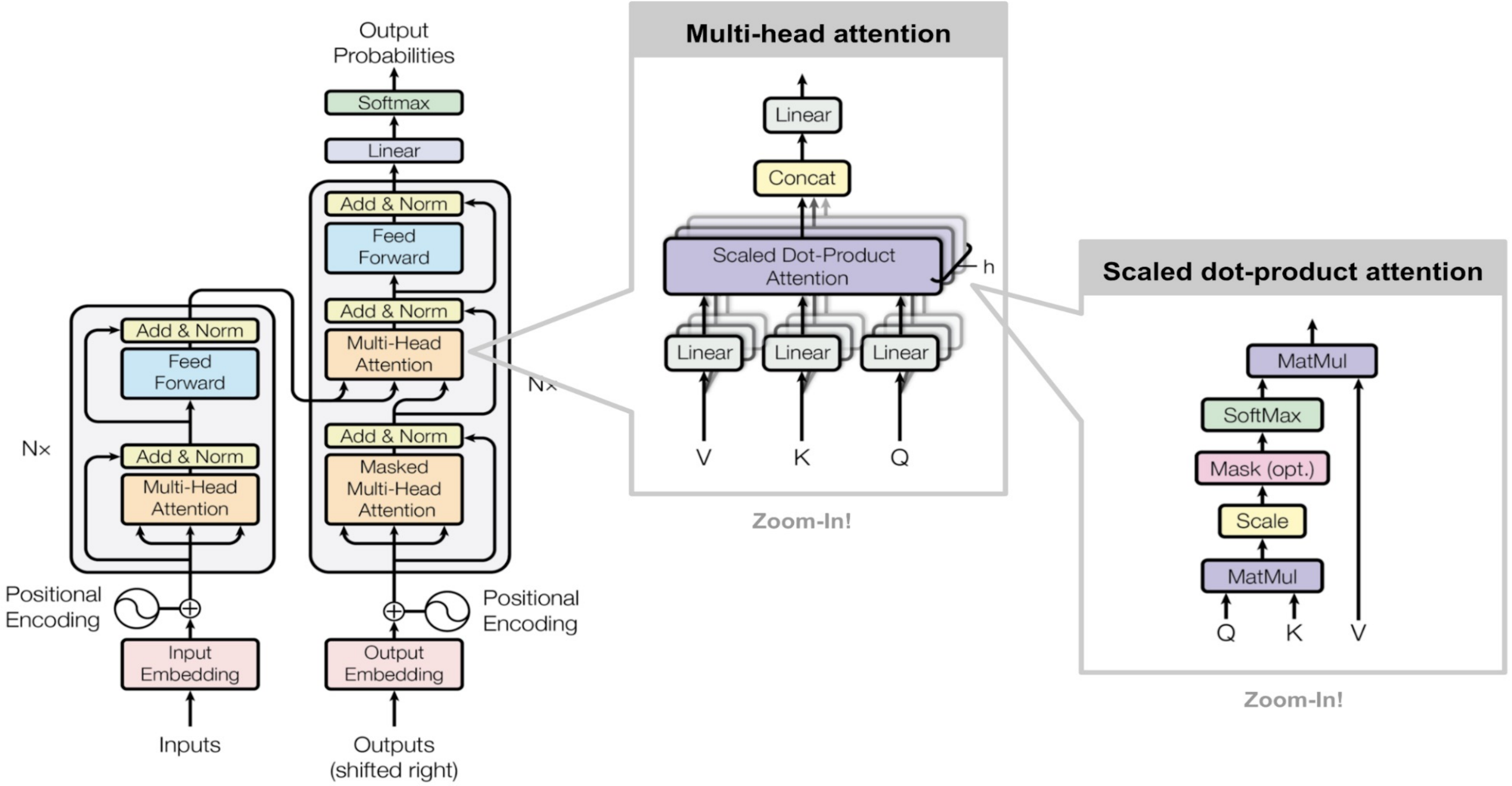
8. The Evolution Path
1.**RNNs** → basic sequence modeling, but poor long-term memory.
2.**LSTMs / GRUs** → improved memory with gates, but still sequential and limited for complex Seq2Seq tasks.
3.**Encoder–Decoder (RNN/LSTM/GRU)** → enabled variable-length input/output by separating encoding and decoding.
4.**Encoder–Decoder + Attention** → solved the single-vector bottleneck by letting the decoder attend to all encoder states.
5.**Transformers** → made attention the core computation, removing recurrence and enabling massive parallelization and scalability.

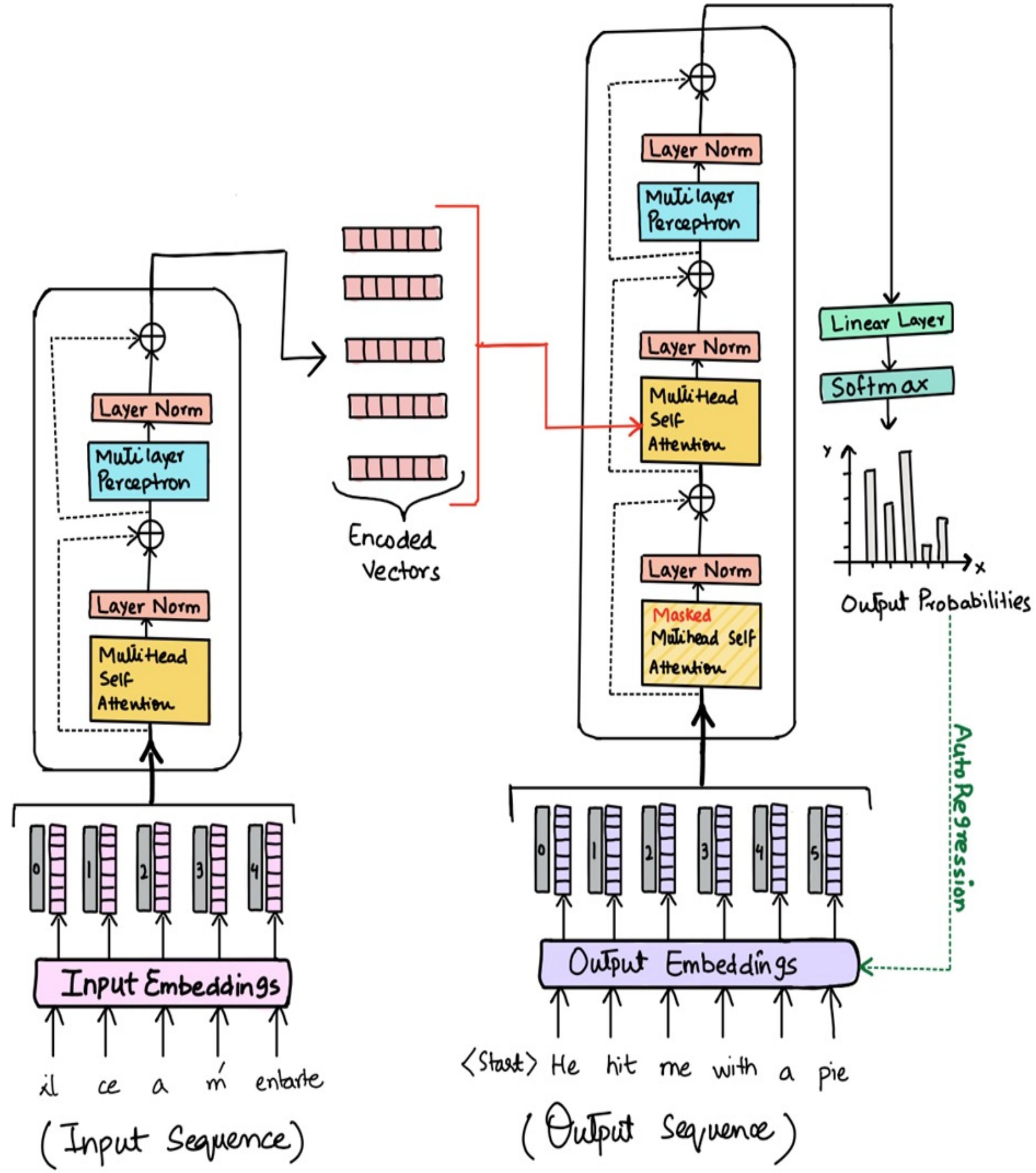
Attention in Sequential (RNN/LSTM/GRU) Encoder-Decoder Models

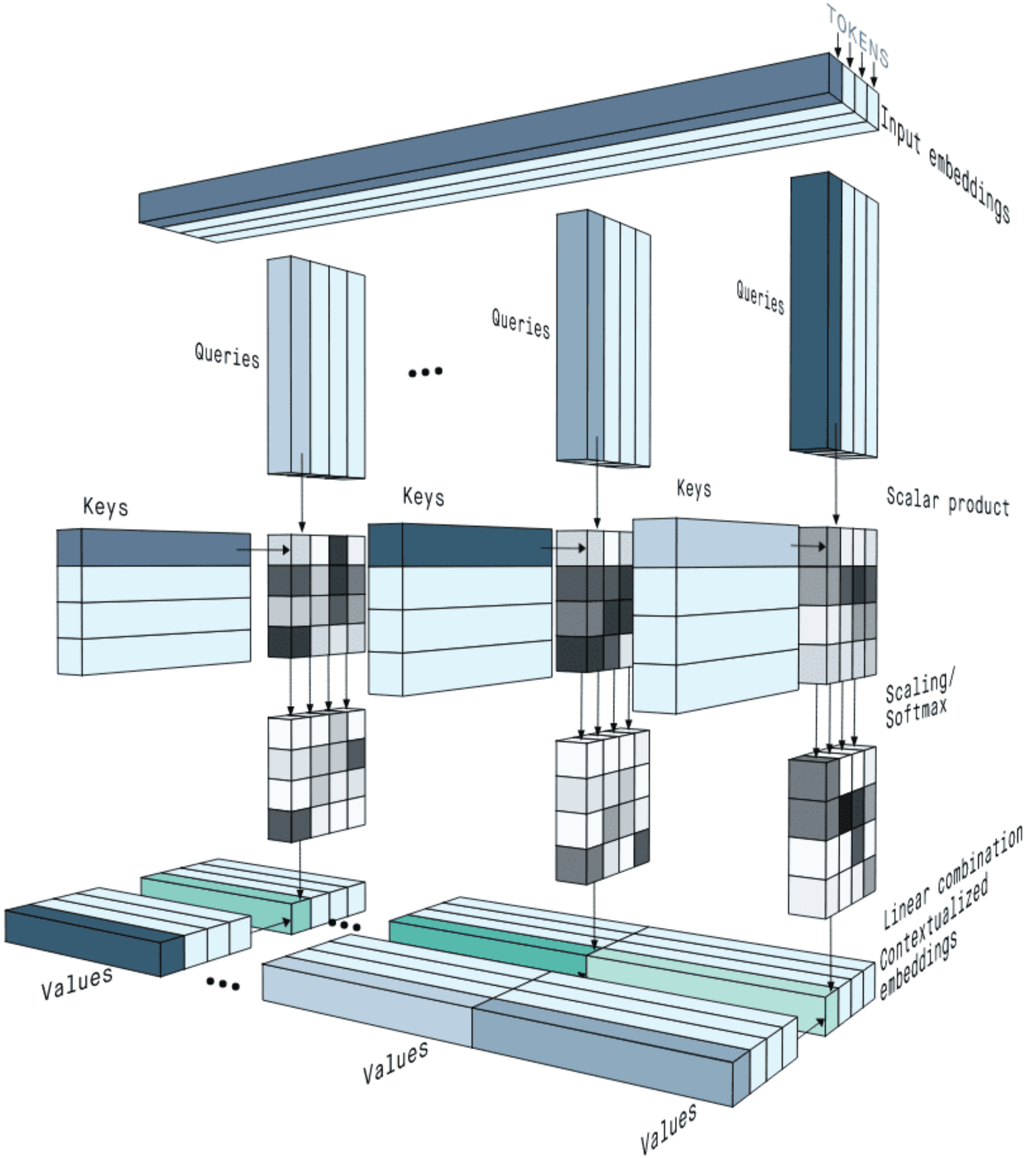


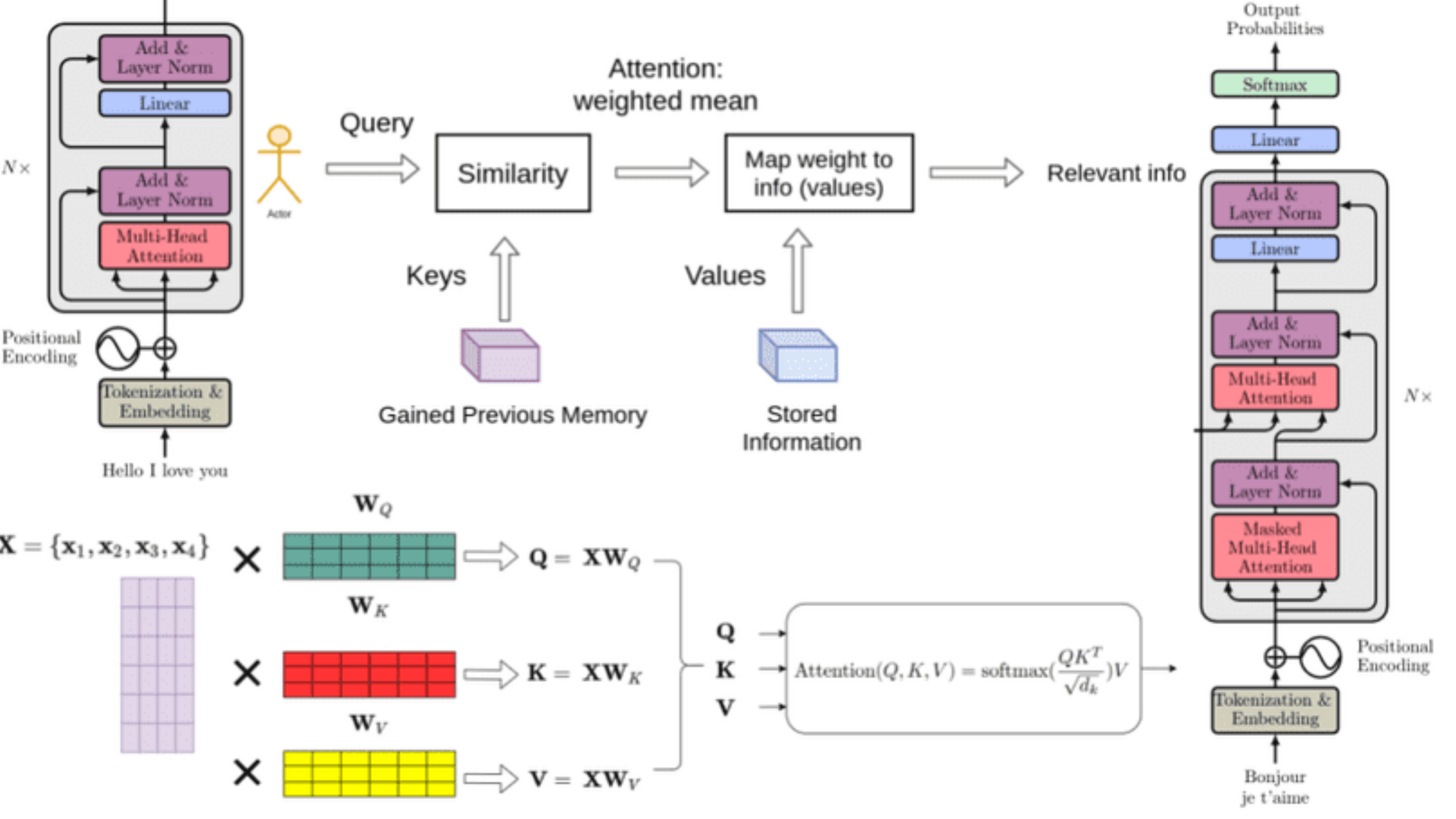


Attention in Transformer (Encoder-Decoder) Models




Layer Norm
Multilayer Perceptron
Layer Norm
MultiHead Self Attention
Input Embeddings
il
ce
a
m'
entarte
(Input Sequence)
Encoded Vectors
Layer Norm
Multilayer Perceptron
Layer Norm
MultiHead Self Attention
Layer Norm
Masked Multihead Self Attention
Output Embeddings
<Start>
He
hit
me
with
a
pie
(Output Sequence)
0
1
2
3
4
5
Linear Layer
Softmax
y
x
Output Probabilities
AutoRegression


TOKENS
Input embeddings
Queries
Queries
Queries
Keys
Keys
Keys
Scalar product
Scaling/ Softmax
Linear combination
Contextualized embeddings
Values
Values
Values



Add & Layer Norm
Linear
Add & Layer Norm
Multi-Head Attention
N×
Positional Encoding
Tokenization & Embedding
Hello I love you
Actor
Query
Attention: weighted mean
Similarity
Map weight to info (values)
Relevant info
Keys
Values
Gained Previous Memory
Stored Information
Output Probabilities
Softmax
Linear
Add & Layer Norm
Linear
Add & Layer Norm
Multi-Head Attention
N×
Add & Layer Norm
Masked Multi-Head Attention
Positional Encoding
Tokenization & Embedding
Bonjour je t'aime
$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
$\mathbf{W}_Q$
$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$
$\mathbf{W}_K$
$\mathbf{K} = \mathbf{X}\mathbf{W}_K$
$\mathbf{W}_V$
$\mathbf{V} = \mathbf{X}\mathbf{W}_V$
Q
K
V
$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$