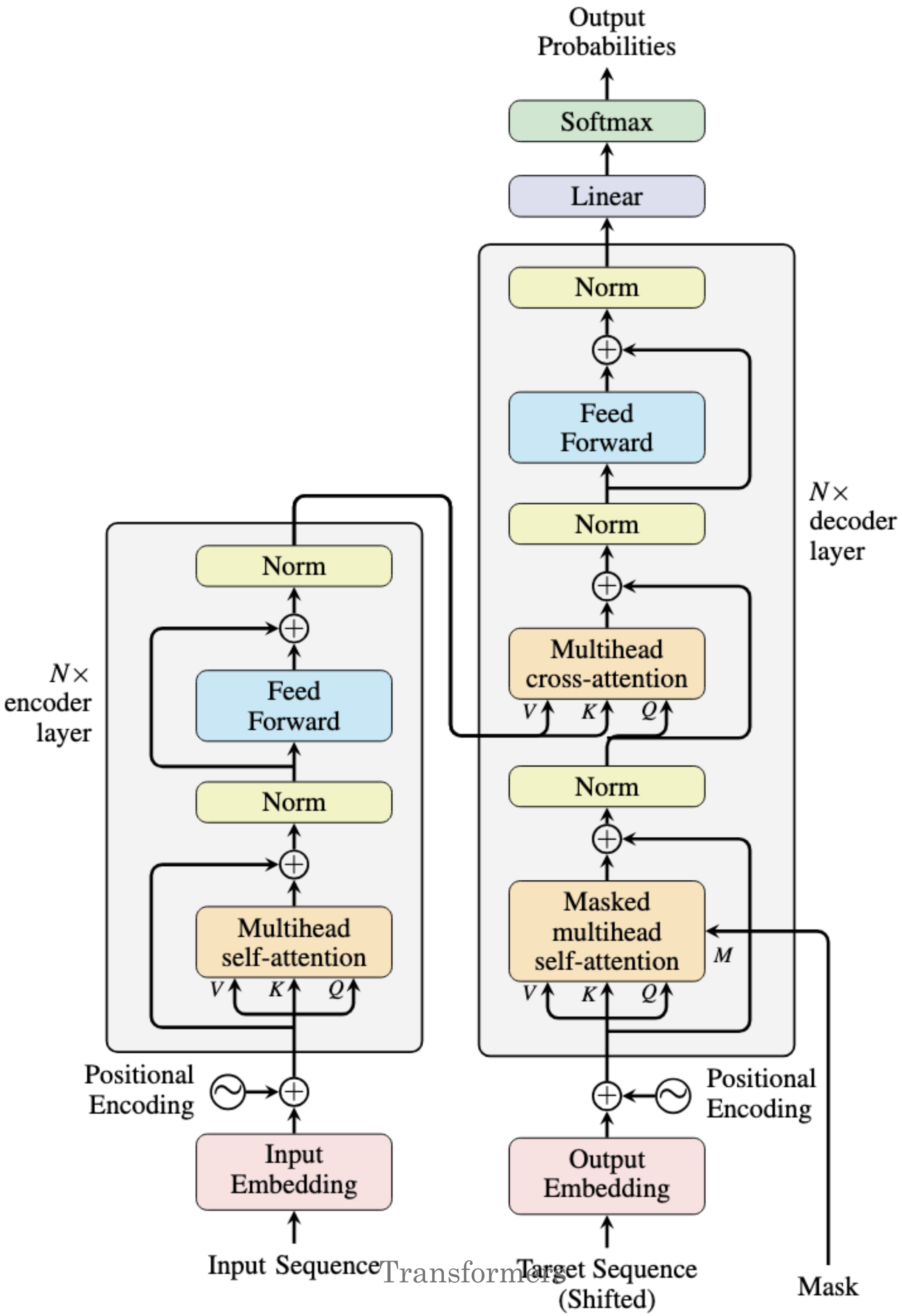
Transformers

Encoder Decoder Models


Output Probabilities
Softmax
Linear
Norm
Feed Forward
Norm
Multihead cross-attention
V K Q
Norm
Masked multihead self-attention
M
V K Q
N×
decoder
layer
Norm
Feed Forward
Norm
Multihead self-attention
V K Q
N×
encoder
layer
Positional Encoding
Input Embedding
Input Sequence
Positional Encoding
Output Embedding
Target Sequence (Shifted)
Mask

Tokenization

Motivation

Models can't understand raw text as characters or words directly.

Concept

Breaks text into "units of meaning" (tokens).

How It Helps

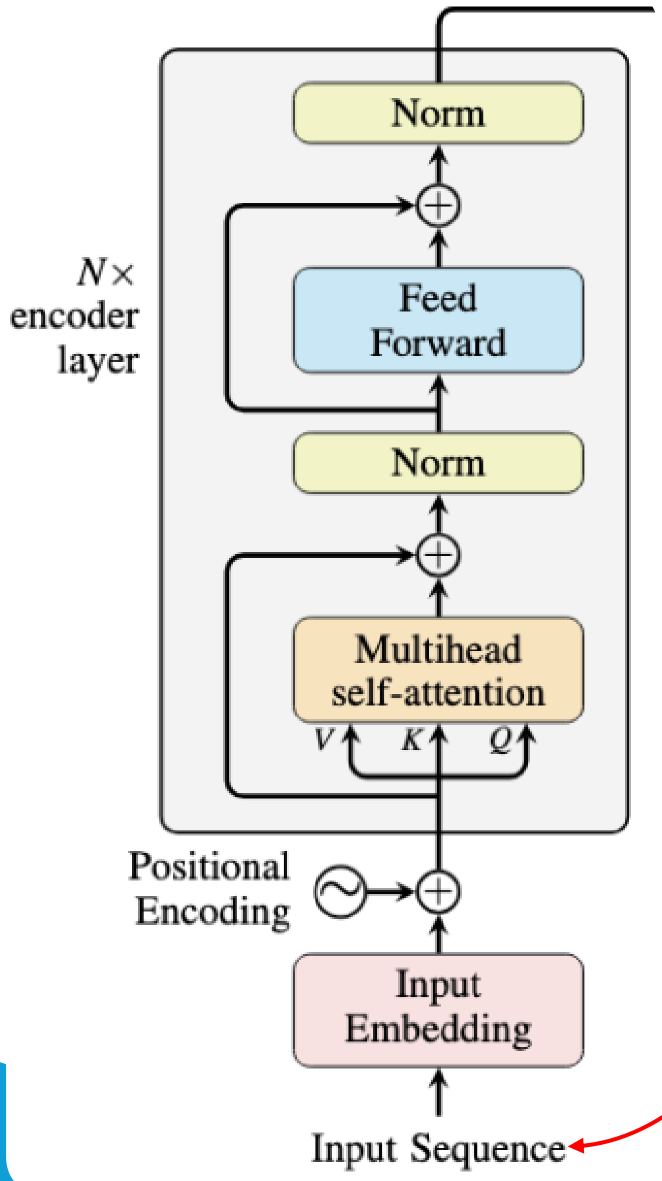
Converts language into manageable discrete pieces models can learn patterns over.

Takeaway

Tokenization turns language into learnable units.

Embeddings

Motivation	Token IDs are meaningless numbers; model needs semantic meaning.
Concept	Maps tokens to dense vectors where similar words have similar positions.
How It Helps	Captures meaning: “king” close to “queen”, “run” close to “walk”.
Takeaway	*Embeddings convert tokens into meaningful numerical representations.*



Input Sequence

Token embeddings (given)

- Apple: [1, 2, 1, 2]
- Innovative: [3, 2, 1, 2]
- Company: [4, 1, 2, 1]

Positional Encoding

Motivation

Transformer has **no recurrence** and **no convolution**, so it has no sense of order.

Concept

Adds position information to embeddings.

How It Helps

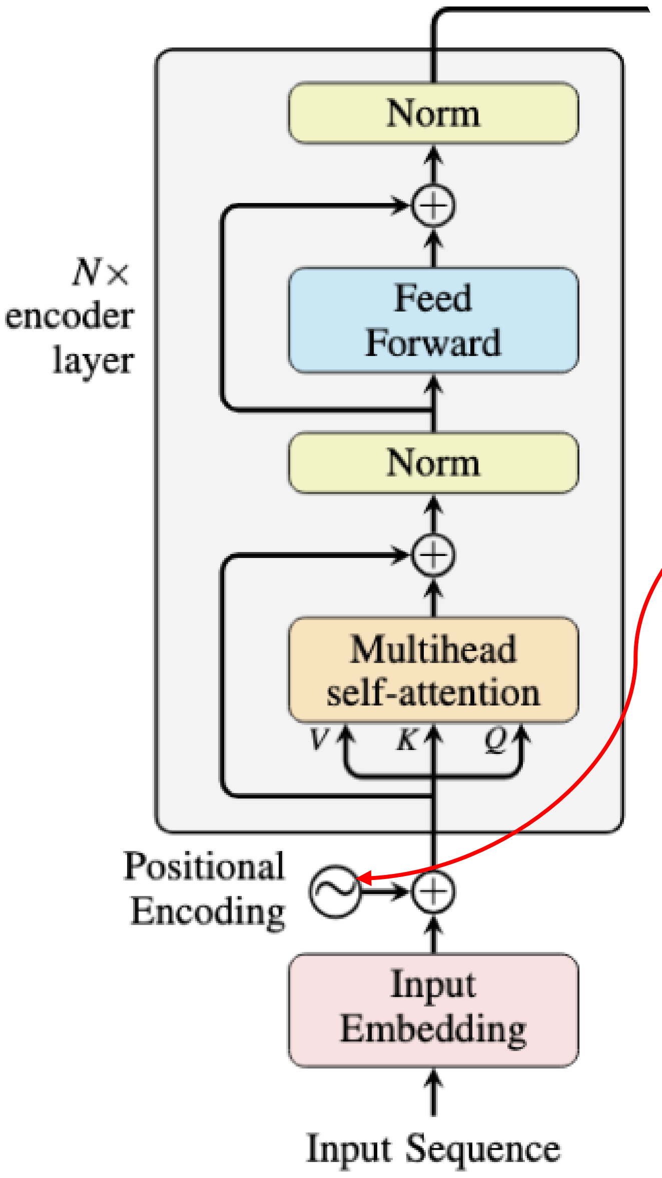
Allows model to know “who comes before whom”.

Takeaway

Transforms bag of words → ordered sentence understanding.

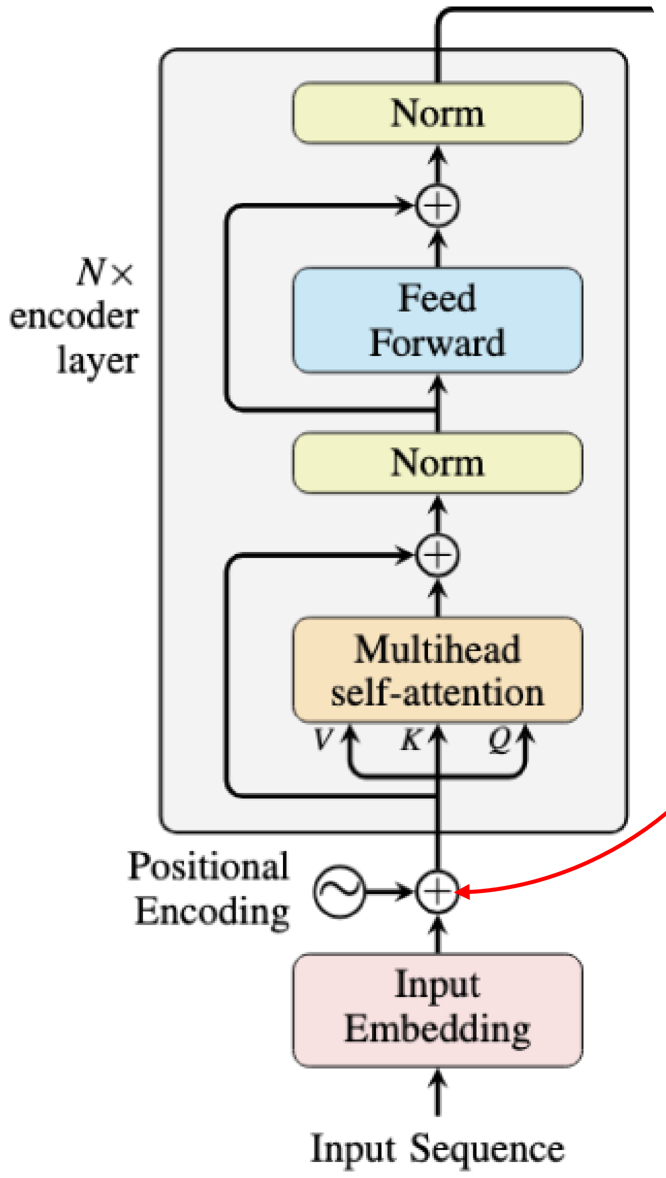
Positional Encoding -History

- **Why not Simple Number?**
- **Relative Difference?**
- **Limited Number?**



Positional Encoding

- Formulas (repeated):
- Numeric PE for positions 0,1,2 (rounded shown to 10 decimals)
- $$\text{PE} = \begin{bmatrix} 0.0000000000 & 1.0000000000 & 0.0000000000 & 1.0000000000 \\ 0.8414709848 & 0.5403023059 & 0.0099998333 & 0.9999500004 \\ 0.9092974268 & -0.4161468365 & 0.0199986667 & 0.9998000067 \end{bmatrix}$$
- Each row = positional vector for position 0,1,2.)



Position Encoding + Token embeddings

$$X_{pos} = X + PE$$

$$X + PE = \begin{bmatrix} 1.0000000000 & 3.0000000000 & 1.0000000000 & 3.0000000000 \\ 3.8414709848 & 2.5403023059 & 1.0099998333 & 2.9999500004 \\ 4.9092974268 & 0.5838531635 & 2.0199986667 & 1.9998000067 \end{bmatrix}$$

Attention

Motivation

Not all words in a sentence are equally important.

Concept

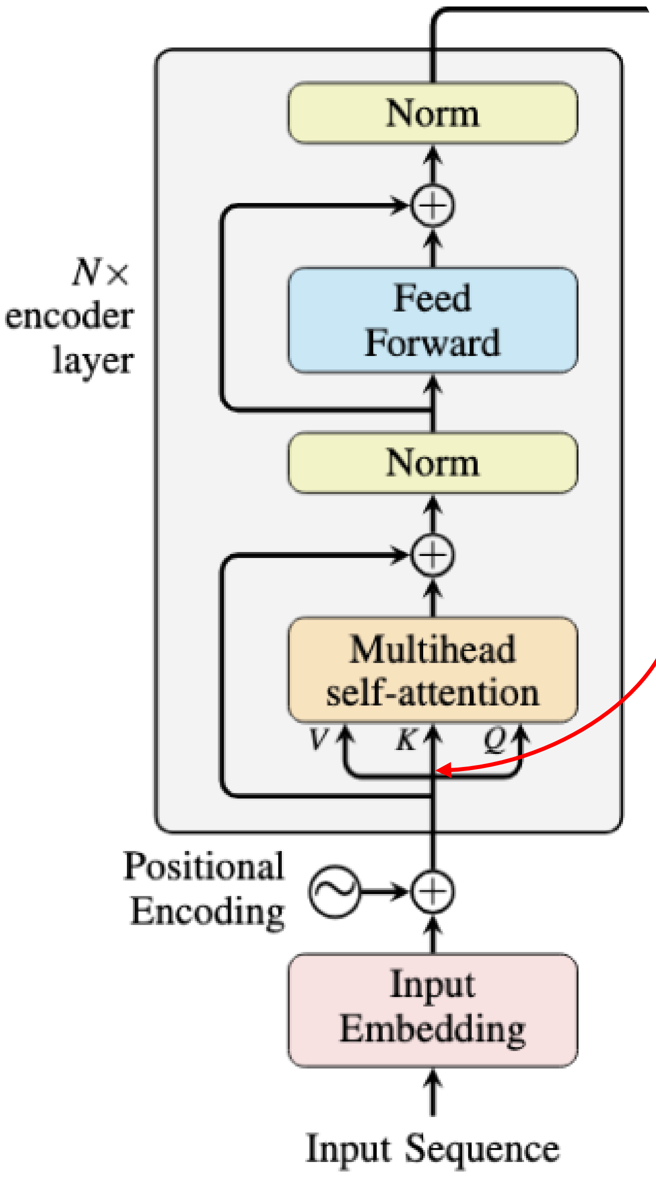
Model learns **where to focus** dynamically.

How It Helps

Captures relationships: “The animal didn’t cross the road because **it** was tired”
→ understands “it = animal”

Takeaway

Attention lets the model think selectively rather than equally.



Query , Key and Value

Query	Key	Value
[[1, 2, 3, 1], [4, 1, 2, 3], [1, 1, 2, 2], [1, 2, 1, 1]]	[[1, 2, 2, 1], [1, 2, 2, 3], [1, 3, 4, 2], [1, 2, 3, 1]]	[[1, 2, 2, 1], [1, 2, 2, 3], [1, 3, 4, 2], [1, 2, 3, 1]]

Why Do We Need Q, K, and V?

Motivation

Attention is about **deciding what information is relevant**.

We need a mechanism to:

- Ask a question
- Check which parts match the question
- Retrieve useful information

Concept

Transformer separates these roles into **Query (Q)**, **Key (K)**, and **Value (V)**.

Takeaway

Attention works like searching and retrieving information.

What is Query (Q)?

Each word wants to ask:
"Which other words are important for me?"

Query represents **what this word is looking for**.

You ask a question in Google:

- "Best laptop for AI"
- This question = **Query**

Query = the question being asked.

What is Key (K)?

The model needs something to **match against the query**.

Keys describe **what each word offers**.

Webpage titles & descriptions in Google search:

- Each page has metadata
- Google compares your query to this metadata
- Metadata = **Keys**

Key = what can be matched with the query.

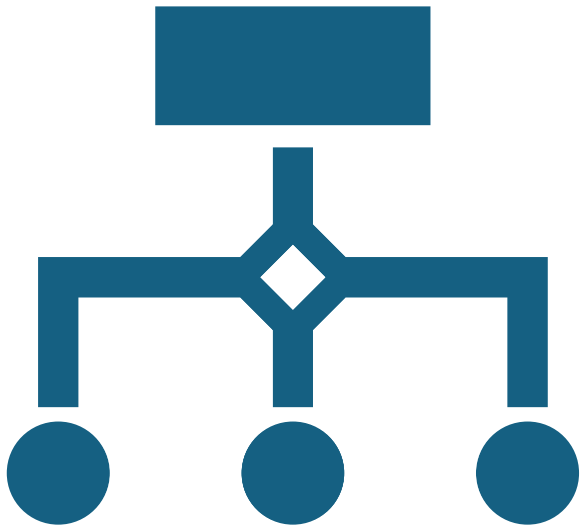
What is Value (V)?

Once relevance is known, we need **actual information**.

Value contains the **information to pass forward**.

- The actual content of the webpage you open
- Content = **Value**

Value = the information you retrieve.



Why Q, K, and V Must Be Different

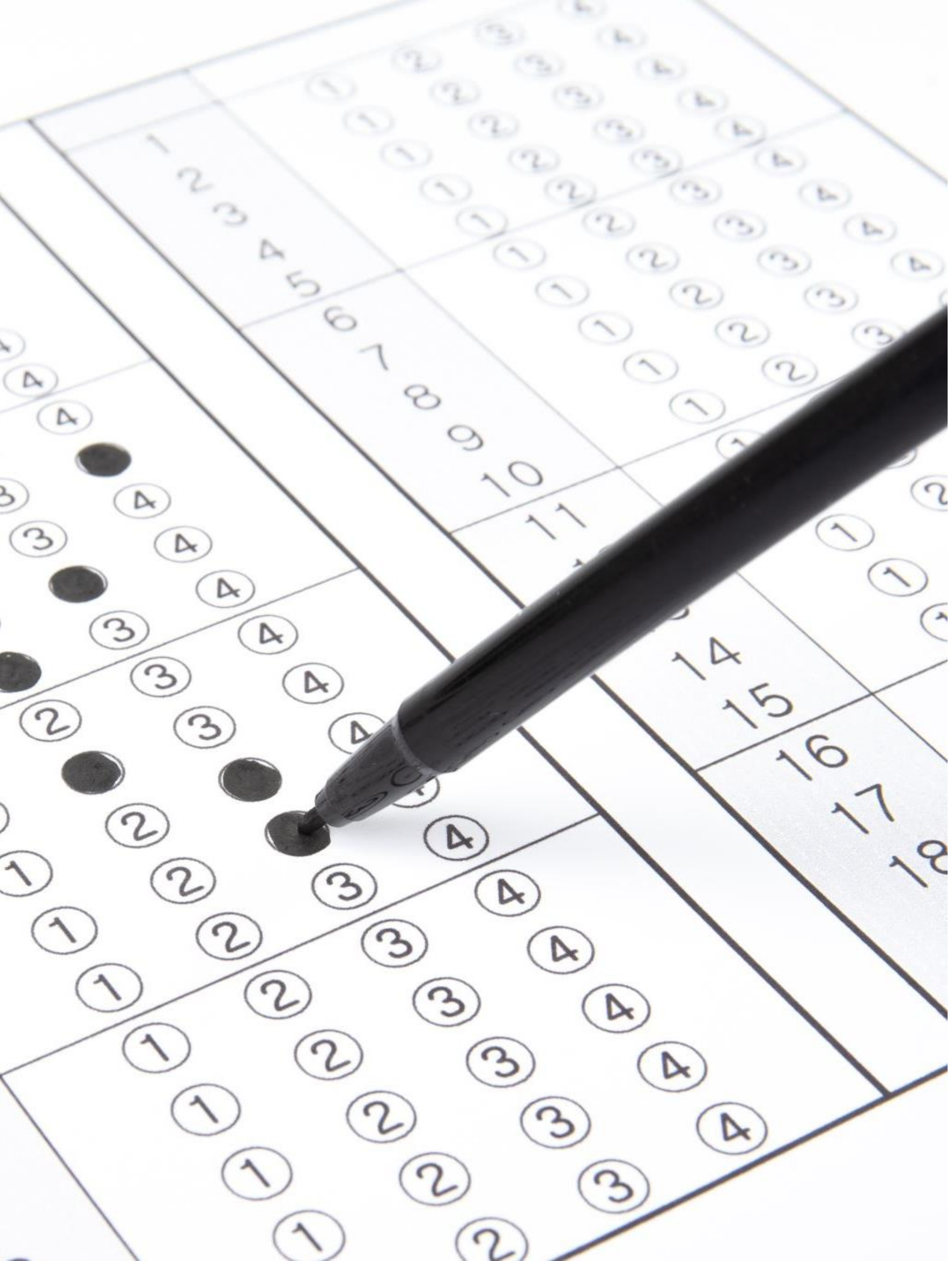
One representation cannot:

- Ask a question
- Describe itself
- Provide information
  all at once effectively.

Separate learned projections allow:

- Q → asking
- K → matching
- V → transferring information

Different roles require different representations.



Why Are Q, K, V Learned (Trainable Matrices)?

- Fixed rules can’t capture complex language patterns.

- Model learns:
  - What to ask (Q)
  - How to match (K)
  - What information matters (V)

- A student learns:
  - How to ask better questions
  - How to recognize good answers
  - Which parts of answers are important
- *Q, K, V are learned to adapt to language.*

Intuition

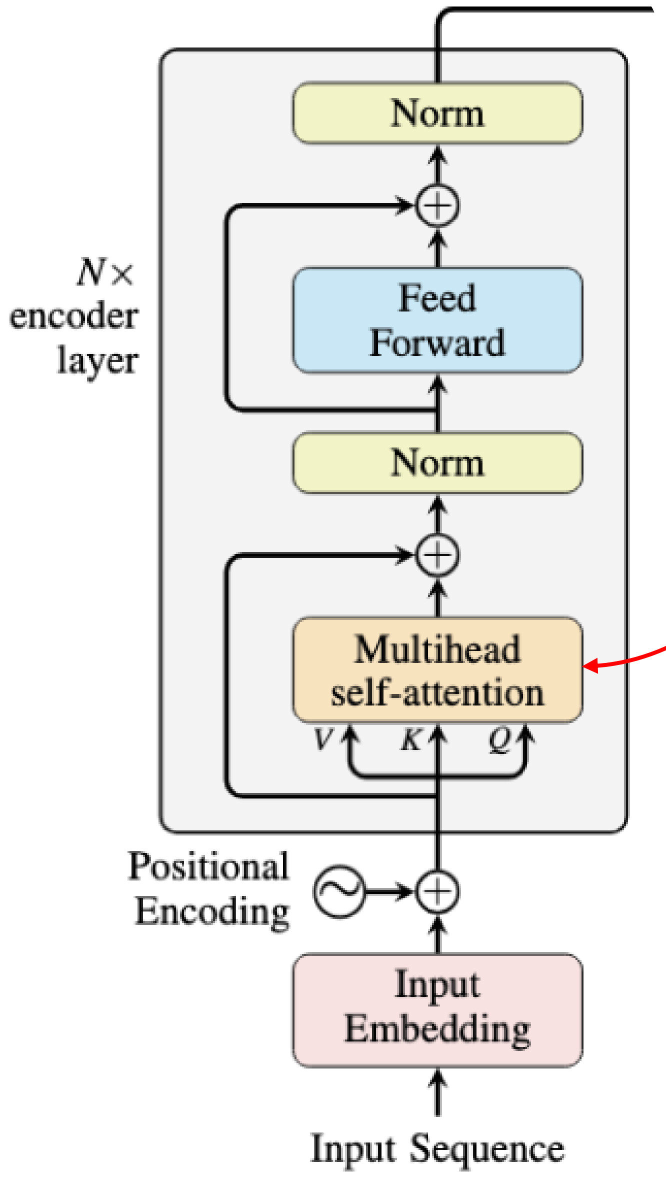
- **Scenario**

  Teacher asks a question.

  - **Query** → The question being asked
  - **Keys** → Students raising hands with topic tags
  - **Values** → Actual answers students give
- **Attention**
  - Teacher listens more to students whose raised-hand topic matches the question.
- **Takeaway**
  - *Attention selects values using Q-K similarity.*

Summary

- Q → What am I looking for?
- K → What do I contain?
- V → What information do I give?



X_{pos} * (**Query, Key and Value)**

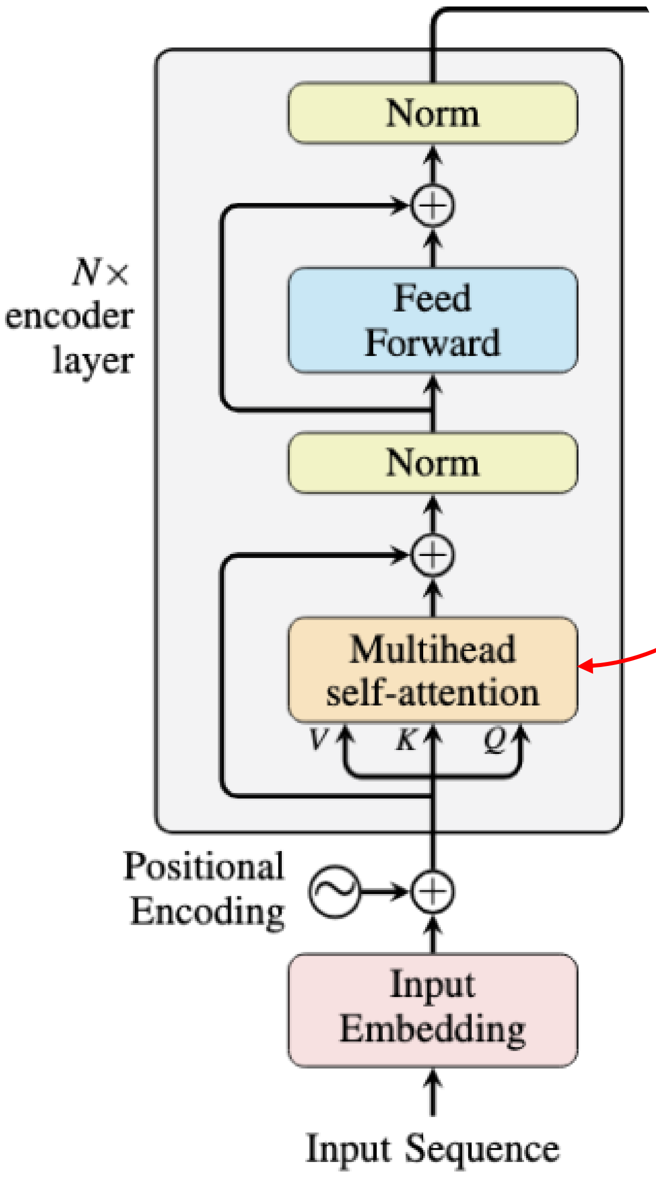
- $Q = \begin{bmatrix} 17.0000000000 & 12.0000000000 & 14.0000000000 & 15.0000000000 \\ 18.0126300420 & 17.2331441097 & 21.6249672332 & 16.4823275695 \\ 11.2645087540 & 16.4220466971 & 21.9353959474 & 12.7006542572 \end{bmatrix}$

- $K = \begin{bmatrix} 8.0000000000 & 17.0000000000 & 21.0000000000 & 15.0000000000 \\ 10.3917231244 & 21.7934460822 & 25.8033959159 & 16.4823275695 \\ 9.5129492636 & 21.0458971940 & 25.0656958673 & 12.7006542572 \end{bmatrix}$

- $V = \begin{bmatrix} 17.0000000000 & 13.0000000000 & 11.0000000000 & 9.0000000000 \\ 18.0126300420 & 18.2431439430 & 12.9320254303 & 11.4017229578 \\ 11.2645087540 & 18.4420453638 & 10.0968024271 & 11.5329479303 \end{bmatrix}$

$Q \cdot K^\top$ in Attention

- **Purpose:** Measures how much each query should focus on each key.
- **Intuition:** Dot product shows **similarity/alignment** between query and key vectors.
- **Result:** Produces a **score matrix** used to weight values and generate attention output.

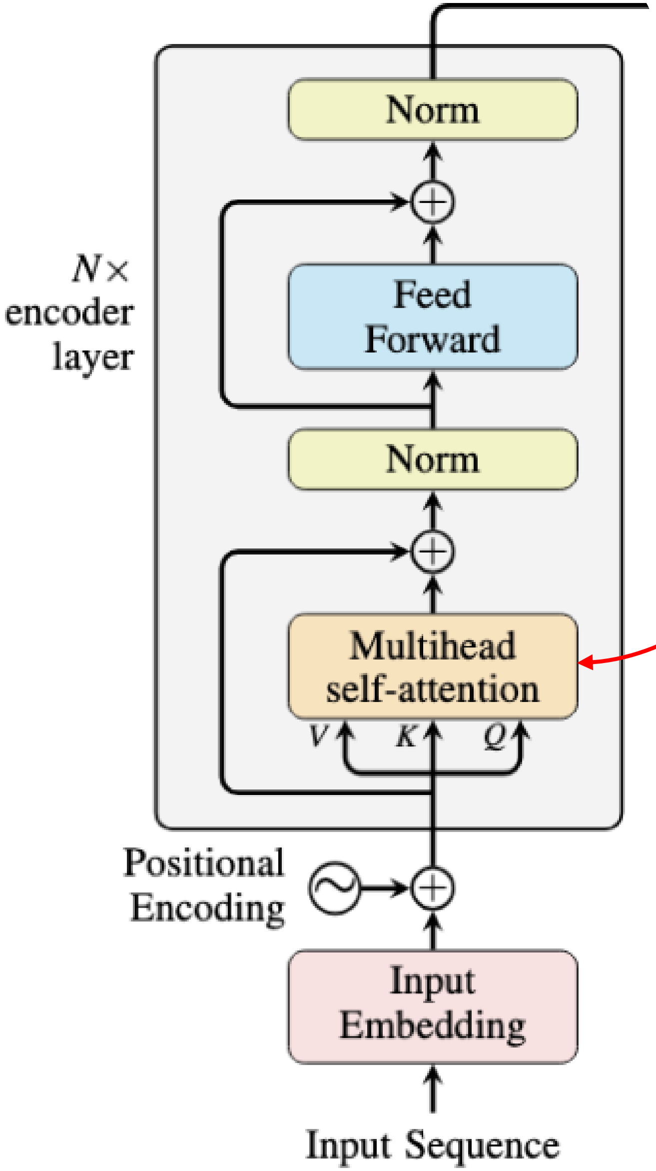


Attention score matrix $S = \frac{QK^T}{\sqrt{d_k}}$

With $d_k = 4$,$\sqrt{d_k} = 2$.

- First compute QK^Tthen divide by 2. Numeric result (3×3):

$$S = \frac{QK^T}{2} = \begin{bmatrix} 429.5000000000 & 523.3315512336 & 477.8502299053 \\ 569.2118578205 & 696.2082872083 & 642.7107052897 \\ 510.2219963190 & 625.1473476883 & 581.9539944344 \end{bmatrix}$$

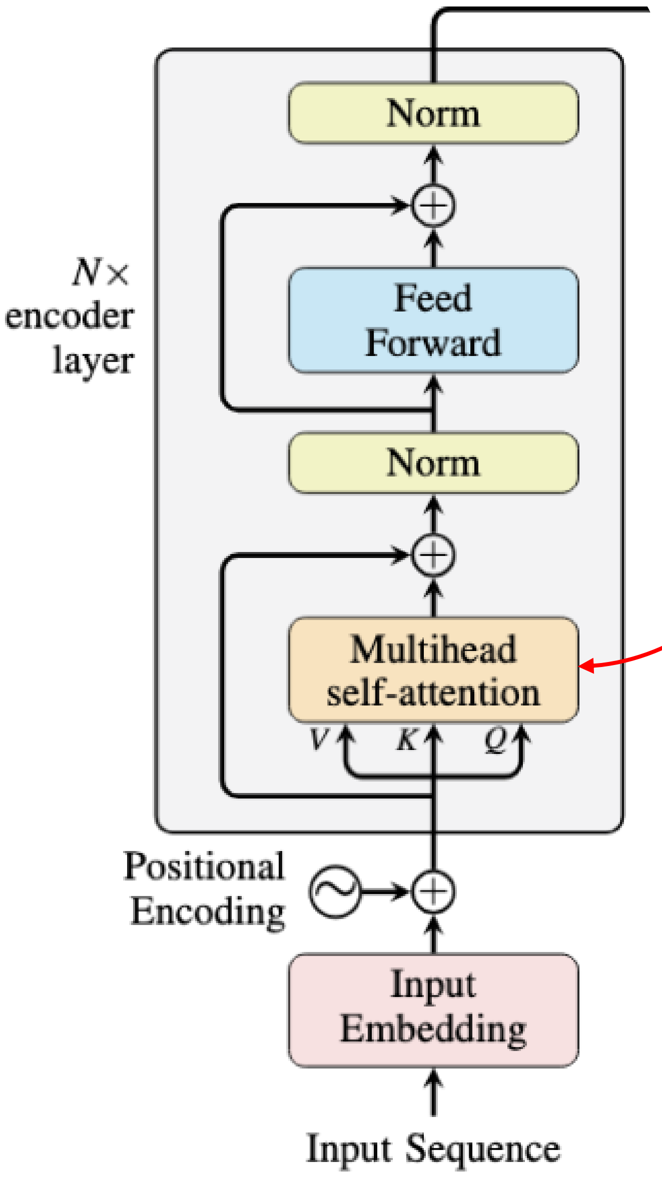


Attention Score matrix $\boldsymbol{S} = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})$

$$S = \text{Softmax}\left(\left(\frac{QK^T}{\sqrt{d_k}}\right) V\right) = \begin{bmatrix} 0 & 1.0000 & 0 \\ 0 & 1.0000 & 0 \\ 0 & 1.0000 & 0 \end{bmatrix}$$

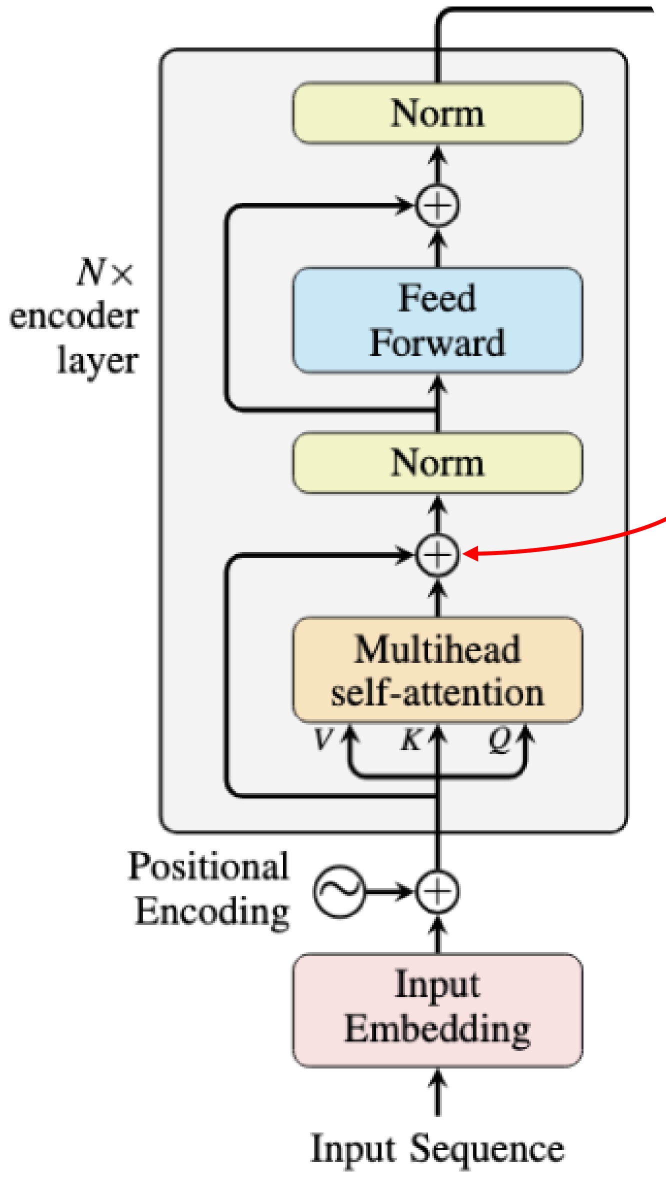
SoftMax in Attention

- Raw attention scores are not interpretable.
- Converts scores → probabilities.
- Ensures values are normalized and comparable.
- *Softmax makes attention a meaningful "importance distribution".*



Attention Score matrix S = Softmax$\left(\frac{QK^T}{\sqrt{d_k}}\right)$ V

$$S = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V = \begin{bmatrix} 10.3917 & 21.7934 & 25.8034 & 16.4823 \\ 10.3917 & 21.7934 & 25.8034 & 16.4823 \\ 10.3917 & 21.7934 & 25.8034 & 16.4823 \end{bmatrix}$$



Attention + PE

$$Output = S + PE = \begin{bmatrix} 10.3917 & 22.7934 & 25.8034 & 17.4823 \\ 11.2332 & 22.3337 & 25.8134 & 17.4823 \\ 11.3010 & 21.3773 & 25.8234 & 17.4821 \end{bmatrix}$$

Residual Connections

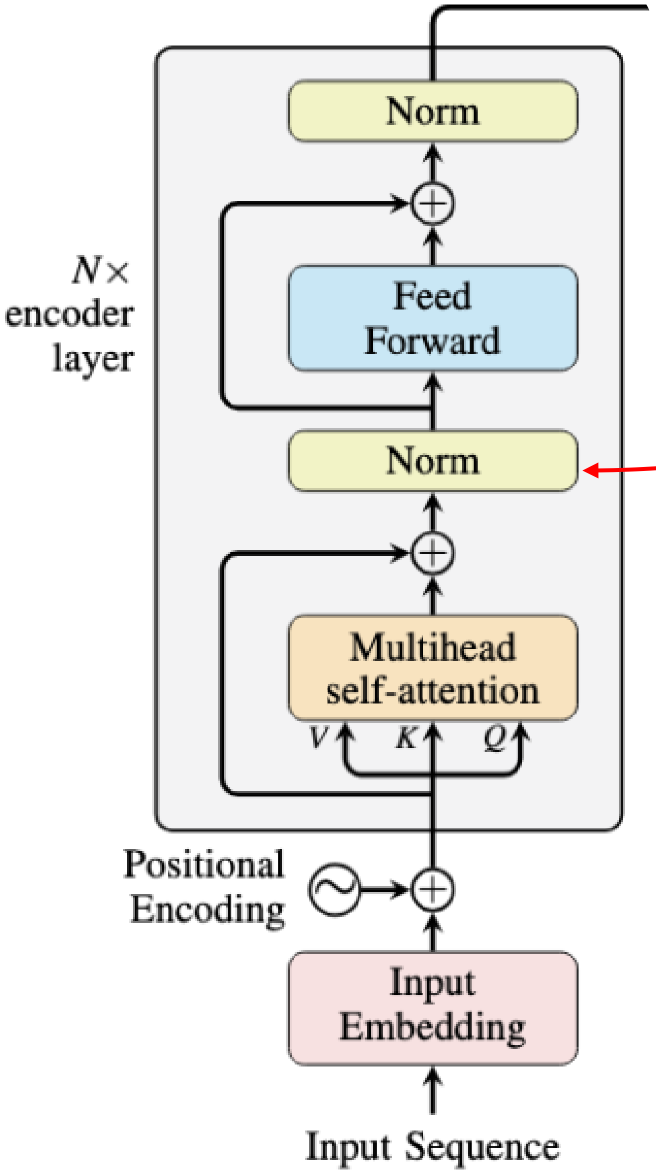
1. **D**eep networks = vanishing gradient / information loss.
2. Adds shortcut path.
3. Stabilizes training, preserves information.
4. *Prevents forgetting useful signals.*

Normalization

Why Normalization?

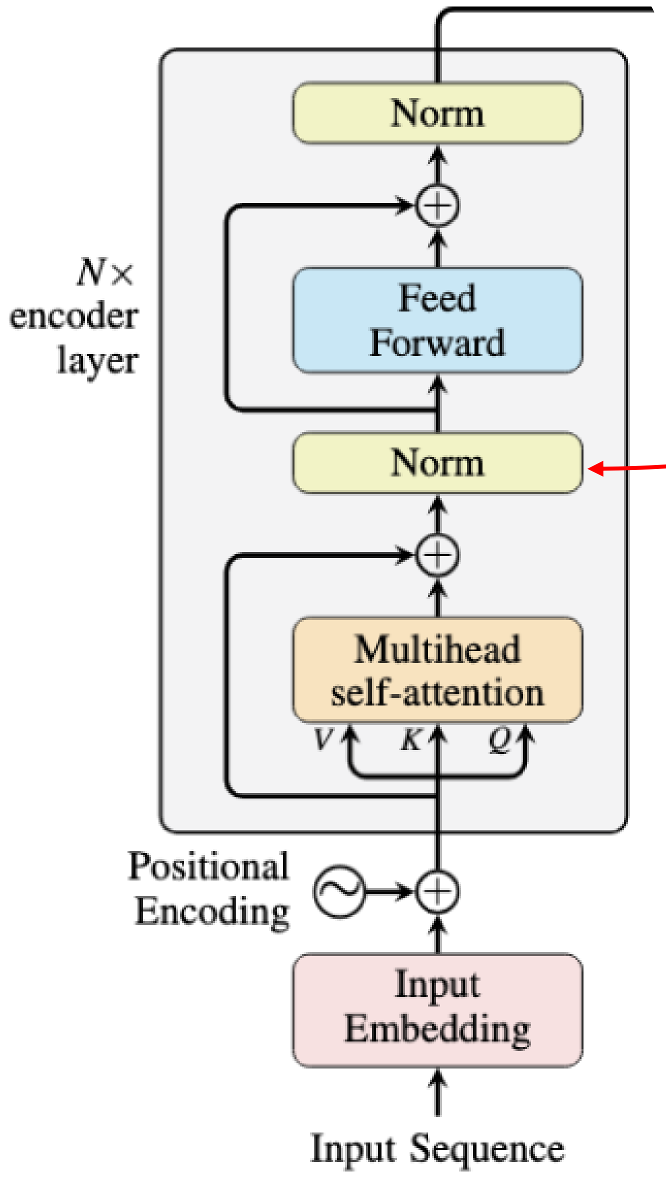
Batch Normalization

Layer Normalization



Layer Normalization

- For each row x:
- $\mu = \frac{1}{d}\sum x_i$
- $\text{LayerNorm}(x_i) = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$
- d :number of features
- ϵ: numerical stability



Layer Normalization

$$\textbf{Output} = \begin{bmatrix} -1.4909 & 0.6280 & 1.1423 & -0.2794 \\ -1.4575 & 0.5693 & 1.2047 & -0.3165 \\ -1.4427 & 0.4465 & 1.2801 & -0.2838 \end{bmatrix}$$

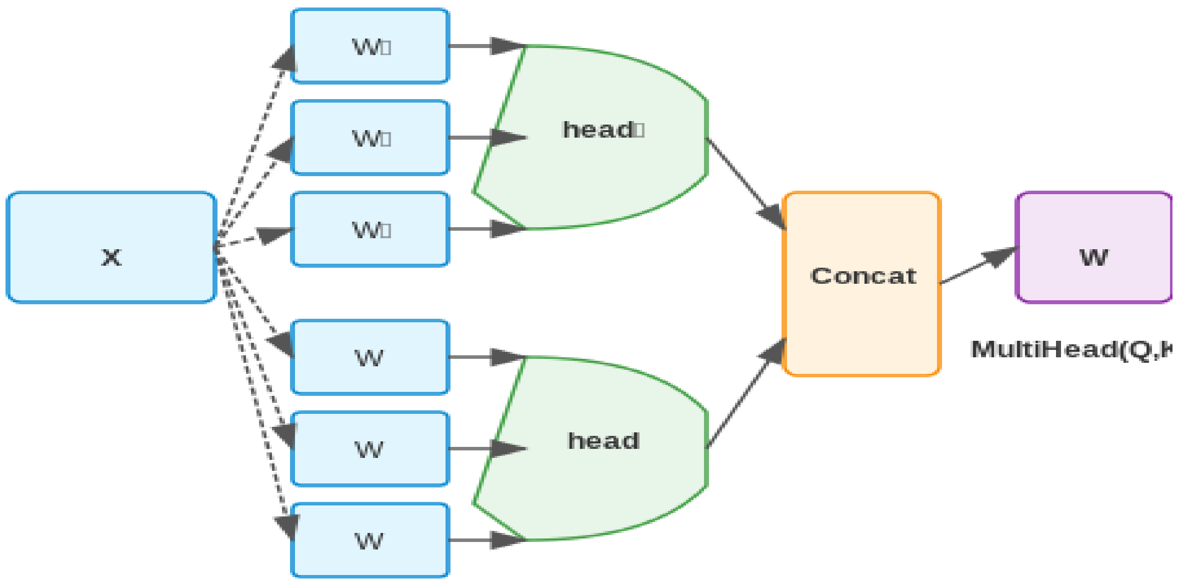
Layer Normalization

1. Activations explode / vanish → unstable model.
2. Normalizes output distributions.
3. Faster, more stable training.
4. *Keeps learning balanced and controlled.*

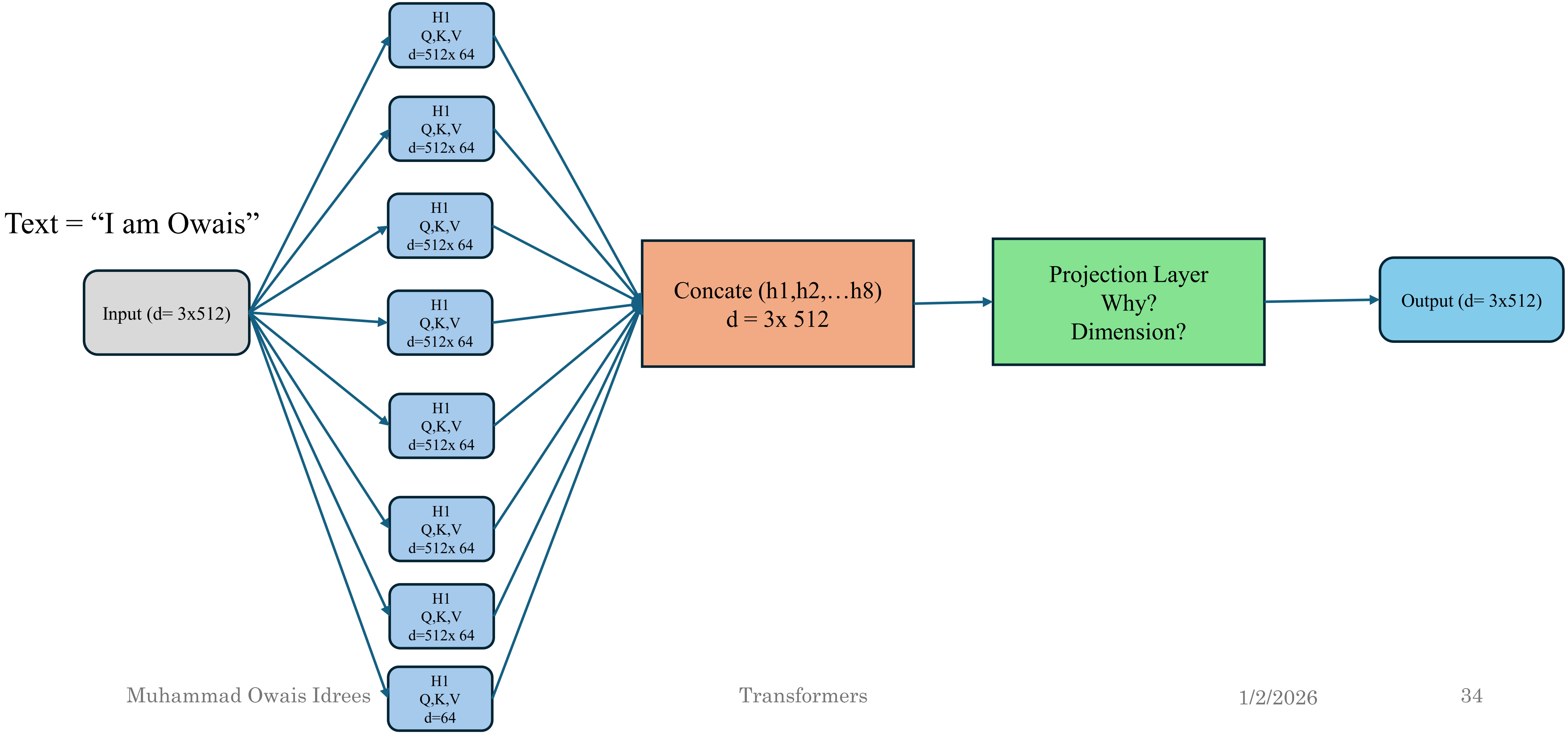
Multi-Head Attention

- **Motivation**
  - One attention head = one perspective → too limited.
- **Concept**
  - Multiple heads learn **different relationships** (syntax, meaning, position…).
- **How It Helps**
  - Richer understanding.
- **Takeaway**
  - *Multiple heads = multiple viewpoints.*

Multi-Head Attention

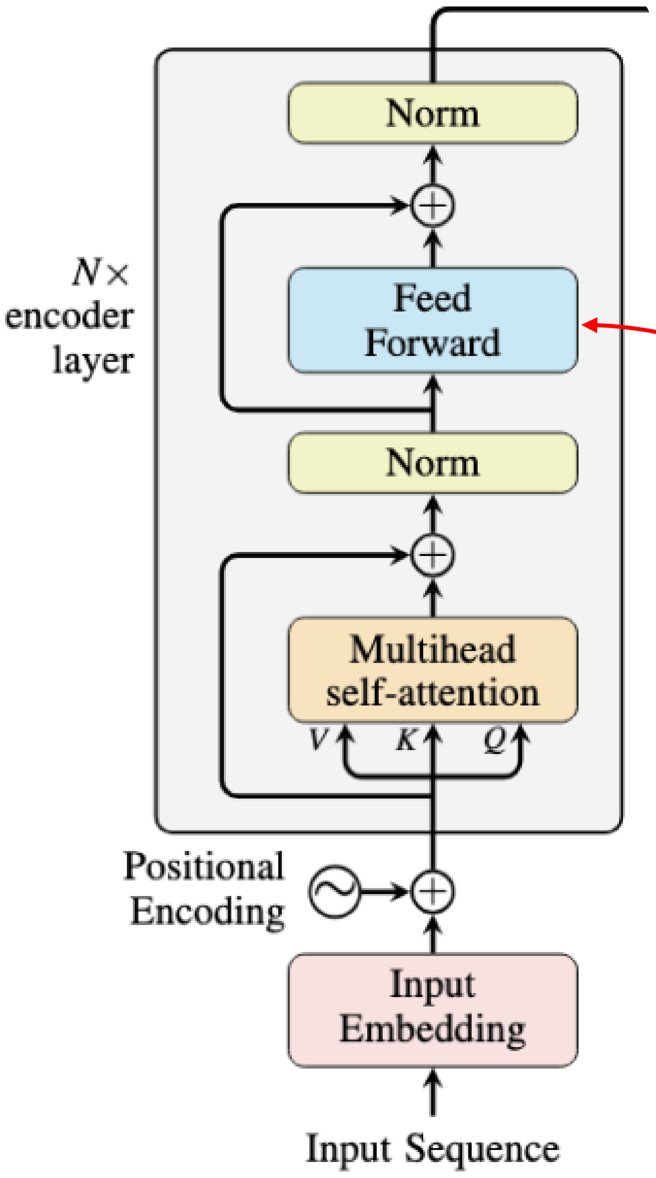


Multi-Head Attention



Feed Forward Network

1. Attention alone only mixes information; no transformation.
2. Applies non-linear learning per token.
3. Learns complex features.
4. *Adds intelligence beyond just "mixing words".*



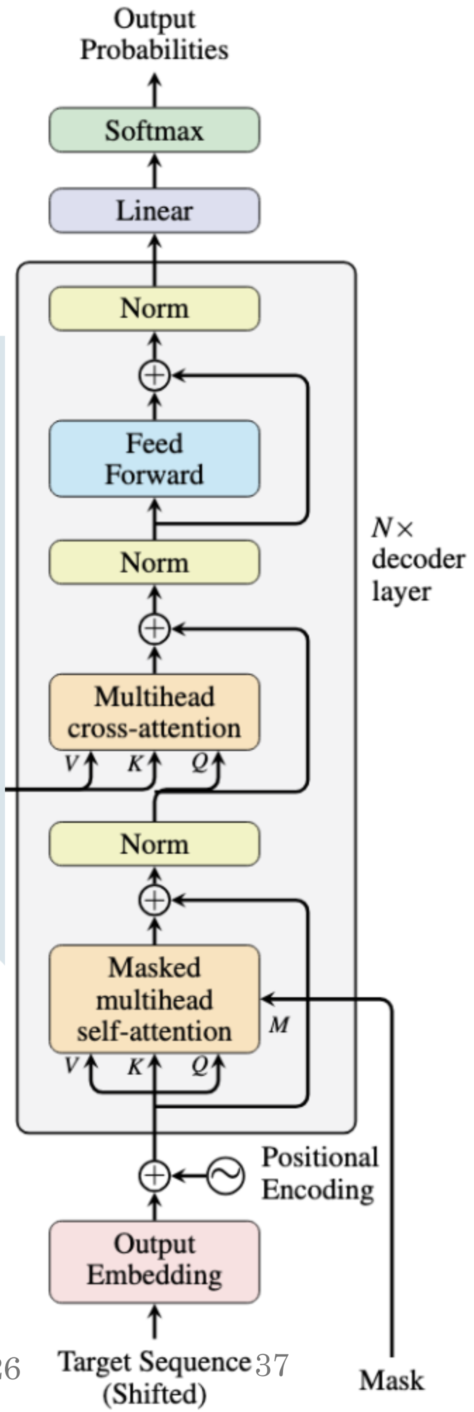
Feed Forward

Model	d_{model}	**Hidden neurons (d_{ff})**
Transformer (Original)	512	**2048**
BERT Base	768	**3072**
BERT Large	1024	**4096**
GPT-2 Small	768	**3072**

How many neurons should be used in the hidden layer and output layer of our model?

The Transformer FFN expands each token from d_{model} to $4 \times d_{model}$ neurons in the hidden layer before projecting back.

Decoder


Output Probabilities
Softmax
Linear
Norm
Feed Forward
Norm
Multihead cross-attention
V K Q
Norm
Masked multihead self-attention
M
V K Q
N× decoder layer
Positional Encoding
Output Embedding
Target Sequence (Shifted)
Mask

Decoder Part

The Decoder consists of:

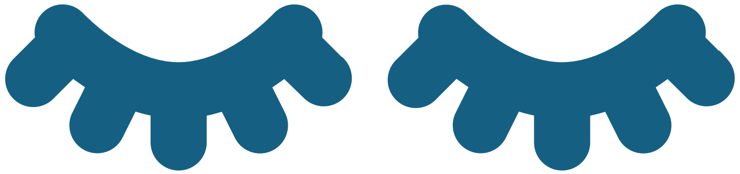
Masked Self-Attention
Cross (Encoder–Decoder) Attention
Feed Forward Network
Add & Layer Normalization

Key difference from Encoder:

Uses **masking**
Uses **cross attention** with encoder output

Masked Self-Attention

- Masked Self-Attention allows the decoder to:
  - Attend only to **past tokens**
  - Prevent access to **future tokens**
- This ensures **autoregressive generation**.



Why Do We Mask?

- During training, the full target sequence is available
- But during inference, future tokens are unknown
- Masking prevents **information leakage**

- **Key Idea:**
    - A token should not see future tokens.

How Do We Mask?

- We apply a **look-ahead mask**
- Upper triangular matrix is masked
- Masked positions are set to $-\infty$

- Before softmax:

$$QK^\top + Mask$$

- After softmax:

Masked positions → probability = 0

Masking Matrix (Example)

For a 3-token sequence:

$$\text{Mask} = \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{bmatrix}$$

- Raw Attention Scores:

$$QK^\top = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix}$$

- Apply Mask:

$$\begin{bmatrix} 2 & -\infty & -\infty \\ 3 & 5 & -\infty \\ 4 & 6 & 8 \end{bmatrix}$$

Masked Attention

After Softmax:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.12 & 0.88 & 0 \\ 0.02 & 0.12 & 0.86 \end{bmatrix}$$

Masked Attention

Decoder Masked Attention Flow

$$\text{MaskedAttention}(Q, K, V) = softmax\left(\frac{QK^{\top}}{\sqrt{d_k}} + Mask\right)V$$

- Q, K, V all come from **decoder**
- Mask is applied before softmax

After Masked Attention

- **What Happens?**
  - Same as Encoder:
    - Add & LayerNorm
    - Output is passed to **Cross Attention**

Cross Attention

- Cross Attention connects:
  - **Decoder queries**
  - **Encoder outputs**
- Purpose:
  - Decoder focuses on **relevant encoder tokens**

Cross Attention

Component	Comes From
Queries (Q)	Decoder
Keys (K)	Encoder
Values (V)	Encoder

Cross Attention

Assume encoder output:

$$E = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Decoder representation:

$$D = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Scores=Qdecoder * Kencoder⊤

Result:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Cross Attention

After softmax:

$$\begin{bmatrix} 0.33 & 0.33 & 0.33 \\ 0.33 & 0.33 & 0.33 \end{bmatrix}$$

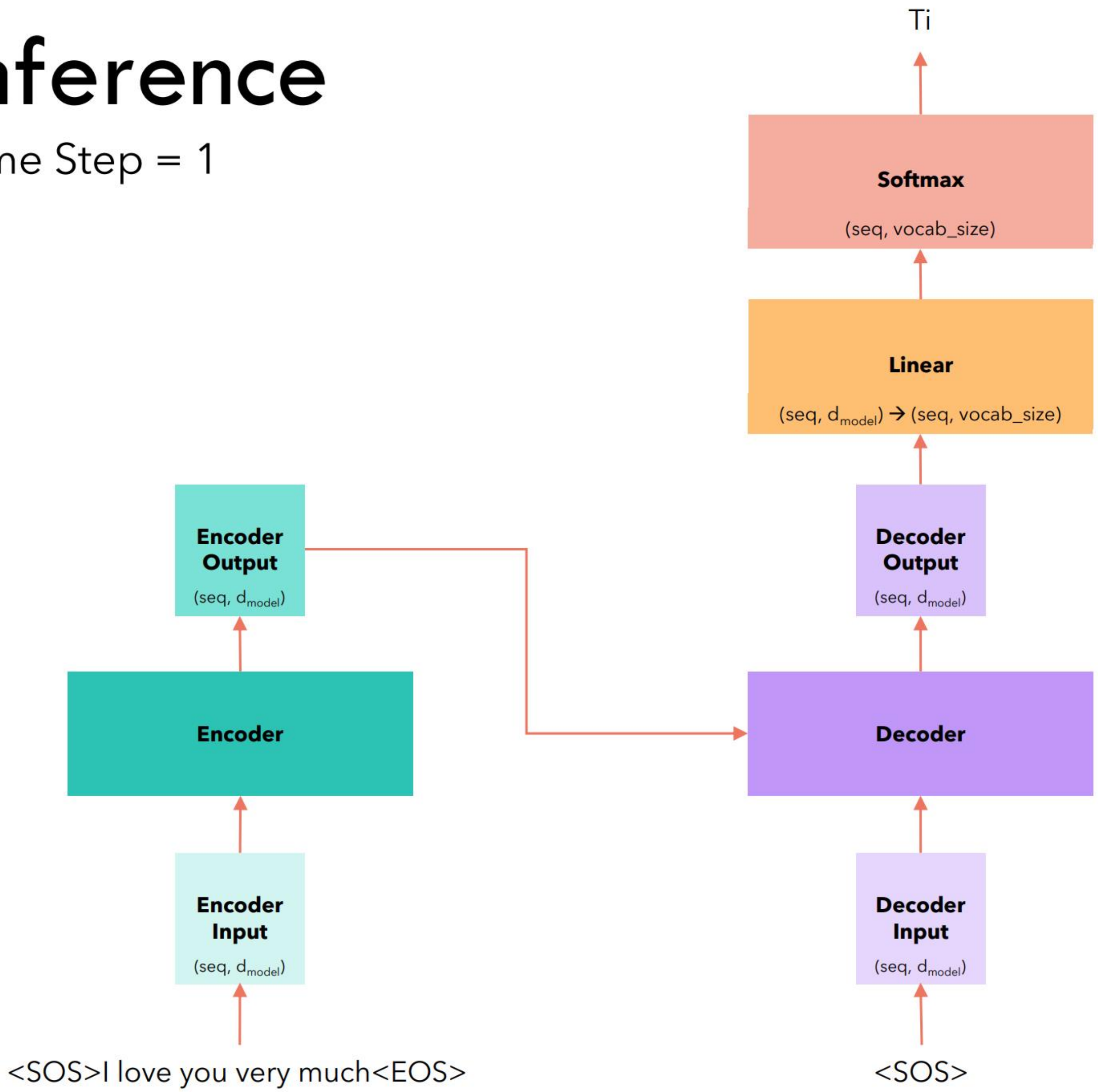
Final Output:

$$\text{Attention} \times V_{encoder}$$

→Decoder selectively absorbs encoder information.

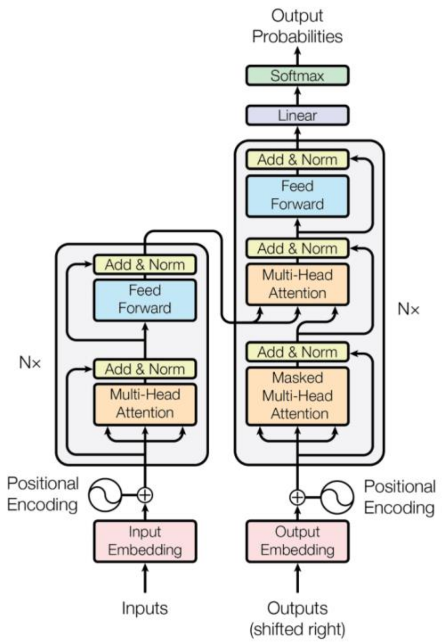
Inference

Time Step = 1



We select a token from the vocabulary corresponding to the position of the token with the maximum value.

The output of the last layer is commonly known as **logits**



* Both sequences will have same length thanks to padding