Transformers

AI with Deep Learning
EE4016

Prof. Lai-Man Po

Department of Electrical Engineering
City University of Hong Kong

https://medium.com/@lmpo/the-transformer-architecture-and-the-power-of-self-attention-6a0b59005c11

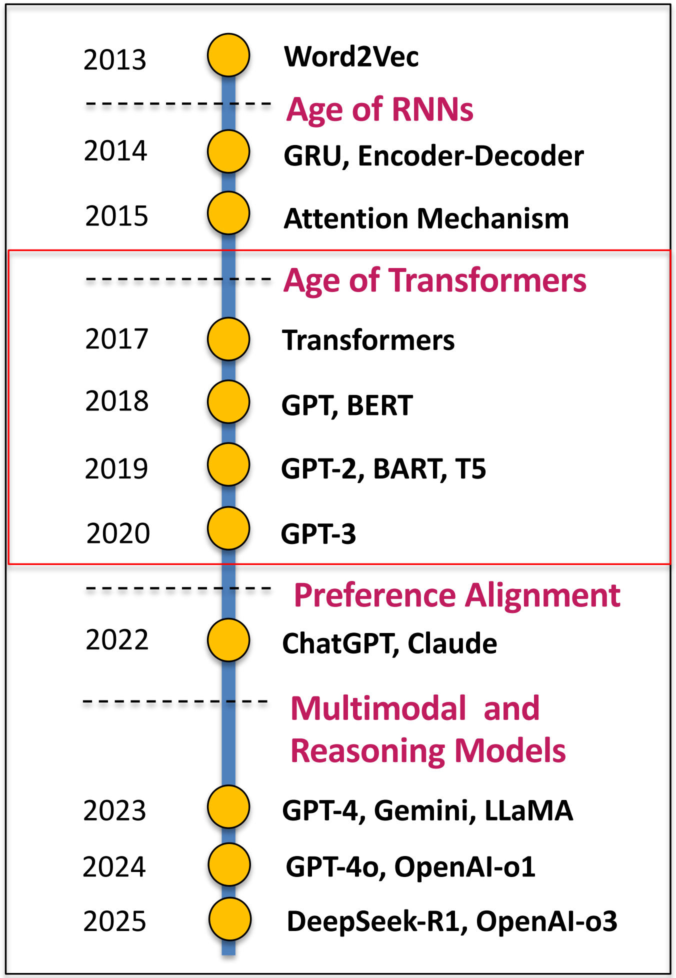
Message: Assignment 2

Image Classification with CNN

- The assignment 2 is now available in the schedule webpage for download. The deadline for the assignment 2 is **Saturday of Week 9 (Nov. 1, 2025).**
  - https://www.ee.cityu.edu.hk/~lmpo/ee4016/pdf/2026_EE4016_Ass02.pdf
  - Colab: https://colab.research.google.com/drive/1W-CpyU3mwWr_ueSm_86C-do6pm361VMe?usp=sharing#scrollTo=4tPgLqmBveuL
- **The answers of the section A must be handwritten** and then scan the answer sheets into a single pdf file.
- Submit the answer sheets and Colab notebook of the Assignment 2 as a zip file to this CANVAS assignment 2:
  - Filename format : Assignment02_StudentName_StudentID.zip
  - Filename example: Assignment02_Chen_Hoi_501234567.zip

LLMs: From Word2Vec to DeepSeek-R1 (2012-2025)

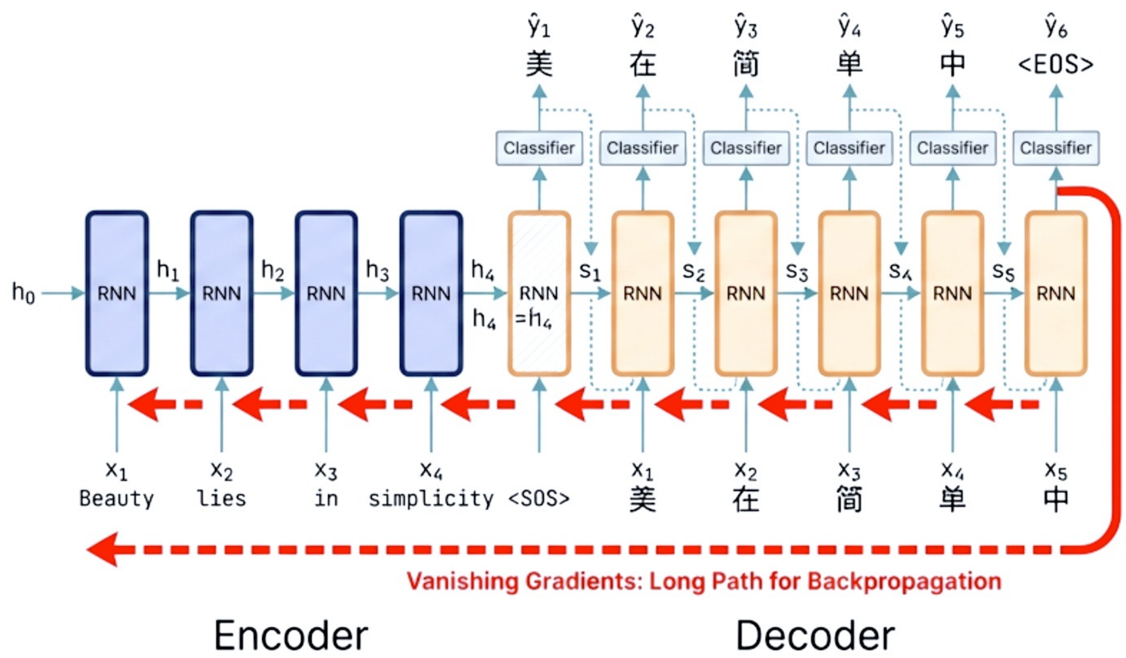
1. Tokenization and Word2Vec: BPE, CBOW, Skip-Gram
2. RNNs, LSTM, GRU, Seq-to-Seq (Encoder-Decoder) and Attention Mechanisms
3. Transformers with Self-Attention
4. Lager Language Models (LLMs): BERT, GPT, BART, T5
5. Preference Alignment by SFT and RLHF: ChatGPT, Claude, LLaMA
6. Multimodal Models: GPT-4, GPT-4o, Gemine, LLaVA
7. Reasoning Models: OpenAI-o1, DeepSeek-R1



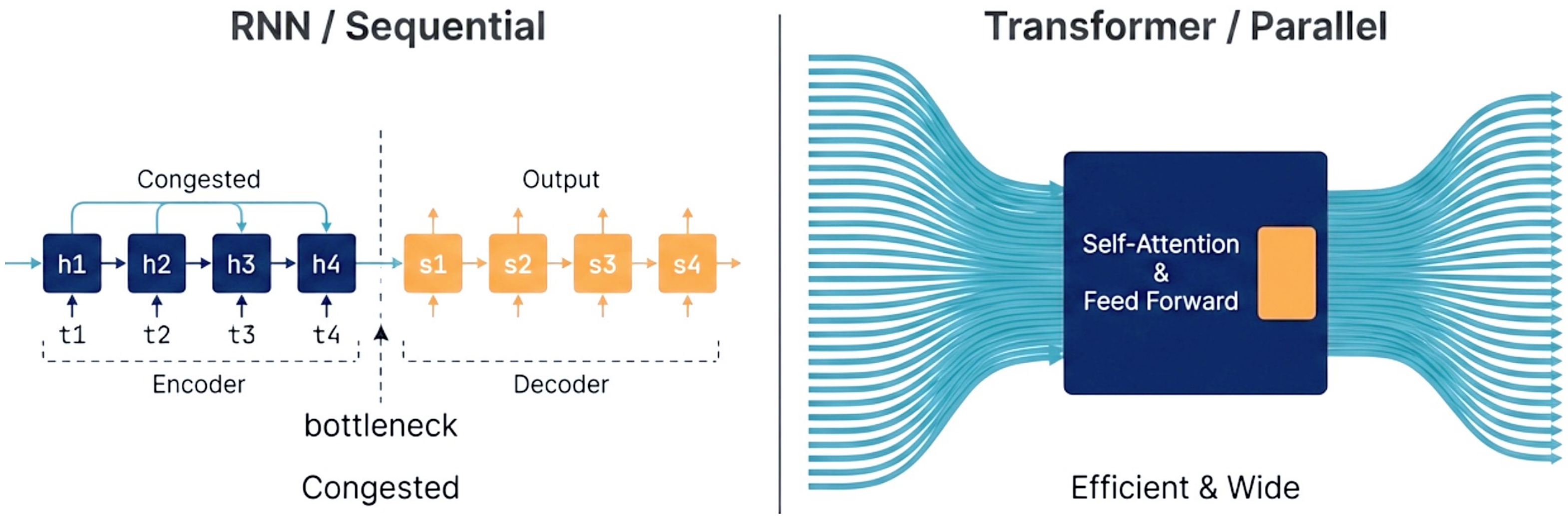
The Bottleneck of Sequential Processing

Before Transformers, NLP relied on Recurrent Neural Networks (RNNs). These models **processed tokens one by one**, carrying a hidden state forward.

1. **Sequential Dependency**: Processing occurs linearly (t1 → t2 → t3), preventing parallelization.
2. **Vanishing Gradients**: The model "forgets" early context in long sequences as gradients diminish.



Replacing Recurrence with Attention



Paradigm Shift: **From Recurrence (Sequential) to Attention (Parallel)**

- Simultaneous processing of all input tokens.
- Eliminates sequential dependency and bottlenecks.
- Enables long-range dependencies over any distance.

The Transformer's Paper (2017 June)

Attention Is All You Need

https://arxiv.org/pdf/1706.03762.pdf

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

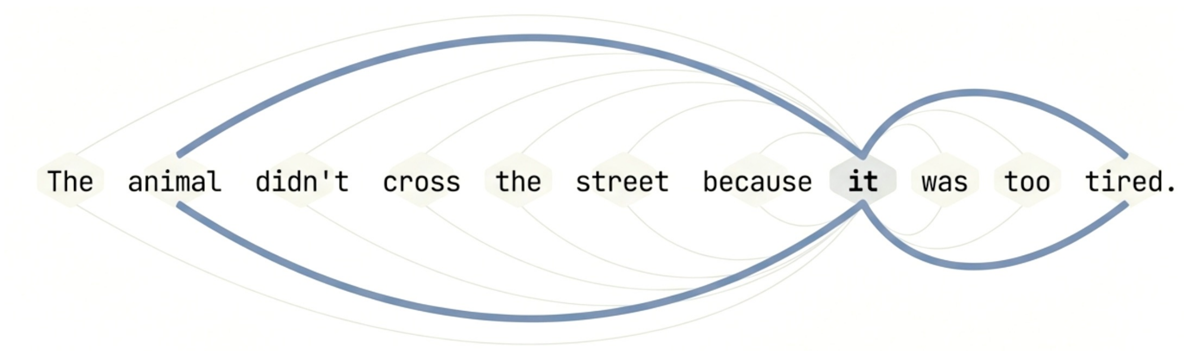
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

The **Transformer** replaces recurrence with attention mechanisms, allowing the model to consider all tokens in a sequence simultaneously.

The Heart of the Transformer: Self-Attention.

The Paradigm Shift: Attention Is All You Need

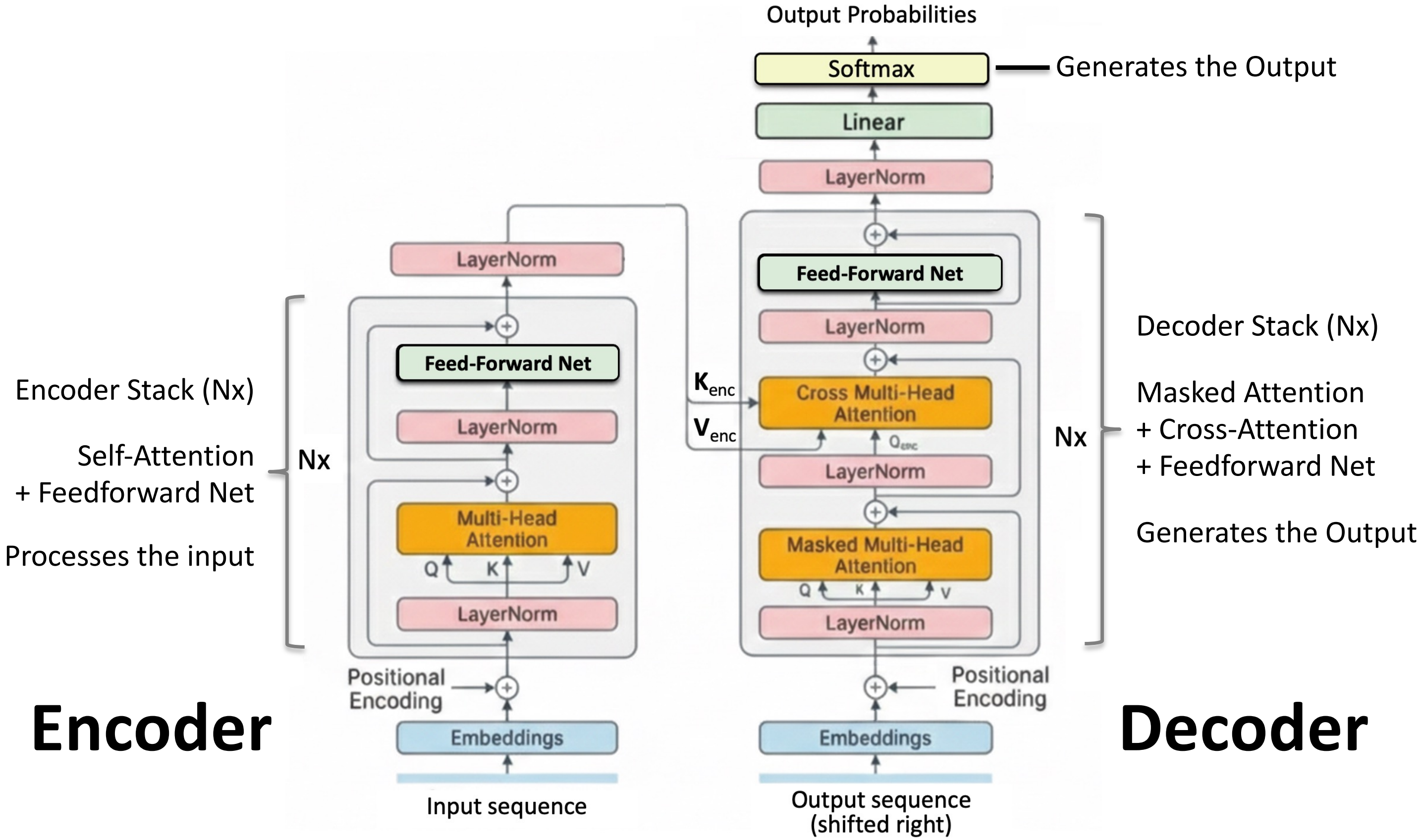


Self-Attention allows the Transformer to associate "it" with "animal" rather than "street"

Advantage 1: Global Dependencies. The model can pay attention to specific words no matter how distant they are.

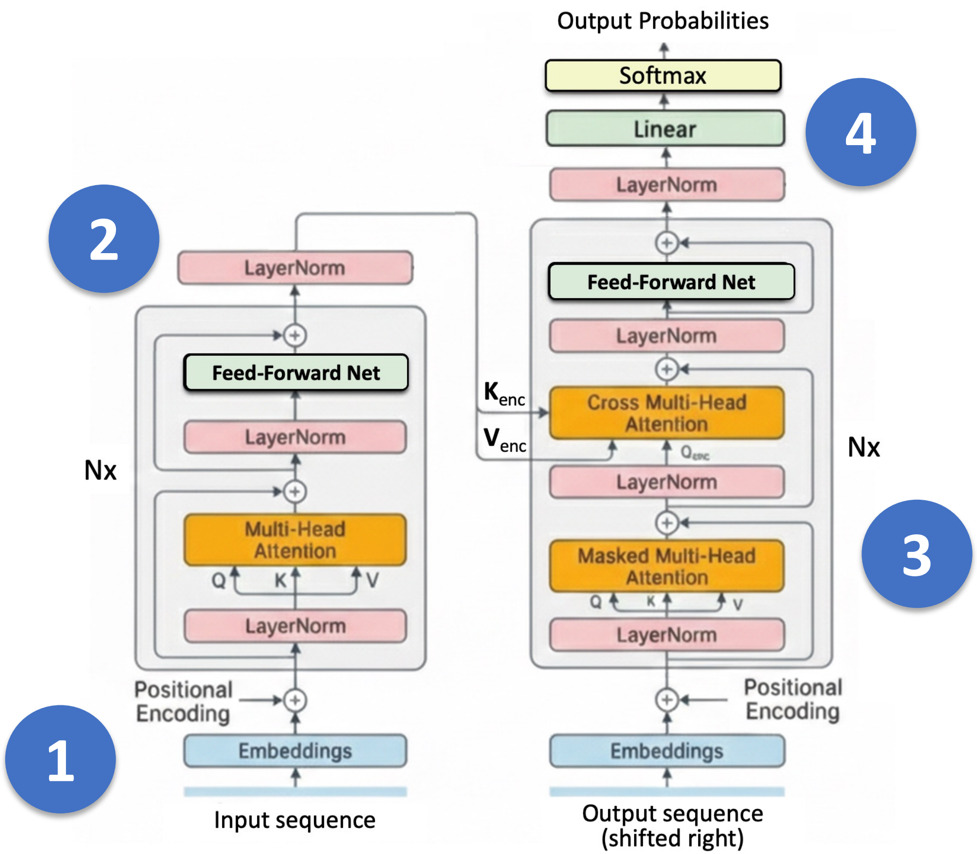
Advantage 2: Parallelization. Massive speed boosts by processing all input tokens

The Transformer Encoder-Decoder Architecture



While the Encoder processes inputs for understanding and the Decoder generates outputs, both rely on an identical foundation: transforming discrete text into mathematical vectors.

The Transformer Workflow



1. Input Embeddings & Positional Encoding

Word embeddings + Sinusoidal positional encoding

2. Encoder Stack (Nx Layers)

Multi-head Attention => FFNN

3. Decoder Stack (Nx Layers)

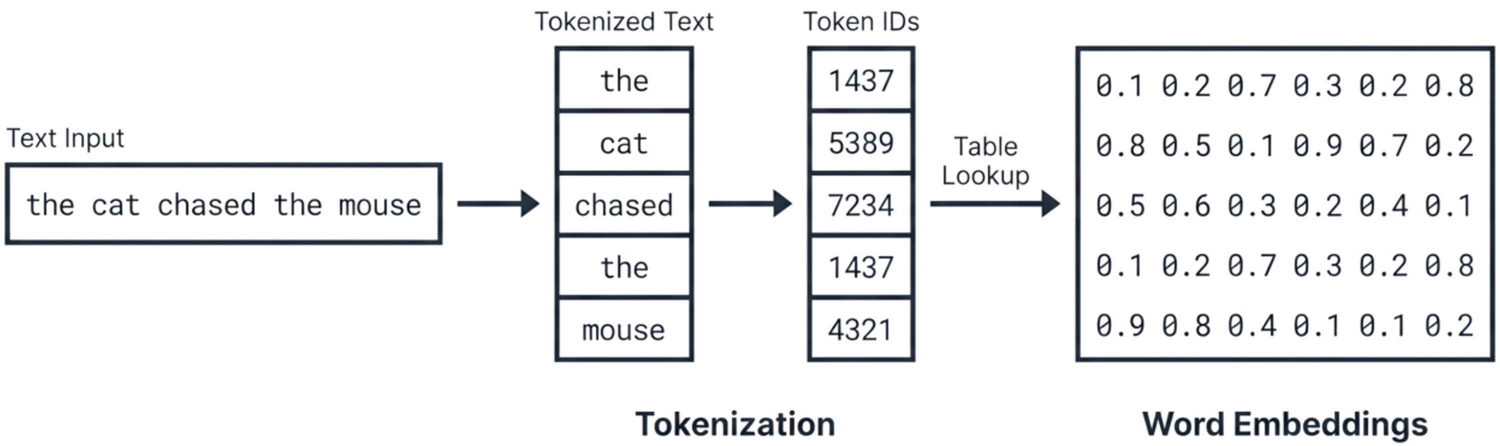
Masked multi-head attention => Cross multi-head attention => FFNN

4. Output Prediction

Linear Layer => Softmax output

Step 1 – Embeddings

From Words to Vectors



$$\mathbf{E} = \begin{bmatrix} \leftarrow & \mathbf{e}_1 & \rightarrow \\ \leftarrow & \mathbf{e}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{e}_n & \rightarrow \end{bmatrix} \in \mathbb{R}^{n \times d}$$

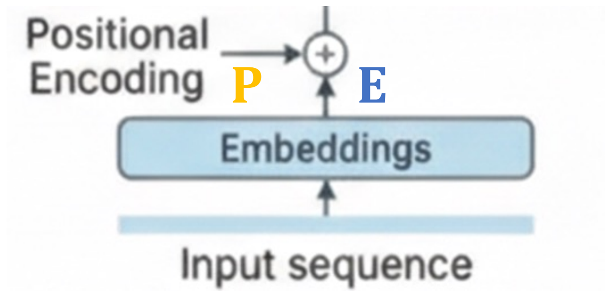
$d = 512$ and $n = 256$

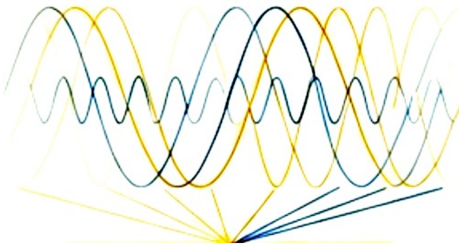
- Each token (subword) is converted to a 512-dimensional embedding vector $\mathbf{e}_\mathbf{j}$.
- Vectors are stacked into an embedding matrix $\mathbf{E}$ of size 256 × 512
  - Context length (max sequence length) $n \times d$ model dimension
- Embeddings capture semantic meaning—e.g., “King” and “Queen” are closer than “King” and “Apple”.

Step 1 - Positional Encoding

Since the Transformer processes all words simultaneously (parallel), it has no inherent sense of order. We add a positional vector-derived from sine and cosine functions-to give every word a specific "coordinate" in the sentence.

$$\mathbf{X} = \mathbf{E} + \mathbf{P}$$



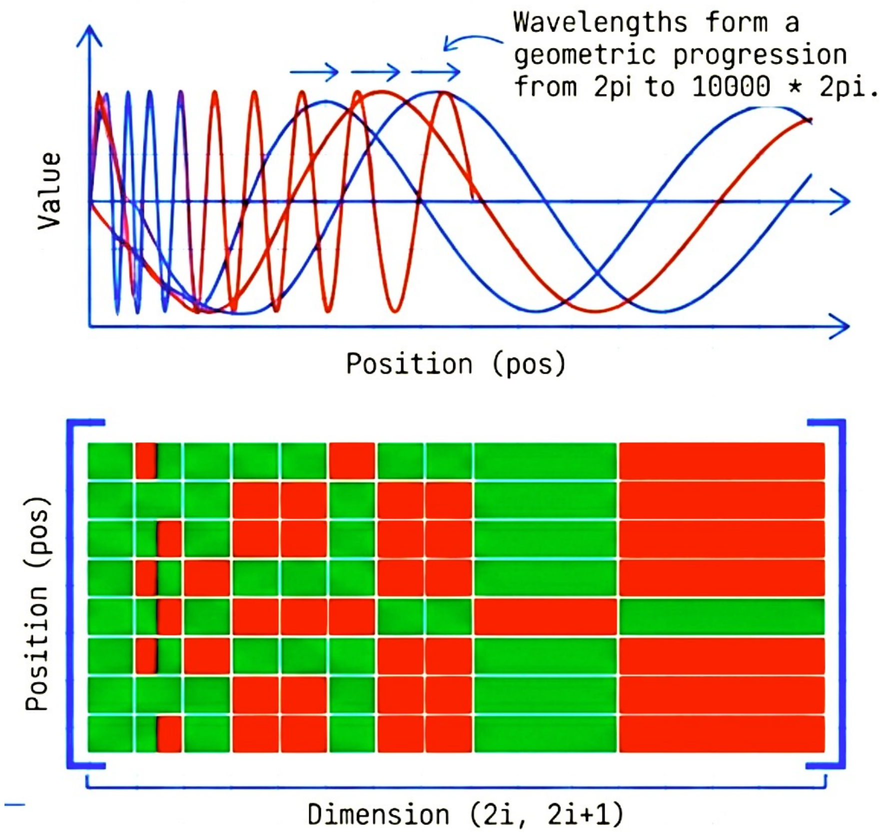


Sine and Cosine functions inject unique position data into the embeddings.

Embedding Matrix **E** + Positional Encoding **P** = Input Matrix **X**

$$\mathbf{E} = \begin{bmatrix} \leftarrow \mathbf{e}_1 \rightarrow \\ \leftarrow \mathbf{e}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{e}_n \rightarrow \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} \leftarrow \mathbf{p}_0 \rightarrow \\ \leftarrow \mathbf{p}_1 \rightarrow \\ \vdots \\ \leftarrow \mathbf{p}_{n-1} \rightarrow \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \leftarrow \mathbf{x}_1 \rightarrow \\ \leftarrow \mathbf{x}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{x}_n \rightarrow \end{bmatrix}$$

Sinusoidal Positional Encoding



The Sinusoidal Positional Encoding uses different frequencies to generate a unique vector for each position (pos) in the sequence.

Formulas: The encoding uses **sine** for even indices ($2i$) and **cosine** for odd indices ($2i + 1$) of the vector dimensions:

$$\mathrm{PE}(\mathrm{pos}, 2i) = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$$

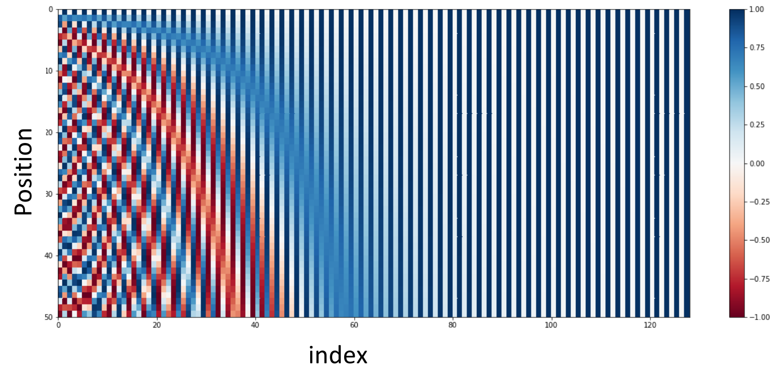
$$\mathrm{PE}(\mathrm{pos}, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Example

The frequency of the functions determines the rate at which positions differ, providing unique representations for each position.

Even number indices of vector component $\mathrm{PE}(\mathrm{pos}, 2i) = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$

Odd number indices of vector component $\mathrm{PE}(\mathrm{pos}, 2i+1) = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right)$



Positional Encoding Matrix $\mathbf{P}$ *with* $d = 4$

Text Sequence	*pos* of token	$i = 0$	$i = 0$	$i = 1$	$i = 1$	
I	0	$PE(0,0) = \sin(0/1) = 0$	$PE(0,1) = \cos(0/1) = 1$	$PE(0,2) = \sin(0/100) = 0$	$PE(0,3) = \cos(0/100) = 1$	$\mathbf{p}_0$
am	1	$PE(1,0) = \sin(1/1) = 0.84$	$PE(1,1) = \cos(1/1) = 0.54$	$PE(1,2) = \sin(1/100) = 0.01$	$PE(1,3) = \cos(1/100) = 0.9999500$	$\mathbf{p}_1$
a	2	$PE(2,0) = \sin(2/1) = 0.91$	$PE(2,1) = \cos(2/1) = -0.43$	$PE(2,2) = \sin(2/100) = 0.02$	$PE(2,3) = \cos(2/100) = 0.9999875$	$\mathbf{p}_2$
Robot	3	$PE(3,0) = \sin(3/1) = 0.14$	$PE(3,1) = \cos(3/1) = -0.99$	$PE(3,2) = \sin(3/100) = 0.03$	$PE(3,3) = \cos(3/100) = 0.9999944$	$\mathbf{p}_3$
	Index:	0	1	2	3	

For even index 2 with $\mathrm{pos} = 1$ and $i = 1$

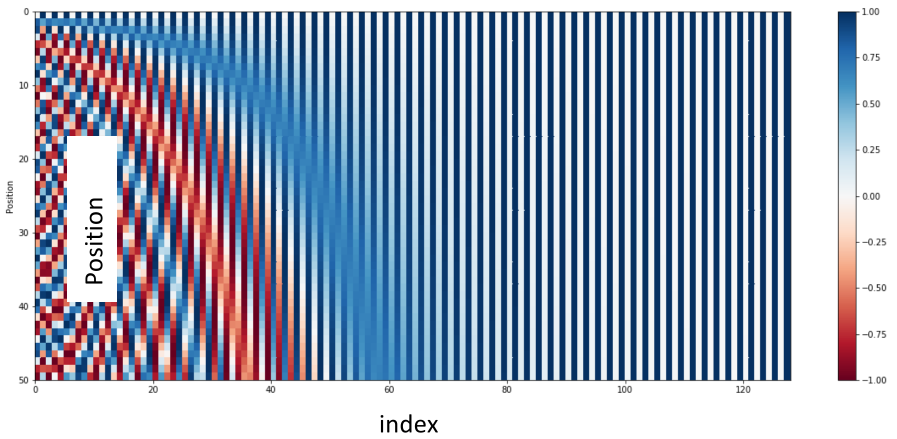
$$\mathrm{PE}(1,2) = \sin\left(\frac{1}{10000^{\frac{2\times 1}{4}}}\right) = \sin\left(\frac{1}{100}\right) = 0.01$$

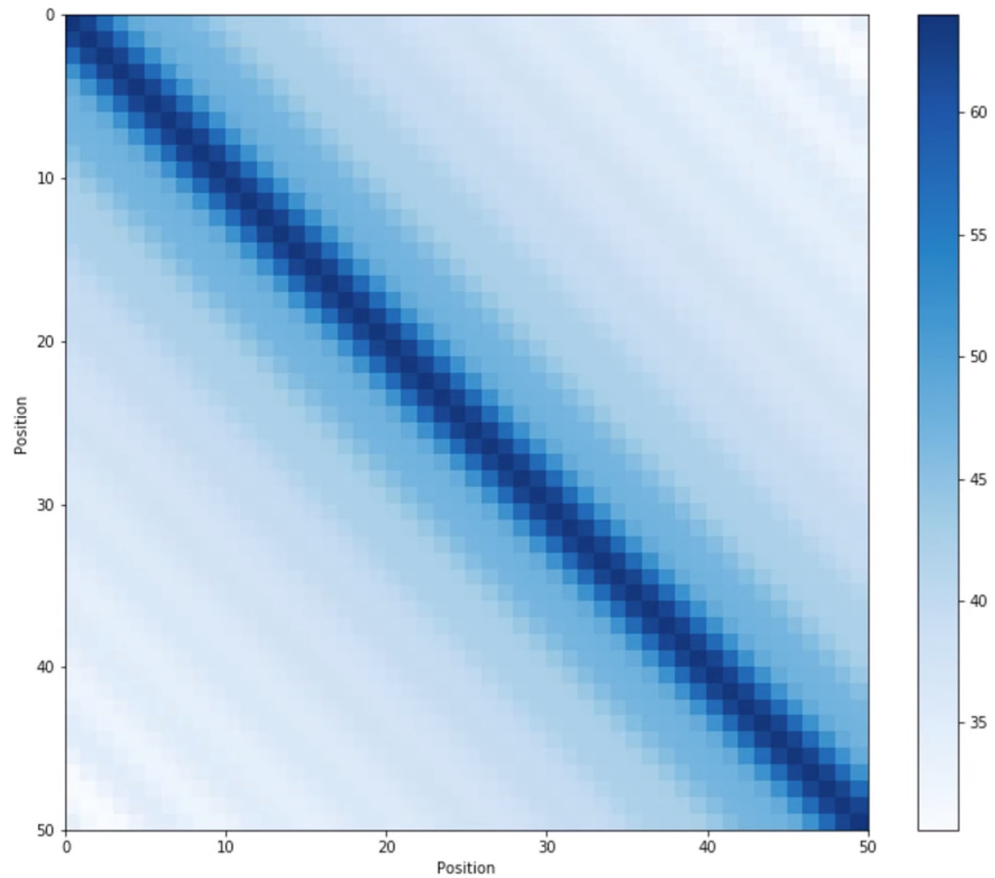
For odd index 3 with $\mathrm{pos} = 1$ and $i = 1$

$$\mathrm{PE}(1,3) = \cos\left(\frac{1}{10000^{\frac{2\times 1}{4}}}\right) = \cos\left(\frac{1}{100}\right) = 0.9999500$$

Sinusoidal Positional Encoding

- Distance between neighboring positions
  - **Symmetrical**
  - **Decay nicely with time**

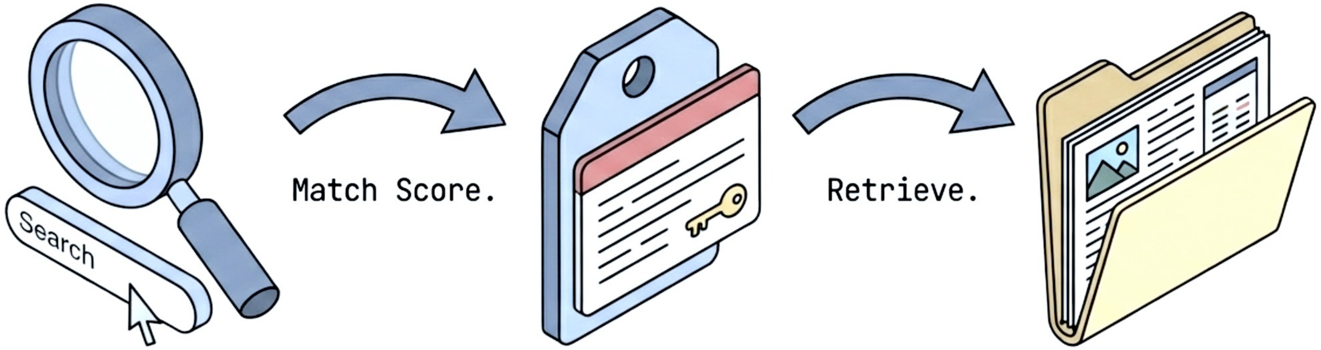




Dot product of position embeddings for all time-steps

Step 2 - Multi-Head Self-Attention

The Database Analogy with the Concept of Relationship Mapping



Query (**Q**):
What I am looking for.

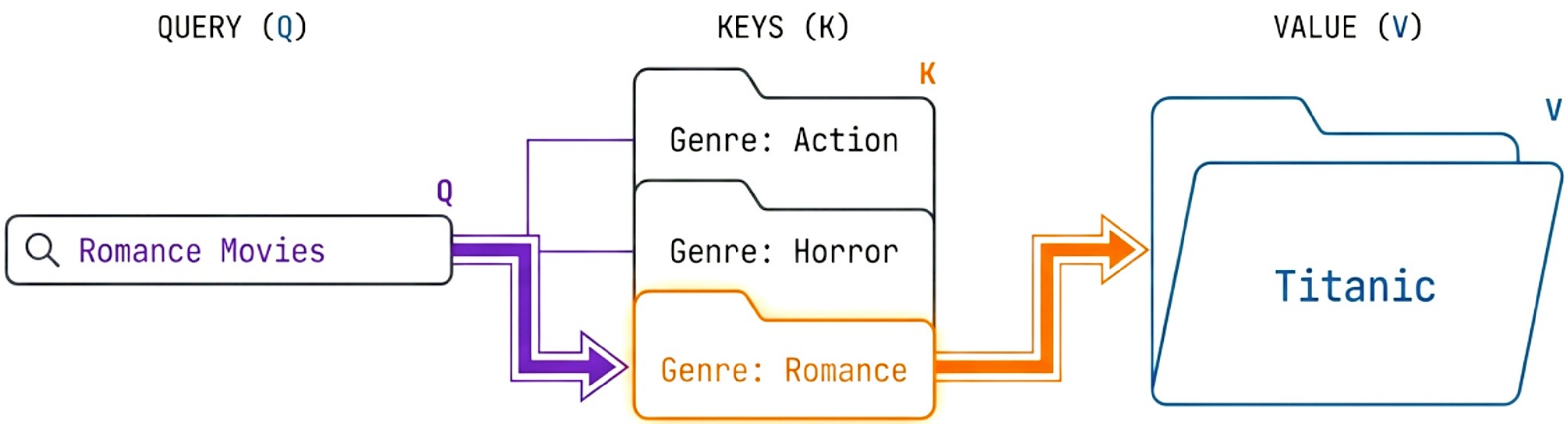
Key (**K**):
How the data is indexed.

Value (**V**):
The actual content.

The model creates **Q**, **K**, and **V** vectors for every word. It compares the Query to all Keys. If they match (high score), it extracts the Value.

Calculating Relevance: Movie Database

An infographic illustrating the "Database Query" analogy using the movie example from the narrative.



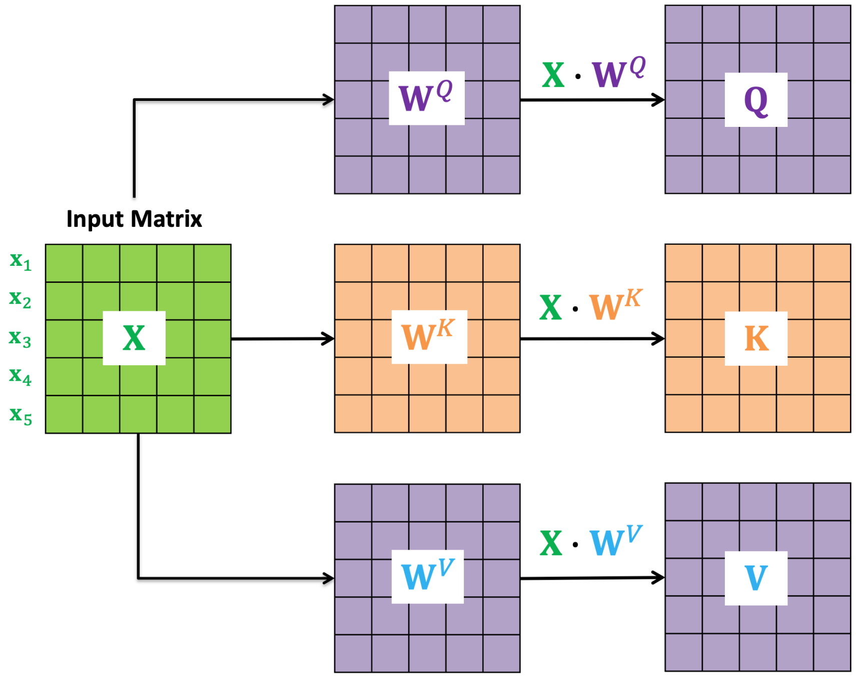
Like searching a database, the Query (Romance Movies) matches a specific Key (Genre: Romance) to retrieve the associated Value (Titanic).

Linear Projections: Q, K, and V

For every token, we generate three new vectors by multiplying the input X by learned weights:

- **Query:** $\mathbf{Q} = \mathbf{X}\,\mathbf{W}^{Q} \in \mathbb{R}^{n \times d_k}$
- **Key:** $\mathbf{K} = \mathbf{X}\,\mathbf{W}^{K} \in \mathbb{R}^{n \times d_k}$
- **Value:** $\mathbf{V} = \mathbf{X}\,\mathbf{W}^{V} \in \mathbb{R}^{n \times d_v}$

These projections transform the general embedding into specialized vectors for retrieval process.



Matrix Representation of X, Q, K, V

Input Matrix

$$\mathbf{X} = \begin{bmatrix} \leftarrow \mathbf{x}_1 \rightarrow \\ \leftarrow \mathbf{x}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{x}_n \rightarrow \end{bmatrix} \in \mathbb{R}^{n \times d}$$

Query Matrix

$$\mathbf{Q} = \begin{bmatrix} \leftarrow \mathbf{q}_1 \rightarrow \\ \leftarrow \mathbf{q}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{q}_n \rightarrow \end{bmatrix} = \mathbf{X}\,\mathbf{W}^{Q} \in \mathbb{R}^{n \times d_k}$$

Key Matrix

$$\mathbf{K} = \begin{bmatrix} \leftarrow \mathbf{k}_1 \rightarrow \\ \leftarrow \mathbf{k}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{k}_n \rightarrow \end{bmatrix} = \mathbf{X}\,\mathbf{W}^{K} \in \mathbb{R}^{n \times d_k}$$

Value Matrix

$$\mathbf{V} = \begin{bmatrix} \leftarrow \mathbf{v}_1 \rightarrow \\ \leftarrow \mathbf{v}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{v}_n \rightarrow \end{bmatrix} = \mathbf{X}\,\mathbf{W}^{V} \in \mathbb{R}^{n \times d_v}$$

where n is the length of the input sequence, d is the dimension of the input embeddings, d_k is the dimension of the $\mathbf{k}_i$ key vectors and is d_v the dimension of the $\mathbf{v}_i$ value vectors.

Scaled Dot-Product Attention

The mechanism that allows the model to 'focus'.

Calculate Similarity
(Dot Product)

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^{T}}{\sqrt{d_k}}\right)\mathbf{V}$$

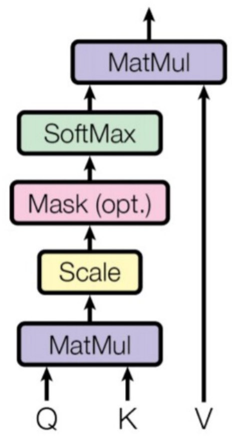
Software normalize
to probabilities (0-1)

Scale to prevent
exploding gradients

Weight the Values by
the probability scores

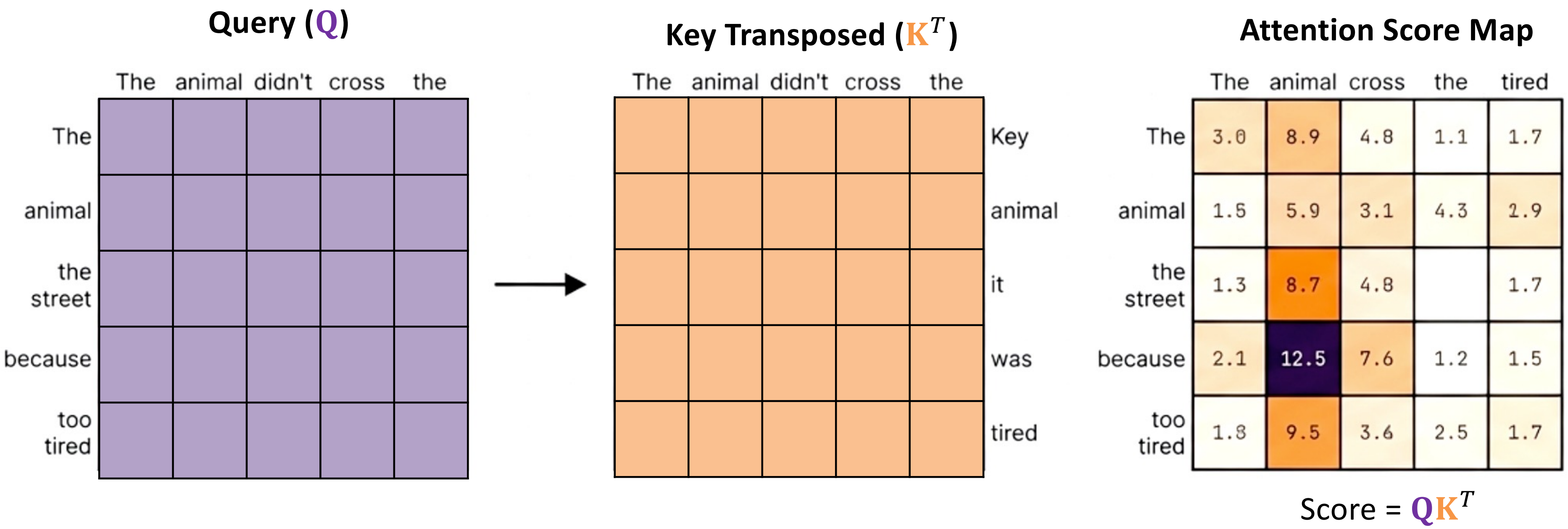
Dividing by the square root of the dimension prevents gradients from exploding in high-dimensional spaces.

Think of a File Retrieval System:
- **Query**: What you are looking for. - **Key**: The label on the file folder. - **Value**: The content inside the folder. The model compares your Query to the Keys to decide how much of the Value to retrieve.



The Dot Product: Calculating Relevance

Matrix Multiplication



High Dot Product = High Similarity. The model calculates how much focus 'It' should place on 'Animal".

Scaled Dot-Product Scores Matrix

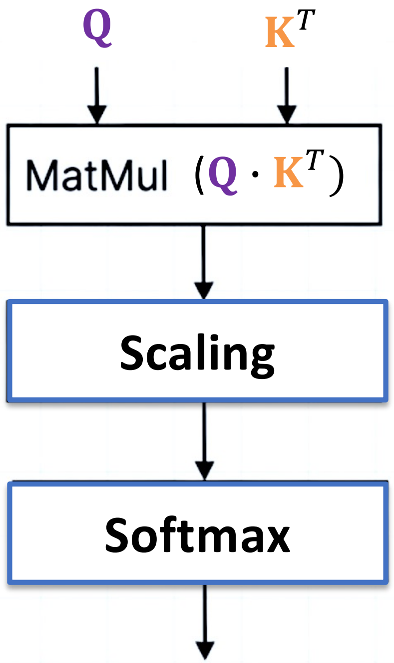
Attention scores are computed as scaled dot products:

$$\text{Attention Scores} = \frac{1}{\sqrt{d_k}} \begin{bmatrix} \mathbf{q}_1\mathbf{k}_1^T & \mathbf{q}_1\mathbf{k}_2^T & \cdots & \mathbf{q}_1\mathbf{k}_n^T \\ \mathbf{q}_2\mathbf{k}_1^T & \mathbf{q}_2\mathbf{k}_2^T & \cdots & \mathbf{q}_2\mathbf{k}_n^T \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{q}_n\mathbf{k}_1^T & \mathbf{q}_n\mathbf{k}_2^T & \cdots & \mathbf{q}_n\mathbf{k}_n^T \end{bmatrix} = \frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}$$

where each entry $\mathbf{q}_i\mathbf{k}_j^T$ measures similarity between query $\mathbf{q}_i$ and key $\mathbf{k}_j$.

- The scaling factor $\frac{1}{\sqrt{d_k}}$ prevents large dot products in high dimensions, avoiding vanishing gradients during softmax—ensuring stable, efficient training.

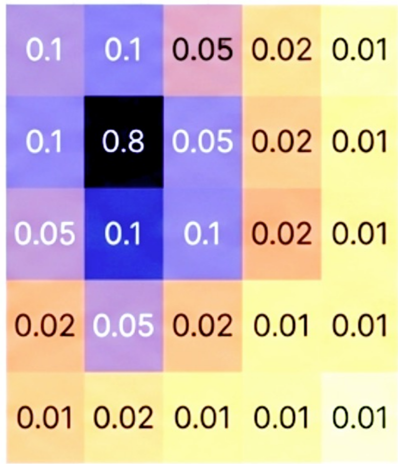
Scaling & Softmax Normalization



Scale by $\sqrt{d_k}$ to prevent exploding gradients (e.g. $\sqrt{64} = 8$)

Normalize to probabilities (0% to 100%).

Heatmap

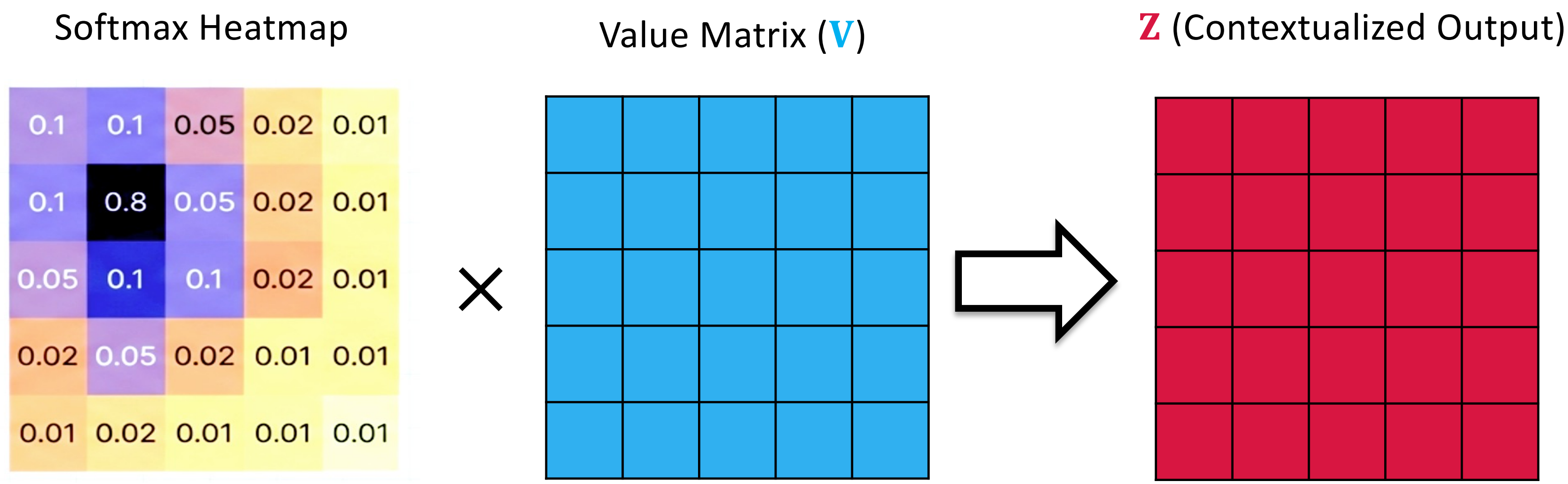


$$\text{Softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^T}{\sqrt{d_k}}\right)$$

Scaling: Stabilizes the math for high-dimensional vectors.

Softmax: Converts raw scores into probabilities (0 to 1)

The Weighted Sum

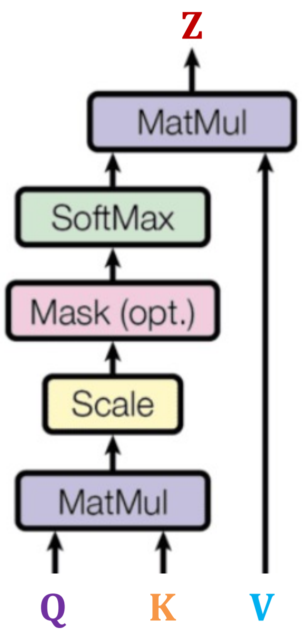


$$\mathrm{Softmax}\left(\frac{\mathbf{Q}\times\mathbf{K}^T}{\sqrt{d_k}}\right)$$

$$\mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathbf{Z} = \mathrm{Softmax}\left(\frac{\mathbf{Q}\times\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

Filtering: We keep the information (Values) of **relevant words** and drown out the noise.

Matrix Form: Scaled Dot-Product Attention



Query Matrix

$$\mathbf{Q} = \begin{bmatrix} \leftarrow & \mathbf{q}_1 & \rightarrow \\ \leftarrow & \mathbf{q}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{q}_n & \rightarrow \end{bmatrix} = \mathbf{X}\,\mathbf{W}^Q$$

Key Matrix

$$\mathbf{K} = \begin{bmatrix} \leftarrow & \mathbf{k}_1 & \rightarrow \\ \leftarrow & \mathbf{k}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{k}_n & \rightarrow \end{bmatrix} = \mathbf{X}\,\mathbf{W}^K$$

Value Matrix

$$\mathbf{V} = \begin{bmatrix} \leftarrow & \mathbf{v}_1 & \rightarrow \\ \leftarrow & \mathbf{v}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{v}_n & \rightarrow \end{bmatrix} = \mathbf{X}\,\mathbf{W}^V$$

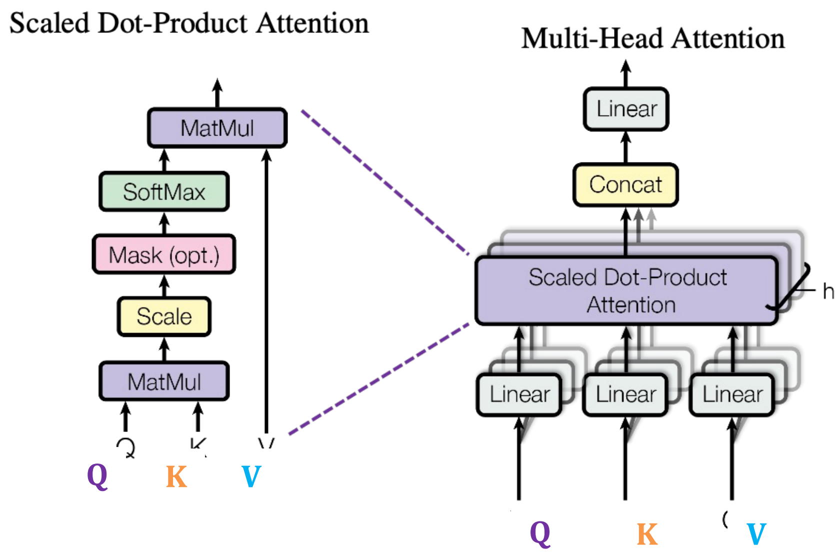
Input Matrix

$$\mathbf{X} = \begin{bmatrix} \leftarrow & \mathbf{x}_1 & \rightarrow \\ \leftarrow & \mathbf{x}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{x}_n & \rightarrow \end{bmatrix}$$

$$\mathbf{Z} = \begin{bmatrix} \leftarrow & \mathbf{z}_1 & \rightarrow \\ \leftarrow & \mathbf{z}_2 & \rightarrow \\ & \vdots & \\ \leftarrow & \mathbf{z}_n & \rightarrow \end{bmatrix} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

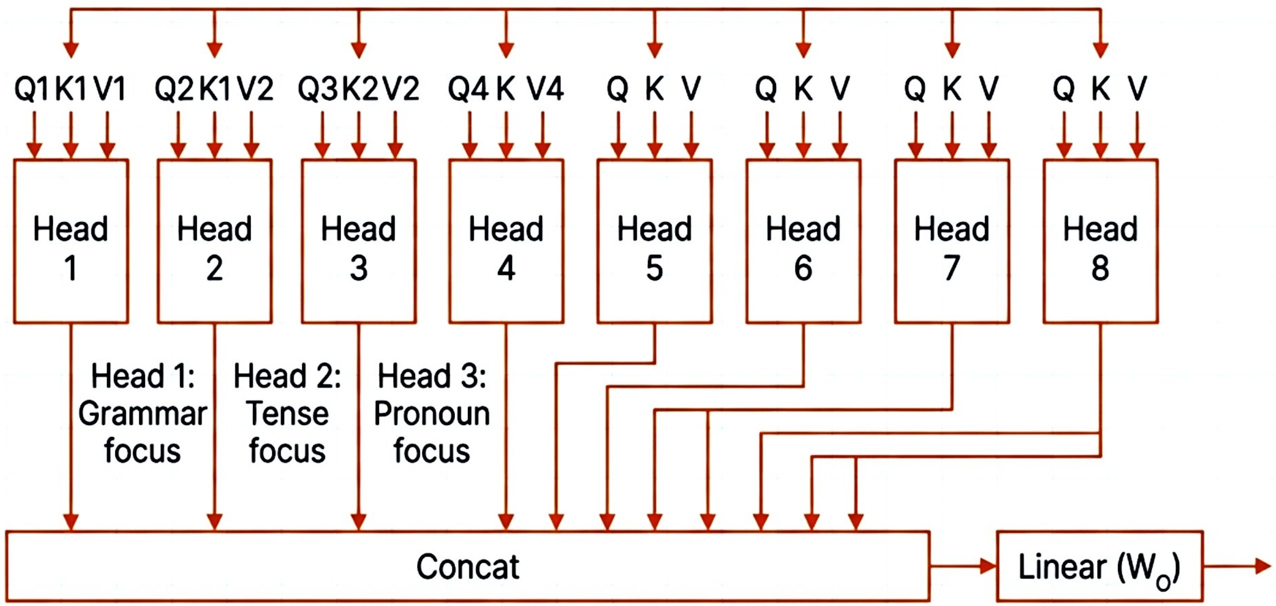
Multi-Head Attention

- Multi-Head Attention is an extension of self-attention, allowing the model to focus on different parts of the input sequence simultaneously.
- This is achieved by applying several attention mechanisms in parallel.

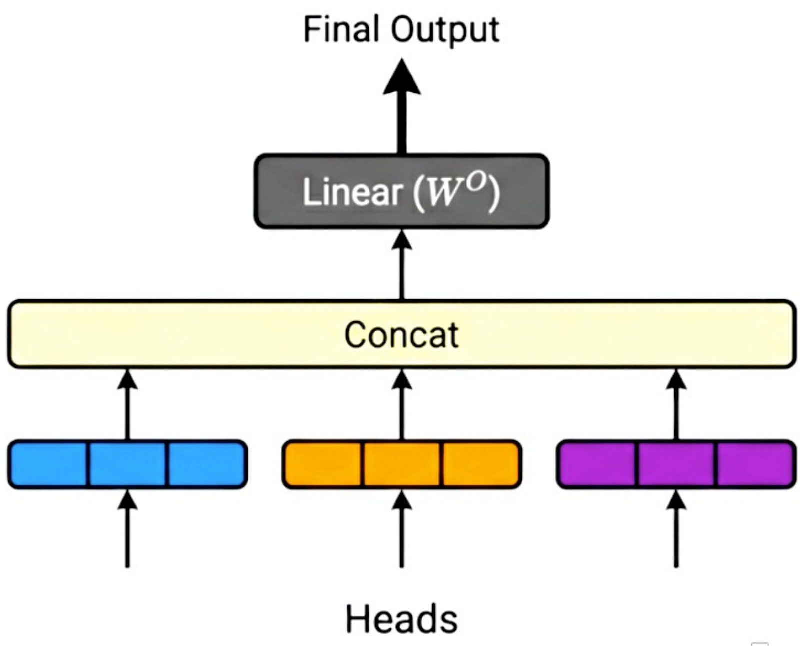


Multi-Head Attention

Why one focus isn't enough. The model learns multiple types of relationships simultaneously.



Unifying the Heads



Concatenation: The insights from all heads are linked end to end.

Projection: A final weight matrix ($\mathbf{W}^O$) compresses this multi-perspective information back into a single stream of size d (or d_{model})

$$\mathbf{O} = \text{Multihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)\mathbf{W}^O$$

Understanding Multi-Head Attention

1. Linear Projections: The input is projected into multiple sets of queries, keys, and values, each corresponding to a different attention head. For $i = 1,2, \dots, h$

$$\mathbf{Q}_i = \mathbf{X}\,\mathbf{W}_i^Q \quad \mathbf{K}_i = \mathbf{X}\,\mathbf{W}_i^K \quad \mathbf{V}_i = \mathbf{X}\,\mathbf{W}_i^V$$

where $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$, $\mathbf{W}_i^V$ are learned weight matrices for the i–th head.

2. Scaled Dot-Product Attention: Each set undergoes the scaled dot-product attention mechanism independently.

$$\text{head}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = \text{Softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^T}{\sqrt{d_k}}\right)\mathbf{V}_i$$

3. Concatenation: The outputs of all attention heads are combined through concatenation.

$$\text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)$$

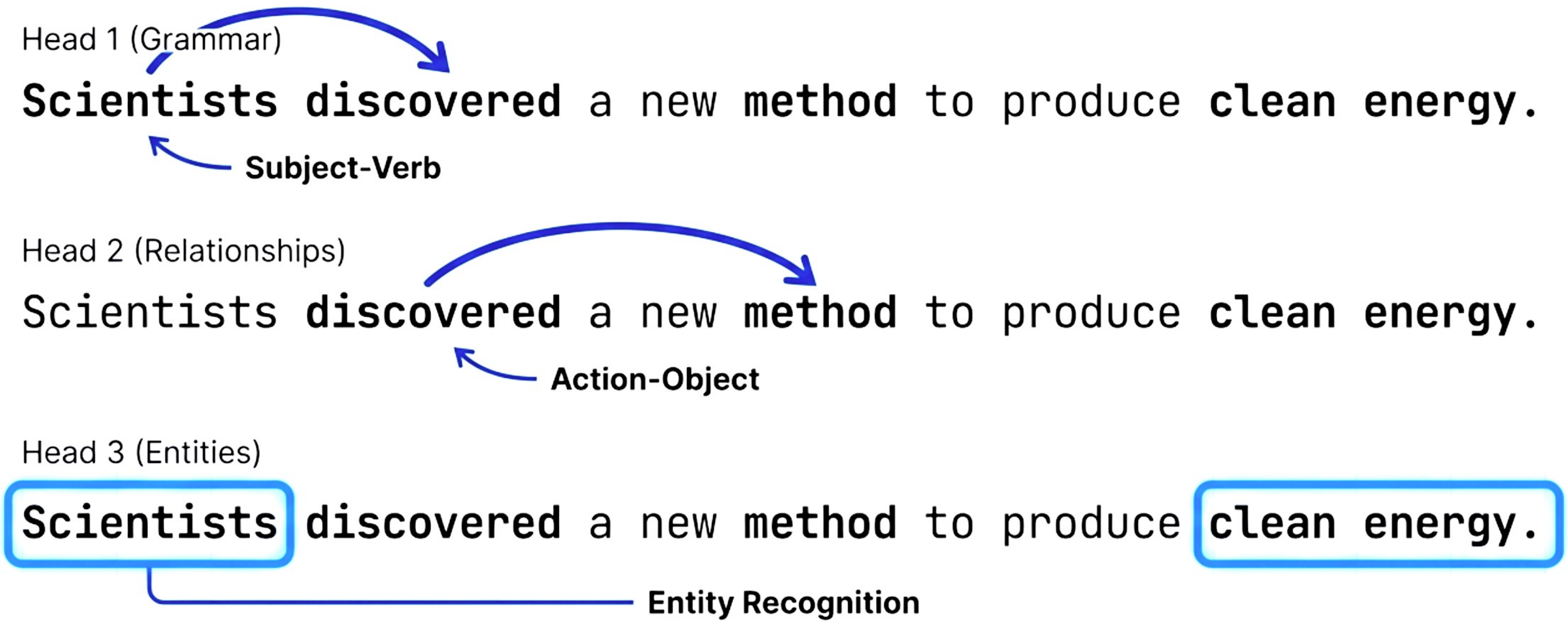
4. Final Linear Projection: The concatenated output is projected using a learned weight matrix to produce the final output.

$$\mathbf{O} = \text{Multihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)\mathbf{W}^O$$

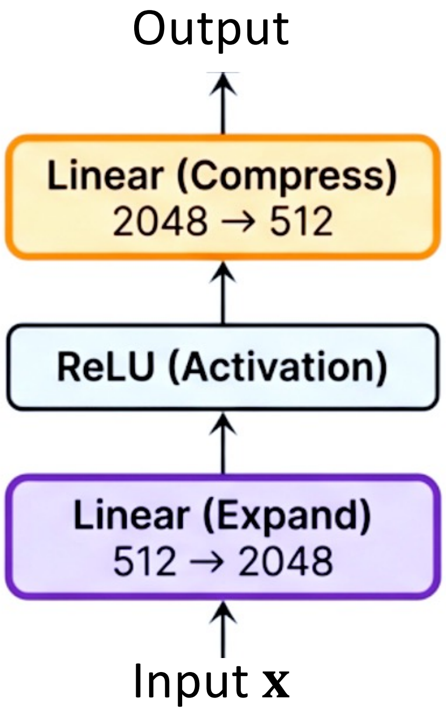
where $\mathbf{W}^O$ is the weight matrix for the final linear projection.

Why Multiple Heads?

The model builds a composite understanding of syntax, semantics, and entities all at once.



The Feed-Forward Network



After Attention, the position-wise **Feed-Forward Network (FFN)** processes each toke individually

Expand-Compress Architecture: Projects data into a higher dimension to extract features, then compresses it back.

Formula:

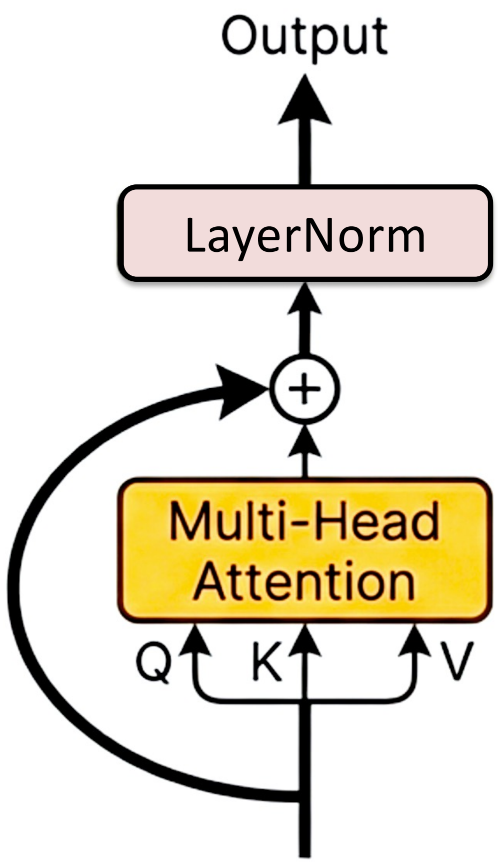
$$\mathrm{FFN}(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

Applied identically to every position. This is where the model "processes" the information gathered by attention.

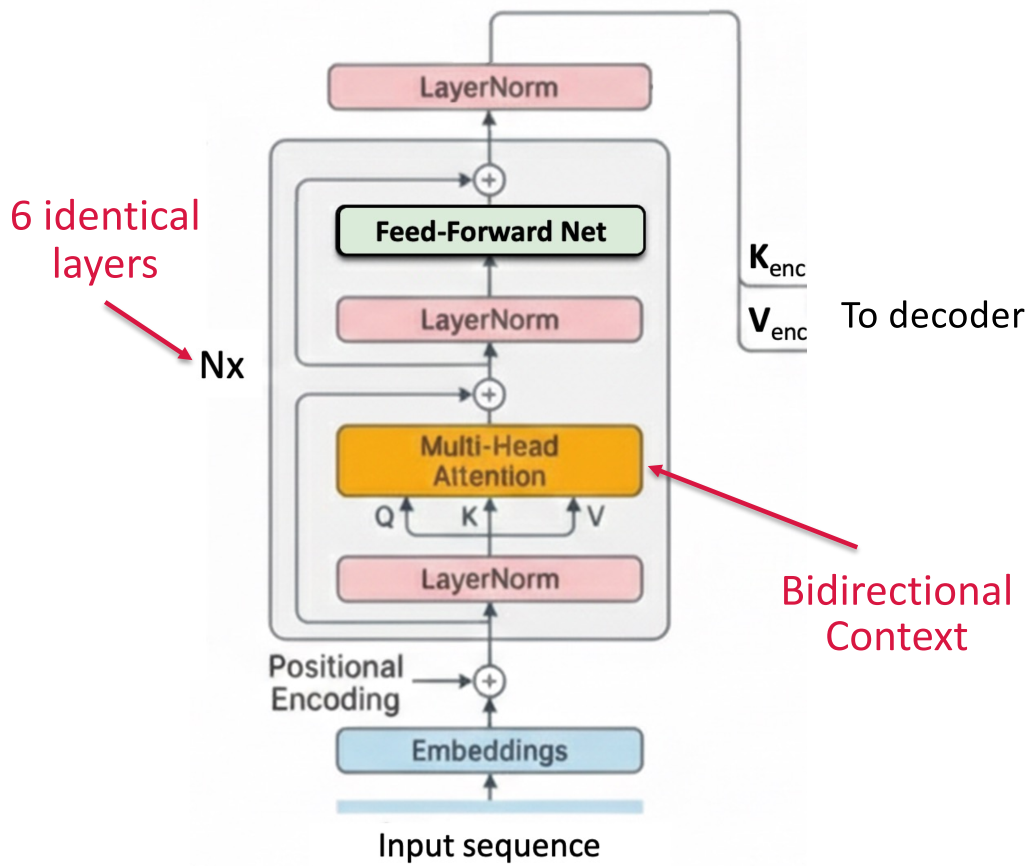
Stability & Flow: Add and Norm

Deep networks struggle to train due to vanishing gradients.

- **Residual connection (add):**
  - $\mathbf{x} + \mathrm{Sublayer}(\mathbf{x})$
  - Crucial for deep networks. Allows gradients to flow through the network without vanishing.
- **LayerNorm:**
  - Normalizes the output to stabilize learning time.



The Encoder Stack

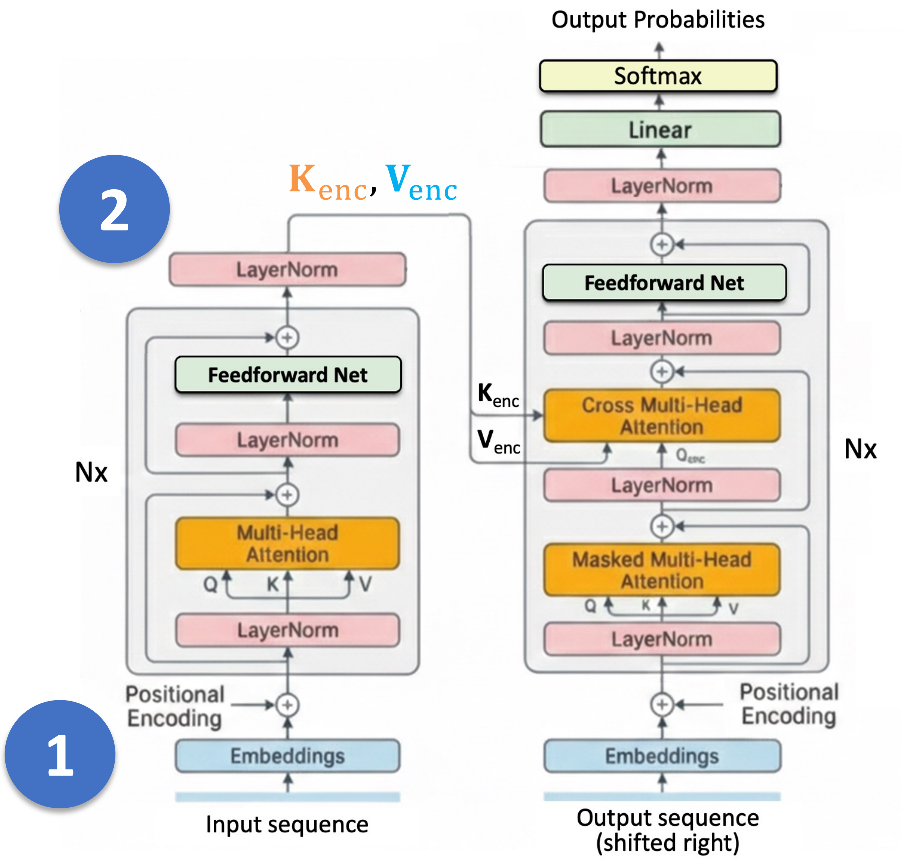


- **Role:** Understanding & Contextualization.
- **Characteristics:** Unmasked, Bidirectional (sees past and future tokens).
- **Output**: $\mathbf{K}_{\mathrm{enc}}$ and $\mathbf{V}_{\mathrm{enc}}$ matrices passed to the Decode

The Result of the Encoder

The Encoder produces a rich, **context-aware** matrix K_{enc}, V_{enc} containing the 'meaning' of the sentence.

This is passed to the Decoder to guide generation.



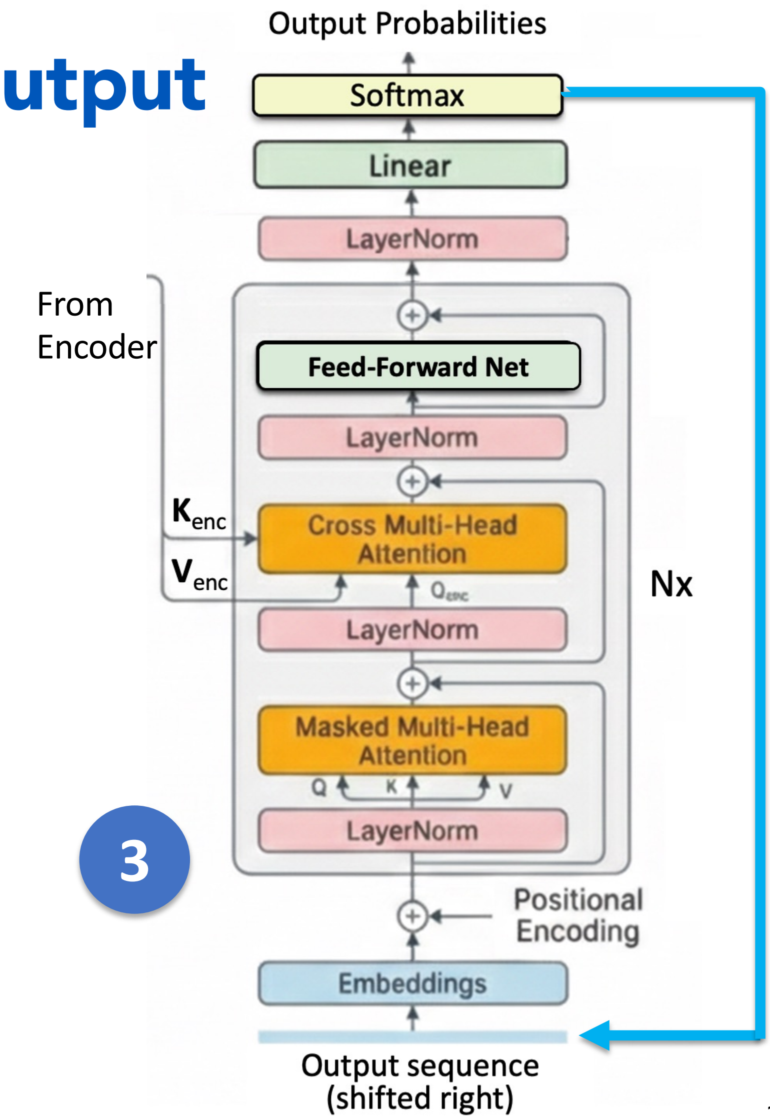
The Decoder: Generating the Output

The Decoder is **Autoregressive**.

1. Generates output one token at a time.
2. **Uses previously generated output as input for the next step.**
3. Stops when it predicts an <EOS> (End of Sentence) token.

Key Differences:

1. **Masked Attention:** Ensures the model predicts based only on past history.
2. **Cross-Attention:** The interface where the Decoder reads the Encoder's memory,



Decoder: Masked Self-Attention

You cannot see the future.

Within the Decoder, Masked Self-Attention is used with future tokens are masked out.

- When predicting the word at position 3, the model is strictly forbidden from 'seeing' positions 4 and 5.

Token Position (Output Sequence)

	1	2	3	4	5
1	0.9	$-\infty$	$-\infty$	$-\infty$	$-\infty$
2	0.1	0.8	$-\infty$	$-\infty$	$-\infty$
3	0.2	0.3	0.5	$-\infty$	$-\infty$
4	0.0	0.1	0.2	0.7	$-\infty$
5	0.0	0.0	0.1	0.2	0.7

Token Position (Attention Query)

During training, the model cannot "see" future tokens. Masking sets future positions to negative infinity before Softmax, ensuring they have zero attention weight.

Mask Matrix $\mathbf{M}$

The mask matrix $\mathbf{M}$ is used to prevent attention to future tokens. It contains values of:

- 0 for tokens that can be attended to
- $-\infty$ for future tokens

$$\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}$$

0.268	~~0.119~~	~~0.134~~	~~0.148~~	~~0.179~~
0.124	0.278	~~0.201~~	~~0.128~~	~~0.154~~
0.147	0.132	0.262	~~0.097~~	~~0.218~~
0.210	0.128	0.206	0.212	~~0.119~~
0.146	0.158	0.152	0.143	0.227

Attention Score Map

$+$

$$\mathbf{M}$$

0	$-\infty$	$-\infty$	$-\infty$	$-\infty$
0	0	$-\infty$	$-\infty$	$-\infty$
0	0	0	$-\infty$	$-\infty$
0	0	0	0	$-\infty$
0	0	0	0	0

Mask Matrix

$=$

$$\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}$$

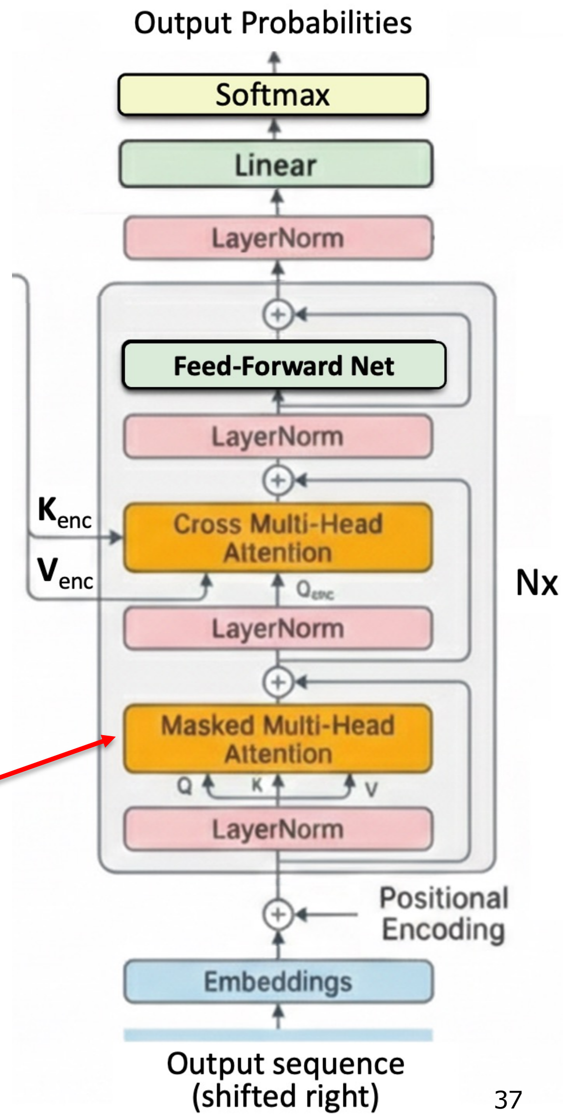
0.268	$-\infty$	$-\infty$	$-\infty$	$-\infty$
0.124	0.278	$-\infty$	$-\infty$	$-\infty$
0.147	0.132	0.262	$-\infty$	$-\infty$
0.210	0.128	0.206	0.212	$-\infty$
0.146	0.158	0.152	0.143	0.227

Marked Self-Attention

$$\text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right) =$$

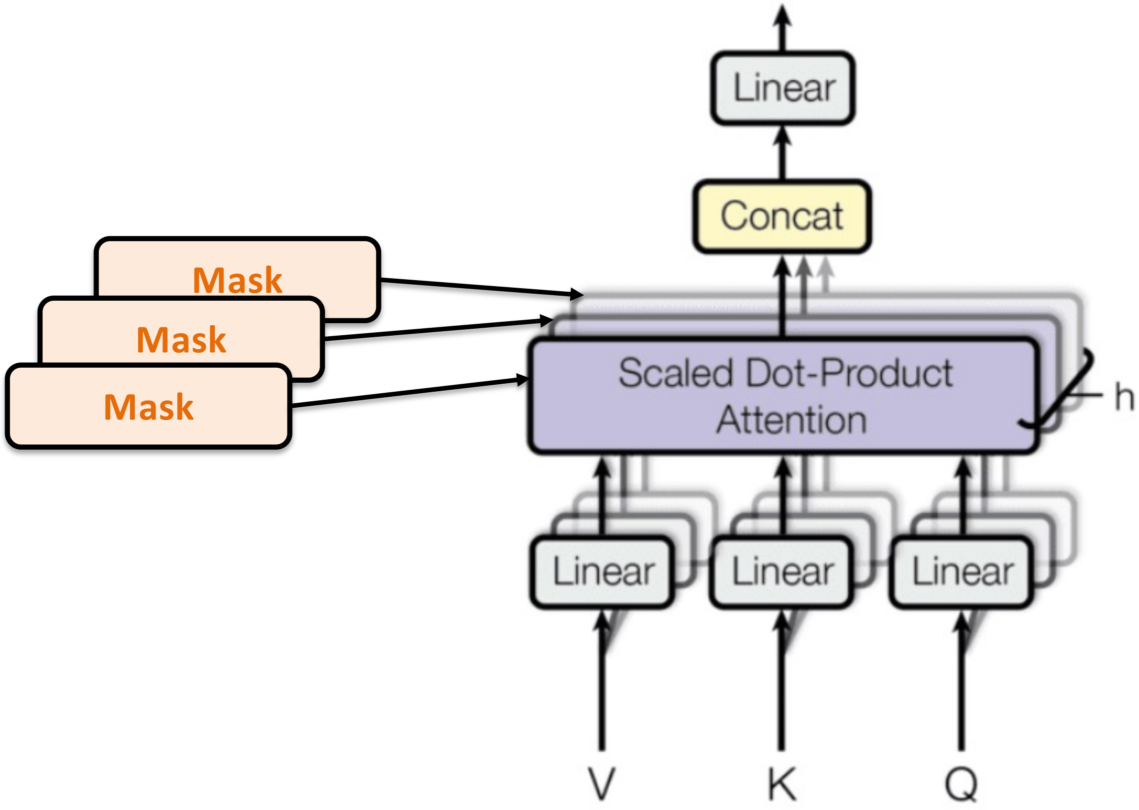
0.9	$-\infty$	$-\infty$	$-\infty$	$-\infty$
0.1	0.8	$-\infty$	$-\infty$	$-\infty$
0.2	0.3	0.5	$-\infty$	$-\infty$
0.0	0.1	0.2	0.7	$-\infty$
0.0	0.0	0.1	0.2	0.7

$$\text{MaskedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}$$



Masked Multi-Head Attention

- Masked Multi-Head Attention is a type of Masked Self-Attention that uses multiple heads.
- It ensures that during training, the model does not "cheat" by looking at future tokens while predicting the next token in a sequence.



Masked Multi-Head Attention

Masked Multi-Head Attention is a type of Masked Self-Attention that uses multiple heads. This mechanism can be described by the following equations:

$$\text{MaskedMultihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)\mathbf{W}^O$$

where

$$\text{head}_i = \text{MaskedAttention}(\mathbf{QW}_i^{\boldsymbol{Q}}, \mathbf{KW}_i^{\boldsymbol{K}}, \mathbf{VW}_i^{\boldsymbol{V}})$$

Here:

- $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the query, key, and value matrices, respectively.
- $\mathbf{QW}_i^{\boldsymbol{Q}}$, $\mathbf{KW}_i^{\boldsymbol{K}}$, and $\mathbf{VW}_i^{\boldsymbol{V}}$ are the learnable projection matrices for each attention head.
- $\mathbf{W}^O$ is the output projection matrix.
- h is the number of attention heads.

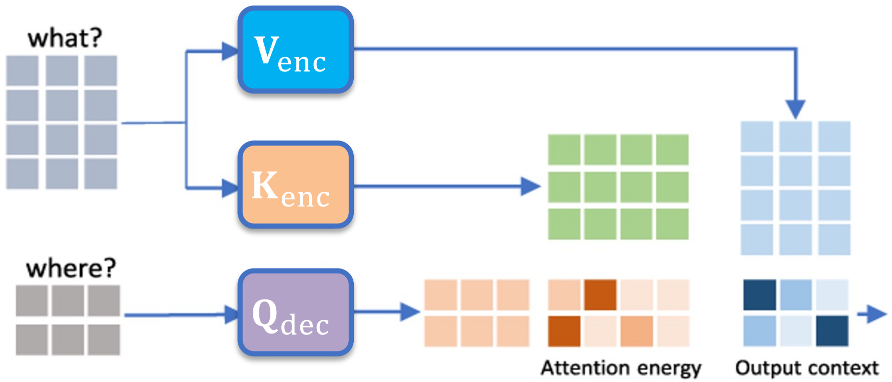
By using multiple heads, the model can attend to different parts of the input sequence simultaneously, capturing complex dependencies between tokens while ensuring that each token only attends to past tokens.

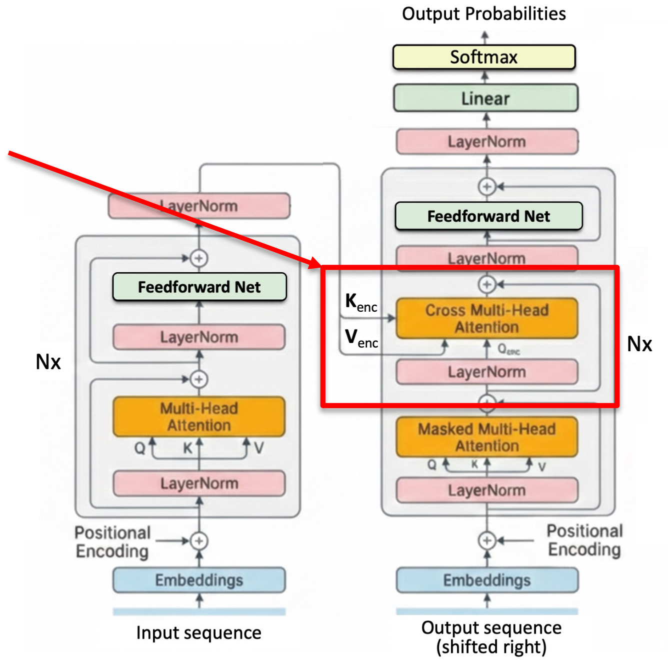
Cross-Attention: The Handshake

Cross-Attention Layer:

- This is the bridge between understanding and generating.
- The Decoder Query ($\mathbf{Q}_{\mathrm{dec}}$) 'looks back' at the Encoder's output Keys ($\mathbf{K}_{\mathrm{enc}}$) and Values ($\mathbf{V}_{\mathrm{enc}}$) to decide which part of the source sentence is relevant for the next translated word.

$$\mathrm{CrossAttention}(\mathbf{Q}_{\mathrm{dec}}, \mathbf{K}_{\mathrm{enc}}, \mathbf{V}_{\mathrm{enc}}) = \mathrm{Softmax}\left(\frac{\mathbf{Q}_{\mathrm{dec}}\mathbf{QK}^{T}_{\mathrm{enc}}}{\sqrt{d_k}}\right)\mathbf{V}_{\mathrm{enc}}$$

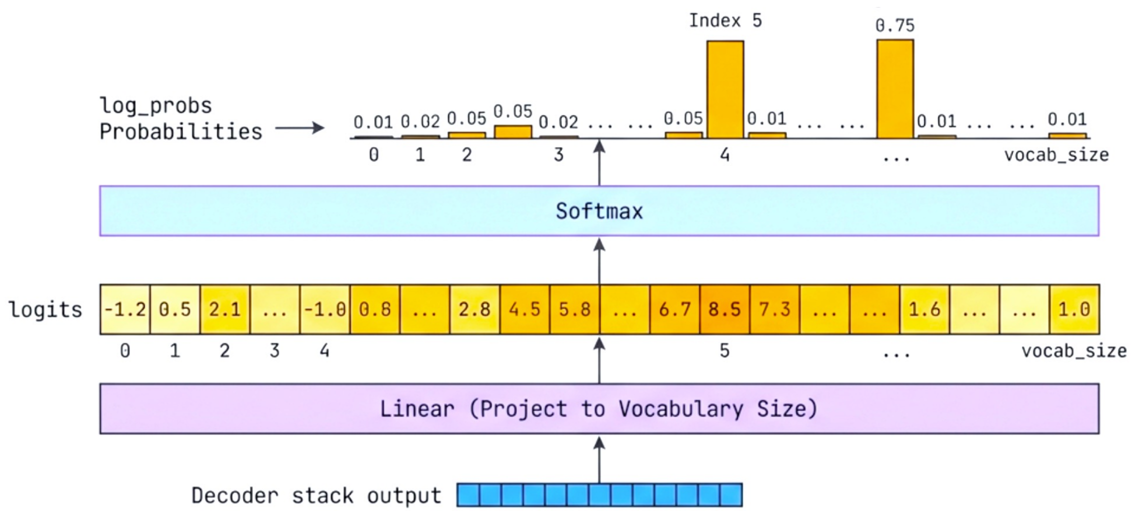


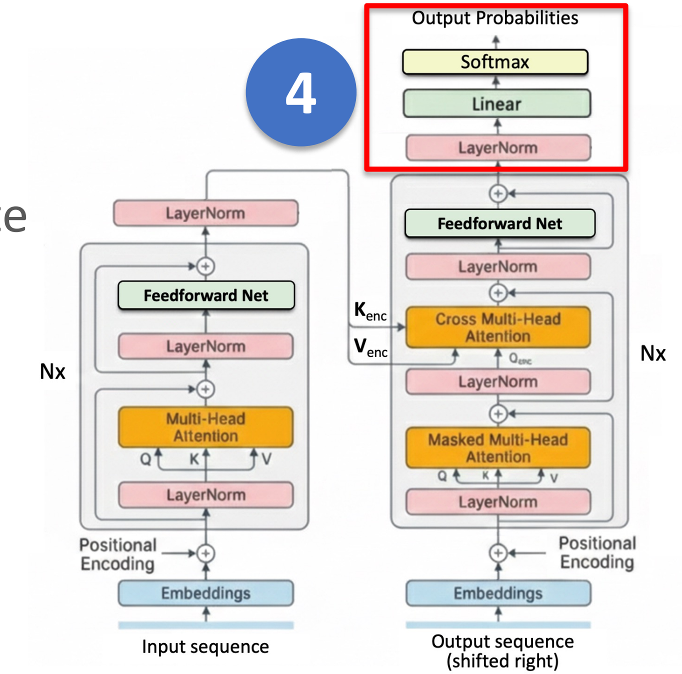


The Final Projection

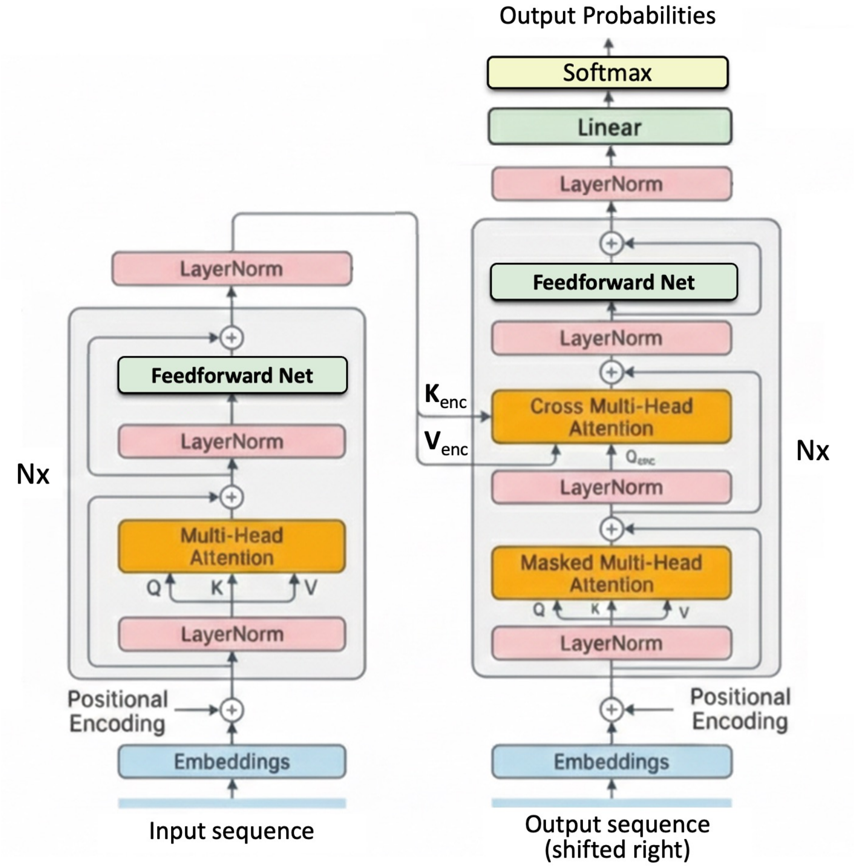
Projecting back to the vocabulary size (~50k words) and selecting the most likely next token.

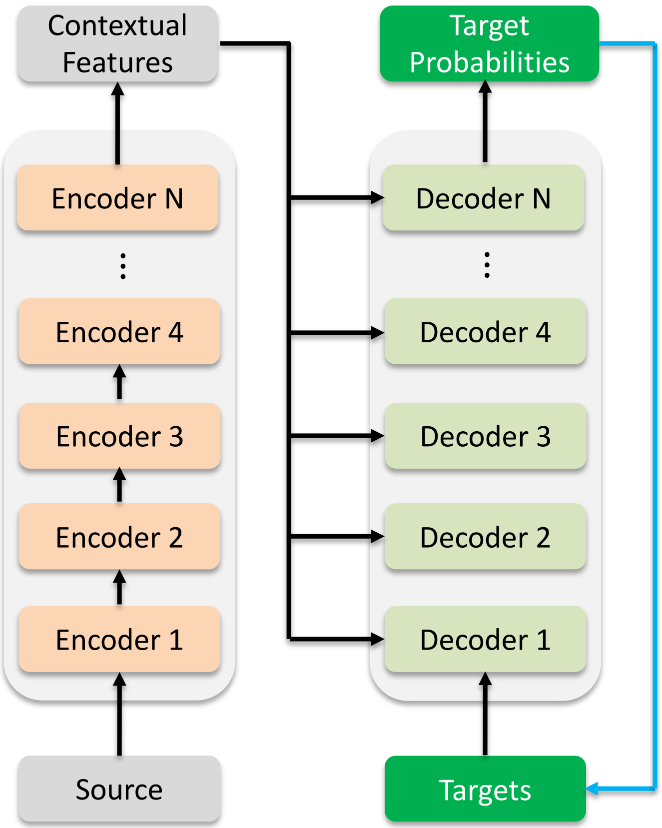
- **Linear Layer**: Maps vector to vocabulary size.
- **Softmax**: Converts scores to probabilities.
- **Output:** The token with the highest probability is selecte





Transformer's Architecture and Self-Attentions





Training and Inference

- **Training:** During training, the model is trained using teacher forcing, where the ground truth output sequence is fed into the decoder. The loss function is typically the cross-entropy loss between the predicted tokens and the ground truth tokens.
- **Inference:** During inference, the model generates the output sequence one token at a time. The previously generated tokens are fed back into the decoder to generate the next token, until the end of the sequence is reached.

Training Methodology

OBJECTIVE FUNCTION

- Cross-Entropy Loss (Minimizing error between prediction and target).

OPTIMIZER

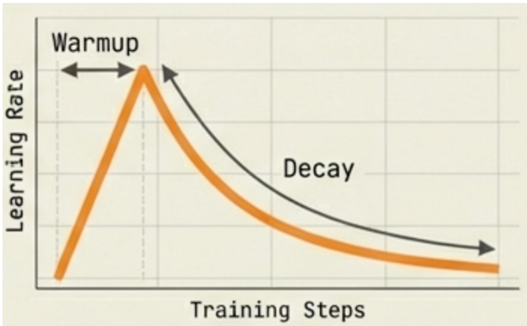
- Adam (Beta1=0.9, Beta2=0.98).

REGULARIZATION

- Dropout (0.1) applied to residuals and attention weights.

LEARNING RATE

- Linear Warmup + Inverse



Transformer's Machine Translation Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

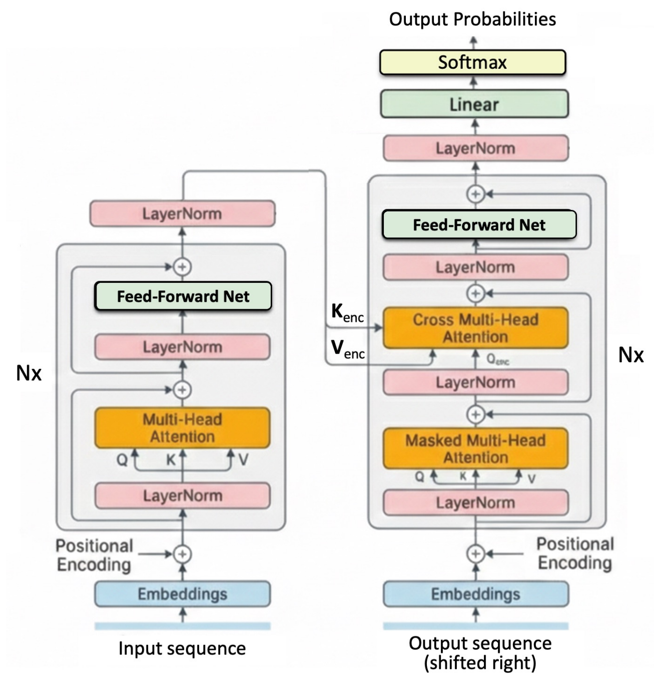
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- The Transformer model outperforms all previously released models and combinations, while being trained at a much lower cost compared to other competitive models.
- BLEU is a metric that measures how similar the machine-generated translation is to a set of reference translations.

Why The Transformer Changed Everything

The key architectural innovations that fueled the LLM revolution.

1. **Parallelization:** Unlike RNNs, Transformers process entire sequences at once. Training is massive and fast.
2. **Global Context:** Attention allows any token to look at any other token instantly, regardless of distance.
3. **Scalability:** Residuals and LayerNorm allow for deep stacking, enabling the massive LLMs (GPT, BERT, Claude) we use today.



Attention Is All You Need

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{T}}{\sqrt{d_k}}\right)\mathbf{V}$$

By treating language as an information retrieval task-matching Queries to Keys to retrieve Values-the Transformer captures the complex, non-linear relationships of huma communication. This single equation is the foundation of the Generative AI revolution.