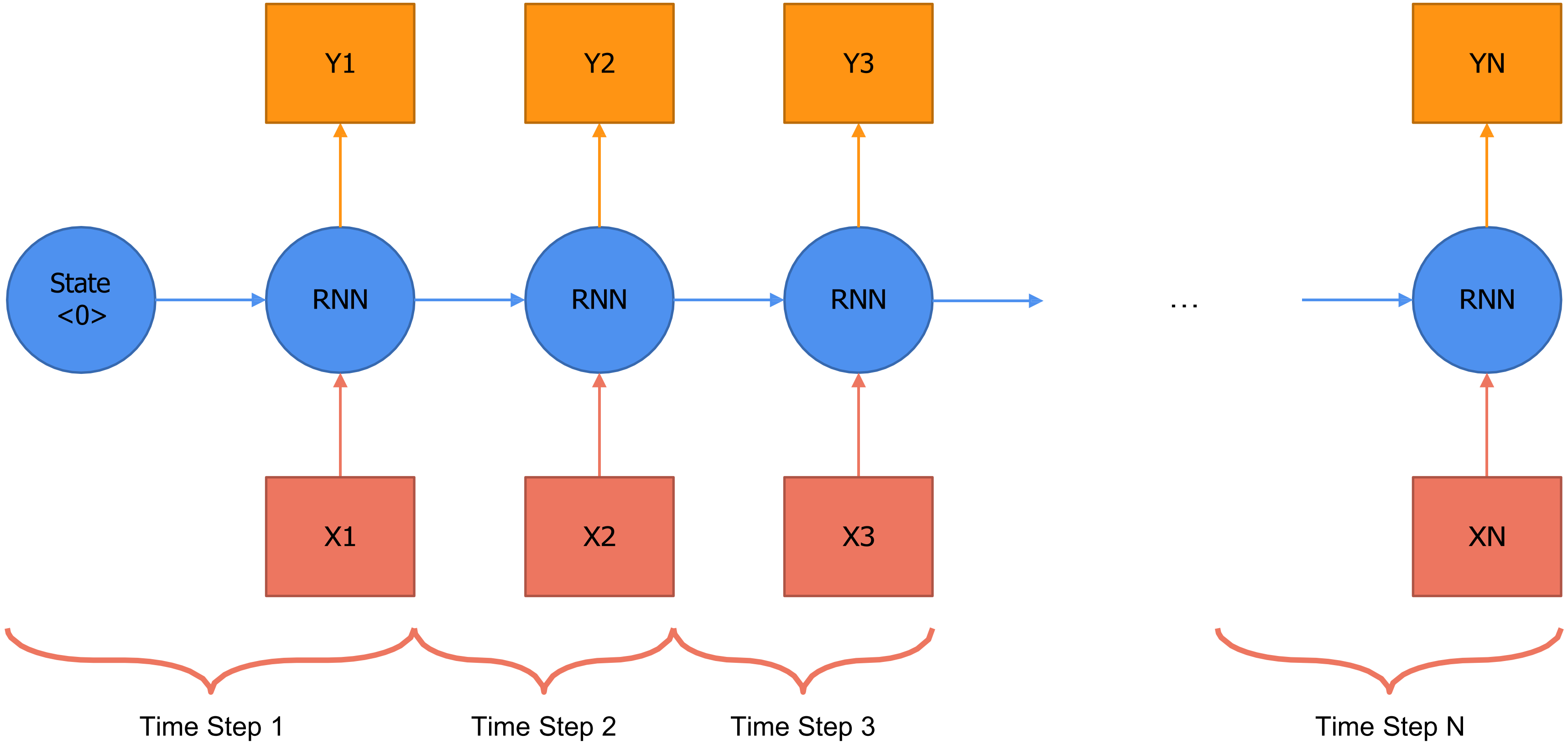
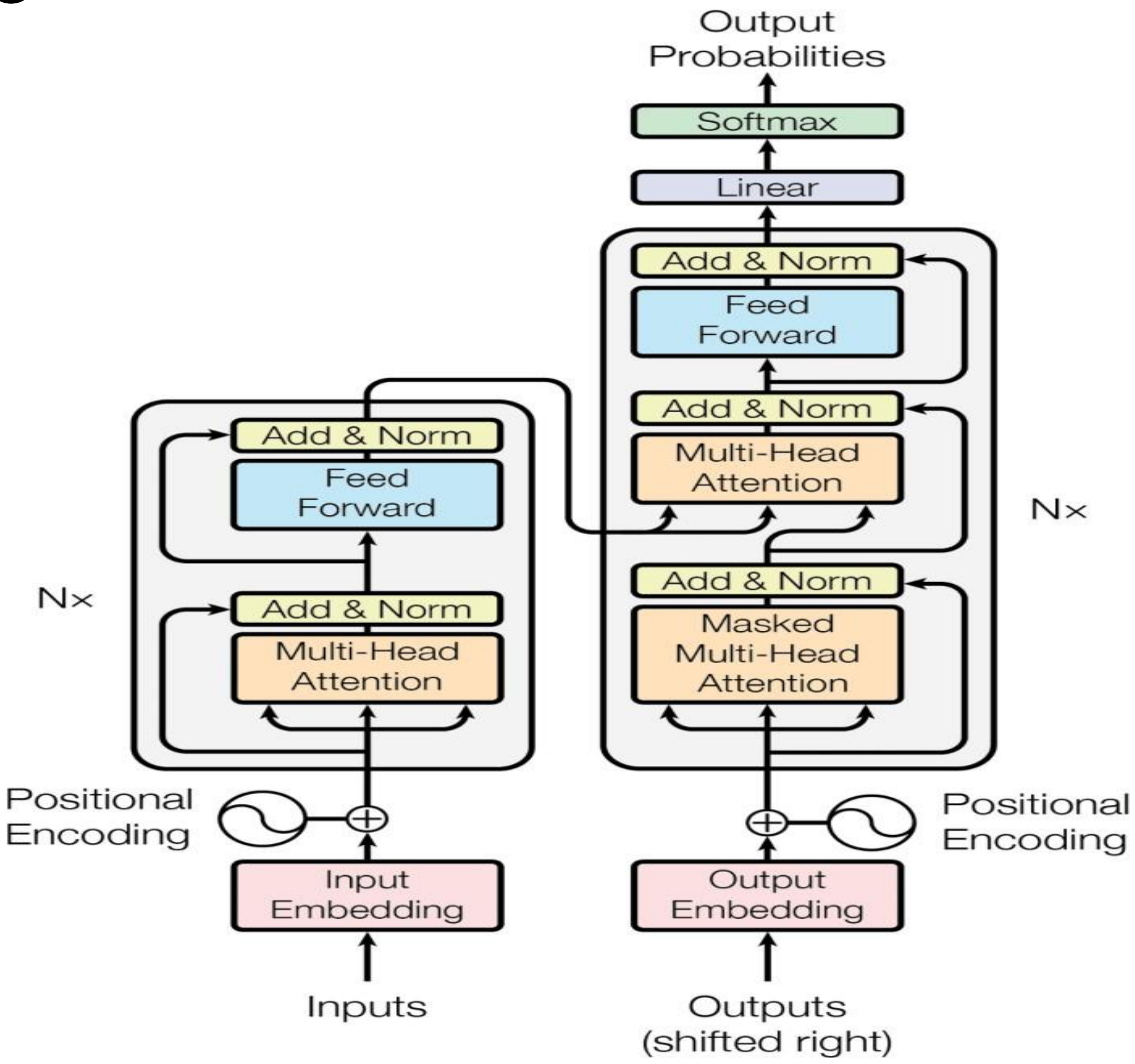
Recurrent Neural Networks (RNN)



Problems with RNN (among others)

1. Slow computation for long sequences
2. Vanishing or exploding gradients
3. Difficulty in accessing information from long time ago

Introducing the Transformer



https://github.com/VijayKanagaraj7/Transformers

What is a Transformer?

A **Transformer** is a deep learning model architecture.

It was introduced in the paper **"Attention is All You Need" (2017)** by Vaswani et al.

It revolutionized **NLP (Natural Language Processing)** by removing the need for RNNs or CNNs.

Used in models like BERT, GPT, T5, etc.

What are Transformers used for?

- **Text generation** (e.g., ChatGPT, Bard)
- **Language translation** (e.g., English ↔ Tamil)
- **Text summarization**
- **Sentiment analysis**
- **Question answering**
- **Code generation** and more...

Basic Idea Behind Transformers:

They use a concept called **"Attention"**.

Instead of reading words one by one like RNNs, transformers **look at the whole input at once**.

They focus on **important words** using **attention scores**.

Key Parts of Transformer Architecture:

2. **Encoder-Decoder Structure**

- **Encoder**: Processes the input sentence.
- **Decoder**: Generates the output sentence (used in translation, summarization, etc.).
- **GPT uses only decoder**, **BERT uses only encoder**, **T5 uses both**.

Encoder Architecture (Repeated N times):

Each encoder block contains:

a. **Self-Attention Layer**

- Every word looks at **all other words** in the sentence.
- Helps understand context.
Example: "He ate the apple because **it** was tasty" — what does "it" refer to?

b. **Feed-Forward Neural Network**

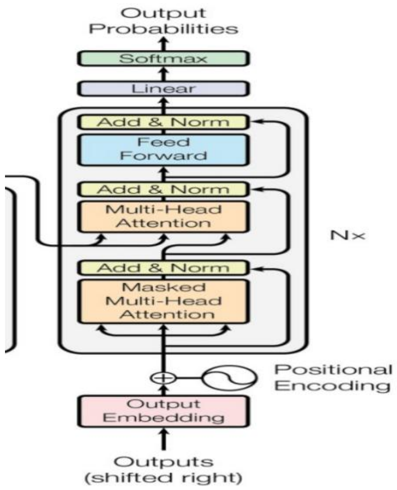
A simple 2-layer MLP (multi-layer perceptron) for more processing.

c. **Add & Normalize**

Adds original input back (residual connection) + normalizes for stability.

Decoder Architecture (Also repeated N times):

Each decoder block contains:



a. **Masked Self-Attention**

Looks only at previous words (for predicting the next word during generation).

b. **Encoder-Decoder Attention**

Connects encoder’s understanding with decoder’s output generation.

c. **Feed-Forward Network**

-

Same as in encoder.

d. **Add & Normalize**

- Again, residual connections + normalization.

- Attention Mechanism (Core Concept)

- Uses **Query, Key, Value (Q, K, V)** vectors.

- Calculates **attention scores** to decide which words to focus on.

- Scaled Dot-Product Attention is the formula used.

- Extended to **Multi-Head Attention** — lets the model attend to information from different perspectives.

Training a Transformer:

- Uses large datasets and **backpropagation**.
- Trained with objectives like:
  - **Masked Language Modeling (BERT)**
  - **Next Token Prediction (GPT)**

Advantages of Transformers:

◆ **Fast training** (parallel processing)
◆ **Better context handling**
◆ **Scalable** (used in massive models with billions of parameters)

Real-World Transformer Models:

◆ **BERT** – used for understanding text.
◆ **GPT** – used for generating text.
◆ **T5** – text-to-text tasks (e.g., summarization, translation).
◆ **ViT** – Vision Transformer (for images)

What is an input embedding?

1. Input Embedding

Words are converted into **vectors (numbers)** using embeddings.

Positional encoding is added to show the position of each word (since there's no recurrence).

Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT
Input IDs (position in the vocabulary)	105	6587	5475	3578	65	6587
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819
	…	…	…	…	…	…
	1.658	3633.421	7679.805	2716.194	2141.081	3633.421
	2671.529	8390.473	4506.025	5119.949	735.147	8390.473

We define d_{model} = 512, which represents the size of the embedding vector of each word

https://github.com/VijayKanagaraj7/Transformers

What is positional encoding?

- We want each word to carry some information about its position in the sentence.
- We want the model to treat words that appear close to each other as “close” and words that are distant as “distant”.
- We want the positional encoding to represent a pattern that can be learned by the model.

Positional encoding:

- Positional encoding in Transformers is a technique used to inject information about the order of tokens in a sequence.Transformers have no inherent sense of position due to their parallel processing nature.

- It works by adding a unique vector to each input token's embedding based on its position in the sequence, allowing the model to distinguish between different positions. Typically, this is done using sinusoidal functions that generate a fixed pattern of values for each position and dimension, enabling the model to learn relative positions and generalize to sequences of different lengths during training and inference.

What is positional encoding?

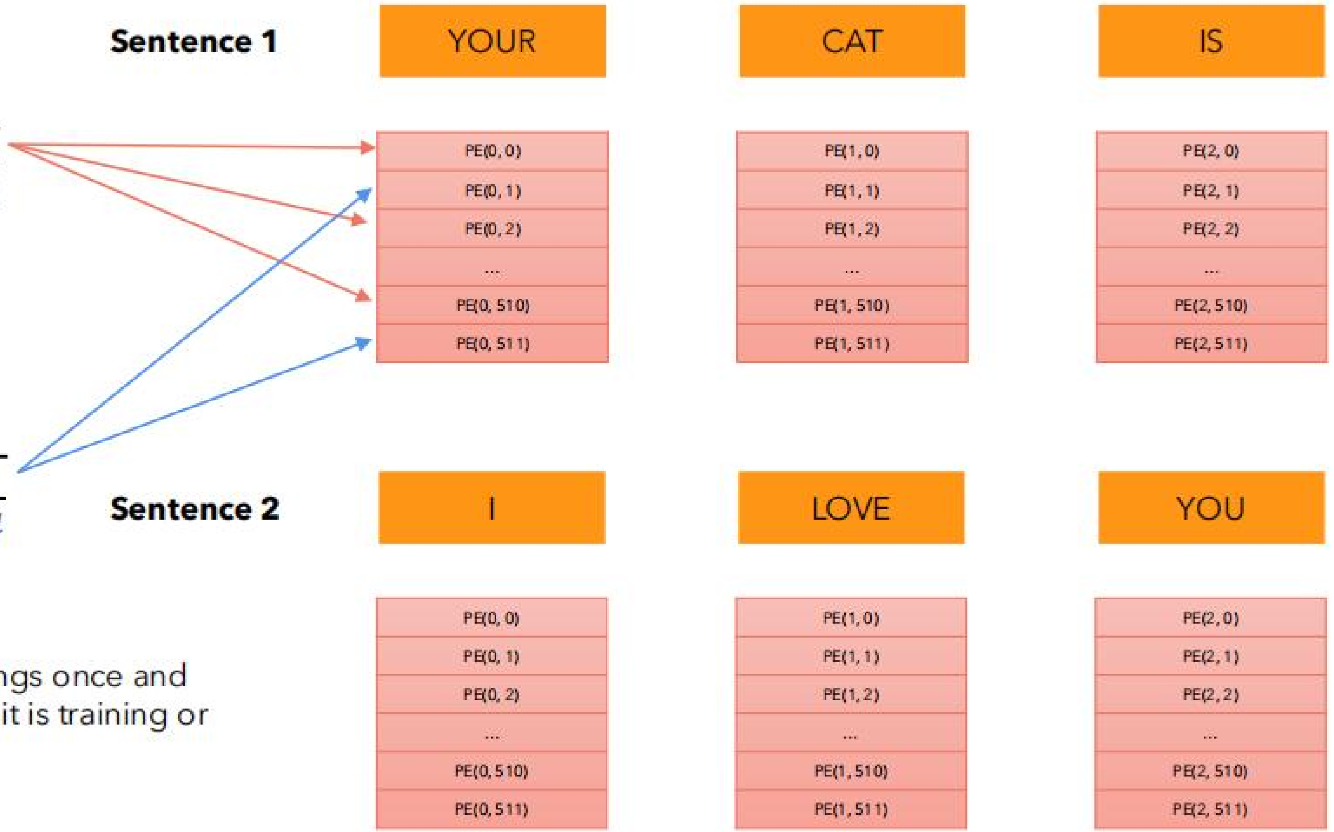
	YOUR	CAT	IS	A	LOVELY	CAT
Original sentence	YOUR	CAT	IS	A	LOVELY	CAT
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819
	…	…	…	…	…	…
	1.658	3633.421	7679.805	2716.194	2141.081	3633.421
	2671.529	8390.473	4506.025	5119.949	735.147	8390.473
	+	+	+	+	+	+
Position Embedding (vector of size 512). Only computed once and reused for every sentence during training and inference.	…	1664.068	…		…	1281.458
	…	8080.133	…		…	7902.890
	…	2620.399	…		…	912.970
	…	…	…		…	3821.102
	…	9386.405	…		…	1659.217
	…	3120.159	…		…	7018.620
	=	=	=	=	=	=
Encoder Input (vector of size 512)	…	1835.479	…	…	…	1452.869
	…	11356.483	…	…	…	11179.24
	…	11813.218	…	…	…	10105.789
	…	…	…	…	…	…
	…	13019.826	…	…	…	5292.638
	…	11510.632	…	…	…	15409.093

https://github.com/VijayKanagaraj7/Transformers

What is positional encoding?

$$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

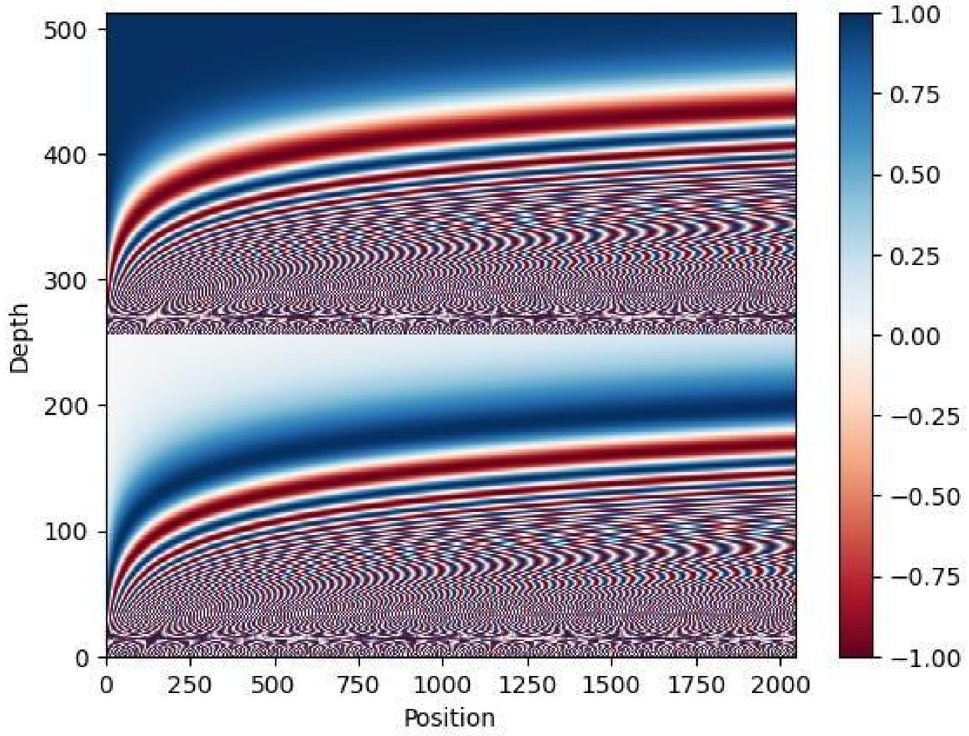


We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

https://github.com/VijayKanagaraj7/Transformers

Why trigonometric functions?

Trigonometric functions like **cos** and **sin** naturally represent a pattern that the model can recognize as continuous, so relative positions are easier to see for the model. By watching the plot of these functions, we can also see a regular pattern, so we can hypothesize that the model will see it too.



https://github.com/VijayKanagaraj7/Transformers

What is Self-Attention?

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

Self-Attention allows the model to relate words to each other.

In this simple case we consider the sequence length **seq** = 6 and $\mathbf{d_{model}} = \mathbf{d_k} = 512$.

The matrices **Q**, **K** and **V** are just the input sentence.

$$\text{softmax}\left(\frac{\underset{(6,\,512)}{\mathbf{Q}} \times \underset{(512,\,6)}{\mathbf{K^T}}}{\sqrt{512}}\right) =$$

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	**1**
CAT	0.124	0.278	0.201	0.128	0.154	0.115	**1**
IS	0.147	0.132	0.262	0.097	0.218	0.145	**1**
A	0.210	0.128	0.206	0.212	0.119	0.125	**1**
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	**1**
CAT	0.195	0.114	0.203	0.103	0.157	0.229	**1**

(6, 6)

* all values are random.

* for simplicity consider d only one head, which makes $d_{model} = d_k$.

https://github.com/VijayKanagaraj7/Transformers

How to compute Self-Attention?

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

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

(6, 6)

X

V

(6, 512)

=

Attention

(6, 512)

Each row in this matrix captures not only the meaning (given by the embedding) or the position in the sentence (represented by the positional encodings) but also each word's interaction with other words.

Self-Attention in detail

- Self-Attention is permutation invariant.
- Self-Attention requires no parameters. Up to now the interaction between words has been driven by their embedding and the positional encodings. This will change later.
- We expect values along the diagonal to be the highest.
- If we don't want some positions to interact, we can always set their values to $-\infty$ before applying the *softmax* in this matrix and the model will not learn those interactions. We will use this in the decoder.

a. **Self-Attention Layer**

Every word looks at **all other words** in the sentence.

Helps understand context.
Example: "He ate the apple because **it** was tasty" — what does "it" refer to?

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

https://github.com/VijayKanagaraj7/Transformers

Multi-head Attention

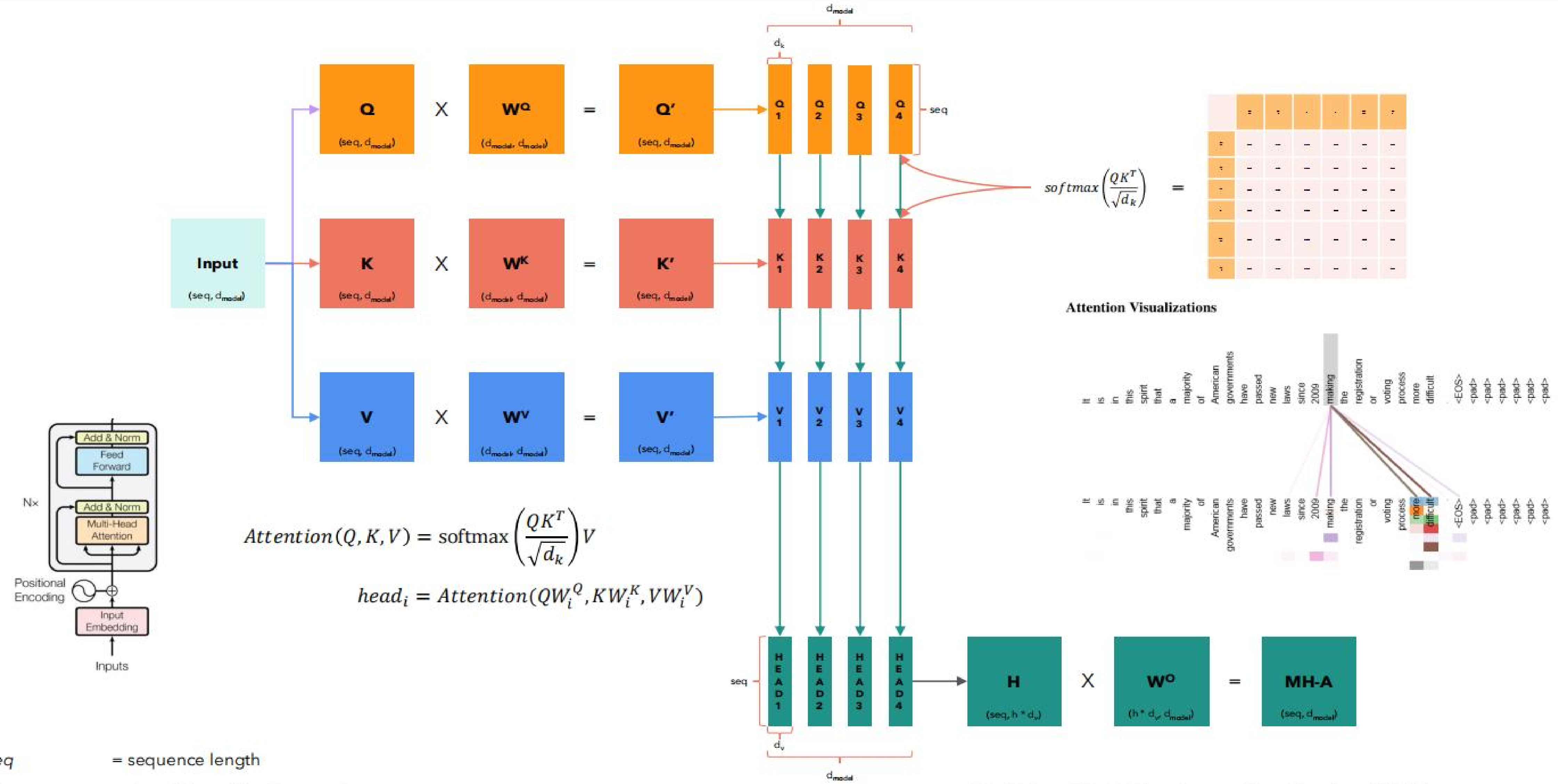
1. What is Multi-Head Attention?

It's the **core of the Transformer** that helps the model **focus on different parts of the sentence at once**.

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

$$MultiHead(Q, K, V) = Concat(head_1 \ \dots head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$



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

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q,K,V) = Concat(head_1 \ldots head_h)W^O$$

seq = sequence length

d_{model} = size of the embedding vector

h = number of heads

How it works (Step by Step):

1. The input (word embeddings) is passed through 3 learned matrices to get:

Q (Query)
K (Key)
V (Value)

2. We compute Attention Scores using:

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

$$MultiHead(Q,K,V) = Concat(head_1 \ \ldots head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

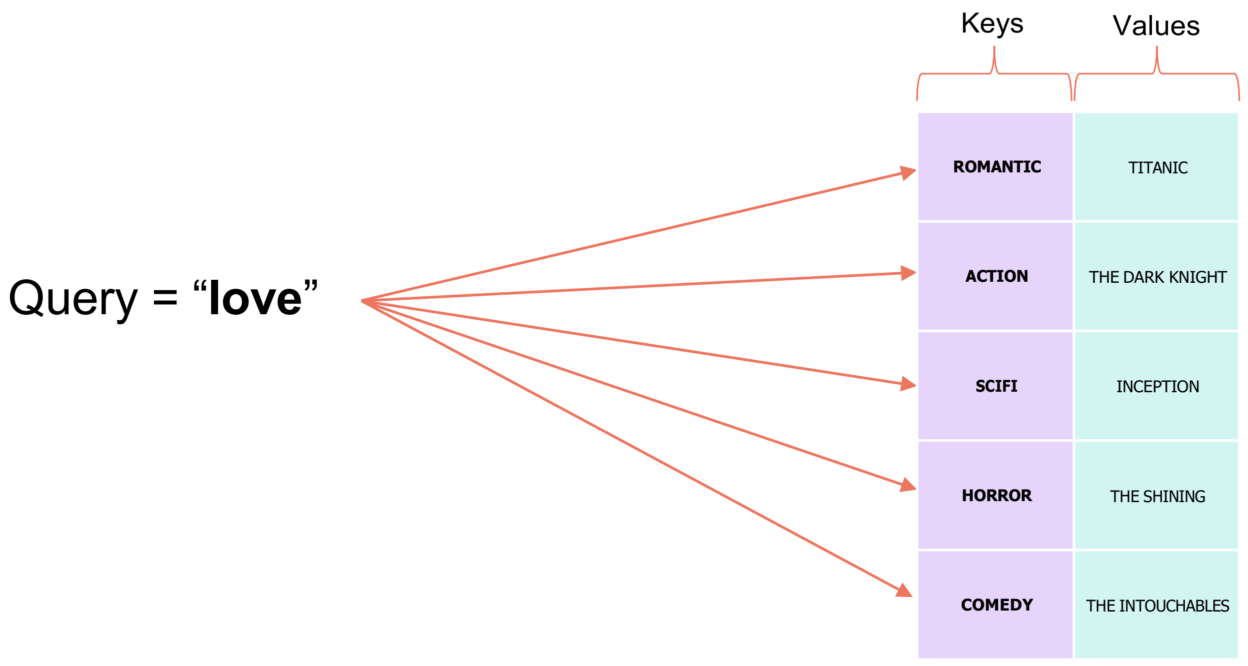
3. We split this into multiple heads:

Each head gets a portion of Q, K, V.
Each head learns different relationships.

4. Outputs from all heads are concatenated and passed through a final linear layer.

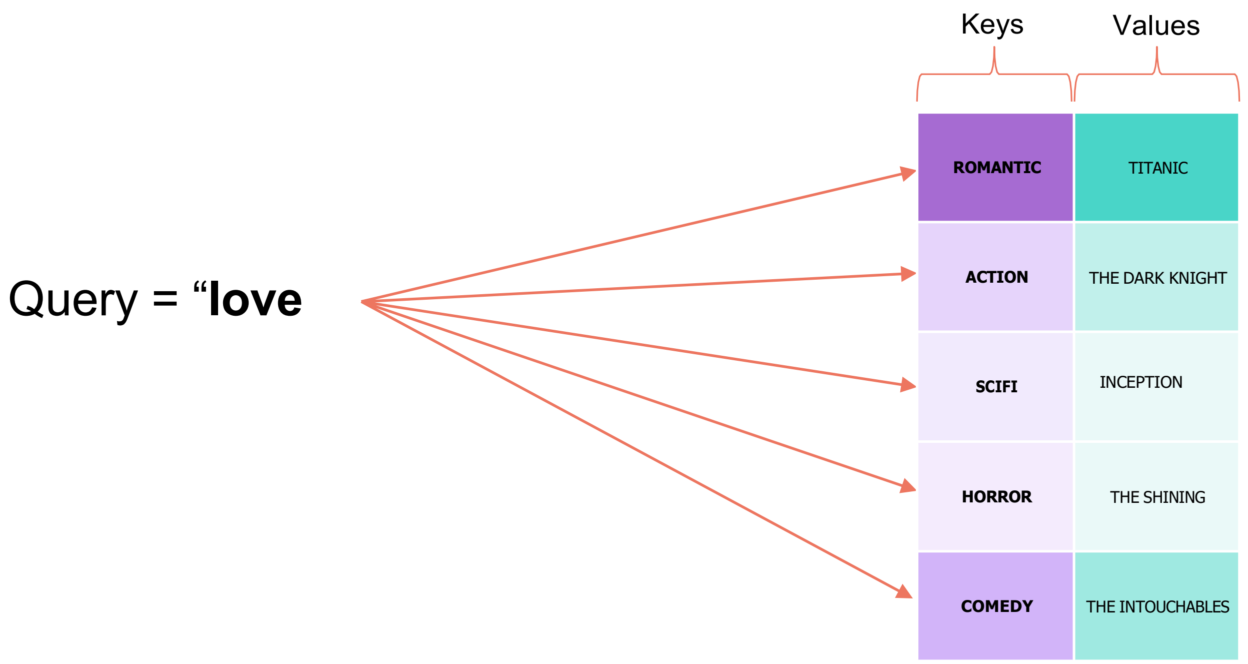
Why query, keys and values?

The Internet says that these terms come from the database terminology or the Python-like dictionaries.



Why query, keys and values?

The Internet says that these terms come from the database terminology or the Python-like dictionaries.



What is Masked Multi-Head Attention?

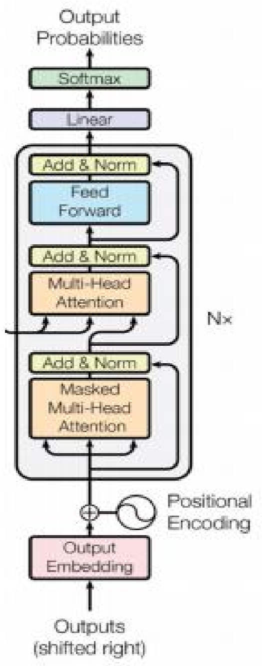
How it works:

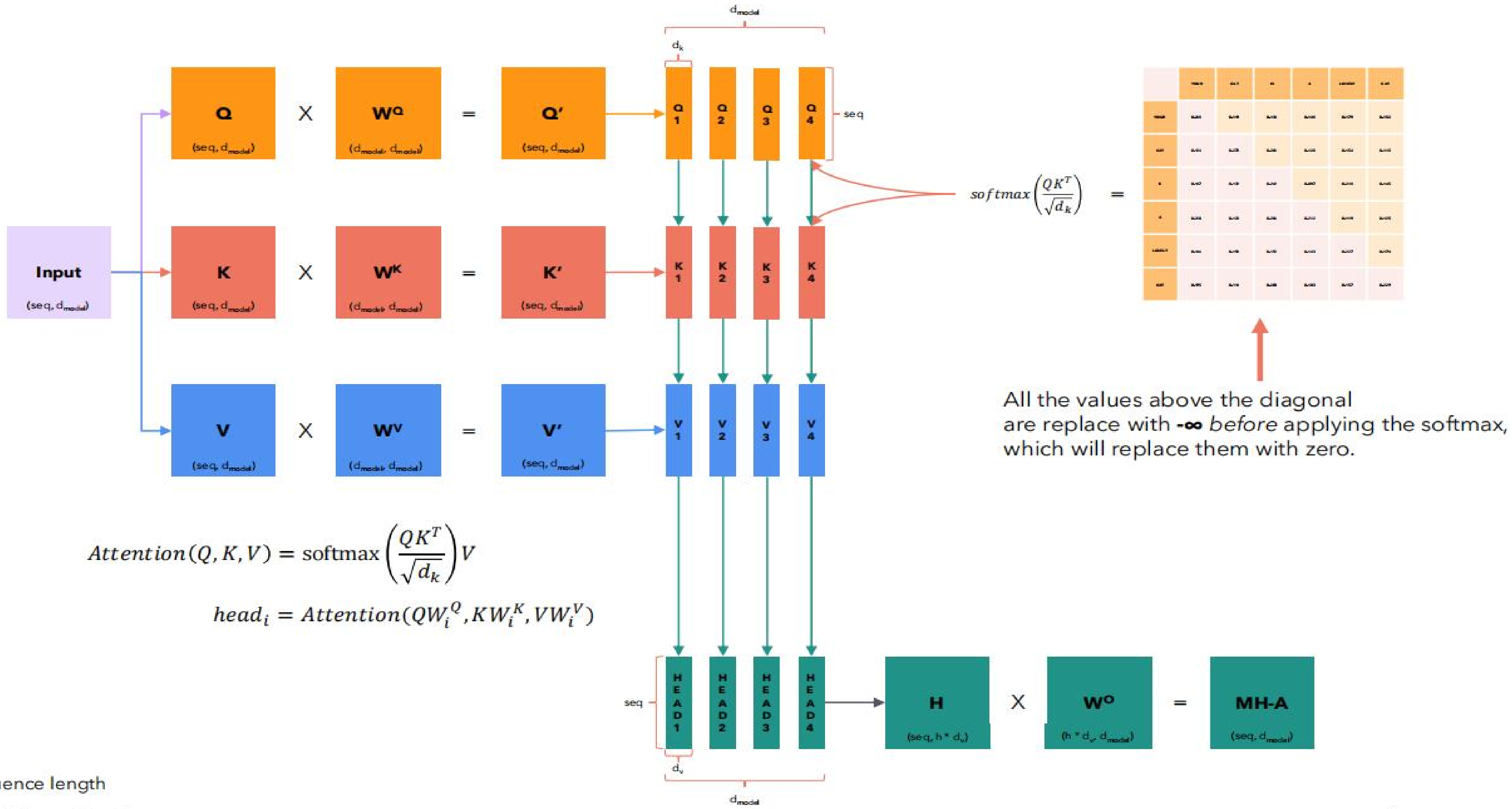
It works exactly like multi-head attention **but with a mask**.

The mask is a matrix where:

> Positions that are **not allowed to attend** (future words) are set to **-∞** (so softmax = 0).
> This blocks attention to future words

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	~~0.119~~	~~0.134~~	~~0.148~~	~~0.179~~	~~0.152~~
CAT	0.124	0.278	~~0.201~~	~~0.128~~	~~0.154~~	~~0.115~~
IS	0.147	0.132	0.262	~~0.097~~	~~0.218~~	~~0.145~~
A	0.210	0.128	0.206	0.212	~~0.119~~	~~0.125~~
LOVELY	0.146	0.158	0.152	0.143	0.227	~~0.174~~
CAT	0.195	0.114	0.203	0.103	0.157	0.229





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

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

$$MultiHead(Q, K, V) = Concat(head_1 \; ... \, head_h)W^O$$

seq = sequence length

d_{model} = size of the embedding vector

h = number of heads

Training

Time Step = 1

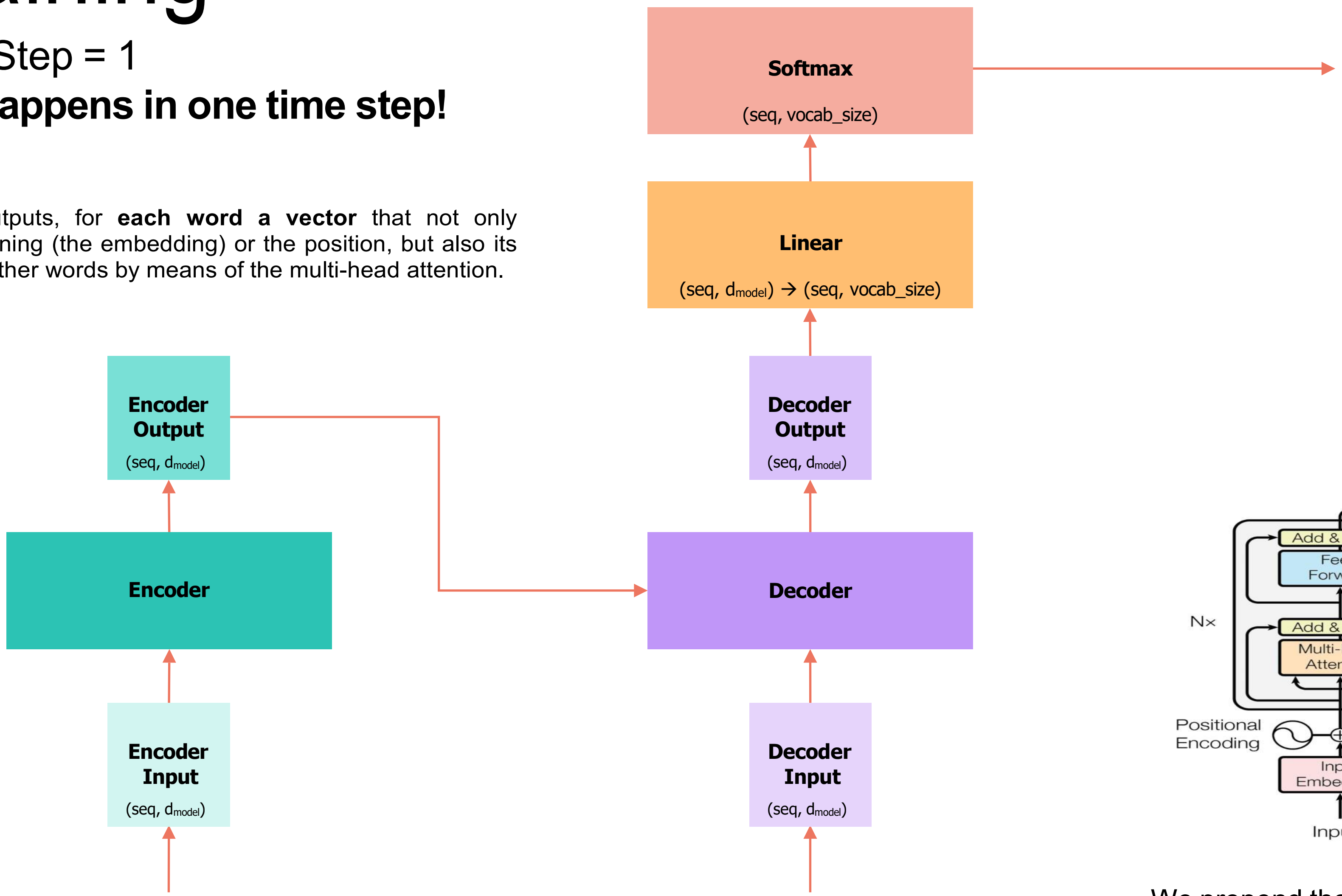
It all happens in one time step!

ti amo molto il cibo <EOS>

This is called the "label" or the "target"

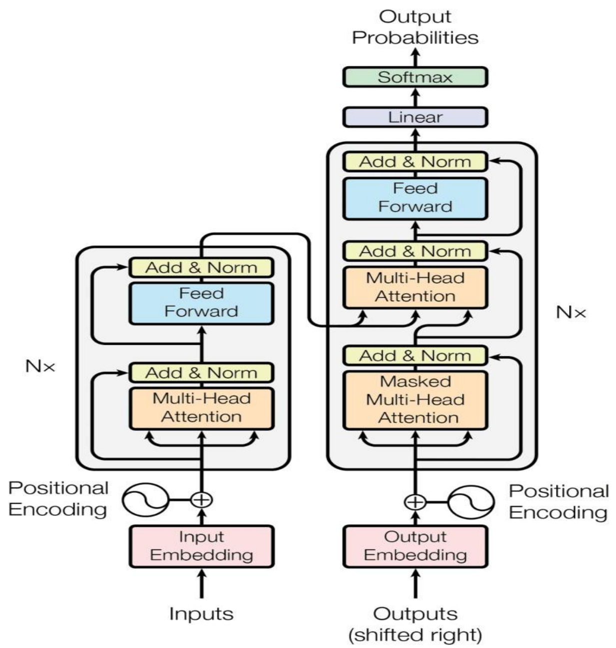
Cross Entropy Loss

The encoder outputs, for **each word a vector** that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.



<SOS> **What bro very wrong bro** <EOS>

<SOS> **ti amo molto il cibo**



We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

https://github.com/VijayKanagaraj7/Transformers

Inference

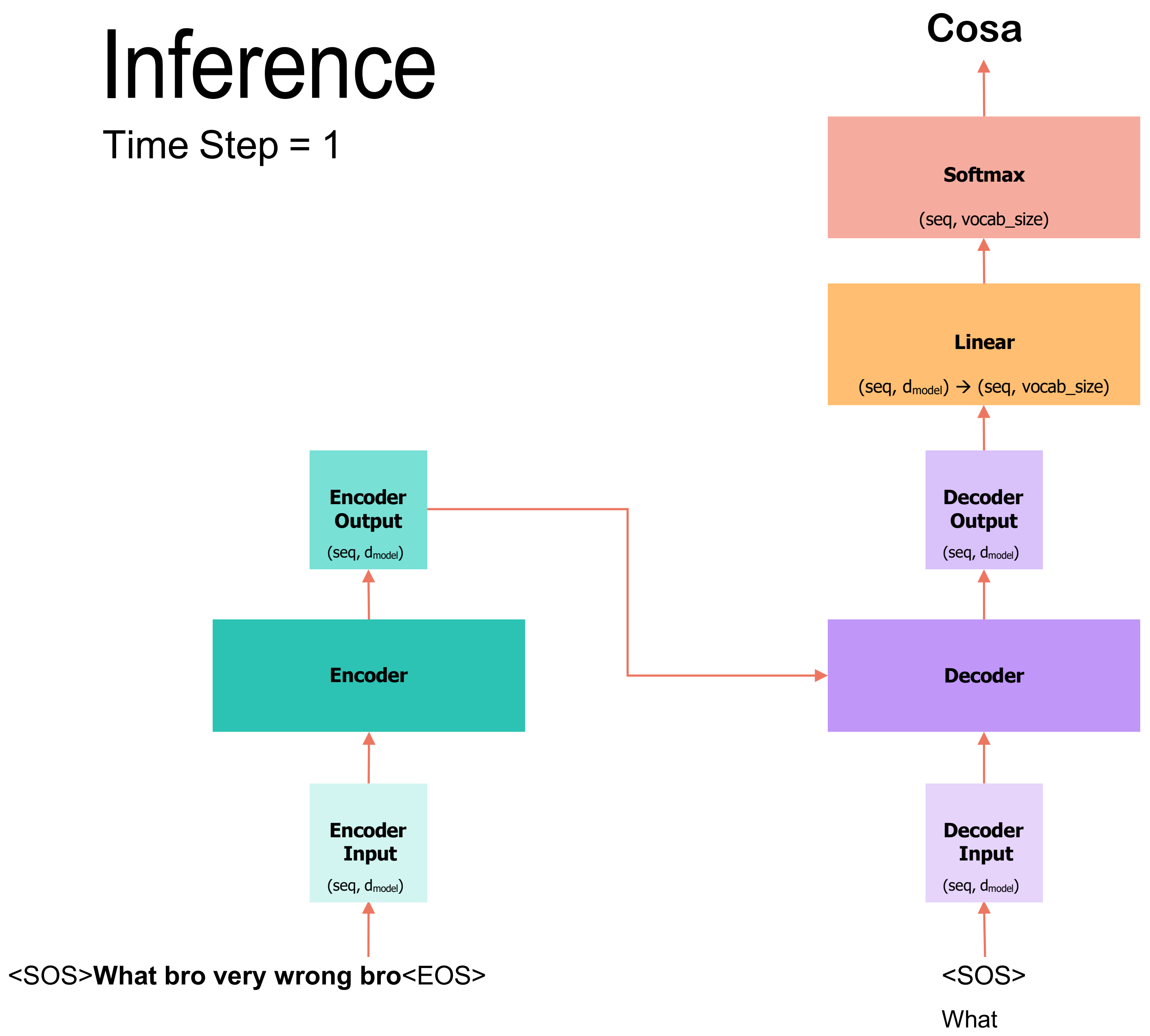
What bro very wrong bro

Cosa fratello qual coosa fratello

<SOS> - Start of sentence
<EOS> - End of sentence

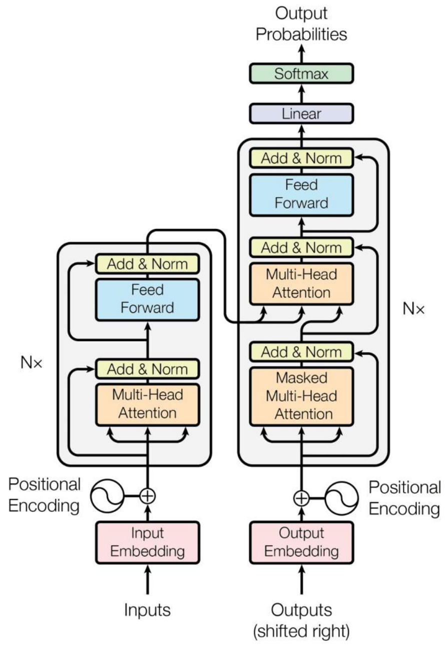
Inference

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

https://github.com/VijayKanagaraj7/Transformers

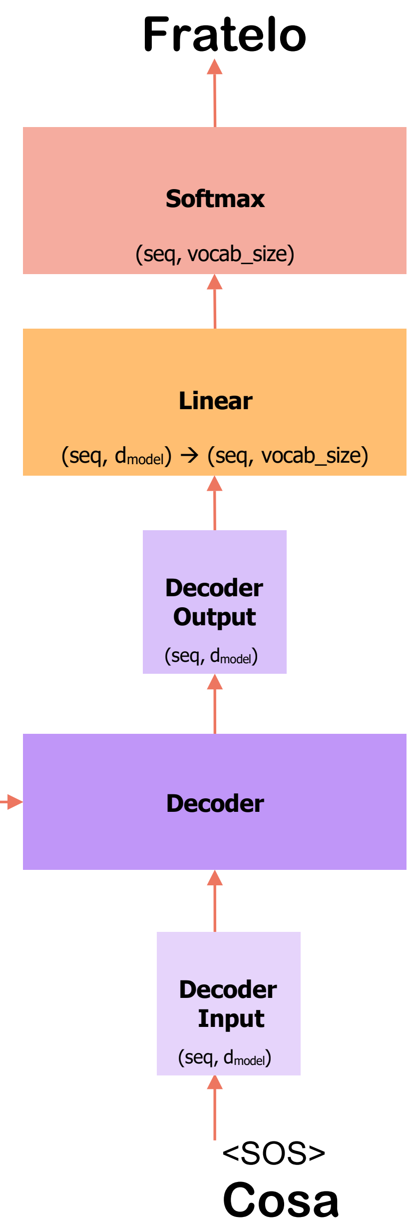
Inference

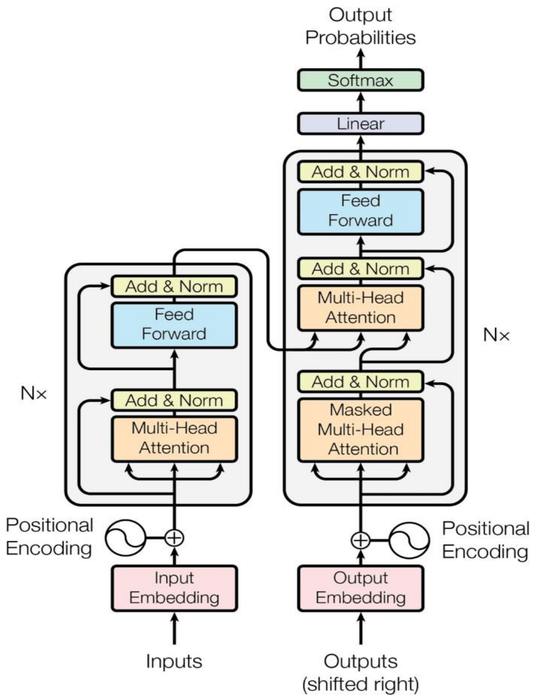
Time Step = 2

Since decoder input now contains **two** tokens, we select the softmax corresponding to the second token.

Use the encoder output from the first time step

<SOS>**What bro very wrong bro**<EOS>



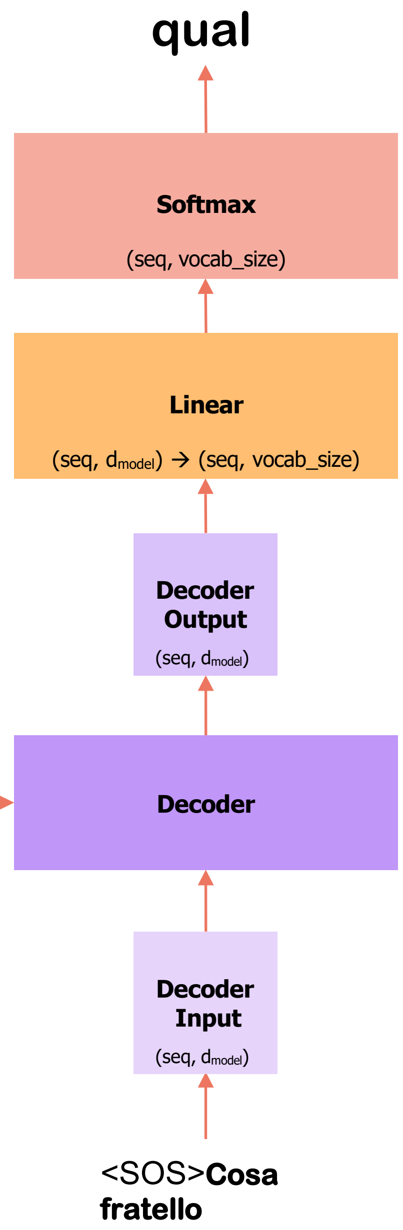


Append the previously output word to the decoder input

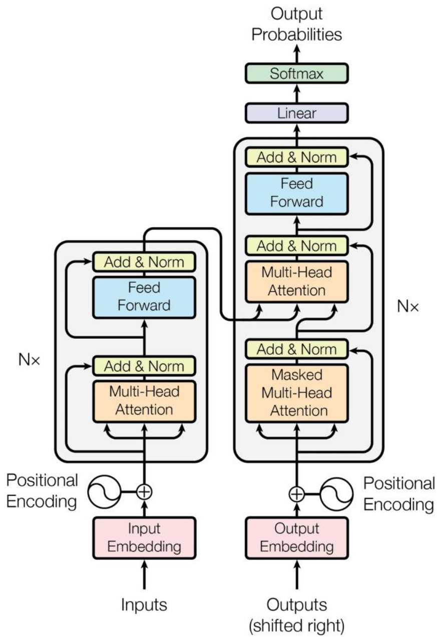
https://github.com/VijayKanagaraj7/Transformers

Inference

Time Step = 3



Since decoder input now contains **three** tokens, we select the softmax corresponding to the third token.



Use the encoder output from the first time step

<SOS>**What bro very wrong bro**<EOS>

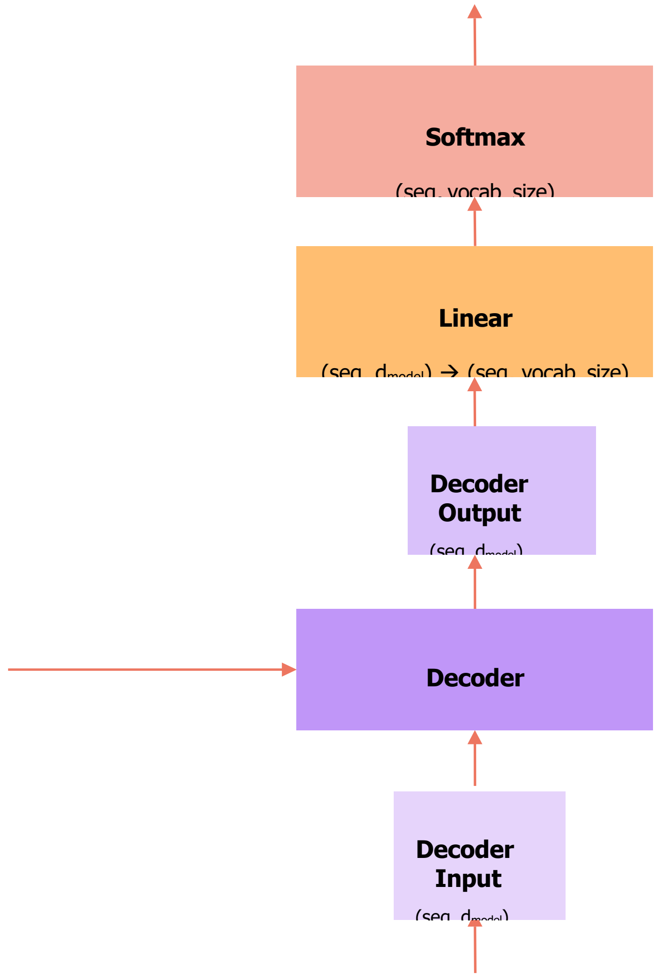
Append the previously output word to the decoder input

https://github.com/VijayKanagaraj7/Transformers

Inference

Time Step = 4

<EOS>frateo



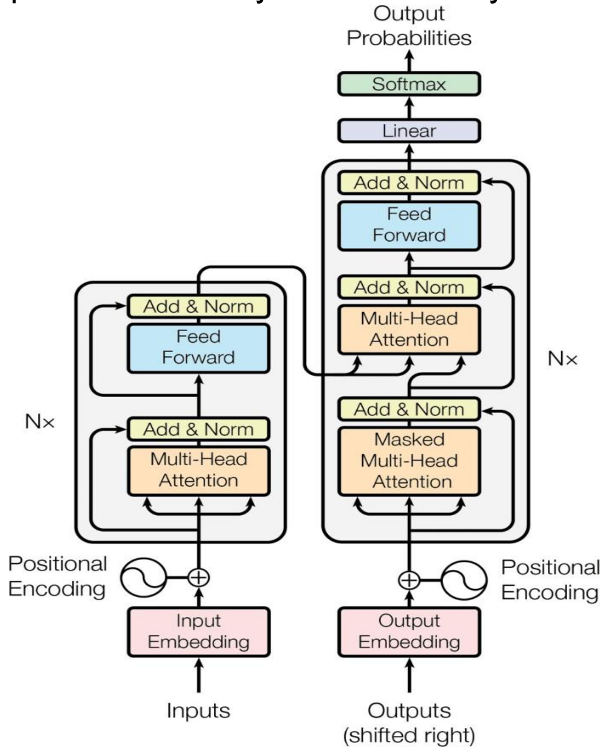
Use the encoder output from the first time step

<SOS>**What bro very wrong bro**<EOS>

<SOS> **Cosa fratello qual**

Since decoder input now contains **four** tokens, we select the softmax corresponding to the fourth token.

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



Append the previously output word to the decoder input

https://github.com/VijayKanagaraj7/Transformers

We selected, at every step, the word with the maximum softmax value. This strategy is called **greedy** and usually does not perform very well..

- Unlike greedy decoding (which picks the highest probability token at each step), beam search keeps track of the **top-k most probable sequences (called beams)** at each decoding step, where k is the **beam width.**
- **Beam Search** is a **heuristic search algorithm** used for decoding sequences in models like **Transformers**

Why Use Beam Search in Transformers?

Transformers (like GPT, BERT for generation, T5, etc.) output probabilities over a vocabulary at each step. Beam search improves the quality of the final output by:

✓ Leads to more **context accurate** and **natural-sounding outputs**.
✓ **Explores multiple options** instead of just one path.
Increase fluency and relevance of generated text.
Generates more coherent and grammatically correct outputs

THANK YOU

NEXT TOPIC (building a transformer from scratch)

WILL BE POSTED SOON@ https://github.com/VijayKanagaraj7/Transformers