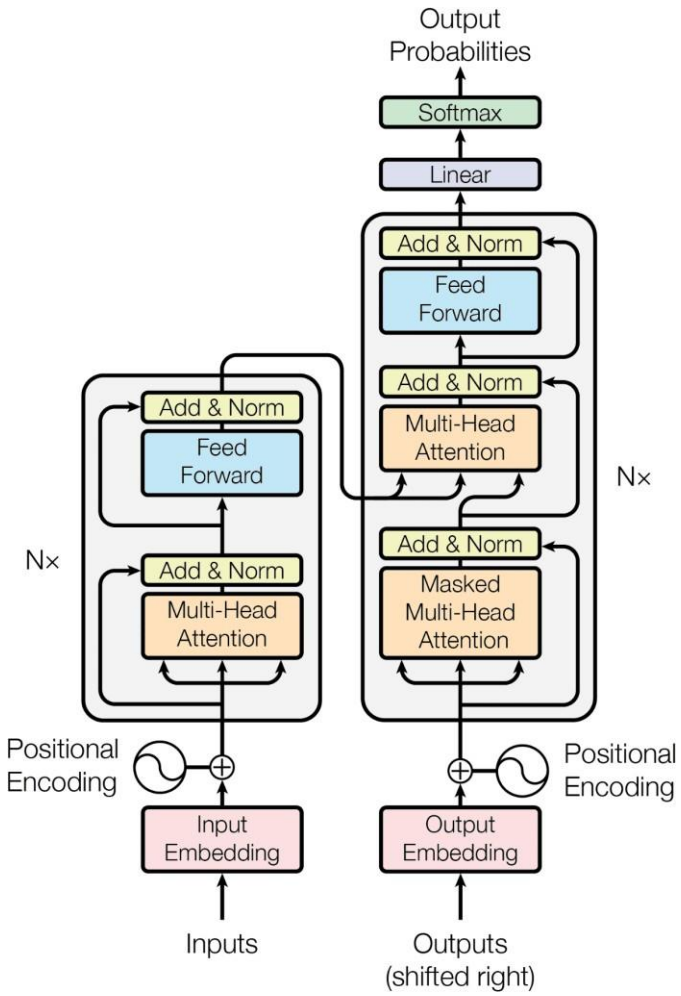
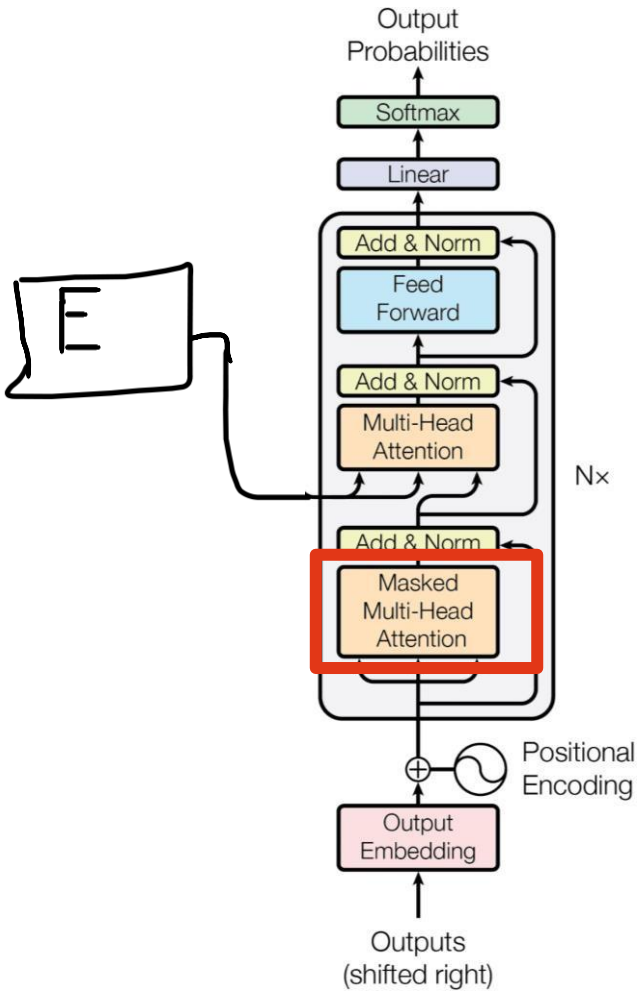


Transformers

Paper: Attention is all you need

Decoder

Cross attention: keys, values from encoder
Queries from decoder

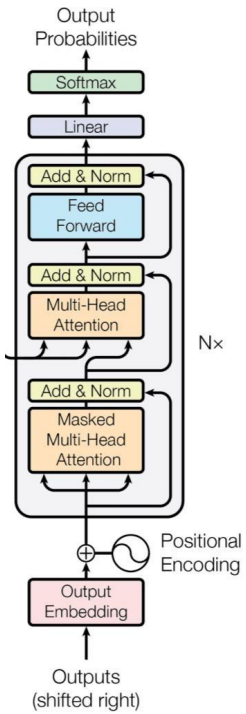


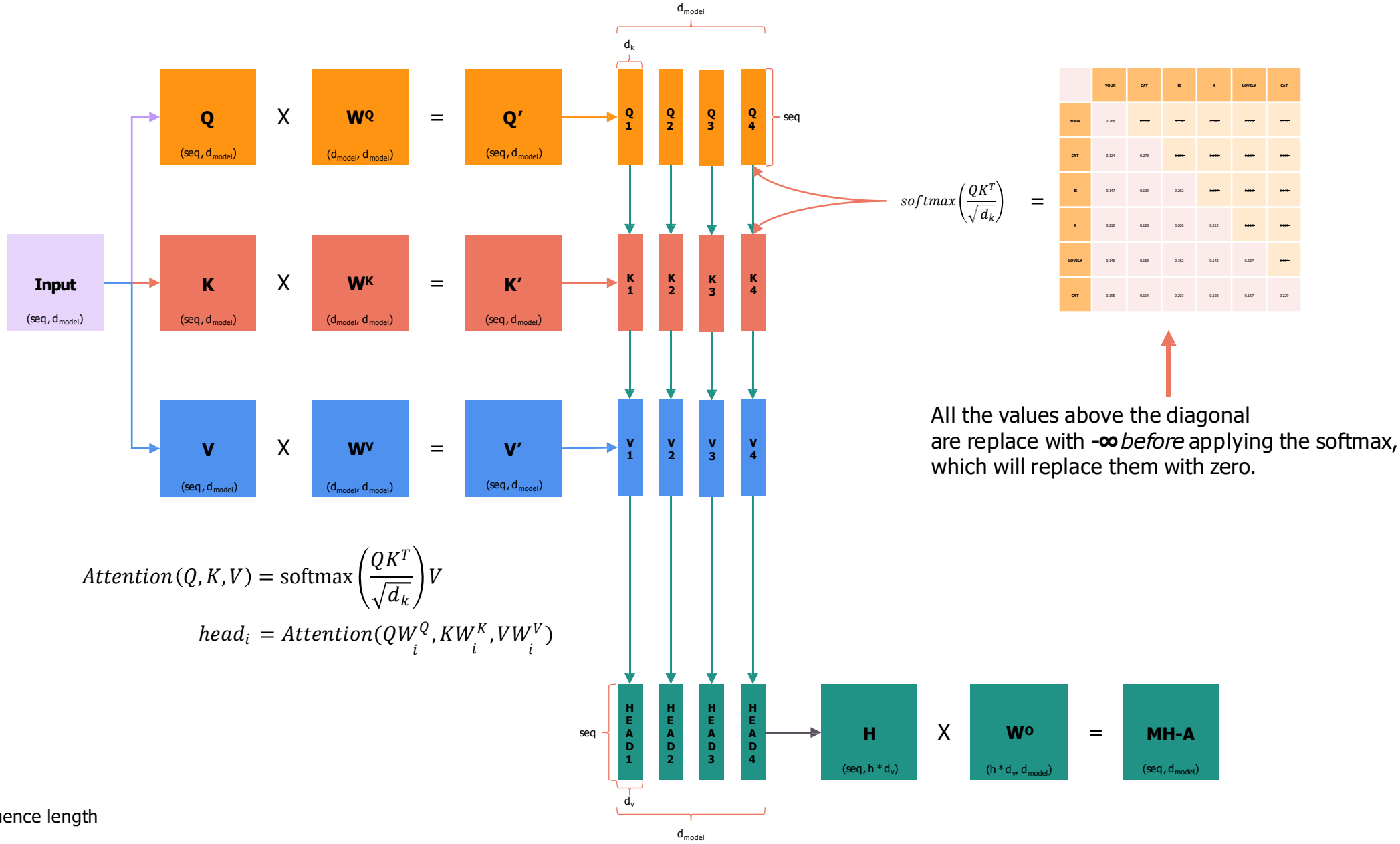
Cross attention:
The Query matrix is used to attend to the Key matrix, and the output is weighted by the Value matrix. This process allows the decoder to "ask" the encoder about specific parts of the source language input, and use that information to generate the target language output.
The cross-attention weights tell us how much each source language token should be "attended" to, in order to generate the next target language token.

What is Masked Multi-Head Attention?

Our goal is to make the model causal: it means the output at a certain position can only depend on the
words on the previous positions. The model **must not** be able to see 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





All the values above the diagonal are replace with **-∞** *before* applying the softmax, which will replace them with zero.

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

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

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

seq = sequence length

d_{model} = size of the embedding vector

h = number of heads

$d_k = d_v$ = d_{model} / h

Inference and training of a Transformer model

Training

I love you very much

Ti amo molto

Training

Time Step = 1

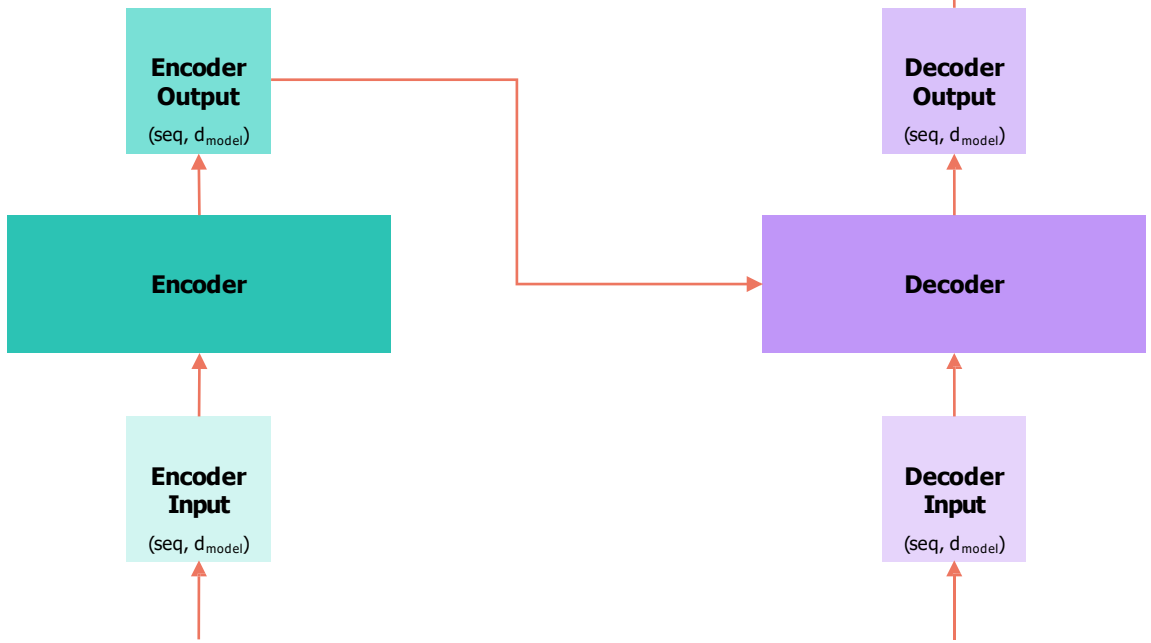
It all happens in one time step!

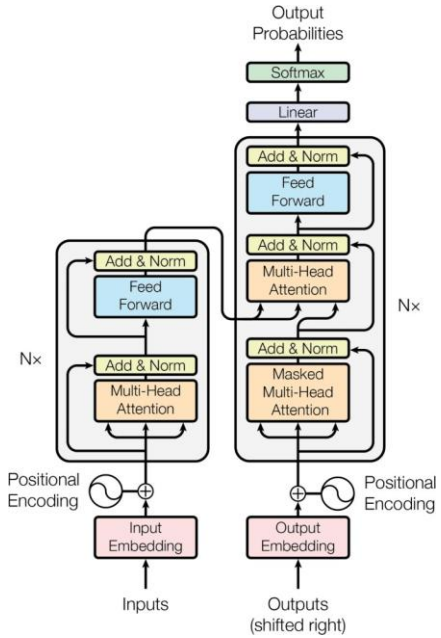
Ti amo molto <EOS>

*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.





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

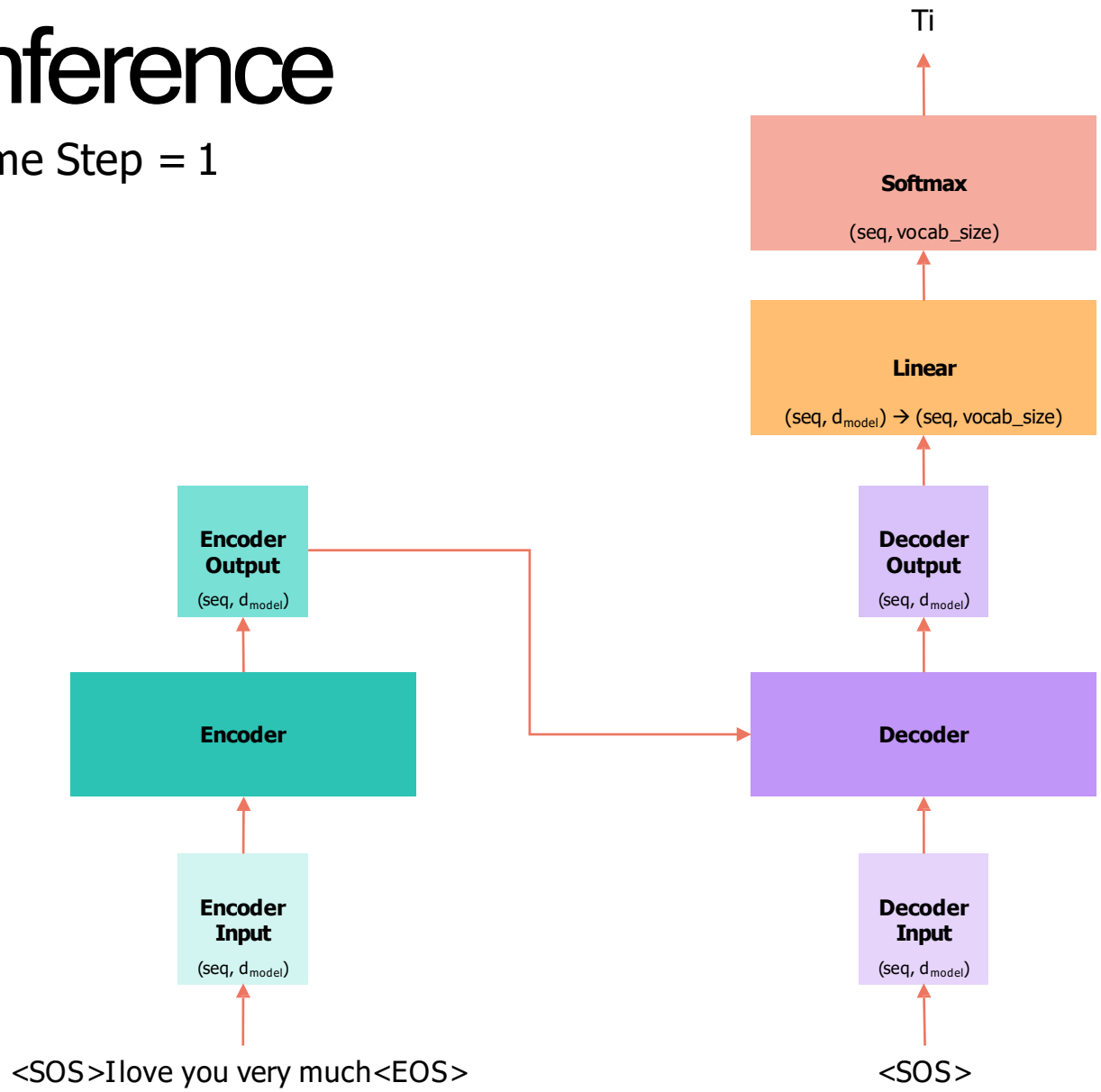
Inference

I love you very much

Ti amo molto

Inference

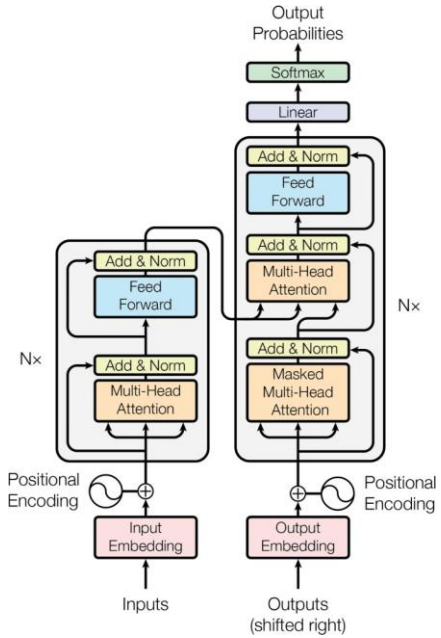
Time Step = 1



- In a loop, the decoder generates one token at a time, and the generated token is used as input for the next step.
- The process continues until an end-of-sequence token is generated or a maximum sequence length is reached.

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

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



* Both sequences will have same length thanks to padding

Inference

Time Step = 2

amo

Softmax

(seq, vocab_size)

Linear

(seq, d_{model}) → (seq, vocab_size)

Decoder Output

(seq, d_{model})

Decoder

Use the encoder output from the first time step

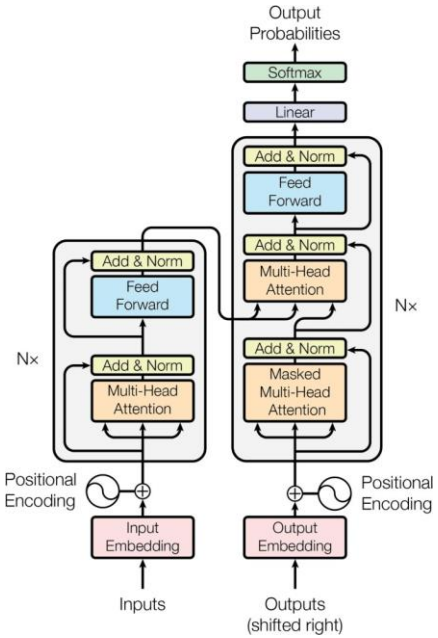
Decoder Input

(seq, d_{model})

<SOS>Ilove you very much<EOS>

<SOS> ti

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



Append the previously output word to the decoder input

Inference

Time Step = 3

molto

Softmax

(seq, vocab_size)

Linear

(seq, d_{model}) → (seq, vocab_size)

Decoder Output

(seq, d_{model})

Decoder

Use the encoder output from the first time step

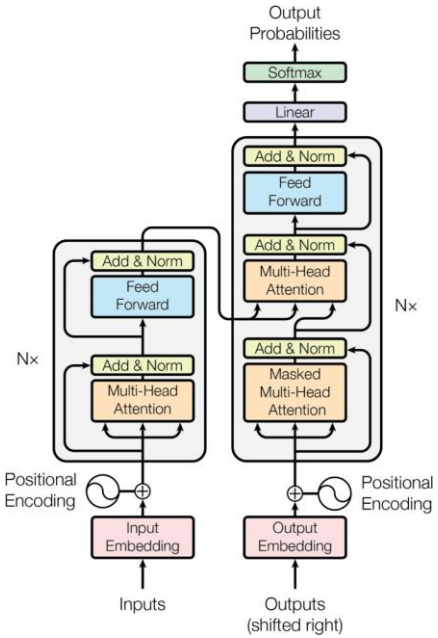
Decoder Input

(seq, d_{model})

<SOS>Ilove you very much<EOS>

<SOS> ti amo

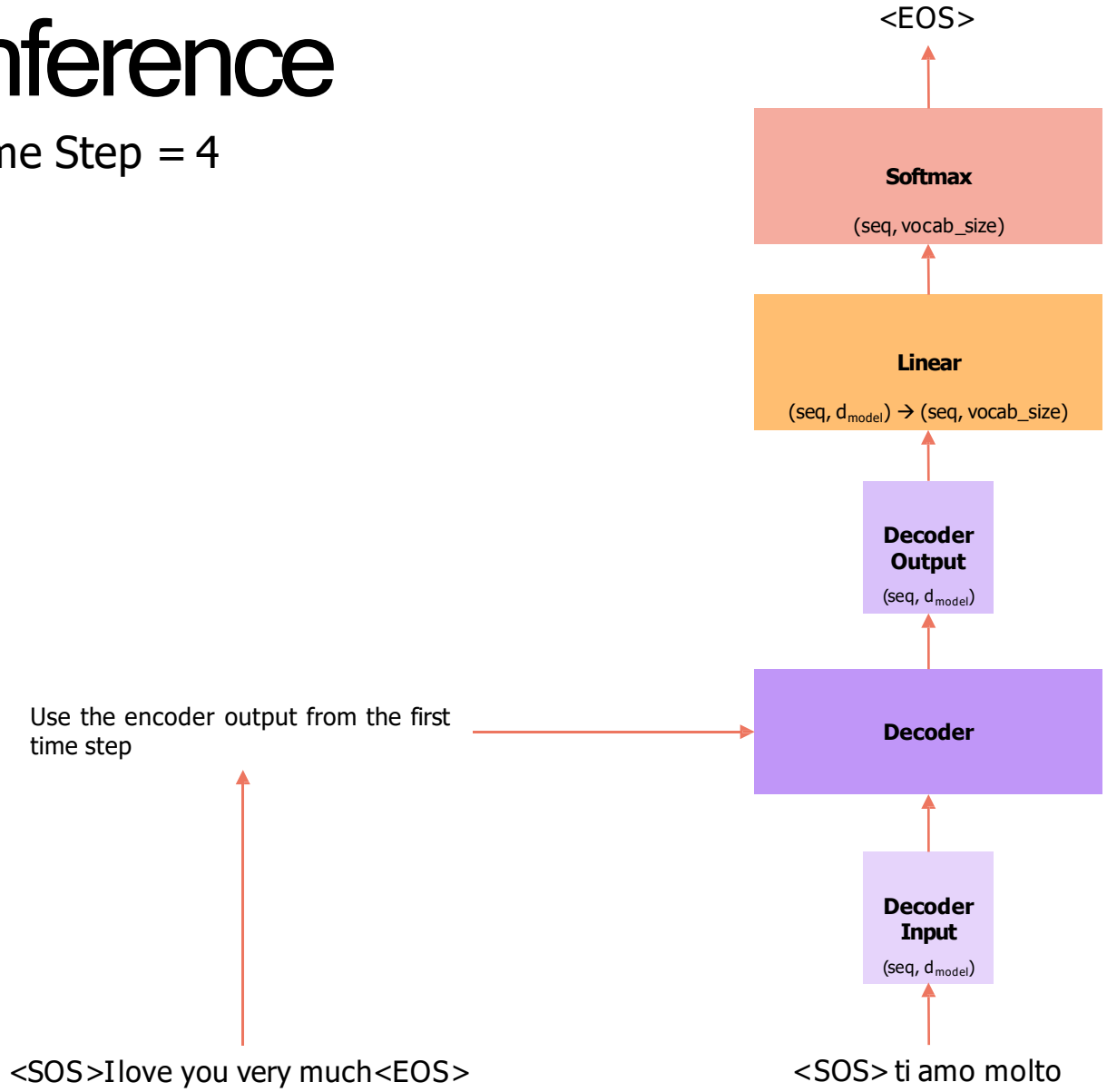
Since decoder input now contains **three** tokens, we select the softmax corresponding to the third token.



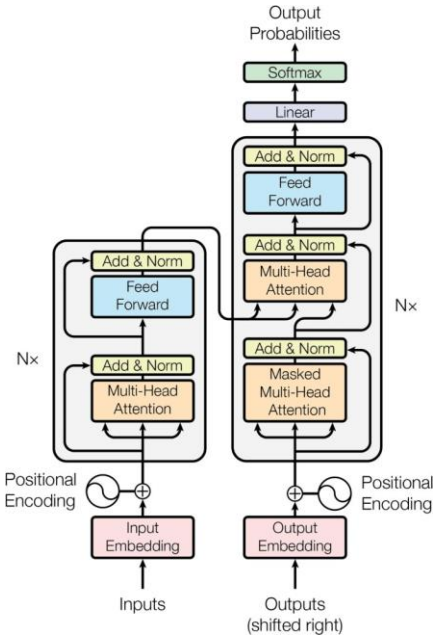
Append the previously output word to the decoder input

Inference

Time Step = 4



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



Append the previously output word to the decoder input

Inference strategy

- We selected, at every step, the word with the maximum softmax value. This strategy is called **greedy** and usually does not perform very well.
- A better strategy is to select at each step the top *B* words and evaluate all the possible next words for each of them and at each step, keeping the top *B* most probable sequences. This is the **Beam Search** strategy and generally performs better.