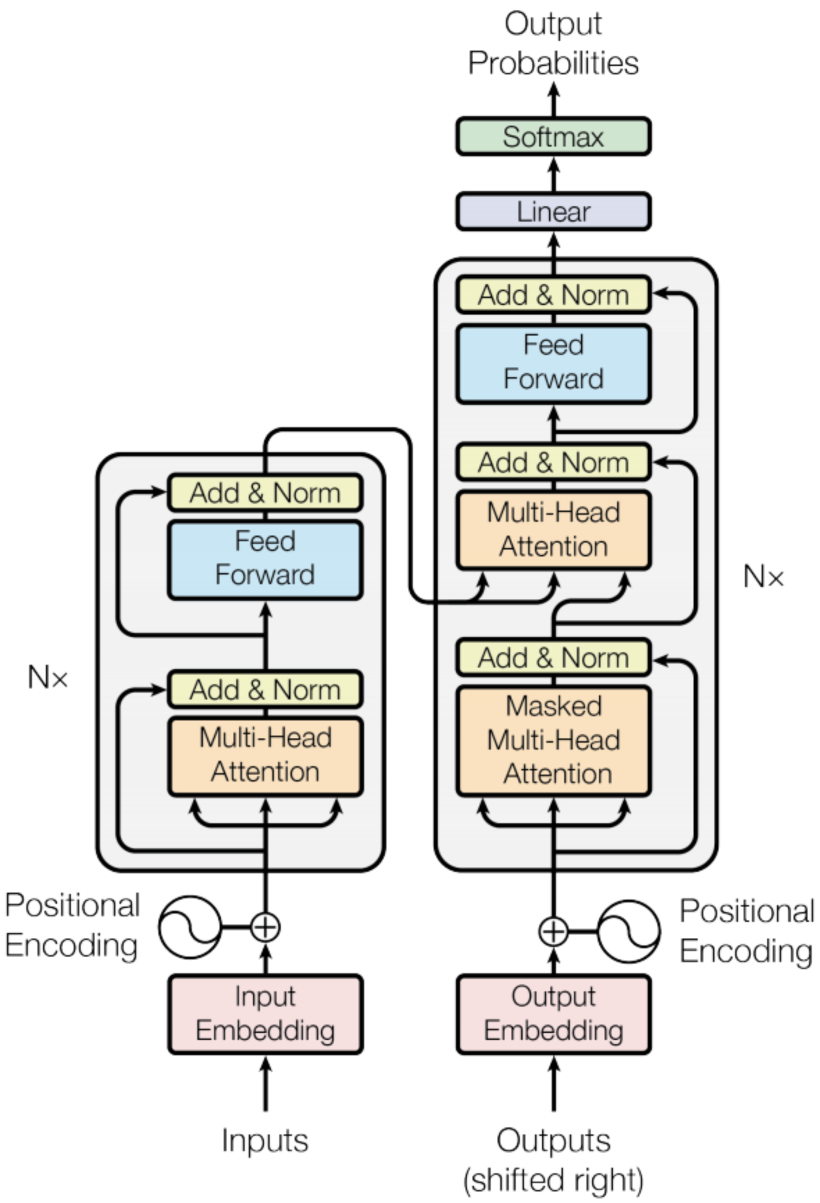
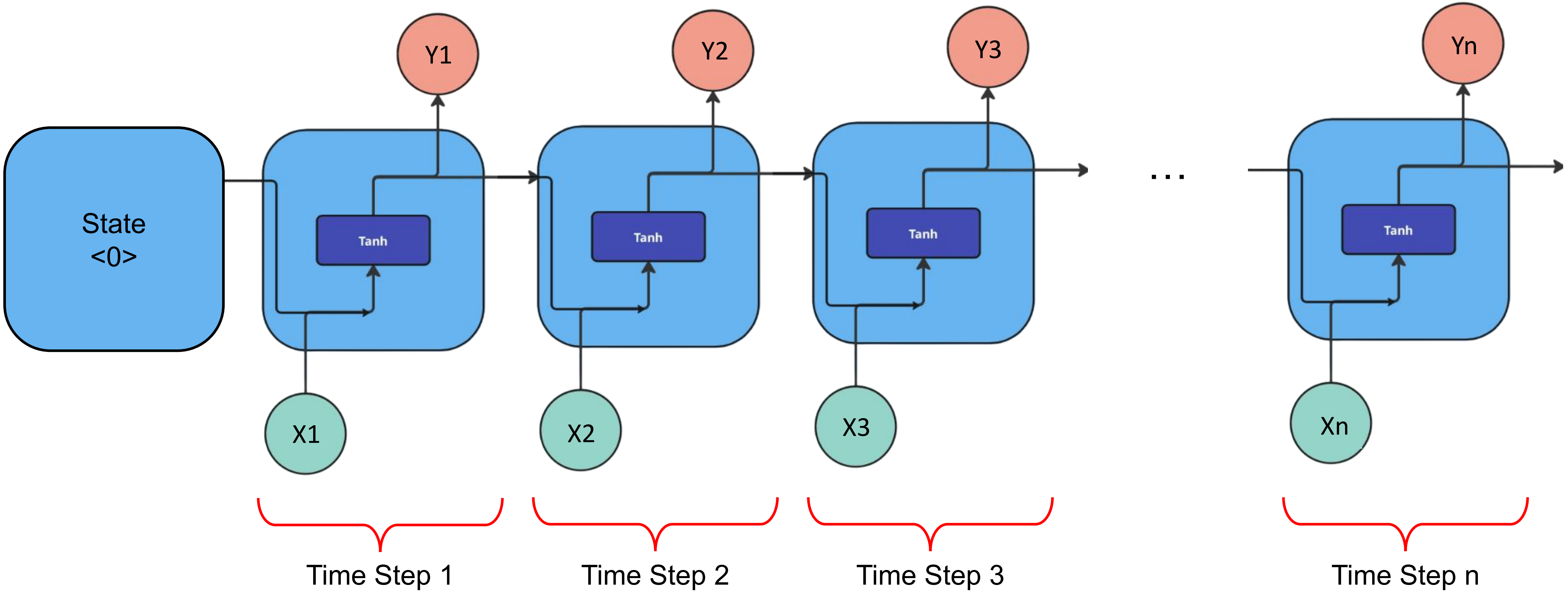


Transformers

Complete Architecture

Khushal Wadhwa

Recurrent Neural Networks



Problems

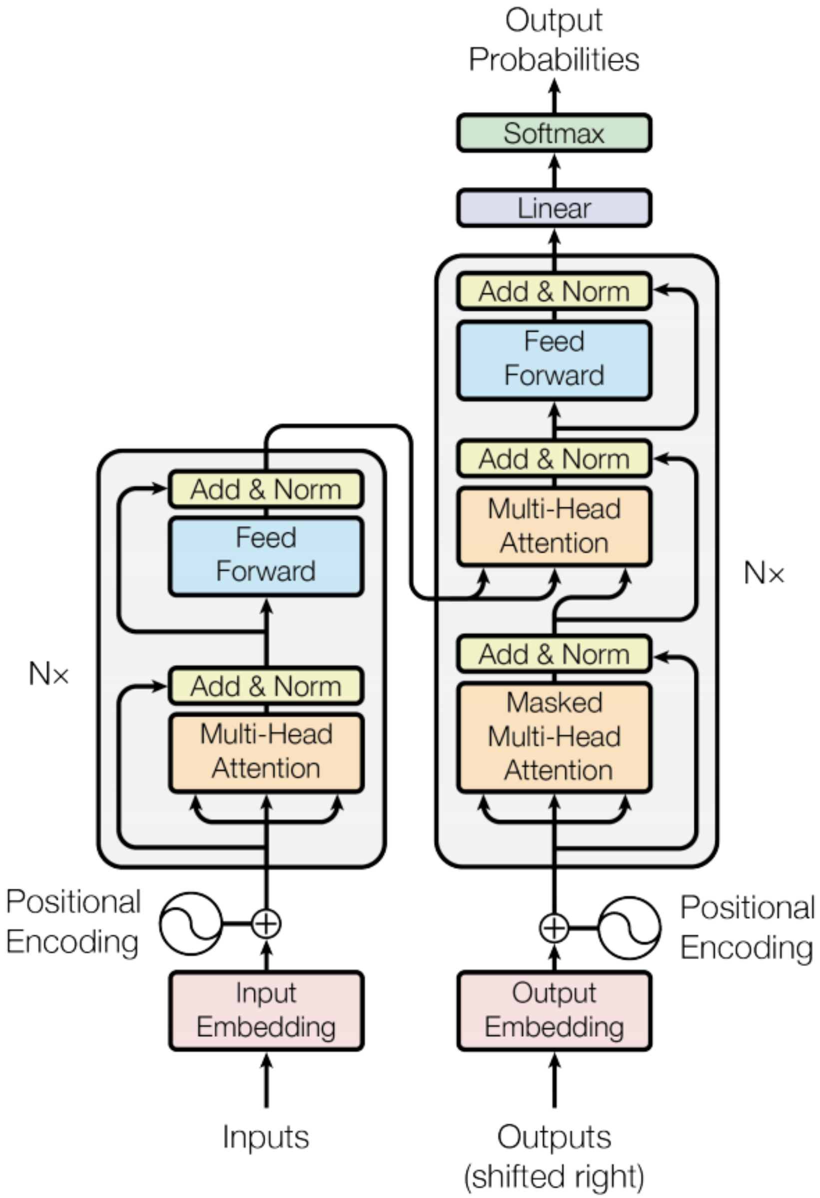
- Time steps **-** process words one by one.

  To understand word 10, you must process words 1–9 first.

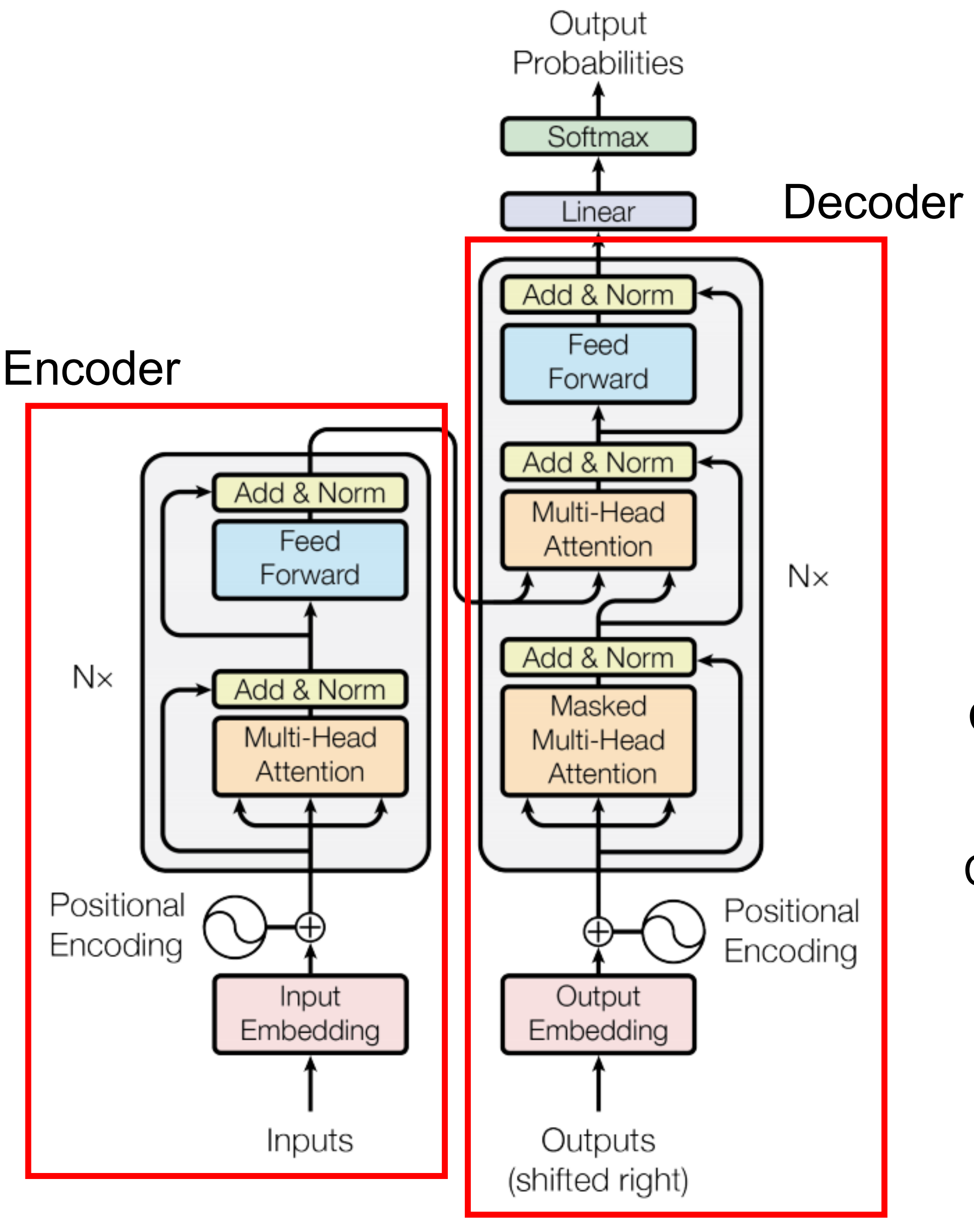
  Forgets the beginning by the time it reaches the end.

- Slow computation as can’t be parallelized.

- Vanishing/exploding gradients.



Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)



ChatGPT
Gemini
BERT

Protein Folding
(Alphafold)

Computer Vision
(ViT)

Code Generation

Attention Is All You Need

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

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

1 Introduction

Recurrent neural networks, long short-term memory [12] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [29, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [31, 21, 13].

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

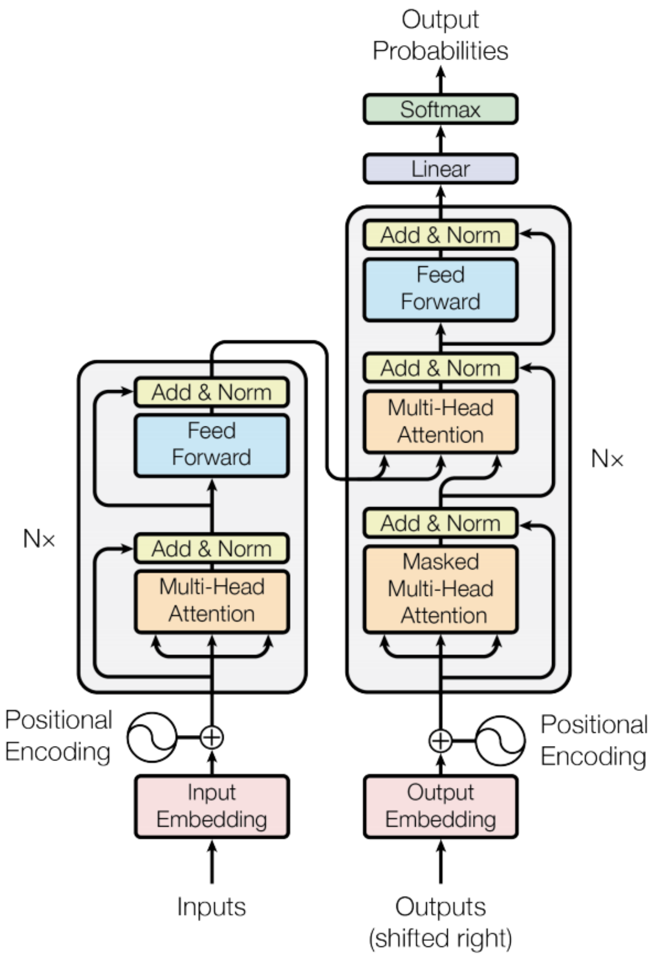
†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

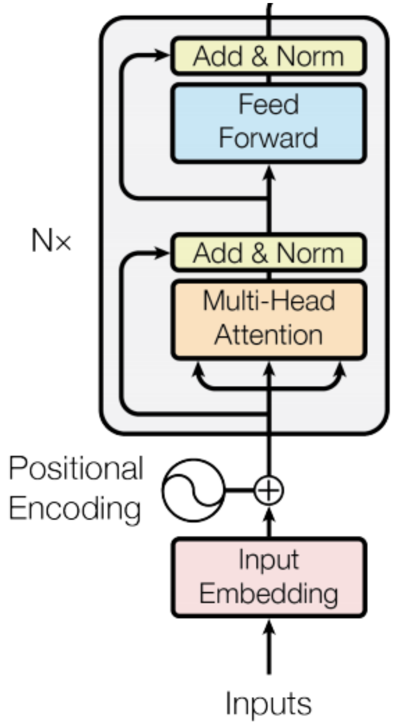
Objectives

- Encoder and Decoder
- Embeddings
- Positional Encoding
- Attention mechanisms
  Self-Attention, Multi-Head Attention, Masked Attention
- Addition & Normalization
- FFN
- Linearization

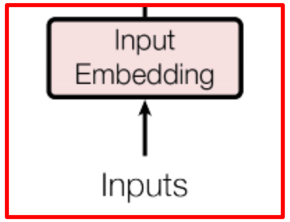


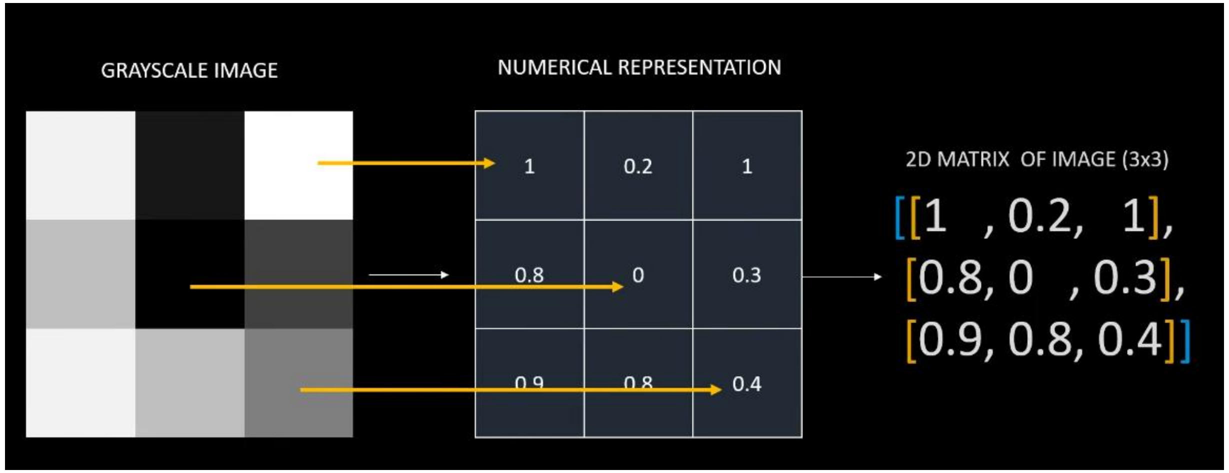
Encoder (The Reader)

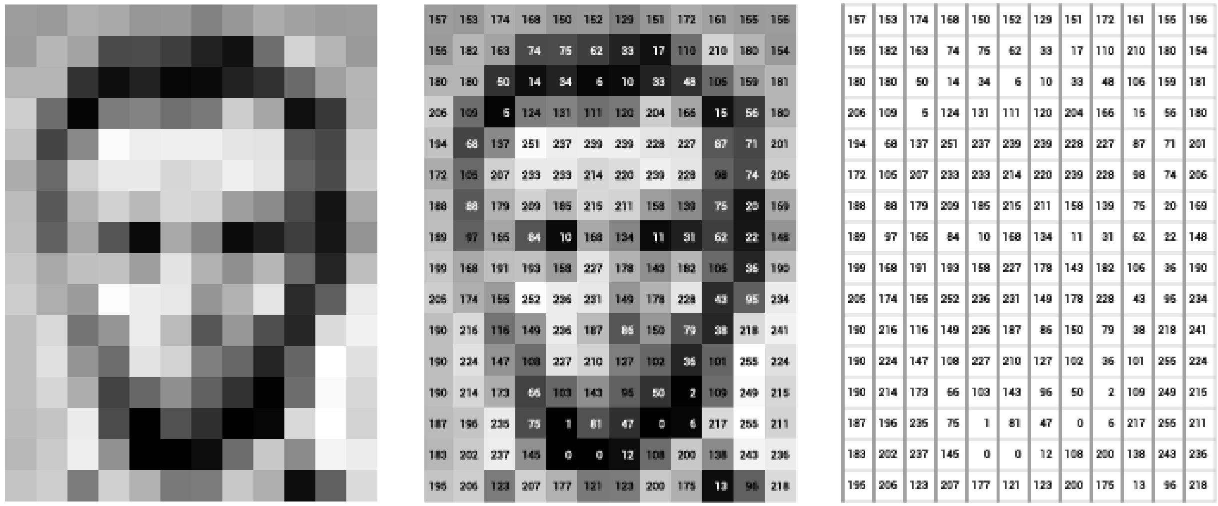
Takes the input (e.g., English) and
converts it into a rich mathematical map.



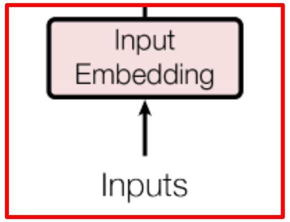
1. Input Embedding







1. Input Embedding



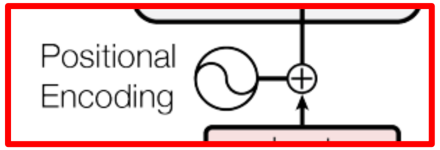
Original sentence (tokens)	YOUR	CAT	IS	A	LOVELY	CAT
Input IDs (position in the vocabulary)	105	6587	5475	3578	65	6587
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819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

Sentence → Tokenization → Vectorization

d_{model} = size of the embedding vector of each token (here 512)

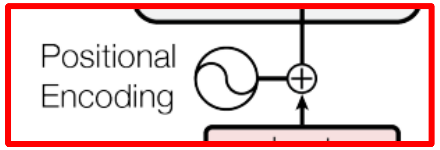
No position identifications

2. 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

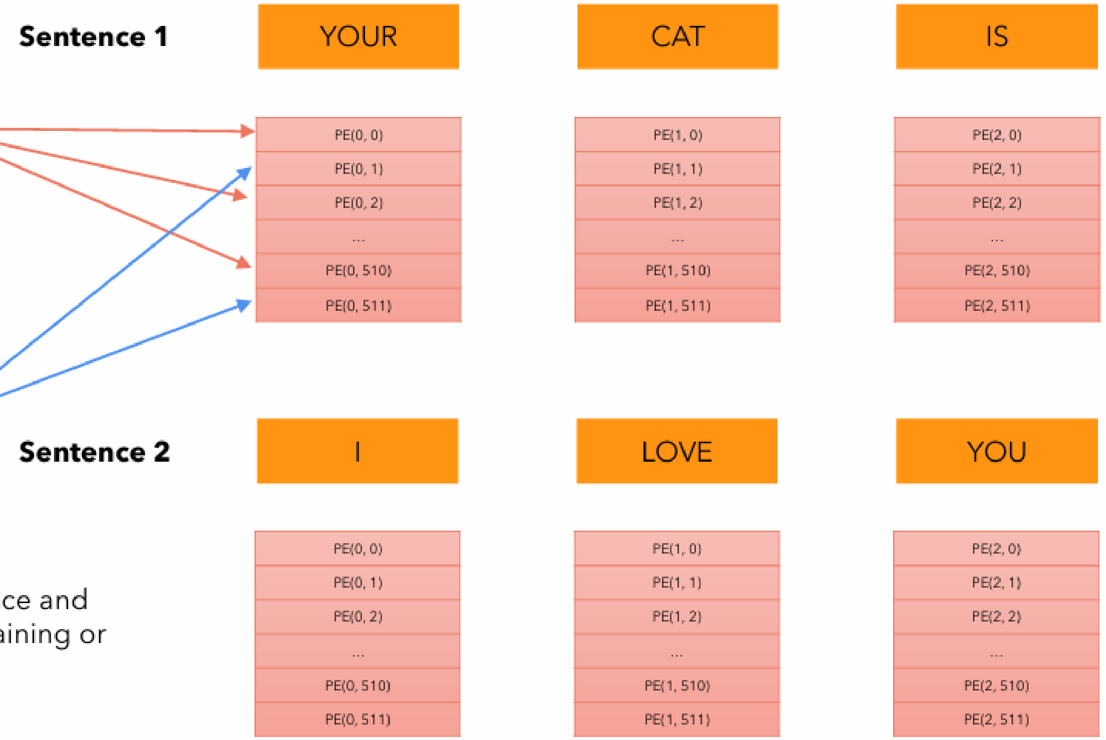
2. Positional Encoding



- Vanilla Transformers (Attention is All You Need):

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

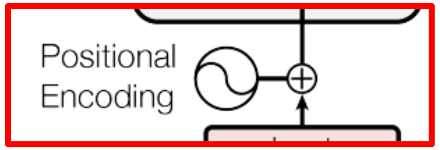
$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$



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

- Absolute Positional Embeddings:
  trainable lookup tables instead of sinusoids

2. Positional Encoding



- RoPE (Rotary Positional Embeddings):
  Most modern mechanism
  Rotates the vectors by a position-dependent angle

For a 2D slice:

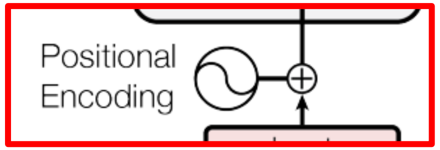
$$\begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

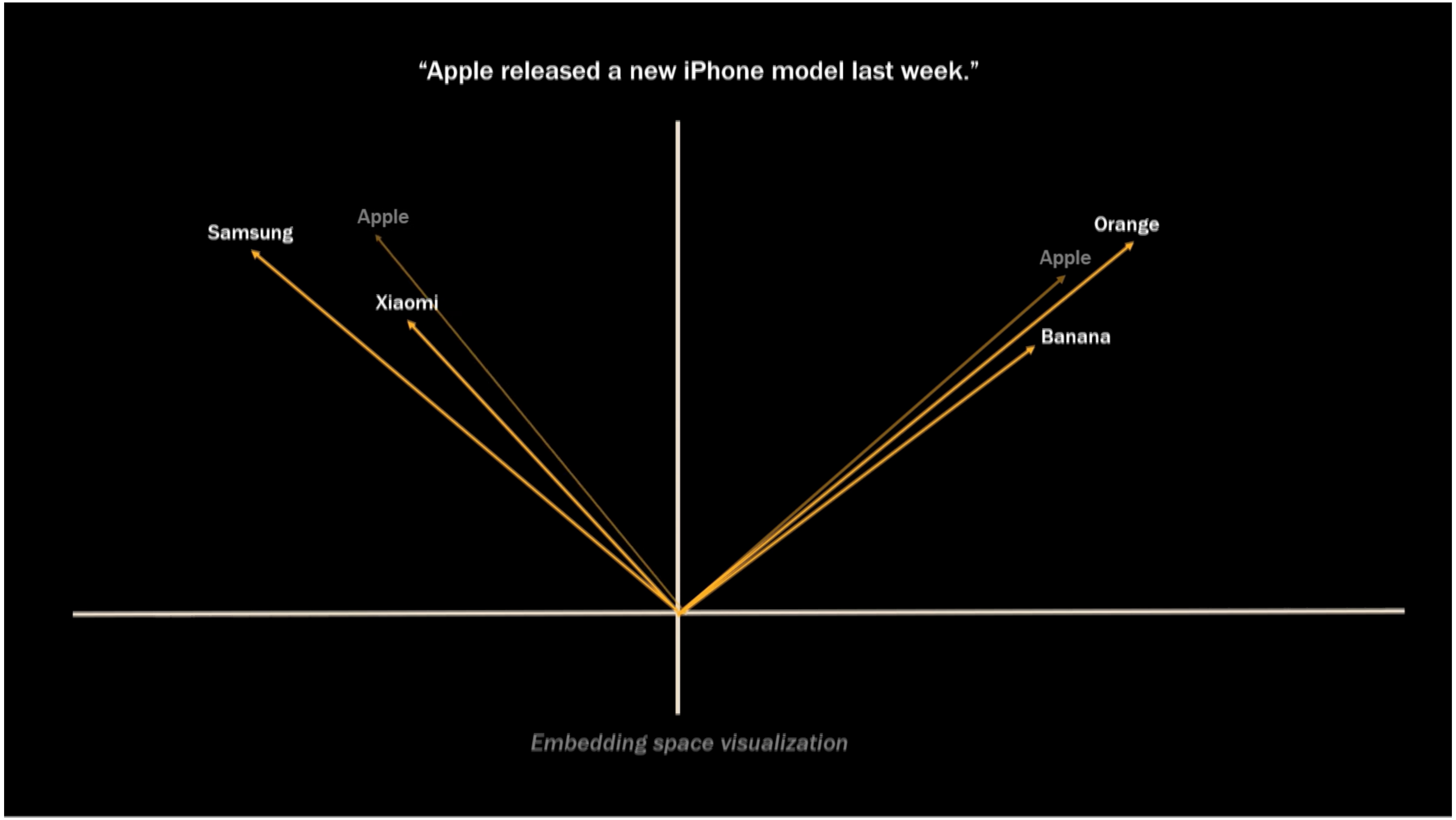
Where:

$$\theta(pos, i) = pos \cdot 10000^{-2i/d}$$

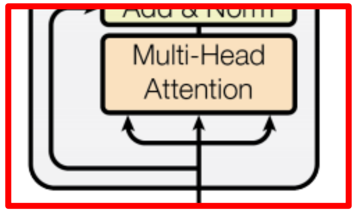
More stable attention behaviour

2. Positional Encoding





3. Attention: Self-Attention



- Allows model to relate words to each other
- **Query (Q)**: What I’m looking for (e.g., "Who is the subject?").
- **Key (K)**: The labels of all other words in the room.
- **Value (V)**: The actual content of those words.

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

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

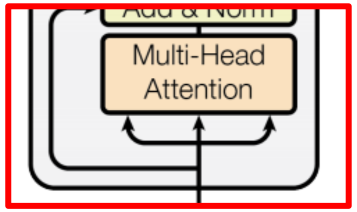
	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

(6, 6)

* all values are random.

$$\boxed{s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}}$$

3. Attention: Self-Attention



- **Query (Q)**: Tells the model what each token wants to find.
- **Key (K)**: Tells the model what each token contains.
- **Value (V)**: Provides the information used in the output.

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

	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)

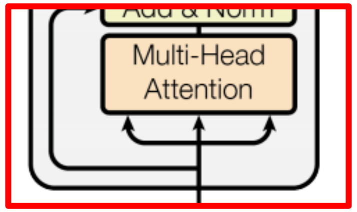
=

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.

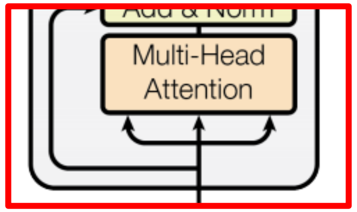
3. Attention: Self-Attention

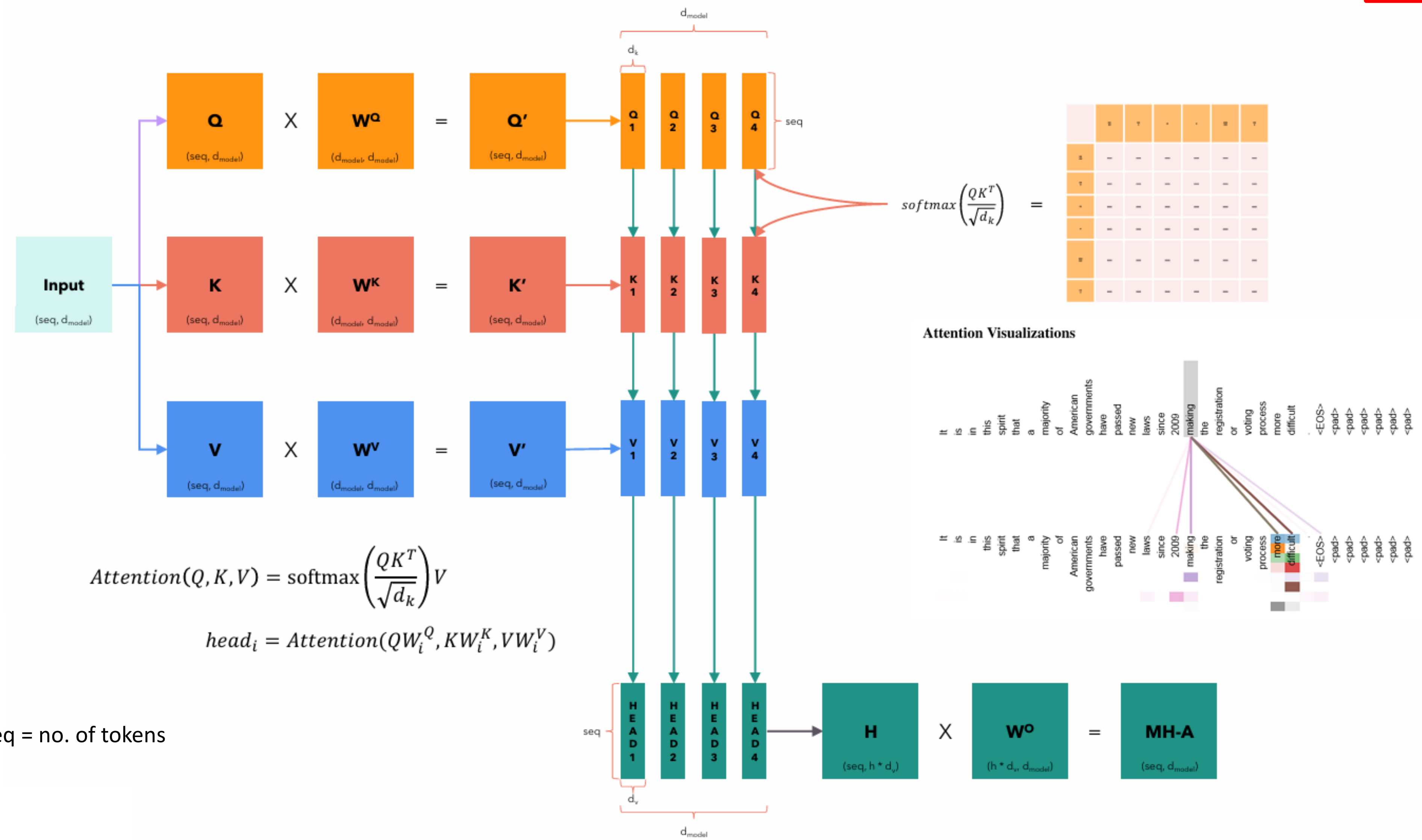


- Permutation Invariant
- We expect the values along the diagonal to be the highest
- **Interesting:** if we don't want some positions to interact, we can set their values to **$-\infty$** before applying softmax to this matrix and the model will not learn those interactions (will be useful later).

	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

3. Attention: Multi-Head Attention





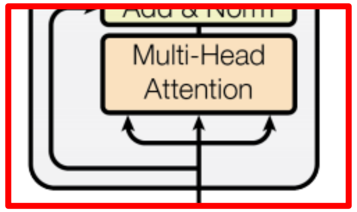
$$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) =$$

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

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

Note: seq = no. of tokens

3. Attention: Multi-Head Attention



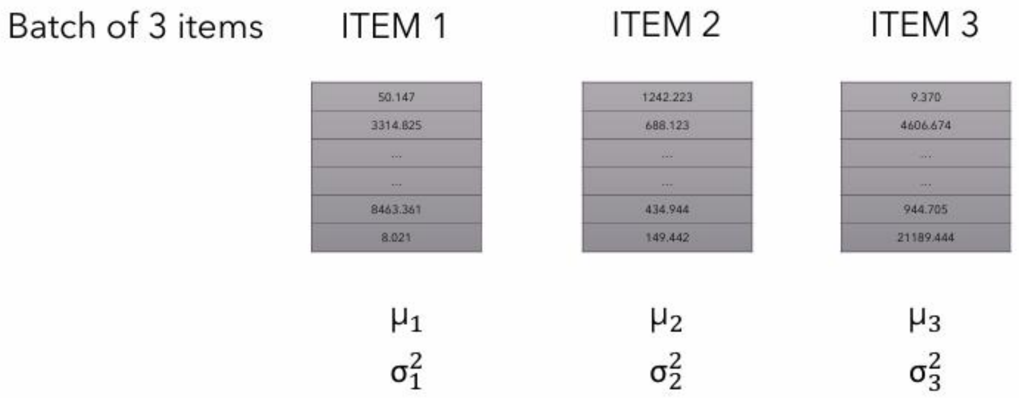
$$MultiHead(Q,K,V) = Concat(head_1 \ \dots head_h)W^O$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

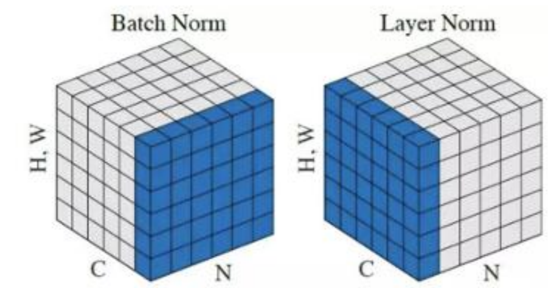
Important notations:

- d_{model}: main embedding size of model
- h: number of attention heads – each head learns its own Q, K, V projections
  Why?
  a) Each head can learn a different relationship, for example:
     Head 1 | Finds subjects
     Head 2 | Tracks verbs
     Head 3 | Tracks long-range dependencies
     Head 4 | Pays attention to punctuation
  b) Parallel power: each head runs independently on the GPU, then they are concatenated together.
- d_{head}: dimension of each attention head – each head operates on a lower-dimensional subspace
     $d_{head} = d_{model} / h$

4. Add & Norm

- Add (Residual Connection)
  Add the original input x back to the sublayer output y
  z = x + y
- Norm (Layer Normalization)
  Normalize the summed vector to stabilize training





$$\hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

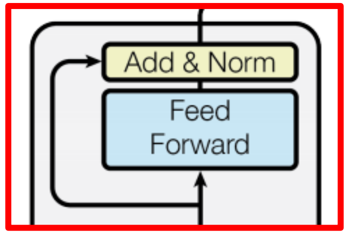
We also introduce two parameters, usually called **gamma** (multiplicative) and **beta** (additive) that introduce some fluctuations in the data, because maybe having all values between 0 and 1 may be too restrictive for the network. The network will learn to tune these two parameters to introduce fluctuations when necessary.

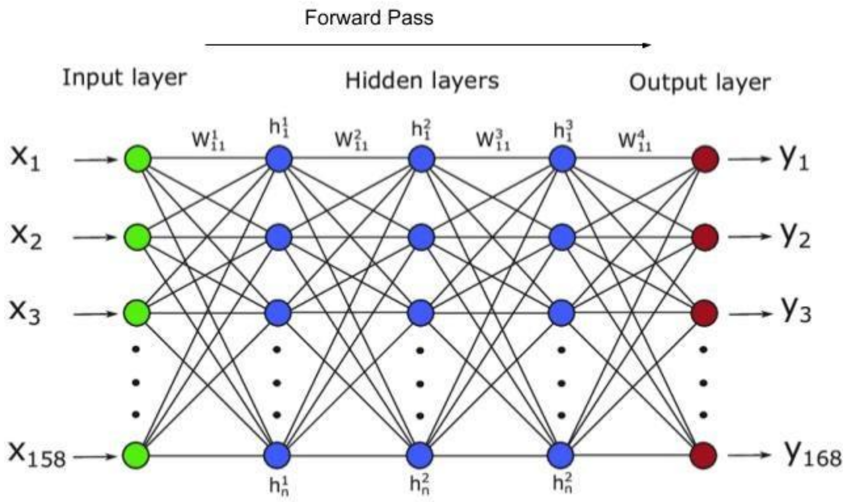
In the original 2017 Transformer:
LayerNorm comes **after** Add
- Called **Post-LN**

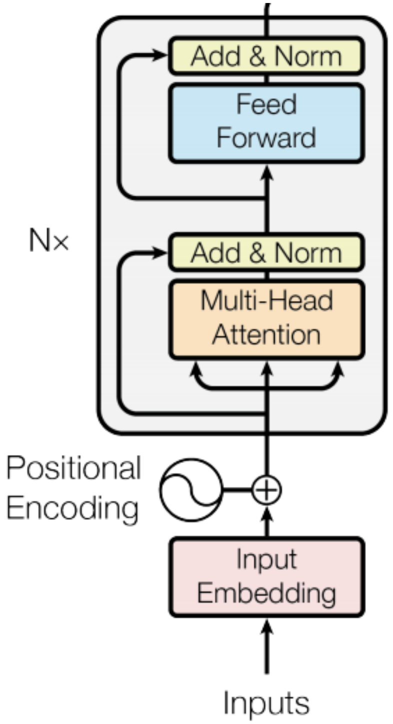
Modern LLMs use:
LayerNorm **before** sublayers
- Called **Pre-LN**

Why?
- Pre-LN is more stable for deep transformers (40+ layers).

5. Feed Forward Network



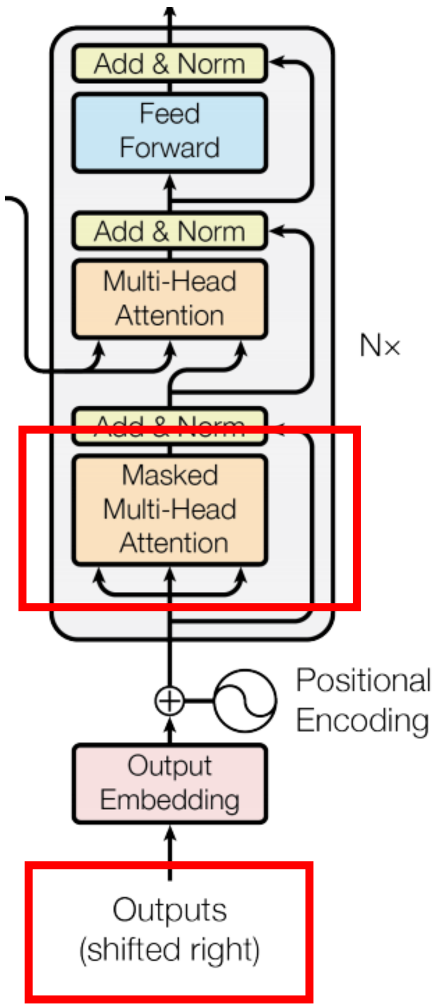




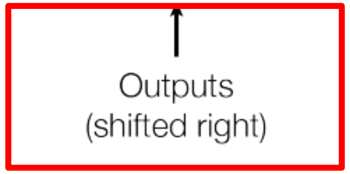
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs

Decoder (The Writer)

Uses the encoder map to generate output (e.g., French) one token at a time.



6. Right shifted outputs



The grass is green <EOS>

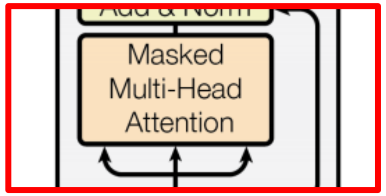
घास हरी है <EOS>

<s> घास हरी है

- <s>
- <SOS>
- <CLS>

The start-token <s> is prepended, and every target token is moved one step forward.

7. Attention: Masked Multi-Head

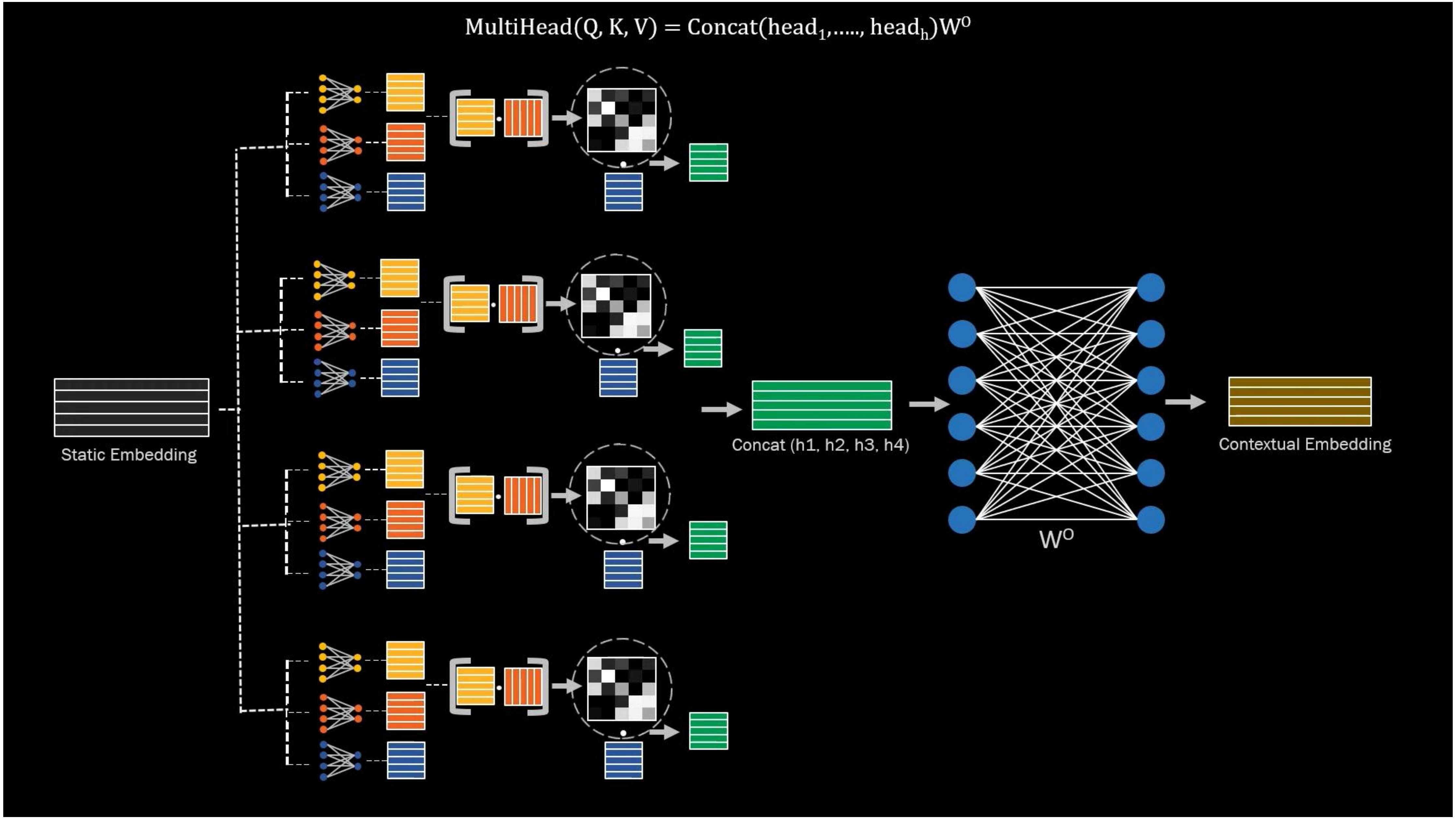


- Goal is to make the model causal – the output at certain positions can only depend on the words in the previous positions. It must not be able to see future words.
- Model is blocked from seeing the future words. How?

	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

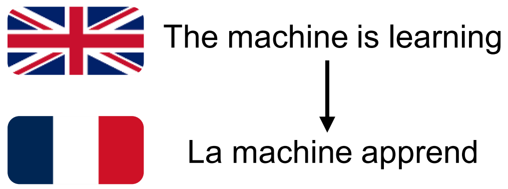
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
		$-\infty$	$-\infty$	$-\infty$	$-\infty$
			$-\infty$	$-\infty$	$-\infty$
				$-\infty$	$-\infty$
					$-\infty$

	0	0	0	0	0
		0	0	0	0
			0	0	0
				0	0
					0

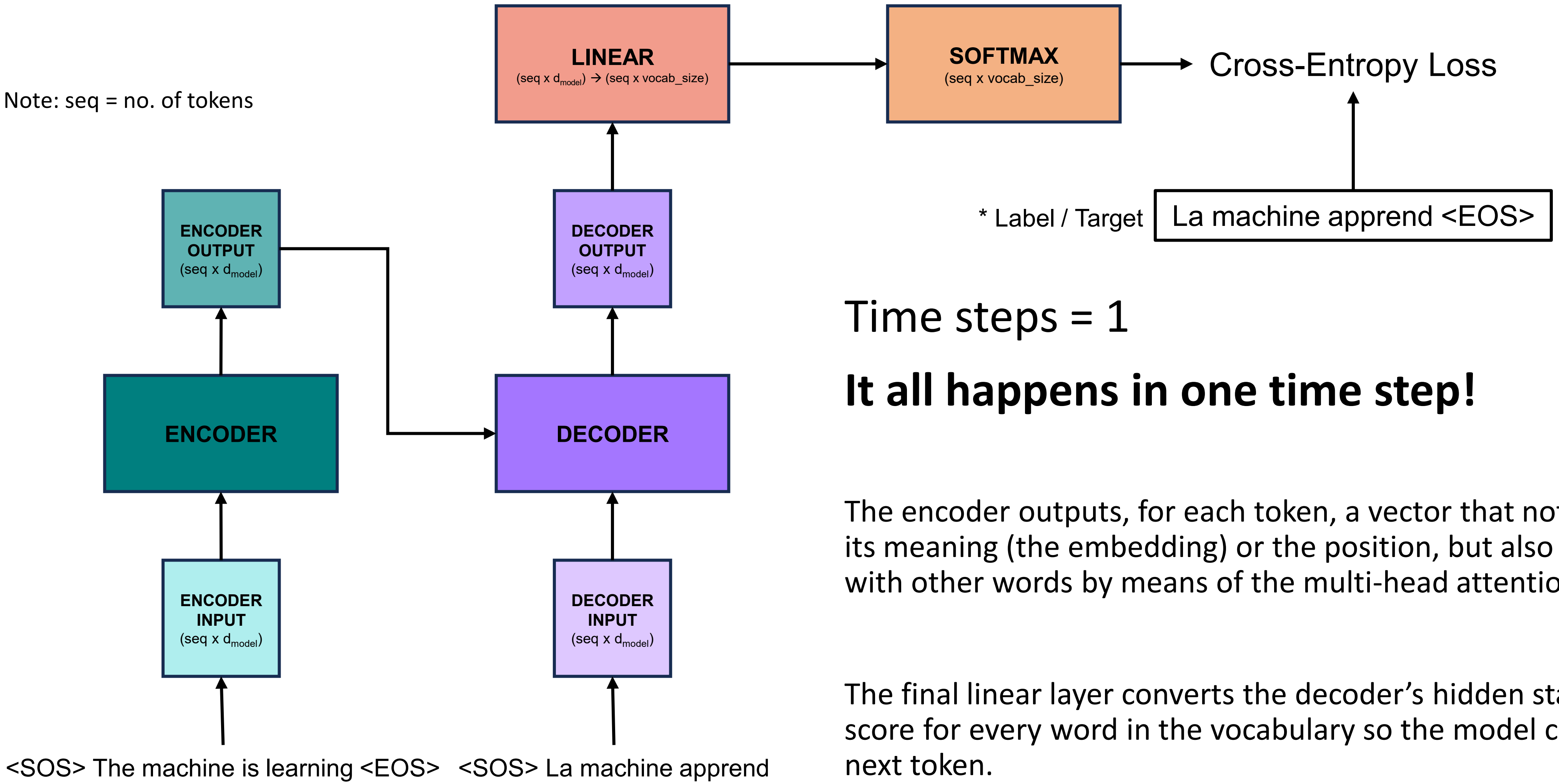


MultiHead(Q, K, V) = Concat(head1,....., headh)W^O
Static Embedding
Concat (h1, h2, h3, h4)
W^O
Contextual Embedding

Training the Transformer Model



Note: seq = no. of tokens



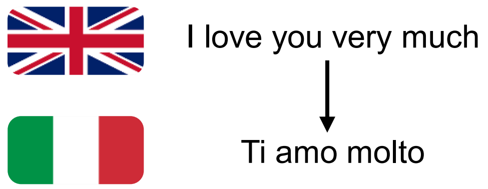
Time steps = 1

It all happens in one time step!

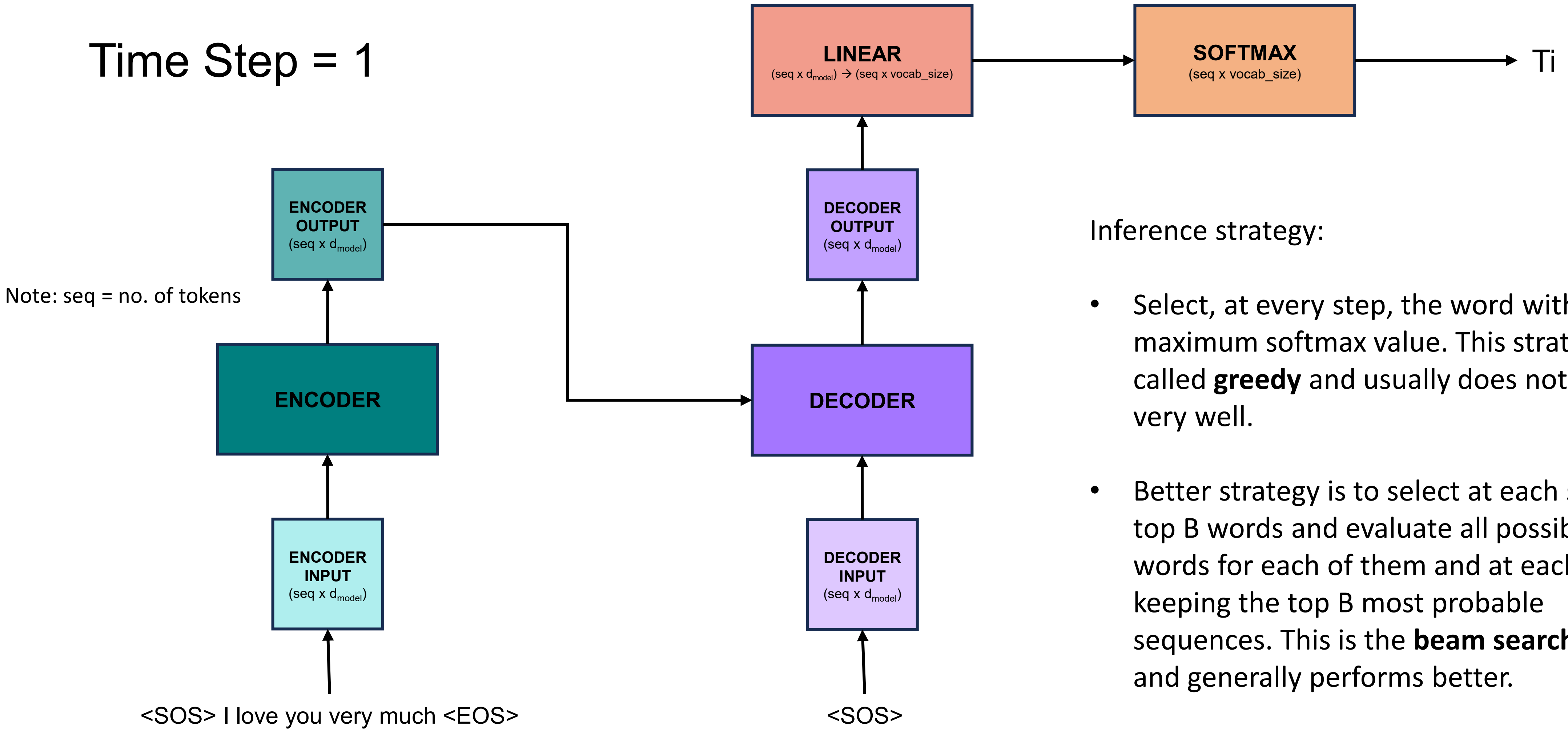
The encoder outputs, for each token, a vector that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.

The final linear layer converts the decoder's hidden state into a score for every word in the vocabulary so the model can choose the next token.

Transformer Model Inference



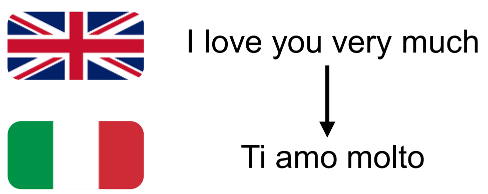
Time Step = 1



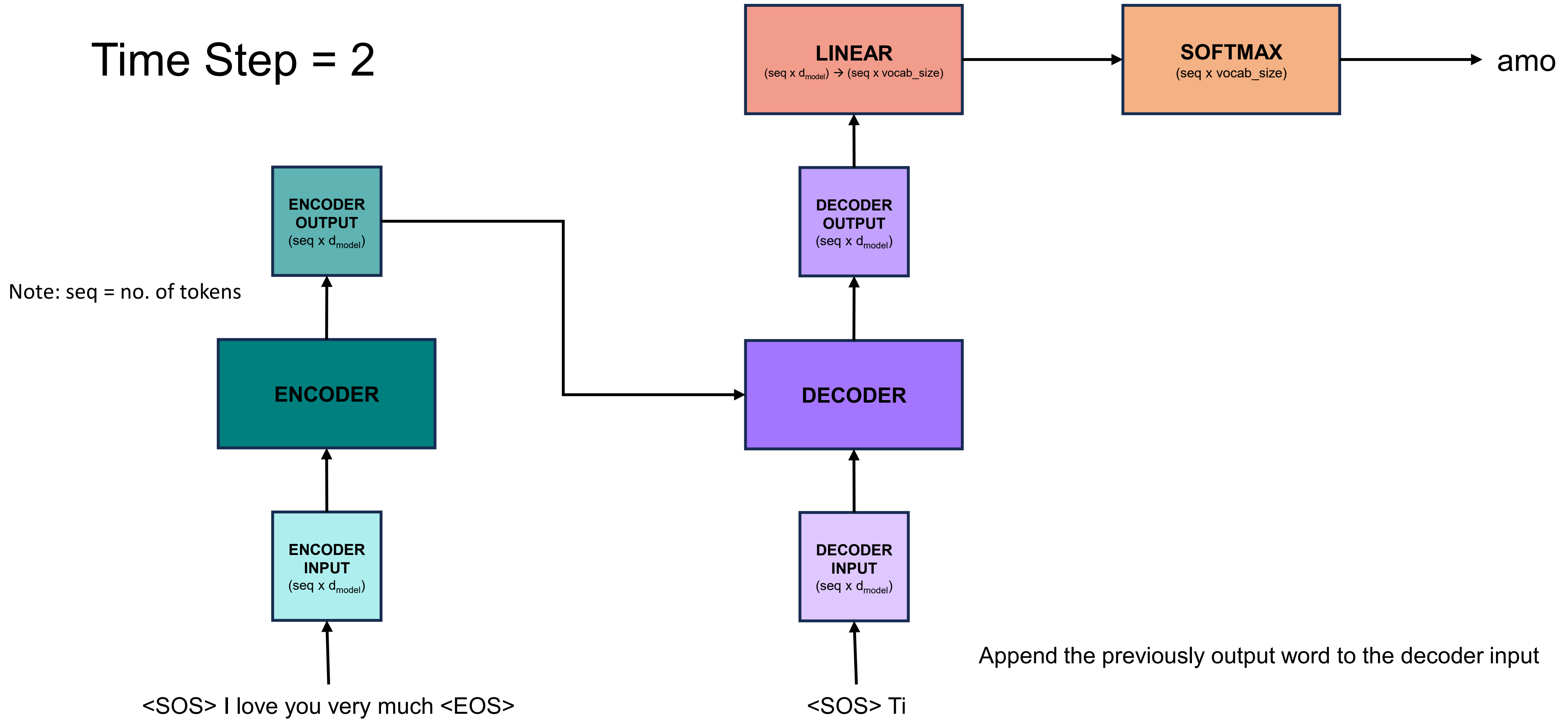
Inference strategy:

- Select, at every step, the word with the maximum softmax value. This strategy is called **greedy** and usually does not perform very well.
- Better strategy is to select at each step the top B words and evaluate all 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.

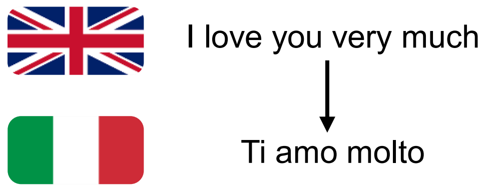
Transformer Model Inference



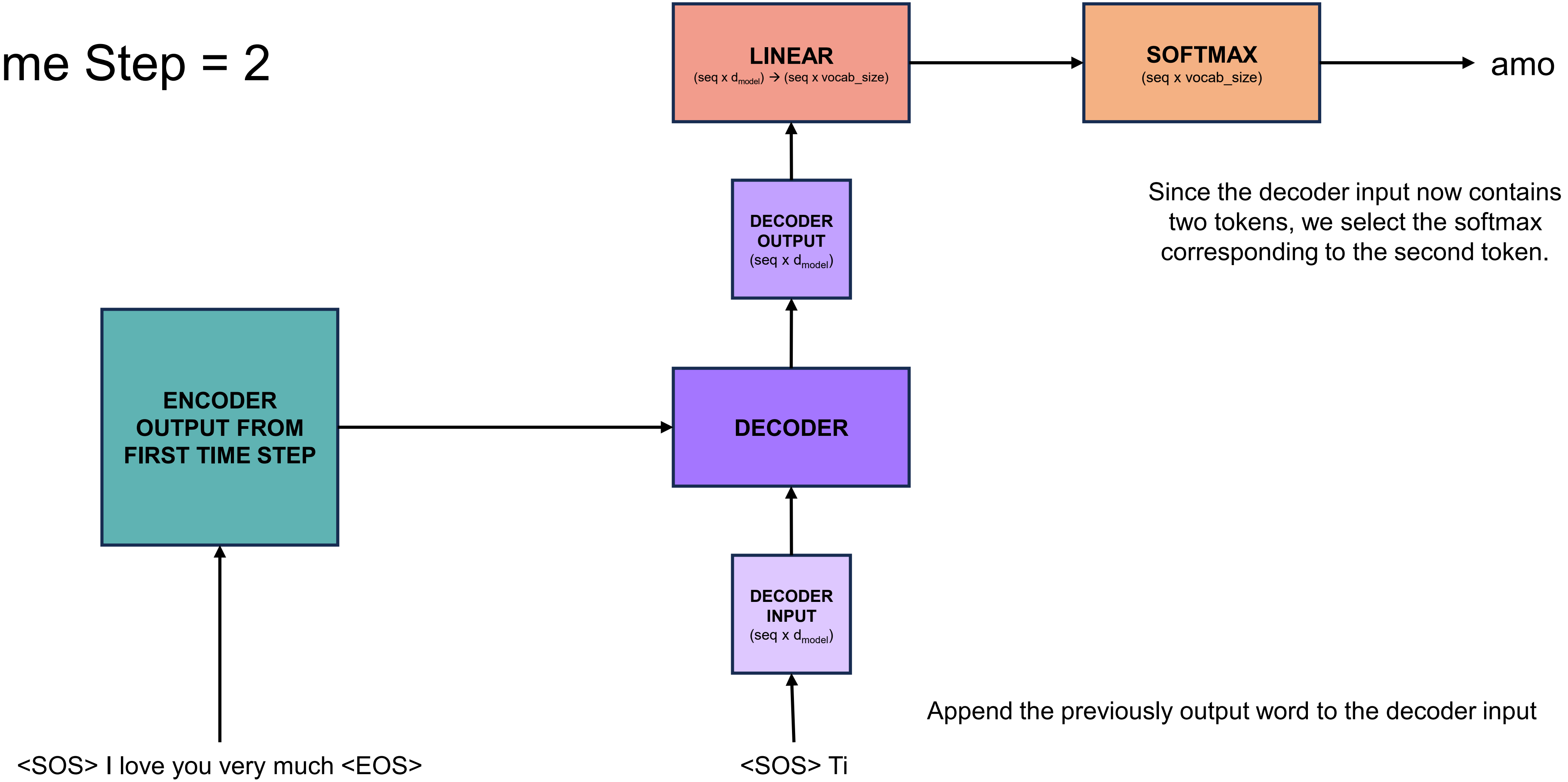
Time Step = 2



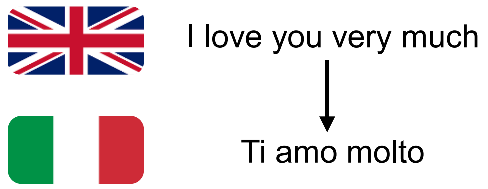
Transformer Model Inference



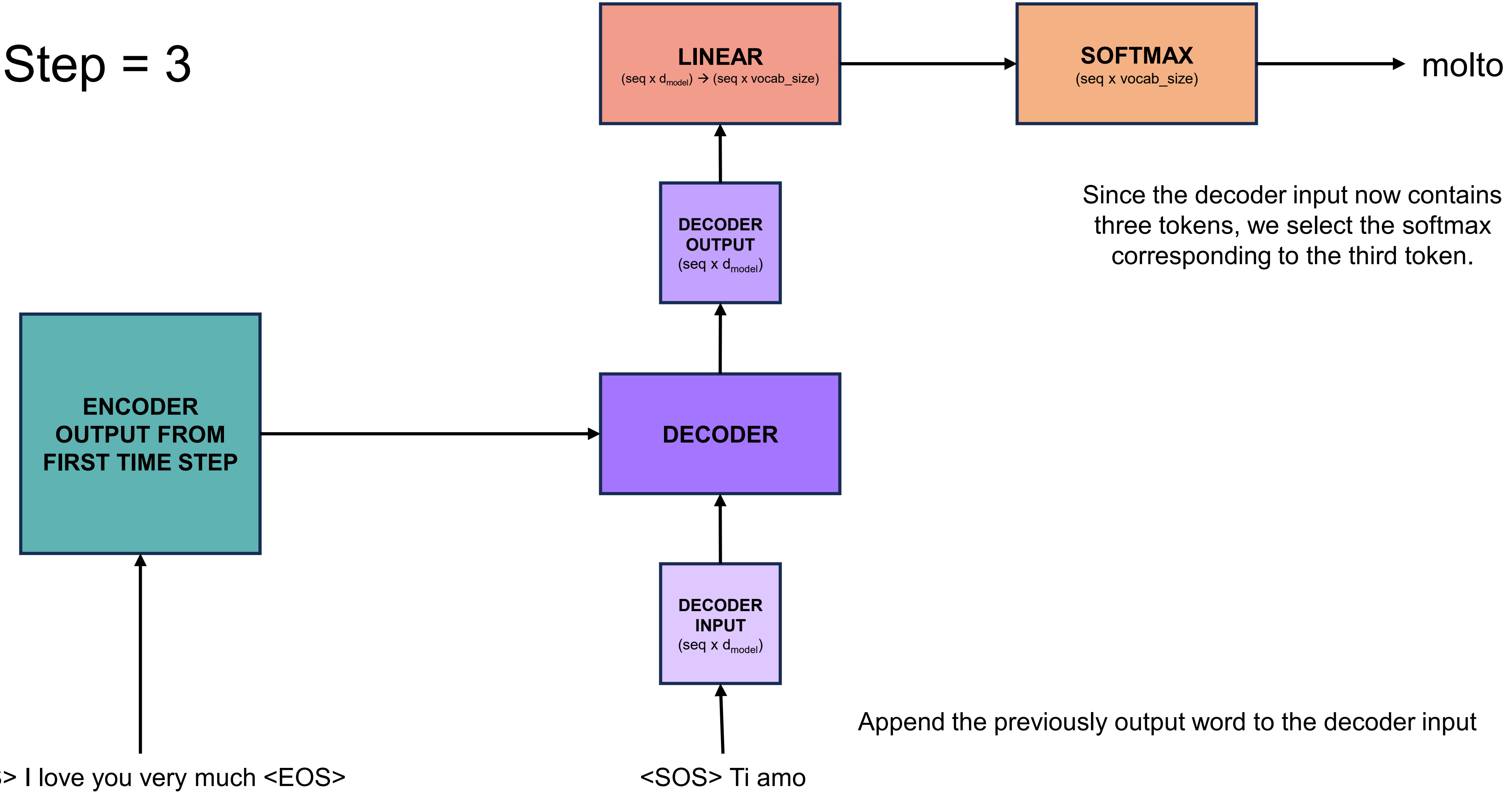
Time Step = 2



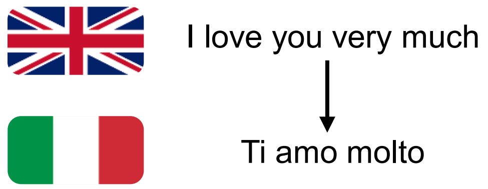
Transformer Model Inference



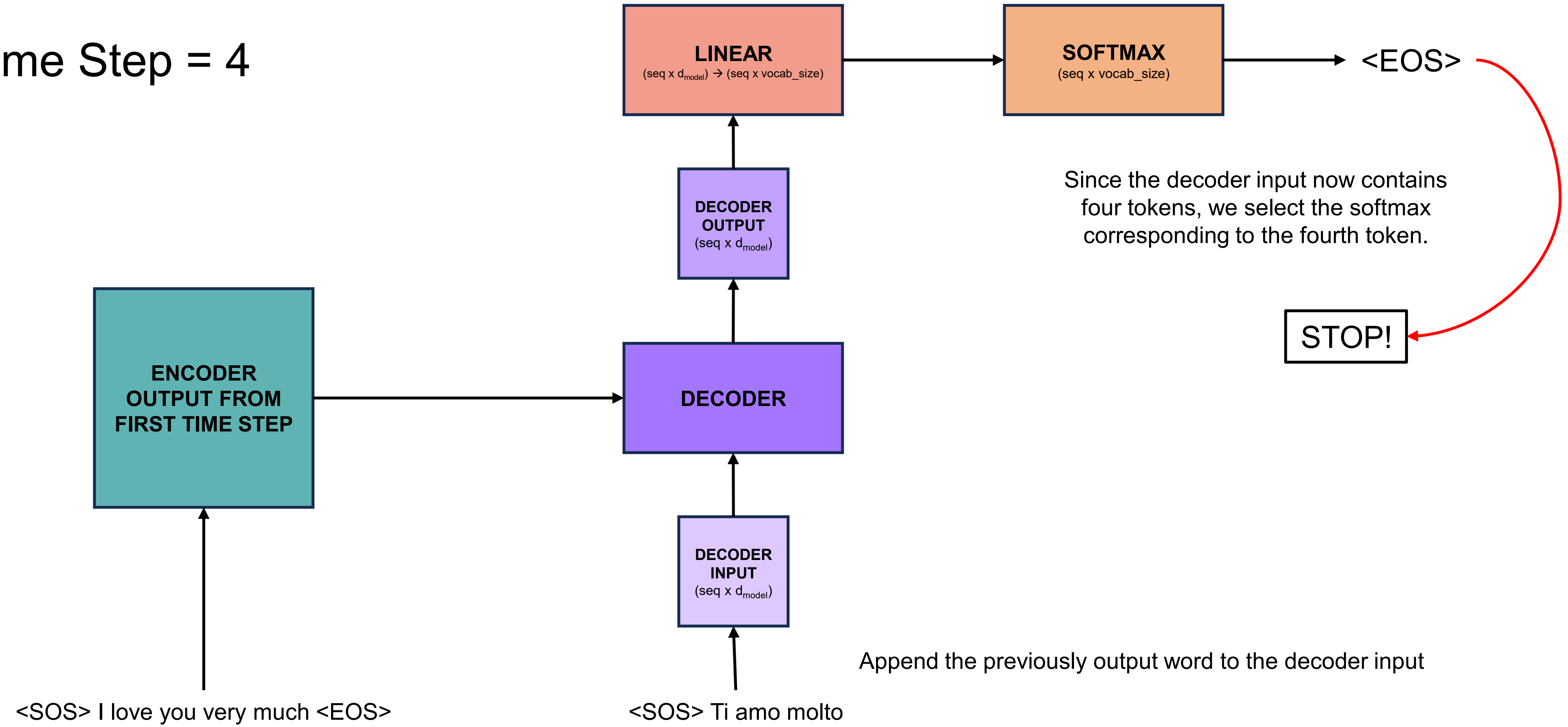
Time Step = 3



Transformer Model Inference



Time Step = 4



Real Models

1. BERT (2018) – *Encoder-only model*
   - Uses 12-24 encoder layers
   - Bidirectional self-attention
   - Absolute positional embeddings
   - Used for: classification, QA, semantic search

   ➢ BERT Base
   - Embedding size (d_{model}) = 768
   - 12 encoder layers
   - 12 attention heads per layer
   - Total heads = 12 x 12 = 144 heads across the model
   - Head dimension (d_{head}) = 768 / 12 = 64

   ➢ BERT Large
   - Embedding size (d_{model}) = 1024
   - 24 encoder layers
   - 16 attention heads per layer
   - Total heads = 24 x 16 = 384 heads across the model
   - Head dimension (d_{head}) = 1024 / 16 = 64

2. GPT-2 / 3 (2019-20) – *Decoder-only model*
   - Stack of masked self-attention decoder blocks
   - Uses learned absolute positional embeddings
   - Used for: text generation, reasoning, creative tasks

   ➢ GPT-2 Small (117M trainable parameters)
   - Embedding size (d_{model}) = 768
   - 12 decoder layers
   - 12 attention heads per layer
   - Total heads = 12 x 12 = 144 heads across the model
   - Head dimension (d_{head}) = 768 / 12 = 64

   ➢ GPT-3 175B (175B trainable parameters)
   - Embedding size (d_{model}) = 12288
   - 96 encoder layers
   - 96 attention heads per layer
   - Total heads = 96 x 96 = 9216 heads across the model
   - Head dimension (d_{head}) = 12288 / 96 = 128

Real Models

3. T5 (2018) – *Encoder-Decoder (Seq2Seq)*
   - Converts every task into text → text
   - Relative position bias instead of absolute embeddings
   - Excellent for: translation, summarization, rewriting
   - ➢ T5 Small
     - Embedding size (d_{model}) = 512
     - 6 encoder layers
     - 6 decoder layers
     - 8 attention heads per layer
     - Total heads = 12 x 8 = 96 heads across the model
     - Head dimension (d_{head}) = 512 / 8 = 64
   - ➢ T5 11B
     - Embedding size (d_{model}) = 4096
     - 24 encoder layers
     - 24 decoder layers
     - 128 attention heads per layer
     - Total heads = 48 x 128 = 6144 heads across the model
     - Head dimension (d_{head}) = 4096 / 128 = 32

4. BART (2019) – *Denoising Autoencoder (Seq2Seq)*
   - BERT-like encoder + GPT-like decoder
   - Trained by corrupting text and reconstructing it
   - Good for: summarization, sequence generation

5. LLaMA (2023) & LLaMA-2 / 3 (2024-25) – *Decoder-only model*
   - Modern open-source LLMs
   - RoPE for long context
   - Uses SwiGLU feed-forward networks
   - Highly efficient & strong at reasoning

Real Models

6. Mistral / Mixtral (2023-24)
   - Decoder-only with grouped-query attention (faster inference)
   - Uses Mixture-of-Experts (MoE) in Mixtral
   - State-of-the-art (SOTA) open models with excellent scaling

7. Vision Transformers (ViT, 2020) – *Transformers for Images*
   - Split images into fixed-size patches
   - Apply standard transformer encoder
   - This shows transformers generalize beyond text

Sources

- Transformer from Scratch, Umar Jamil
  https://github.com/hkproj/transformer-from-scratch-notes

- How Attention Mechanism Works in Transformer Architecture, Under The Hood
  https://www.youtube.com/watch?v=KMHkbXzHn7s

- Attention is All You Need, Vaswani et al.
  https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf