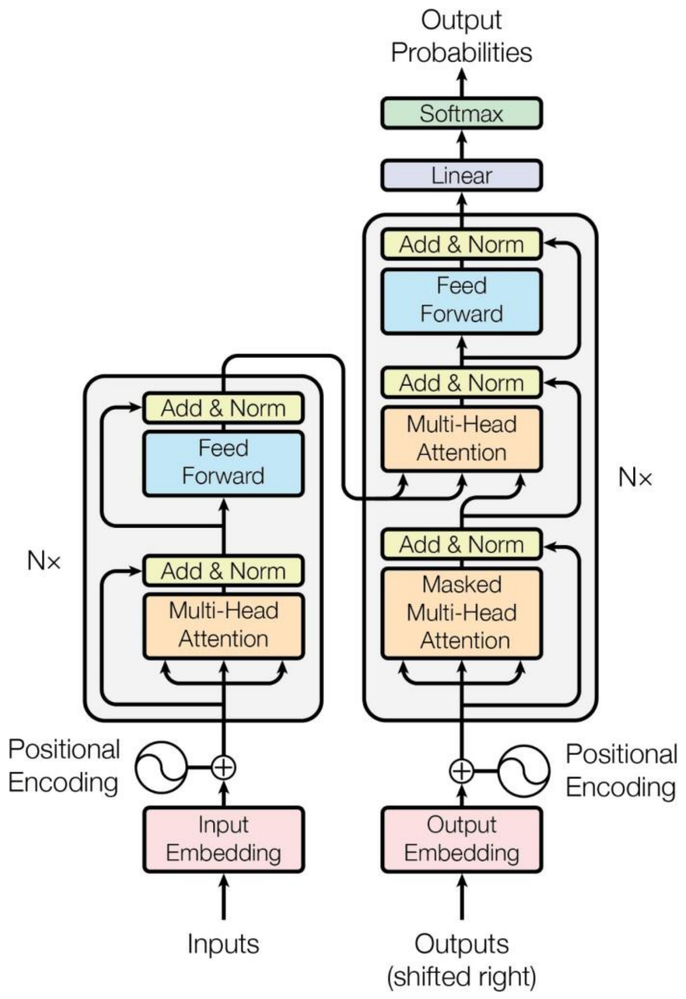
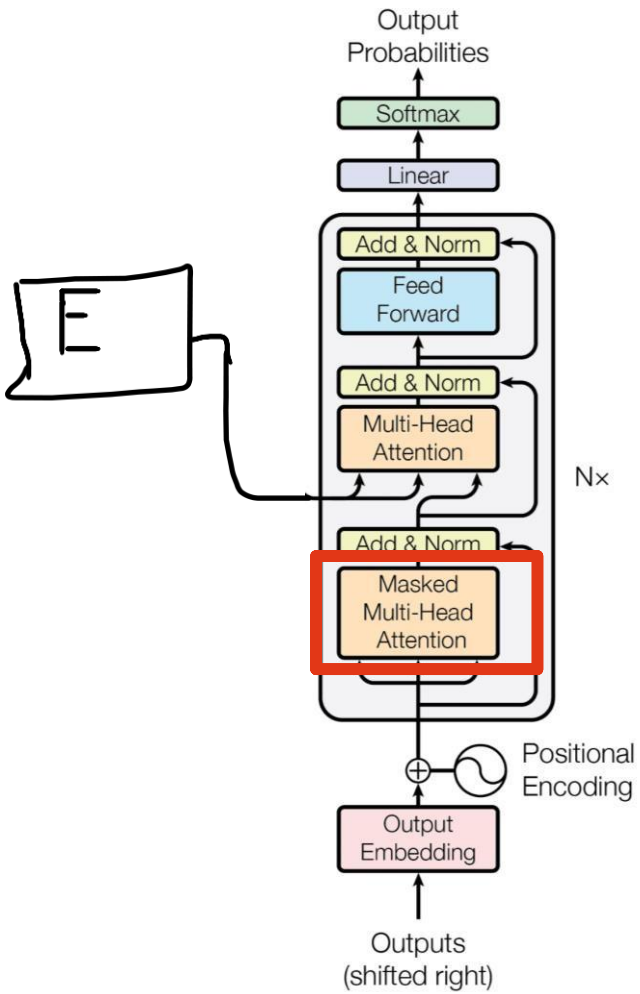


Transformers

Paper: Attention is all you need

Decoder

Cross attention: keys, values from encoder
Queries from decoder

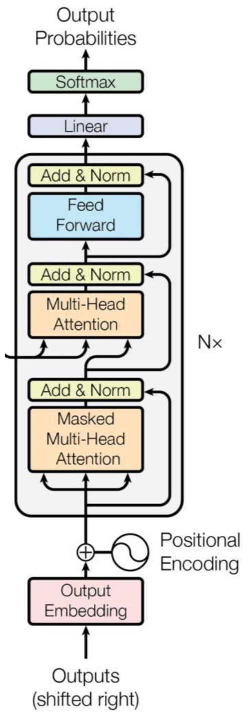


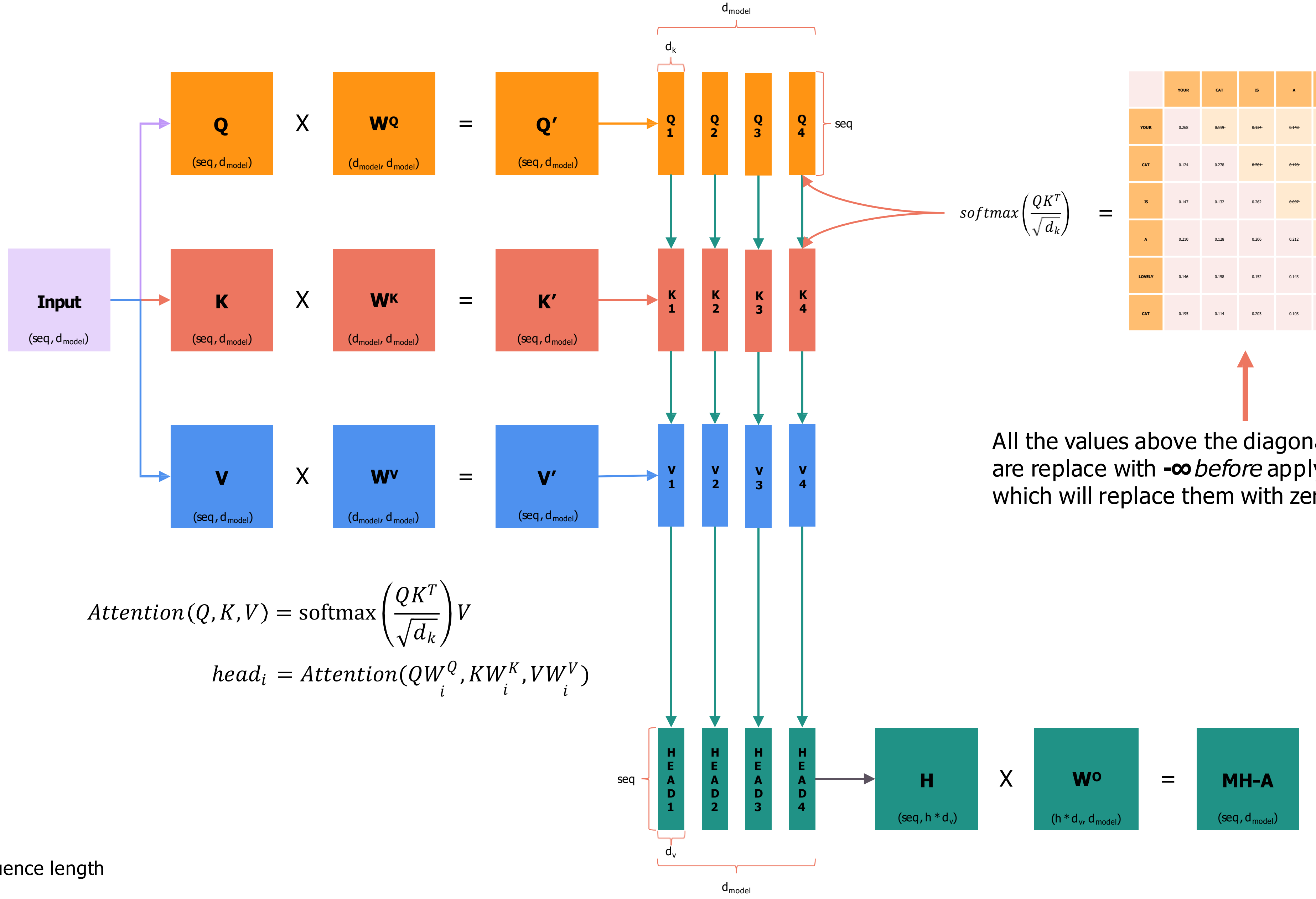
Cross attention:
The Query matrix is used to attend to the Key matrix, and the output is weighted by the Value matrix. This process allows the decoder to "ask" the encoder about specific parts of the source language input, and use that information to generate the target language output.
The cross-attention 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





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

	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.

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

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

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

seq = sequence length

d_{model} = size of the embedding vector

h = number of heads

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

Inference and training of a Transformer model

Training

I love you very much

Ti amo molto

Training

Time Step = 1

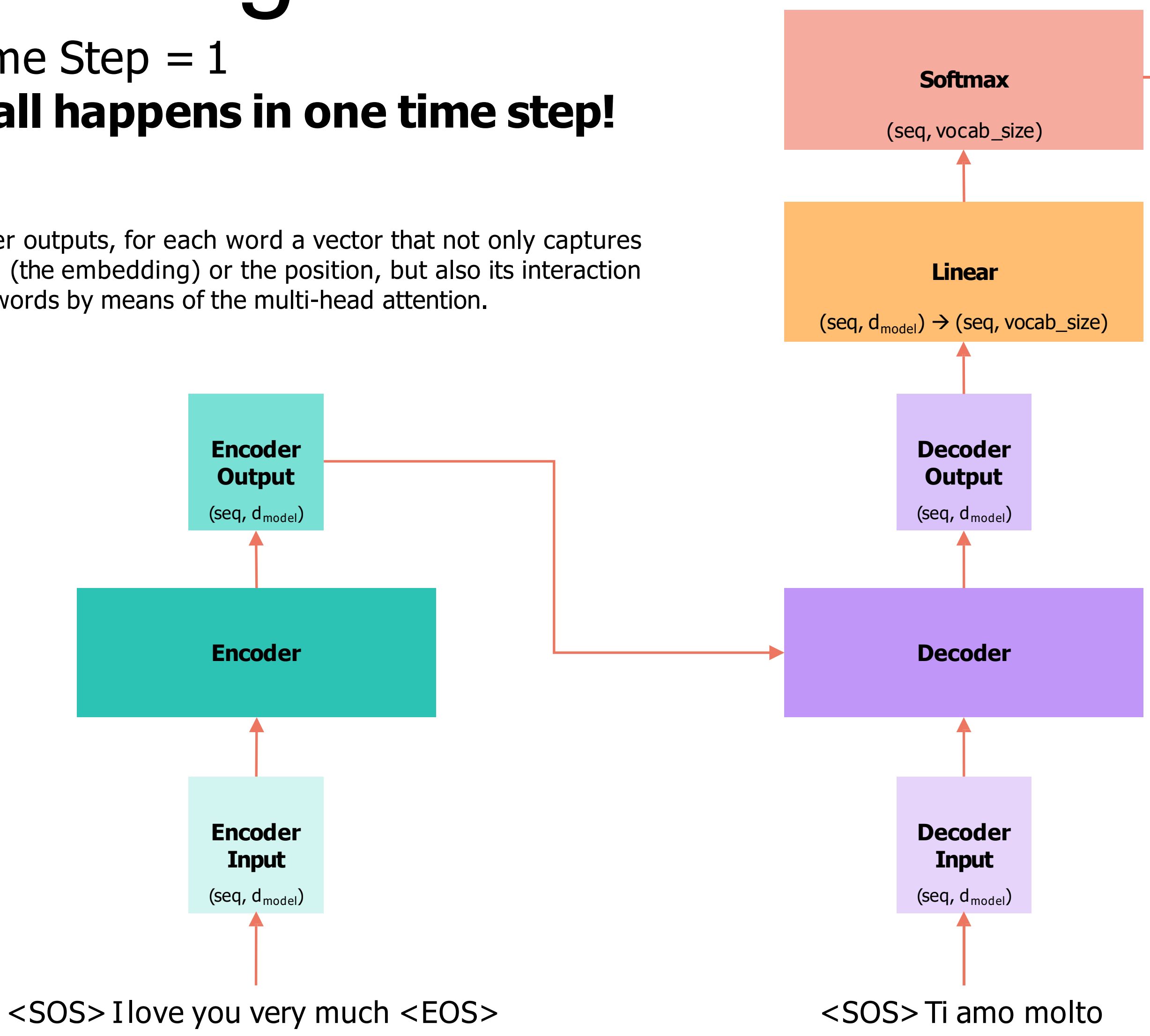
It all happens in one time step!

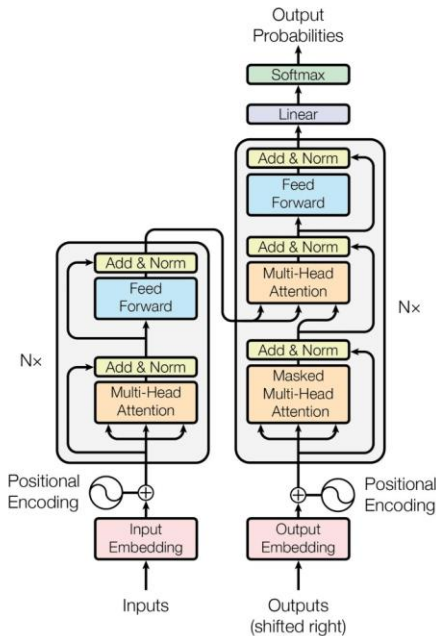
Ti amo molto <EOS>

* 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.

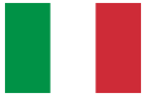




We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

Inference

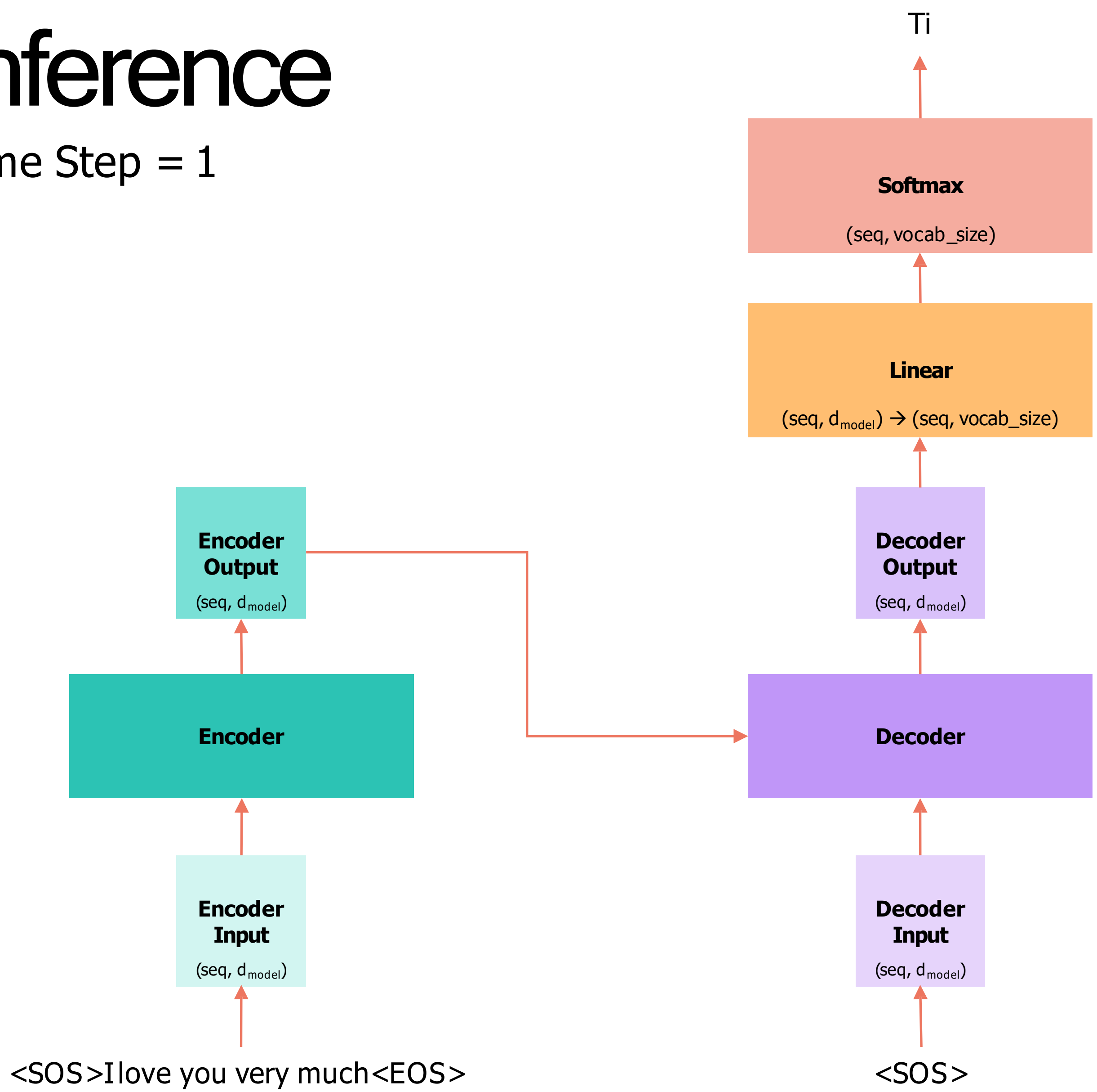
I love you very much



Ti amo molto

Inference

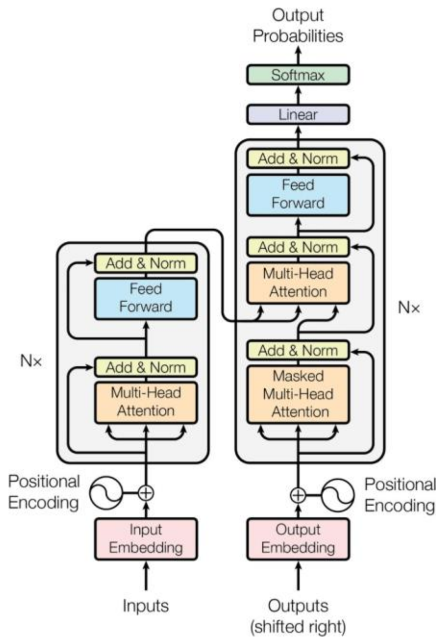
Time Step = 1



- 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

Decoder Input

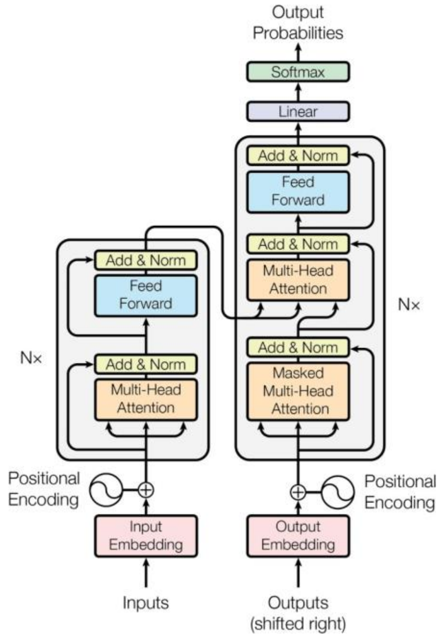
(seq, d_{model})

<SOS> ti

Use the encoder output from the first time step

<SOS>I love you very much<EOS>

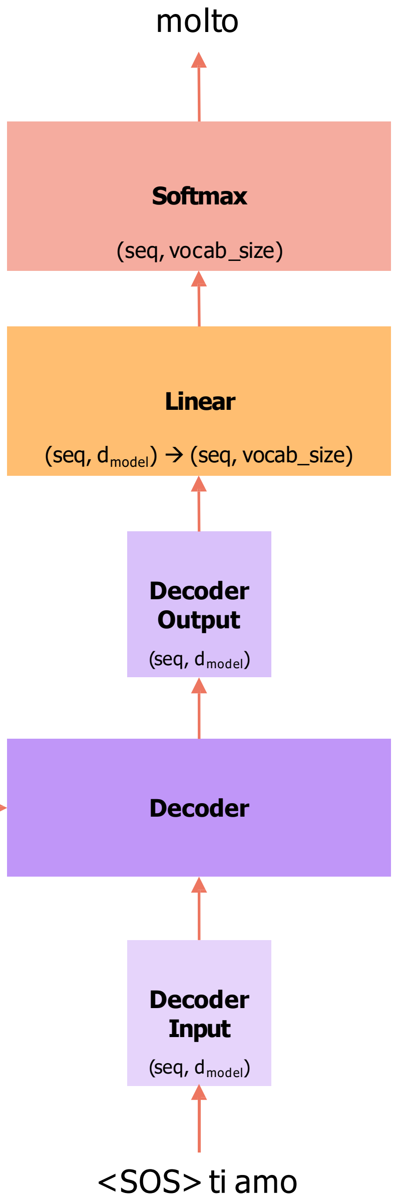
Since decoder input now contains **two** tokens, we select the softmax corresponding to the second token.



Append the previously output word to the decoder input

Inference

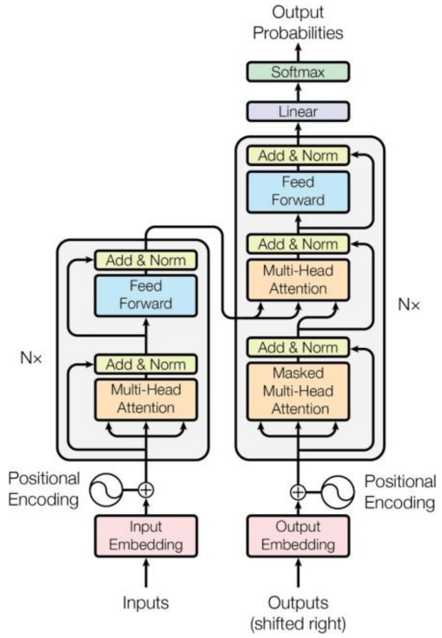
Time Step = 3



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

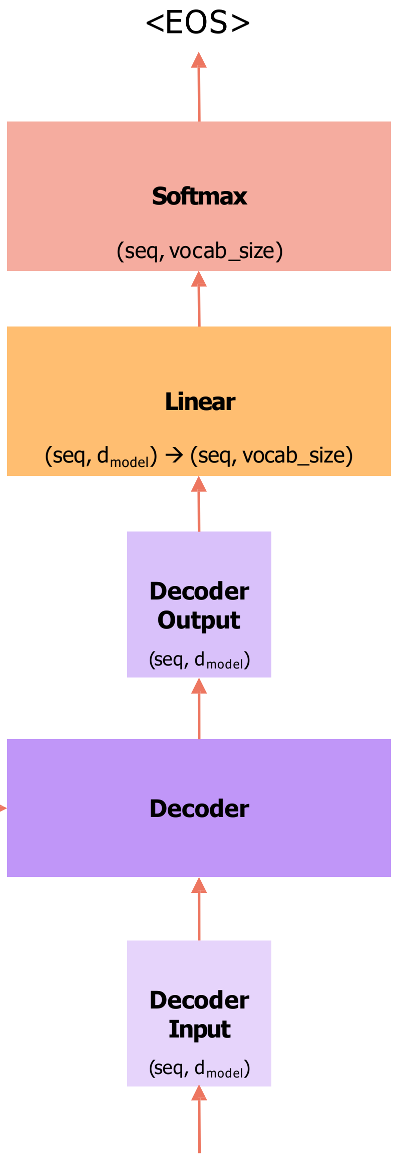
<SOS>I love you very much<EOS>



Append the previously output word to the decoder input

Inference

Time Step = 4

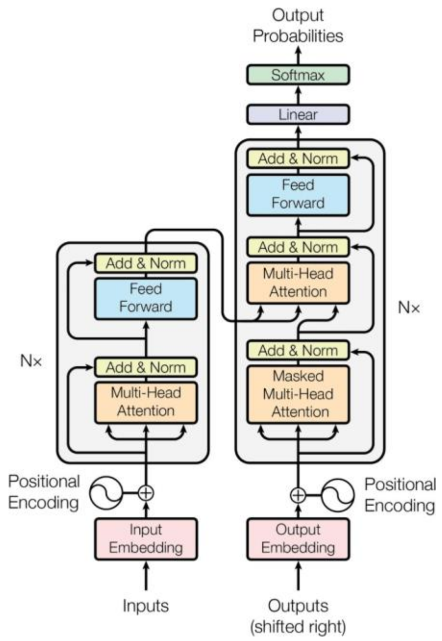


Since decoder input now contains **four** tokens, we select the softmax corresponding to the fourth token.

Use the encoder output from the first time step

<SOS>I love you very much<EOS>

<SOS> ti amo molto

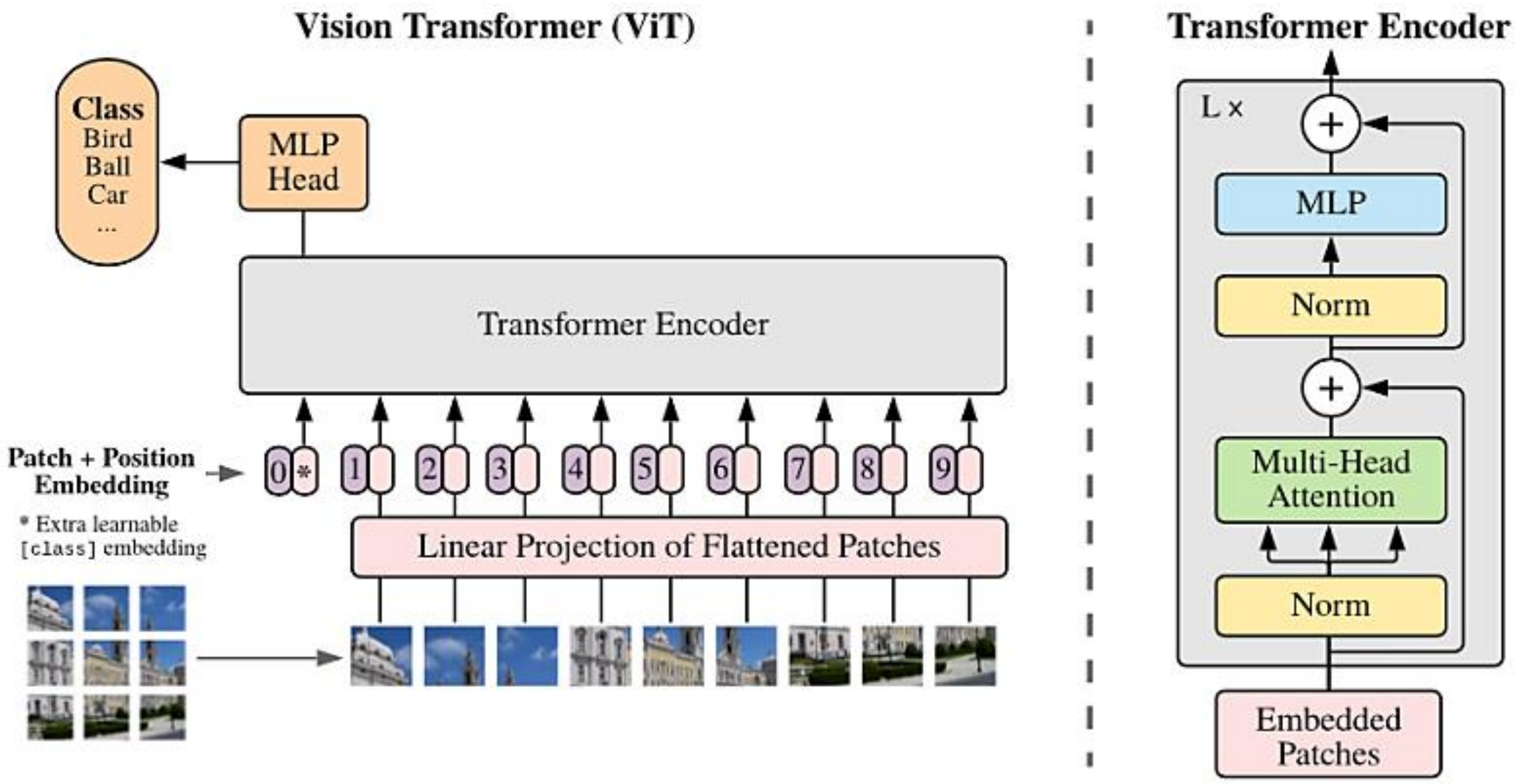


Append the previously output word to the decoder input

Inference strategy

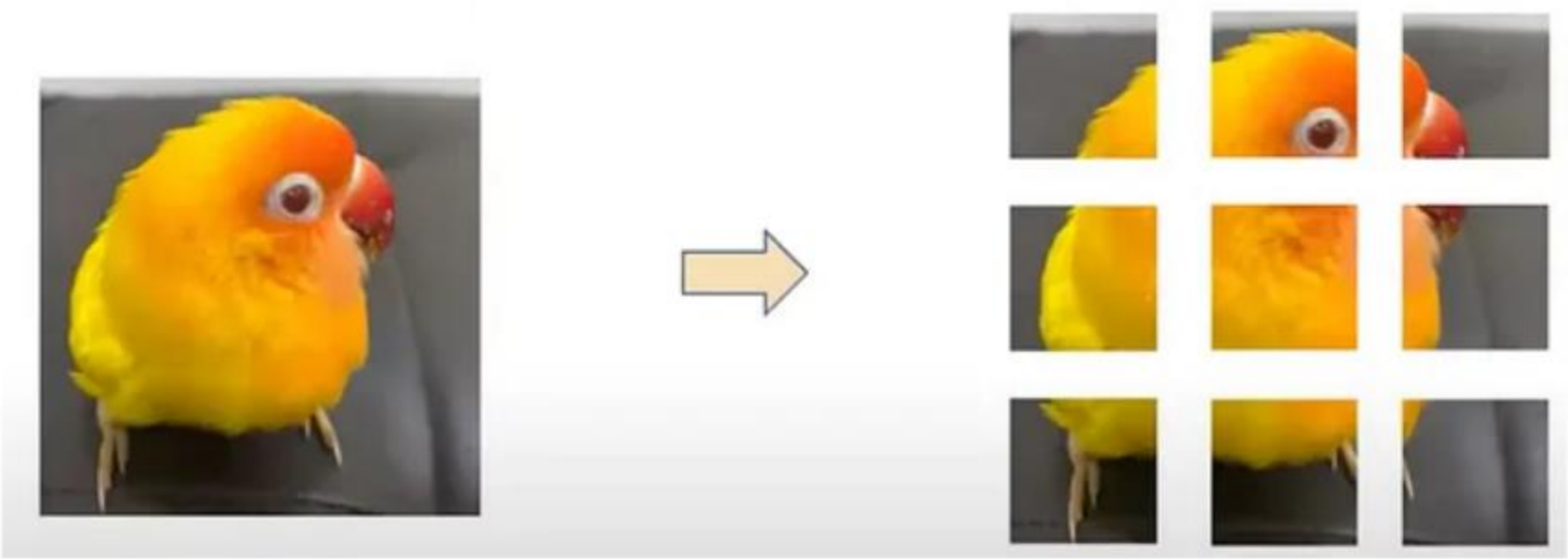
- We selected, at every step, the word with the maximum softmax value. This strategy is called **greedy** and usually does not perform very well.
- A better strategy is to select at each step the top *B* words and evaluate all the possible next words for each of them and at each step, keeping the top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

Vision Transformers ViT



https://viso.ai/deep-learning/vision-transformer-vit/

Image to patch



https://khvmaths.medium.com/vision-transformer-understanding-the-underlying-concept-83d699d71180

Patch Embedding

A standard Transformer takes 1D sequence of token embedding as input. Therefore, for a 2D image, we need to reshape the image into a sequence of flattened 2D patches.

Let's take an image size of 48×48, assuming we split into fixed-size patch of 16×16, therefore we will have 9 patches. Just take $(48/16) \times (48/16)$ and you will get 9 patches. Let's illustrate that in picture below.

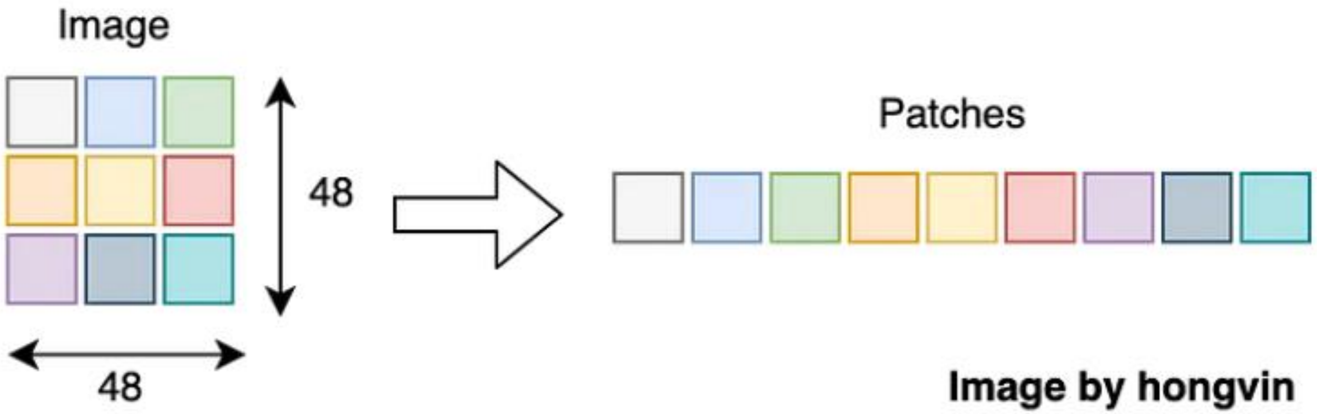
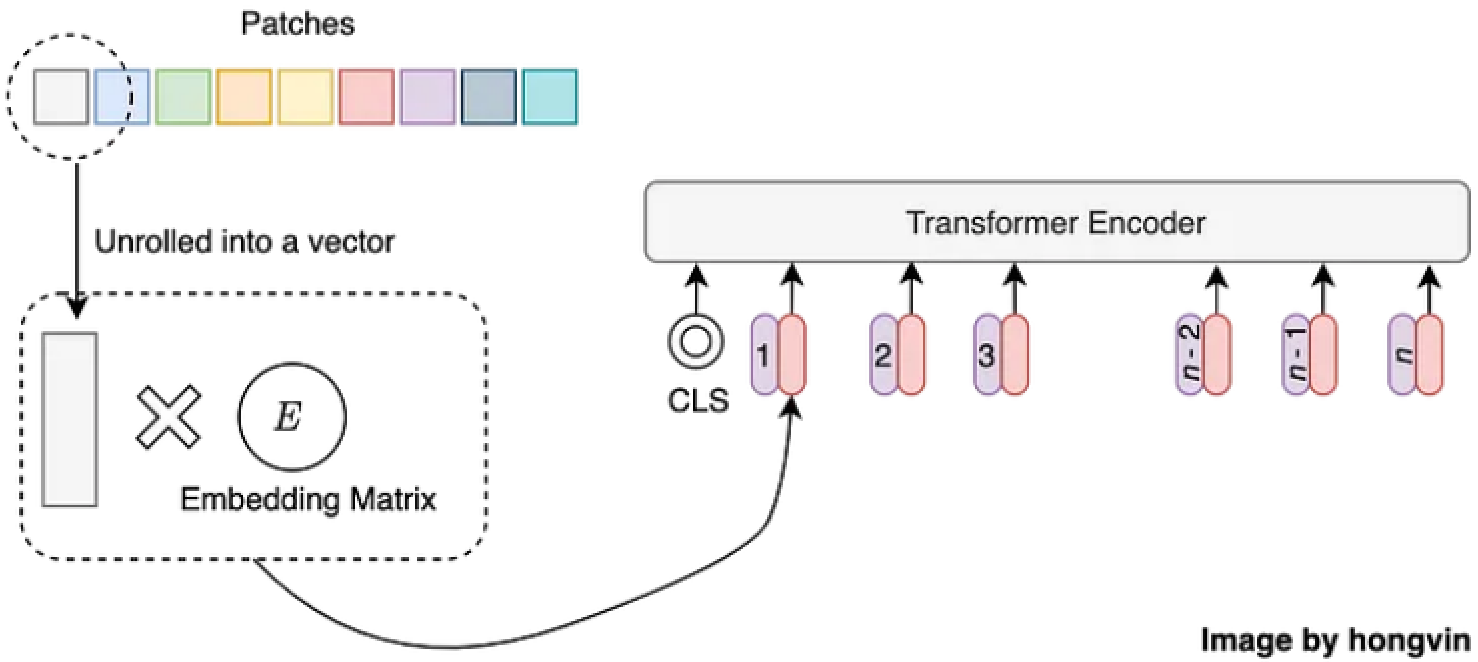


Image by hongvin

the image patches are linearly projected into a vector using a learned embedding matrix **E**.



1. Image of size (H×W×C) is split into *n* patches of size (P×P×C).
2. The patches are flatten. The size of flattened patch has vector shape of ($1 \times P^2 * C$).
3. The flattened patches is multiplied with embedding tensor, **E** of shape ($P^2 * C \times d$). The final embedded patches is now having shape of ($1 \times d$). d is the model dimension.
4. A *cls* token is prepended to the sequence of patch embeddings.
5. Positional embedding is added to the sequence.It learns the positional information for each of the patches.

Image: 224x224x3

224/16=14 patches..14x14

14x14 patches of 16x16 size

Each patch is flattened 16x16x3= 768 x1

1. Patch Splitting (No Change in Overall Size)

- Suppose we have an image of **224×224×3**.
- We divide it into **16×16 patches** → this results in **(224/16) × (224/16) = 14 × 14 = 196 patches**.
- Each patch is of size **16×16×3** pixels.

2. Flattening & Tokenization

- Each **16×16×3** patch is flattened into a **1D vector of size 768 (16×16×3)**.
- So, we now have **196 such vectors (tokens)**.
- These **196 tokens** are then projected into a higher-dimensional embedding space (e.g., 768D) using a **learnable linear layer**.

3. Why Do We Do This Instead of Feeding the Whole Image?

- **Self-Attention Mechanism** in transformers works on **sequences of tokens**, not on raw images like CNNs.
- Instead of processing the entire 224×224 image at once, we treat it as a sequence of **196 tokens**.
- This enables the **global attention mechanism**, meaning each patch can attend to every other patch, capturing **long-range dependencies**.

4. Computational Complexity

For an input sequence of **N tokens** (patches), self-attention has a **quadratic complexity**:

$O(N^2\ d)$where:

- **N** = number of patches (e.g., 196 for a 224×224 image with 16×16 patches)
- **d** = embedding dimension (e.g., 768)

Drawbacks of ViT

1. **High Computational Complexity:** The self-attention mechanism in standard ViTs has quadratic complexity with respect to the number of image patches. This makes them computationally expensive, especially for high-resolution images.
2. **Data Hungery:** ViTs often require large amounts of training data to perform well. They don't have the inductive biases of CNNs (such as locality and translation equivariance), which allow CNNs to generalize better from smaller datasets. This data hunger can be a problem when training data is limited.
3. **Slow Training** – Due to the large number of parameters and the lack of spatial hierarchies (like pooling layers in CNNs), training ViTs takes longer and requires extensive tuning.
4. **Sensitivity to Patch Size:** The performance of ViTs can be sensitive to the choice of patch size. Choosing an inappropriate patch size can negatively impact performance. Finding the optimal patch size often requires experimentation.
5. **Lack of Spatial Inductive Biases** – Unlike CNNs, ViTs do not inherently capture local spatial features, making them less effective for tasks requiring fine-grained spatial understanding without additional architectural modifications