8.5

Understanding Self-Attention

Part 4: Masked Attention And Positional Encoding

Sebastian Raschka and the Lightning AI Team

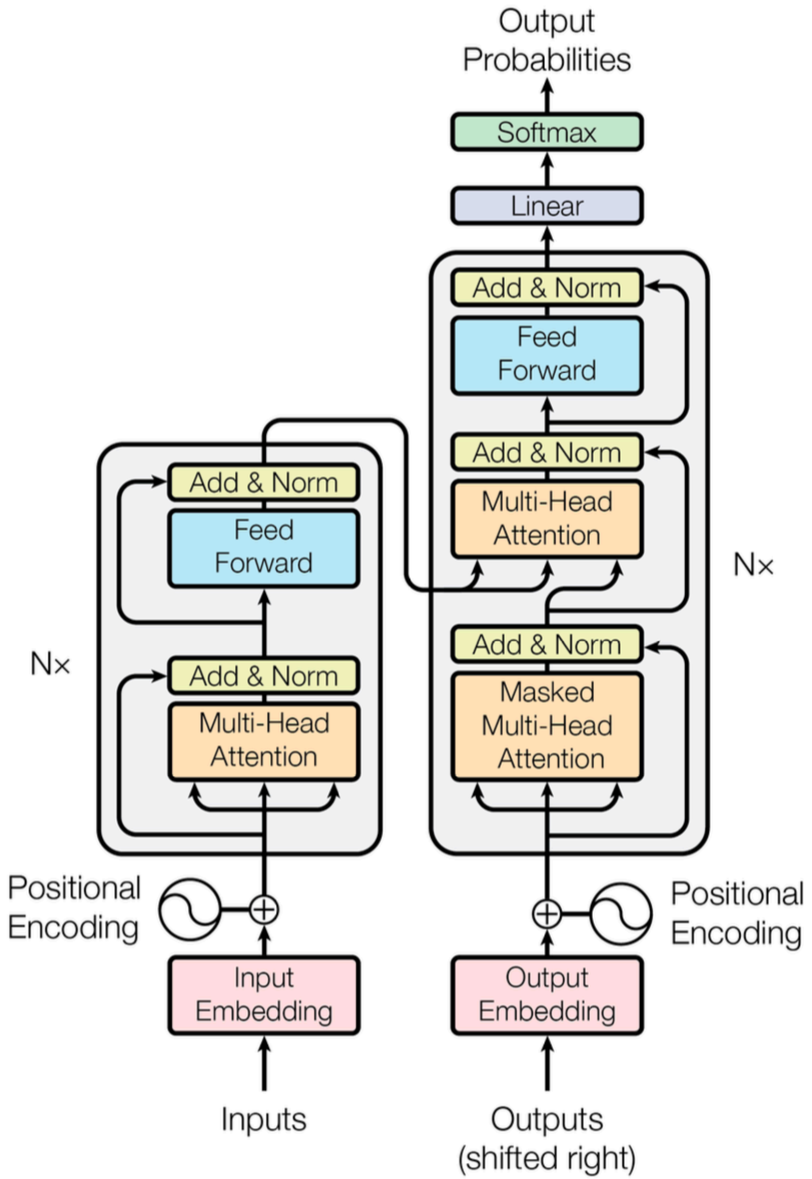
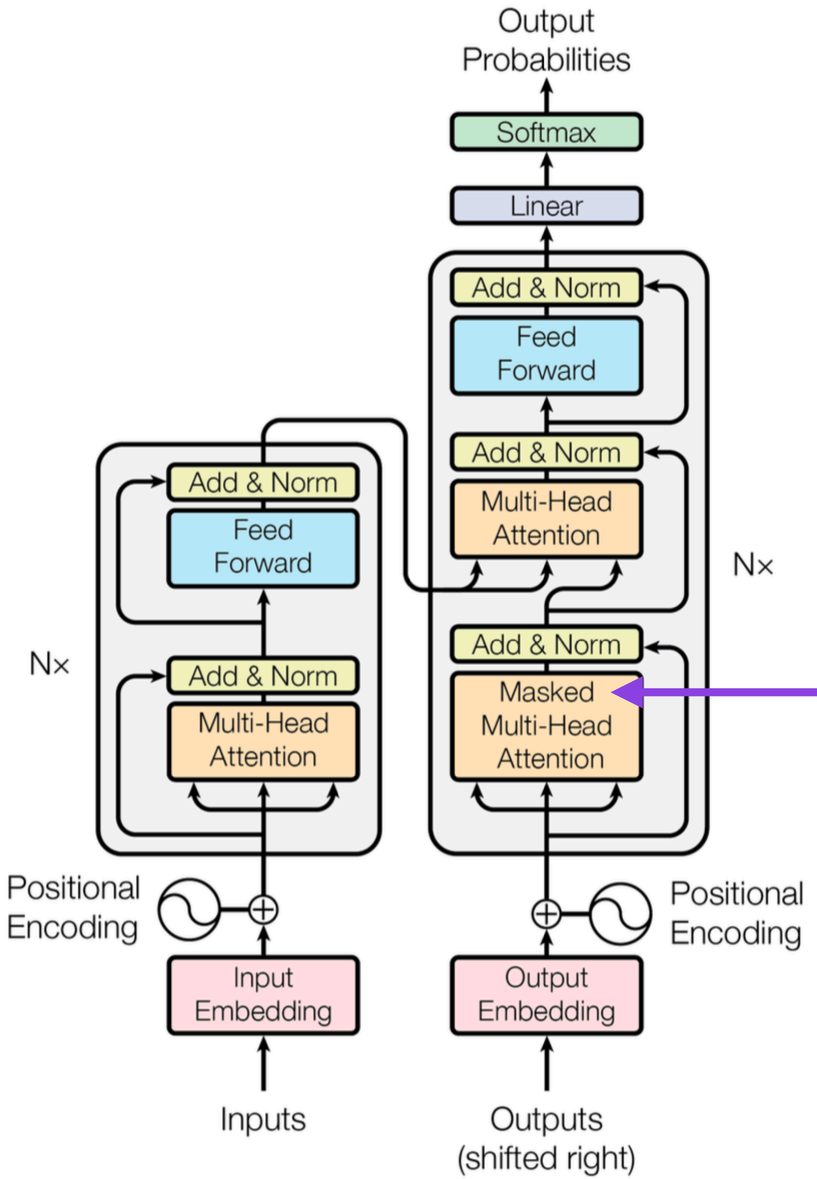


Figure 1: The Transformer - model architecture.



Masked?

Figure 1: The Transformer - model architecture.

Masked Multi-Head Attention

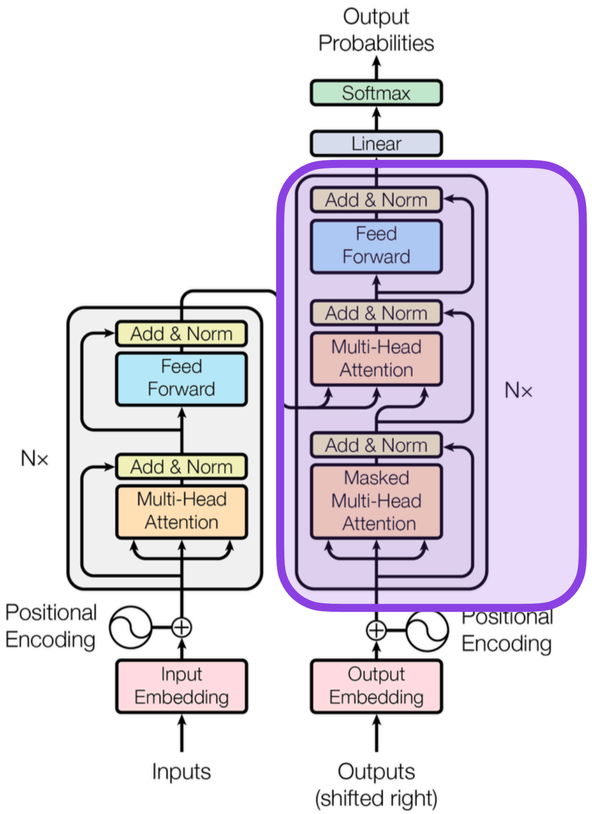
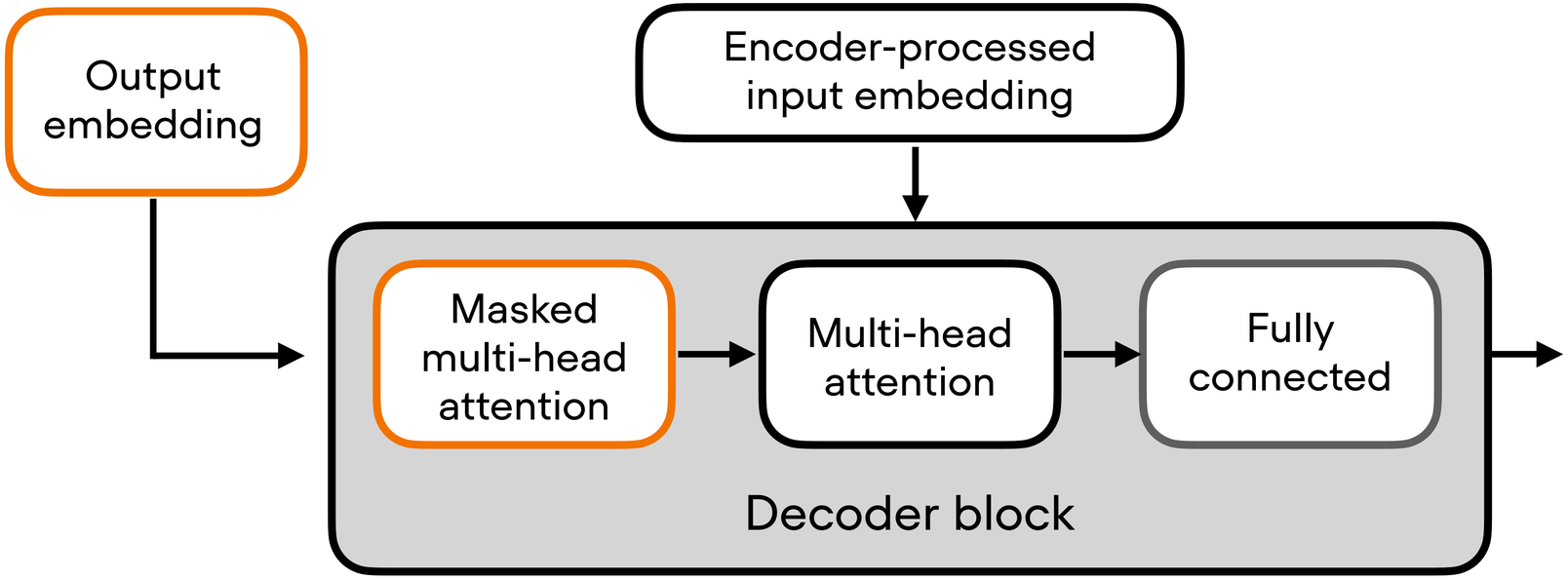


Figure 1: The Transformer - model architecture.



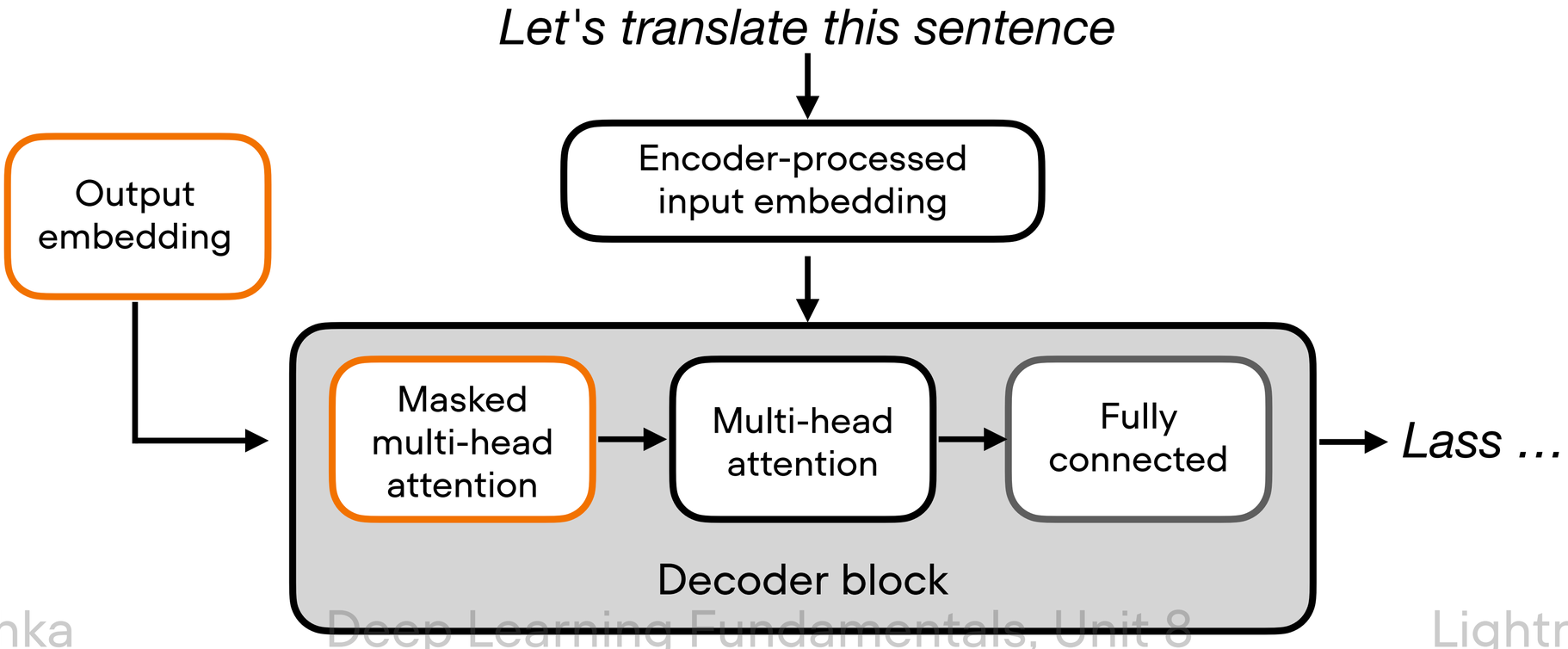
(A concise sketch of the decoder block)

English → German Translation

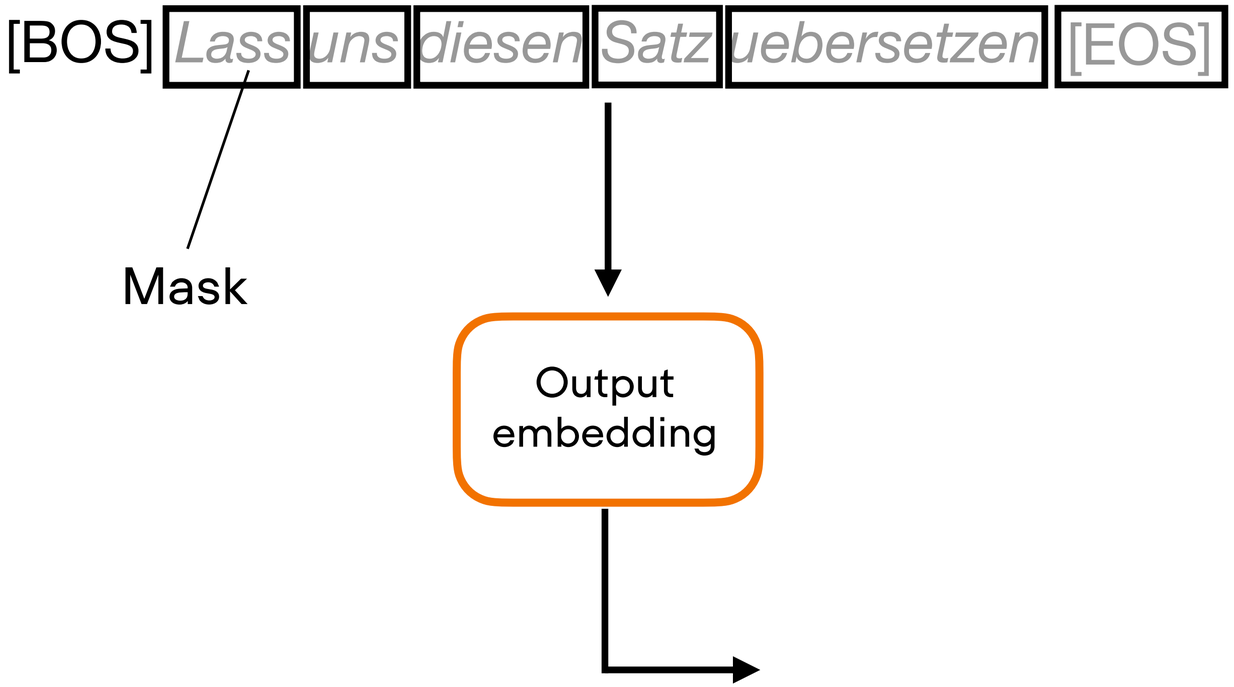
Input: *Let's translate this sentence*

Target: *Lass uns diesen Satz uebersetzen*

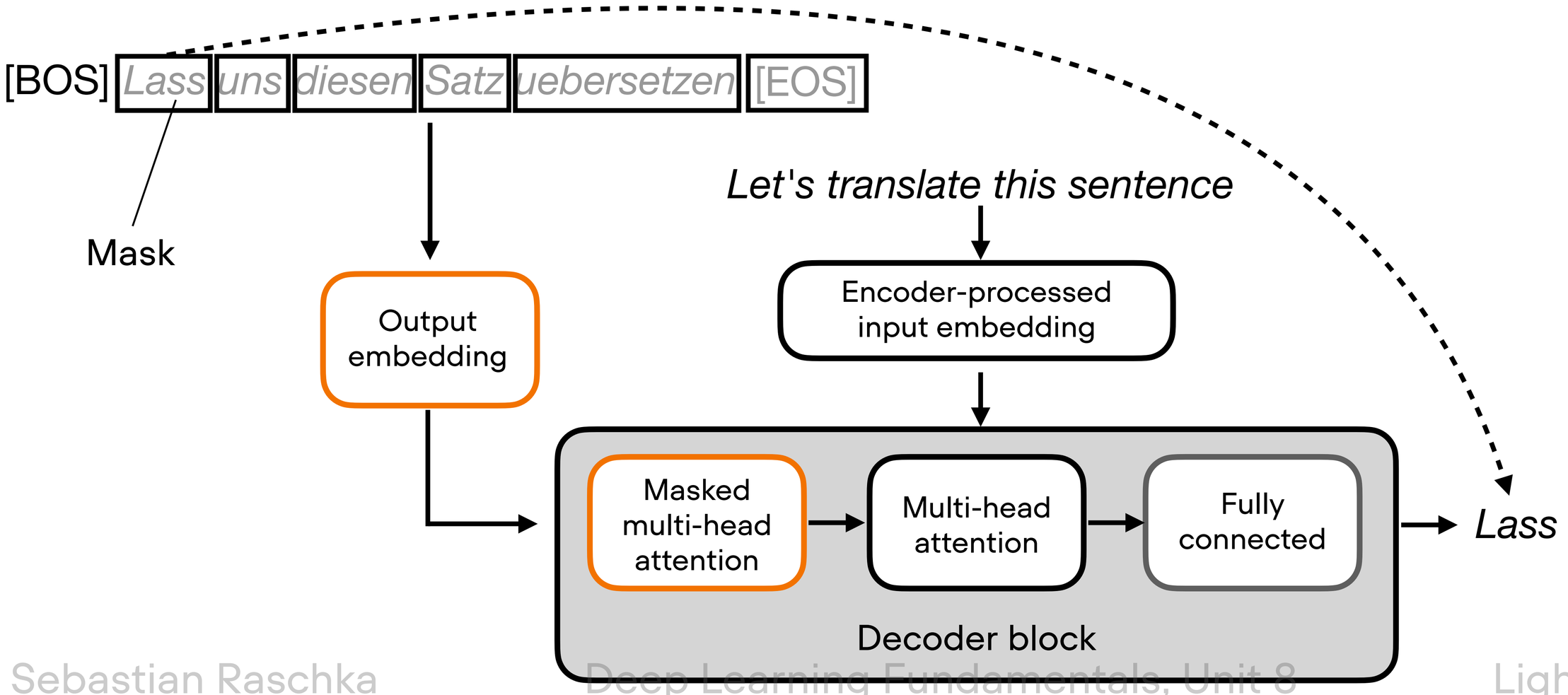
Task is to translate the input into German



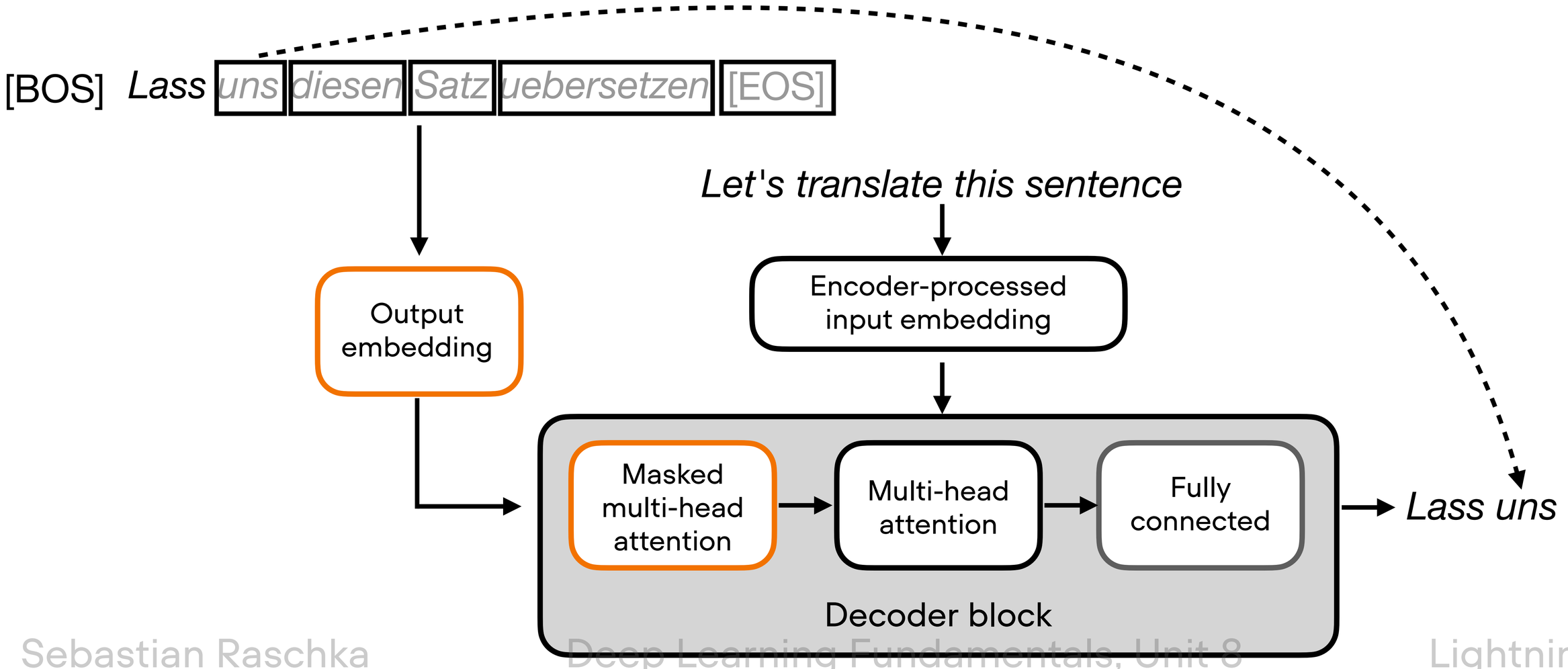
Masked attention masks out tokens the model hasn't seen before (here visualized on the input instead of embedding)



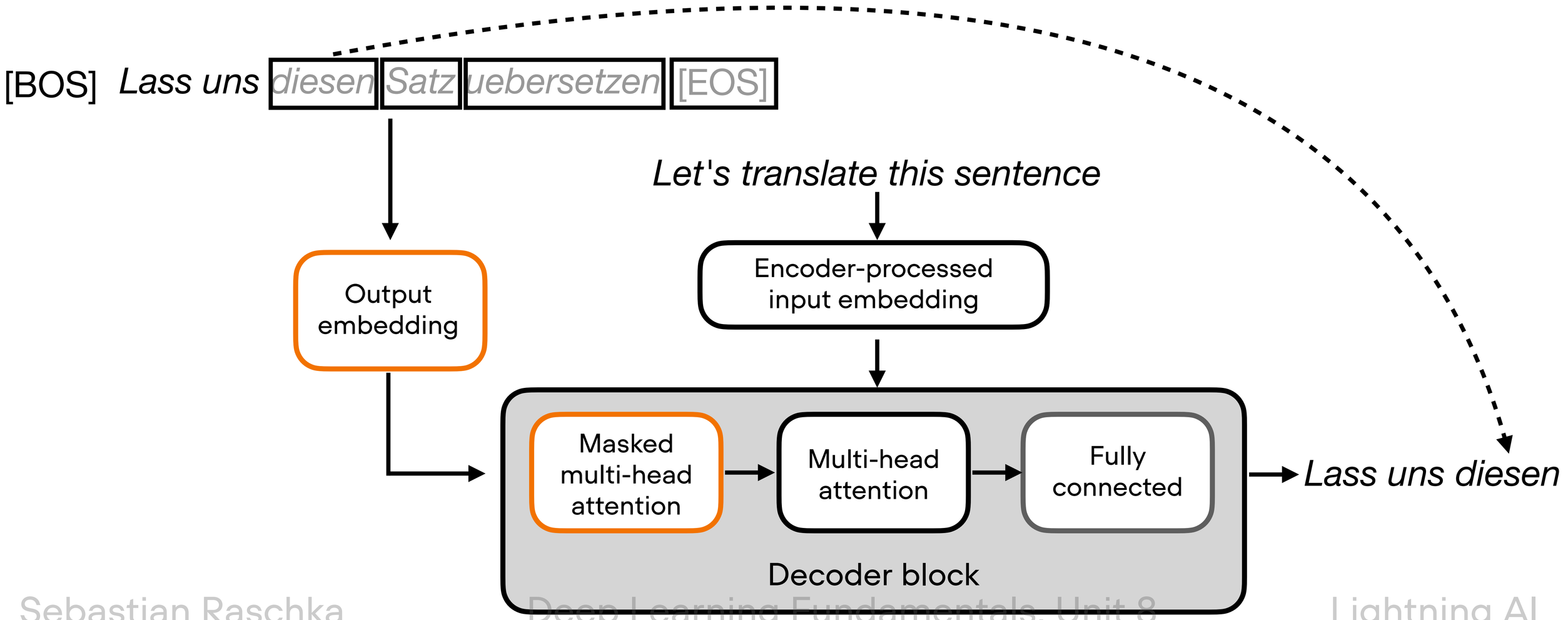
Masked attention masks out tokens the model hasn't seen before (here visualized on the input instead of embedding)



Time step 2


[BOS]
Lass
uns
diesen
Satz
uebersetzen
[EOS]
Let's translate this sentence
Output embedding
Encoder-processed input embedding
Masked multi-head attention
Multi-head attention
Fully connected
Decoder block
Lass uns

Time step 3


[BOS]
Lass uns
diesen
Satz
uebersetzen
[EOS]
Let's translate this sentence
Output embedding
Encoder-processed input embedding
Masked multi-head attention
Multi-head attention
Fully connected
Decoder block
Lass uns diesen

Positional encodings?

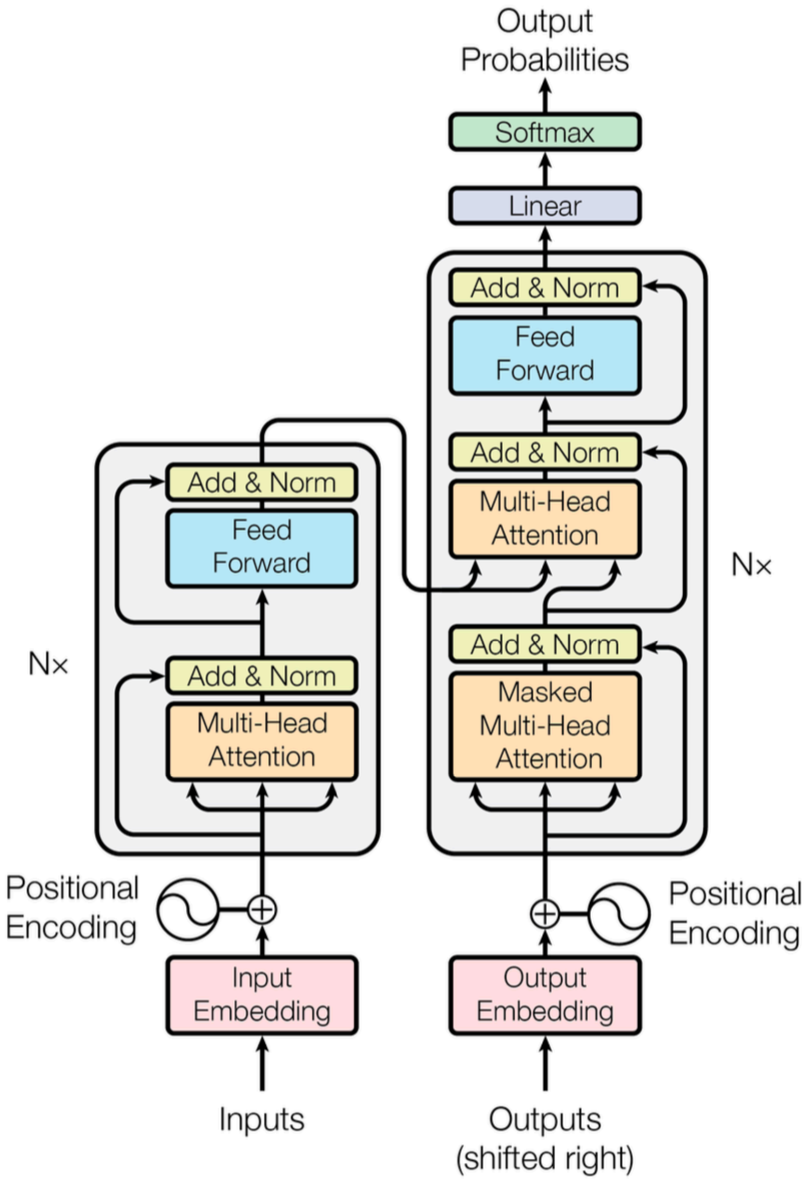


Figure 1: The Transformer - model architecture.

Why positional encodings?

Self-attention and fully-connected layers
are permutation invariant

```
import torch

input_values = torch.tensor([[1.,  5., -2.],  # 1st example
                             [-1., .3, 1.4]]) # 2nd example

torch.manual_seed(123)
layer = torch.nn.Linear(3, 1)

with torch.no_grad():
    print(layer(input_values))
```

```
tensor([[0.6514],
        [0.0574]])
```

```python
import torch

input_values = torch.tensor([[1.,  5., -2.],  # 1st example
                             [-1., .3, 1.4]]) # 2nd example

torch.manual_seed(123)
layer = torch.nn.Linear(3, 1)

with torch.no_grad():
    print(layer(input_values))
```

```
tensor([[0.6514],
        [0.0574]])
```

```python
shuffle = [2, 0, 1]
input_values = input_values[:, shuffle]
layer.weight = torch.nn.Parameter(layer.weight[:, shuffle])
```

```
import torch

input_values = torch.tensor([[1.,  5., -2.],  # 1st example
                             [-1., .3, 1.4]]) # 2nd example

torch.manual_seed(123)
layer = torch.nn.Linear(3, 1)

with torch.no_grad():
    print(layer(input_values))
```

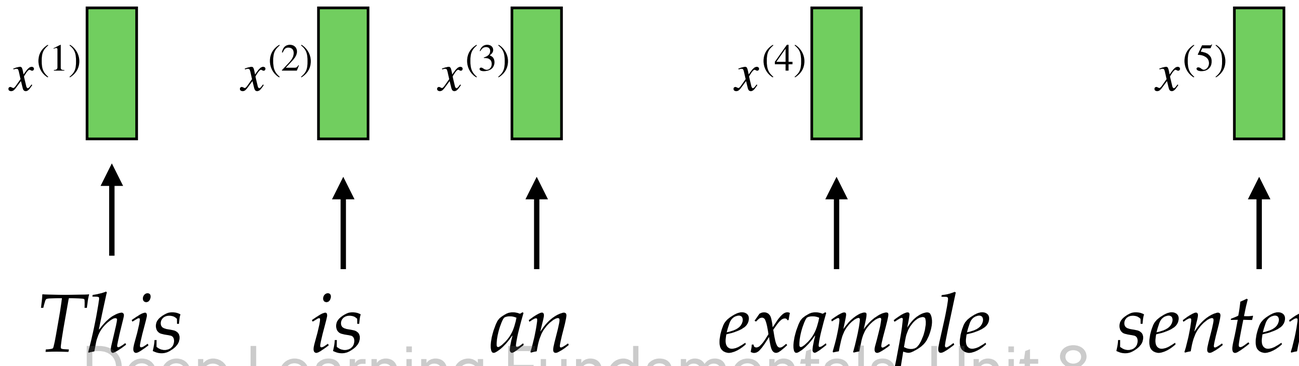
```
tensor([[0.6514],
        [0.0574]])
```

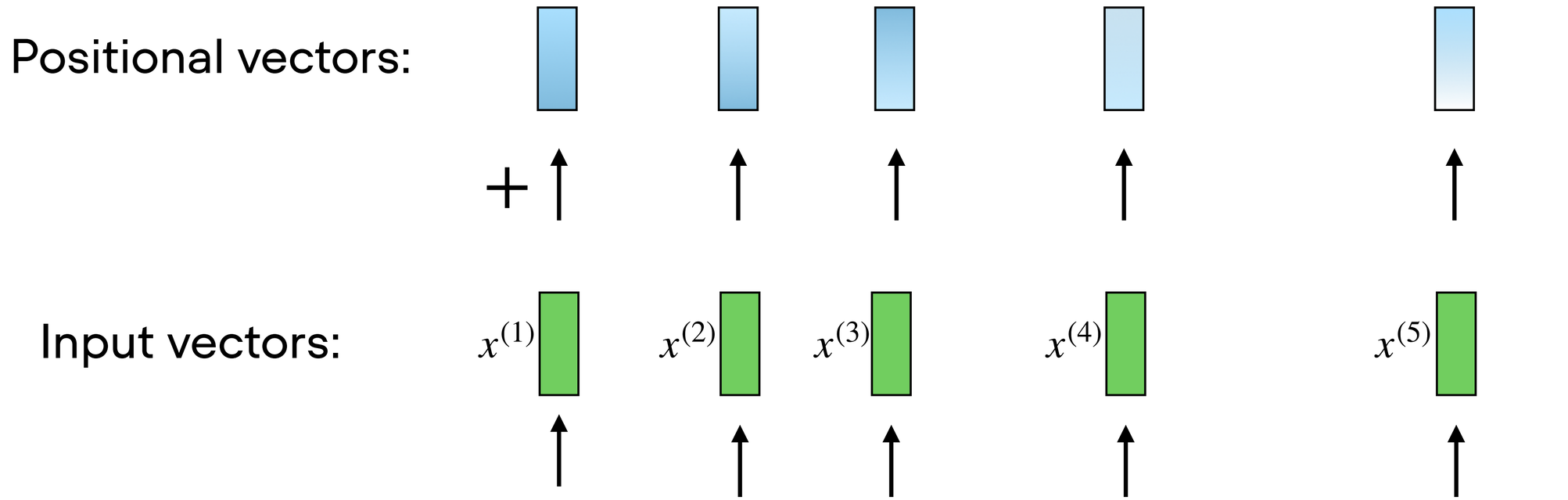
```
shuffle = [2, 0, 1]
input_values = input_values[:, shuffle]
layer.weight = torch.nn.Parameter(layer.weight[:, shuffle])
```

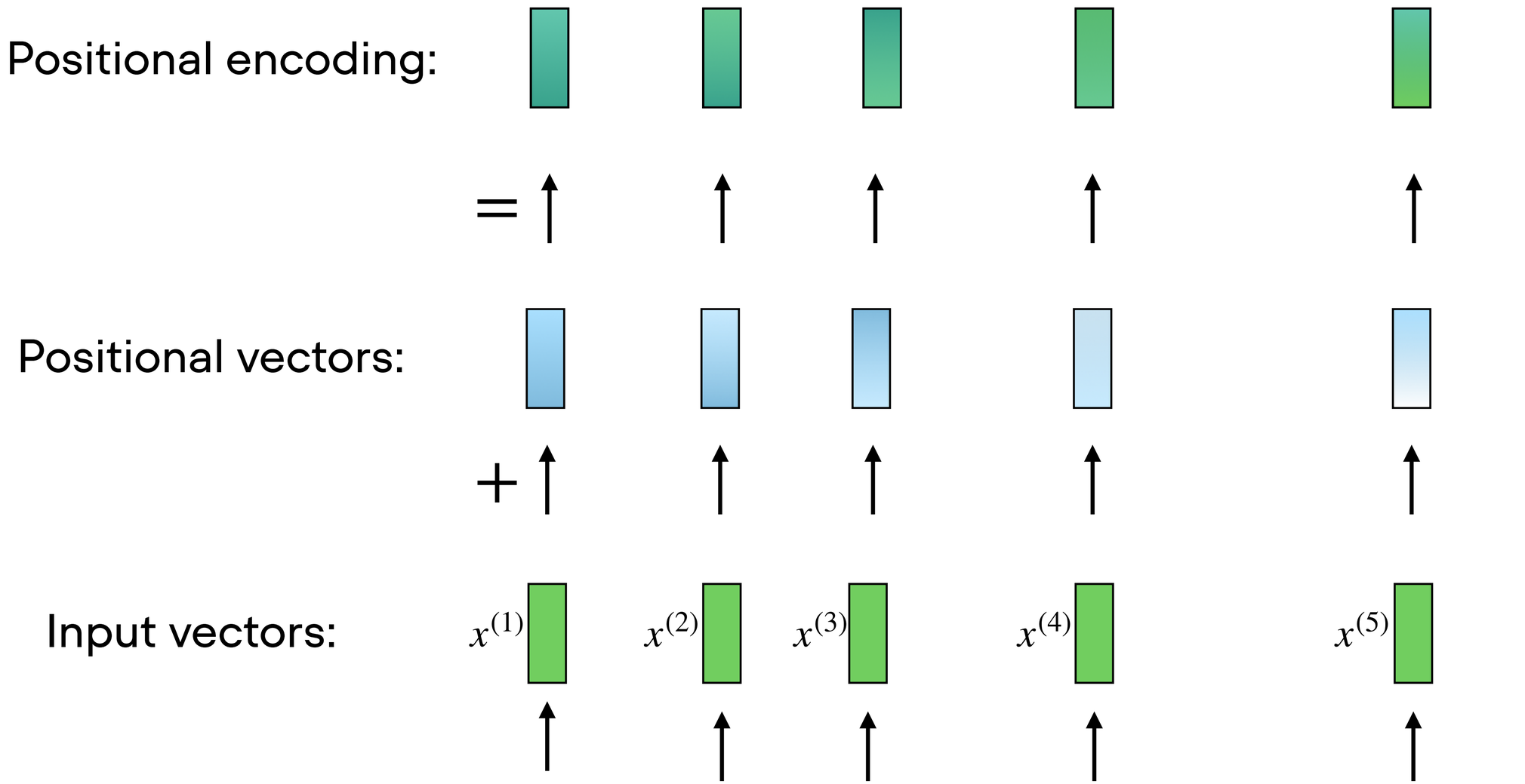
```
with torch.no_grad():
    print(layer(input_values))
```

```
tensor([[0.6514],
        [0.0574]])
```

Input vectors:




Positional vectors:
Input vectors:
$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(4)}$
$x^{(5)}$
This
is
an
example
sentence


Positional encoding:
=
Positional vectors:
+
Input vectors:
$x^{(1)}$
$x^{(2)}$
$x^{(3)}$
$x^{(4)}$
$x^{(5)}$
This
is
an
example
sentence

Next: Large language models