Visual Transformers (ViTs)

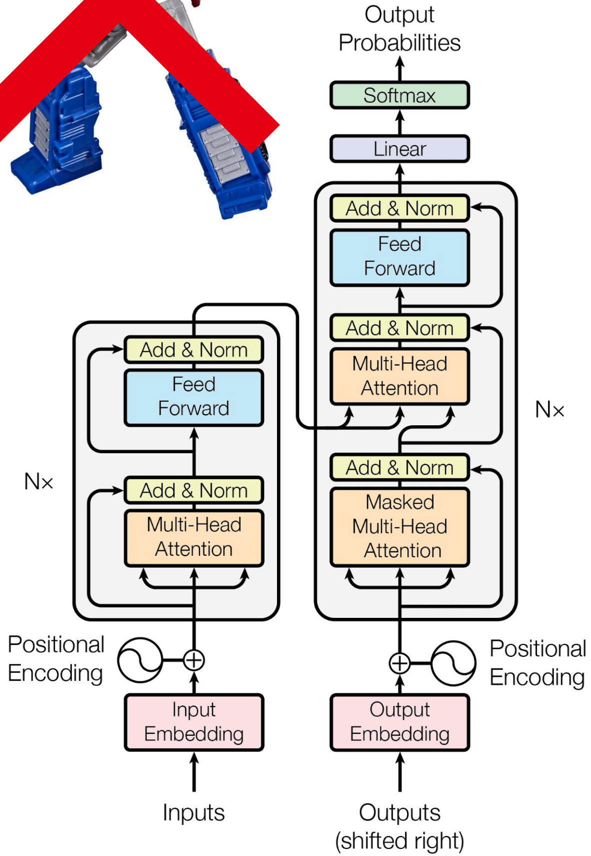
Krzysztof Król

But what even are "**transformers**"?

1. A deep learning architecture
2. Based on the **attention** mechanism
3. Scalable and parallel-izable by default
4. Multi-modal (text, image, sound etc.)

Variants:

- **encoder-only** (BERT)
- **decoder-only** (GPT)
- **encoder + decoder** (Translation models)



History - “Attention is all you need”

- published in 2017
- mostly google-authored
- unassuming title
- architecture **almost unchanged** since then

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

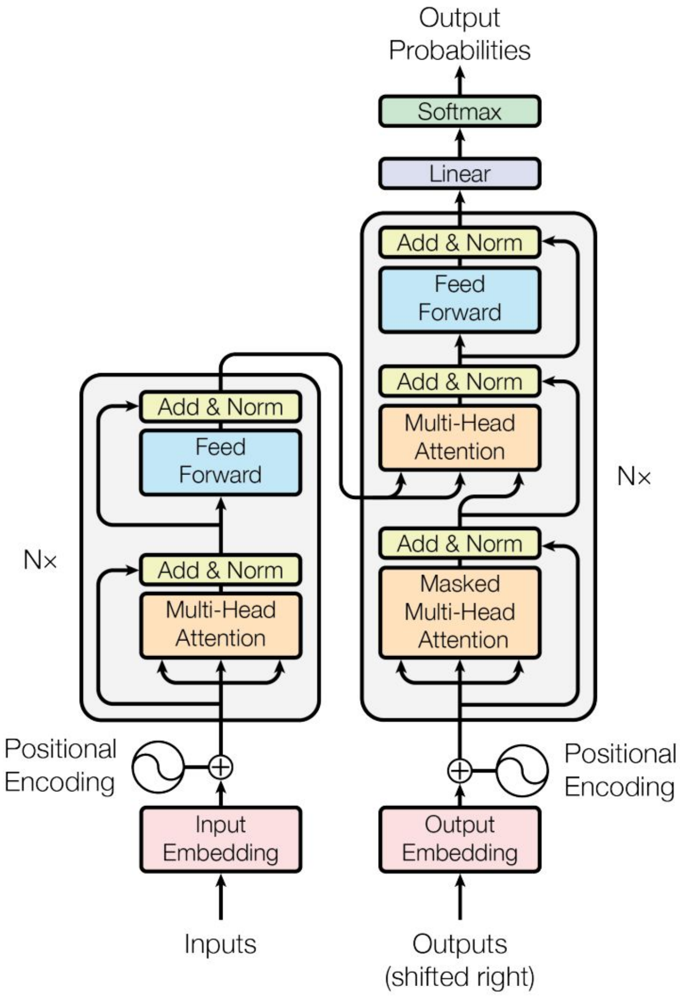


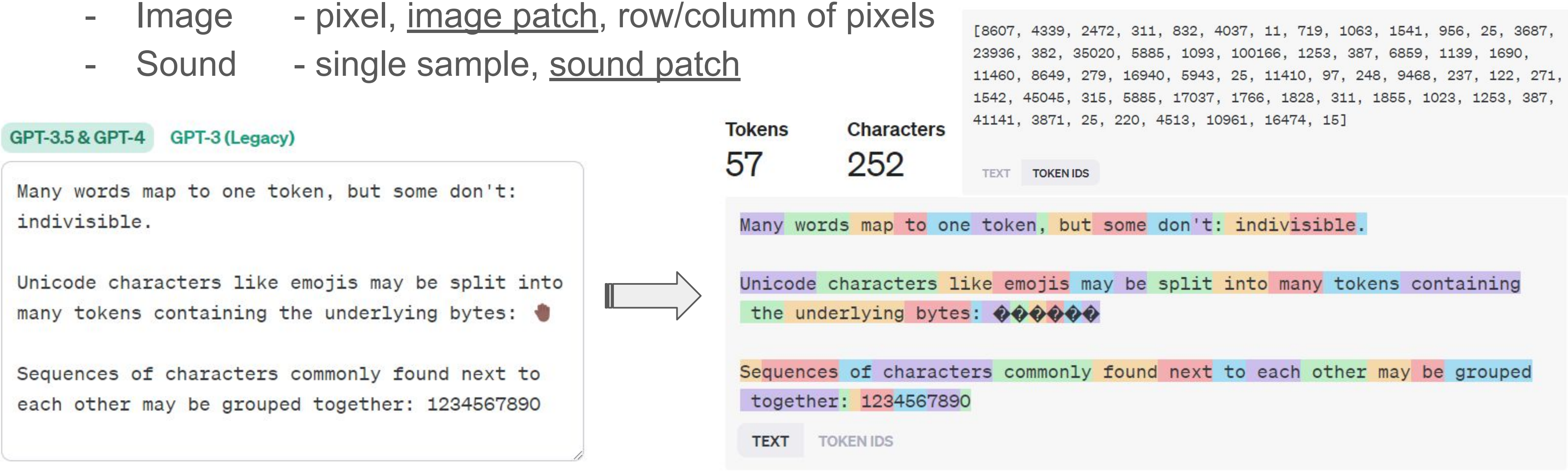
Figure 1: The Transformer - model architecture.

Tokens

Represented as a node in a **directional computation graph**.

The most fundamental unit of input. They can be:

- Text - character, sub-word part, word, sentence
- Image - pixel, image patch, row/column of pixels
- Sound - single sample, sound patch

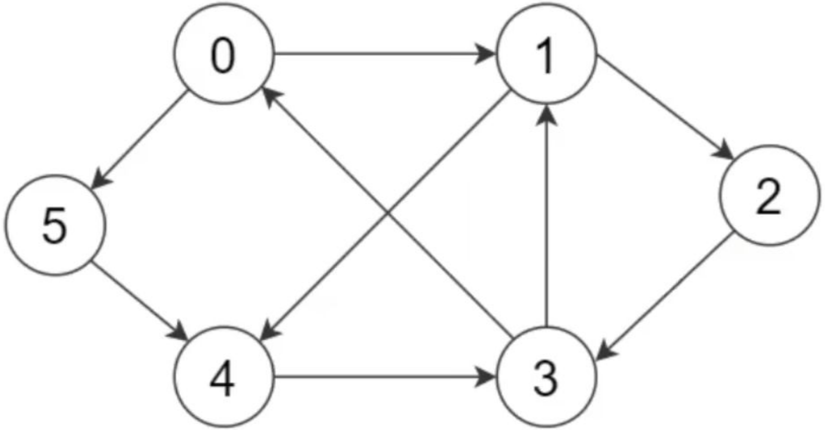


Attention

Mechanism defining how nodes (**tokens**) "communicate" with each other.

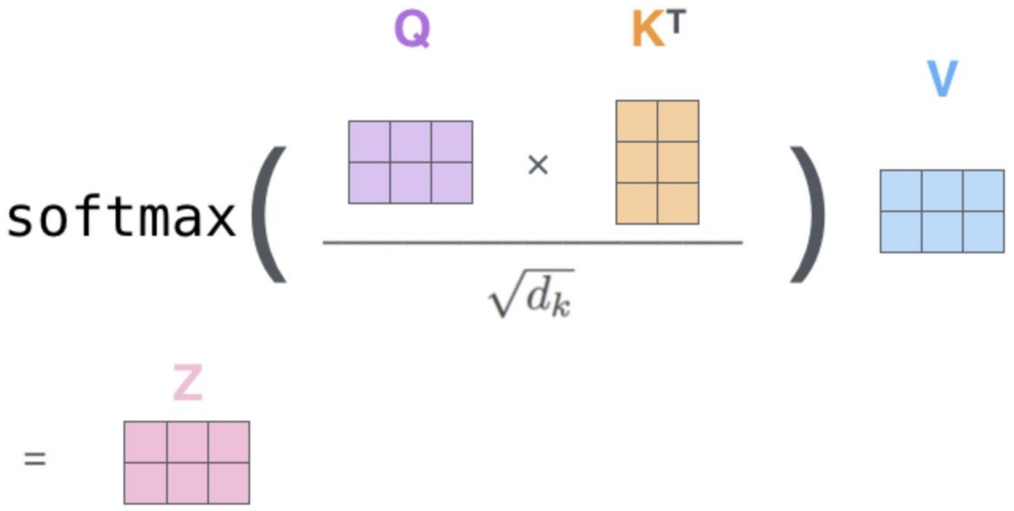
From a node's POV:

- **Query** (Q) - what i'm looking for
- **Key** (K) - what i have
- **Value** (V) - what i will communicate



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

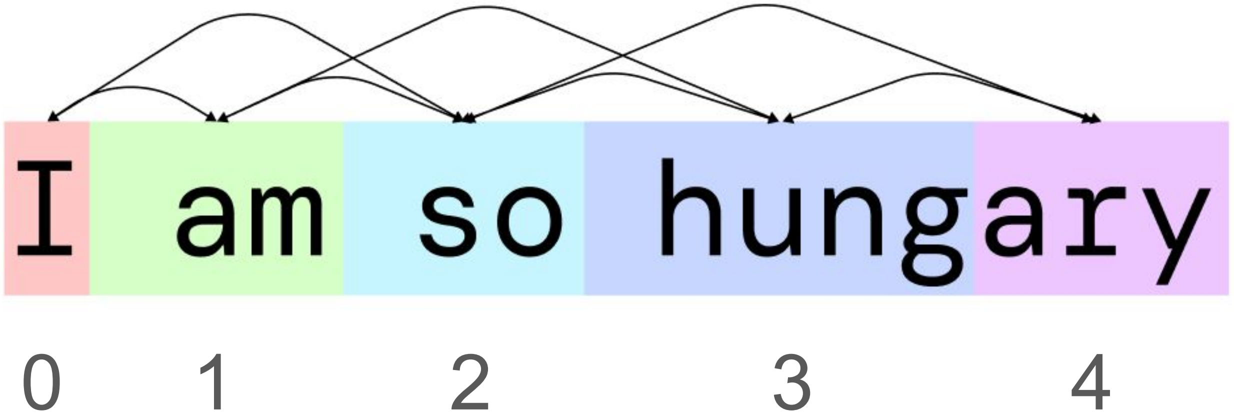
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

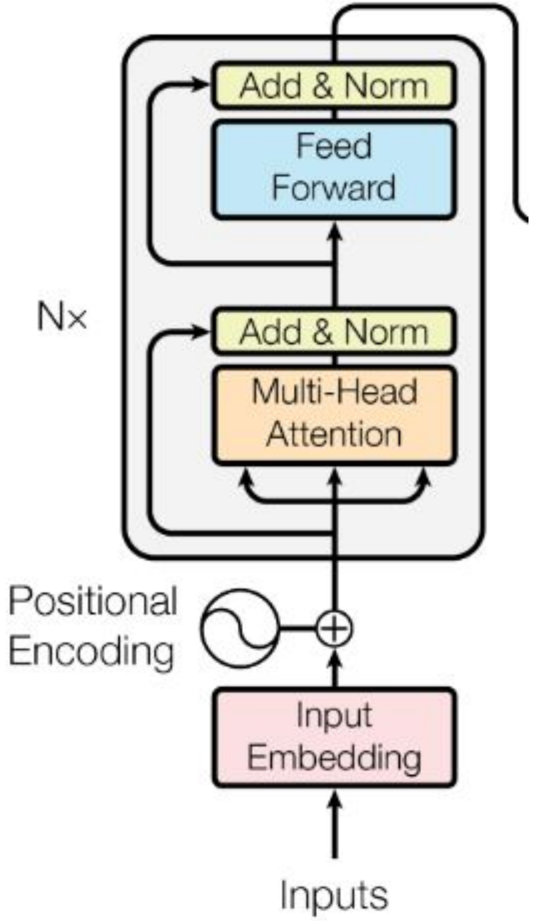


The self-attention calculation in matrix form

Encoder

All **nodes** communicate with each other.



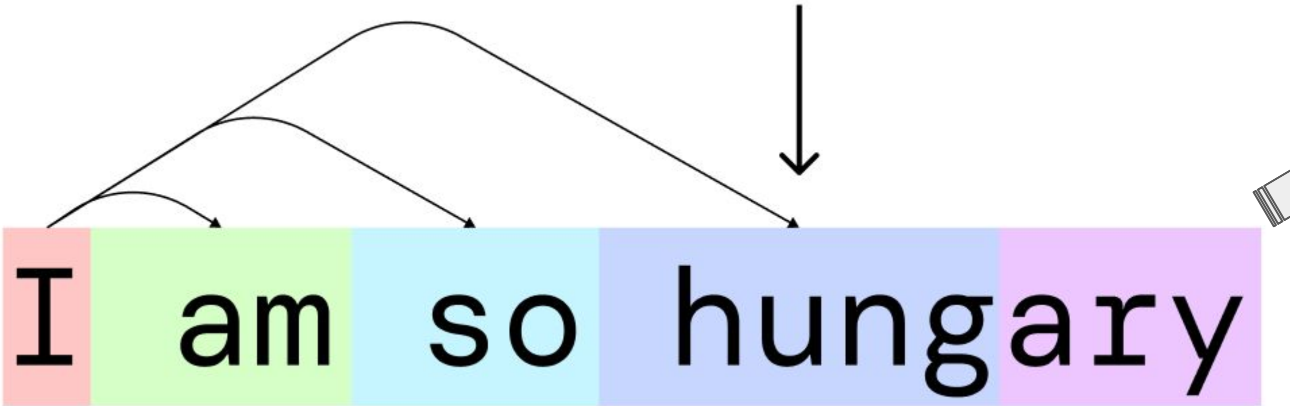


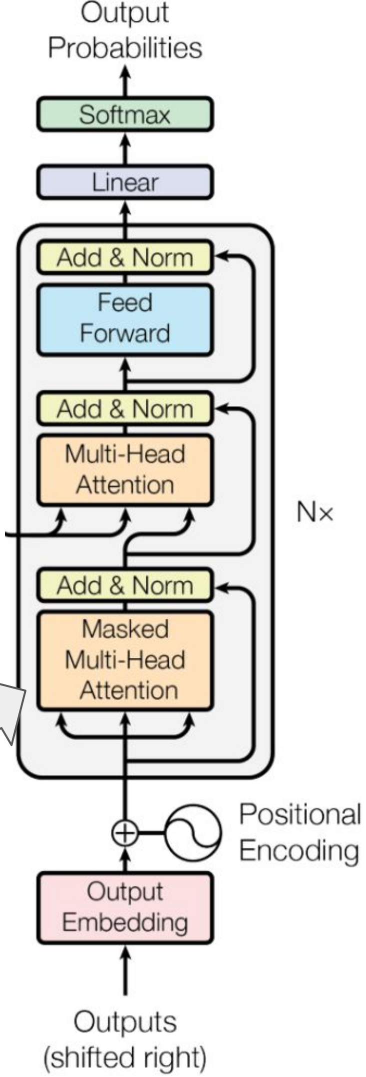
Decoder

```
INPUTS xb.shape=torch.Size([4, 8])
 xb=tensor([[24, 43, 58,  5, 57,  1, 46, 43],
        [44, 53, 56,  1, 58, 46, 39, 58],
        [52, 58,  1, 58, 46, 39, 58,  1],
        [25, 17, 27, 10,  0, 21,  1, 54]])

TARGETS yb.shape=torch.Size([4, 8])
 yb=tensor([[43, 58,  5, 57,  1, 46, 43, 39],
        [53, 56,  1, 58, 46, 39, 58,  1],
        [58,  1, 58, 46, 39, 58,  1, 46],
        [17, 27, 10,  0, 21,  1, 54, 39]])
```

```
when input is [24] the target: 43
when input is [24, 43] the target: 58
when input is [24, 43, 58] the target: 5
when input is [24, 43, 58, 5] the target: 57
when input is [24, 43, 58, 5, 57] the target: 1
when input is [24, 43, 58, 5, 57, 1] the target: 46
when input is [24, 43, 58, 5, 57, 1, 46] the target: 43
when input is [24, 43, 58, 5, 57, 1, 46, 43] the target: 39
when input is [44] the target: 53
when input is [44, 53] the target: 56
when input is [44, 53, 56] the target: 1
when input is [44, 53, 56, 1] the target: 58
when input is [44, 53, 56, 1, 58] the target: 46
when input is [44, 53, 56, 1, 58, 46] the target: 39
```





Masked Multi-Head attention

```
tril
```

✓ 0.0s

```
tensor([[1., 0., 0., 0., 0., 0., 0., 0.],
        [1., 1., 0., 0., 0., 0., 0., 0.],
        [1., 1., 1., 0., 0., 0., 0., 0.],
        [1., 1., 1., 1., 0., 0., 0., 0.],
        [1., 1., 1., 1., 1., 0., 0., 0.],
        [1., 1., 1., 1., 1., 1., 0., 0.],
        [1., 1., 1., 1., 1., 1., 1., 0.],
        [1., 1., 1., 1., 1., 1., 1., 1.]])
```

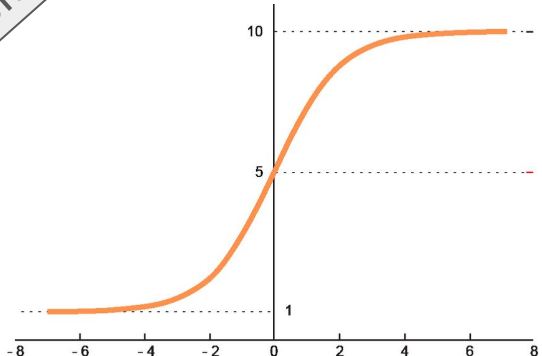
+

```
weights = torch.zeros((T, T))
weights = weights.masked_fill(tril == 0, float('-inf'))
# weights = F.softmax(weights, dim=-1)
weights
```

```
tensor([[0., -inf, -inf, -inf, -inf, -inf, -inf, -inf],
        [0., 0., -inf, -inf, -inf, -inf, -inf, -inf],
        [0., 0., 0., -inf, -inf, -inf, -inf, -inf],
        [0., 0., 0., 0., -inf, -inf, -inf, -inf],
        [0., 0., 0., 0., 0., -inf, -inf, -inf],
        [0., 0., 0., 0., 0., 0., -inf, -inf],
        [0., 0., 0., 0., 0., 0., 0., -inf],
        [0., 0., 0., 0., 0., 0., 0., 0.]])
```

Softmax

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




```
weights = torch.zeros((T, T))
weights = weights.masked_fill(tril == 0, float('-inf'))
weights = F.softmax(weights, dim=-1)
weights
```

✓ 0.0s

```
tensor([[1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.5000, 0.5000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.3333, 0.3333, 0.3333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.2500, 0.2500, 0.2500, 0.2500, 0.0000, 0.0000, 0.0000, 0.0000],
        [0.2000, 0.2000, 0.2000, 0.2000, 0.2000, 0.0000, 0.0000, 0.0000],
        [0.1667, 0.1667, 0.1667, 0.1667, 0.1667, 0.1667, 0.0000, 0.0000],
        [0.1429, 0.1429, 0.1429, 0.1429, 0.1429, 0.1429, 0.1429, 0.0000],
        [0.1250, 0.1250, 0.1250, 0.1250, 0.1250, 0.1250, 0.1250, 0.1250]])
```

Residual pathways and **layer normalization**

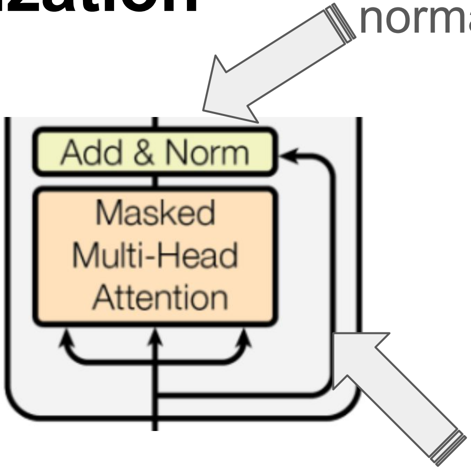
Residual pathways:

- Reduce overfitting in a single step
- stabilizes gradient descent

Layer normalization:

- Guarantees **mean = 0** and **variance = 1**
- reduces **covariance shift** (dependencies between each layer)

Layer normalization



Residual Pathway

Feed forward = Old friend: MLP

You, 5 days ago | 1 author (You)

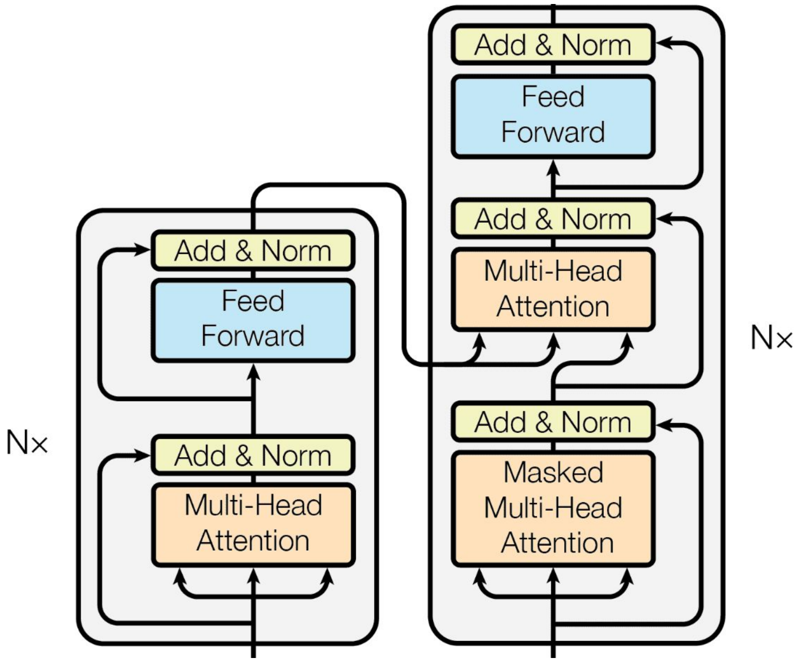
```
class FeedForward(nn.Module):
    """A simple linear layer followed by non-linearity"""

    def __init__(self, no_of_embedding_dims):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(no_of_embedding_dims, 4 * no_of_embedding_dims),
            nn.ReLU(),
            nn.Linear(4 * no_of_embedding_dims, no_of_embedding_dims),
            nn.Dropout(dropout),
        )

    def forward(self, x):
        return self.net(x)
```

How to scale?

Just add more blocks…

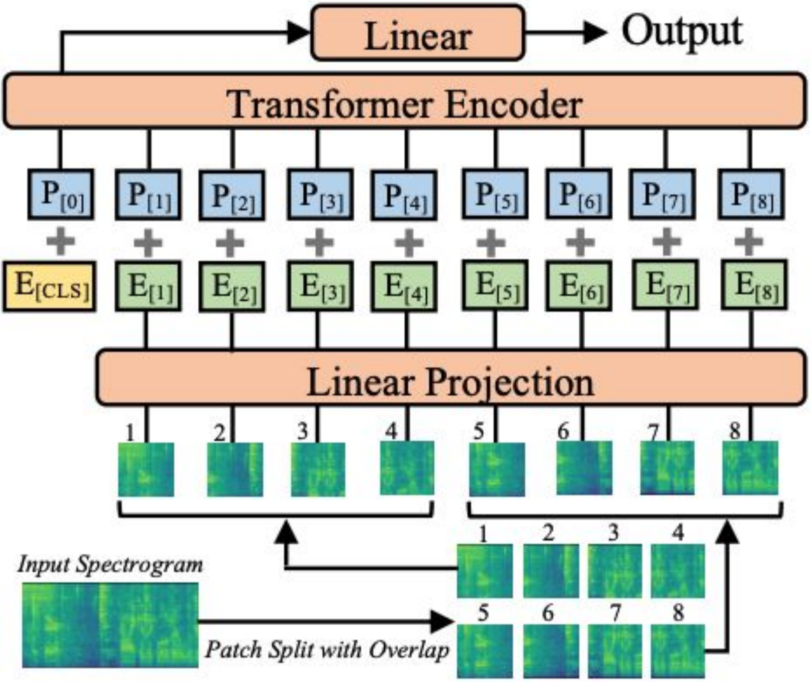


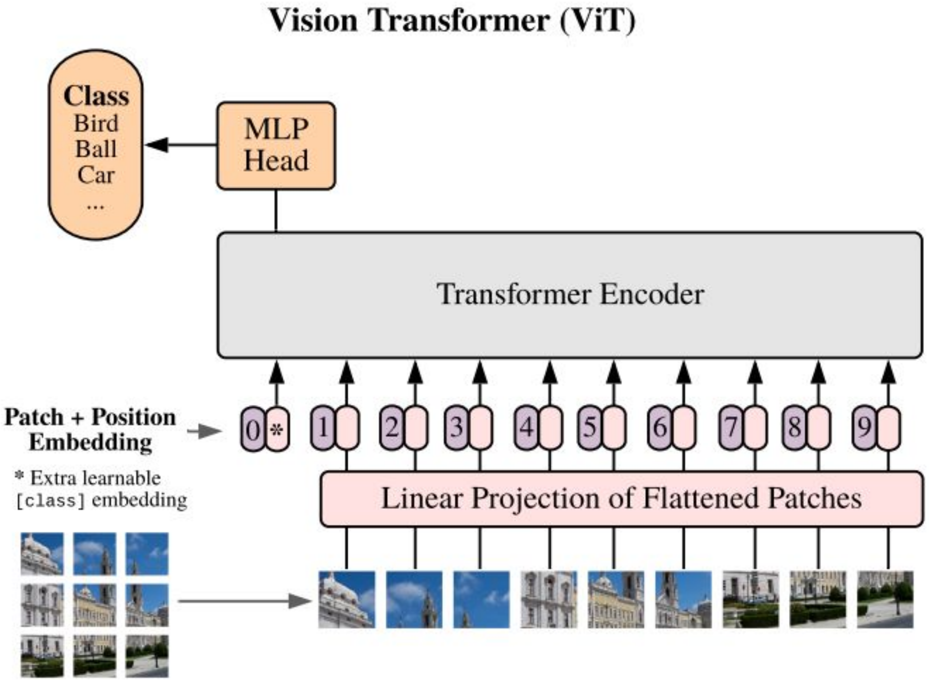
You, 5 days ago | 1 author (You)

```
class BigramLanguageModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.token_embedding_table = nn.Embedding(vocabulary_size, no_of_embedding_dimensions)
        self.position_embedding_table = nn.Embedding(block_size, no_of_embedding_dimensions)
        self.blocks = nn.Sequential(*[Block(no_of_embedding_dimensions, no_of_heads=number_of_heads) for _ in range(number_of_layers)])
        self.layer_norm = nn.LayerNorm(no_of_embedding_dimensions)
        self.language_model_head = nn.Linear(no_of_embedding_dimensions, vocabulary_size)
```

Different input domains

Just patch it, encode it, embed it and throw it in a transformer…

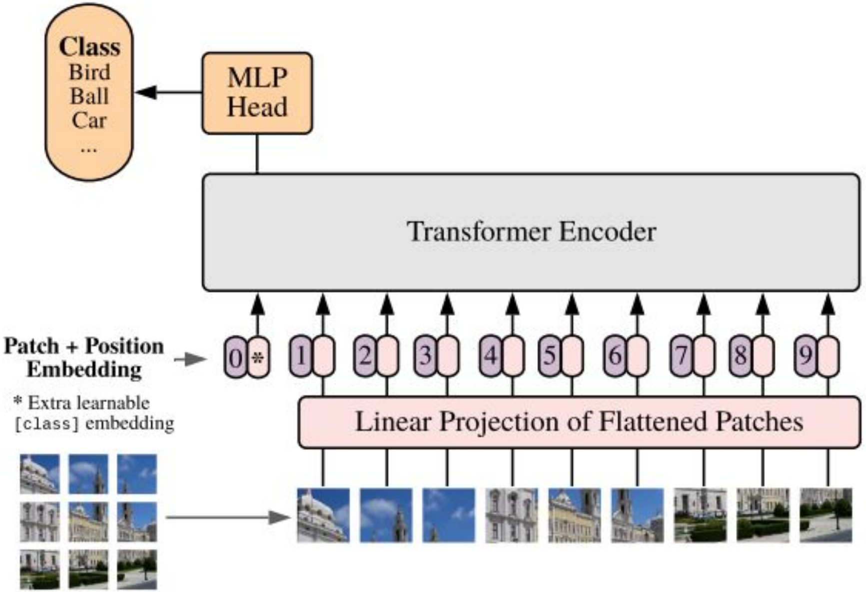




An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*],
Xiaohua Zhai[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,**
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby[*,†]
[*]equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

Vision Transformer (ViT)



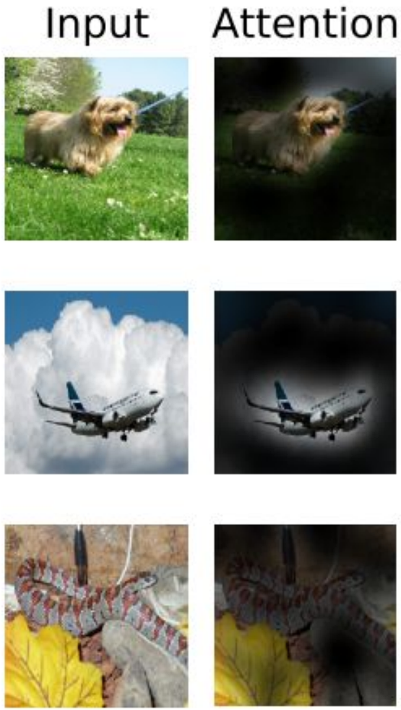
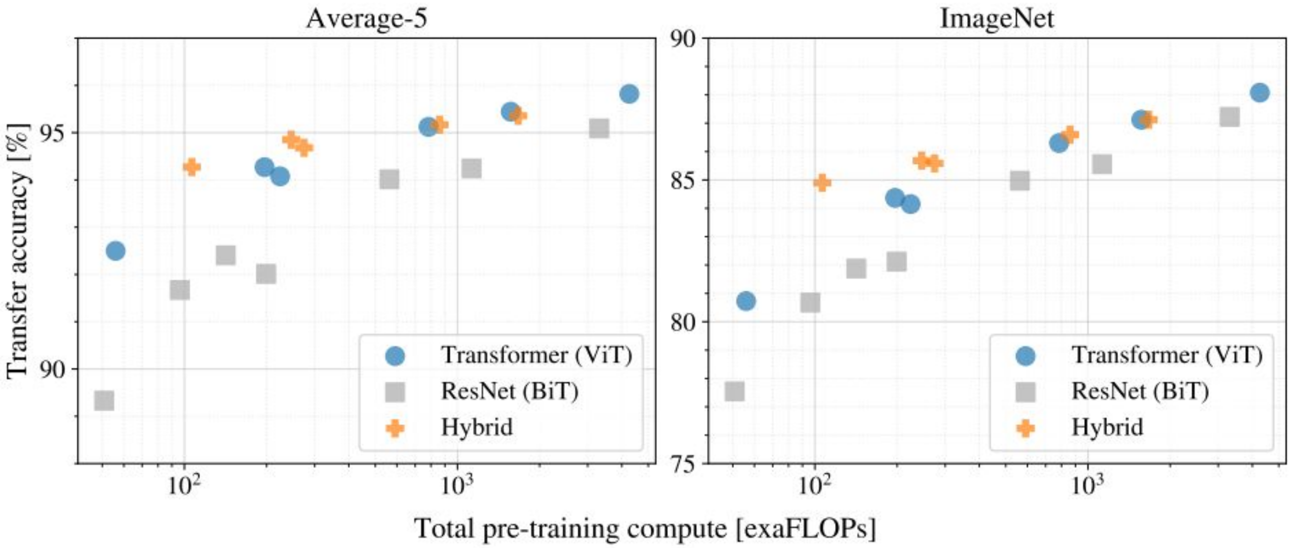


Figure 6: Representative examples of attention from the output token to the input space. See Appendix D.7 for details.

Some results:

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k



Bonus - **Multi-modality**

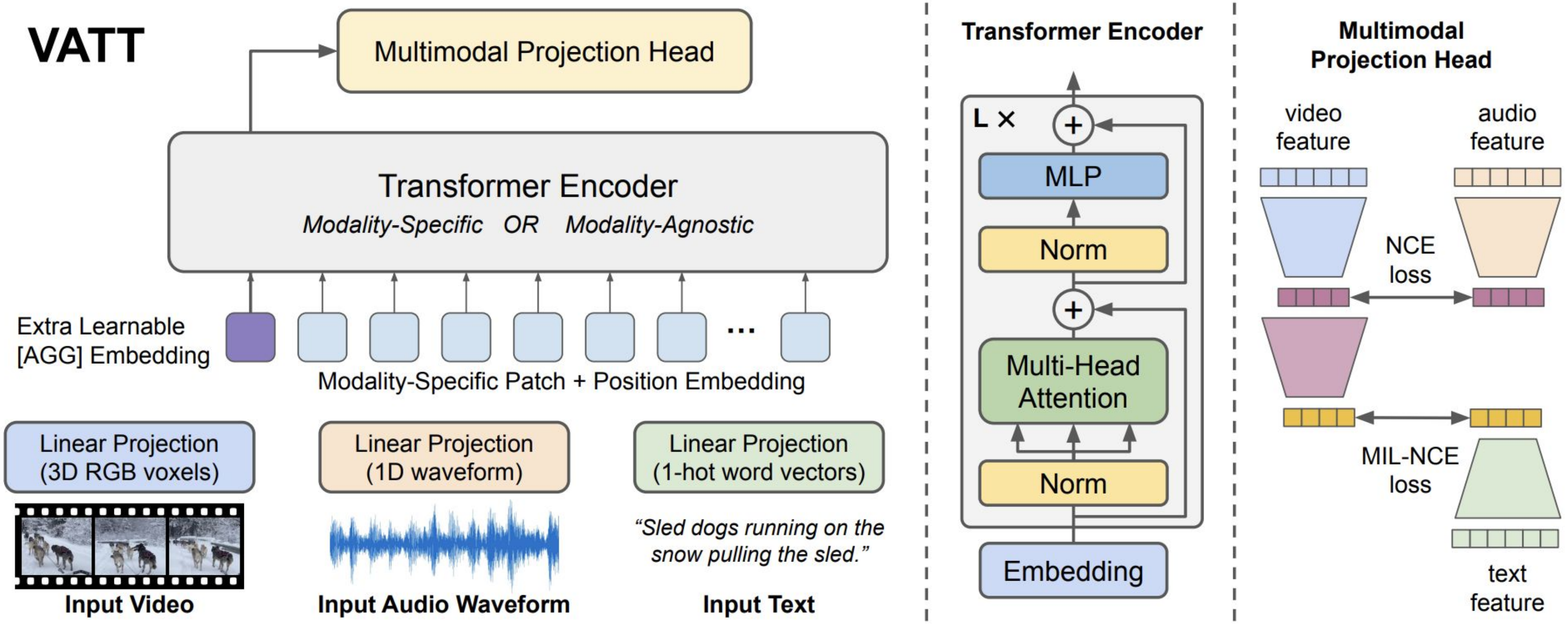


Figure 1. **Overview of the VATT architecture and the self-supervised, multimodal learning strategy**. VATT linearly projects each modality into a feature vector and feeds it into a Transformer encoder. We define a semantically hierarchical common space to account for the granularity of different modalities and employ the noise contrastive estimation to train the model.