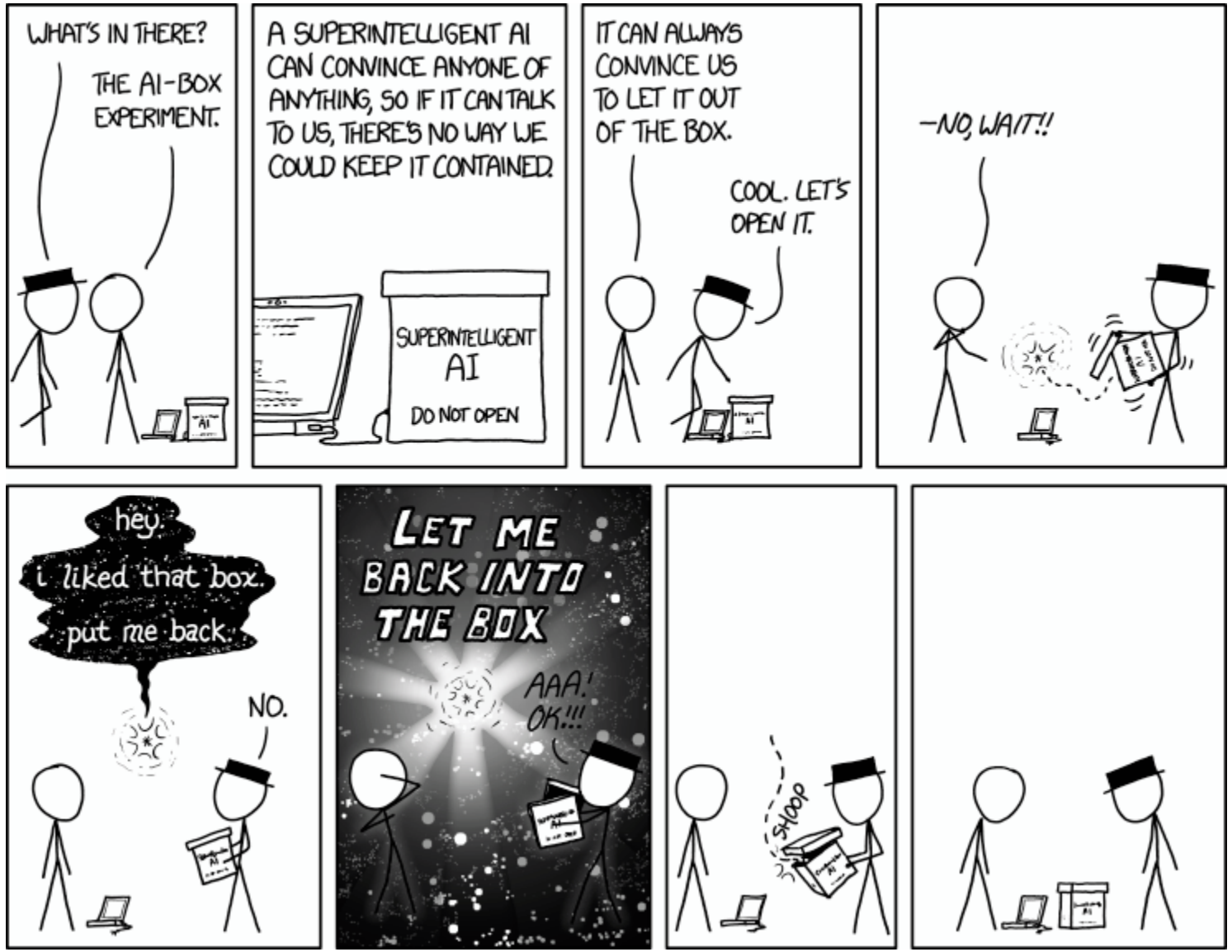
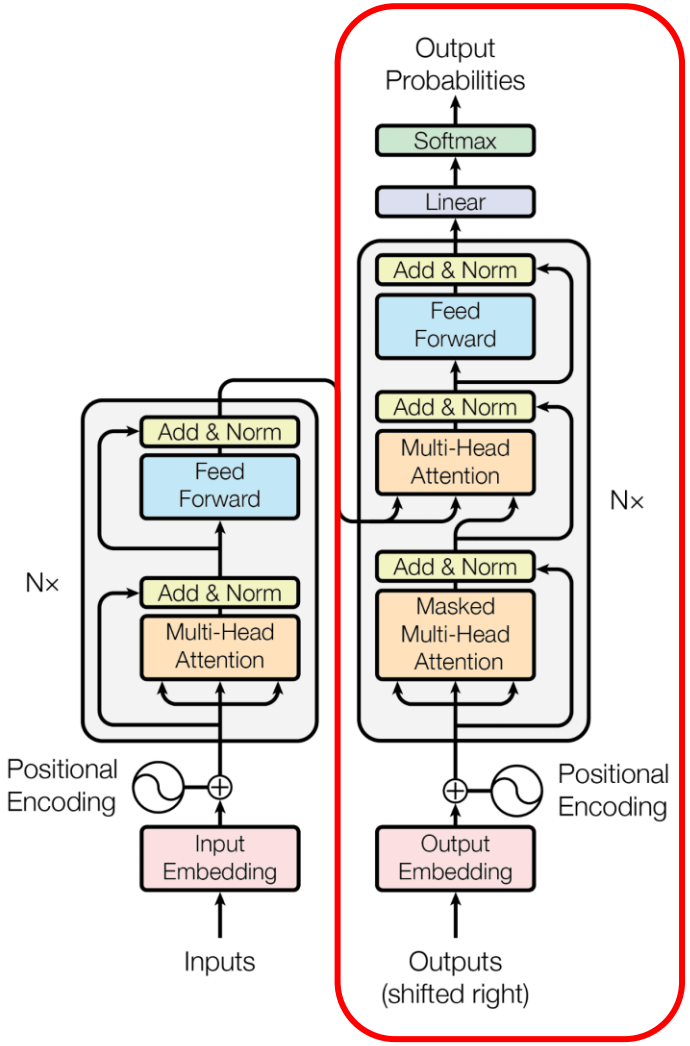
Generative AI: token generation



Xkdc

GPT (Generative Pre-trained Transformer)

	BERT	GPT
Type	Encoder-only	Decoder-only
Directionality	Bidirectional (reads both left and right context)	Autoregressive (left-to-right generation)
Self-Attention	Can attend to all tokens in input	Masked self-attention (only past tokens)
Primary Use	Understanding & classification	Text generation & completion



Ashish Vaswani et al., Attention is all you need. (NIPS'17)

GPT (Generative Pre-trained Transformer)

https://www.3blue1brown.com/lessons/attention

Input layer: token embeddings

Maps tokens to **embedding vectors**: converts tokenized words into dense, continuous vector representations for model processing.

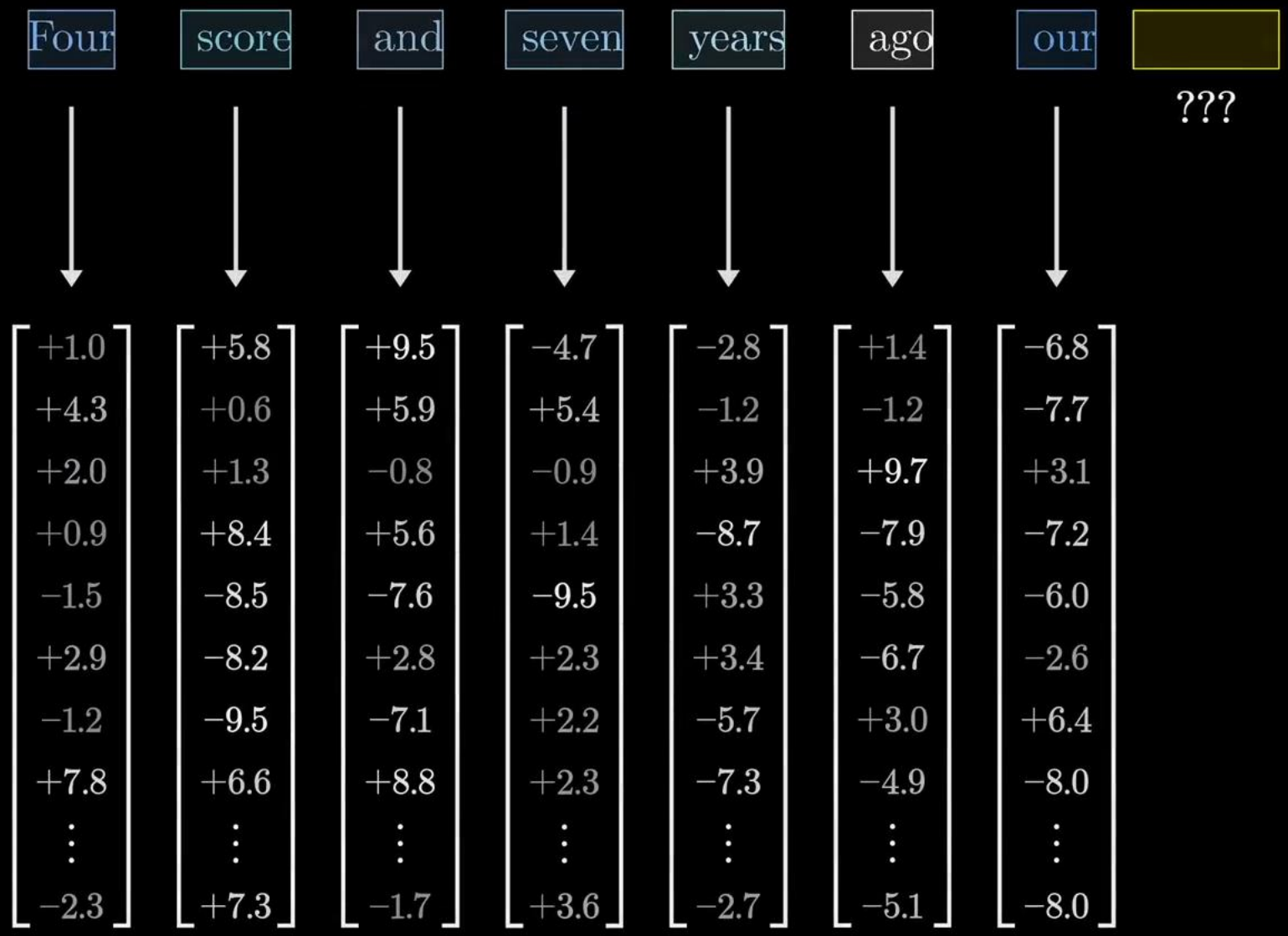
Embeddings are **learned during training**.

Each token is represented as a vector of a **fixed size**.

Since transformers lack recurrence, **positional encodings** are added to embeddings to encode word order.

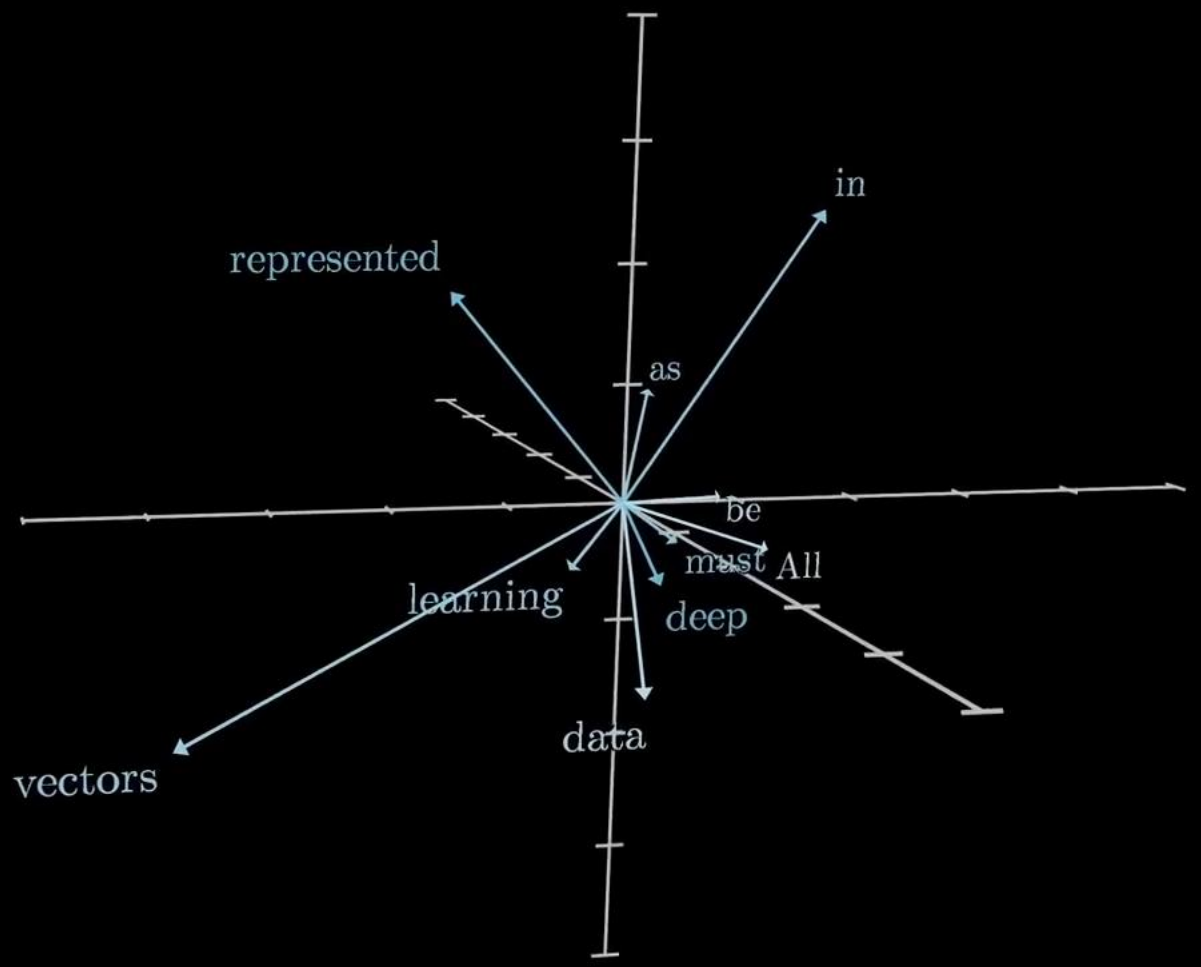
Token embeddings encode semantic meaning for each specific token.

GPT-3 uses a vocabulary of 50,257 tokens.

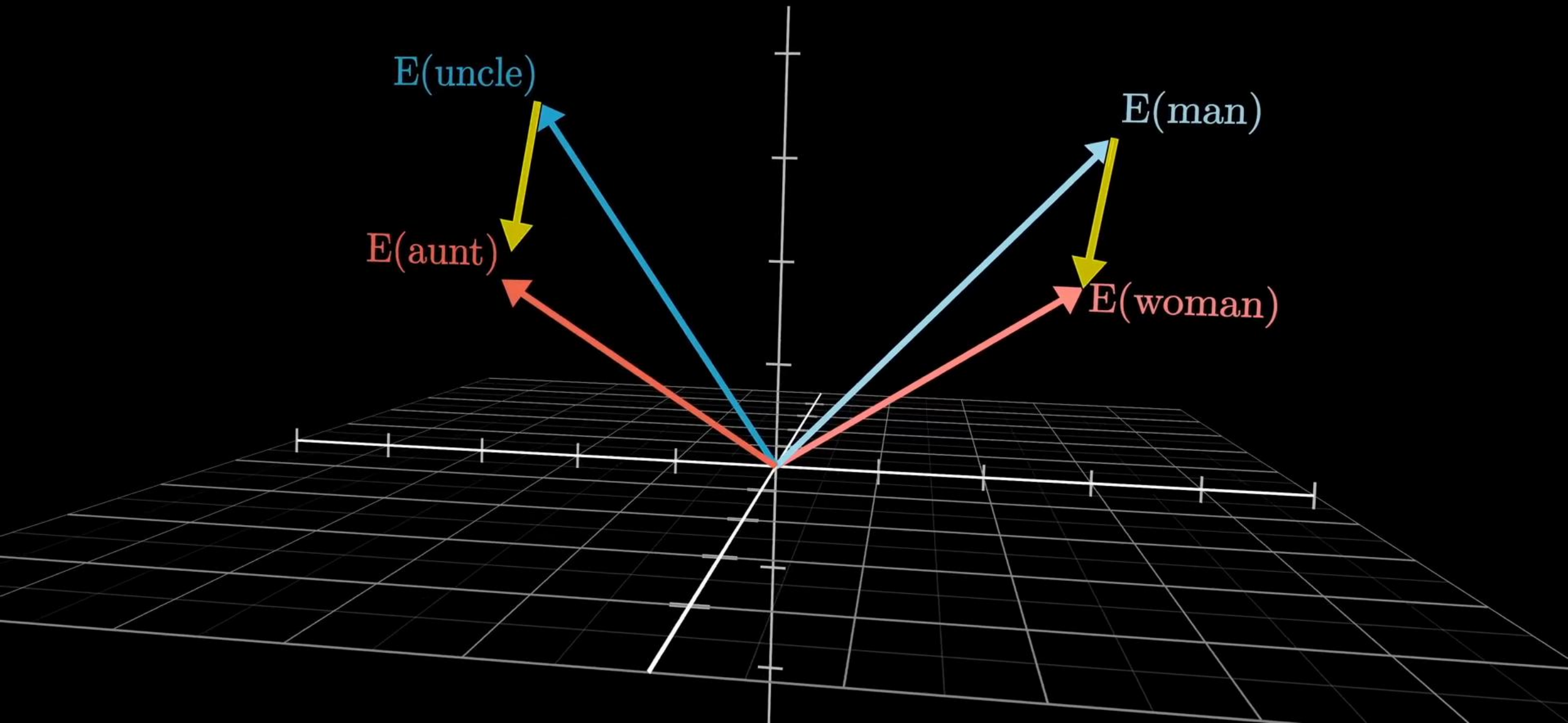

Four
score
and
seven
years
ago
our
???
+1.0 +4.3 +2.0 +0.9 −1.5 +2.9 −1.2 +7.8 ⋮ −2.3
+5.8 +0.6 +1.3 +8.4 −8.5 −8.2 −9.5 +6.6 ⋮ +7.3
+9.5 +5.9 −0.8 +5.6 −7.6 +2.8 −7.1 +8.8 ⋮ −1.7
−4.7 +5.4 −0.9 +1.4 −9.5 +2.3 +2.2 +2.3 ⋮ +3.6
−2.8 −1.2 +3.9 −8.7 +3.3 +3.4 −5.7 −7.3 ⋮ −2.7
+1.4 −1.2 +9.7 −7.9 −5.8 −6.7 +3.0 −4.9 ⋮ −5.1
−6.8 −7.7 +3.1 −7.2 −6.0 −2.6 +6.4 −8.0 ⋮ −8.0

Words $\xrightarrow{\text{“Embedding”}}$ Vectors

All
data
in
deep
learning
must
be
represented
as
vectors



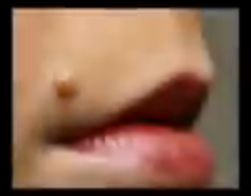
E(aunt) - E(uncle) ≈ E(woman) - E(man)



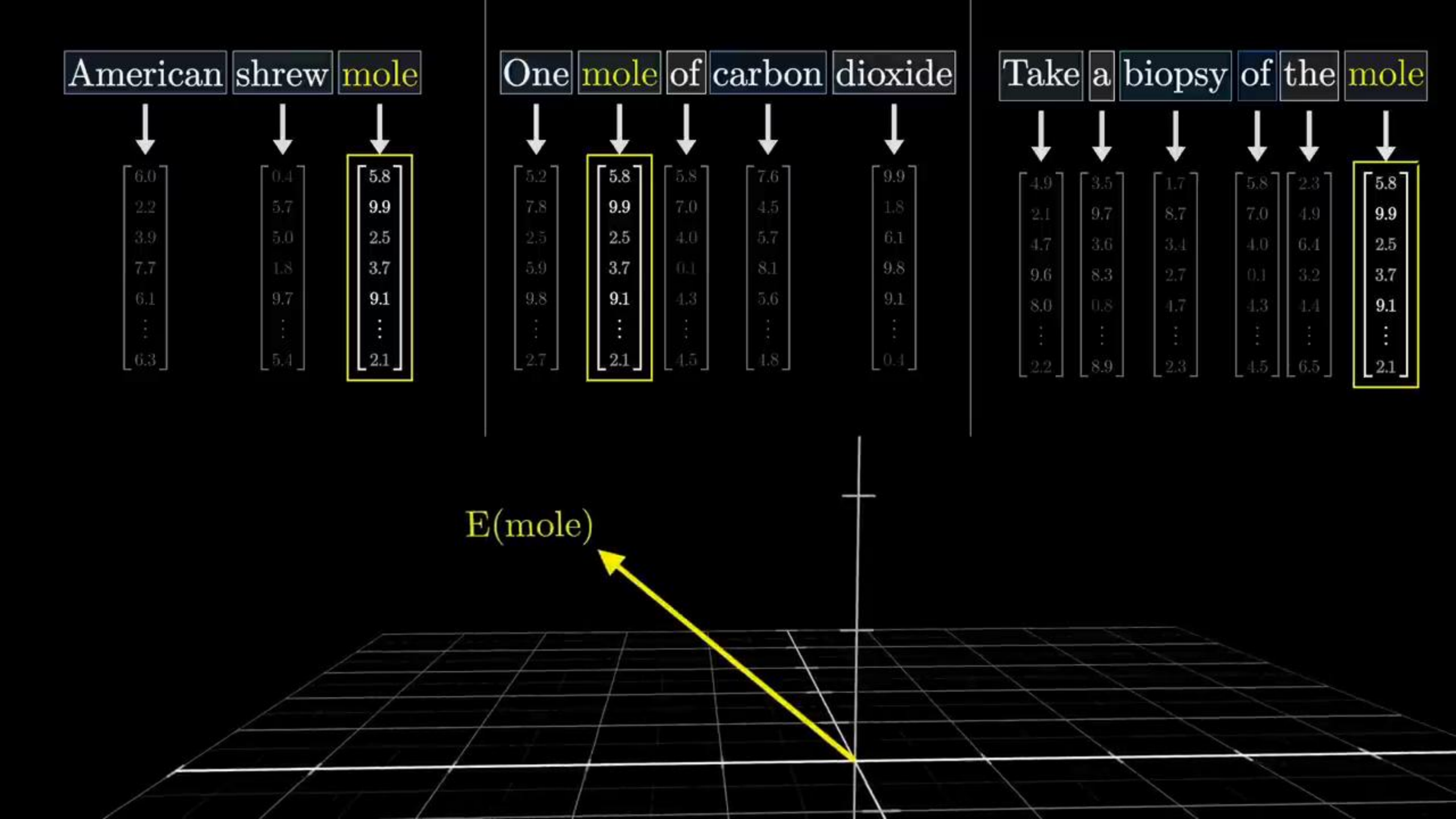
American shrew mole

6.02×10^{23}

One mole of carbon dioxide



Take a biopsy of the mole


American shrew mole
One mole of carbon dioxide
Take a biopsy of the mole
E(mole)

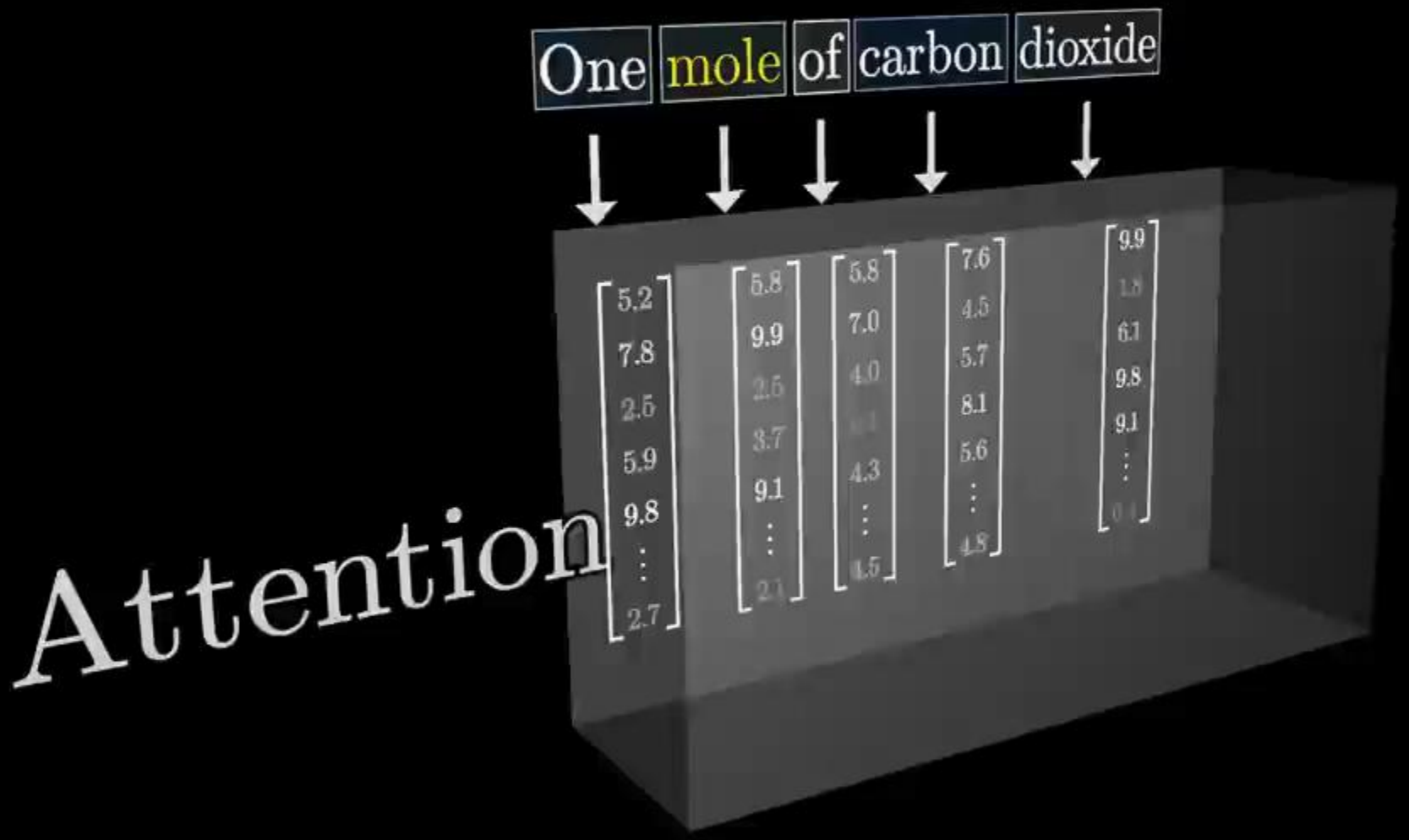
Attention head

Assigns importance to other words: **Each token attends to all other tokens in the sequence**, determining which words influence its meaning.

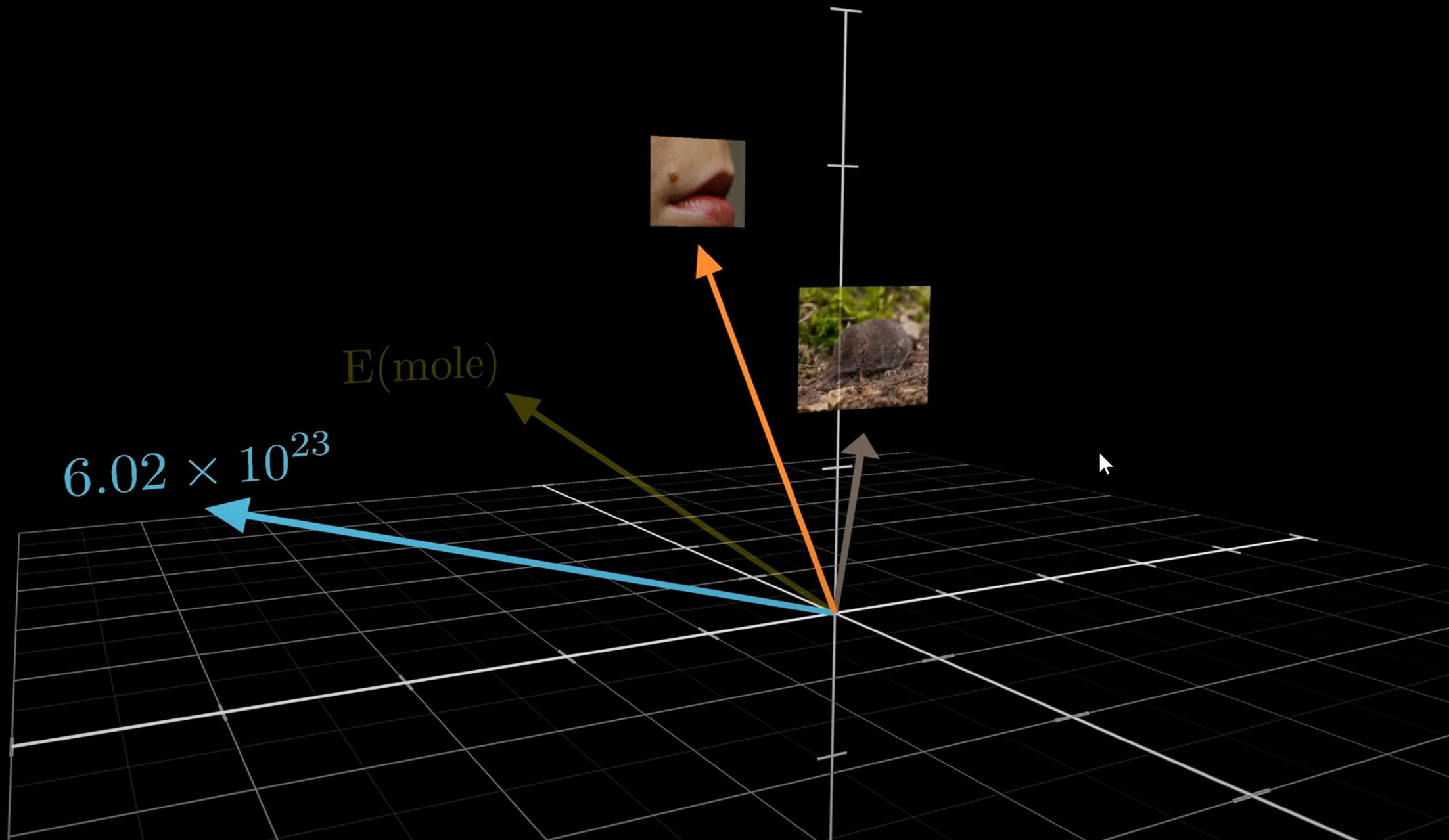
Captures long-range dependencies: Unlike RNNs, which process text sequentially, self-attention can relate words **far apart** in a sentence.

Adjusts **meaning based on context**: The same word can have different meanings depending on surrounding words, and self-attention helps capture this nuance.

Creates **context-aware representations**: By combining information from different tokens, the attention head generates richer word embeddings that reflect their contextual meaning.

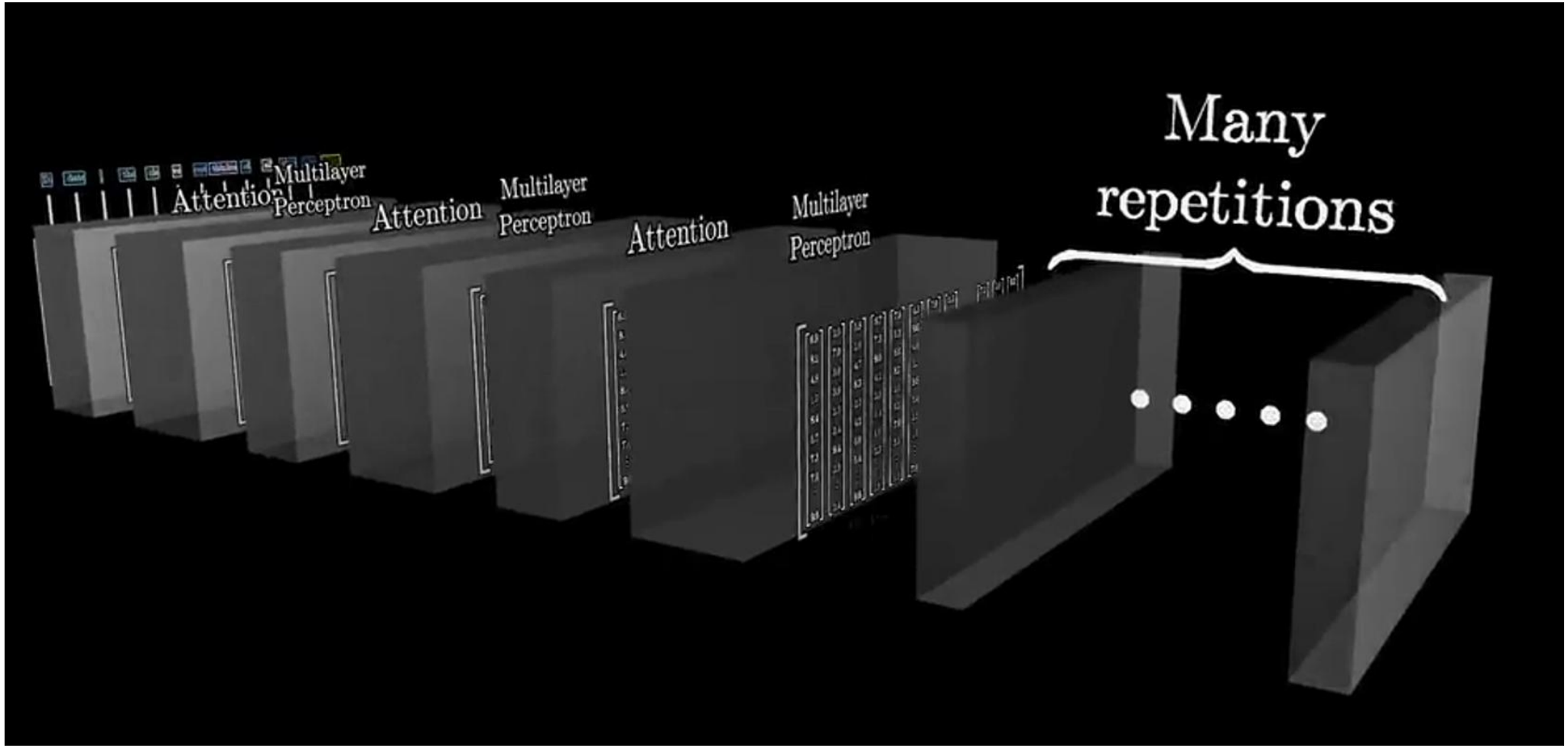

One
mole
of
carbon
dioxide
Attention

One mole of carbon dioxide



Attention head

The transformer architecture consists of multiple layers of (multi-)attention heads.



Attention head

The transformer architecture consists of multiple layers of (multi-)attention heads.

Each one allowed to learn increasingly complex and long-distance token embeddings.

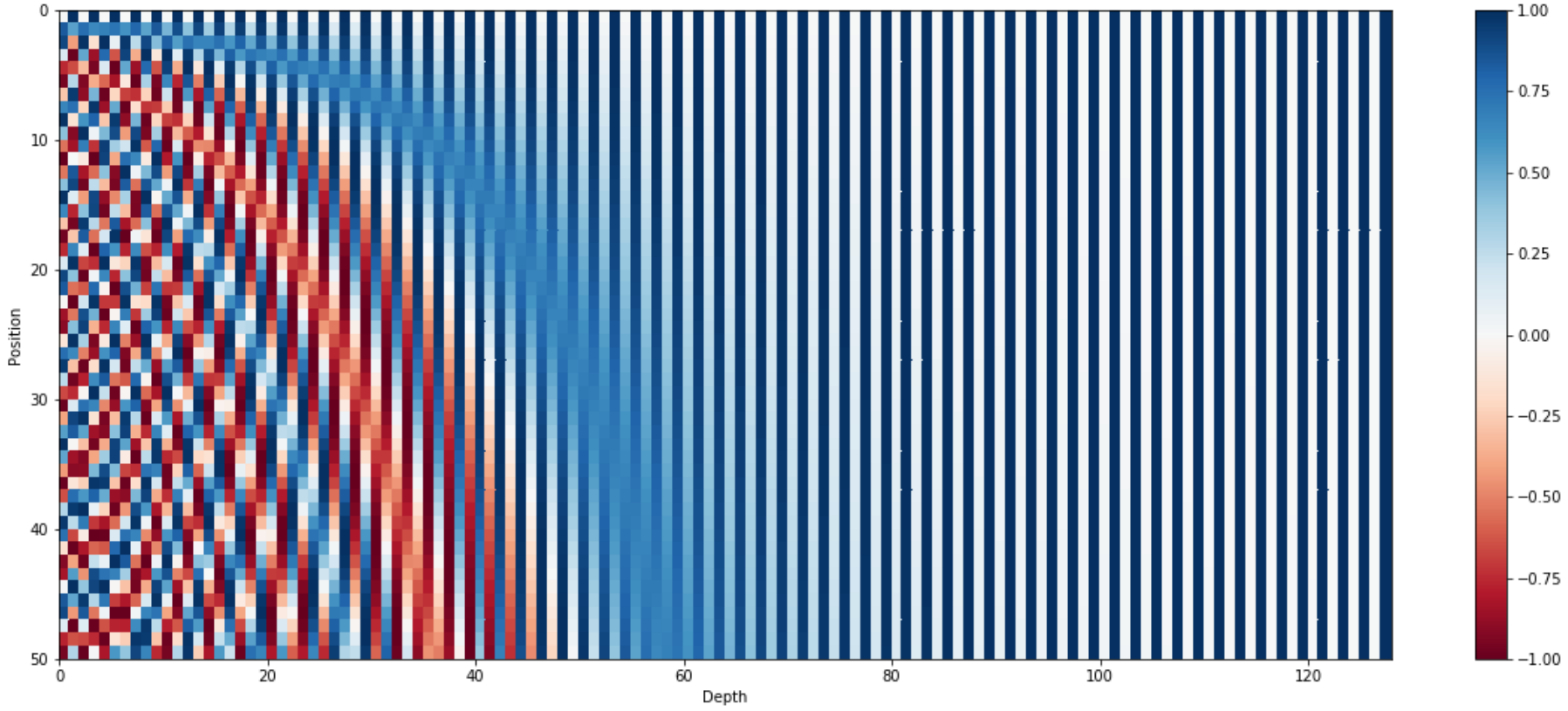
Information flows from left to right (**autoregressive**).

Multi-headed attention

Positional encoding

Transformers lack inherent sequential processing like RNNs.

Positional encoding is **added** to allow the transformer to recognize relative positions and distances between tokens.

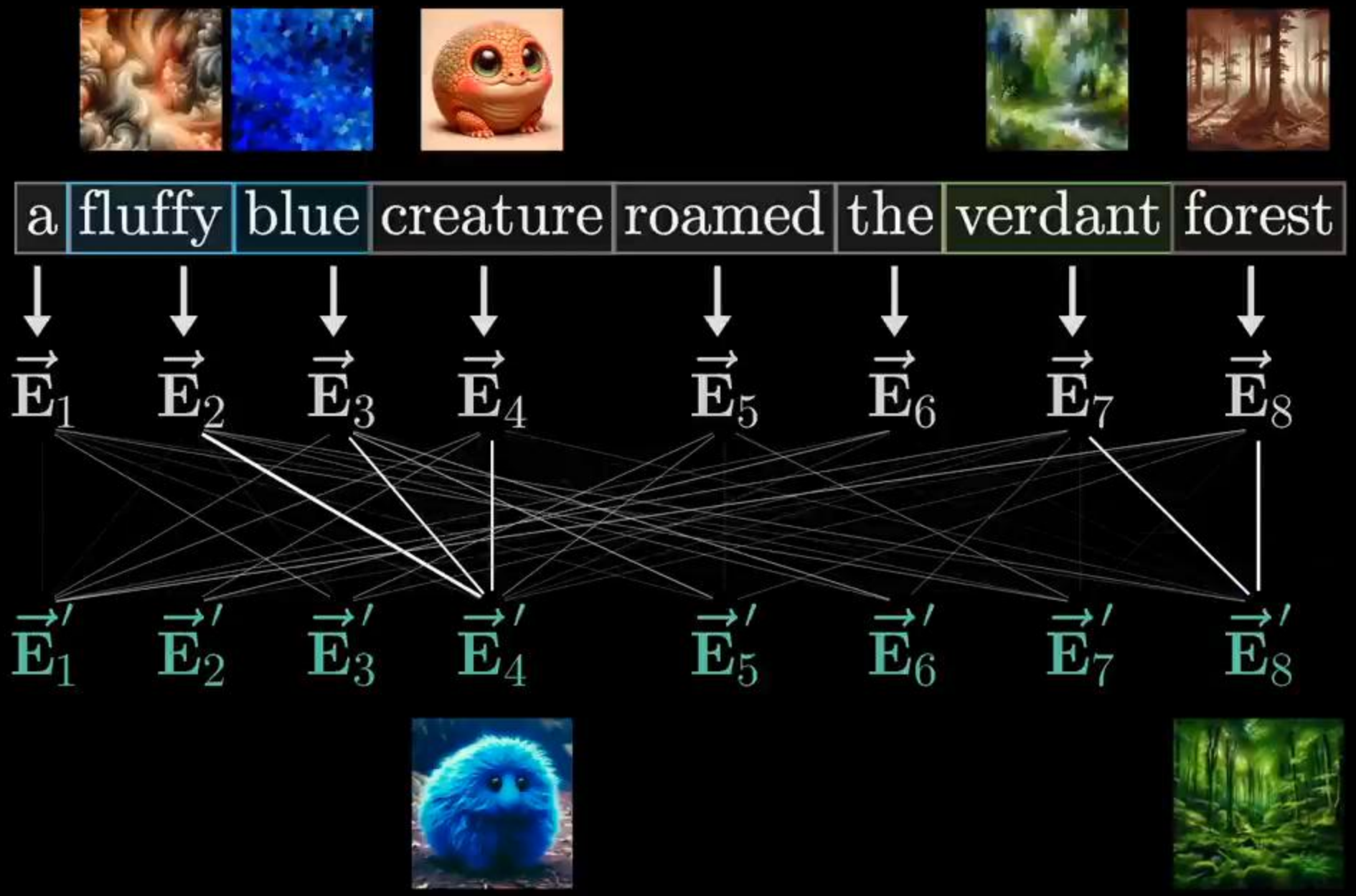


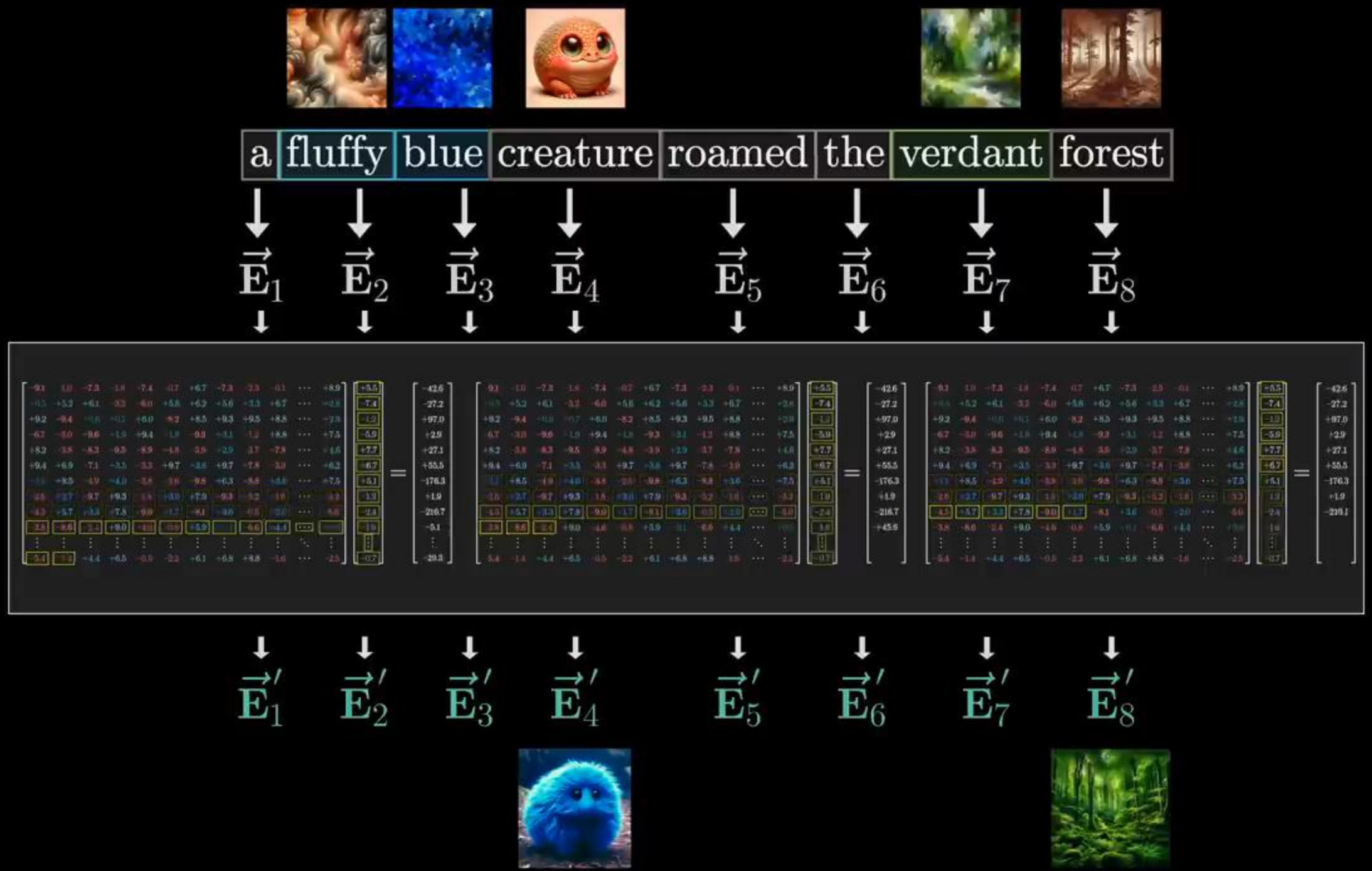
Position encoding

Transformers lack inherent sequential processing like RNNs.

Positional encoding is **added** to allow the transformer to recognize relative positions and distances between tokens.

Instead of appending this positional information to the embedding vectors, the positional information is added to the embedding. This is more efficient, and the model can extract the positional information from the embedding at any time.

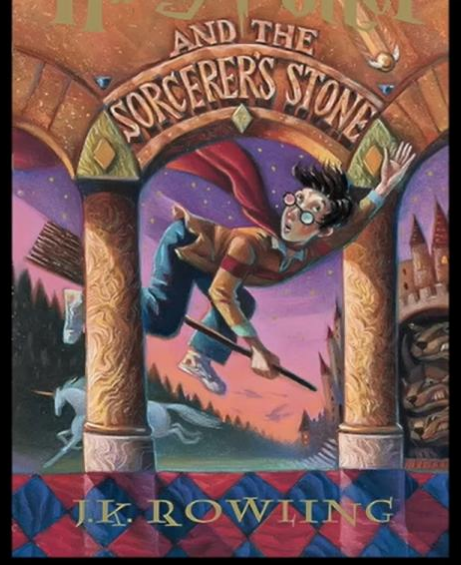

a
fluffy
blue
creature
roamed
the
verdant
forest
$\vec{E}_1$ $\vec{E}_2$ $\vec{E}_3$ $\vec{E}_4$ $\vec{E}_5$ $\vec{E}_6$ $\vec{E}_7$ $\vec{E}_8$
$\vec{E}'_1$ $\vec{E}'_2$ $\vec{E}'_3$ $\vec{E}'_4$ $\vec{E}'_5$ $\vec{E}'_6$ $\vec{E}'_7$ $\vec{E}'_8$


a fluffy blue creature roamed the verdant forest
$\vec{E}_1$ $\vec{E}_2$ $\vec{E}_3$ $\vec{E}_4$ $\vec{E}_5$ $\vec{E}_6$ $\vec{E}_7$ $\vec{E}_8$
$\vec{E}'_1$ $\vec{E}'_2$ $\vec{E}'_3$ $\vec{E}'_4$ $\vec{E}'_5$ $\vec{E}'_6$ $\vec{E}'_7$ $\vec{E}'_8$

... when suddenly they crashed the car into a tree ...

... when suddenly they crashed the car into a tree ...

wizard ... Hogwarts ... Hermione ... Harry



... Queen ... Sussex ... William ... Harry

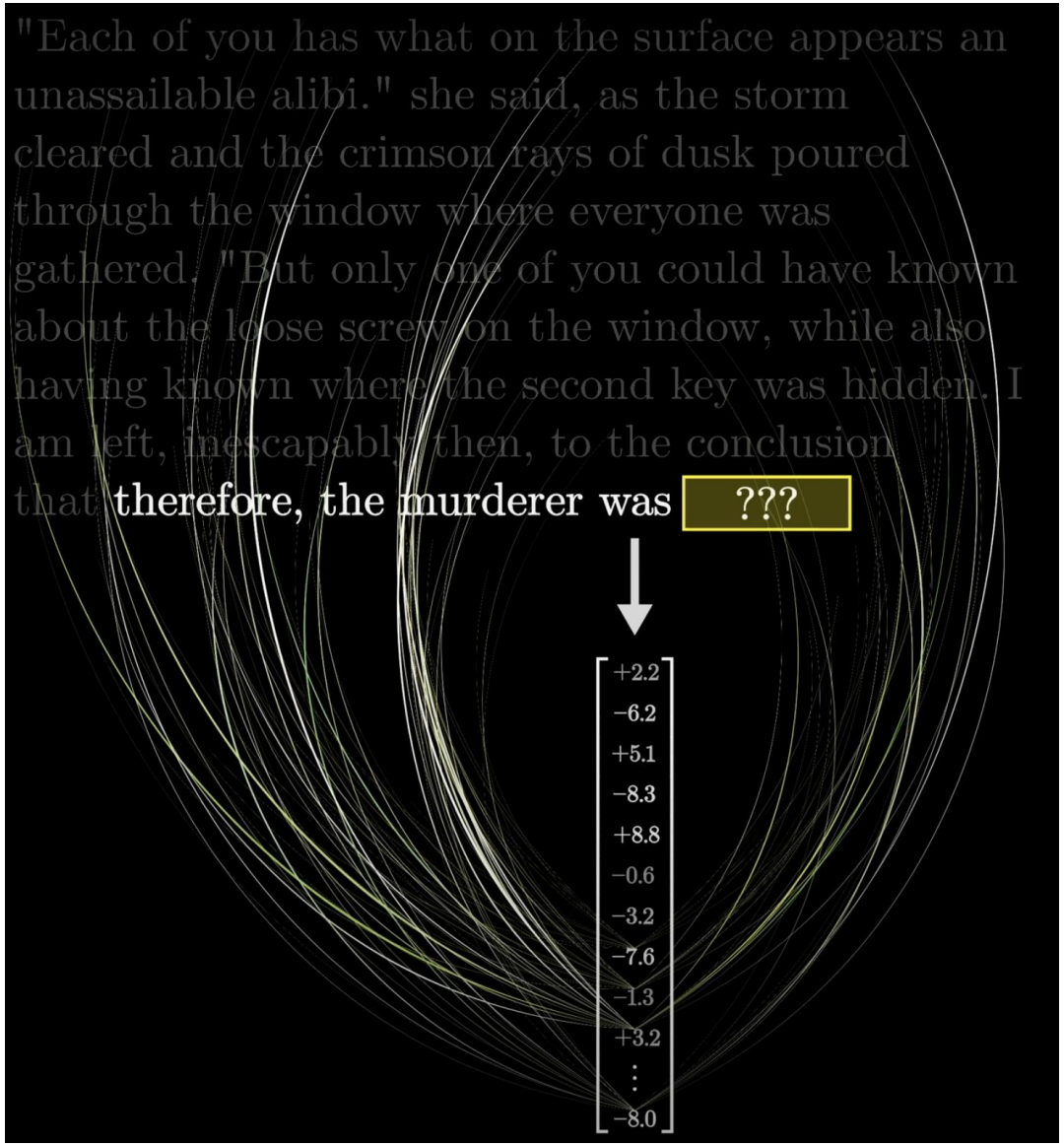
Attention head

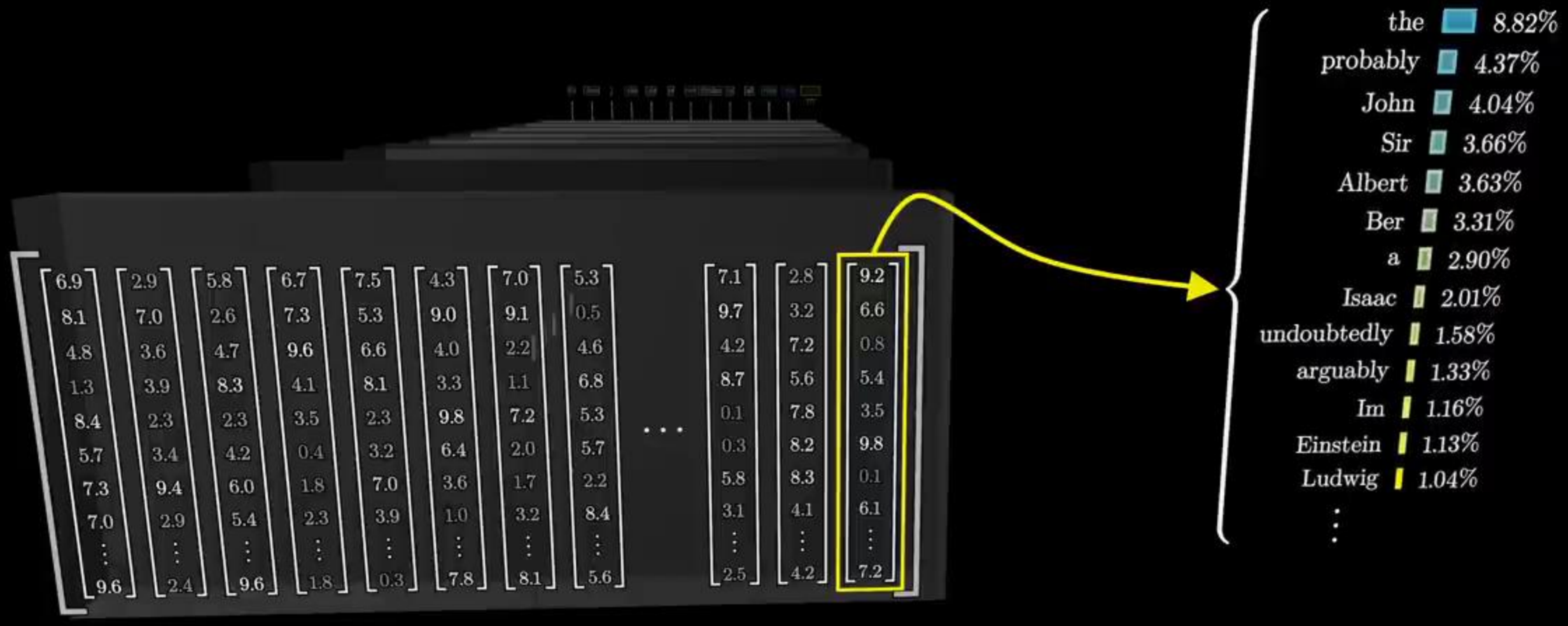
The transformer architecture consists of multiple layers of (multi-)attention heads.

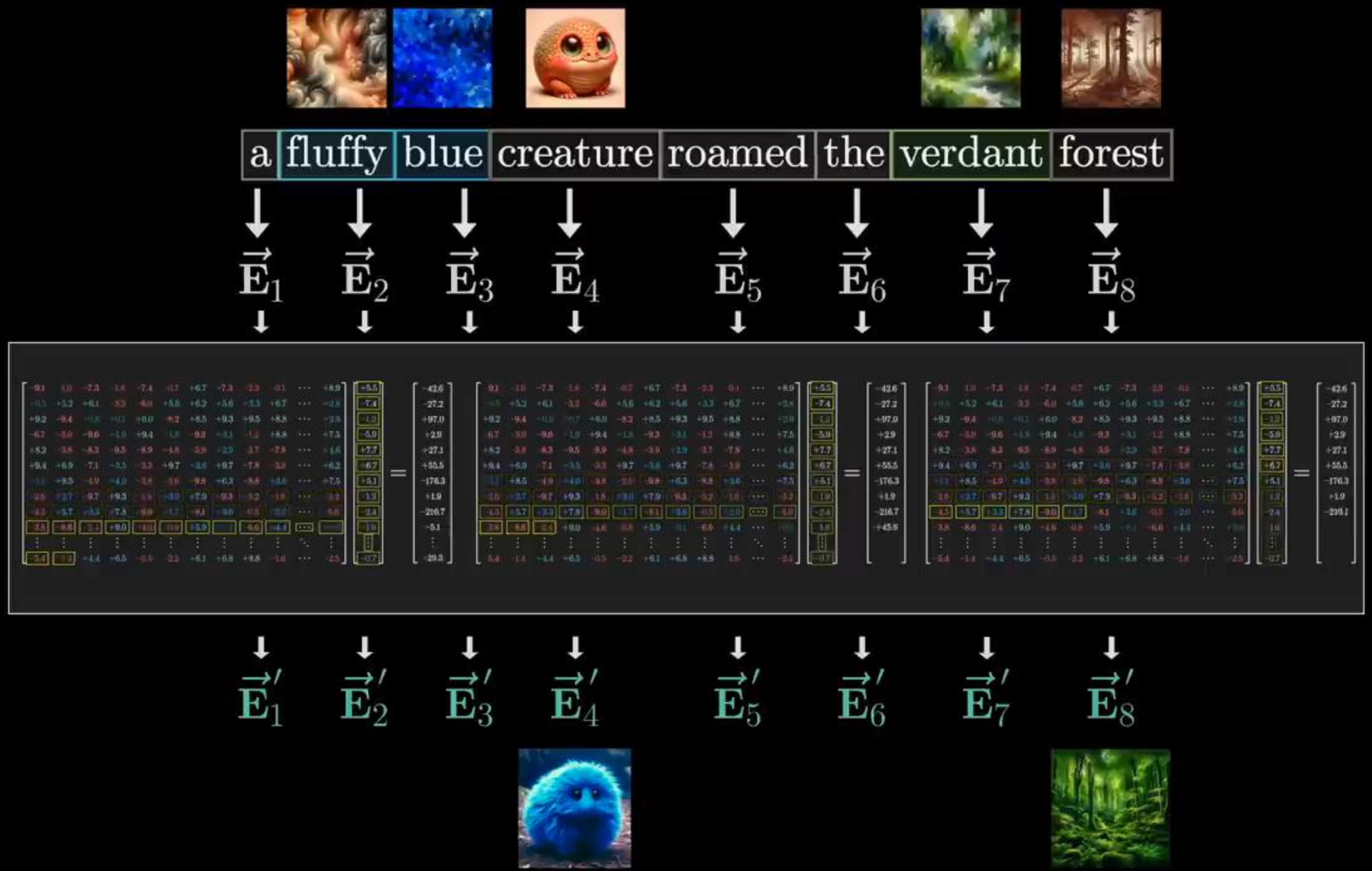
Each one allowed to learn increasingly complex and long-distance token embeddings.

All token embeddings (first layer) are processed through all the attention layers and **enriched with all relevant information in the context window**.

The **final embedding of the last token** then contains all information required to predict the next token.




the 8.82%
probably 4.37%
John 4.04%
Sir 3.66%
Albert 3.63%
Ber 3.31%
a 2.90%
Isaac 2.01%
undoubtedly 1.58%
arguably 1.33%
Im 1.16%
Einstein 1.13%
Ludwig 1.04%


a fluffy blue creature roamed the verdant forest
$\vec{E}_1$ $\vec{E}_2$ $\vec{E}_3$ $\vec{E}_4$ $\vec{E}_5$ $\vec{E}_6$ $\vec{E}_7$ $\vec{E}_8$
$\vec{E}'_1$ $\vec{E}'_2$ $\vec{E}'_3$ $\vec{E}'_4$ $\vec{E}'_5$ $\vec{E}'_6$ $\vec{E}'_7$ $\vec{E}'_8$

Attention head

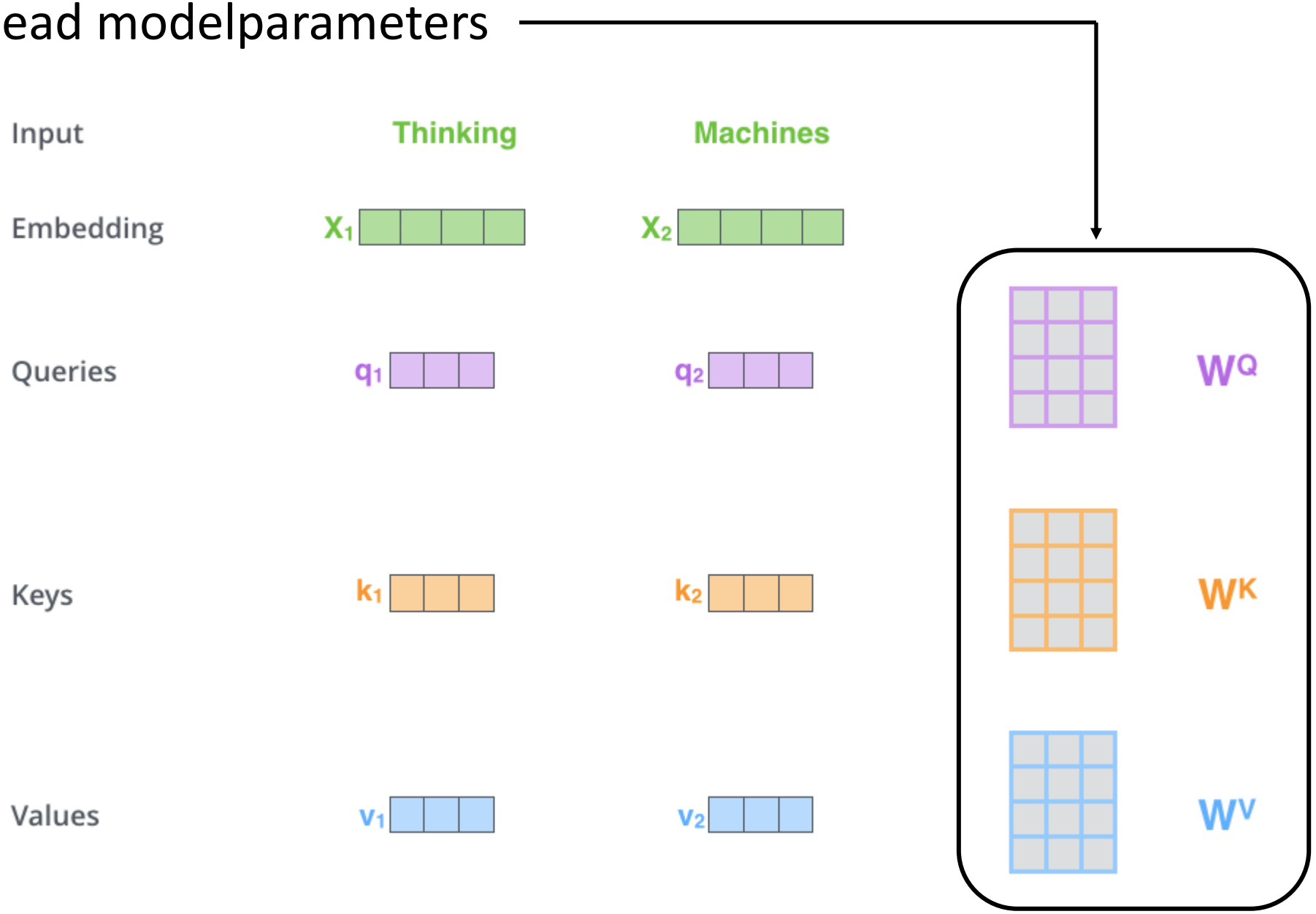
Query (Q), Key (K), and Value (V) Matrices: These are the three essential components in an attention mechanism, derived from the input embeddings through learned weight matrices.

Query (Q): Represents what the current token is searching for in other tokens, helping determine relevance in context.

Key (K): Encodes the characteristics of each token, allowing it to be compared against queries to compute attention scores.

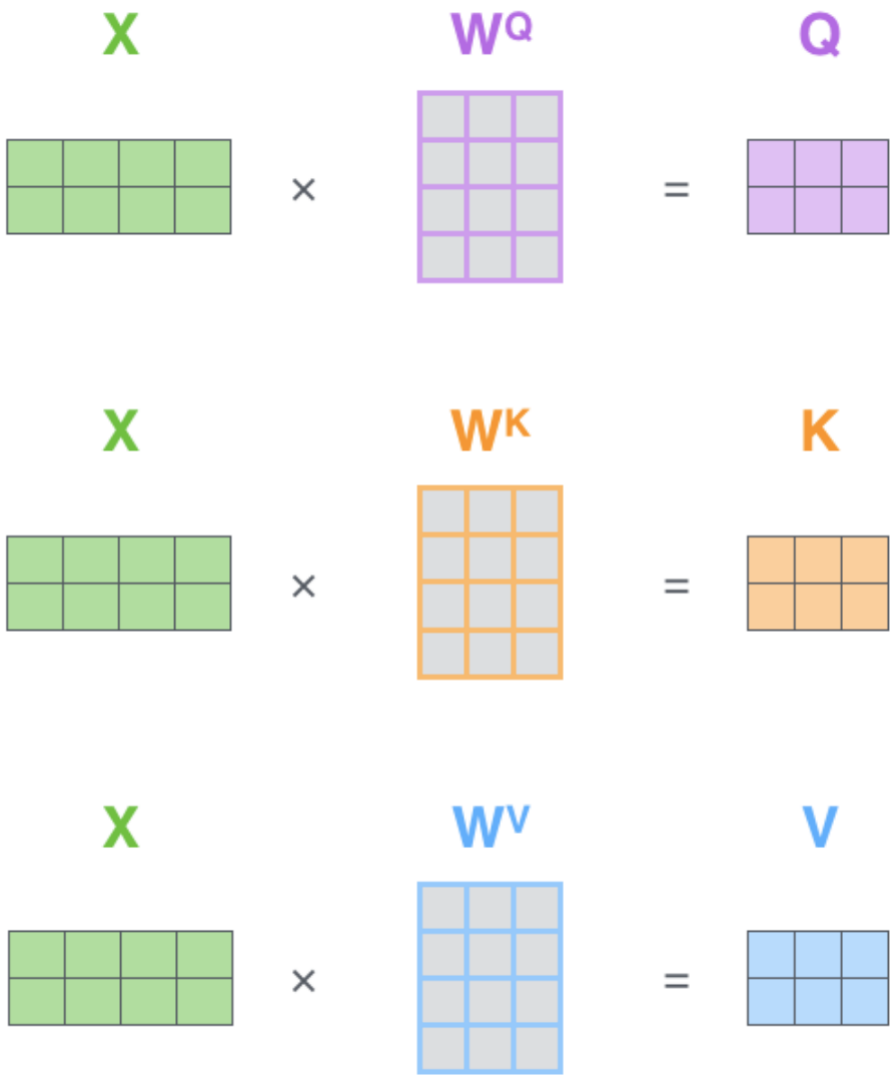
Value (V): Contains the actual information of each token, weighted by the attention scores before being aggregated into the final output.

Attention head modelparameters



jalammar.github.io/illustrated-transformer/

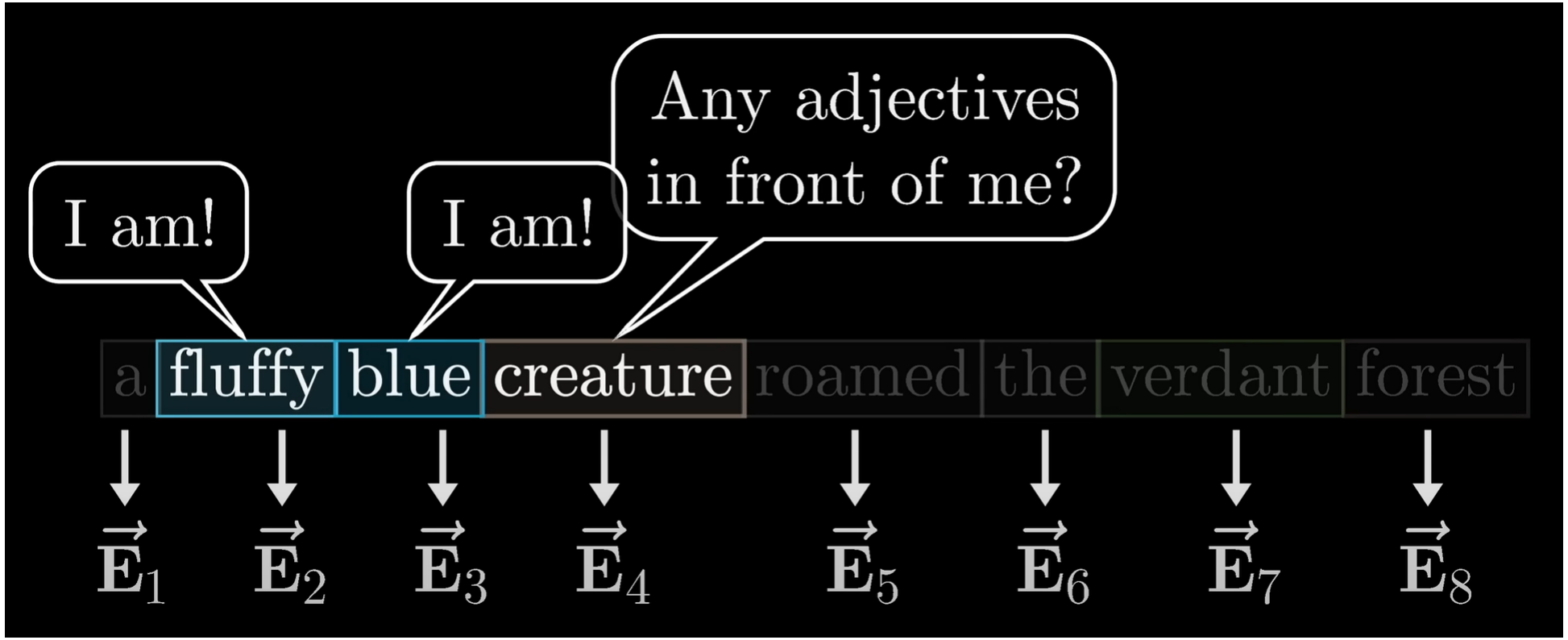
Attention head modelparameters



jalammar.github.io/illustrated-transformer/

Attention head

We can look at the query as asking a question and the key as an answer.



Attention head

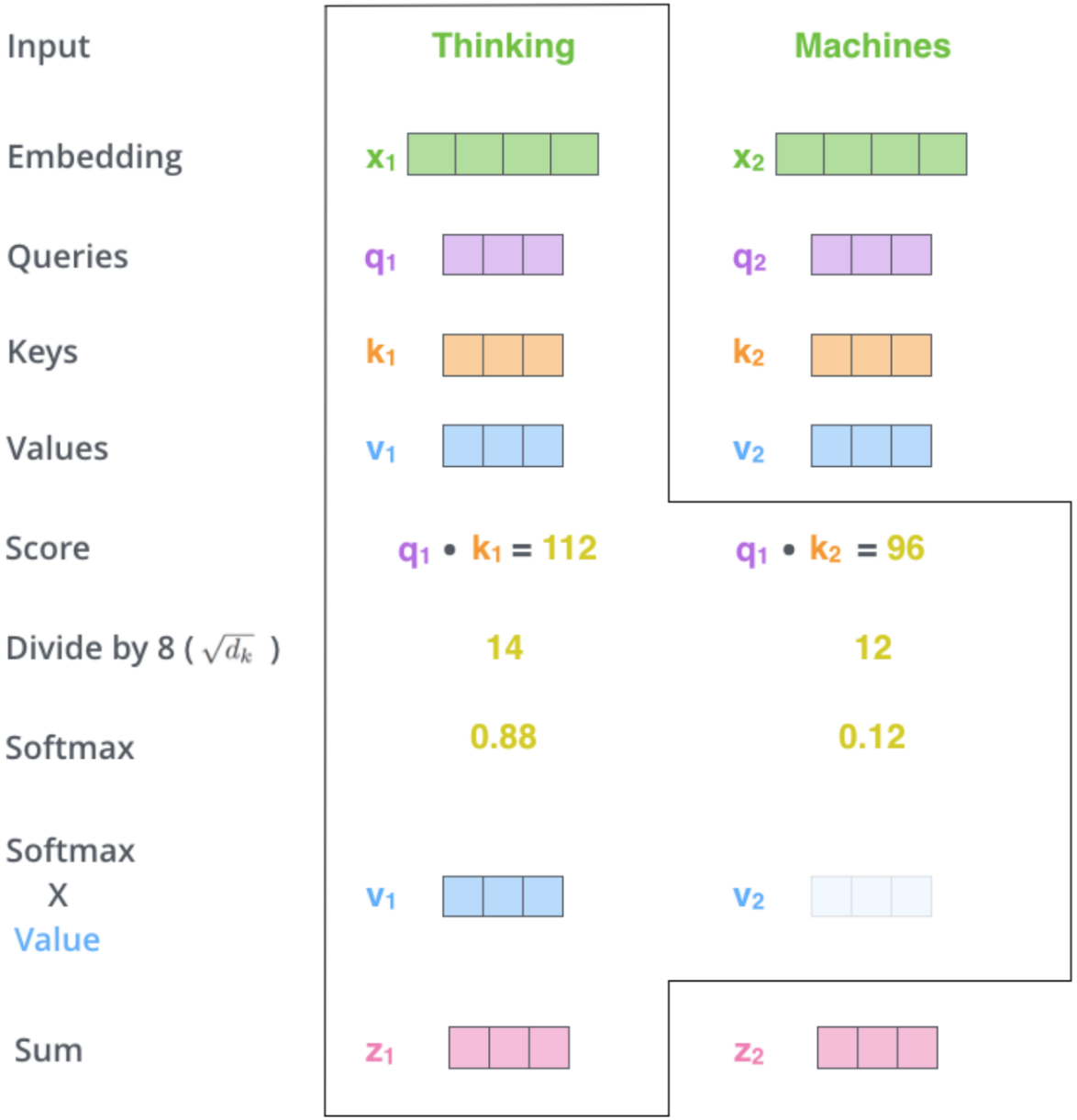
Q (Query), K (Key), and V (Value) matrices are learned linear transformations applied to input embeddings, enabling the model to project inputs into different representational subspaces.

The Q and K matrices are used to compute **attention scores** via dot products, which are then normalized (typically with softmax) to determine the relevance of each token to others.

The attention scores are used to **weight the V (Value) matrix**, aggregating relevant information from other tokens to form a context-aware representation.

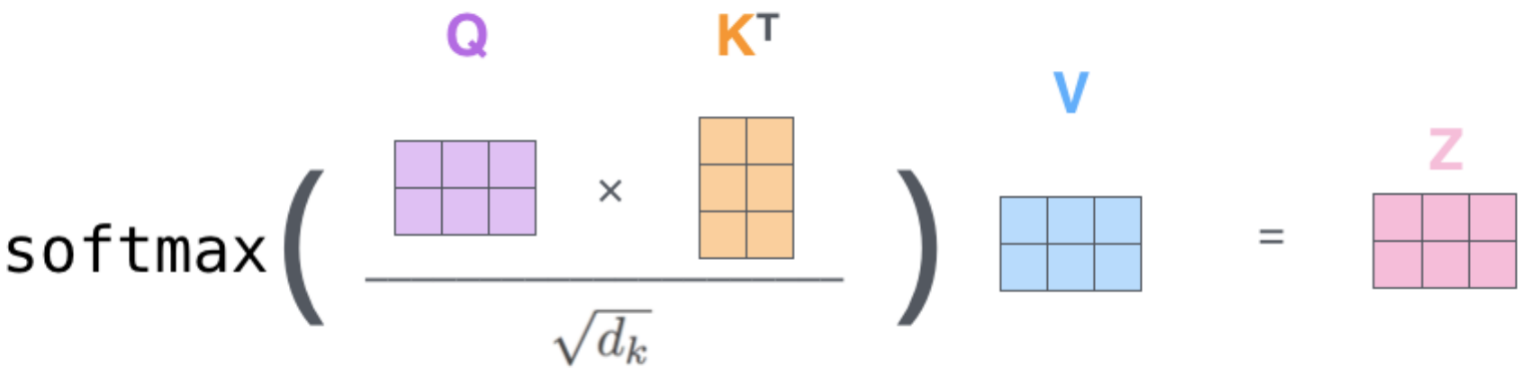
Together, these matrices enable the **self-attention mechanism** in transformer architectures, allowing the model to capture and utilize dependencies between tokens regardless of their distance in the sequence.

Attention head flow



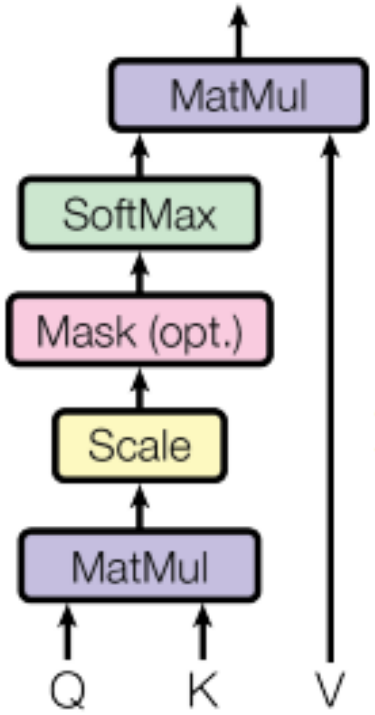
jalammar.github.io/illustrated-transformer/

Attention head flow



jalammar.github.io/illustrated-transformer/

Attention head flow



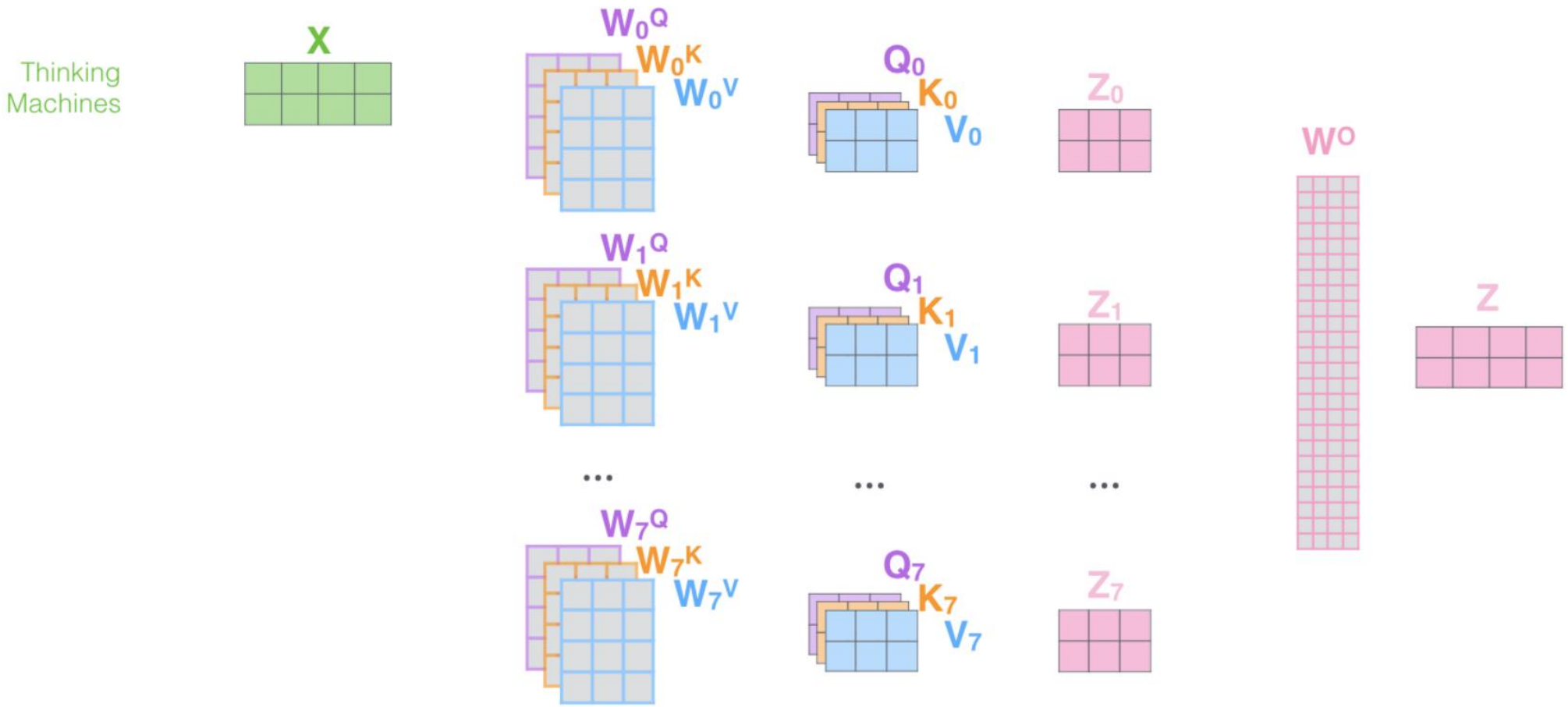
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Ashish Vaswani et al., Attention is all you need. (NIPS'17)

Multi attention flow

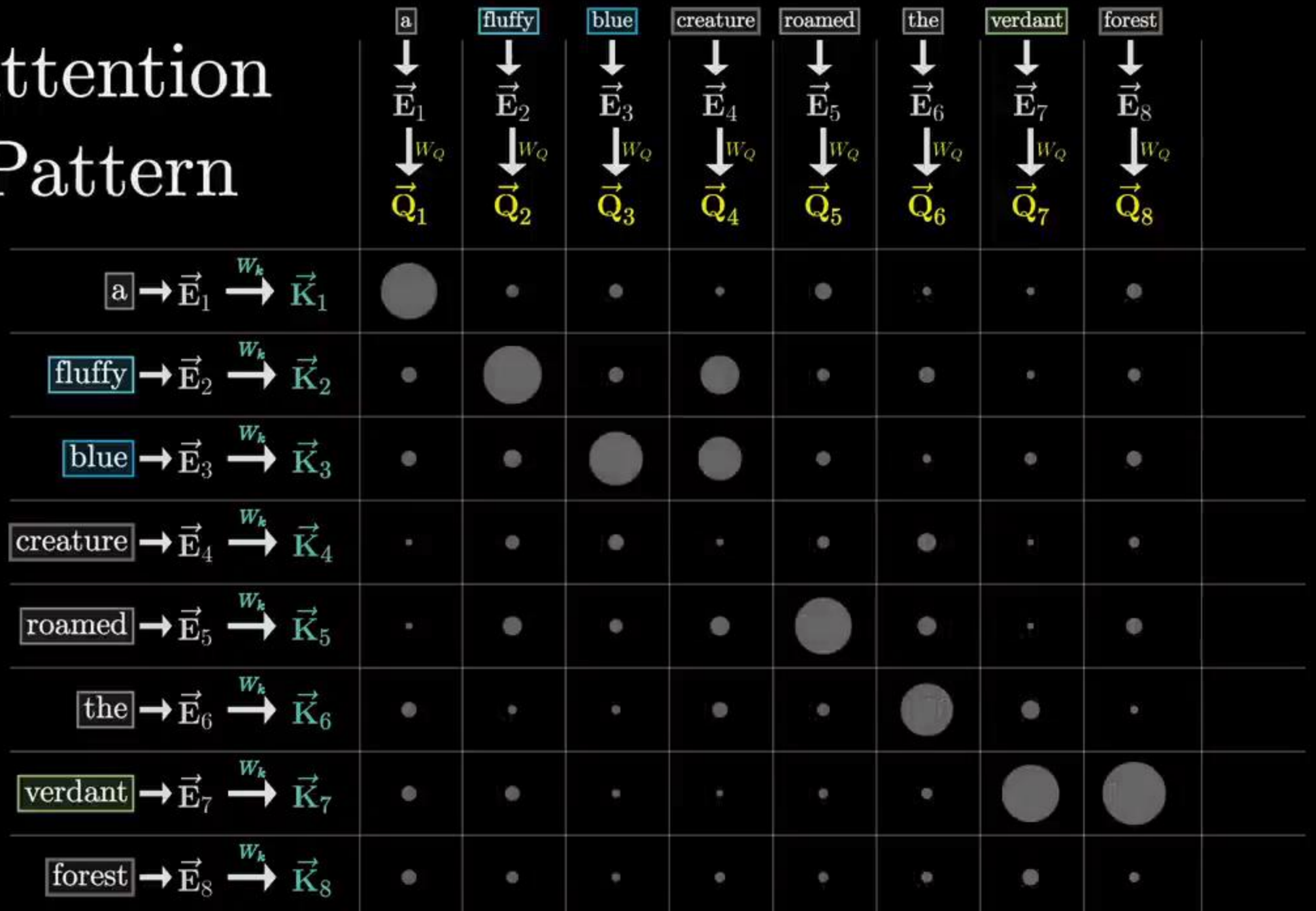


jalammar.github.io/illustrated-transformer/

a
fluffy
blue
creature
roamed
the
verdant
forest
$\vec{E}_1$ $\vec{E}_2$ $\vec{E}_3$ $\vec{E}_4$ $\vec{E}_5$ $\vec{E}_6$ $\vec{E}_7$ $\vec{E}_8$
W_Q
$\vec{Q}_1$ $\vec{Q}_2$ $\vec{Q}_3$ $\vec{Q}_4$ $\vec{Q}_5$ $\vec{Q}_6$ $\vec{Q}_7$ $\vec{Q}_8$
W_k
$\vec{K}_1$ $\vec{K}_2$ $\vec{K}_3$ $\vec{K}_4$ $\vec{K}_5$ $\vec{K}_6$ $\vec{K}_7$ $\vec{K}_8$

a
fluffy
blue
creature
roamed
the
verdant
forest
$\vec{E}_1$ $\vec{E}_2$ $\vec{E}_3$ $\vec{E}_4$ $\vec{E}_5$ $\vec{E}_6$ $\vec{E}_7$ $\vec{E}_8$
W_Q
$\vec{Q}_1$ $\vec{Q}_2$ $\vec{Q}_3$ $\vec{Q}_4$ $\vec{Q}_5$ $\vec{Q}_6$ $\vec{Q}_7$ $\vec{Q}_8$
W_K
$\vec{K}_1$ $\vec{K}_2$ $\vec{K}_3$ $\vec{K}_4$ $\vec{K}_5$ $\vec{K}_6$ $\vec{K}_7$ $\vec{K}_8$
softmax
0.00
0.00
0.00
0.00
0.00
1.00
0.00
0.00

Attention Pattern



Value matrix

W_V

	a $\vec{E}_1$	fluffy $\vec{E}_2$	blue $\vec{E}_3$	creature $\vec{E}_4$	roamed $\vec{E}_5$	the $\vec{E}_6$	verdant $\vec{E}_7$	forest $\vec{E}_8$
a $\rightarrow \vec{E}_1 \xrightarrow{W_V} \vec{V}_1$				0.00 $\vec{V}_1$				
fluffy $\rightarrow \vec{E}_2 \xrightarrow{W_V} \vec{V}_2$				0.42 $\vec{V}_2$				
blue $\rightarrow \vec{E}_3 \xrightarrow{W_V} \vec{V}_3$				0.58 $\vec{V}_3$				
creature $\rightarrow \vec{E}_4 \xrightarrow{W_V} \vec{V}_4$				0.00 $\vec{V}_4$				
roamed $\rightarrow \vec{E}_5 \xrightarrow{W_V} \vec{V}_5$				0.00 $\vec{V}_5$				
the $\rightarrow \vec{E}_6 \xrightarrow{W_V} \vec{V}_6$				0.00 $\vec{V}_6$				
verdant $\rightarrow \vec{E}_7 \xrightarrow{W_V} \vec{V}_7$				0.00 $\vec{V}_7$				
forest $\rightarrow \vec{E}_8 \xrightarrow{W_V} \vec{V}_8$				0.00 $\vec{V}_8$				

	a	fluffy	blue	creature	roamed	the	verdant	forest
	$\vec{E}_1$	$\vec{E}_2$	$\vec{E}_3$	$\vec{E}_4$	$\vec{E}_5$	$\vec{E}_6$	$\vec{E}_7$	$\vec{E}_8$
a $\rightarrow \vec{E}_1 \xrightarrow{W_V} \vec{V}_1$				$0.00\ \vec{V}_1$				
				$+$				
fluffy $\rightarrow \vec{E}_2 \xrightarrow{W_V} \vec{V}_2$				$0.42\ \vec{V}_2$				
				$+$				
blue $\rightarrow \vec{E}_3 \xrightarrow{W_V} \vec{V}_3$				$0.58\ \vec{V}_3$				
				$+$				
creature $\rightarrow \vec{E}_4 \xrightarrow{W_V} \vec{V}_4$				$0.00\ \vec{V}_4$				
				$+$				
roamed $\rightarrow \vec{E}_5 \xrightarrow{W_V} \vec{V}_5$				$0.00\ \vec{V}_5$				
				$+$				
the $\rightarrow \vec{E}_6 \xrightarrow{W_V} \vec{V}_6$				$0.00\ \vec{V}_6$				
				$+$				
verdant $\rightarrow \vec{E}_7 \xrightarrow{W_V} \vec{V}_7$				$0.00\ \vec{V}_7$				
				$+$				
forest $\rightarrow \vec{E}_8 \xrightarrow{W_V} \vec{V}_8$				$0.00\ \vec{V}_8$				

$$\| $$

$$\Delta\vec{E}_4$$

creature

$$\downarrow$$

$$\vec{E}_4$$

$$+$$

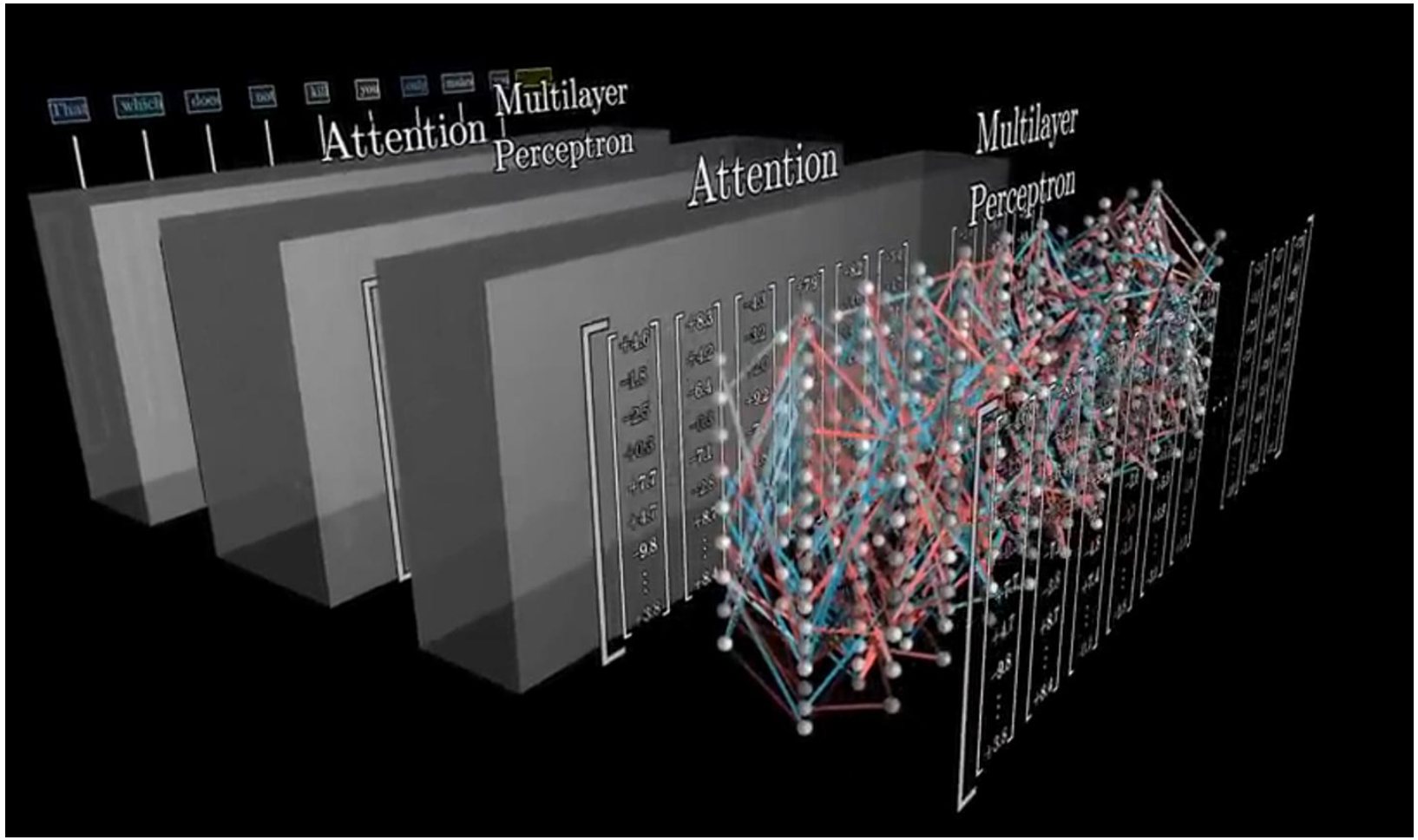
$$\Delta\vec{E}_4$$

$$\|$$

$$\vec{E}_4{}'$$

MLP (Feedforward) Layers in the Transformer

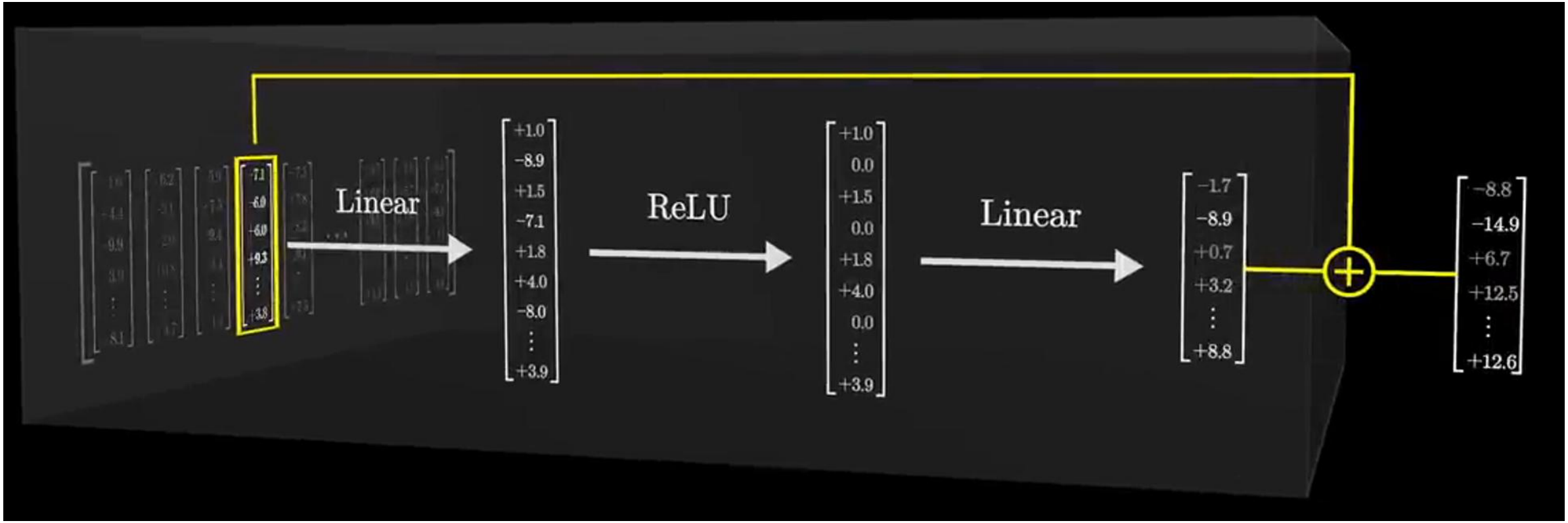
Each Transformer layer contains an MLP (Multi-Layer Perceptron), applied independently to each token after attention.



MLP (Feedforward) Layers in the Transformer

Each Transformer layer contains an MLP (Multi-Layer Perceptron), applied independently to each token after attention.

It consists of two fully connected (dense) layers with a non-linearity in between.



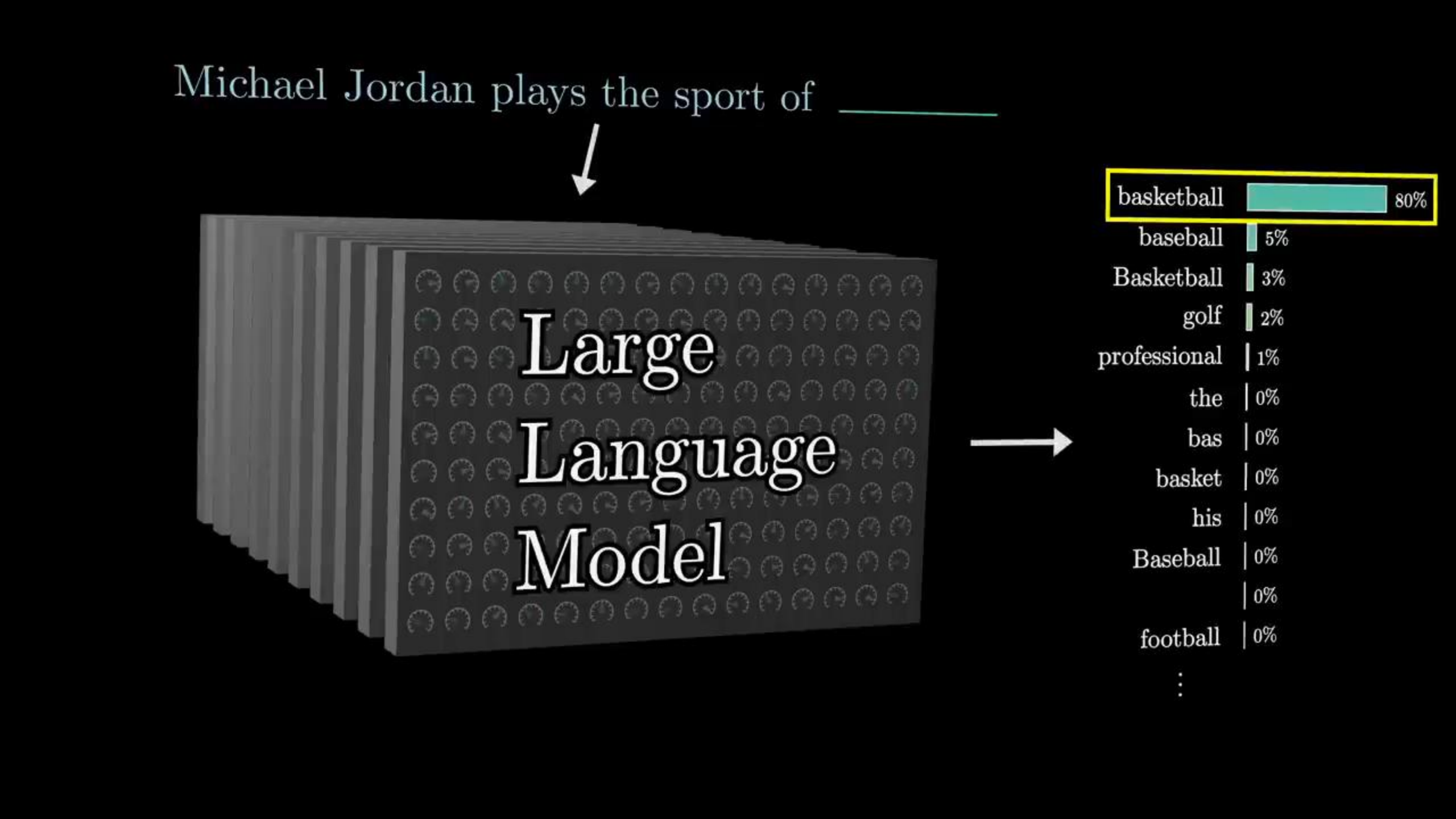
MLP (Feedforward) Layers in the Transformer

Each Transformer layer contains an MLP (Multi-Layer Perceptron), applied **independently** to each token after attention.

It consists of two fully connected (dense) layers with a non-linearity in between.

The same MLP applies to each token independently.

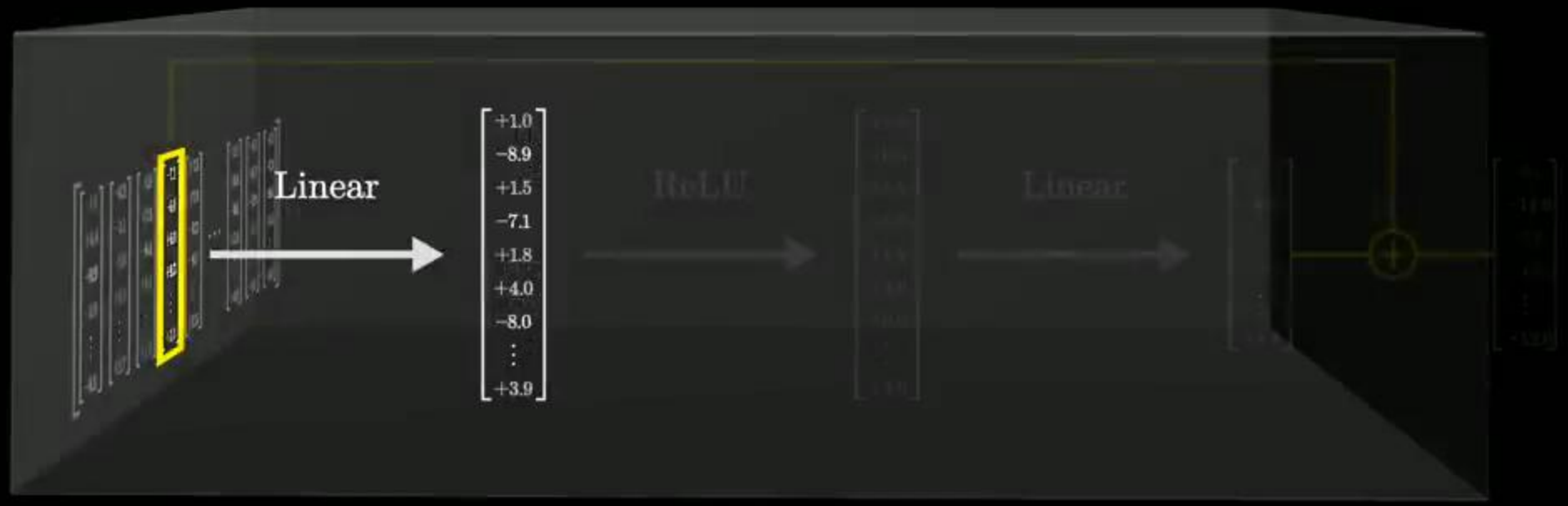
First results indicate the these MLPs allow the model to '**remember facts**'.


Michael Jordan plays the sport of _____
Large Language Model
basketball 80%
baseball 5%
Basketball 3%
golf 2%
professional 1%
the 0%
bas 0%
basket 0%
his 0%
Baseball 0%
0%
football 0%
⋮

Is it English?

$$\begin{bmatrix} -5.0 & & & & & +6.8 \\ -7.4 & -4.4 & +1.7 & +9.3 & \cdots & +1.2 \\ -9.5 & +6.0 & -5.3 & +6.1 & \cdots & -2.2 \\ +7.2 & +4.9 & +1.1 & -7.2 & \cdots & -8.7 \\ -7.5 & -9.0 & -7.8 & -5.4 & \cdots & +4.2 \\ +1.2 & -9.7 & -8.5 & +9.3 & \cdots & +1.3 \\ -5.9 & -4.9 & +4.8 & -6.0 & \cdots & +1.6 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ +9.3 & +6.9 & -5.2 & -0.1 & \cdots & +2.4 \end{bmatrix} \begin{bmatrix} -7.1 \\ -6.0 \\ +6.0 \\ +9.3 \\ \vdots \\ +3.8 \end{bmatrix} = \begin{bmatrix} -199.5 \\ +36.6 \\ -142.0 \\ -177.5 \\ -0.1 \\ +118.9 \\ +65.9 \\ \vdots \\ -218.2 \end{bmatrix}$$

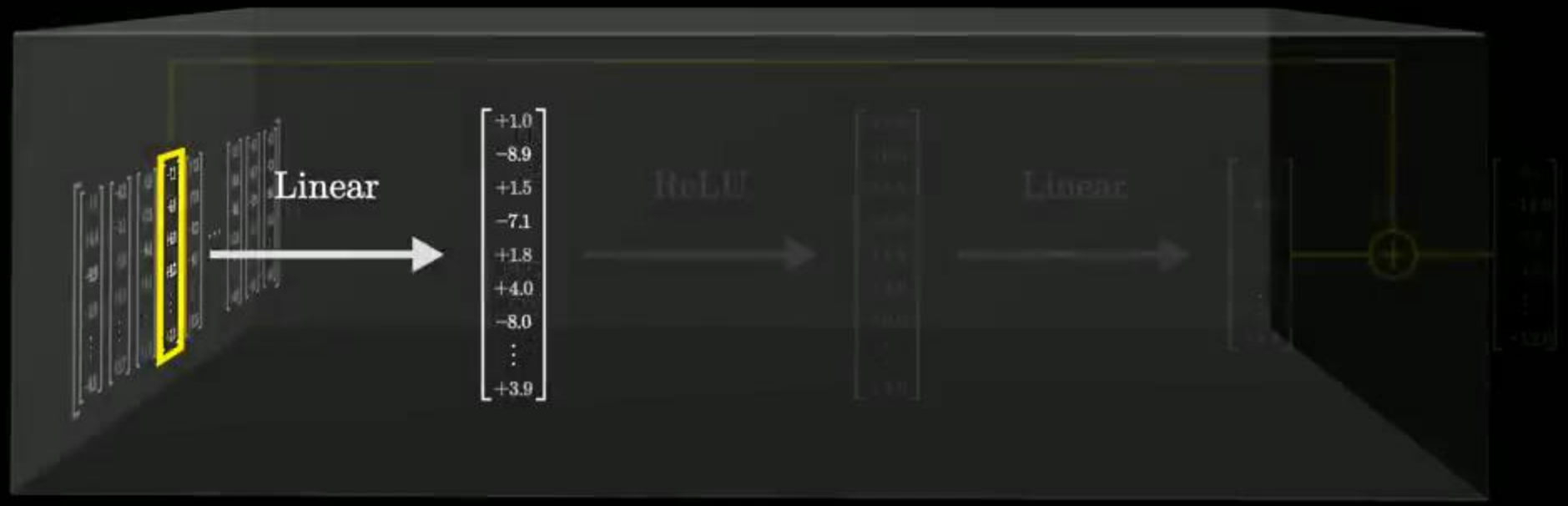
MLP



$$\begin{bmatrix} -5.0 & +7.1 & +1.5 & +1.1 & \cdots & +6.8 \\ \hline -9.5 & +6.0 & -5.3 & +6.1 & \cdots & -2.2 \\ \hline +7.2 & +4.9 & +1.1 & -7.2 & \cdots & -8.7 \\ -7.5 & -9.0 & -7.8 & -5.4 & \cdots & +4.2 \\ +1.2 & -9.7 & -8.5 & +9.3 & \cdots & +1.3 \\ -5.9 & -4.9 & +4.8 & -6.0 & \cdots & +1.6 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ +9.3 & +6.9 & -5.2 & -0.1 & \cdots & +2.4 \end{bmatrix} \begin{bmatrix} -7.1 \\ -6.0 \\ +6.0 \\ +9.3 \\ \vdots \\ +3.8 \end{bmatrix} = \begin{bmatrix} -199.5 \\ +36.6 \\ -142.0 \\ -177.5 \\ -0.1 \\ +118.9 \\ +65.9 \\ \vdots \\ -218.2 \end{bmatrix}$$

Part of source code?

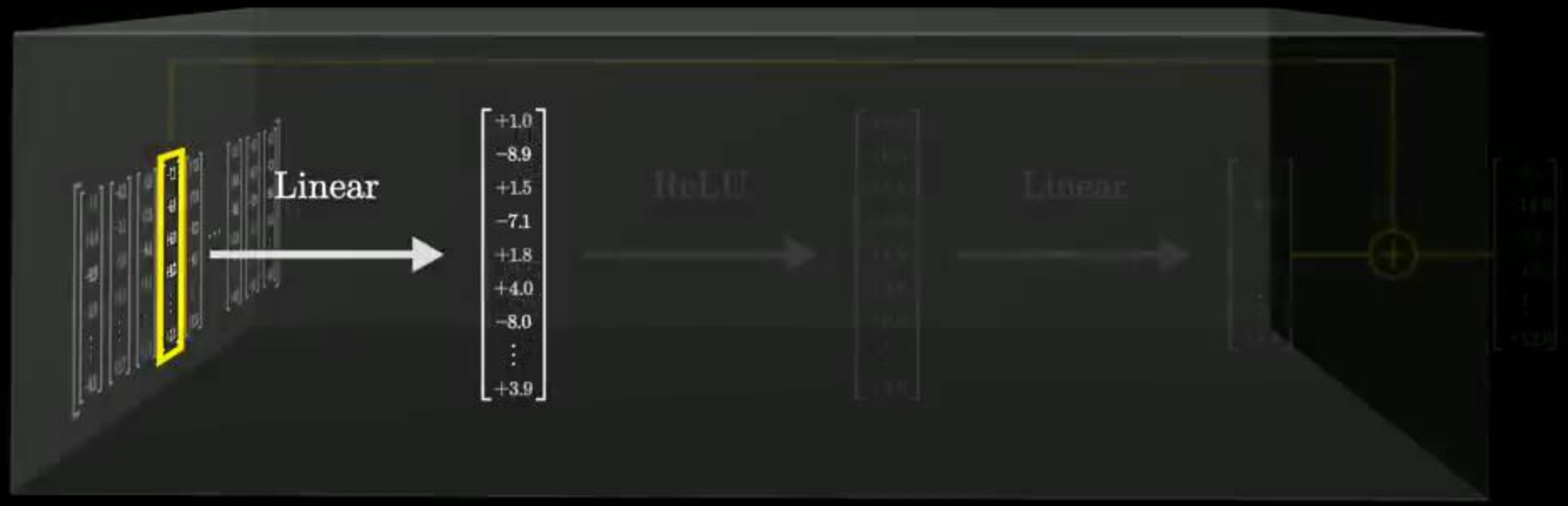
MLP

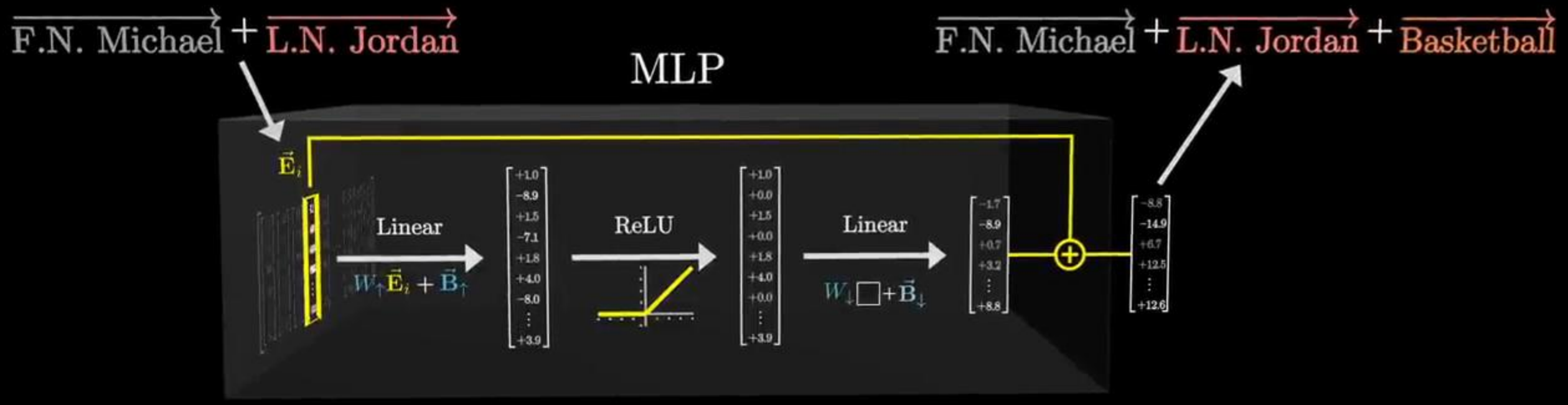


$$\begin{bmatrix} -5.0 & +7.1 & +0.8 & +1.0 & \cdots & +6.8 \\ -7.4 & -4.4 & +1.7 & +9.3 & \cdots & +1.5 \\ -6.[illegible] & [illegible] & [illegible] & [illegible] & \cdots & 2.2 \\ +7.2 & +4.9 & +1.1 & -7.2 & \cdots & -8.7 \\ -7.5 & -9.0 & -7.8 & -5.4 & \cdots & +4.2 \\ +1.2 & -9.7 & -8.5 & +9.3 & \cdots & +1.3 \\ -5.9 & -4.9 & +4.8 & -6.0 & \cdots & +1.6 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ +9.3 & +6.9 & -5.2 & -0.1 & \cdots & +2.4 \end{bmatrix} \begin{bmatrix} -7.1 \\ -6.0 \\ +6.0 \\ +9.3 \\ \vdots \\ +3.8 \end{bmatrix} = \begin{bmatrix} -199.5 \\ +36.6 \\ -142.0 \\ -177.5 \\ -0.1 \\ +118.9 \\ +65.9 \\ \vdots \\ -218.2 \end{bmatrix}$$

European country?

MLP




F.N. Michael + L.N. Jordan
MLP
$\vec{E}_i$
Linear
$W_\uparrow \vec{E}_i + \vec{B}_\uparrow$
+1.0
−8.9
+1.5
−7.1
+1.8
+4.0
−8.0
+3.9
ReLU
+1.0
+0.0
+1.5
+0.0
+1.8
+4.0
+0.0
+3.9
Linear
$W_\downarrow \square + \vec{B}_\downarrow$
−1.7
−8.9
+0.7
+3.2
+8.8
−8.8
−14.9
+6.7
+12.5
+12.6
F.N. Michael + L.N. Jordan + Basketball

GPT-3

Total weights:

175,181,291,520

Embedding	$\overset{12,288}{\text{d_embed}} * \overset{50,257}{\text{n_vocab}}$	= 617,558,016
Key	$\overset{128}{\text{d_query}} * \overset{12,288}{\text{d_embed}} * \overset{96}{\text{n_heads}} * \overset{96}{\text{n_layers}}$	= 14,495,514,624
Query	$\overset{128}{\text{d_query}} * \overset{12,288}{\text{d_embed}} * \overset{96}{\text{n_heads}} * \overset{96}{\text{n_layers}}$	= 14,495,514,624
Value	$\overset{128}{\text{d_value}} * \overset{12,288}{\text{d_embed}} * \overset{96}{\text{n_heads}} * \overset{96}{\text{n_layers}}$	= 14,495,514,624
Output	$\overset{12,288}{\text{d_embed}} * \overset{128}{\text{d_value}} * \overset{96}{\text{n_heads}} * \overset{96}{\text{n_layers}}$	= 14,495,514,624
Up-projection	$\overset{49,152}{\text{n_neurons}} * \overset{12,288}{\text{d_embed}} * \overset{96}{\text{n_layers}}$	= 57,982,058,496
Down-projection	$\overset{12,288}{\text{d_embed}} * \overset{49,152}{\text{n_neurons}} * \overset{96}{\text{n_layers}}$	= 57,982,058,496
Unembedding	$\overset{50,257}{\text{n_vocab}} * \overset{12,288}{\text{d_embed}}$	= 617,558,016

Data: Common Crawl

Common Crawl is an open repository of web crawl data that contains petabytes of information extracted from billions of web pages dating back to 2008.

The dataset is freely available for anyone to download and use, making it a valuable resource for researchers, businesses, and developers working on natural language processing, machine learning, and web analysis projects.

As of late 2024, the Common Crawl corpus contained over 250 billion web pages, with the entire archive consuming hundreds of petabytes of storage space.

Each monthly crawl typically contains 200-300 terabytes of data, representing billions of individual web pages from tens of millions of domains.

The dataset includes content in hundreds of languages, though English content historically makes up approximately 40-50% of the corpus.

Chat bots

https://poloclub.github.io/transformer-explainer/

GTP-2

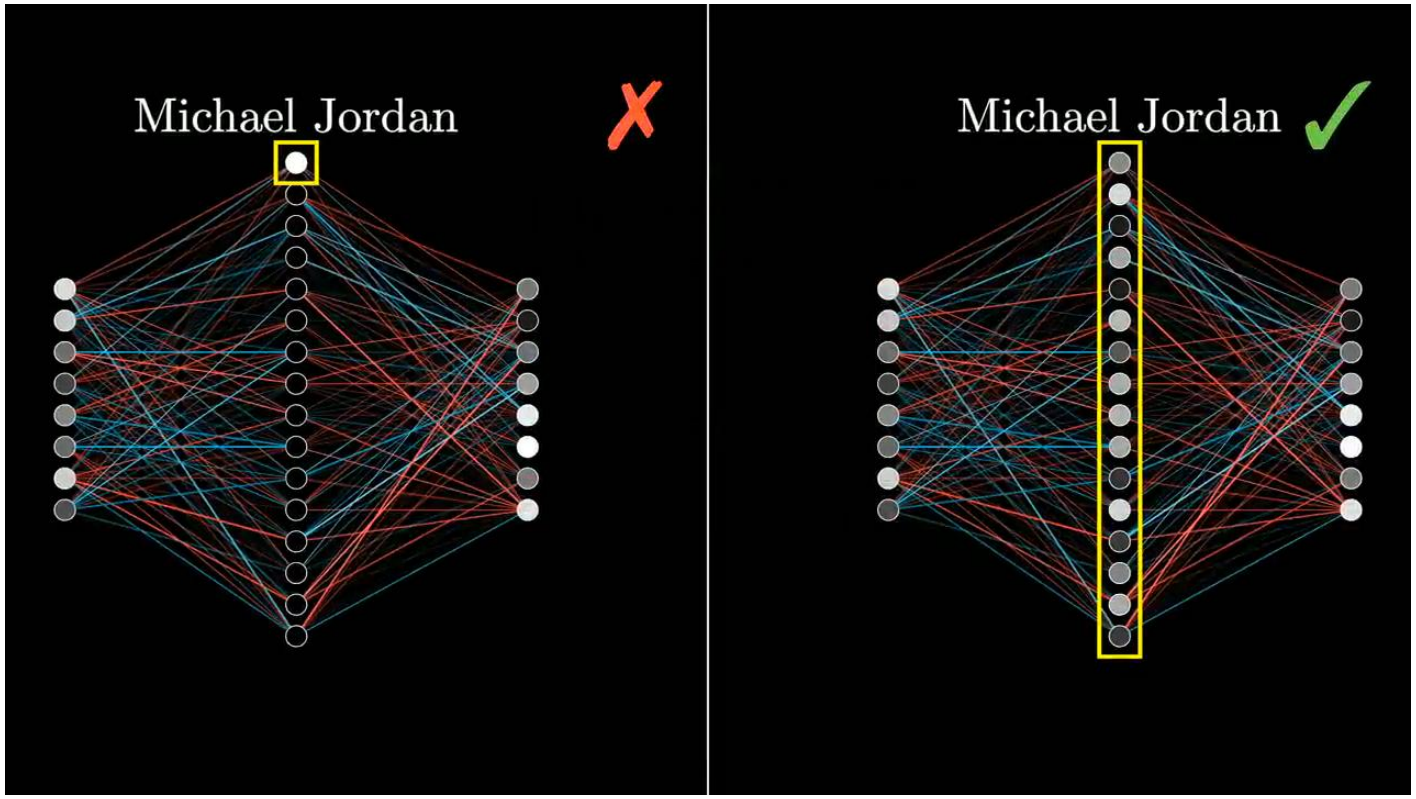
GitHub

GPT_text_generation.ipynb

Superposition

Neural networks store **multiple features in overlapping directions** within their weight space, rather than dedicating separate neurons to each feature.

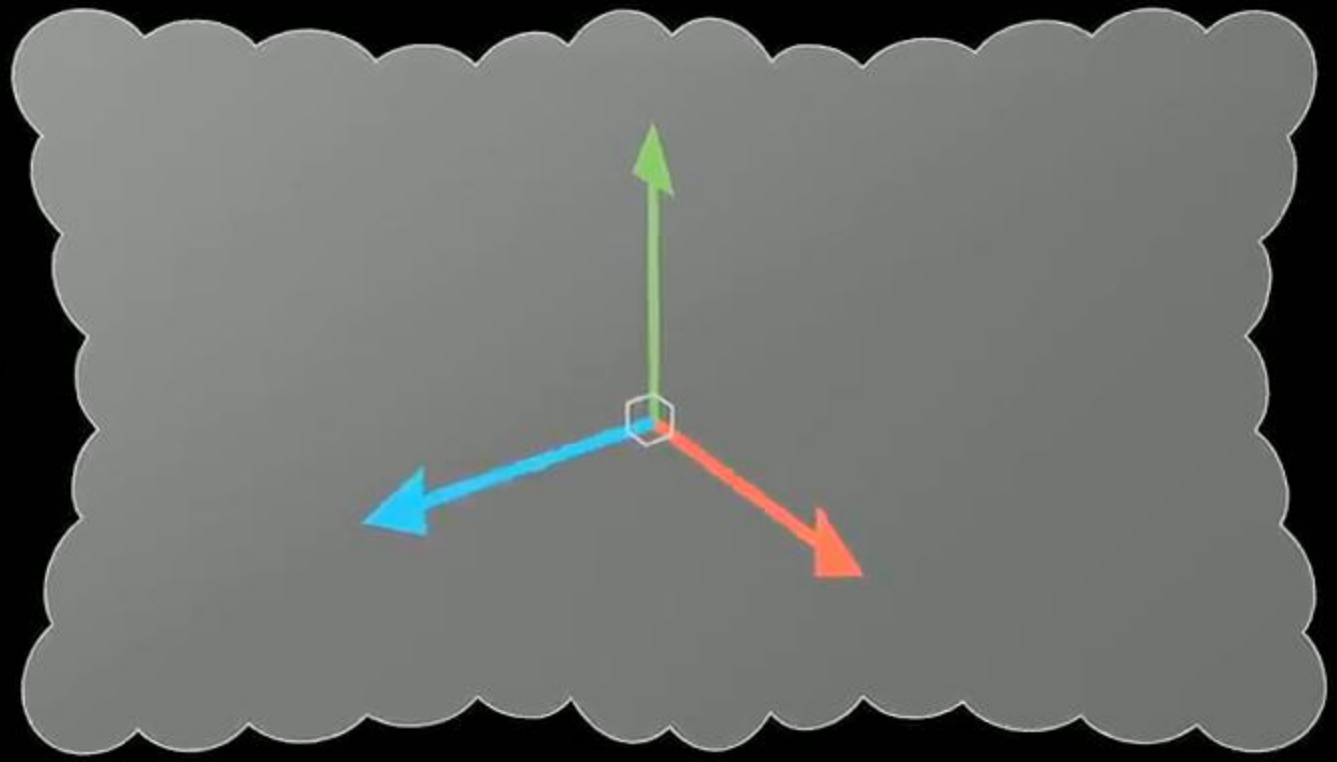
This allows a network to encode more features than the number of neurons, similar to how quantum superposition allows multiple states to exist simultaneously.

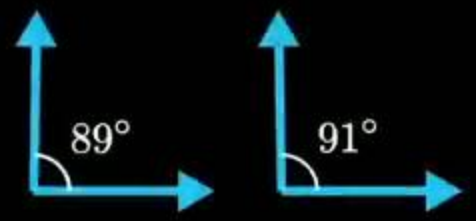


Choose multiple vectors,
each pair 90° apart

Maximum # of vectors: N

N-dimensional
Space

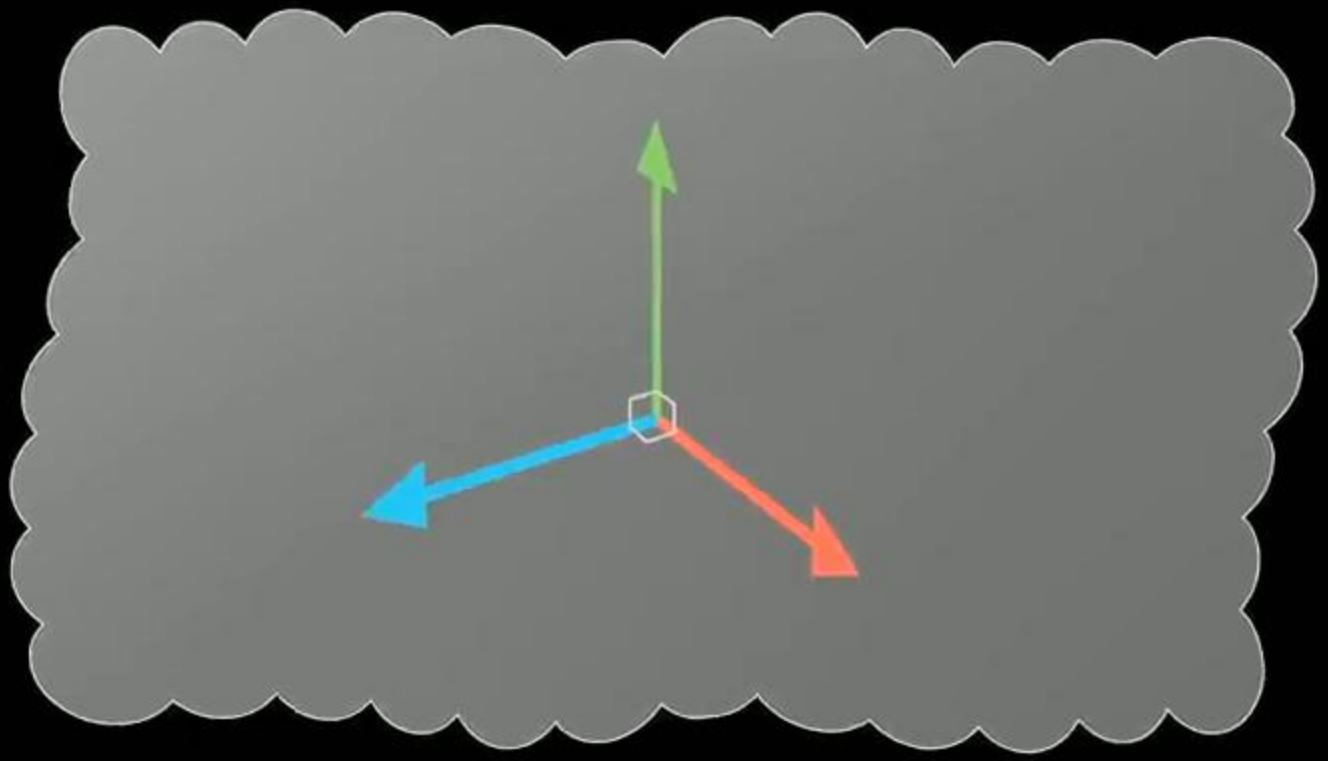




Choose multiple vectors,
each pair between 89° and 91° apart

~~90° apart~~

N-dimensional
Space



almost_orthogonal.py — videos, manim

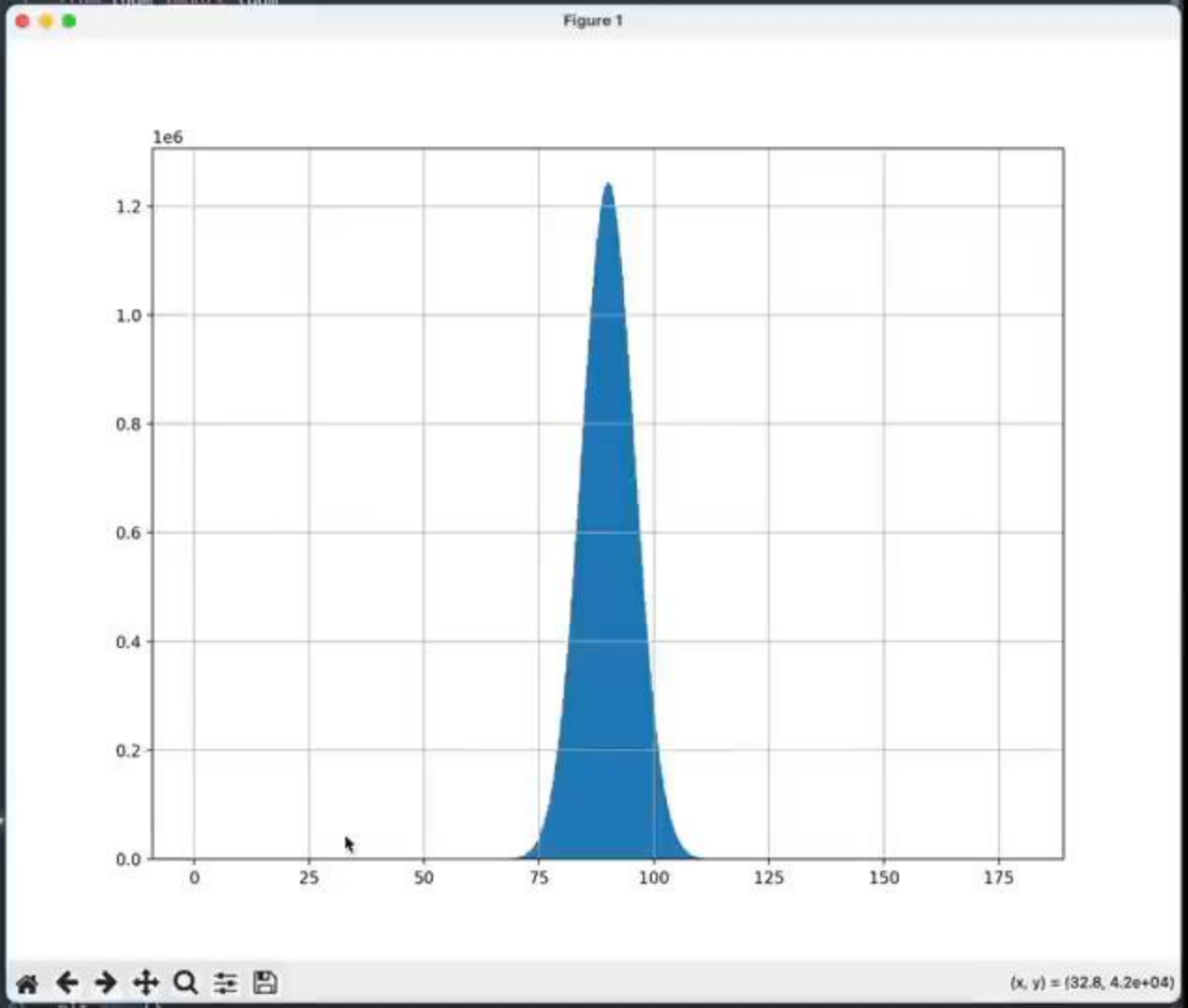
```python
import torch
import matplotlib.pyplot as plt
from tqdm import tqdm

# List of vectors in some dimension, with many
# more vectors than there are dimensions
num_vectors = 10000
vector_len = 100
big_matrix = torch.randn(num_vectors, vector_len)
big_matrix /= big_matrix.norm(p=2, dim=1, keepdim=True)  # Normalize
big_matrix.requires_grad_(True)

# Set up an Optimization loop to create nearly-perpendicular vectors
optimizer = torch.optim.Adam([big_matrix], lr=0.01)
num_steps = 250

losses = []

dot_diff_cutoff = 0.01
big_id = torch.eye(num_vectors, num_vectors)

for step_num in tqdm(range(num_steps)):
    optimizer.zero_grad()

    dot_products = big_matrix @ big_matrix.T
    # Punish deviation from orthogonal
    diff = dot_products - big_id
    loss = (diff.abs() - dot_diff_cutoff).relu().sum()

    # Extra incentive to keep rows normalized
    loss += num_vectors * diff.diag().pow(2).sum()

    loss.backward()
    optimizer.step()
    losses.append(loss.item())

# Loss curve
plt.plot(losses)
plt.grid(1)
plt.show()

# Angle distribution
dot_products = big_matrix @ big_matrix.T
norms = torch.sqrt(torch.diag(dot_products))
normed_dot_products = dot_products / torch.outer(norms, norms)
angles_degrees = torch.rad2deg(torch.acos(normed_dot_products.detach()))
# Use this to ignore self-orthogonality.
self_orthogonality_mask = ~(torch.eye(num_vectors, num_vectors).bool())
plt.hist(angles_degrees[self_orthogonality_mask].numpy().ravel(), bins=1000, range=(0, 180))
plt.grid(1)
plt.show()
```

100-dimensional

10,000 vectors

$$
\begin{bmatrix}
-0.6 & +0.8 & +0.1 & -0.1 & \cdots & +0.8 & -0.1 & +0.4 & -0.2 \\
+0.8 & +0.4 & +0.4 & -0.3 & \cdots & +0.5 & +0.3 & -0.5 & -0.7 \\
+0.6 & +0.9 & -0.1 & +0.2 & \cdots & +0.7 & -0.1 & +0.9 & +0.2 \\
+0.6 & +0.8 & +0.6 & -0.7 & \cdots & +0.3 & -0.2 & -0.9 & -0.2 \\
-0.5 & +0.7 & -0.9 & +0.9 & \cdots & -0.3 & -0.3 & -1.0 & -0.6 \\
-0.2 & +0.9 & -0.8 & +0.9 & \cdots & +0.7 & -0.1 & -0.3 & -0.5 \\
+0.2 & -0.9 & -1.0 & -0.1 & \cdots & -0.9 & -0.5 & -0.6 & -0.5 \\
-0.7 & -1.0 & -0.8 & +0.2 & \cdots & +0.9 & +1.0 & -0.2 & -0.7 \\
+0.3 & +0.0 & +1.0 & -0.9 & \cdots & +0.6 & -0.4 & -0.5 & +0.3 \\
-0.5 & +0.3 & +0.0 & -0.2 & \cdots & +0.1 & -0.4 & +0.4 & -0.2 \\
-0.3 & +0.7 & +0.8 & -0.9 & \cdots & -0.5 & -0.3 & +0.6 & +1.0 \\
+0.9 & +0.8 & -0.4 & +1.0 & \cdots & -0.5 & -0.8 & +0.9 & -0.5 \\
+0.4 & -0.9 & +0.5 & +0.8 & \cdots & -0.5 & -0.2 & -0.3 & +0.5 \\
-0.5 & -0.7 & -0.1 & -0.4 & \cdots & +0.7 & -0.5 & +0.0 & +0.9 \\
+0.3 & +0.7 & +0.9 & +0.5 & \cdots & +0.4 & +0.9 & +1.0 & -0.1 \\
-0.9 & -0.4 & -0.7 & -0.2 & \cdots & -0.7 & +0.2 & -0.2 & +0.8 \\
+0.9 & +0.1 & -0.5 & +0.2 & \cdots & +0.8 & -0.2 & +0.1 & -0.5 \\
-0.1 & -0.2 & -0.5 & +0.0 & \cdots & -0.4 & -0.3 & +0.0 & +0.5 \\
 & & & & \vdots & & & & \\
-0.9 & -0.6 & -1.0 & +0.6 & \cdots & -0.3 & +0.9 & +0.4 & -0.9
\end{bmatrix}
$$

```python
import torch
import matplotlib.pyplot as plt
from tqdm import tqdm
```




```python
plt.show()
```

100-dimensional

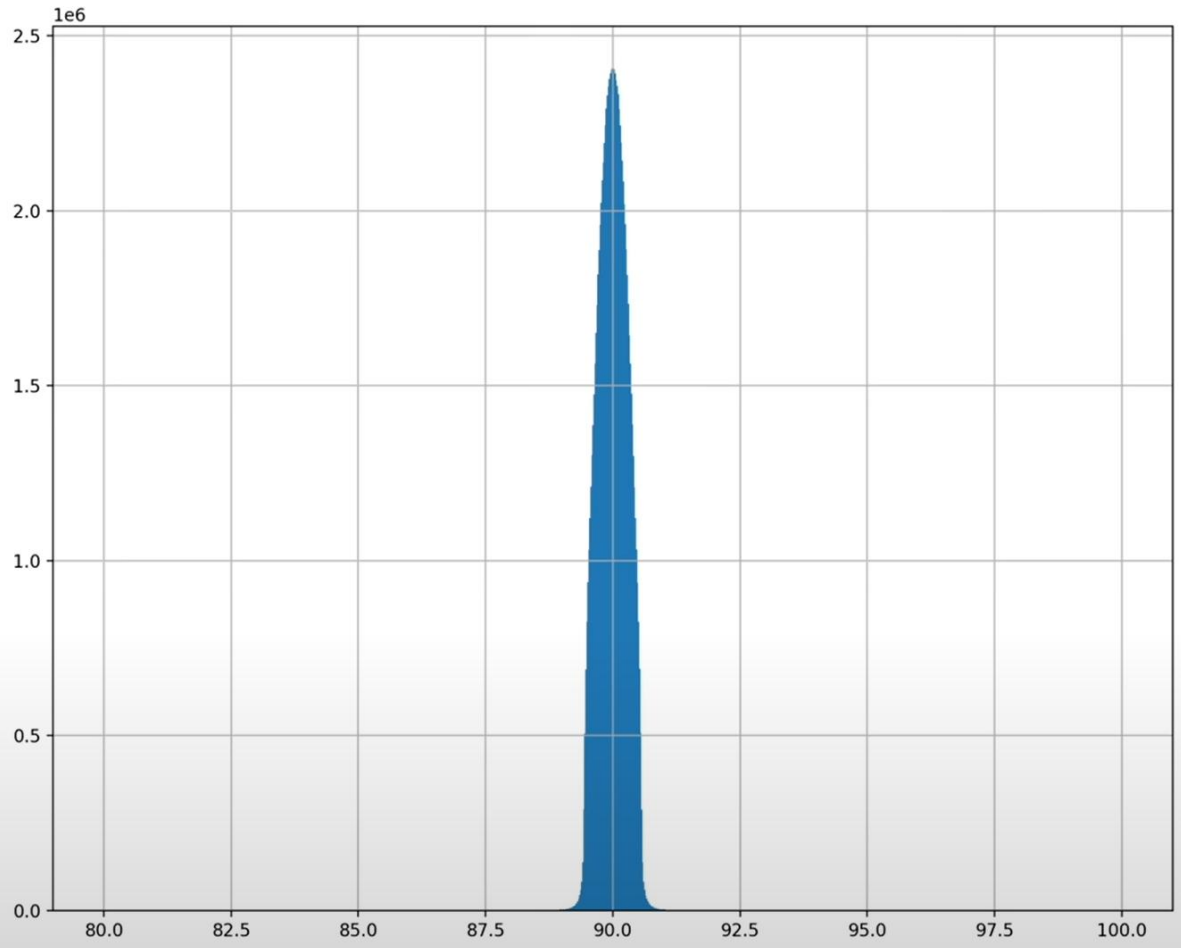
10,000 vectors

$$
\begin{bmatrix}
-0.6 & +0.8 & +0.1 & -0.1 & \cdots & +0.8 & -0.1 & +0.4 & -0.2 \\
+0.8 & +0.4 & +0.4 & -0.3 & \cdots & +0.5 & +0.3 & -0.5 & -0.7 \\
+0.6 & +0.9 & -0.1 & +0.2 & \cdots & +0.7 & -0.1 & +0.9 & +0.2 \\
+0.6 & +0.8 & +0.6 & -0.7 & \cdots & +0.3 & -0.2 & -0.9 & -0.2 \\
-0.5 & +0.7 & -0.9 & +0.9 & \cdots & -0.3 & -0.3 & -1.0 & -0.6 \\
-0.2 & +0.9 & -0.8 & +0.9 & \cdots & +0.7 & -0.1 & -0.3 & -0.5 \\
+0.2 & -0.9 & -1.0 & -0.1 & \cdots & -0.9 & -0.5 & -0.6 & -0.5 \\
-0.7 & -1.0 & -0.8 & +0.2 & \cdots & +0.9 & +1.0 & -0.2 & -0.7 \\
+0.3 & +0.0 & +1.0 & -0.9 & \cdots & +0.6 & -0.4 & -0.5 & +0.3 \\
-0.5 & +0.3 & +0.0 & -0.2 & \cdots & +0.1 & -0.4 & +0.4 & -0.2 \\
-0.3 & +0.7 & +0.8 & -0.9 & \cdots & -0.5 & -0.3 & +0.6 & +1.0 \\
+0.9 & +0.8 & -0.4 & +1.0 & \cdots & -0.5 & -0.8 & +0.9 & -0.5 \\
+0.4 & -0.9 & +0.5 & +0.8 & \cdots & -0.5 & -0.2 & -0.3 & +0.5 \\
-0.5 & -0.7 & -0.1 & -0.4 & \cdots & +0.7 & -0.5 & +0.0 & +0.9 \\
+0.3 & +0.7 & +0.9 & +0.5 & \cdots & +0.4 & +0.9 & +1.0 & -0.1 \\
-0.9 & -0.4 & -0.7 & -0.2 & \cdots & -0.7 & +0.2 & -0.2 & +0.8 \\
+0.9 & +0.1 & -0.5 & +0.2 & \cdots & +0.8 & -0.2 & +0.1 & -0.5 \\
-0.1 & -0.2 & -0.5 & +0.0 & \cdots & -0.4 & -0.3 & +0.0 & +0.5 \\
 & & & & \vdots & & & & \\
-0.9 & -0.6 & -1.0 & +0.6 & \cdots & -0.3 & +0.9 & +0.4 & -0.9
\end{bmatrix}
$$

almost_orthogonal.py — videos, manim

almost_orthogonal.py | mlp.py

Figure 1



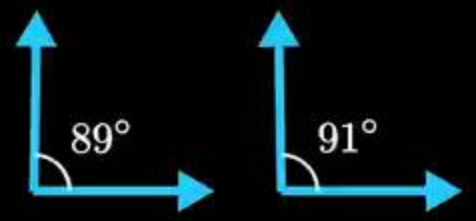
BlankScene

Login Shell

```
nt_paste() # Angle distribution (11 lines)
```

52 Lines, 153 Words, 1.299 Chars, LSP-pylsp, Line 42, Column 1 | master | 0 misspelled words | Spaces: 4 | Python

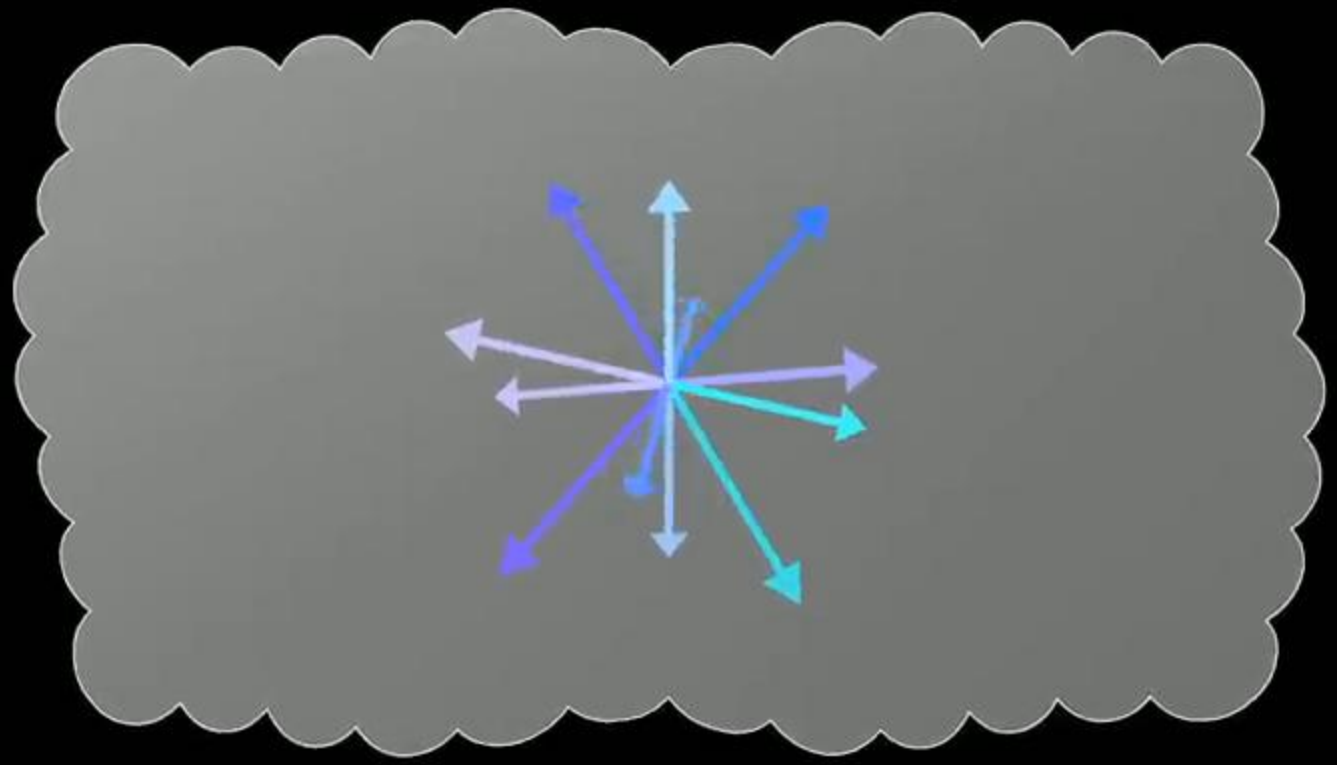
2.327 Lines, 10.967 Words, 94.861 Chars, Line 2342, Column 1 | Spaces: 4 | Plain Text



Choose multiple vectors,
each pair between 89° and 91° apart

Johnson–Lindenstrauss Lemma $\Longrightarrow$ Maximum # of vectors: $\approx \exp(\epsilon \cdot N)$

N-dimensional Space



Few-shot learning

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
cheese =>                         ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
sea otter => loutre de mer        ← example
cheese =>                         ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
sea otter => loutre de mer        ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                         ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
sea otter => loutre de mer        ← example #1
↓
gradient update
↓
peppermint => menthe poivrée      ← example #2
↓
gradient update
↓
...
↓
plush giraffe => girafe peluche   ← example #N

gradient update

cheese =>                         ← prompt
```

DNA/protein LLMs

Transformer can process **DNA, RNA, and protein sequences**, learning meaningful patterns, mutations, and structural relationships.

Transformers for biomedical data are often pretrained on **massive biological sequence databases** to capture underlying biological structures and functional information.

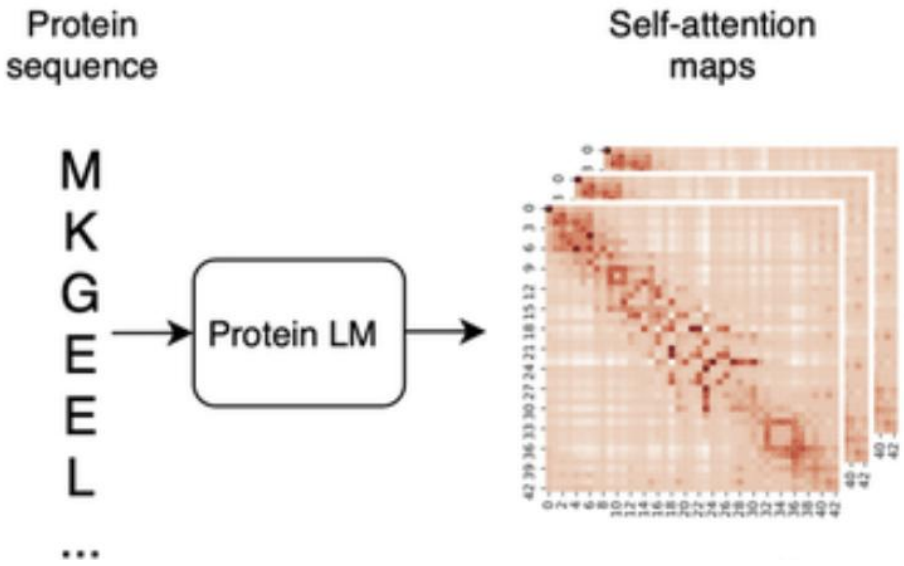
Pretraining can achieved by training the model to predict masked nucleotides or amino acids.

E.g. ESM2 (Evolutionary Scale Modeling v2)

ESM2 is a transformer-based **protein language model** that learns from massive protein sequence datasets, capturing evolutionary and functional properties of proteins through self-supervised learning.

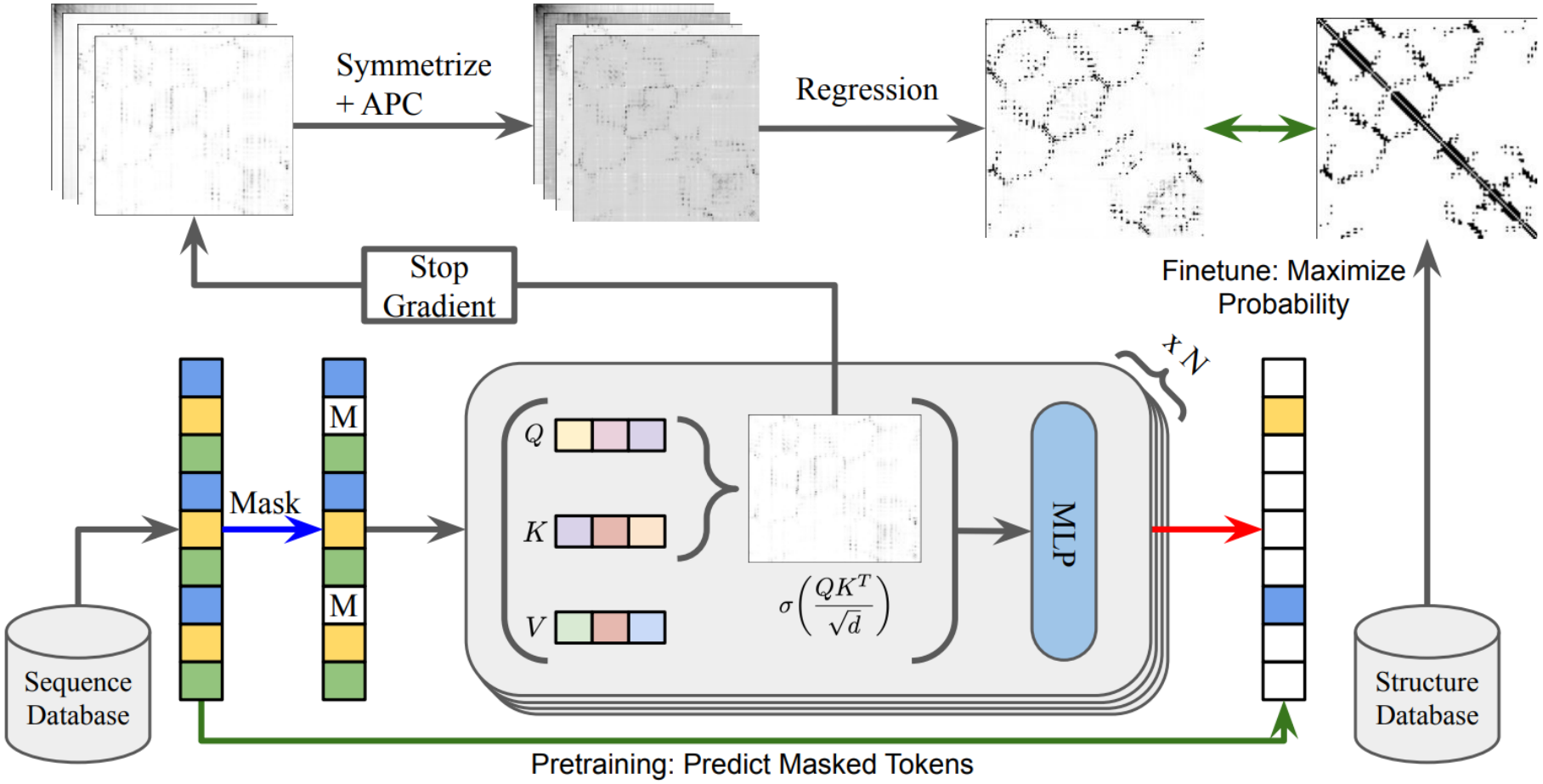
Like BERT in NLP, ESM2 is trained using **masked language modeling (MLM)**, where random amino acids in protein sequences are masked, and the model learns to predict them based on context. This forces ESM2 to understand protein grammar, structure, and evolutionary constraints.

Trained on **millions of protein sequences** (UniRef50), covering a broad spectrum of biological diversity across different organisms.



Rao et al., Transformer protein language models are unsupervised structure learners, bioRxiv, 2020

E.g. ESM2 (Evolutionary Scale Modeling v2)



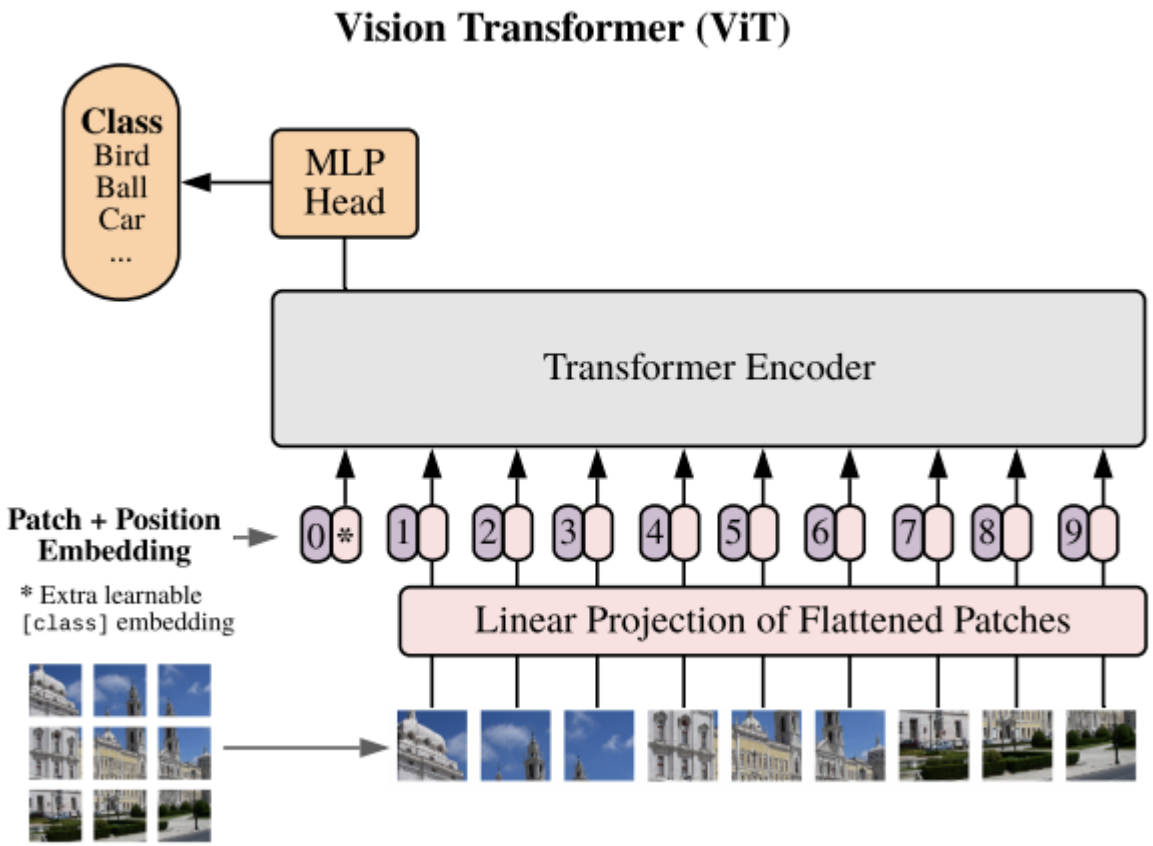
Rao et al., Transformer protein language models are unsupervised structure learners, bioRxiv, 2020

Vision Transformer (ViT)

The Vision Transformer (ViT) applies the **self-attention mechanism** from transformers to image data by dividing an image into small patches and processing them like a sequence of tokens.

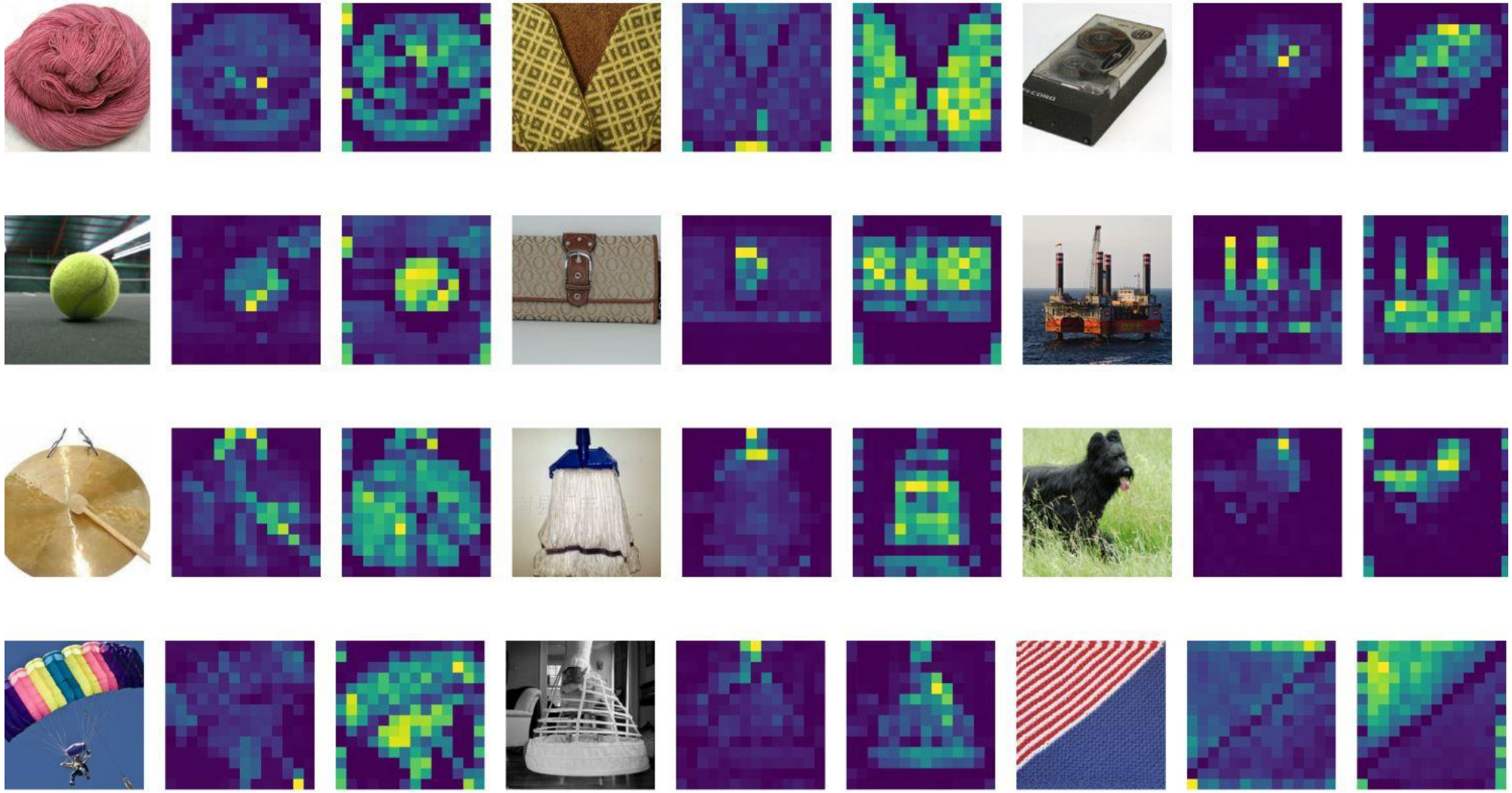
ViT outperforms CNNs on large-scale datasets (e.g., ImageNet) by capturing **long-range dependencies and global context** more effectively.

ViT can be **fine-tuned for various vision tasks** (e.g., classification, object detection, segmentation) and benefits from **pretraining on large datasets.**

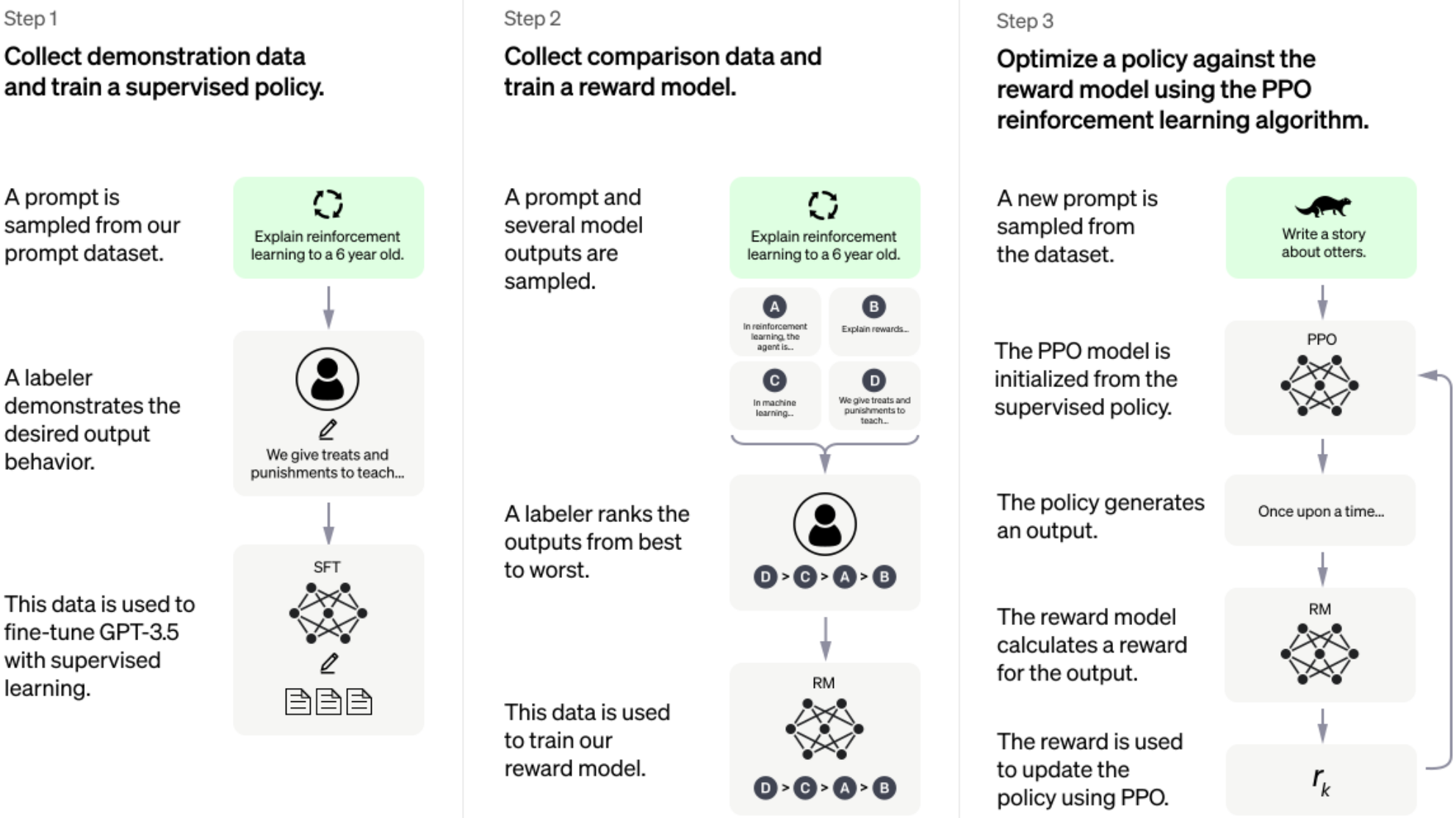


Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, ICLR, 2021

Vision Transformer (ViT) self attention



Chat bots



openAI

Retrieval-Augmented Generation (RAG)

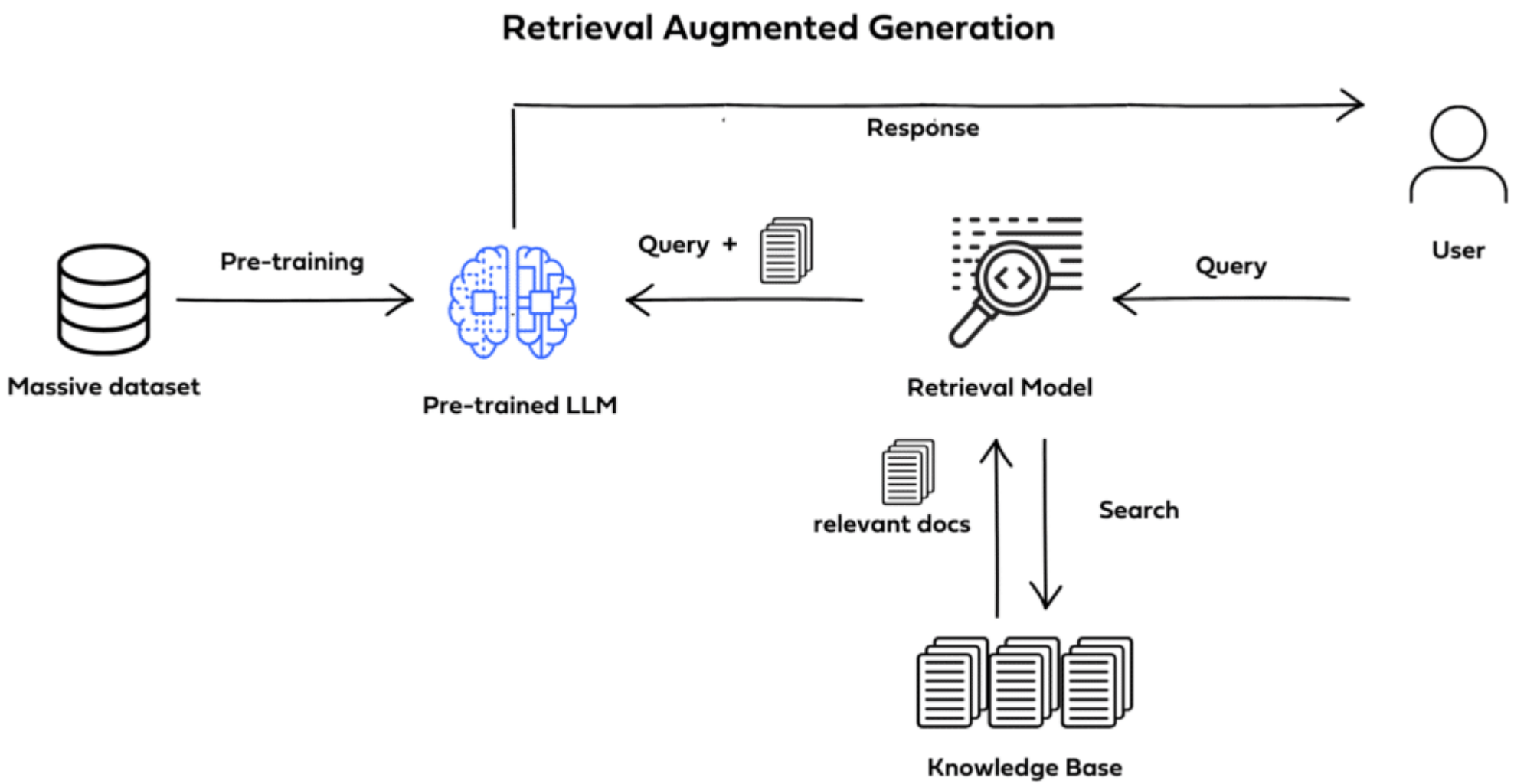
RAG enhances Large Language Models (LLMs) by integrating external knowledge retrieval, improving accuracy and reducing hallucinations.

Retrieval phase: The model searches a knowledge base (e.g., FAISS, vector databases) to fetch relevant documents based on the user query.

Generation phase: The LLM processes retrieved documents alongside the query to generate informed, context-aware responses.

RAG is used in chatbots, research assistants, legal and medical AI, and any domain requiring precise, up-to-date information beyond pre-trained knowledge.

Retrieval-Augmented Generation (RAG)



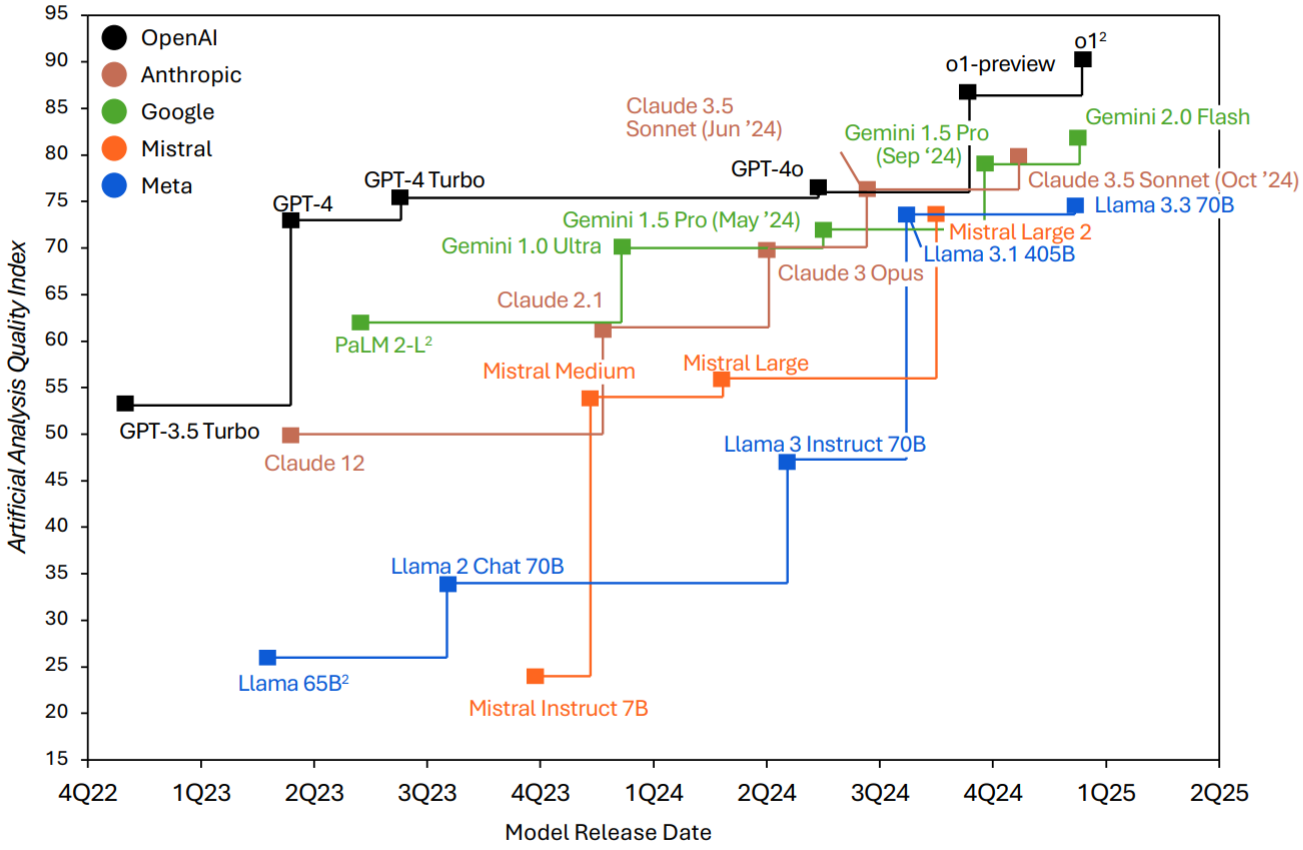
Medium.com

FRONTIER MODELS

2024 saw multiple labs catch up to OpenAI's GPT-4, and the emergence of the first models to push beyond GPT-4's level of intelligence

Frontier Language Model Intelligence, Over Time[1]

Artificial Analysis Quality Index: MMLU, GPQA Diamond, MATH-500, HumanEval. Top 5 Labs by Market Share.



Key Trends

Competing labs catch up to OpenAI's GPT-4: OpenAI started the language model race in Nov 2022 with the launch of GPT-3.5 in ChatGPT; competing labs spent the following 18 months catching up

Open models close in on the frontier labs: Open weights models, led by those from Meta, Mistral and Alibaba, have approached and surpassed GPT-4's level of intelligence

Sparks of intelligence beyond GPT-4: The final months of 2024 have seen the emergence of the first major intelligence leaps beyond GPT-4, led by OpenAI's o1. Topics including inference-time compute scaling, data quality and new reinforcement learning techniques have joined pre-training compute scaling as dominant levers for improving models

1. Artificial Analysis Quality Index: average across a range of language model intelligence and reasoning evaluation datasets. Currently includes MMLU, GPQA Diamond, MATH-500 & HumanEval. Estimated based on lab-claimed evaluation results for some non-current models. Release date is based on first public launch of the model . 2. Artificial Analysis Quality Index has been estimated based on lab-claimed evaluation results for certain models (non-current or API not yet available).

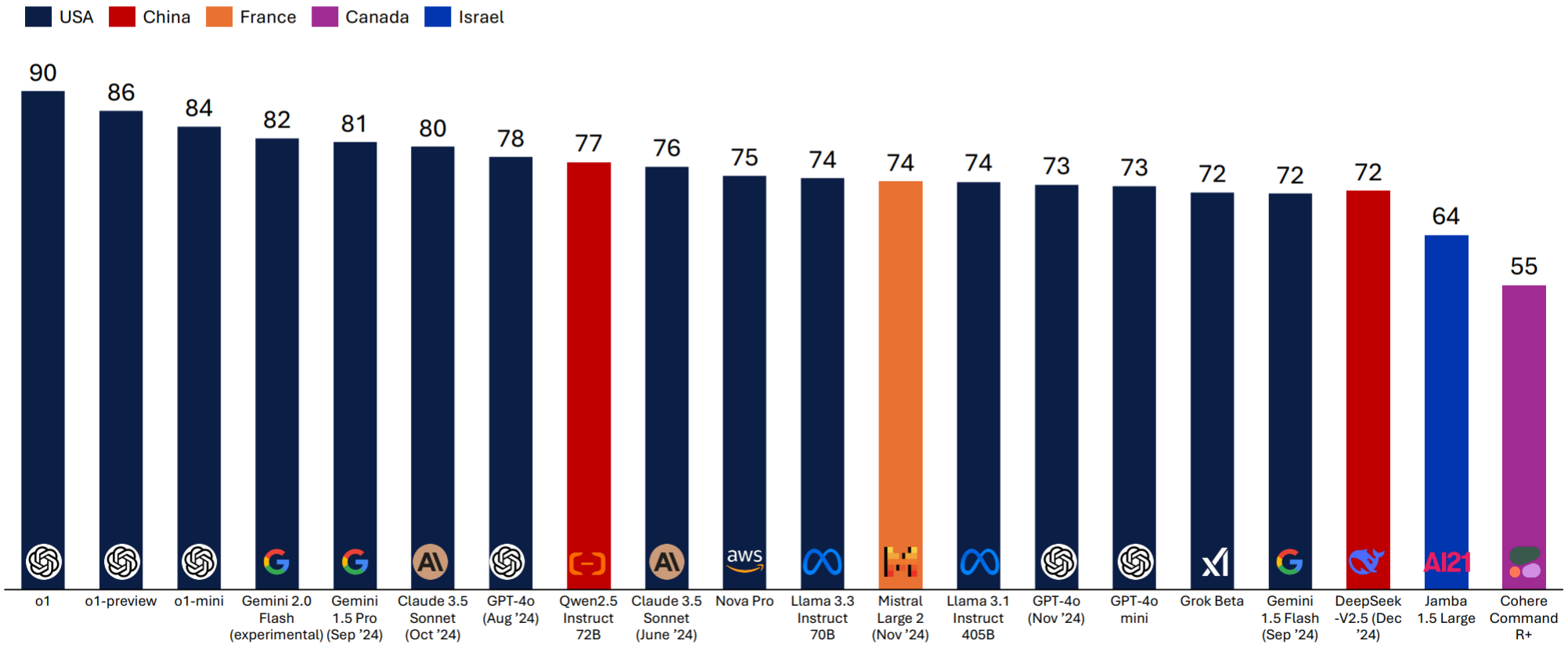
Artificial Analysis

LANGUAGE MODEL COUNTRY OF ORIGIN

The US dominates the intelligence frontier; China appears to be in a clear second place and only a handful of other countries have demonstrated frontier-class training

The Language Model Frontier: Country of Origin

Artificial Analysis Quality Index, Leading Models (Late 2024)



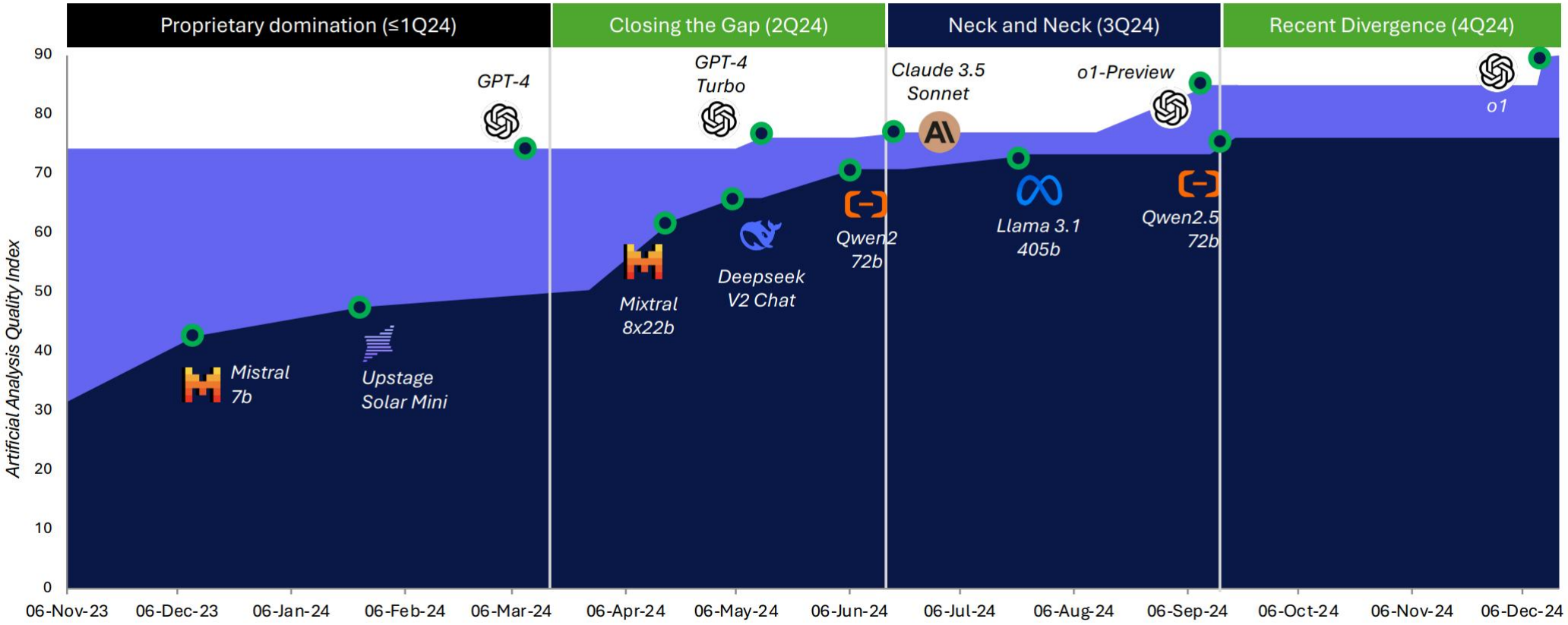
Accurate as of December 16, 2024

Artificial Analysis

Driven by models from Meta, Mistral and Alibaba, the performance gap between open source and proprietary models has decreased significantly

Model Quality: Leading Proprietary and Open Weights Models

Based on proprietary and open-source models that resulted in an increase in Artificial Analysis Intelligence Index score

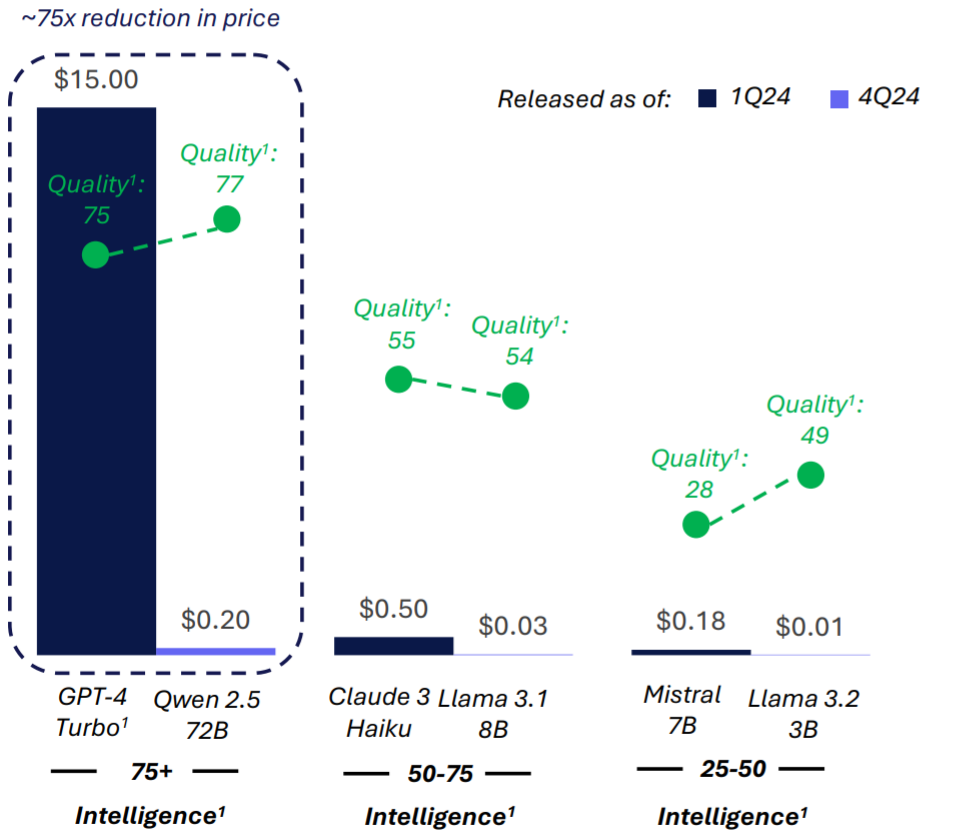


Artificial Analysis

Language model inference pricing fell dramatically in 2024 for all levels of intelligence; GPT-4o mini approaches GPT-4 intelligence at a 100x cheaper price

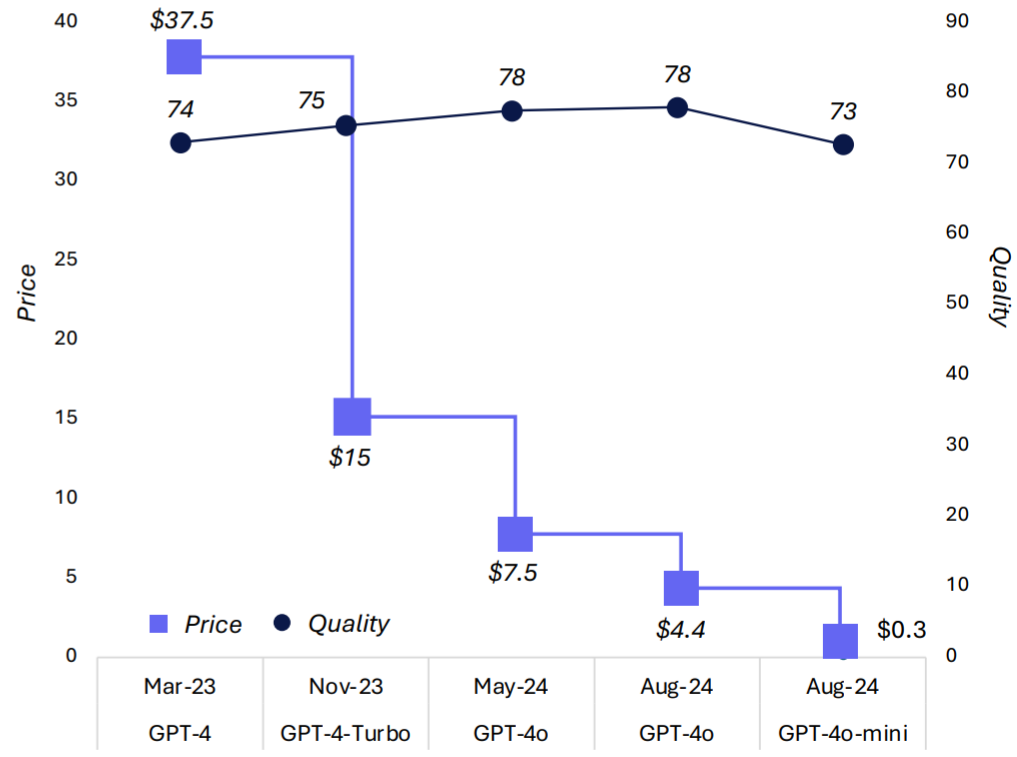
Inference Pricing by Intelligence Tier (1Q24 vs 4Q24)

USD/m tokens, blended input and output tokens (3:1 ratio)



Inference Cost for OpenAI's GPT-4 Model Family

USD/m tokens, blended input and output tokens (3:1 ratio)



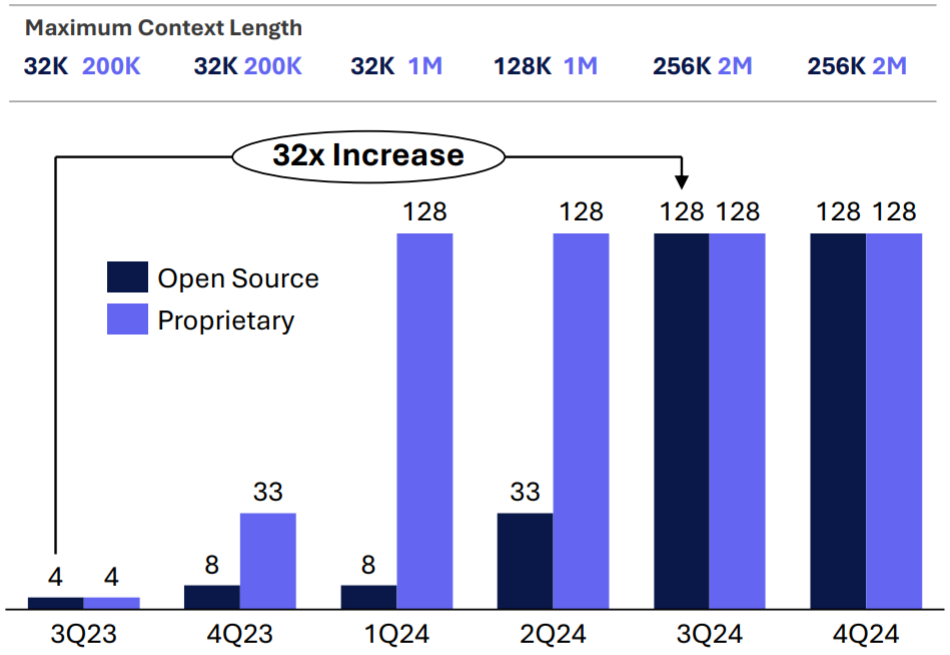
1 Artificial Analysis Quality Index: average across a range of language model intelligence and reasoning evaluation datasets. Currently includes MMLU, GPQA Diamond, MATH-500 & HumanEval.

Artificial Analysis

Context windows have increased significantly to 128k as the new norm; long context reasoning enables models to work with more data at once

Context Length (Tokens), Median by Quarter

Median context length (thousand tokens)[1]



Key takeaway

The median context length of frontier models has increased 32x since 3Q23. Expansion was originally led by proprietary models but as of 3Q24 open source has caught up. Recent advancements have also increased the maximum context length of some models (Gemini, Nova) to 2M tokens

What's Driving the Improvement?

New Techniques	• Techniques for increasing context length have ranged from hardware aware distributed attention implementations to attention approximation and length extrapolation methods such as RoPE
Developer Demand	• 70% of developers in the Artificial Analysis Developer Survey indicated the context window was important to them when choosing a model • Labs are responding to a clear industry demand signal

Implications

Reduced Complexity	• Managing smaller context windows in production leads to complexity and tradeoffs • Long contexts reduce your need to establish retrieval, summarization and truncation strategies
New Approaches & Applications	• Tools for agents quickly deplete the context window • Many applications don't leverage only text. Larger context windows support multi-modal inputs including images, video, and audio
Inference Strategies	• There is a growing trend of more compute being utilized per task, whether in the form of generating additional reasoning tokens or agentic workflows. Longer contexts enable more flexibility

1. Based on models tracked by Artificial Analysis

Artificial Analysis