

Topic: Attention Is All You Need

About Me:

I am **Alex (Alexander)**, did **M.Sc. (Tech) in Electronics** from **NIT Warangal (2007)**

17+ Years of Industry Experience blending **Embedded Systems (12 Yrs)** and **Artificial Intelligence / Machine Learning (5 Yrs).**



The Problem: RNNs & LSTMs forget context

Before transformers (2017), models like **RNNs** and **LSTMs** processed words **one by one (Sequential)**.

- They remembered **recent words**, but **forgot older ones** (long sentences were hard).
- They had a fixed “**memory flow**,” so they **couldn’t easily look back** at all words at once.



Researchers wanted a system that could:

- Look at **any word in a sentence**, no matter how far apart.
- Weigh **how important** each other word is — dynamically.

Tokenization — Breaking Text into Pieces

Idea: Splitting text into smaller units called tokens (words)

Purpose: Converts raw text into a structured format that a model can process.

Example:

Text: “AI changes the world.”

Tokens: [“AI”, “changes”, “the”, “world”, “.”]

Types:

Word-level: Each word is a token → simple but large vocabulary.

Vectorization — Assigning Numbers to Tokens

Idea: Machines understand numbers, not words.

Goal: Convert each token into a numerical representation (vector).

Common Techniques:

- **One-Hot Encoding:** Binary vector → 1 for the token, 0 for others.
- **Count/TF-IDF Vectors:** Represent word frequency or importance in a document.

Example:

If vocabulary = [AI, changes, world],

“AI world” → [1, 0, 1]

Limitation: No meaning captured — “king” and “queen” are just numbers with no relation.

Embedding — Learning Meaningful Representations

Idea: Represent tokens as **dense numeric vectors** where **similar words are closer** in the vector space.

Goal: Capture *semantic meaning* and *context*.

Example:

“King – Man + Woman ≈ Queen”

Techniques:

- **Word2Vec / GloVe:** Pre-trained embeddings capturing word relationships.

The Insight: “Attention” as a lookup system

Asked:

“When **humans read, we don’t process words one by one**. We look at the **whole sentence** and focus on the **parts that matter**.”

I can read it , can You?

fi yuo cna raed tihs, yuo hvae a sgtrane mnid too. Cna
yuo raed tihs? Olny 55 plepoe out of 100 can.

i cdnuolt blveiee taht I cluod aulacly uesdnatnrd waht I
was rdanieg. The phaonmneal pweor of the hmuon
mnid, aoccdrnig to a rscheearch at Cmabrigde
Uinervtisy, it dseno't mtaetr in waht oerdr the ltteres in a
wrod are, the olny iproamtnt tihng is taht the frsit and
lsat ltteer be in the rghit pclae. The rset can be a taotl
mses and you can sitll raed it whotuit a pboerm. Tihs is
bcuseae the huamn mni d deos not raed ervey lteter by
istlef, but the wrod as a wlohe. Azanmig huh? yaeh and
I awlyas tghuhot slpeling was ipmorant! if you can raed
tihs forwrad it.

ONLY POST IF YOU CAN READ IT ?

So they created **Attention** — a mechanism to let each word **attend to** (focus on) all other words.

But they needed a **mathematical way to compute how much attention each word should pay to others.**

The Inspiration: Information Retrieval (Search Engines)

They borrowed an idea from **information retrieval / databases**.

When you search for something, you have:

- A **Query** (what you're looking for)
- **Keys** (descriptions of stored items)
- **Values** (actual stored information)

You compare your **query** with all **keys** → find **which items are relevant** → then **retrieve their values**.

💡 They thought:

“What if each word could *query* all other words to find which are most relevant, and then *retrieve* their meanings accordingly?”

The Implementation: Q, K, and V for words

So for each word/Token:

- It becomes a **Query** (**what I need to know**).
- It also has a **Key** (**what information I offer**).
- And a **Value** (**the meaning I carry**).

They compute **attention scores = $Q \times K^T$**

Then weight all **Values (V)** by those scores → that gives a new *context-aware representation*.

The Breakthrough: “Attention Is All You Need” (2017)

This paper replaced recurrence (RNN) with *pure attention* using this Q-K-V setup.

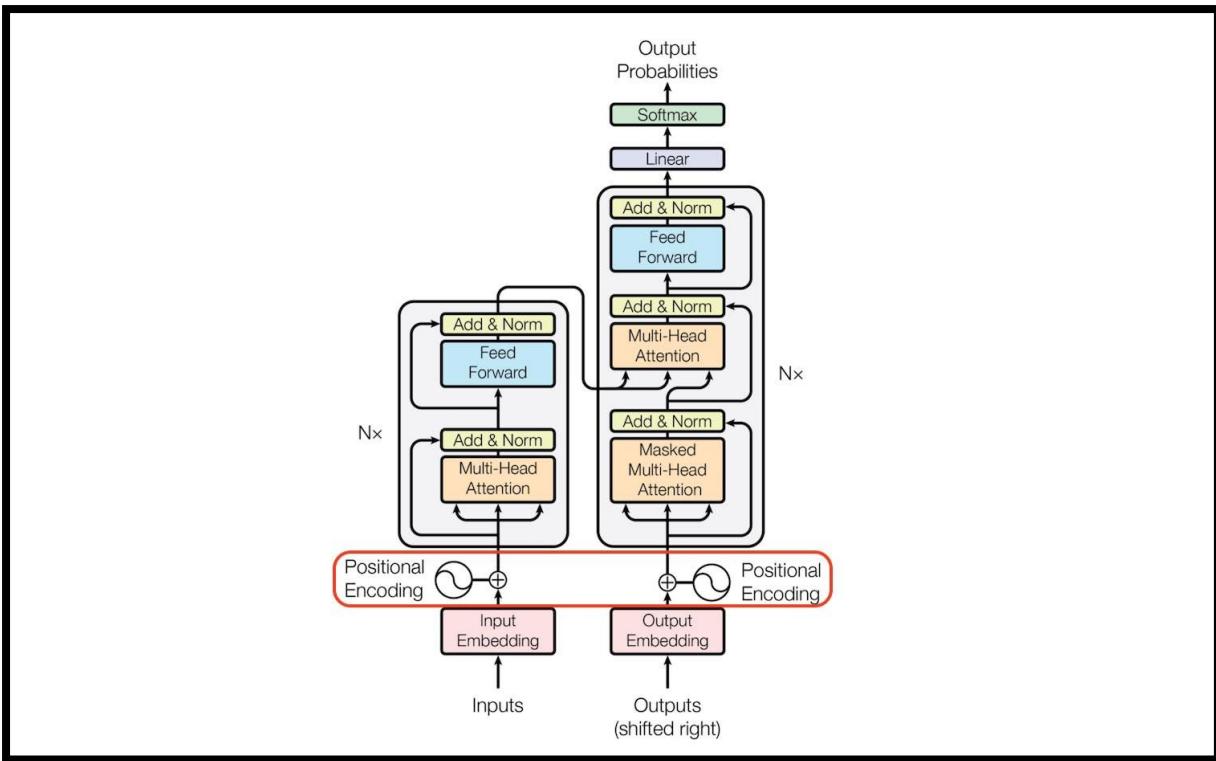
It allowed:

- Parallel processing of all words
- Better long-range understanding
- Huge efficiency and accuracy gains

In short

Came up with Q–K–V by combining:

- **Human intuition** (focus attention like we do)
- **Search engine math** (query, key, value)
- **Neural networks** (to learn what to attend to automatically)



Single-Head Attention

```
[56s] import numpy as np
import gensim.downloader as api
import matplotlib.pyplot as plt

# 1. Load pretrained GloVe embeddings
model = api.load("glove-wiki-gigaword-50")
words = ["the", "cat", "sat", "on", "table"]
X = np.array([model[word] for word in words])

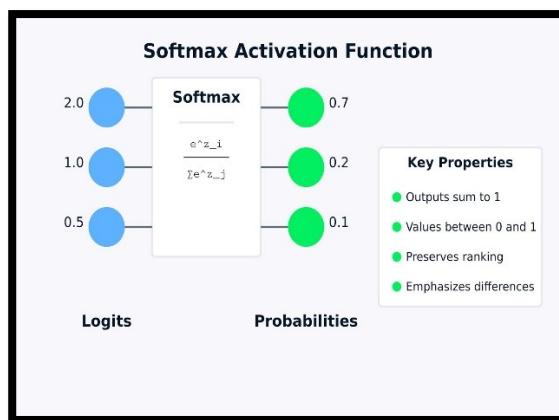
print(X.shape)
np.random.seed(42)

[=====] 100.0% 66.0/66.0MB downloaded
(5, 50)
```

Multiplying Matrices

$$\begin{bmatrix} 3 & 4 \\ 7 & 2 \\ 5 & 9 \end{bmatrix} \times \begin{bmatrix} 3 & 1 & 5 \\ 6 & 9 & 7 \end{bmatrix}$$

3 x 2 \leftarrow 2 x 3

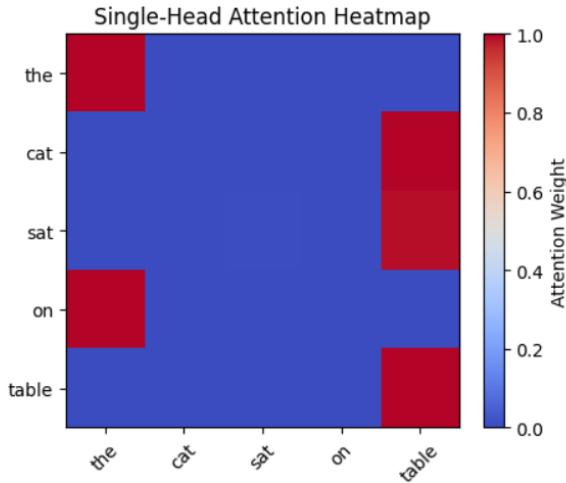


```
[1] head_dim = X.shape[1] # Single head covers all 50 dimensions
[2] 
[3] # Random weights for Q, K, V (50 -> 50)
[4] W_Q = np.random.rand(X.shape[1], head_dim)
[5] W_K = np.random.rand(X.shape[1], head_dim)
[6] W_V = np.random.rand(X.shape[1], head_dim)
[7] 
[8] # Compute Q, K, V
[9] Q = X @ W_Q
[10] K = X @ W_K
[11] V = X @ W_V
[12] 
[13] print(Q.shape)
[14] 
[15] (5, 50)
```

```
[1] scores = Q @ K.T / np.sqrt(head_dim)
[2] 
[3] # Softmax
[4] exp_x = np.exp(scores - np.max(scores, axis=-1, keepdims=True))
[5] attn_weights = exp_x / np.sum(exp_x, axis=-1, keepdims=True)
[6] 
[7] # Weighted sum
[8] output = attn_weights @ V
[9] 
[10] # -----
[11] # Show Results
[12] # -----
[13] print("Original shape:", X.shape)
[14] print("Output shape (single head):", output.shape)
[15] print("\nFirst word original (first 5 dims):", X[0][:5])
[16] print("First word attention output (first 5 dims):", output[0][:5])
[17] 
[18] # -----
[19] # Attention Heatmap
[20] # -----
[21] plt.figure(figsize=(5, 4))
[22] plt.imshow(attn_weights, cmap='coolwarm')
[23] plt.xticks(range(len(words)), words, rotation=45)
[24] plt.yticks(range(len(words)), words)
[25] plt.colorbar(label="Attention Weight")
[26] plt.title("Single-Head Attention Heatmap")
[27] plt.show()
```

```
→ Original shape: (5, 50)
Output shape (single head): (5, 50)

First word original (first 5 dims): [ 0.418    0.24968 -0.41242  0.1217   0.34527]
First word attention output (first 5 dims): [-4.31502585 -3.90981072 -2.11083877 -1.51211468 -1.90636801]
```



Multi-Head Attention

```
[9] 31s ➔ import numpy as np
import gensim.downloader as api
import matplotlib.pyplot as plt

# 1. Load pretrained GloVe embeddings
model = api.load("glove-wiki-gigaword-50")
words = ["the", "cat", "sat", "on", "table"]
X = np.array([model[word] for word in words]) # (5, 50)

# Multi-Head Self-Attention
np.random.seed(42)
num_heads = 10
head_dim = X.shape[1] // num_heads # 10 for 50-dim embeddings
print(X.shape)
```

→ (5, 50)

```

] 1s  all_heads = []
Os   attention_weights_all = []

    for h in range(num_heads):
        # Random weights for this head (50 -> head_dim)
        W_Q = np.random.rand(X.shape[1], head_dim)
        W_K = np.random.rand(X.shape[1], head_dim)
        W_V = np.random.rand(X.shape[1], head_dim)

        # Q, K, V
        Q = X @ W_Q
        K = X @ W_K
        V = X @ W_V
        print(Q.shape)

        # Attention scores
        scores = Q @ K.T / np.sqrt(head_dim)

        # Softmax
        exp_x = np.exp(scores - np.max(scores, axis=-1, keepdims=True))
        attn_weights = exp_x / np.sum(exp_x, axis=-1, keepdims=True)

        # Weighted sum
        out = attn_weights @ V

        all_heads.append(out)
        attention_weights_all.append(attn_weights)

    output_multihead = np.hstack(all_heads)

```

```

(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)

```

```

print("Original shape:", X.shape)
print("Multi-head output shape:", output_multihead.shape)
print("\nFirst word original (first 5 dims):", X[0][:5])
print("First word multi-head (first 5 dims):", output_multihead[0][:5])

Original shape: (5, 50)
Multi-head output shape: (5, 50)

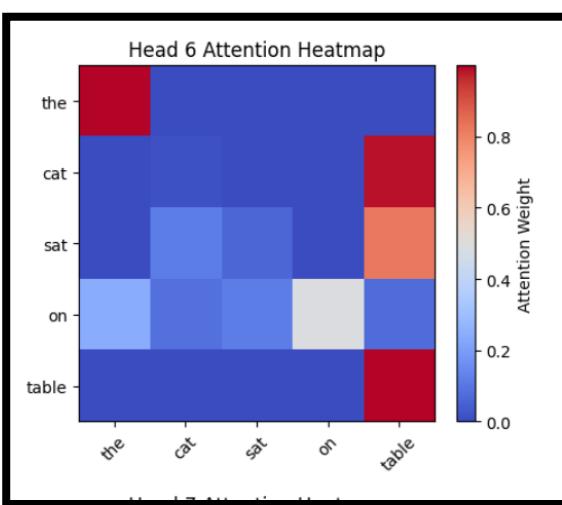
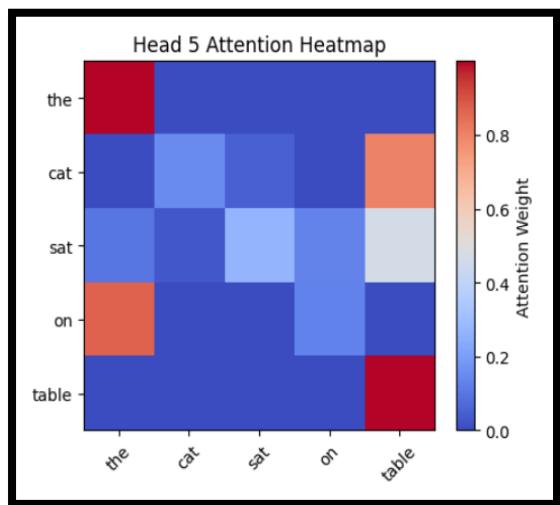
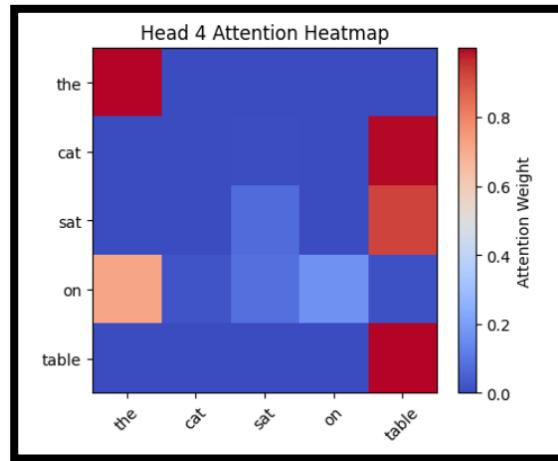
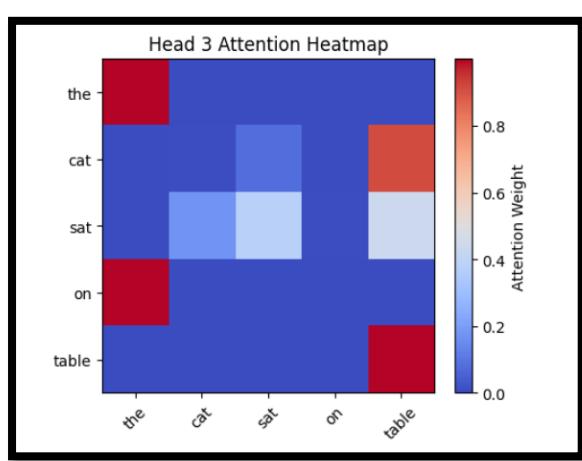
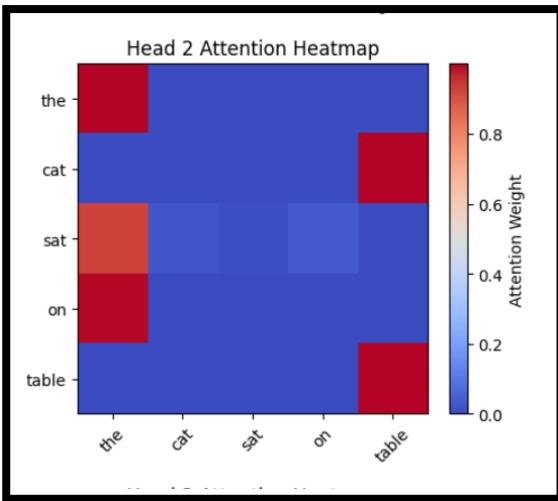
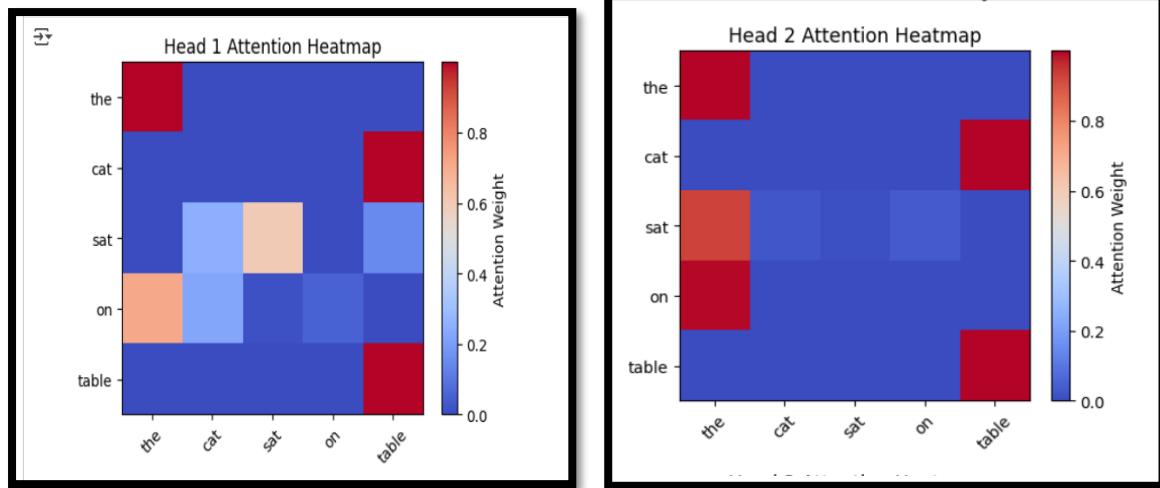
First word original (first 5 dims): [ 0.418   0.24968 -0.41242  0.1217   0.34527]
First word multi-head (first 5 dims): [-4.37999318 -1.99506183 -2.05712457 -0.08937276 -1.72769596]

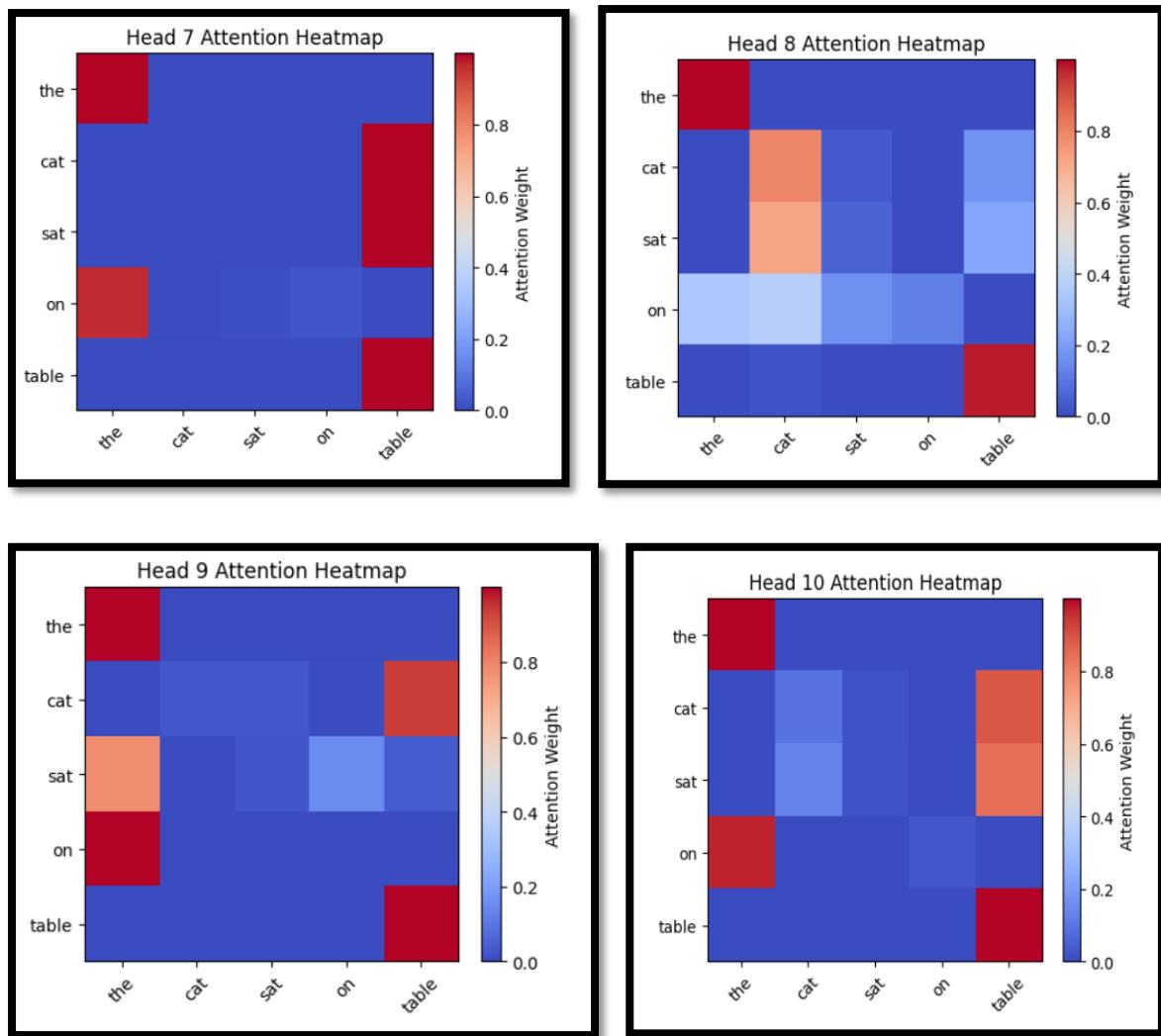
```

```

13] 1s  for h, attn_weights in enumerate(attention_weights_all):
    plt.figure(figsize=(5, 4))
    plt.imshow(attn_weights, cmap='coolwarm')
    plt.xticks(range(len(words)), words, rotation=45)
    plt.yticks(range(len(words)), words)
    plt.colorbar(label="Attention Weight")
    plt.title(f"Head {h+1} Attention Heatmap")
    plt.show()

```





Feature	Single Head	Multi Head
Context capture	One pattern at a time	Many patterns at once
Representation	Shallow	Deep and diverse
Long dependencies	Limited	Strong
Computation	Faster	Slightly heavier but more powerful

Add & Normalization

we apply **residual connection + layer normalization**:

1. Helps retain **original information** and **prevent vanishing gradients**.
2. Stabilizes **training by keeping values in a similar range**.
3. Ensures smooth learning even in very deep networks.

A Feed-Forward Network (FFN)

A **Feed-Forward Network (FFN)** in a Transformer is a **two-layer fully connected neural network** applied **separately and identically** to each token (position) in the sequence

Purpose: The attention mechanism is mostly linear. FFN introduces **non-linear transformations** (via ReLU/GELU) to help the model capture complex relationships.

Demo:

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

Thank You All

Any Questions /Query

What “dimension” means

The dimension (size) of a **Word2Vec** vector is the number of features in which each word is represented — e.g., 50, 100, 300, 768, etc.

It's not computed from data directly — it's a **hyperparameter chosen before training**.

Type	Strategy / Formula	Explanation
Rule of thumb	$d \approx \sqrt{V}$ or $d \approx \log_2(V)$	where V = vocabulary size. This ensures enough capacity to capture variety without overfitting.

Empirical tuning	Try $d = [50, 100, 200, 300]$	Train & evaluate on your downstream task (e.g., text classification). Pick the dimension that yields the best performance.
Corpus size-based	$d \propto (\text{Total tokens})^{0.25}$	Larger corpus \Rightarrow higher dimension can be used. Smaller corpus \Rightarrow smaller dimension to prevent overfitting.
Semantic complexity-based	If the domain has many subtle distinctions (like legal/medical text) \rightarrow higher d (e.g. 300–500). Simpler corpora (like tweets/news) \rightarrow smaller d (50–100).	

Corpus Type	Vocabulary Size	Suggested Dim
Small corpus (10k–50k words)	10k	50–100
Medium corpus (100k–500k)	100k	200–300
Large corpus (1M+)	1M	300–600

Intuitive analogy

Think of dimension d like resolution in an image:

- Low $d \rightarrow$ blurry meaning.
- High $d \rightarrow$ sharper meaning (but more costly to store/train).

Summary Formula-like Guide

If you want something actionable:

$$\text{Recommended Dim} = k \times \sqrt{\log(V)}$$

where k is between **30–50** depending on corpus complexity.