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"

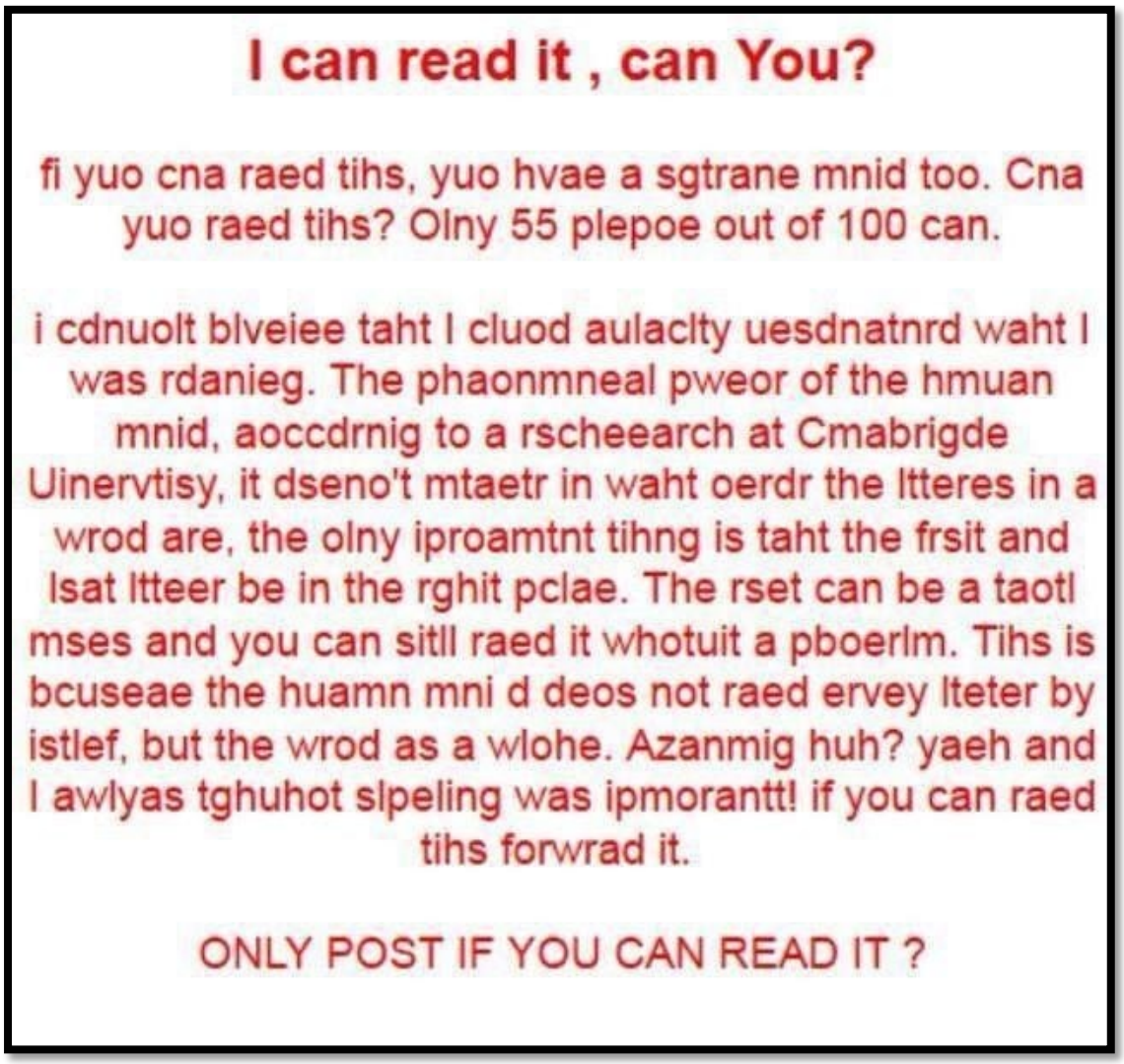
Techniques:

- **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**.”



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)

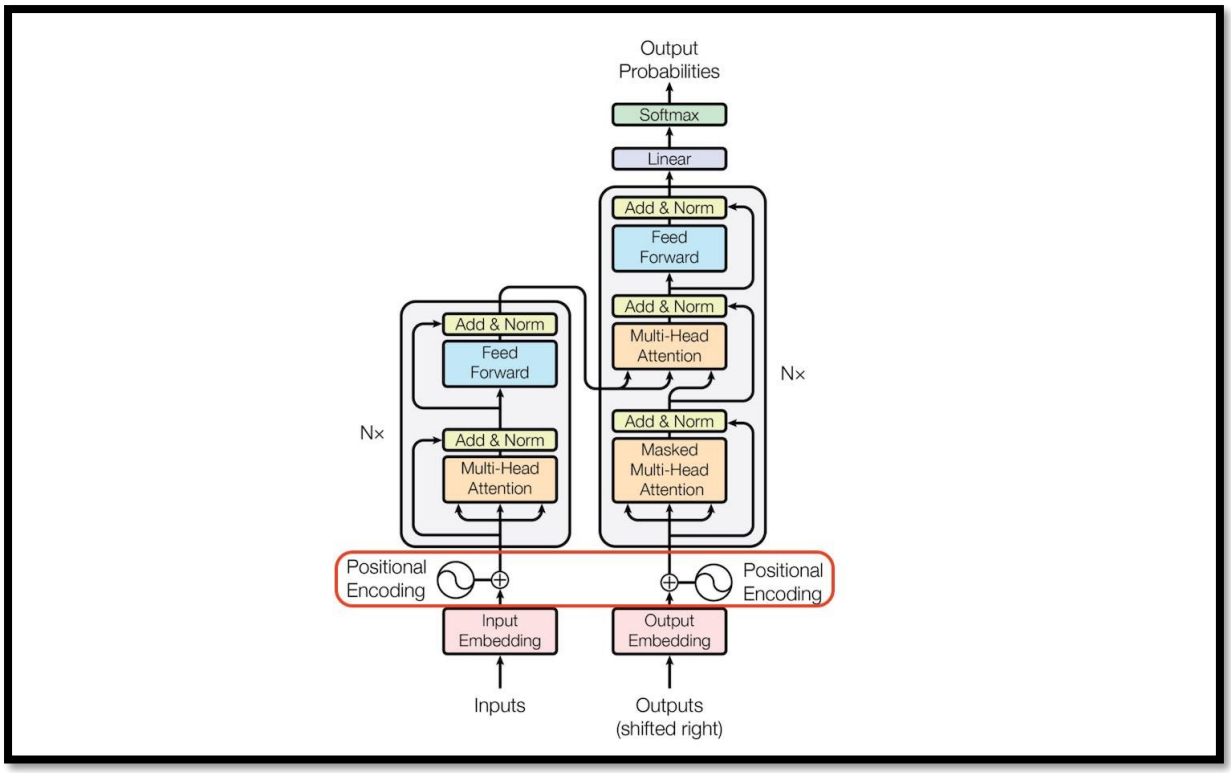
This paper replaced recurrence (RNN) with *pure attention* using this Q-K-V setup.
It allowed:

- 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)

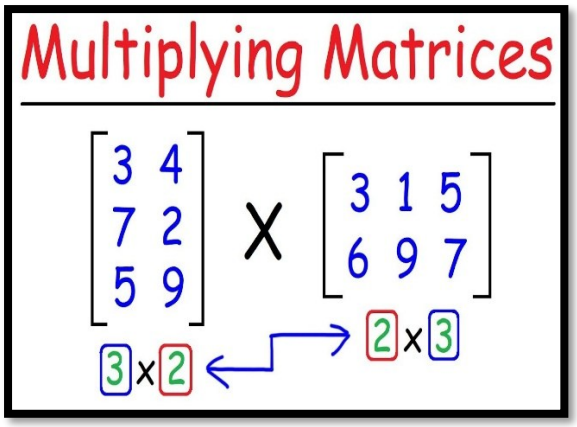


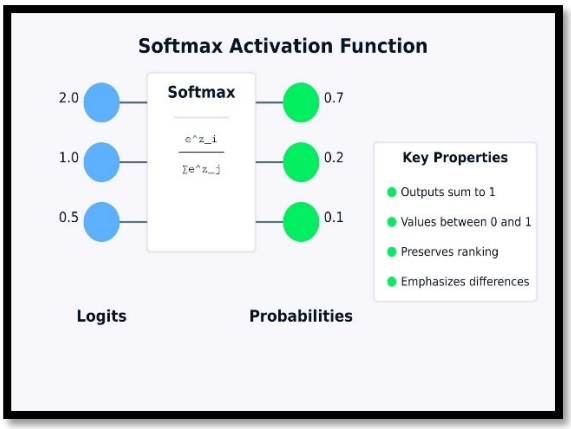
Single-Head Attention

```
]
56s
```

```python
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)
```





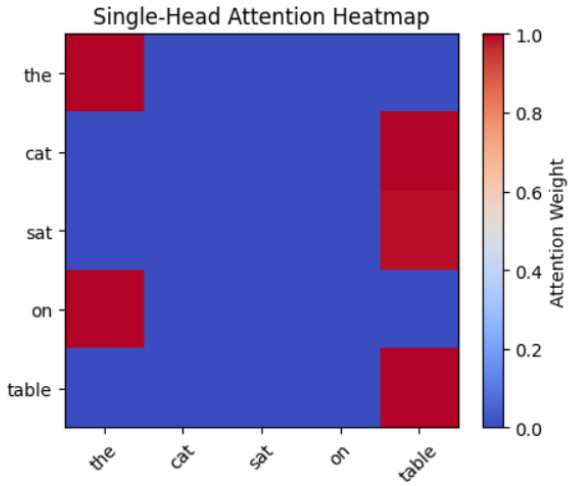

```
head_dim = X.shape[1]  # Single head covers all 50 dimensions

# Random weights for Q, K, V (50 -> 50)
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)

# Compute Q, K, V
Q = X @ W_Q
K = X @ W_K
V = X @ W_V

print(Q.shape)
```

```
(5, 50)
```

```
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
output = attn_weights @ V

# -------------------------
# Show Results
# -------------------------
print("Original shape:", X.shape)
print("Output shape (single head):", output.shape)
print("\nFirst word original (first 5 dims):", X[0][:5])
print("First word attention output (first 5 dims):", output[0][:5])

# -------------------------
# Attention Heatmap
# -------------------------
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("Single-Head Attention Heatmap")
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
```

```python
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)
```

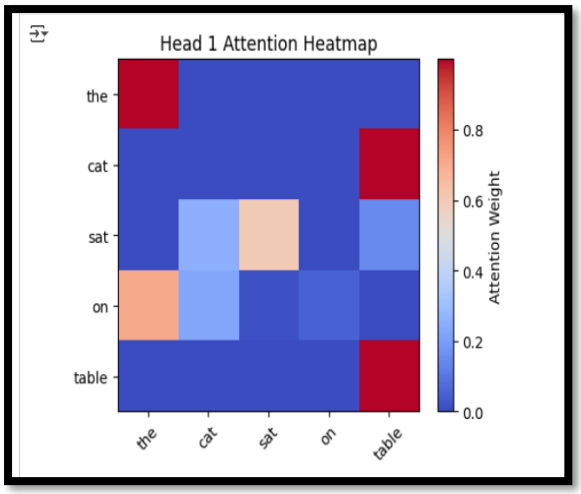
```
all_heads = []
attention_weights_all = []

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)
```

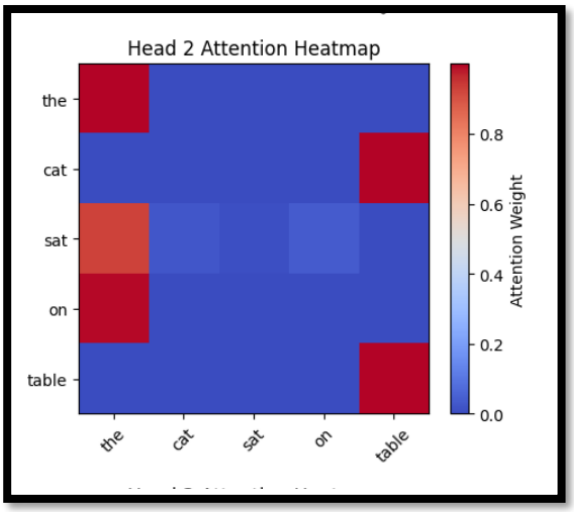
```
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
(5, 5)
```

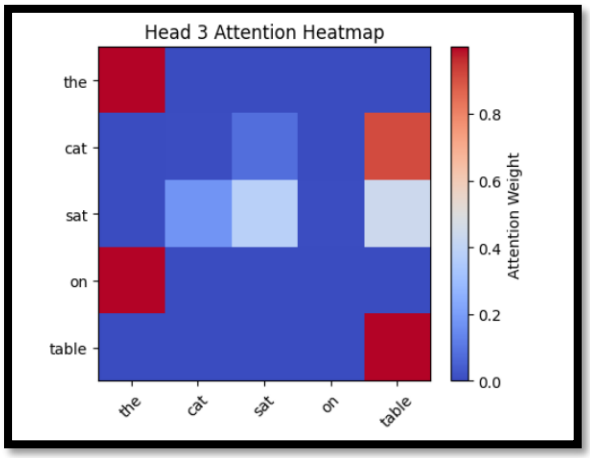
```
print("Original shape:", X.shape)
print("Multi-head output shape:", output_multihead.shape)
print("\nFirst word original (first 5 dims):", X[0][:5])
print("First word multi-head (first 5 dims):", output_multihead[0][:5])
```

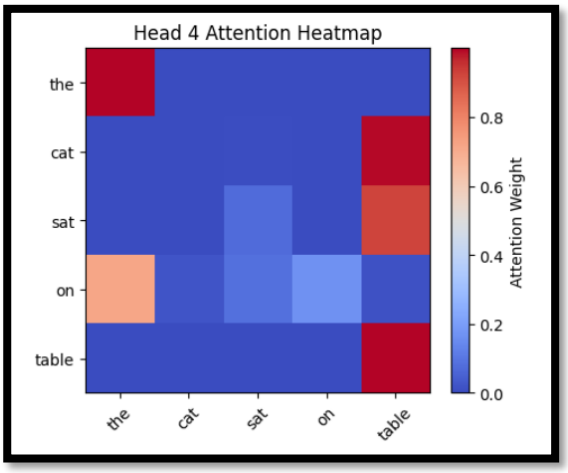
```
Original shape: (5, 50)
Multi-head output shape: (5, 50)

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]
```

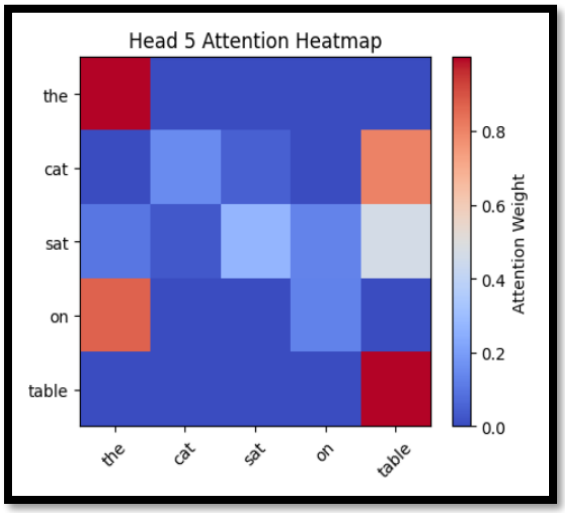
```
for h, attn_weights in enumerate(attention_weights_all):
    plt.figure(figsize=(5, 4))
    plt.imshow(attn_weights, cmap='coolwarm')
    plt.xticks(range(len(words)), words, rotation=45)
    plt.yticks(range(len(words)), words)
    plt.colorbar(label="Attention Weight")
    plt.title(f"Head {h+1}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

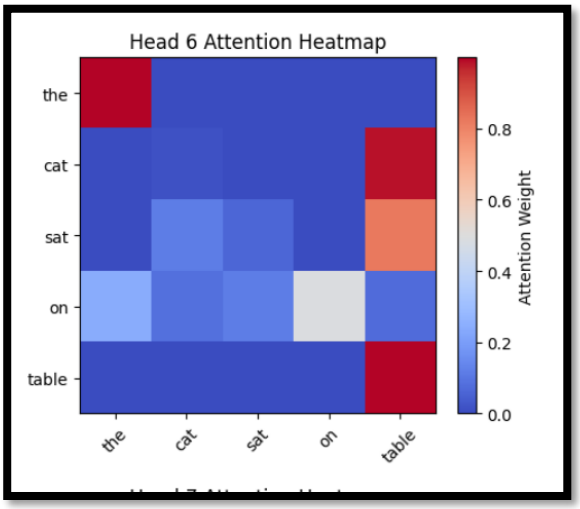

Head 1 Attention Heatmap
the
cat
sat
on
table
Attention Weight
0.0
0.2
0.4
0.6
0.8

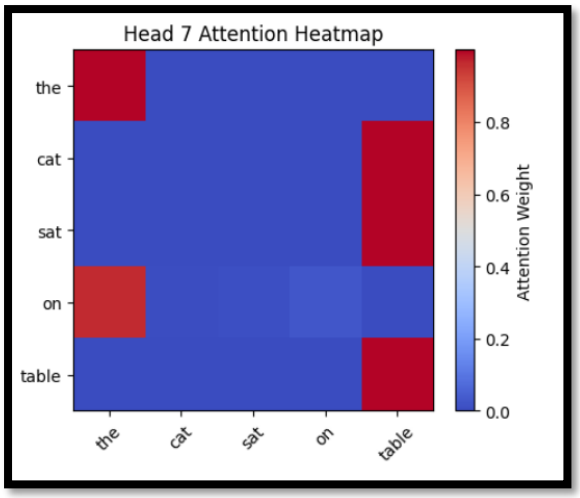

Head 2 Attention Heatmap
the
cat
sat
on
table
Attention Weight
0.0
0.2
0.4
0.6
0.8

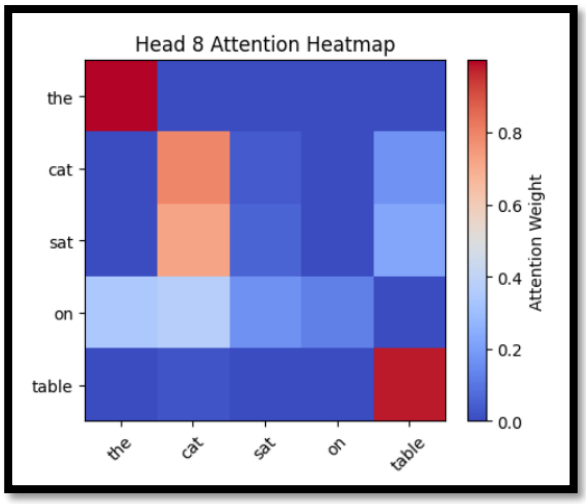

Head 3 Attention Heatmap
the
cat
sat
on
table
Attention Weight
0.0
0.2
0.4
0.6
0.8

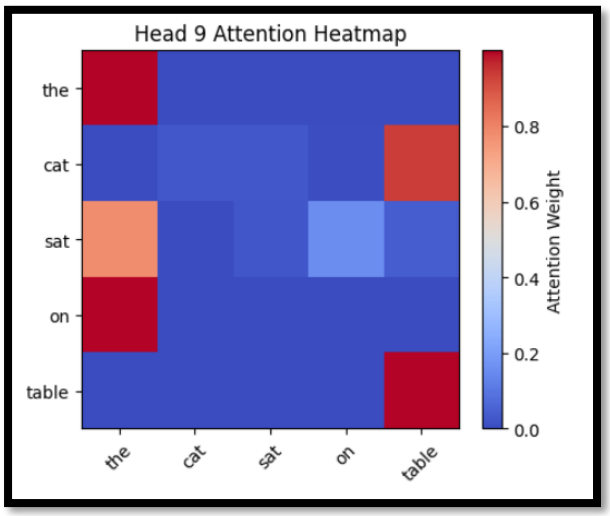

Head 4 Attention Heatmap
the
cat
sat
on
table
Attention Weight
0.0
0.2
0.4
0.6
0.8

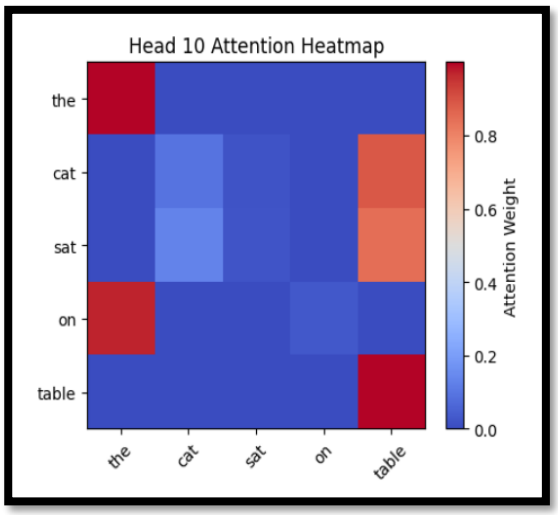

Head 5 Attention Heatmap
the
cat
sat
on
table
Attention Weight
0.0
0.2
0.4
0.6
0.8


Head 6 Attention Heatmap
the
cat
sat
on
table
Attention Weight
0.0
0.2
0.4
0.6
0.8









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