# Multi-Head Self-Attention

How the Transformer looks at multiple relationships simultaneously

# Why Multiple Heads?

A single attention head focuses on one type of relationship, but language is complex and multi-faceted. Multiple heads enable the model to learn different relationships in parallel, capturing diverse linguistic patterns simultaneously.

### Head 1: Coreference

Tracks pronouns and their references—like connecting "it" back to "animal" in a sentence

### Head 2: Grammar

Focuses on grammatical structures such as verb negation patterns like "didn't cross"

### Head 3: Syntax

Captures syntactic dependencies, including prepositional relationships like "on → street"

# Process Overview

Each attention head operates as an independent learner, processing the input through its own lens before combining insights. Here's how the multi-head mechanism works:

### Independent Weight Matrices

Each head maintains its own learnt Q, K, and V weight matrices, enabling specialisation in different aspects of the input

### Parallel Self-Attention

All heads perform self-attention computations independently and simultaneously, maximising efficiency

### Context Vector Generation

Each head produces its own set of context-aware output vectors capturing unique relationships

### Concatenation & Projection

All head outputs are concatenated together and projected back to the original dimension through a learnt linear transformation

# Per-Head Computation

Each head *h* transforms the input queries, keys, and values through its own weight matrices before computing attention. This allows different heads to focus on different representational subspaces.

$$\text{head}_h = \text{Attention}(QW_Q^h, KW_K^h, VW_V^h)$$

The Attention function here is the same scaled dot-product attention we've seen before: computing similarities between queries and keys, applying softmax, and weighting the values accordingly.

| Symbol | Meaning |
| --- | --- |
| h | Head index (which attention head we're referring to) |
| $W_Q^h, W_K^h, W_V^h$ | Learnt weight matrices specific to head *h* |
| Attention(·) | Scaled dot-product attention: Q · $K^T$ → Softmax → weighted V |
| $\text{head}_h$ | Output vector set produced by head h |

# Combining the Heads

After all heads complete their independent attention computations, we need to merge their insights into a single, unified representation. This is achieved through concatenation followed by a learnt linear projection.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

### Concat( )

Joins all head outputs along the feature dimension, creating a larger combined vector

### $W_O$

Output projection matrix that transforms the concatenated representation back to the model dimension

### MultiHead(Q,K,V)

The final combined output capturing multiple perspectives simultaneously

# Example: 8-Head Attention

In practice, transformers commonly use 8 attention heads. Each head specialises in capturing a different linguistic phenomenon, working together to build a comprehensive understanding of the input.

1. **Subject–Verb Agreement**

   Ensures grammatical concordance between subjects and their verbs across long distances

2. **Object Relations**

   Tracks how objects relate to verbs and other sentence constituents

3. **Positional Cues**

   Captures sequential and positional information within the sentence structure

4. **Semantic Relationships**

   Identifies meaning-based connections between words and phrases

All these diverse perspectives are merged to create richer, more nuanced representations that capture the full complexity of language.

# Key Advantages

### Parallel Processing

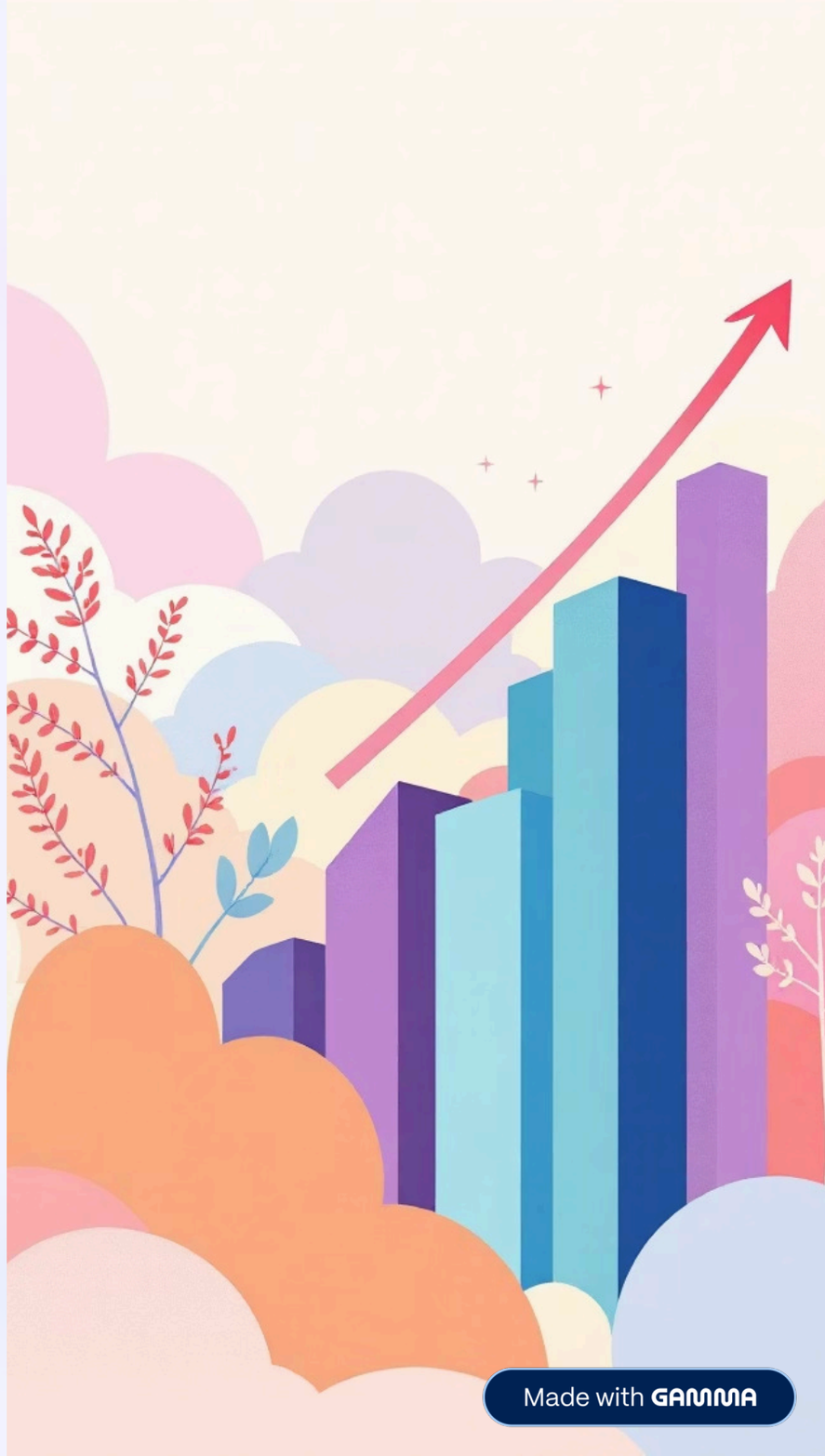
All heads compute attention simultaneously rather than sequentially, dramatically reducing training and inference time compared to recurrent architectures

### Multiple Perspectives

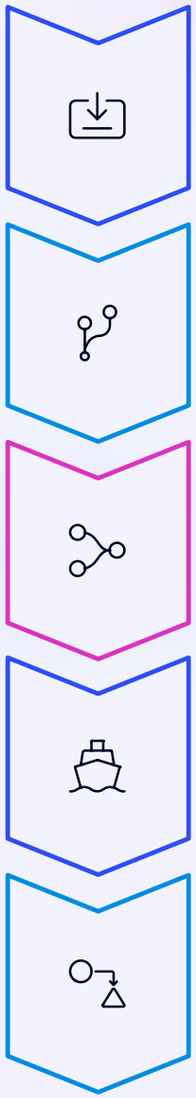
Different heads capture complementary aspects of the input, from syntactic structure to semantic meaning, creating richer contextual representations

### Improved Accuracy

The diverse attention patterns learnt across heads lead to better performance in complex tasks like machine translation and text generation



# Architecture Visualisation



### Input Embeddings

Q, K, V matrices

### Multi-Head Attention

Parallel processing across h heads

### Concatenate

Join head outputs

### Linear Projection

Apply $W_O$

### Output

Final representation

This pipeline executes for every token position in the sequence, with all heads working in parallel to capture diverse relationships before merging their insights.

# Formula Cheat Sheet

A quick reference guide to the key symbols and operations in multi-head self-attention:

| Symbol | Meaning |
| --- | --- |
| Q, K, V | Input matrices for queries, keys, and values derived from the input embeddings |
| $W_Q^h, W_K^h, W_V^h$ | Head-specific weight matrices that transform Q, K, V for each attention head |
| $W_O$ | Output projection weight matrix that combines concatenated head outputs |
| h | Head number or index (e.g., 1 to 8 in an 8-head architecture) |
| Concat | Concatenation operation that joins all head outputs along the feature dimension |
| Attention(·) | Scaled dot-product self-attention function: Q · $K^T$ → Scale → Softmax → weighted V |
| MultiHead(Q,K,V) | The complete multi-head attention output combining all head perspectives |

# Summary

| Concept | Purpose |
| --- | --- |
| **Self-Attention** | Finds word-to-word relationships by computing attention scores between all token pairs in the sequence |
| **Multi-Head Attention** | Learns multiple relationships simultaneously by running several attention mechanisms in parallel, each with its own weights |
| **Scaling & Softmax** | Stabilises gradients through scaling and normalises attention scores into a probability distribution via softmax |
| **Linear Projection** | Combines outputs from all heads into a unified final representation through a learnt transformation matrix |

Multi-head self-attention is the core innovation that enables transformers to capture rich, diverse linguistic patterns efficiently and in parallel—revolutionising natural language processing.