# **How Transformers Work: Embeddings and Self-Attention Explained**

Transformers power modern AI models like GPT, BERT, and ChatGPT.  
 They mainly rely on two core ideas:

1. **Embeddings** – turning words into numerical vectors that capture meaning
2. **Self-Attention** – allowing the model to focus on relevant words in context

## **1. Embeddings — Turning Words into Meaningful Numbers**

Computers cannot understand words directly, so each word must be converted into a numerical representation called an **embedding**.

### **What are embeddings?**

An embedding represents a word as a vector in a high-dimensional space (e.g., 300 or 512 dimensions).  
 Words with similar meanings end up close together in that space.

| **Word** | **Embedding (shortened)** |
| --- | --- |
| king | [0.21, -0.55, 0.68, ...] |
| queen | [0.23, -0.52, 0.70, ...] |
| apple | [-0.40, 0.88, -0.33, ...] |

This allows the model to understand relationships between words, such as:  
 **king - man + woman ≈ queen**

## **2. Positional Encoding — Teaching Word Order**

Transformers process all words simultaneously, unlike RNNs, which process words one by one.  
 Because of this, transformers have no natural sense of word order.  
 To fix this, **positional encodings** are added to the embeddings. These tell the model where each word appears in the sentence.

Example sentence: “I love NLP”

| **Word** | **Embedding** | **Position Encoding** | **Final Input (E + P)** |
| --- | --- | --- | --- |
| I | [0.1, 0.8, 0.5] | [0.5, 0.4, 0.3] | [0.6, 1.2, 0.8] |
| love | [0.9, 0.1, 0.7] | [0.6, 0.2, 0.1] | [1.5, 0.3, 0.8] |
| NLP | [0.3, 0.4, 0.2] | [0.7, 0.1, 0.2] | [1.0, 0.5, 0.4] |

Positional encodings usually follow a wave-like pattern (using sine and cosine functions) so the model can infer the relative positions of words.

### **Diagram: How Input Becomes Embeddings**

Sentence: "The cat sat on the mat"

Tokens: [The] [cat] [sat] [on] [the] [mat]

↓ Token Embeddings

[E1] [E2] [E3] [E4] [E5] [E6]

+ Add Positional Encodings

↓ Final Input to Transformer

[E1+P1] [E2+P2] [E3+P3] [E4+P4] [E5+P5] [E6+P6]

Now the model knows both the meaning and the position of each word.

## **3. Self-Attention — Understanding Context**

Self-attention allows the model to understand how each word relates to the others in the sentence.  
 For instance, in the sentence “The cat sat on the mat,” to understand the word “sat,” the model should pay attention to:

* “cat” (who sat)
* “mat” (where it happened)

That is what self-attention achieves.

### **Step-by-Step Process of Self-Attention**

Each word embedding is transformed into three vectors:

* **Q (Query)** – represents what the word is looking for
* **K (Key)** – represents what information the word contains
* **V (Value)** – represents the information that can be shared

These are learned during training.

#### **Step 1: Compute Attention Scores**

For each pair of words, the dot product of their Query and Key vectors is calculated:

Attention Score = Q • K

This measures how much one word should pay attention to another.

#### **Step 2: Normalize with Softmax**

A softmax function converts these scores into probabilities so that they sum to 1.  
 Relevant words get higher weights.

#### **Step 3: Weighted Sum of Values**

Each Value (V) is multiplied by its attention weight, and the results are summed to form a new, **context-aware representation** for each word.

### **Example Visualization**

For the sentence “The cat sat on the mat,” when focusing on “sat”:

Sentence: The cat sat on the mat

↑

Focus word: “sat”

Attention weights:

The → 0.05

cat → 0.40 (who sat)

sat → 0.10

on → 0.15

the → 0.10

mat → 0.20 (where sat)

Weighted sum → new vector for “sat”

This means “sat” now has a contextual meaning influenced by “cat” and “mat,” helping the model grasp the sentence’s overall meaning.

## **4. Multi-Head Attention — Seeing Multiple Relationships**

Rather than performing attention once, transformers use multiple “heads” in parallel.  
 Each head learns to focus on a different kind of relationship.

| **Head** | **What it learns** |
| --- | --- |
| Head 1 | Subject–verb relationships |
| Head 2 | Object–action relationships |
| Head 3 | Long-distance dependencies |
| Head 4 | Grammatical structure |

The outputs of all heads are concatenated and linearly combined to form a richer understanding.

### **Diagram: Multi-Head Attention**

Input Embeddings

│

▼

┌─────────────────────────────┐

│ Head 1: short-range context │

│ Head 2: long-range relation │

│ Head 3: syntax structure │

│ Head 4: topic focus │

└─────────────────────────────┘

│

Concatenate + Linear Projection

│

▼

Final Contextual Representation

This enables the model to understand multiple layers of relationships between words simultaneously.

## **5. Putting It All Together**

Overall process for a single Transformer layer:

Text Input → Tokenization

↓

Word Embeddings + Positional Encoding

↓

Self-Attention (multiple heads)

↓

Feed-Forward Network

↓

Output (used by next layer or decoder)

Each layer refines the contextual understanding of the input sentence.