🔍 Step 3: Self-Attention — How Transformers Understand Context

"Every word looks at every other word"

A deep dive into the mechanism that enables transformers to capture meaning and relationships in language

The Problem — Why Do We Need Attention?

Consider This Sentence:

> "The animal didn't cross the street because it was too tired."

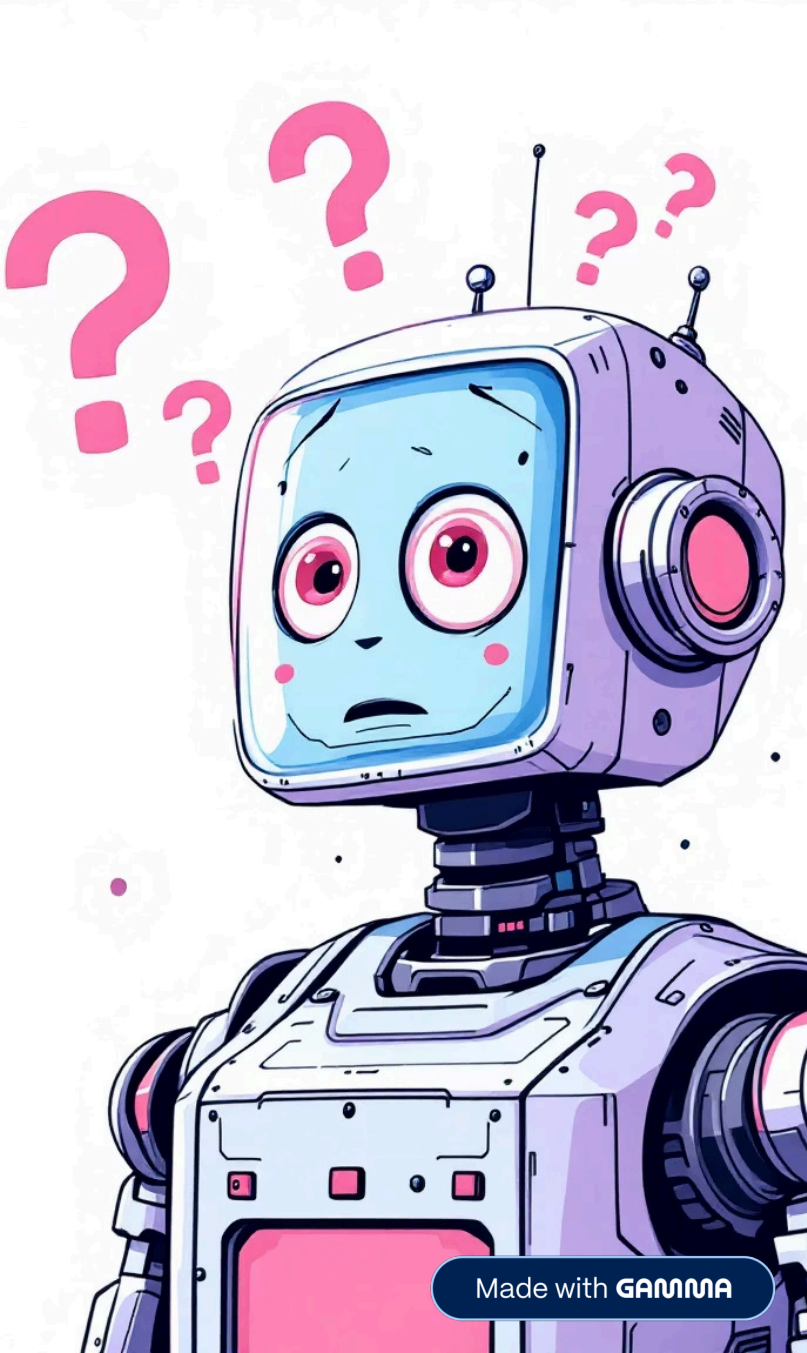
Question: What does **"it"** refer to?

🐾 Answer: The **animal**, not the street.

The Challenge

How does a machine understand this connection? Traditional models like RNNs struggle with long-distance relationships between words.

Transformers solve this elegantly through **Self-Attention** — a mechanism that lets every word examine every other word in context.



What Is Self-Attention?

The Core Idea

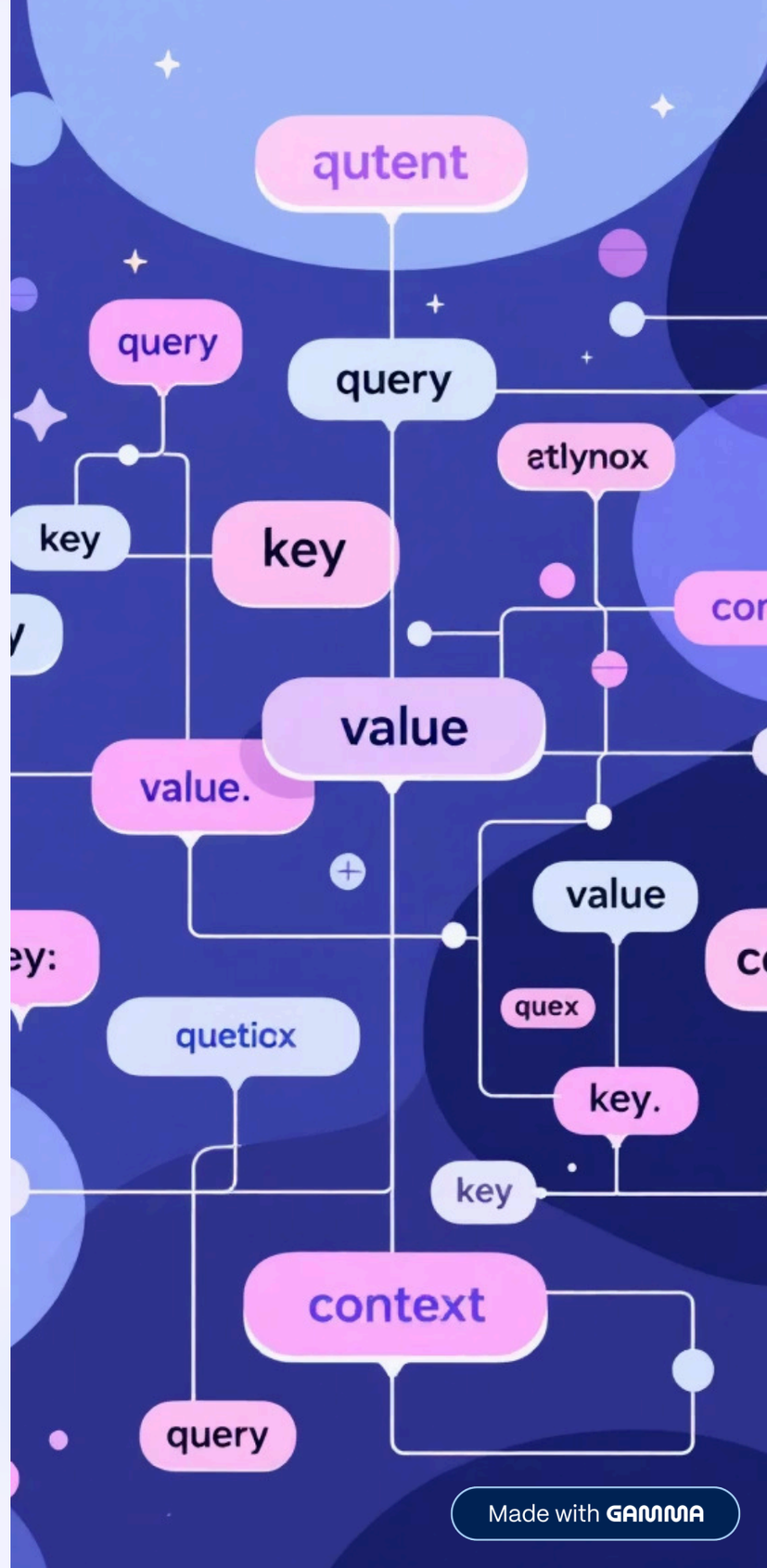
Self-Attention means each word **looks at every other word** in the sentence and decides **how much attention** to give them.

Dynamic Weighting

The model builds understanding by weighting relationships dynamically based on relevance and context.

Example in Action

1. When processing **"it"**, the model:
2. Looks at **"animal"** → **High importance** ✓
3. Looks at **"street"** → Low importance



Why Is Self-Attention So Important?

Context Awareness

Self-attention learns which words relate to each other, capturing semantic relationships that matter for meaning.

Long-Range Dependence

Understands relationships even when words are far apart in a sentence, unlike RNNs which forget over distance.

Dynamic Flexibility

Adjusts focus per word dynamically based on context — no rigid patterns, pure learned understanding.

Rich Representations

Every output vector contains information from the whole sentence, creating truly context-aware word representations.

Self-Attention Through a Human Lens

How Humans Process Language

When we read:

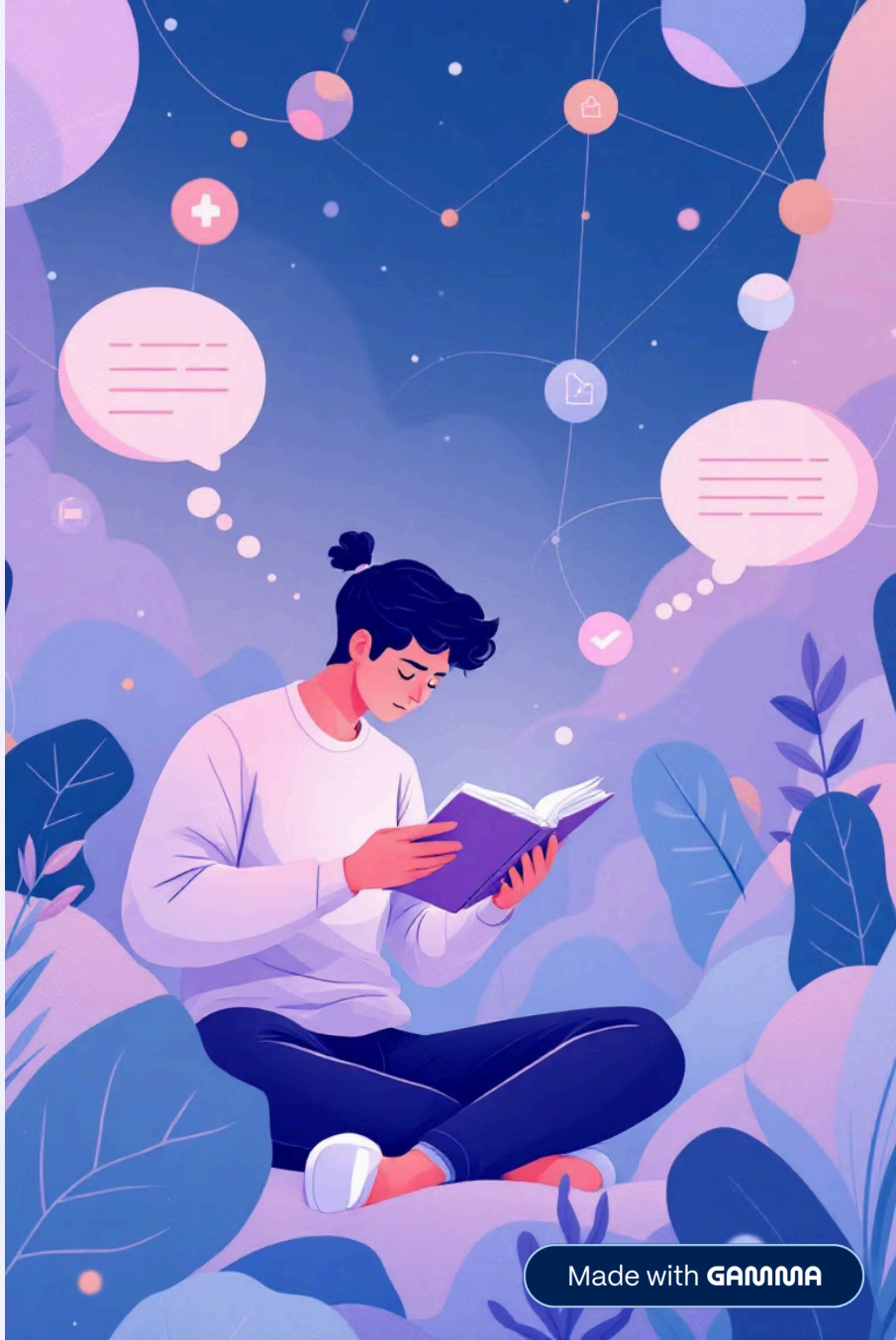
> "The animal didn't cross the street because it was tired."

We *mentally connect* "it" → "animal" automatically. Our brain doesn't process words in isolation.

How Transformers Do It

Self-attention replicates this human ability — it learns these **connections** automatically through training.

Each word becomes context-aware through "looking" at others. The model learns what to pay attention to.



The Main Idea — A High-Level Overview

Self-attention transforms each word by considering all other words in the sequence. Here's how it works:

Create Q, K, and V Vectors

For each word, generate three new vectors: **Query (Q)**, **Key (K)**, and **Value (V)** through learned transformations.

Compare and Score

Compare each word's **Query** with all **Keys** to find relevance scores — how much should this word attend to others?

Apply Softmax

Convert raw scores into probabilities using Softmax, ensuring all attention weights sum to 1.

Weighted Combination

Use attention scores to create a weighted mix of **Value** vectors, producing a new context-rich representation.

Result: Each word gets an updated vector rich with **context** from all other words in the sentence.

Step 1: Creating Q, K, and V Vectors

From each word embedding (dimension = d_model, typically 512), we create three distinct vectors through learned linear transformations:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

Each **W** is a **learned weight matrix** that the model trains to capture different aspects of word meaning.

Q (Query)

Represents what the word is *looking for* — what information it needs from other words.

K (Key)

Represents what the word *contains* — the information it can provide to others.

V (Value)

The *actual information* of the word that will be mixed and passed forward.

Symbol	Meaning
X	Input embeddings (sentence matrix)
WQ, WK, WV	Learned weight matrices
Q, K, V	Query, Key, Value matrices

Step 2: Computing Attention Scores

Now we compare the Query of one word with the Keys of all other words to determine relevance:

$$\text{Score}_{ij} = Q_i \cdot K_j^T$$

This dot product shows **how much word *i* should pay attention to word *j***. Higher scores mean stronger connections.

Example: Processing "it"

When computing attention for the word "it" in our example sentence:

Compared To	Score
"animal"	0.9
"street"	0.2
"tired"	0.6

The high score for "animal" indicates a strong relationship!

Symbol Guide

- **Qi**: Query for word *i*
- **Kj**: Key for word *j*
- **·**: Dot product (measures similarity)
- **T**: Transpose operation
- **Scoreij**: Attention of word *i* to word j

Step 3: Scaling and Applying Softmax

Why Scale?

To keep scores stable and prevent gradients from vanishing, we scale by the square root of the key dimension:

$$\text{Scaled Score}_{ij} = \frac{Q_i \cdot K_j^T}{\sqrt{d_k}}$$

Converting to Probabilities

We then apply **Softmax** to convert raw scores into proper probability distributions (all weights sum to 1):

$$\text{AttentionWeight}_{ij} = \text{Softmax}(\text{ScaledScore}_{ij})$$

"animal"

Strong attention

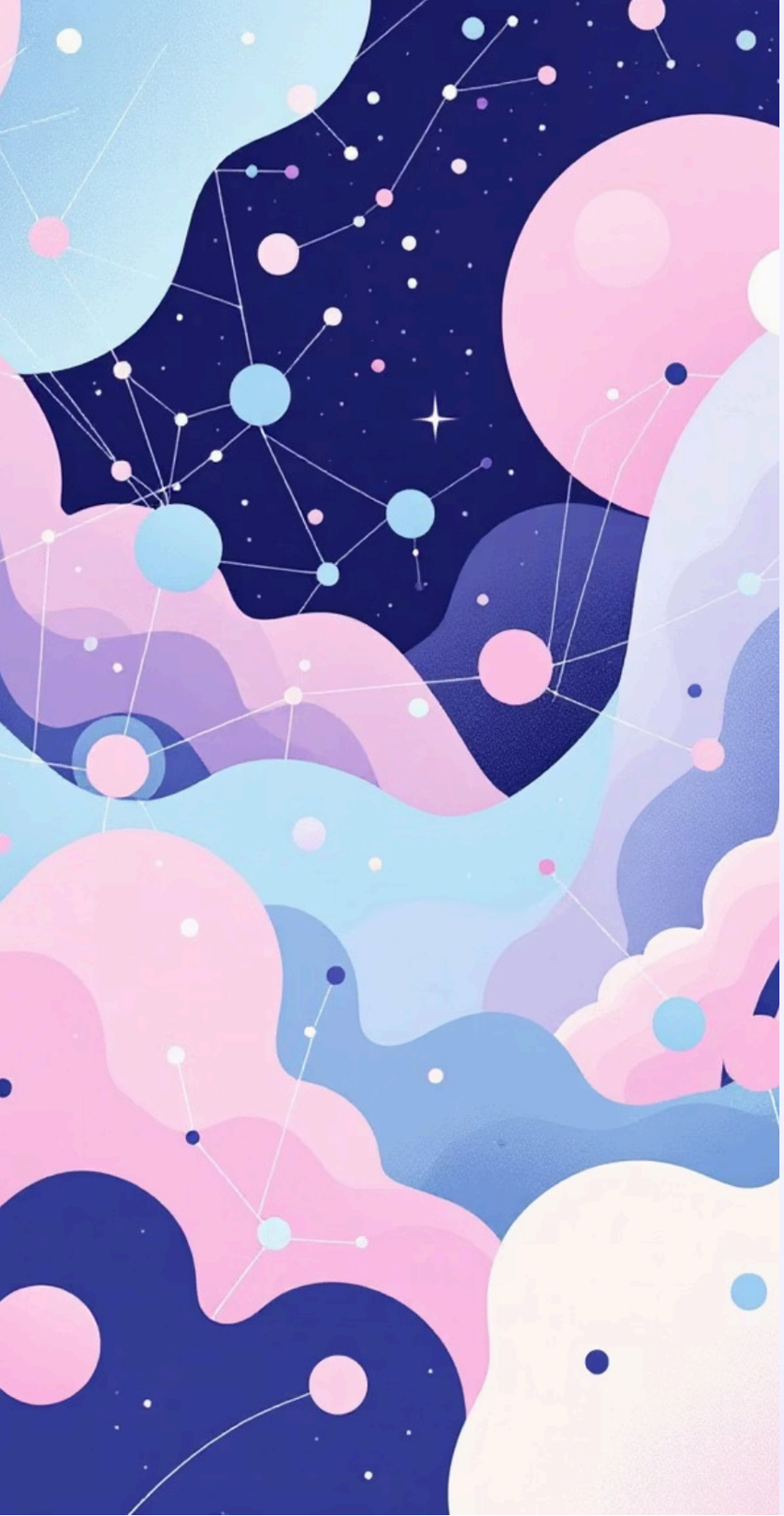
"street"

Minimal attention

"tired"

Moderate attention

Symbol	Meaning
dk	Dimension of Key vector (typically 64)
√dk	Scaling factor — keeps values numerically stable
Softmax	Transforms scores → probabilities (sum = 1)



Step 4: Creating Context-Rich Representations

Finally, each word blends the **Value vectors (V)** of all words using the computed attention weights:

$$\text{Output}_i = \sum_j \text{Weight}_{ij} \times V_j$$

What This Means

The new vector for "it" becomes mostly "animal" information, plus a bit from other relevant words.

Every word now holds **sentence-level understanding**!

Symbol Reference

- **Vj**: Value vector of word *j*
- **×**: Element-wise multiplication
- **∑j**: Sum over all words
- **Outputi**: Context-rich vector for word *i*

Key Insight: Each output vector mixes information from all words based on relevance, creating truly **context-aware representations** that capture the meaning and relationships within the entire sequence.