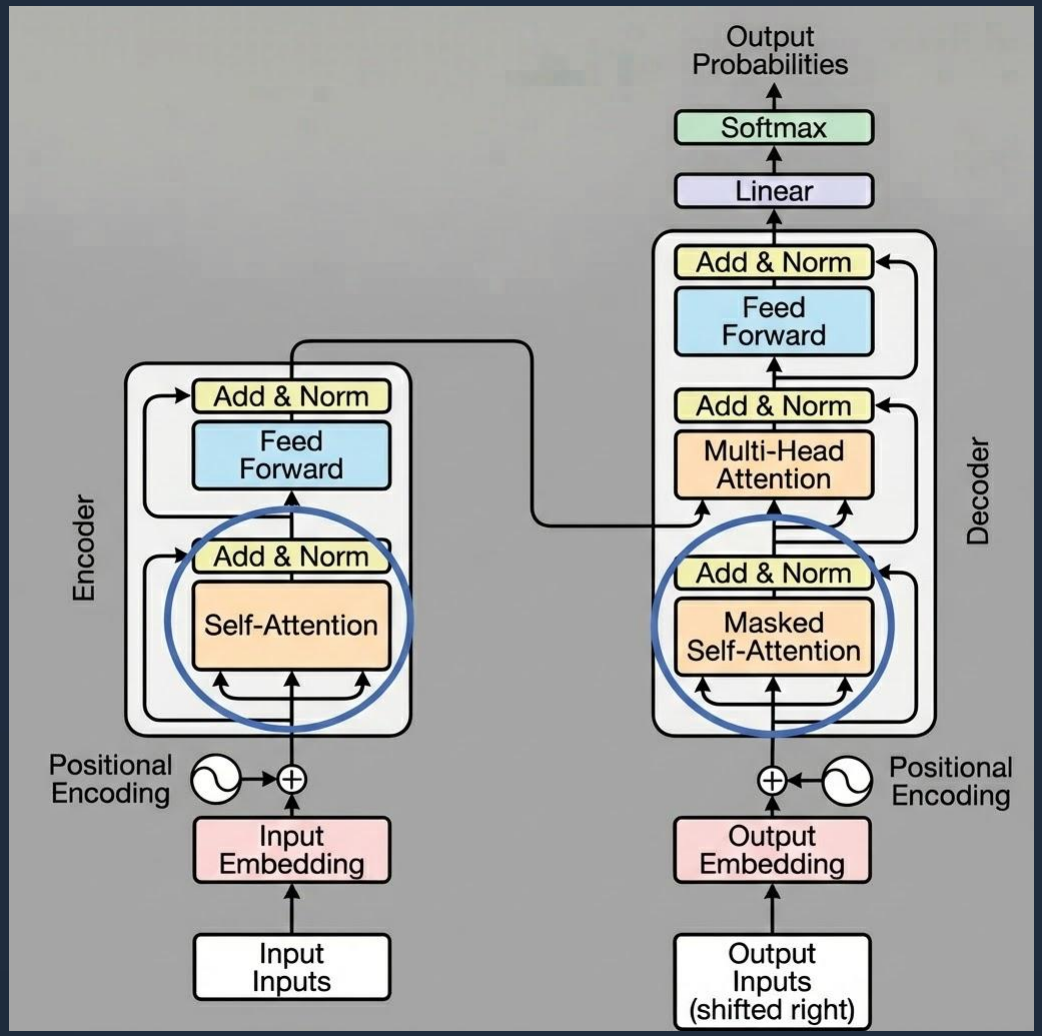
Self-Attention in Transformer

Transformer architecture

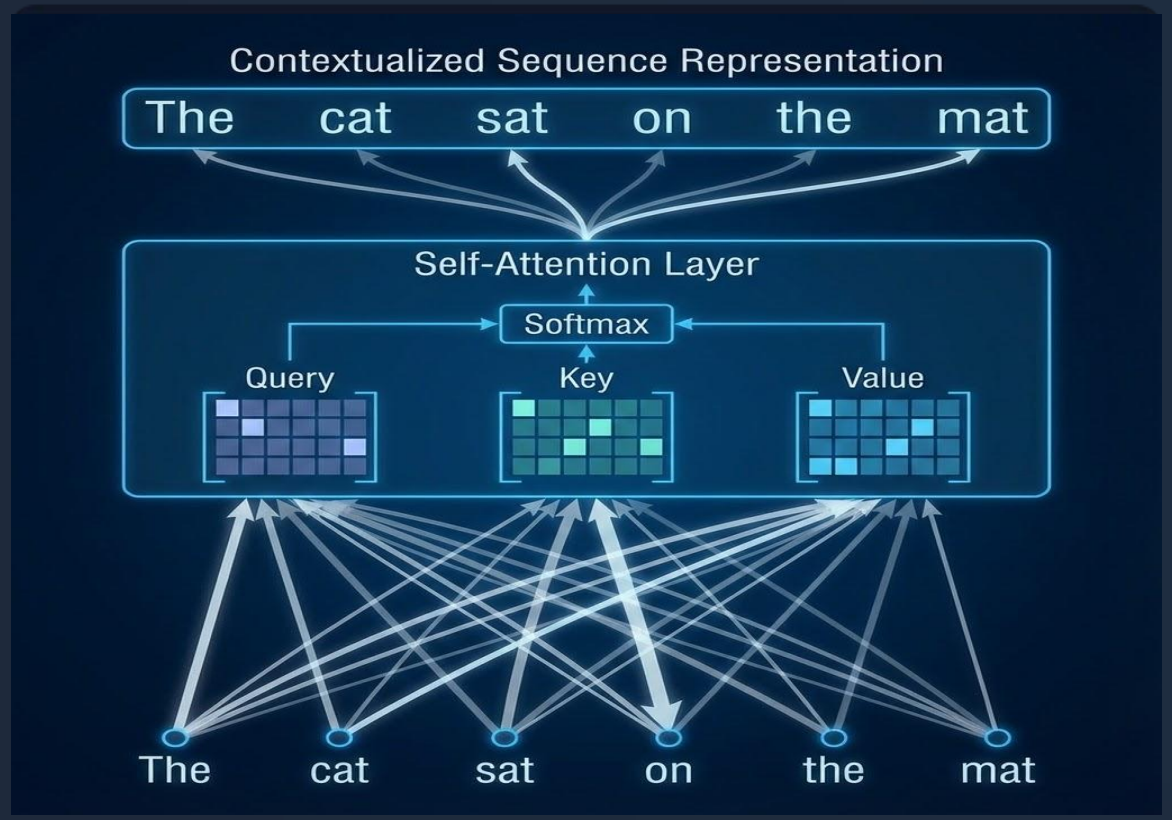


Why Self-Attention? The RNN Problem

- **Sequential Processing:** RNNs/LSTMs read left-to-right. Information must pass through every single step.
- **Signal Decay:** The information signal from an early, important token (like a subject) weakens significantly as the recurrent network processes a long sequence.
- **Result:** Due to the severe signal decay over a long sequence, the model effectively "forgets" the original subject (e.g., "The keys") by the time it reaches the necessary predicate (e.g., "are"), leading to difficulty in resolving grammatical dependencies

Key Takeaway: Sequential models struggle with long-range dependencies because information decays over time steps.

What is Self-Attention?



Definition

A mechanism that allows a model to relate different positions of a single sequence to compute a representation of the sequence itself.

The "Dictionary" Analogy

- When you read the word **"bank"**, you don't know if it means *river bank* or *financial bank*.
- You look at other words like **"money"** or **"water"** to understand the context.
- **Self-Attention** is the mathematical process of looking at the rest of the sentence to clarify the meaning of the current word.

The Self-Attention Solution

Parallel Computation

- **Simultaneous Processing:** Unlike RNNs that process tokens one by one (Sequential), Self-Attention computes relationships for all tokens at the same time.
- **Modern Hardware:** This perfectly exploits GPUs/TPUs, which are designed for massive parallel operations.
- **Result:** Dramatically faster training times for large datasets.

Key Takeaway: Self-attention trades sequential processing for parallel pairwise comparisons, solving the long-range dependency problem.

The Self-Attention Solution

Global context

- **O(1) Path Length:** Every token can attend to every other token directly. There is no "distance" in the network topology.
- **Vanishing Gradient? Gone.** Information doesn't decay as it travels through time steps (because there are no time steps).
- **Dynamic Weights:** Weights are computed from the *content* (I love apple), not just fixed parameters.
- **Result:** The model can pay equal attention to "The keys" (start) and "are" (end) simultaneously.

The Core Formula

$$\text{Attention}(Q, K, V) = \text{softmax}\frac{QK^T}{\sqrt{d_k}}V$$

Think of every word as a mini search engine user.

Query (Q)

"What am I looking for?"

Token "ate" searches for [food, edible objects].

Key (K)

"What defines me?"

Token "apple" tags itself as [fruit, food, red].

Value (V)

"My actual content"

The vector representing the concept of "apple".

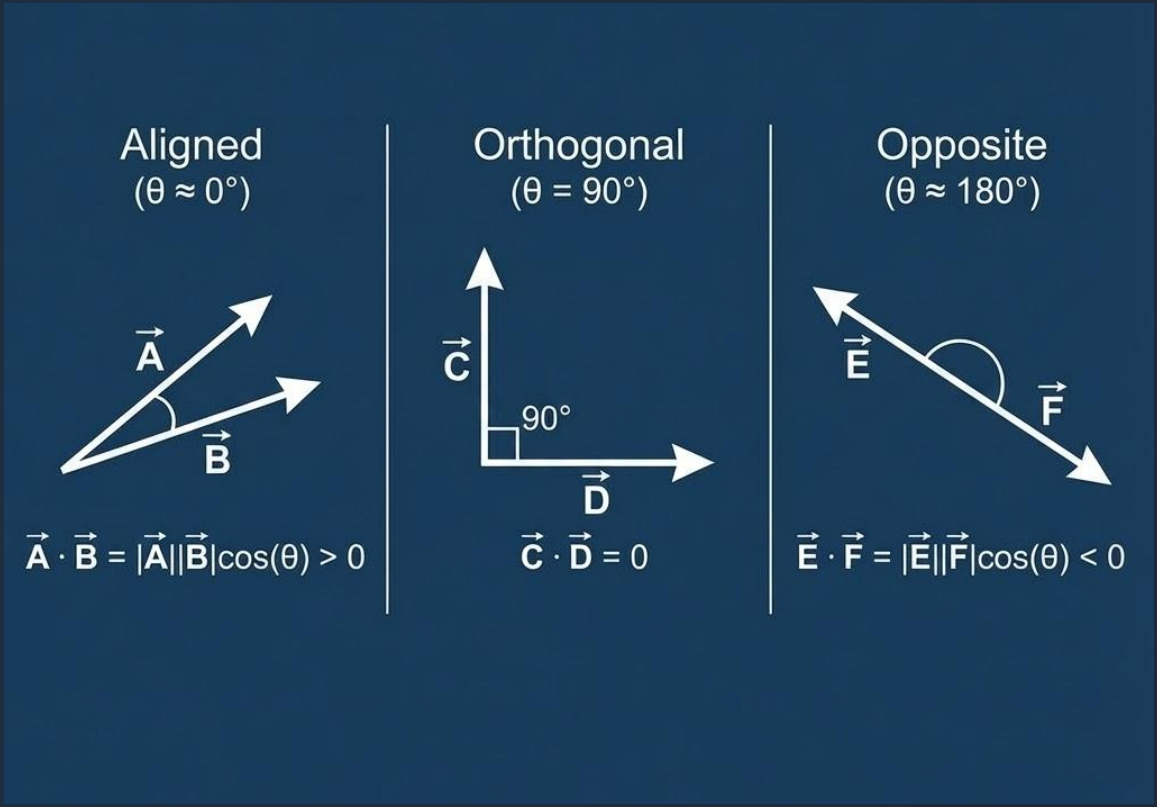
Key Takeaway: We calculate the match between Query and Key. If they match (High Score), we fetch the Value.

Why Dot Product? (The Math of "Matching")

Dot Product = Similarity

Why do we multiply?

- **Geometric Meaning:** The dot product measures how much two vectors point in the **same direction**.
- **Aligned:** (Strong Match)
- **Opposite:** (Strong Mismatch)
- **Orthogonal:** (Irrelevant)



Key Takeaway: The attention score is literally a measurement of "how parallel" the Query vector is to the Key vector.

Why 3 Matrices? (The Reasoning)

Asymmetric Roles

- **Problem:** If we just used the word vector itself, "I" would look for "I". Symmetry is bad here.
- **Solution:** We project the word into different roles.
- **Role 1 (Q):** "I am a subject looking for a verb."
- **Role 2 (K):** "I am a pronoun representing the speaker."
- **Role 3 (V):** "I am the concept of 'self' to be added to the sentence."

Projections = Intelligence

The matrices W_Q, W_K, W_V constitute the learned "intelligence" that decides *how* words relate to each other.

Key Takeaway: Projections decouple a word's "identity" from its "needs" (queries) and "offerings" (keys).

Scaling Deep Dive: Why 1/sqrt{d_k} ?

The "Exploding" Problem

- High-dimensional vectors (e.g., 1024 dim) produce massive dot products (e.g., +500).
- **Softmax Impact:** is astronomical. Softmax outputs become [0, 0, 1, 0].
- **Gradient Death:** When Softmax is saturated (outputs 0 or 1), the gradient is effectively zero. The model stops learning.

The Fix

$$1/\sqrt{d_k}$$

Dividing by \sqrt{d_k} keeps the variance stable, ensuring healthy gradients flow backward.

Limitation With Self-Attention

- **No Inherent Order:** it treats tokens like an unordered set.It doesn't know which word comes first or last.

- **Quadratic Cost ($O(N^2)$):** Comparing every word to every word is expensive. This limits context length.

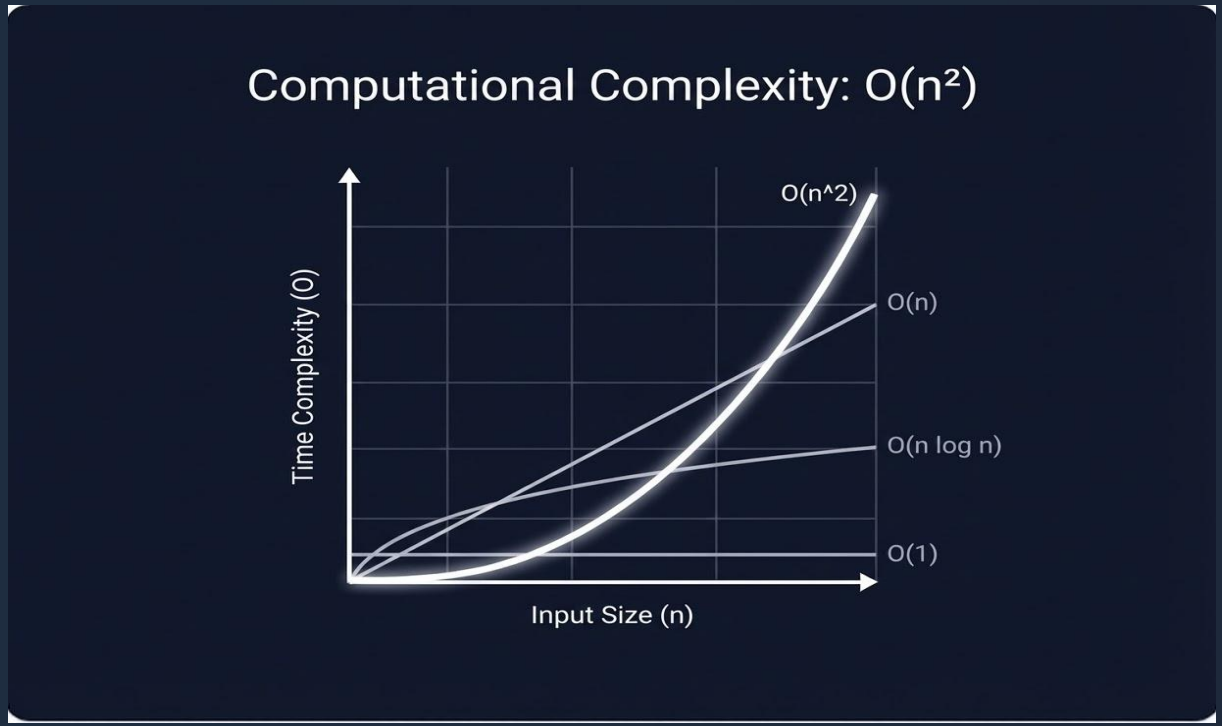
- **Attention Can Be Too Global:** Tokens can attend to everything, even irrelevant tokens which Causes noise, unnecessary mixing, and hallucination risk.

- **No Built-In Hierarchy:** Tokens are treated flatly. model doesn't inherently know phrases or syntax since Attention is uniform across tokens.

The Limitation: O(N²) Complexity

Why we can't have infinite context

- **1,000 words:** 1 million comparisons.
- **10,000 words:** 100 million comparisons.
- **100,000 words:** 10 billion comparisons.



Key Takeaway: The attention matrix size grows quadratically. Doubling the input length quadruples the memory needed.

Part 2: A Worked Example

"I love apple phones"

Step 1: Input Token Embeddings

We start with an embedding dimension d_model = 4. This is our input matrix **X**.

Token	Embedding Vector (X)			
I	1.0	0.5	0.2	0.1
love	0.9	1.1	0.1	0.0
apple	0.1	0.2	1.0	0.5
phones	0.0	0.1	0.4	1.2

$$X = \begin{matrix} 1.0 & 0.5 & 0.2 & 0.1 \\ 0.9 & 1.1 & 0.1 & 0.0 \\ 0.1 & 0.2 & 1.0 & 0.5 \\ 0.0 & 0.1 & 0.4 & 1.2 \end{matrix}$$

Step 2: Model Configuration

Model Dimension

$$d_{model} = 4$$

The size of the input and output vectors for the layer.

Heads

$$h = 2$$

We split the process into 2 parallel "heads" to capture different features.

Dimension per Head

$$d_k = 2$$

Calculated as dmodel /h

Step 3: Learned Projections (Head 1)

Each head has unique projection matrices W_q , W_k and W_v

Query Projection (W_q)

$$\begin{matrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{matrix}$$

Key Projection (W_k)

$$\begin{matrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{matrix}$$

Value Projection (W_v)

$$\begin{matrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{matrix}$$

Step 4: Computing Vectors

We multiply input **X** by the projections to get Q, K, and V vectors.

$$Q = X \cdot W_Q \quad | \quad K = X \cdot W_K \quad | \quad V = X \cdot W_V$$

Token	Query (Q)	Key (K)	Value (V)
I	[1.2, 0.6]	[1.2, 0.6]	[1.2, 0.6]
love	[1.0, 1.1]	[0.9, 1.1]	[0.9, 1.1]
apple	[1.1, 0.7]	[0.7, 0.7]	[1.1, 1.2]
phones	[0.4, 1.3]	[1.3, 0.3]	[1.3, 1.3]

Step 5: Raw Attention Scores

The Formula

We calculate how much each word focuses on every other word using the dot product.

$$\text{Score} = Q \cdot K^T$$

High scores indicate a stronger relationship or relevance between the two tokens.

Raw Scores Matrix

	I	love	apple	phones
I	1.80	1.74	1.26	1.74
love	1.86	2.11	1.47	1.96
apple	1.74	1.76	1.26	1.64
phones	1.26	1.79	1.19	1.91

Step 6 & 7: Scaling and Softmax

1. Divide by $d_k = \sqrt{2} \approx 1.414$ to stabilize gradients.

2. Apply Softmax to convert scores into probabilities (sum to 1).

Attention Weights Matrix

	I	love	apple	phones	Sum
I	0.278	0.266	0.190	0.266	1.0
love	0.248	0.296	0.189	0.267	1.0
apple	0.273	0.277	0.195	0.255	1.0
phones	0.200	0.292	0.191	0.317	1.0

Example: "I" pays 27.8% attention to itself and 19.0% to "apple".

Step 8: Weighted Sum (Head 1 Output)

Calculation for "I"

$$O_I = \sum_j \alpha_{Ij} \cdot V_j$$

0.28(V_I) + 0.27(V_love) + 0.19(V_apple) + 0.27(V_phones)

Result: [1.128, 1.034]

Full Head 1 Output

Token	Output Vector (O1)
I	[1.128, 1.034]
love	[1.119, 1.049]
apple	[1.124, 1.035]
phones	[1.125, 1.082]

Step 9 & 10: Concatenation

Head 2 runs in parallel with its own matrices. We then concatenate the outputs from both heads.

Head 1 (Calculated)
[1.13, 1.03]

+

Head 2 (Simulated)
[0.99, 1.03]

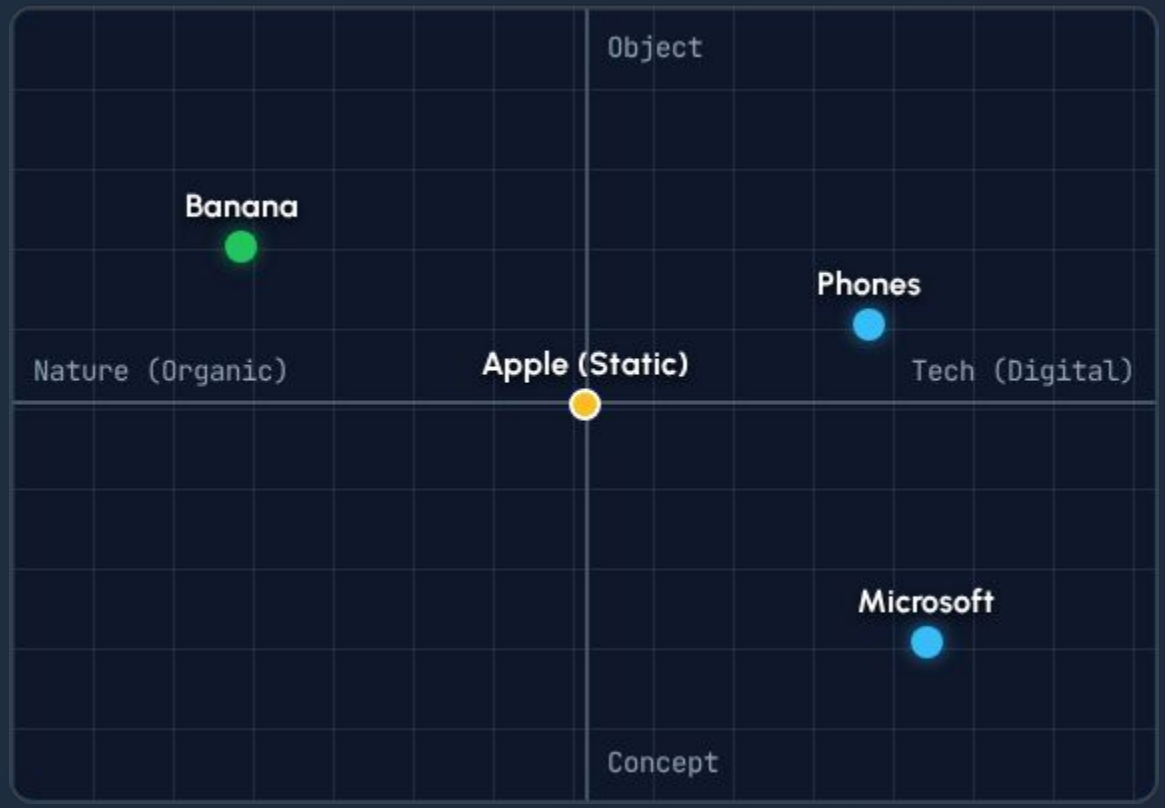
Final Multi-Head Output

Token	Concatenated Vector (dim=4)
I	[1.128, 1.034, 0.99, 1.03]
love	[1.119, 1.049, 1.01, 1.04]
apple	[1.124, 1.035, 1.02, 1.02]
phones	[1.125, 1.082, 1.02, 1.02]

Visualizing the Meaning: Static Embeddings

Before attention, the vector for "**Apple**" is static. It contains all potential meanings mixed together.

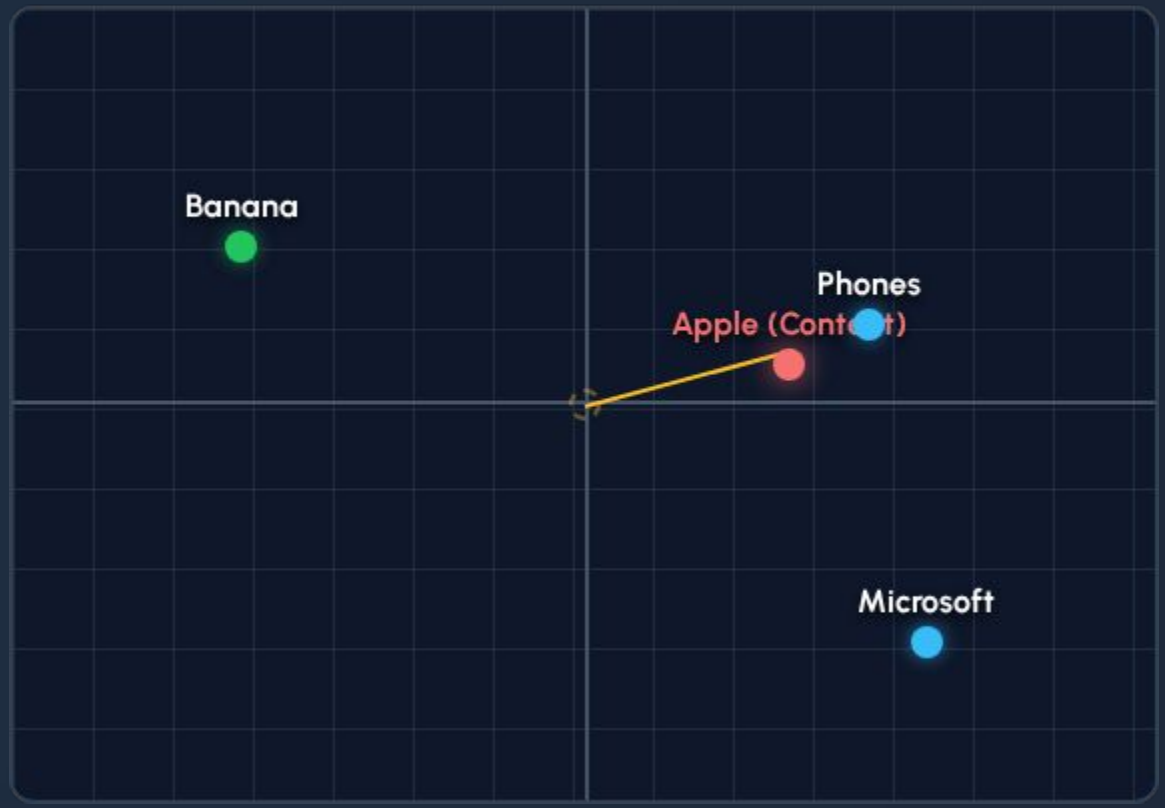
- It sits in a neutral, ambiguous space.
- It is equidistant from **Fruits** (Nature) and **Tech** (Digital).
- The model doesn't know which meaning is correct yet.



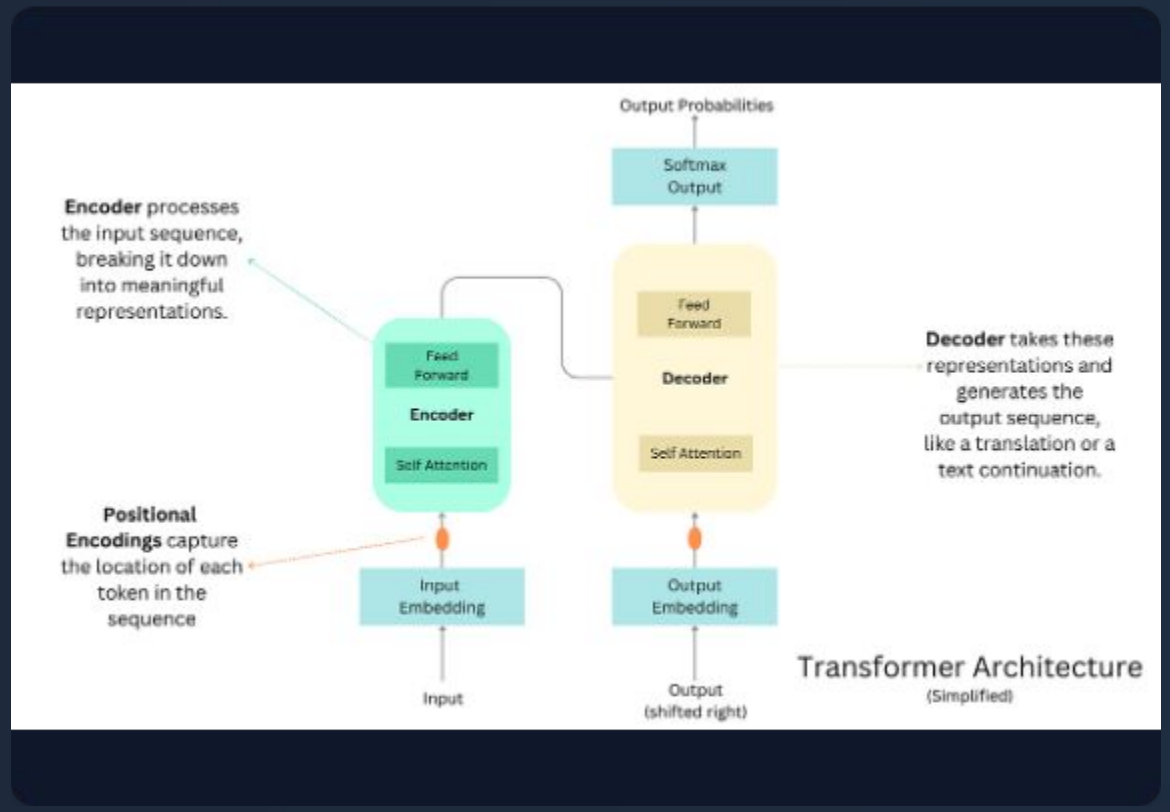
Visualizing the Meaning: The Shift

After **Self-Attention**, the vector updates based on context.

- Because "Apple" attended strongly to **"Phones"**, it pulls information from the Tech cluster.
- The vector physically **shifts** in the high-dimensional space.
- It now sits firmly in the **Tech** semantic region.



Key Takeaways



- **Projections:** Input is projected into Query, Key, and Value spaces.
- **Scores:** Attention scores determine relevance between tokens ($Q \cdot K$).
- **Softmax:** Converts scores to probabilities.
- **Weighted Sum:** Output is a mix of Value vectors based on attention weights.
- **Multi-Head:** Allows the model to capture multiple types of relationships at once.