

## **The Need for Positional Embeddings in Transformers**

To understand the importance of RoPE, let's first revisit why positional embeddings are crucial. Transformer models, by their inherent design, do not consider the order of input tokens.

For instance, phrases like “the **dog** chases the **pig**” and “the pig chases the dog,” though distinct in meaning, are treated indistinguishably as they are perceived as an unordered set of tokens. To maintain the sequence information and thus the meaning, positional embeddings are integrated into the model.

## **How Absolute Positional Embeddings Work**

In the context of a sentence, suppose we have an embedding representing a word. To encode its position, we use another vector of identical dimensionality, where each vector uniquely represents a position in the sentence. For instance, a specific vector is designated for the second word in a sentence. Thus, each sentence position gets its distinct vector. The input for the Transformer layer is then formed by summing the word embedding with its corresponding positional embedding.

## **There are primarily two methods to generate these embeddings:**

1. **Learning from Data:** Here, positional vectors are learned during training, just like other model parameters. We learn a unique vector for each position, say from 1 to 512. However, this introduces a limitation — the maximum sequence length is capped. If the model only learns up to position 512, it cannot represent sequences longer than that.
2. **Sinusoidal Functions:** This method involves constructing unique embeddings for each position using a sinusoidal function. Although the intricate details of this construction are complex, it essentially provides a unique positional embedding for every position in a sequence. Empirical studies have shown that learning from data and using sinusoidal functions offer comparable performance in real-world models.

## **Limitations of Absolute Positional Embeddings**

Despite their widespread use, absolute positional embeddings are not without drawbacks:

1. **Limited Sequence Length:** As mentioned, if a model learns positional vectors up to a certain point, it cannot inherently represent positions beyond that limit.



**2. Independence of Positional Embeddings:** Each positional embedding is independent of others. This means that in the model's view, the difference between positions 1 and 2 is the same as between positions 2 and 500. However, intuitively, positions 1 and 2 should be more closely related than position 500, which is significantly farther away. This lack of relative positioning can hinder the model's ability to understand the nuances of language structure.

## **Understanding Relative Positional Embeddings**

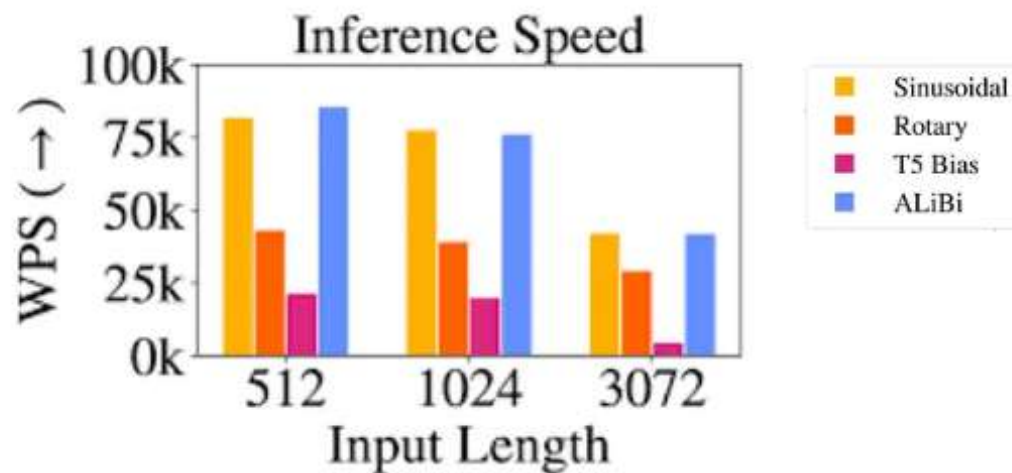
Rather than focusing on a token's absolute position in a sentence, relative positional embeddings concentrate on the distances between pairs of tokens. This method doesn't add a position vector to the word vector directly. Instead, it alters the attention mechanism to incorporate relative positional information.

### **Case Study: The T5 Model**

One prominent model that utilizes relative positional embeddings is T5 (Text-to-Text Transfer Transformer). T5 introduces a nuanced way of handling positional information:

- **Bias for Positional Offsets:** T5 uses a bias, a floating-point number, to represent each possible positional offset. For example, a bias B1 might represent the relative distance between any two tokens that are one position apart, regardless of their absolute positions in the sentence.
- **Integration in Self-Attention Layer:** This matrix of relative position biases is added to the product of the query and key matrices in the self-attention layer. This ensures that tokens at the same relative distance are always represented by the same bias, regardless of their position in the sequence.
- **Scalability:** A significant advantage of this method is its scalability. It can extend to arbitrarily long sequences, a clear benefit over absolute positional embeddings.

### Challenges with Relative Positional Embeddings



Despite their theoretical appeal, relative positional embeddings pose certain practical challenge.

1. **Performance Issues:** Benchmarks comparing T5's relative embeddings with other types have shown that they can be slower, particularly for longer sequences. This is primarily due to the additional computational step in the self-attention layer, where the positional matrix is added to the query-key matrix.
2. **Complexity in Key-Value Cache Usage:** As each additional token alters the embedding for every other token, this complicates the effective use of key-value caches in Transformers. For those unfamiliar, key-value caches are crucial in enhancing efficiency and speed in Transformer models.

Due to these engineering complexities, relative embeddings haven't been widely adopted, especially in larger language models.

# What Are Rotary Positional Embeddings (RoPE)?

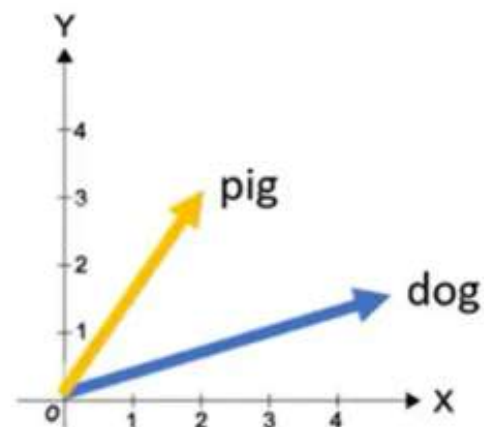
RoPE represents a novel approach in encoding positional information. Traditional methods, either absolute or relative, come with their limitations. Absolute positional embeddings assign a unique vector to each position, which though straightforward, doesn't scale well and fails to capture relative positions effectively. Relative embeddings, on the other hand, focus on the distance between tokens, enhancing the model's understanding of token relationships but complicating the model architecture.

RoPE ingeniously combines the strengths of both. It encodes positional information in a way that allows the model to understand both the absolute position of tokens and their relative distances. This is achieved through a rotational mechanism, where each position in the sequence is represented by a rotation in the embedding space. The elegance of RoPE lies in its simplicity and efficiency, enabling models to better grasp the nuances of language syntax and semantics.



# The Mechanism of Rotary Positional Embeddings

The pig chased the dog



RoPE introduces a novel concept. Instead of adding a positional vector, it applies a rotation to the word vector. Imagine a two-dimensional word vector for “dog.” To encode its position in a sentence, RoPE rotates this vector. The angle of rotation ( $\theta$ ) is proportional to the word’s position in the sentence. For instance, the vector is rotated by  $\theta$  for the first position,  $2\theta$  for the second, and so on. This approach has several benefits:

Top highlight

1. **Stability of Vectors:** Adding tokens at the end of a sentence doesn’t affect the vectors for words at the beginning, facilitating efficient caching.

2. **Preservation of Relative Positions:** If two words, say “pig” and “dog,” maintain the same relative distance in different contexts, their vectors are rotated by the same amount. This ensures that the angle, and consequently the dot product between these vectors, remains constant

### Matrix Formulation of RoPE

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{\{q,k\}}^{(11)} & W_{\{q,k\}}^{(12)} \\ W_{\{q,k\}}^{(21)} & W_{\{q,k\}}^{(22)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}$$

The technical implementation of RoPE involves rotation matrices. In a 2D case, the equation from the paper incorporates a rotation matrix that rotates a vector by an angle of  $M\theta$ , where  $M$  is the absolute position in the sentence. This rotation is applied to the query and key vectors in the self-attention mechanism of the Transformer.

For higher dimensions, the vector is split into 2D chunks, and each pair is rotated independently. This can be visualized as an  $n$ -dimensional corkscrew rotating in space.



the RoFormer paper. Let your head dimension be  $d$  (even), and index the 2D subspaces by  $i = 0, 1, \dots, \frac{d}{2} - 1$ . For a sequence position  $p$  define

$$\omega_i = \frac{1}{10000^{i/d}} \in \text{inv\_freq}[i], \quad \theta_{p,i} = p \omega_i.$$

Split each  $d$ -dimensional vector  $x_p$  into two halves of length  $d/2$ :


$$x_p = [x_{p,0}, x_{p,1}, \dots, x_{p,d-2}, x_{p,d-1}] \longrightarrow (x_{p,2i}, x_{p,2i+1}) \quad (i = 0, \dots, \frac{d}{2} - 1).$$

Then the rotated output  $x'_p$  is given component-wise by applying a  $2 \times 2$  rotation in each subspace  $(2i, 2i + 1)$ :

$$\begin{aligned} x'_{p,2i} &= x_{p,2i} \cos \theta_{p,i} - x_{p,2i+1} \sin \theta_{p,i}, \\ x'_{p,2i+1} &= x_{p,2i} \sin \theta_{p,i} + x_{p,2i+1} \cos \theta_{p,i}. \end{aligned}$$

Equivalently, in matrix form for each  $i$ :

$$\begin{bmatrix} x'_{p,2i} \\ x'_{p,2i+1} \end{bmatrix} = \begin{bmatrix} \cos \theta_{p,i} & -\sin \theta_{p,i} \\ \sin \theta_{p,i} & \cos \theta_{p,i} \end{bmatrix} \begin{bmatrix} x_{p,2i} \\ x_{p,2i+1} \end{bmatrix}.$$



Let's walk through **one concrete numeric example** for a single token (say "cat") at position  $p = 3$  in the sequence, with a tiny head dimension  $d = 4$ . We'll show every step:

---

## 1. Setup

- Head dimension  $d = 4$ , so we have two 2-D subspaces:  $i = 0$  and  $i = 1$ .
- Theta base = 10 000.
- Position  $p = 3$ .

Assume the *raw* (pre-RoPE) vector for this head & token is

$$x_p = [x_{p,0}, x_{p,1}, x_{p,2}, x_{p,3}] = [0.5, -1.0, 1.5, 2.0].$$

---

## 2. Compute $\omega_i$ ("inv\_freq")

$$\omega_i = \frac{1}{10000^{i/d}} \quad (i = 0, 1).$$

1.  $i = 0 : \omega_0 = 1/10000^{0/4} = 1/1 = 1.0.$
2.  $i = 1 : \omega_1 = 1/10000^{1/4} = 1/10^1 = 0.1.$

So

$$\omega = [ 1.0, 0.1 ].$$

---

**3. Build angles  $\theta_{p,i} = p \omega_i$**

With position  $p = 3$ :

$$\theta_{3,0} = 3 \times 1.0 = 3.0, \quad \theta_{3,1} = 3 \times 0.1 = 0.3.$$

---



#### 4. Compute cos and sin

$i$	$\theta_{3,i}$	$\cos(\theta)$	$\sin(\theta)$
0	3.0	$\cos(3.0) \approx -0.99$	$\sin(3.0) \approx 0.14$
1	0.3	$\cos(0.3) \approx 0.96$	$\sin(0.3) \approx 0.30$

#### 5. Split $x_p$ into two halves

$$x_1 = [x_{p,0}, x_{p,1}] = [0.5, -1.0], \quad x_2 = [x_{p,2}, x_{p,3}] = [1.5, 2.0].$$

## 6. Rotate each 2-D pair

For **subspace**  $i = 0$  (coordinates 0 & 1):

$$\begin{pmatrix} x'_{p,0} \\ x'_{p,1} \end{pmatrix} = \begin{bmatrix} \cos(3.0) & -\sin(3.0) \\ \sin(3.0) & \cos(3.0) \end{bmatrix} \begin{pmatrix} 0.5 \\ -1.0 \end{pmatrix} = \begin{pmatrix} (-0.99) \cdot 0.5 - (0.14) \cdot (-1.0) \\ (0.14) \cdot 0.5 + (-0.99) \cdot (-1.0) \end{pmatrix} \approx \begin{pmatrix} -0.495 + 0.14 \\ 0.07 + 0.99 \end{pmatrix} = \begin{pmatrix} -0.355 \\ 1.06 \end{pmatrix}.$$

For **subspace**  $i = 1$  (coordinates 2 & 3):

$$\begin{pmatrix} x'_{p,2} \\ x'_{p,3} \end{pmatrix} = \begin{bmatrix} \cos(0.3) & -\sin(0.3) \\ \sin(0.3) & \cos(0.3) \end{bmatrix} \begin{pmatrix} 1.5 \\ 2.0 \end{pmatrix} = \begin{pmatrix} 0.96 \cdot 1.5 - 0.30 \cdot 2.0 \\ 0.30 \cdot 1.5 + 0.96 \cdot 2.0 \end{pmatrix} \approx \begin{pmatrix} 1.44 - 0.60 \\ 0.45 + 1.92 \end{pmatrix} = \begin{pmatrix} 0.84 \\ 2.37 \end{pmatrix}.$$

---

## 7. Final RoPE'd vector

Concatenate the rotated halves:

$$x'_p = [x'_{p,0}, x'_{p,1}, x'_{p,2}, x'_{p,3}] \approx [-0.355, 1.06, 0.84, 2.37].$$

## In summary

- **Step 1:** Compute two “inverse frequencies”  $\omega_0, \omega_1$ .
- **Step 2:** Multiply by position  $p$  to get angles  $\theta_{p,i}$ .
- **Step 3:** Split the 4-D vector into two 2-D subvectors.
- **Step 4:** Rotate each subvector by its  $\theta_{p,i}$ .
- **Step 5:** Re-assemble into a new 4-D vector with positional information baked in.