Member-only story

Revisiting The Basics: Rotary Position Embeddings (RoPE)

A lesson on Positional Embeddings all the way up to Rotary Position Embeddings (RoPE) from the ground up.

Dr. Ashish Bamania · Following

Published in AI Advances · 7 min read · 5 hours ago

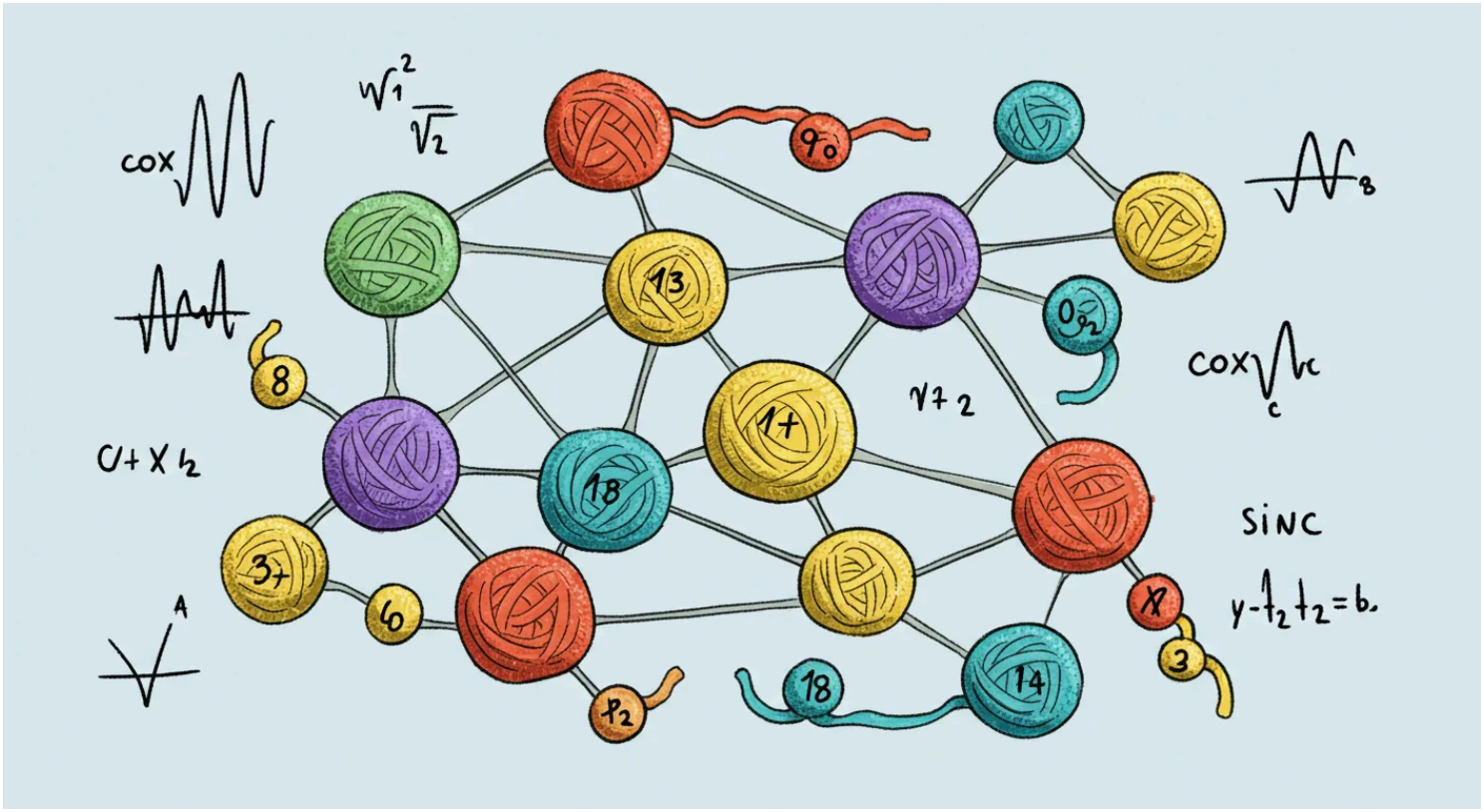


Image generated with Google Imagen-3

Transformers process tokens in parallel rather than sequentially.

This is what gives them the computational advantage over RNNs.

However, this also makes Transformers **position-agnostic**, meaning they do not have a sense of the order of the tokens they process.

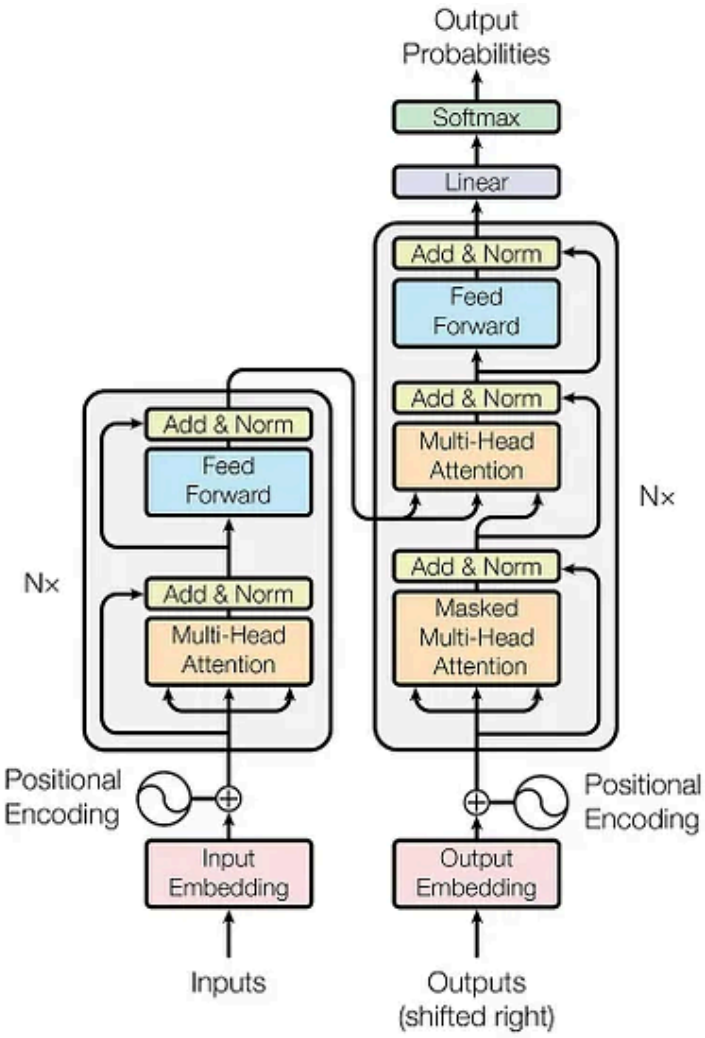
Consider these two sentences:

> 1. *"The cat sits on the mat."*
>
> 2. *"The mat sites on the cat."*

To a Transfromer, both of them are the same.

This isn't good for language processing.

Therefore, positional information in the form of positional embeddings (vectors) is added with token embeddings before Transformers process them.



The Transfromer architecture where the addition of Positional Encoding to the Input embedding can be seen in the lower left and right corners (Image obtained from the research paper titled 'Attention Is All You Need')

Types Of Positional Embeddings

Positional Embeddings are of two main types:

1. **Absolute Positional Embeddings** — where each token is assigned a unique encoding according to its position
2. **Relative Positional Embeddings** — where information about how far apart tokens are from each other is encoded rather than their absolute positions

Each of them can be either:

- **Fixed** (calculated using a mathematical function)
- **Learned** (has trainable parameters that are updated with backpropagation during model training)

Positional Embeddings In The Original Transformer Paper

While relative positional embeddings are popularly used by the T5 Transformer, in the original Transformers paper, the authors use fixed absolute positional embeddings, as shown below.

(Learned positional embeddings are not used in this paper because the fixed embeddings produced nearly identical results.)

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Calculations for Positional Embeddings where 'pos' is the token index, 'i' is the index of the token embedding dimension, and 'd' is the total token embedding dimension

respectively.

The denominator `10000 ^ (2i/ d(model))` controls the wavelength of these functions.

This means that for lower dimensions (smaller `i`), the frequency is high, and the wavelengths are short.

Similarly, for the higher dimensions (larger `i`), the frequency is low, and the wavelengths are long.

These wavelengths form a Geometric progression, with the smallest and the largest values being `2π` and `10000 × 2π`, respectively.

This makes the model **capture the short-term and long-term dependencies in the input sequence using the high and low frequencies, respectively.**

Since the dimensions of these positional embeddings are the same as those of the token embeddings, they can be directly summed as follows before being passed to the Transformer for processing.

$$X' = X + PE$$

X' is the final input embedding passed into the Transformer, X is the original input embedding, and PE is the set of positional embeddings for each token in the input sequence.

These embeddings can also capture the relative relationships between tokens, as their positional embeddings are related linearly (but are not the

best for this).

Improving Learning Further With RoPE

Previous embedding approaches can struggle to capture dependencies in sequences longer than those seen during training.

These approaches also **add** positional embeddings to the token embeddings, which increases the total parameters and complexity. This leads to increased computational training costs and slower inference.

A 2023 research introduced a new way of directly encoding both absolute and relative positions in the attention mechanism.

Their method is called **Rotary Position Embedding** or **RoPE**.

Instead of adding position embeddings, as in the previous approaches, RoPE **rotates** the token embeddings according to their positions.

For a token embedding `x(m)` of dimension `d` at position `m`, it is transformed into Query (`q(m)`) and Key (`k(n)`) vectors as follows using weight matrices `W(q)` and `W(k)`, respectively.

$$q_m = W_q x_m, \quad k_n = W_k x_n$$

RoPE rotates these vectors before they are used in the attention calculation. This is done using a position-dependent rotation matrix `R(m)`.

$$q'_m = R_m q_m, \quad k'_m = R_m k_m$$

`R(m)` acts independently on each **pair** of dimensions in `q` and `k`.

For a case where these vectors are two-dimensional (`d = 2`), the rotation matrix `R(m)` is defined as:

$$R_m = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix}$$

This 2D rotation matrix rotates these vectors **counter-clockwise**, proportional to their position `m` by an angle of `mθ` where `θ` is a constant (`1` in the case of `d = 2`).

For a two-dimension query vector `q(m) = (q1, q2)`, this transformation will result in:

$$\begin{bmatrix} q'_1 \\ q'_2 \end{bmatrix} = \begin{bmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

$$q'_1 = q_1 \cos(m\theta) - q_2 \sin(m\theta)$$
$$q'_2 = q_1 \sin(m\theta) + q_2 \cos(m\theta)$$

However, query and key vectors are usually higher-dimensional (assumed an even number of dimensions).

To work with them, they are paired, and separate 2D rotations are applied to each adjacent pair of dimensions.

For example, for a 4-dimension query vector `q(m) = (q1, q2, q3, q4)`, two independent rotations are applied for `(q1, q2)` and `(q3, q4)` as shown below.

$$(q_1', q_2') = R_m^{\theta_1}(q_1, q_2), \quad (q_3', q_4') = R_m^{\theta_2}(q_3, q_4)$$

These can be expanded to:

$$\begin{bmatrix} q_1' \\ q_2' \end{bmatrix} = \begin{bmatrix} \cos(m\theta_1) & -\sin(m\theta_1) \\ \sin(m\theta_1) & \cos(m\theta_1) \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

$$\begin{bmatrix} q_3' \\ q_4' \end{bmatrix} = \begin{bmatrix} \cos(m\theta_2) & -\sin(m\theta_2) \\ \sin(m\theta_2) & \cos(m\theta_2) \end{bmatrix} \begin{bmatrix} q_3 \\ q_4 \end{bmatrix}$$

$$\begin{aligned} q_1' &= q_1 \cos(m\theta_1) - q_2 \sin(m\theta_1) \\ q_2' &= q_1 \sin(m\theta_1) + q_2 \cos(m\theta_1) \\ q_3' &= q_3 \cos(m\theta_2) - q_4 \sin(m\theta_2) \\ q_4' &= q_3 \sin(m\theta_2) + q_4 \cos(m\theta_2) \end{aligned}$$

The angle of rotation `θ(i)` for each pair of dimensions is different and is calculated as follows:

$$\theta_i = 10000^{-2(i-1)/d}, \quad i = 1, 2, \ldots, d/2$$

This ensures that the **low-frequency rotations for higher dimensions capture long-range dependencies and the high-frequency rotations for lower dimensions capture short-term dependencies.**

This `θ` is similar to what we learned when describing the sinusoidal embeddings from the original Transformers paper.

The overall rotation matrix for higher dimensions looks as follows.

$$R_m = \begin{bmatrix} \cos(m\theta_1) & -\sin(m\theta_1) & 0 & 0 & \ldots \\ \sin(m\theta_1) & \cos(m\theta_1) & 0 & 0 & \ldots \\ 0 & 0 & \cos(m\theta_2) & -\sin(m\theta_2) & \ldots \\ 0 & 0 & \sin(m\theta_2) & \cos(m\theta_2) & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

Making Calculations Computationally Efficient

One can avoid explicit matrix multiplication to make the process of computing the rotated queries and keys computationally efficient.

This can be done by precomputing `cos(mθ(i))` and `sin(mθ(i))` for each position `m`, and then performing simple element-wise multiplications and additions between the terms.

For example, this is how we calculated the rotated query `q'(m)` previously:

$$q'_m = R_m q_m$$

$$\begin{bmatrix} q'_1 \\ q'_2 \\ q'_3 \\ q'_4 \\ \vdots \\ q'_{d-1} \\ q'_d \end{bmatrix} = \begin{bmatrix} \cos(m\theta_1)q_1 - \sin(m\theta_1)q_2 \\ \sin(m\theta_1)q_1 + \cos(m\theta_1)q_2 \\ \cos(m\theta_2)q_3 - \sin(m\theta_2)q_4 \\ \sin(m\theta_2)q_3 + \cos(m\theta_2)q_4 \\ \vdots \\ \cos(m\theta_{d/2})q_{d-1} - \sin(m\theta_{d/2})q_d \\ \sin(m\theta_{d/2})q_{d-1} + \cos(m\theta_{d/2})q_d \end{bmatrix}$$

This can be changed to:

$$R_m q_m = q_m \circ \cos(m\theta) + S(q_m) \circ \sin(m\theta)$$

The computationally efficient calculation for R(m)q(m) where ∘ represents element-wise multiplication and S(q(m)) operation swaps adjacent elements in q(m), negating the even-indexed ones

The complete calculation is shown below:

$$R_m q_m = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ \vdots \\ q_{d-1} \\ q_d \end{pmatrix} \circ \begin{pmatrix} \cos(m\theta_1) \\ \cos(m\theta_1) \\ \cos(m\theta_2) \\ \cos(m\theta_2) \\ \vdots \\ \cos(m\theta_{d/2}) \\ \cos(m\theta_{d/2}) \end{pmatrix} + \begin{pmatrix} -q_2 \\ q_1 \\ -q_4 \\ q_3 \\ \vdots \\ -q_d \\ q_{d-1} \end{pmatrix} \circ \begin{pmatrix} \sin(m\theta_1) \\ \sin(m\theta_1) \\ \sin(m\theta_2) \\ \sin(m\theta_2) \\ \vdots \\ \sin(m\theta_{d/2}) \\ \sin(m\theta_{d/2}) \end{pmatrix}$$

This reduces the time complexity from $O(d^2)$ to $O(d)$ by avoiding explicit matrix multiplication.

Calculating Attention Score

Next, the attention score is calculated as the dot product between the rotated queries and keys as follows:

$$q_m'^{\top} k_n'$$

This can be expanded to:

$$q_m'^{T} k_n' = (R_m q_m)^T (R_n k_n)$$

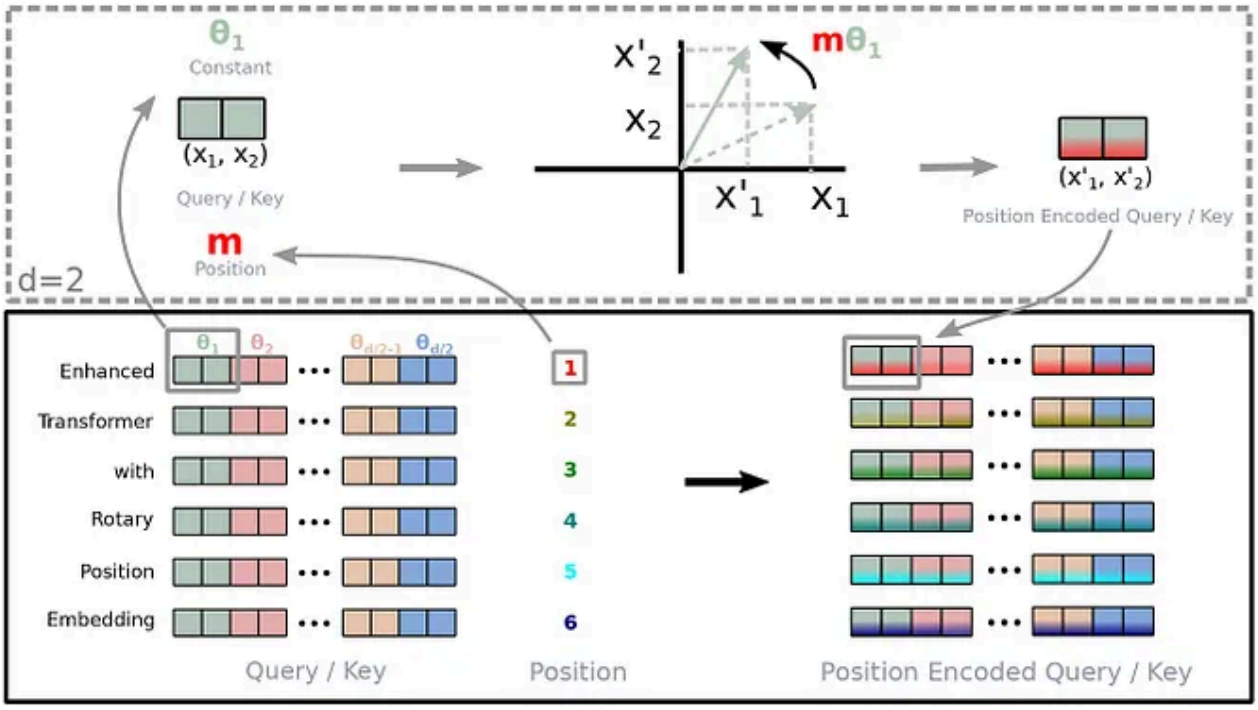
Since the rotation matrices can be written as follows,

$$R_m^{\top} R_n = R_{n-m}$$

The equation becomes:

$$q_m'^{\top} k_n' = q_m^{\top} R_{n-m} k_n$$

Thanks to RoPE, the **attention score now encodes the relative position of tokens** `(n - m)` **without needing additional relative embeddings.**

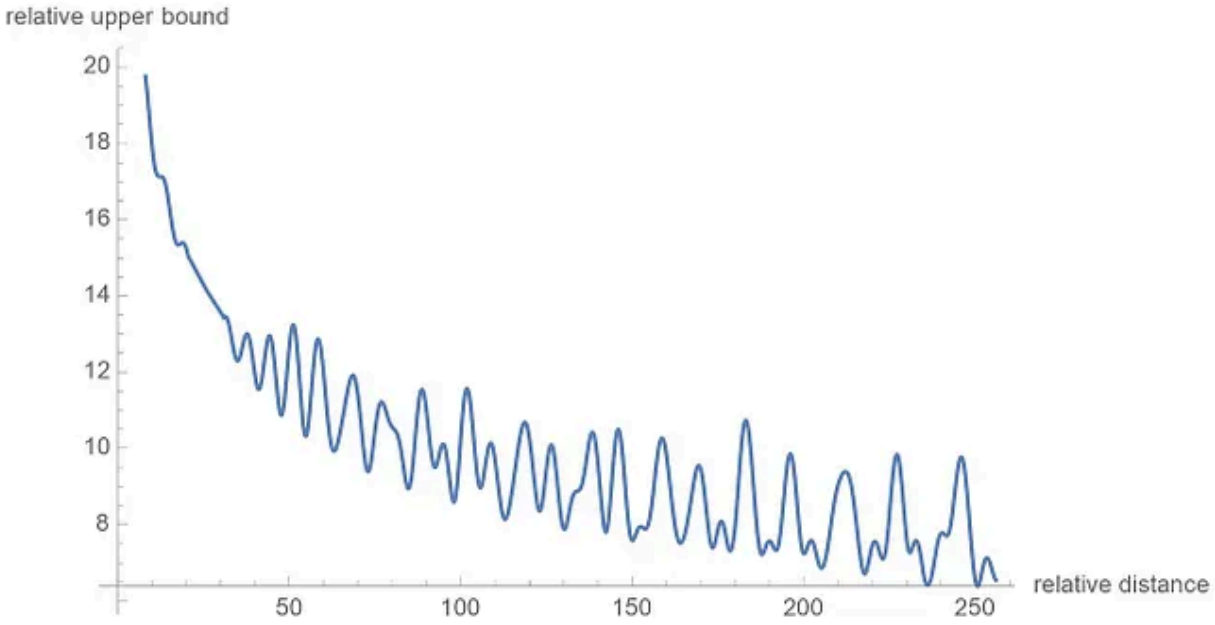


Visual representation of RoPE embeddings (Image from the ArXiv research paper titled 'RoFormer: Enhanced Transformer with Rotary Position Embedding')

RoPE Decay For Long Inter-Token Dependencies

There's another cool property of these embeddings: **the relative importance of the connection between far-apart tokens is lower than closer ones.**

The RoPE research paper provides a mathematical explanation for this, and curious readers are encouraged to read it.



Graph showing the decay of attention scores in RoPE as the relative distance between tokens increases. The further two tokens are apart, the weaker their maximum possible attention score becomes. (Image from the ArXiv research paper titled 'RoFormer: Enhanced Transformer with Rotary Position Embedding')

(Although the original research paper supports this notion, some new research has shown that this might not always be true, and the decay in long-term dependencies is not a universal property of RoPE.)

Which LLMs Use RoPE Today?

RoPE is one reason for the phenomenal performance achieved by modern-day LLMs, such as Meta's Llama 3 models and Google's Gemma models.

DeepSeek-V3 also uses a modified version of RoPE that is compatible with its modified version of Multi-head Attention called Multi-head Latent Attention.

Further Reading

- Research paper titled “**RoFormer: Enhanced Transformer with Rotary Position Embedding**” published in ArXiv
- Research paper titled “**Round and Round We Go! What makes Rotary Positional Encodings useful?**” published in ArXiv
- RoFormer, a BERT-like autoencoding model with rotary position embeddings, on HuggingFace

Subscribe to ‘Into AI’ — my weekly newsletter where I help you explore Artificial Intelligence from the ground up by dissecting the original research papers.

Artificial Intelligence | Data Science | Machine Learning | Technology | Programming

Written by Dr. Ashish Bamania

32K Followers · 414 Following

Following

🍰 I simplify the latest advances in AI, Quantum Computing & Software Engineering for you | 📰 Subscribe to my newsletter here: https://intoai.pub

No responses yet

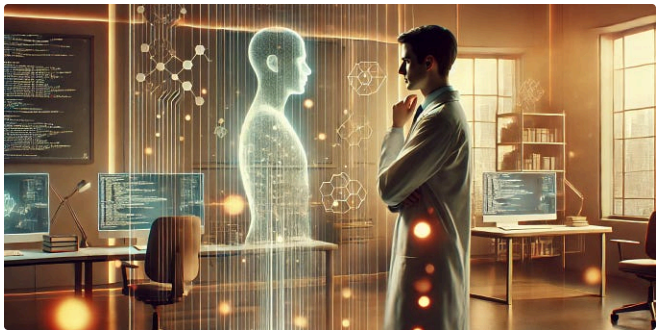
Alex Mylnikov

What are your thoughts?

More from Dr. Ashish Bamania and AI Advances

In Level Up Coding by Dr. Ashish Bamania

Chain-of-Draft (CoD) Is The New King Of Prompting Techniques



In Level Up Coding by Dr. Ashish Bamania

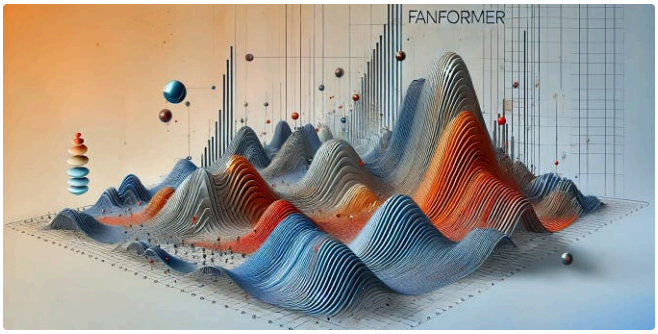
The Open Source “Agentic Reasoning” Beats Google Gemini...

A deep dive into the novel Chain-of-Draft (CoD) Prompting that reducing LLM inferenc...

Mar 3 809 11

A deep dive into the “Agentic Reasoning” framework & the techniques behind it that...

Feb 19 749 15



In Level Up Coding by Dr. Ashish Bamania

‘FANformer’ Is The New Game-Changing Architecture For LLMs

A deep dive into how FANFormer architecture works and what makes it so powerful...

Mar 9 253 7

In Level Up Coding by Dr. Ashish Bamania

DeepSeek-R1 Beats OpenAI’s o1, Revealing All Its Training Secrets...

A deep dive into how DeepSeek-R1 was trained from scratch and how this open-...

Jan 26 1.3K 31

See all from Dr. Ashish Bamania

See all from AI Advances

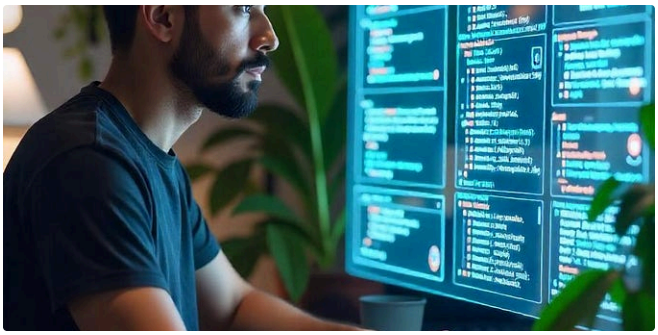
Recommended from Medium

In Towards AI by DarkBones

You're Doing RAG Wrong: How to Fix Retrieval-Augmented...

How To Set Up RAG Locally, Avoid Common Issues, and Improve RAG Retrieval Accuracy.

Mar 8 283 7



Thack

Claude 3.7 Sonnet: the first AI model that understands your enti...

Context is king. Emperor Claude is here. In this exhaustive guide to our newest frontier...

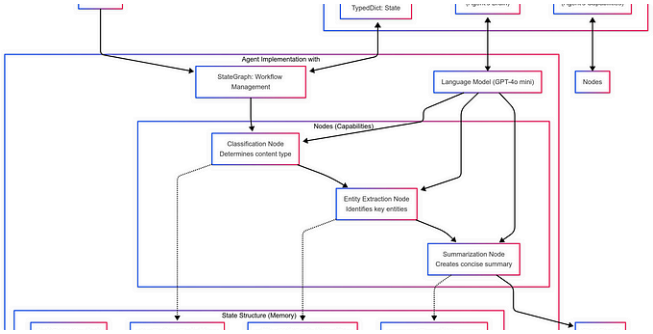
Feb 25 880 29

In Generative AI by Cezary Gesikowski

Microsoft's AI Midlife Crisis—Why Satya Nadella's Caution Should...

The company that missed the internet, fumbled mobile, and played catch-up on the...

Feb 28 613 37

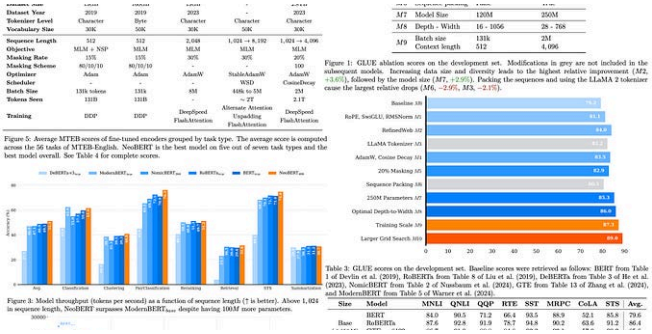


In Data Science Collective by Paolo Perrone

The Complete Guide to Building Your First AI Agent (It's Easier Th...

Three months into building my first commercial AI agent, everything collapsed...

5d ago 1.2K 30



Andrew Lukyanenko

Paper Review: NeoBERT: A Next-Generation BERT

NeoBERT: a next-gen encoder with 4K tokens context length that outperforms larger...

Mar 3 30

In Cubed by Djohra IBERRAKEN

NeurIPS 2024 Top Papers: Comprehensive Overview

© Logo of NeurIPS

Feb 2 146

See more recommendations