IMPERIAL

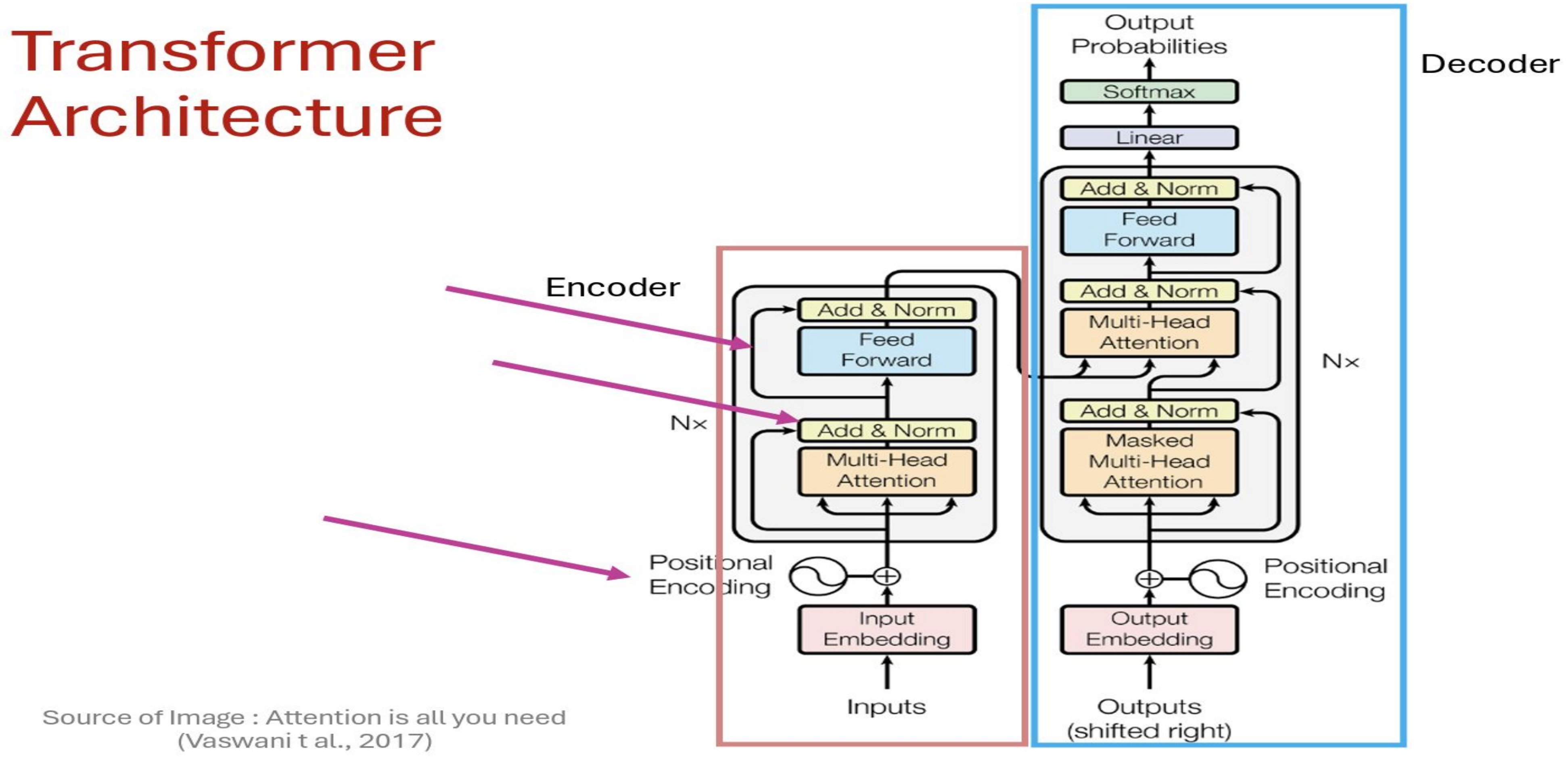
Transformer (2)

08/12/2025

Shamsuddeen Muhammad
Google DeepMind Academic Fellow,
Imperial College London
https://shmuhammadd.github.io/

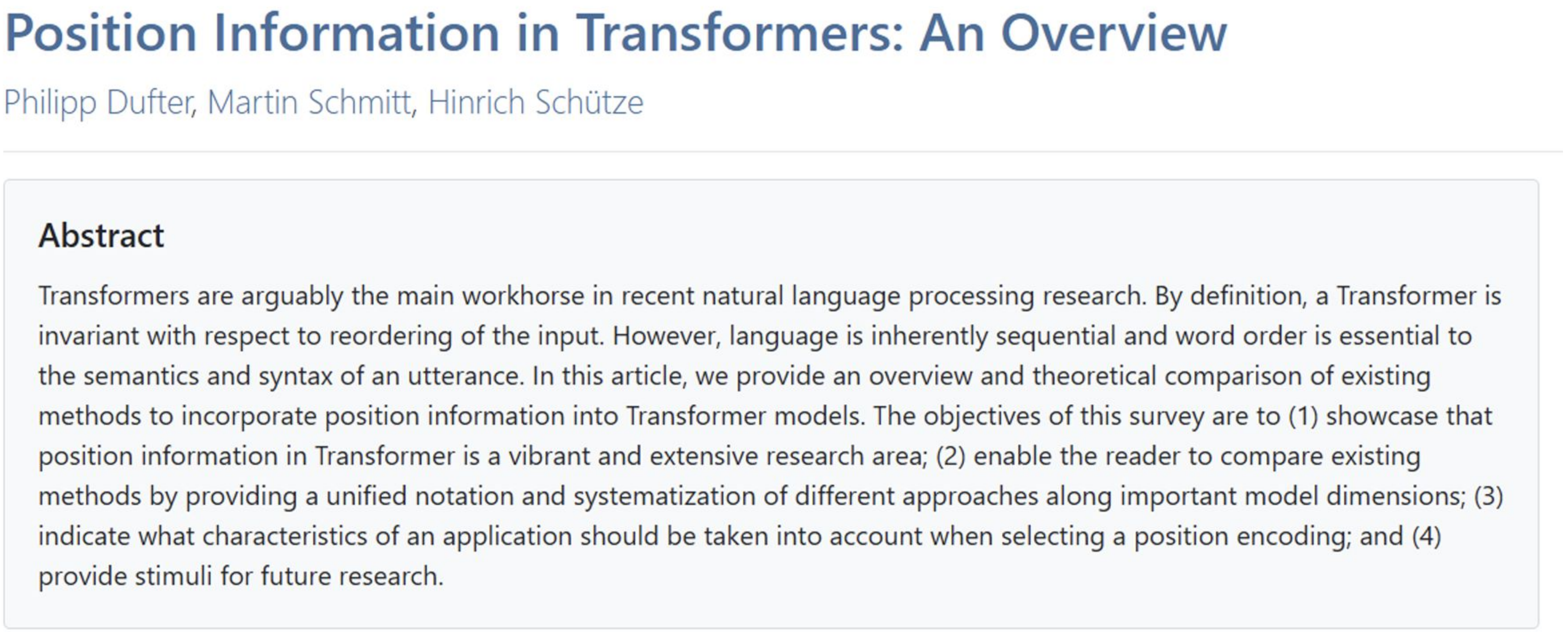
Idris Abdulmumin
Postdoctoral Research Fellow,
DSFSI, University of Pretoria
https://abumafrim.github.io/

Transformer



Positional Encoding

Positional Encoding



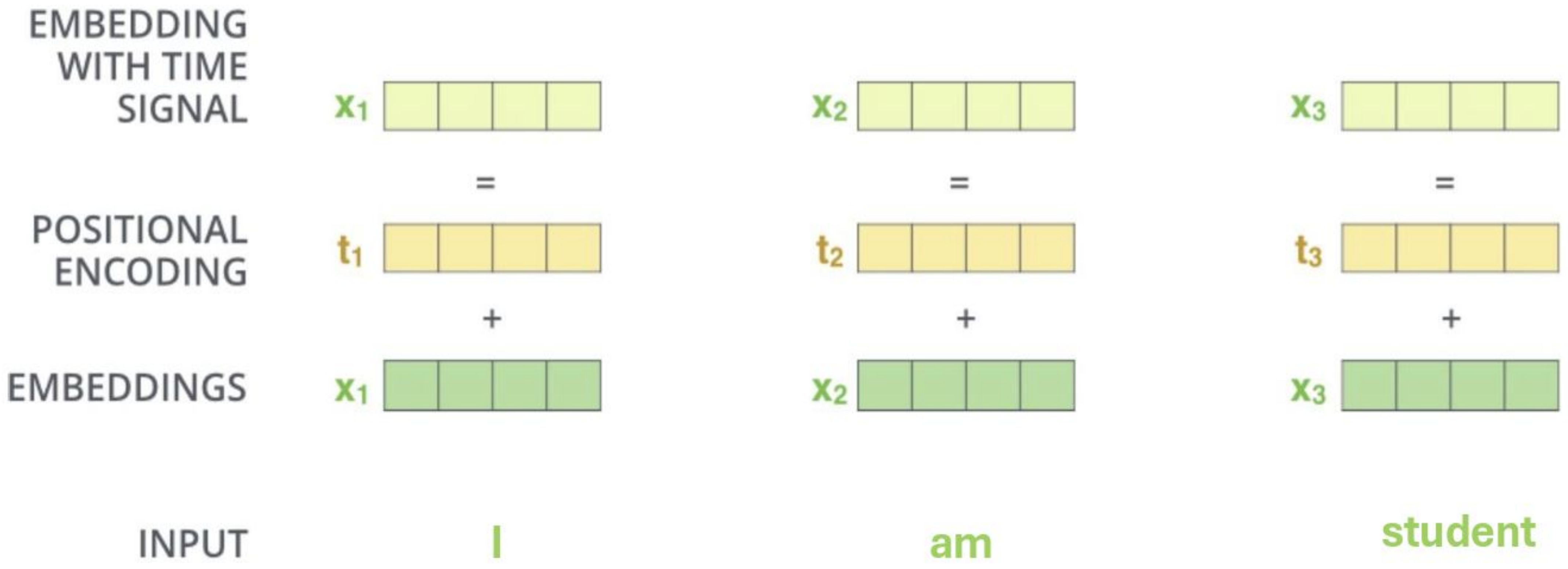
Position Information in Transformers: An Overview

Philipp Dufter, Martin Schmitt, Hinrich Schütze

Abstract

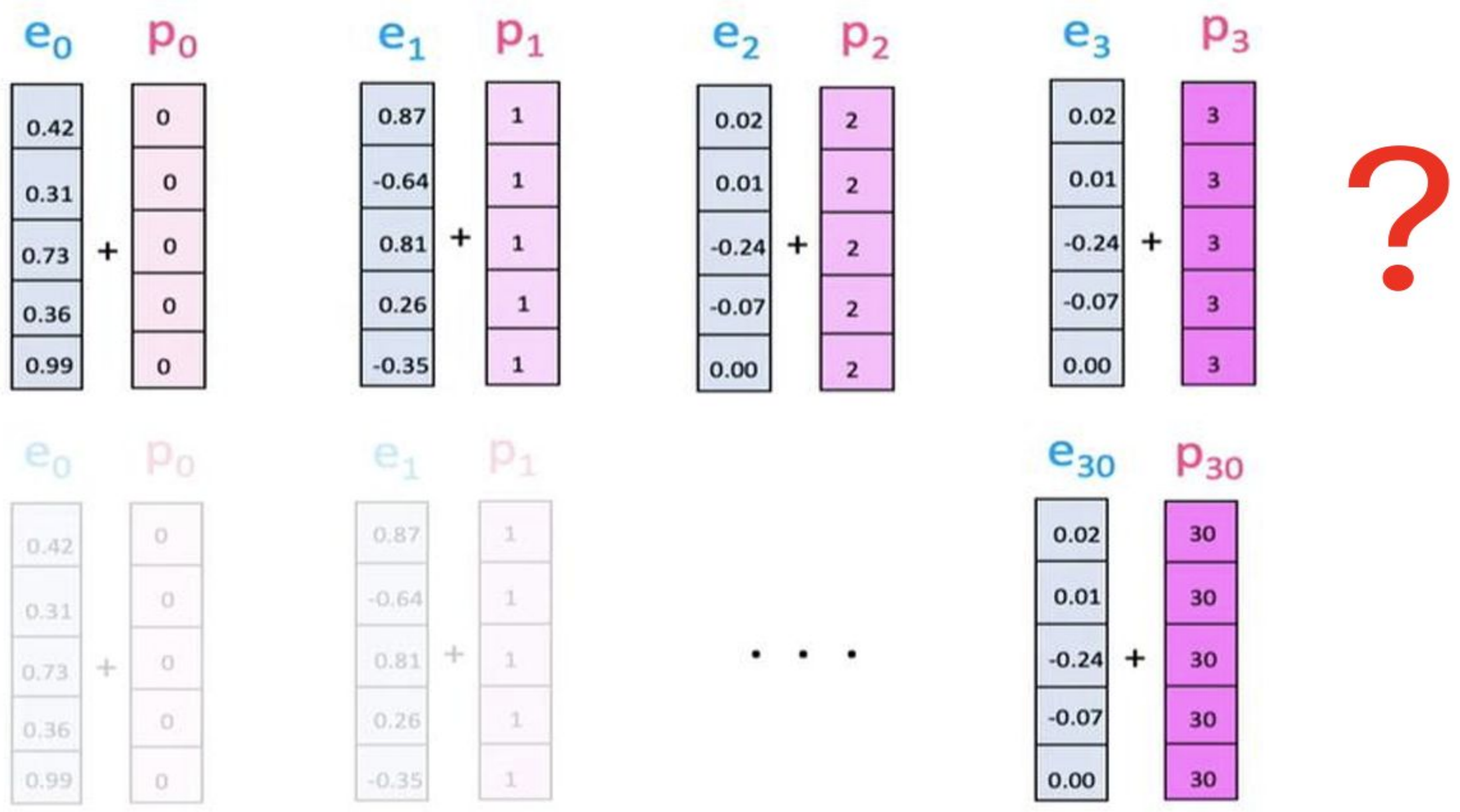
Transformers are arguably the main workhorse in recent natural language processing research. By definition, a Transformer is invariant with respect to reordering of the input. However, language is inherently sequential and word order is essential to the semantics and syntax of an utterance. In this article, we provide an overview and theoretical comparison of existing methods to incorporate position information into Transformer models. The objectives of this survey are to (1) showcase that position information in Transformer is a vibrant and extensive research area; (2) enable the reader to compare existing methods by providing a unified notation and systematization of different approaches along important model dimensions; (3) indicate what characteristics of an application should be taken into account when selecting a position encoding; and (4) provide stimuli for future research.

Positional Encoding

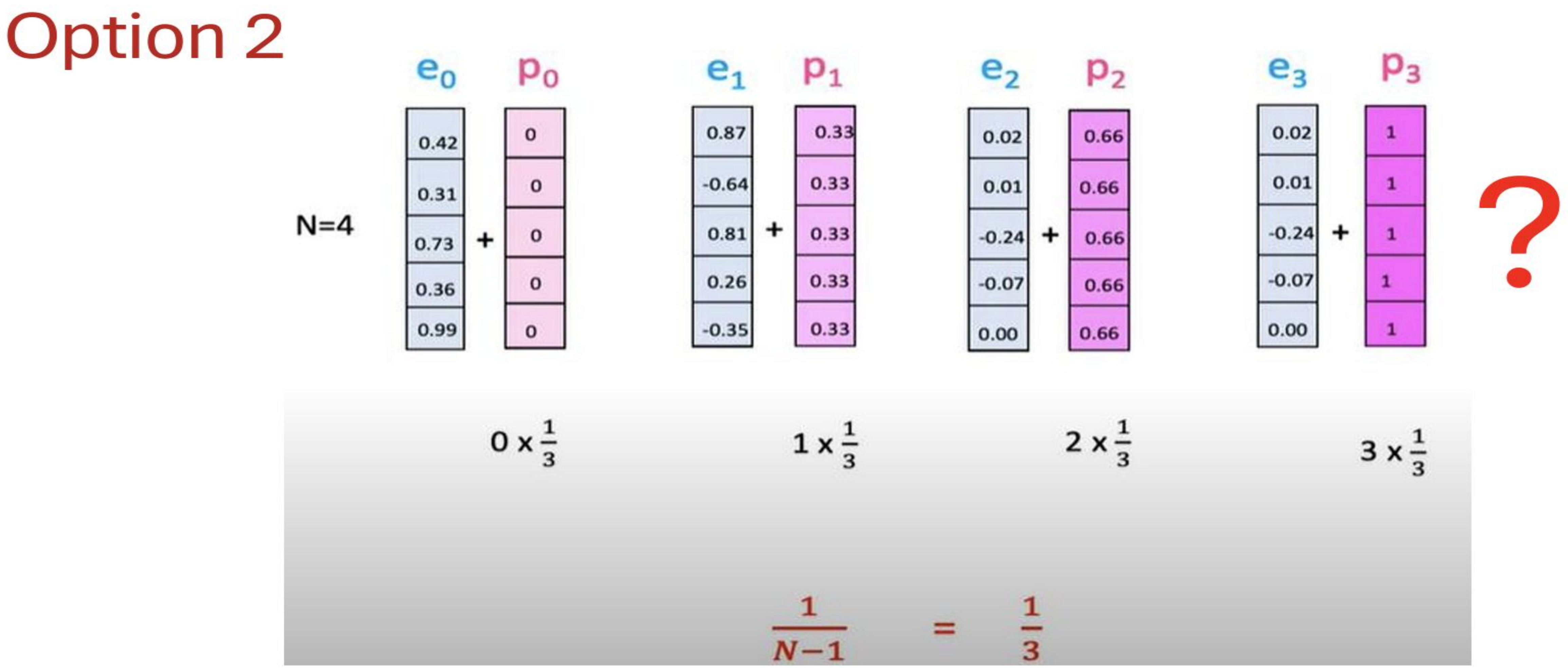


Positional Encoding

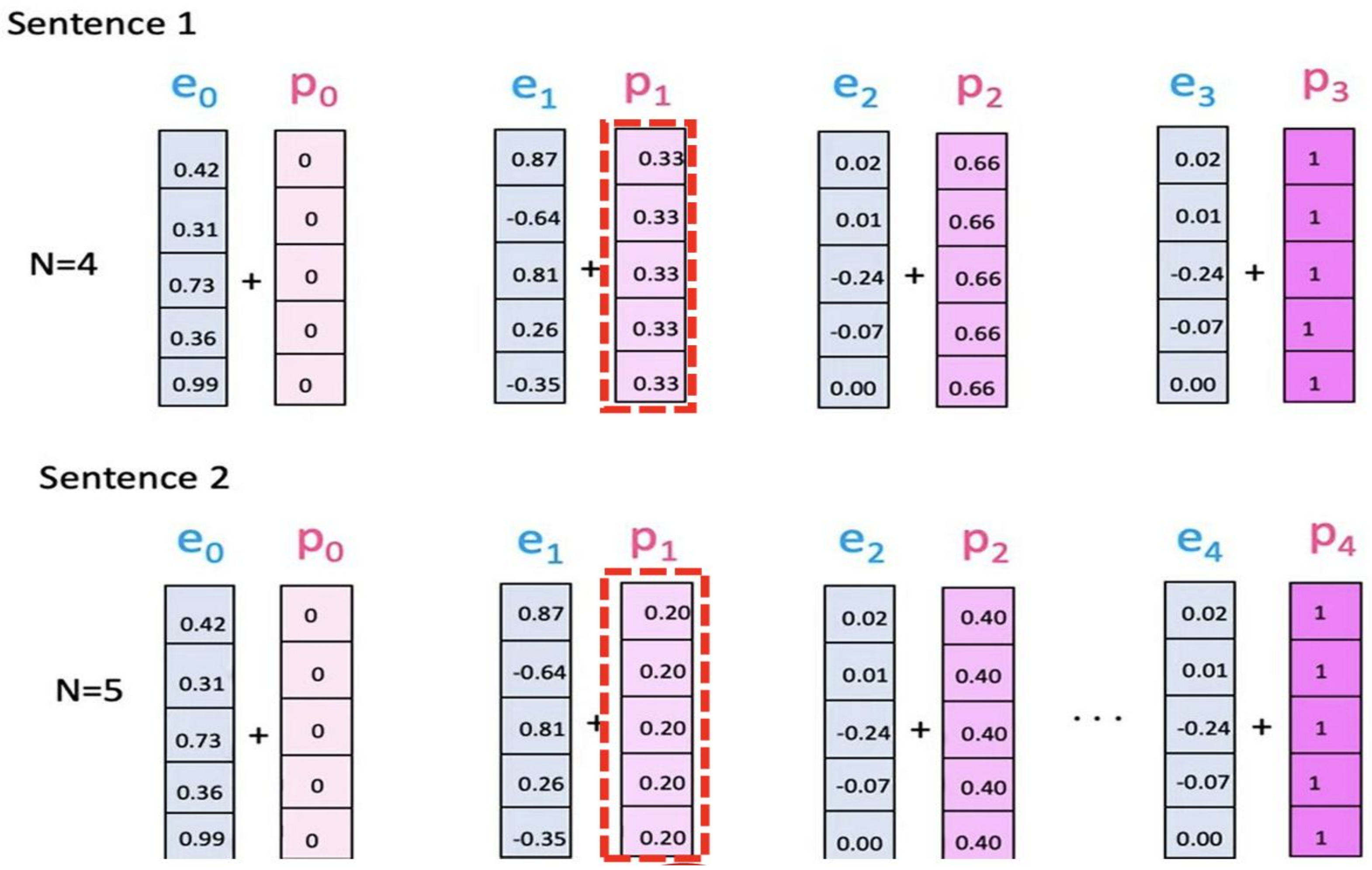
Option 1



Positional Encoding



Positional Encoding

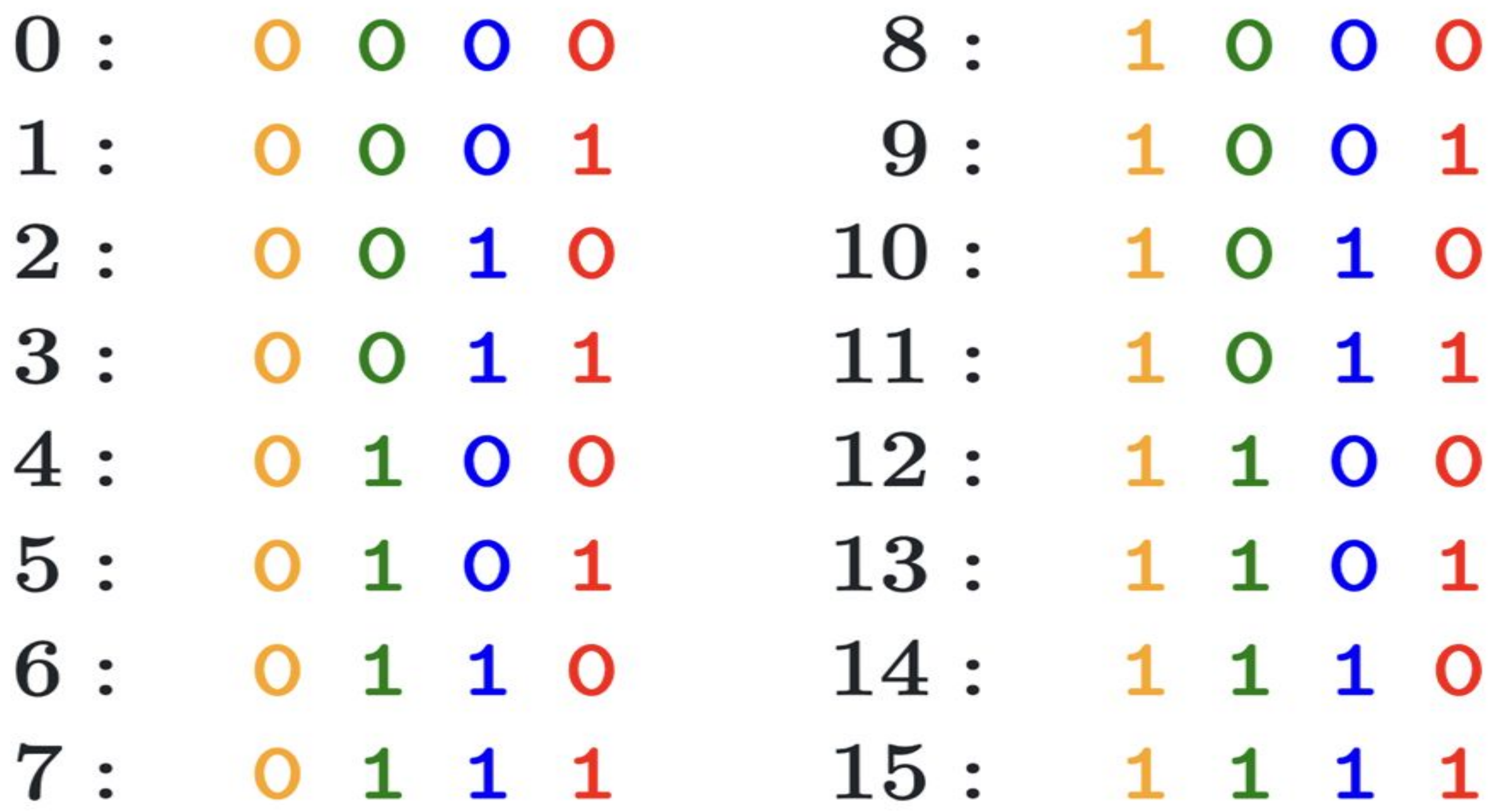


The positional embedding vector at a given position should remain the same irrespective of the length of the sequence

Creating Positional Encodings

- We want something that can generalize to arbitrary sequence lengths. We also may want to make attending to *relative positions* (e.g., tokens in a local window to the current token) easier.
- Distance between two positions should be consistent with variable-length inputs

Intuitive Example



0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

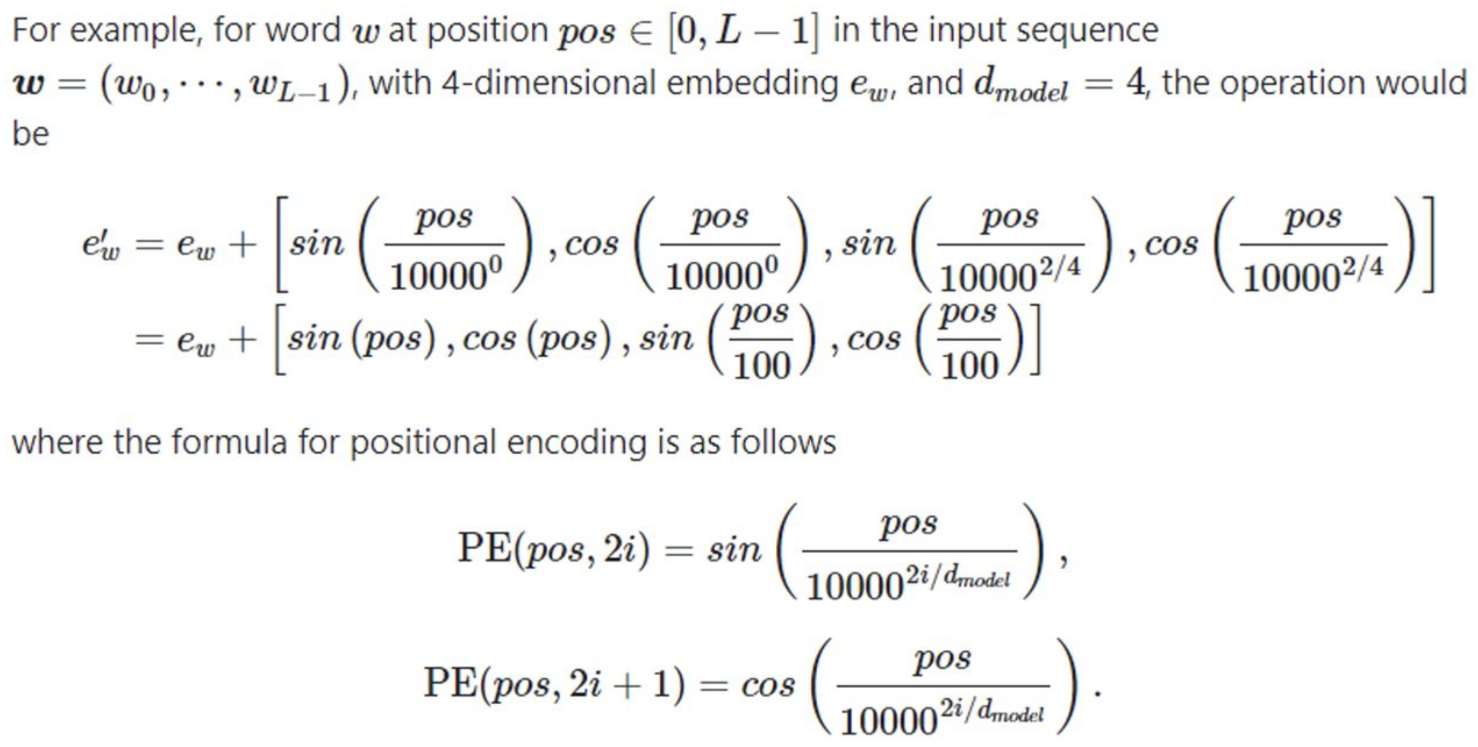
Transformer Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

For $\mathbf{d_{model} = 512}$,
Positional encoding is a 512-dimensional vector
(Note: **Dimension of positional encoding is same as dimension of the word embeddings**)
i = a particular dimension of this vector
pos = position of the word in the sequence

Example



For example, for word w at position $pos \in [0, L-1]$ in the input sequence $\boldsymbol{w} = (w_0, \cdots, w_{L-1})$, with 4-dimensional embedding e_w, and $d_{model} = 4$, the operation would be

$$
\begin{aligned}
e'_w &= e_w + \left[sin\left(\frac{pos}{10000^0}\right), cos\left(\frac{pos}{10000^0}\right), sin\left(\frac{pos}{10000^{2/4}}\right), cos\left(\frac{pos}{10000^{2/4}}\right)\right] \\
&= e_w + \left[sin\left(pos\right), cos\left(pos\right), sin\left(\frac{pos}{100}\right), cos\left(\frac{pos}{100}\right)\right]
\end{aligned}
$$

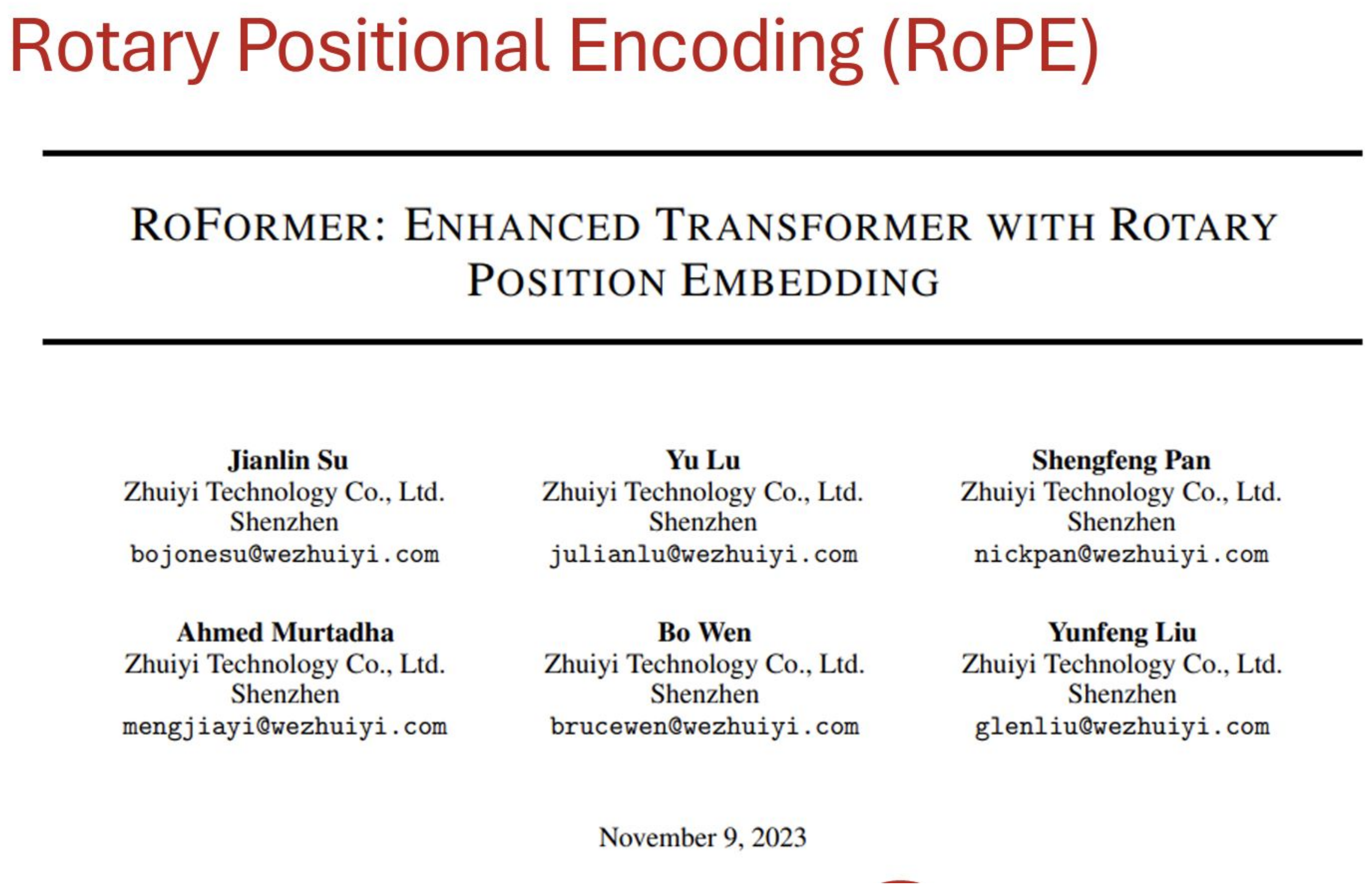
where the formula for positional encoding is as follows

$$
\mathrm{PE}(pos, 2i) = sin\left(\frac{pos}{10000^{2i/d_{model}}}\right),
$$

$$
\mathrm{PE}(pos, 2i+1) = cos\left(\frac{pos}{10000^{2i/d_{model}}}\right).
$$

https://datascience.stackexchange.com/questions/51065/what-is-the-positionalencoding-in-the-transformer-model

Rotary Positional Encoding (RoPE)



Rotary Positional Encoding (RoPE)

ROFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING

Jianlin Su
Zhuiyi Technology Co., Ltd.
Shenzhen
bojonesu@wezhuiyi.com

Yu Lu
Zhuiyi Technology Co., Ltd.
Shenzhen
julianlu@wezhuiyi.com

Shengfeng Pan
Zhuiyi Technology Co., Ltd.
Shenzhen
nickpan@wezhuiyi.com

Ahmed Murtadha
Zhuiyi Technology Co., Ltd.
Shenzhen
mengjiayi@wezhuiyi.com

Bo Wen
Zhuiyi Technology Co., Ltd.
Shenzhen
brucewen@wezhuiyi.com

Yunfeng Liu
Zhuiyi Technology Co., Ltd.
Shenzhen
glenliu@wezhuiyi.com

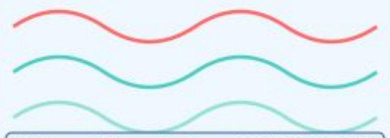
November 9, 2023

Adopted by
- PaLM
- GPT–Neo and GPT-J
- LlaMa 1 and 2

Summary Positional Encoding

Sinusoidal

(Original Transformer)

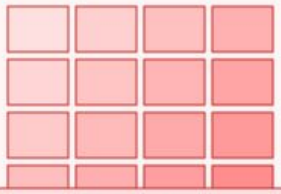


$PE(pos,2i) = \sin(pos/10000^{(2i/d)})$
$PE(pos,2i+1) = \cos(pos/10000^{(2i/d)})$

✓ Deterministic
✓ Extrapolation

Learned

(BERT, GPT)

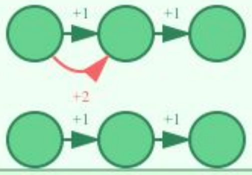


$PE(pos) = E[pos]$ (lookup table)

✓ Task-adaptive
✗ Fixed max length
✗ Poor extrapolation

Relative

(Transformer-XL, T5)



$A_{ij} = Q_i \cdot K_j + R(i-j)$

✓ Length generalization
✓ Relative distances
✗ More complex

RoPE

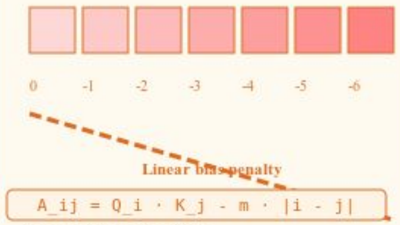
(Rotary Position Embedding)

$f(q,m) = q \cdot e^{(im\theta)},\ \theta = 10000^{(-2k/d)}$

✓ Relative encoding
✓ Good extrapolation
✓ Used in LLaMA, GPT-Neo

ALiBi

(Attention with Linear Biases)



$A_{ij} = Q_i \cdot K_j - m \cdot |i - j|$

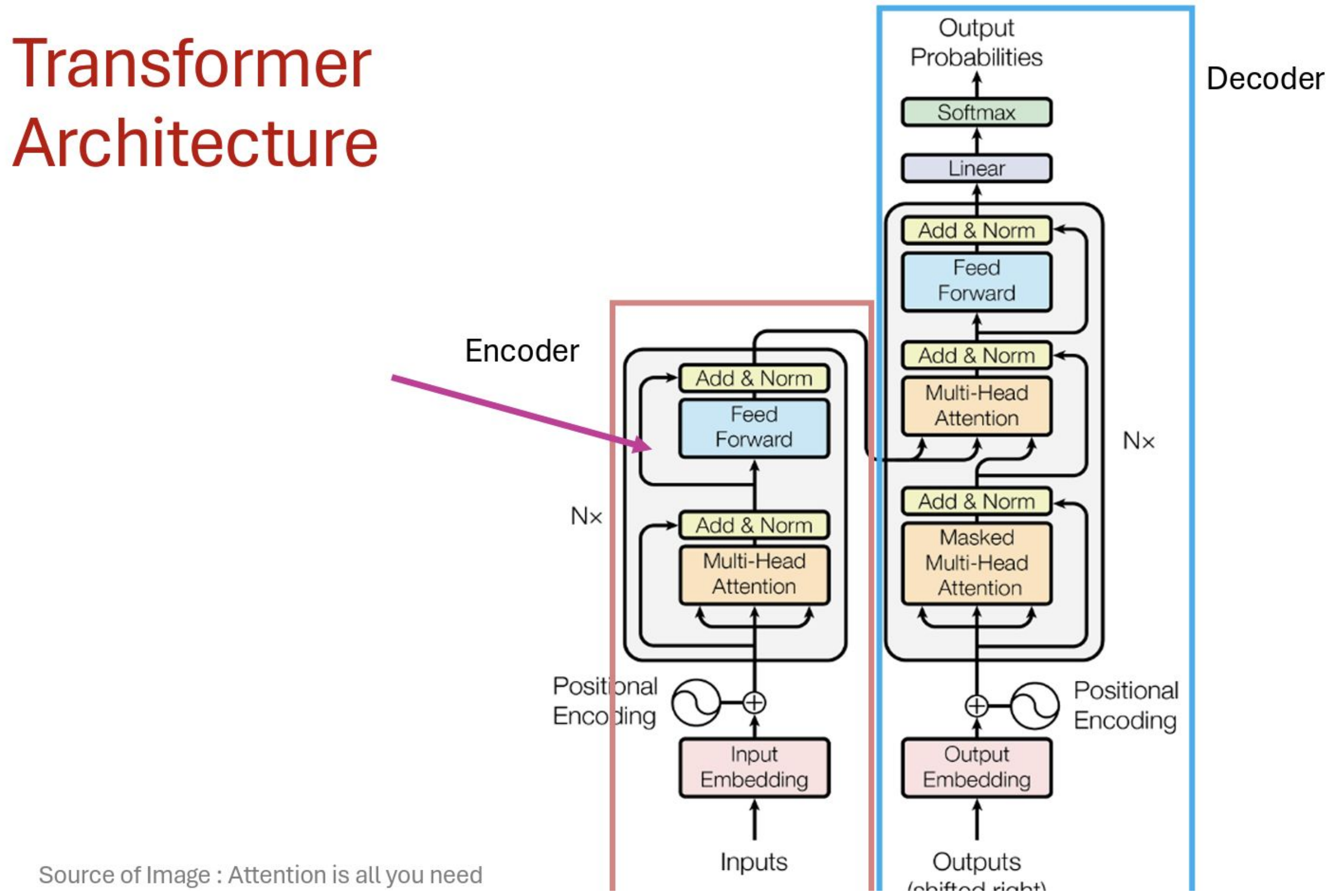
✓ No position embeddings
✓ Excellent extrapolation
✓ Memory efficient

Each method offers different trade-offs between complexity, extrapolation ability, and computational efficiency

Positional Encoding
References

- RoFormer: Enhanced Transformer with Rotary Position Embedding
- Layer Normalization
- Build Better Deep Learning Models with Batch and Layer Normalization

Residual Connections



Residual Connections

Residual connections, or **skip connections**, were introduced in ResNets and adopted in Transformers.

They allow the input to a layer to be added back to its output:

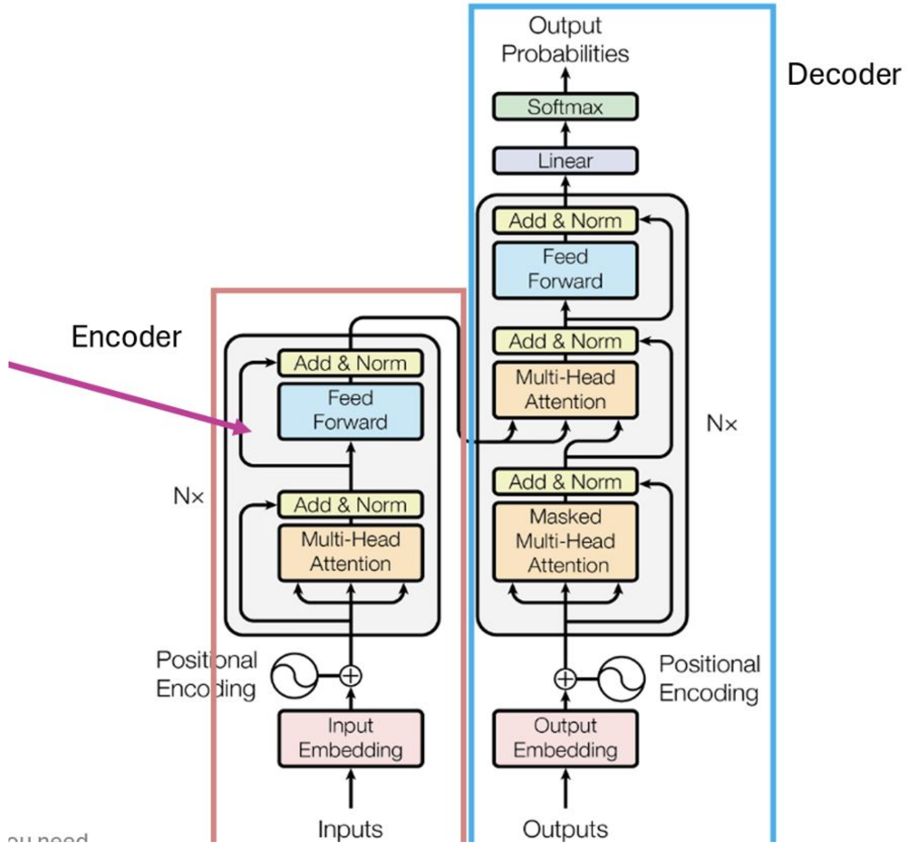
$$\text{Output} = \text{LayerNorm}(X + \text{Sublayer}(X))$$

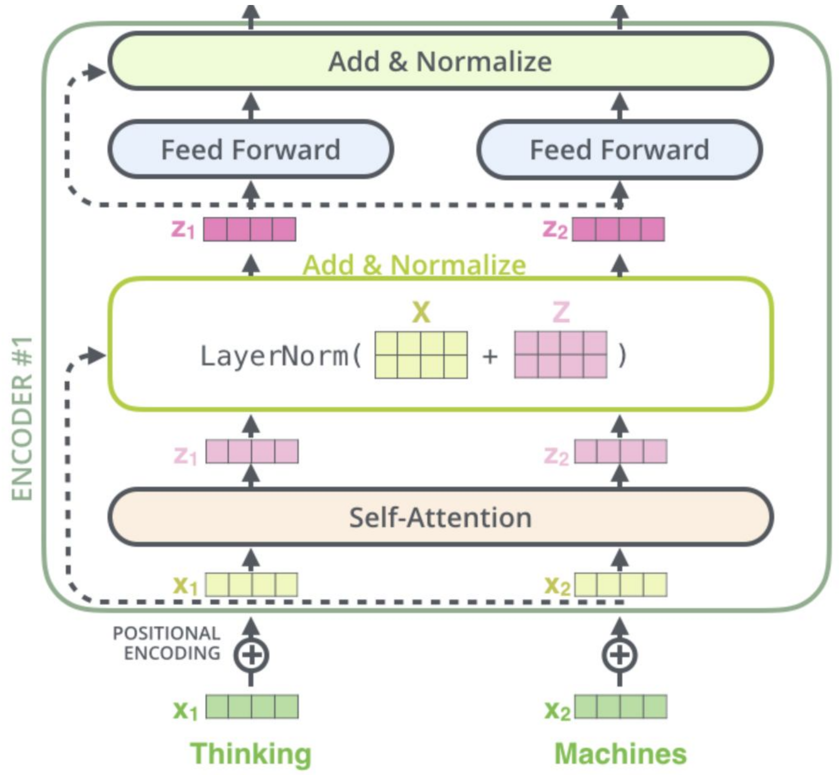
Used in:

- Multi-head self-attention sublayer
- Feed-forward network (FFN) sublayer

Residual Connections

Residual: appear before every LayerNorm in attention and FFN blocks





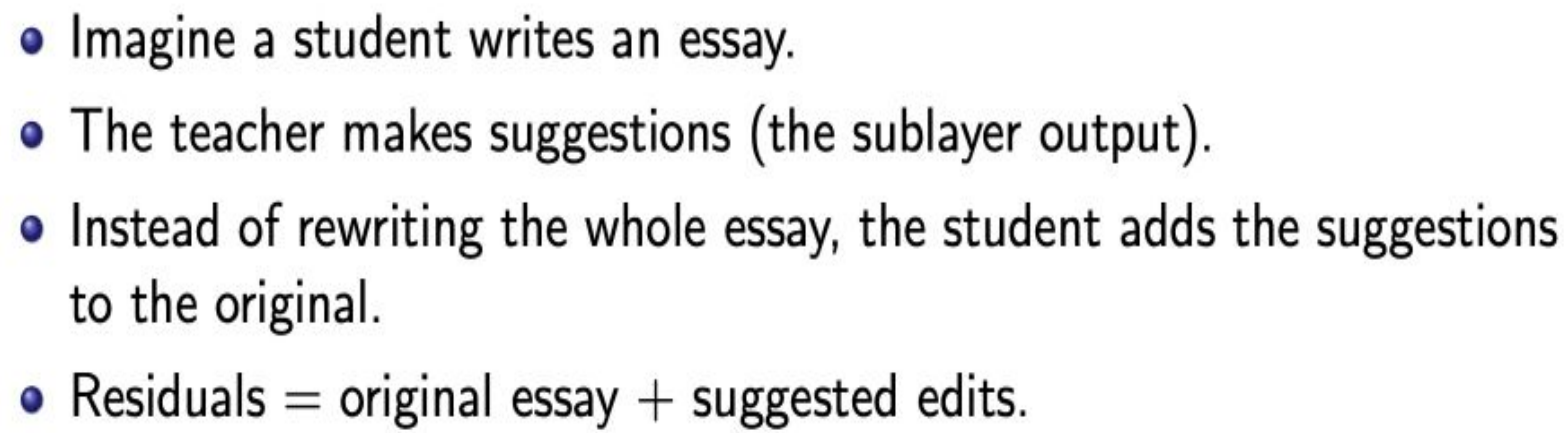
Why Use Residual Connections?

Stabilizes deep models by mitigating vanishing/exploding gradients.

Preserves useful input information during learning.

Allows the network to **learn incremental updates** rather than complete transformations.

Analogy: Essay Drafts



- Imagine a student writes an essay.
- The teacher makes suggestions (the sublayer output).
- Instead of rewriting the whole essay, the student adds the suggestions to the original.
- Residuals = original essay + suggested edits.

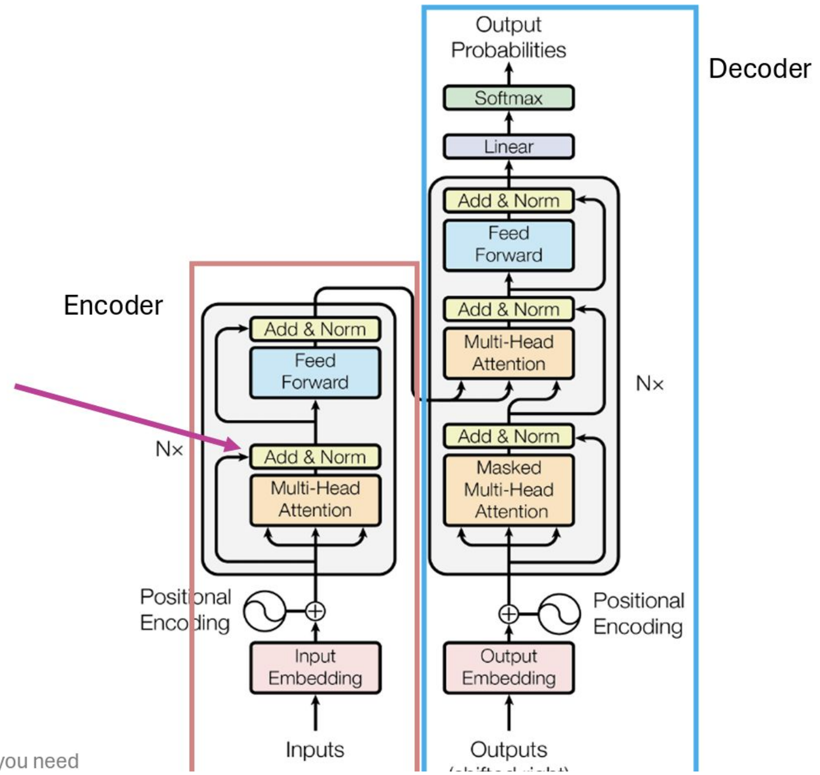
Analogy: Essay Drafts

operation applied **after each sublayer**

- self-attention
- feed-forward

Crucial for information flow.

Transformer Architecture

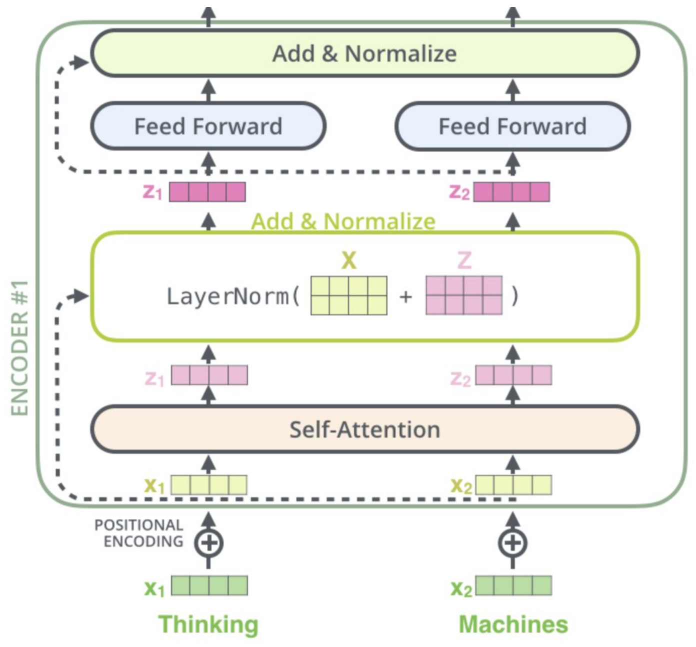


Source of Image : Attention is all you need

Add and Norm operations

- Combines two key operations:
  1. **Add**: A residual connection that adds the input to the sublayer output.
  2. **Norm**: A Layer Normalization step applied after addition.
- Formula:

$$\text{Output} = \text{LayerNorm}(X + \text{Sublayer}(X))$$



Add: example

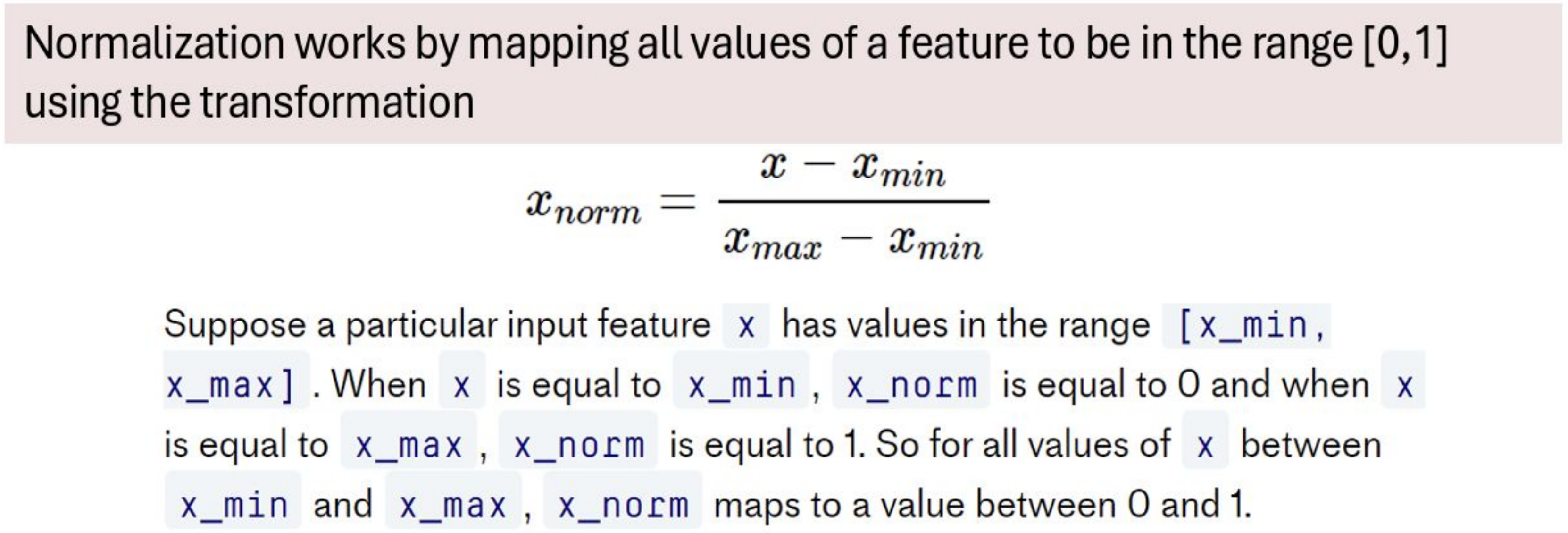
- Suppose you're training a model to translate: “The cat sat on the mat.” → “Le chat s’est assis sur le tapis.”
- Imagine one sublayer (like attention) changes the representation of “cat.” If that change is bad (due to random initialization or bad gradient), residual connections allow the model to **retain the original embedding** of “cat.”
- **Without residual**: The model fully trusts the transformation—even if it's wrong.
- **With residual**: The model can keep the original meaning and gradually improve over time.

Add: analogy

- Imagine a student revising an essay.
- They get suggestions from a teacher (the sublayer), but they don’t delete their original draft.
- Instead, they **add** the suggestions to the original to make a better version.
- If the suggestions aren’t good, the original still survives.

What is Normalization?

- Example: student loans with the age of the student and the tuition as two input features
  - two values are on totally different scales.
    - the age of a student will have a median value in the range 18 to 25 years
    - the tuition could take on values in the range $20K - $50K for a given academic year.



Normalization works by mapping all values of a feature to be in the range [0,1] using the transformation

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Suppose a particular input feature `x` has values in the range `[x_min, x_max]`. When `x` is equal to `x_min`, `x_norm` is equal to 0 and when `x` is equal to `x_max`, `x_norm` is equal to 1. So for all values of `x` between `x_min` and `x_max`, `x_norm` maps to a value between 0 and 1.

Standardization

- Example: student loans with the age of the student and the tuition as two input features
  - two values are on totally different scales.
    - the age of a student will have a median value in the range 18 to 25 years
    - the tuition could take on values in the range \$20K - \$50K for a given academic year.

Standardization transforms the input values such that they follow a distribution with zero mean and unit variance.

$$x_{std} = \frac{x - \mu}{\sigma}$$

In practice, this process of *standardization* is also referred to as *normalization*

Types of Normalization

Batch Normalization

Layer Normalization

Instance Normalization

Weight Normalization

Group Normalization

Example: Norm

- Let's say the output vector for a token is: [2.0,−1.0,3.5,−0.5,4.2]
- Without *normalization*, these values might be too large or too small, making training unstable.
- **LayerNorm** shifts and scales the vector to have:
  - Mean = 0
  - Variance = 1
- This keeps all tokens on a **comparable scale**, allowing faster and more stable convergence.

Analogy: Norm

- Think of training like baking different cakes.
- Layer Norm makes sure all your ingredients (features) are **measured on the same scale**—so you don’t end up adding too much sugar or salt by accident.

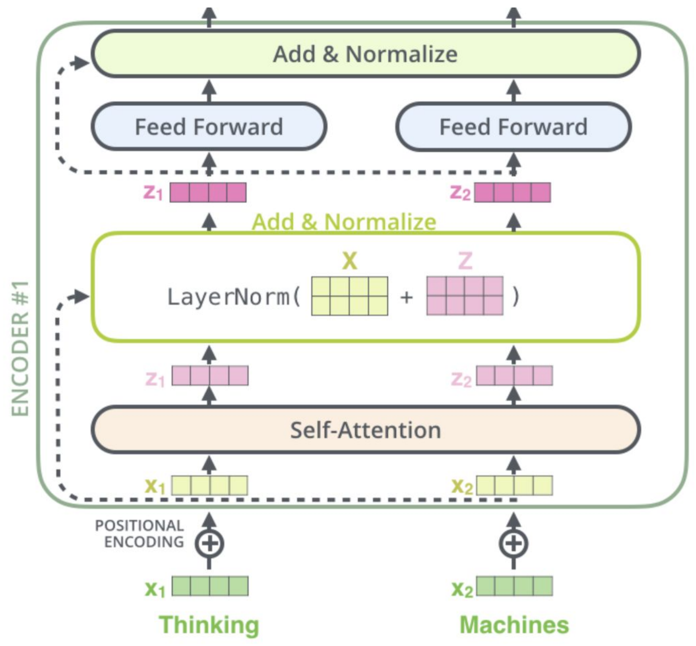
Where is add and Norm Used?

- Applied after each sublayer in both encoder and decoder:
  - Multi-head self-attention
  - Feed-forward network
- Ensures that the model can safely stack many layers.

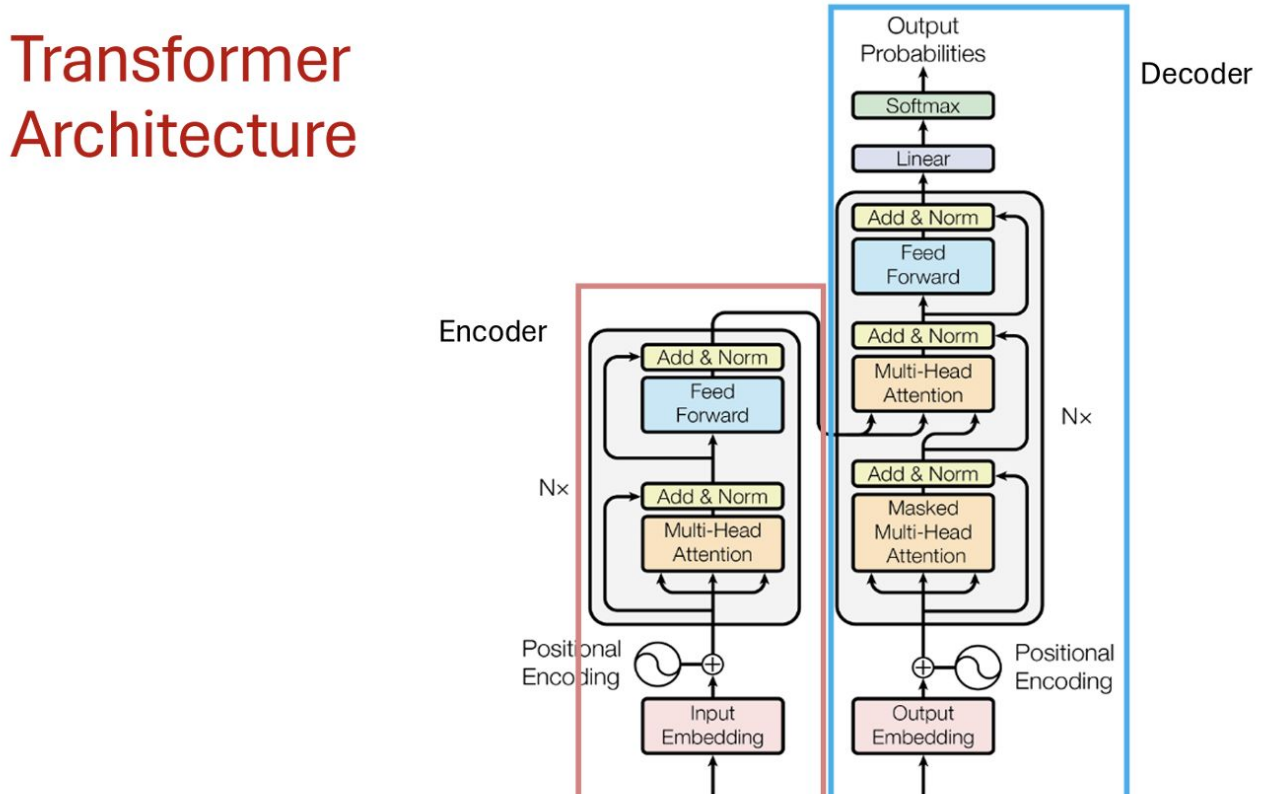
Pattern:

$$\text{LayerNorm}(X + \text{Attention}(X))$$

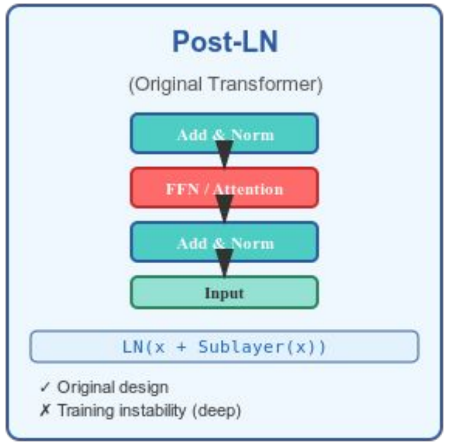
$$\text{LayerNorm}(X + \text{FFN}(X))$$



Where is add and Norm Used?

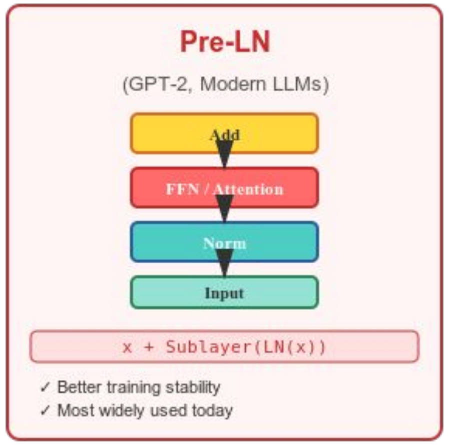


Summary Layer Norm



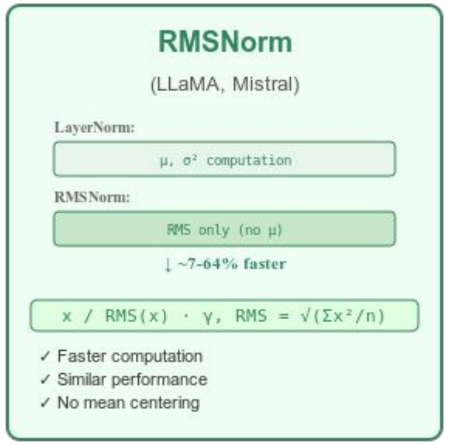
LN(x + Sublayer(x))

✓ Original design
✗ Training instability (deep)



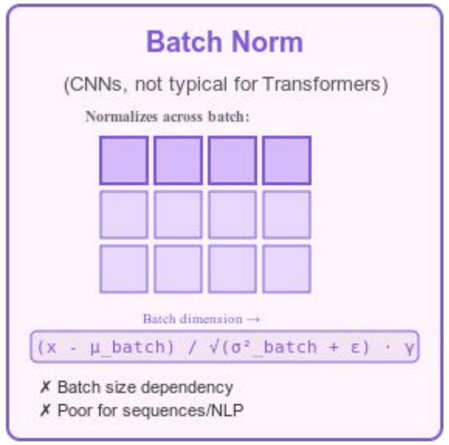
x + Sublayer(LN(x))

✓ Better training stability
✓ Most widely used today



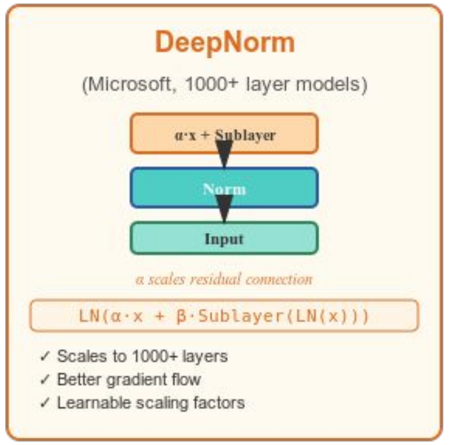
$x / \mathrm{RMS}(x) \cdot \gamma, \mathrm{RMS} = \sqrt{\Sigma x^2/n}$

✓ Faster computation
✓ Similar performance
✓ No mean centering



$(x - \mu\_batch) / \sqrt{\sigma^2\_batch + \varepsilon} \cdot \gamma$

✗ Batch size dependency
✗ Poor for sequences/NLP



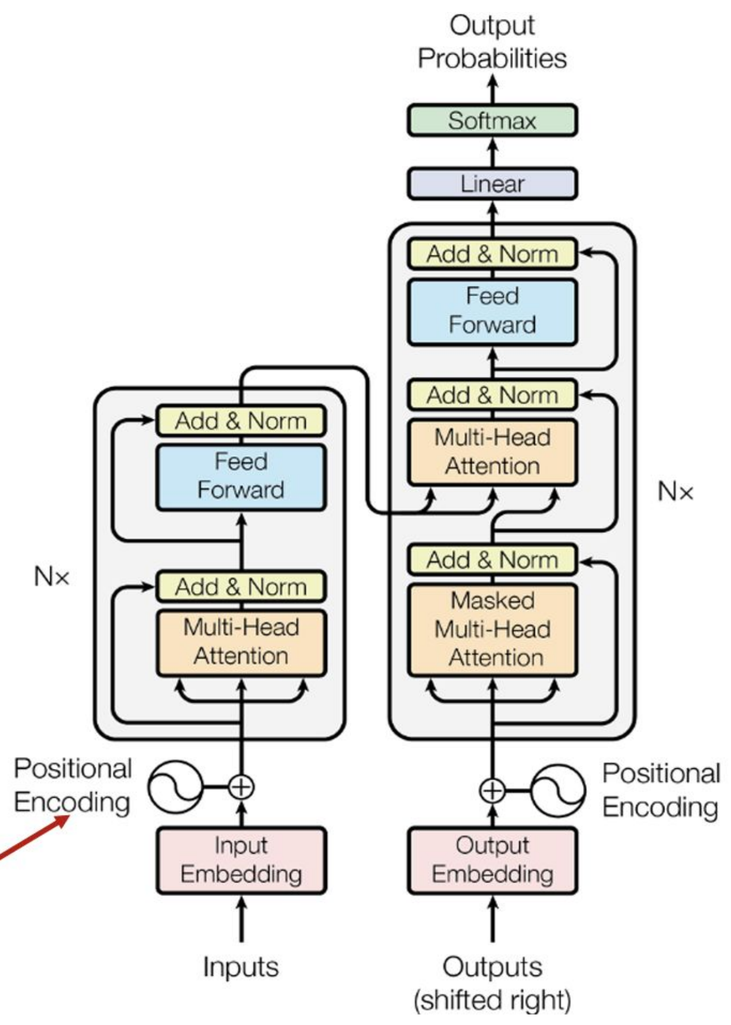
$\mathrm{LN}(\alpha \cdot x + \beta \cdot \mathrm{Sublayer}(\mathrm{LN}(x)))$

✓ Scales to 1000+ layers
✓ Better gradient flow
✓ Learnable scaling factors

Pre-LN and RMSNorm are most common in modern LLMs; DeepNorm enables extremely deep architectures

Summary: Transformer

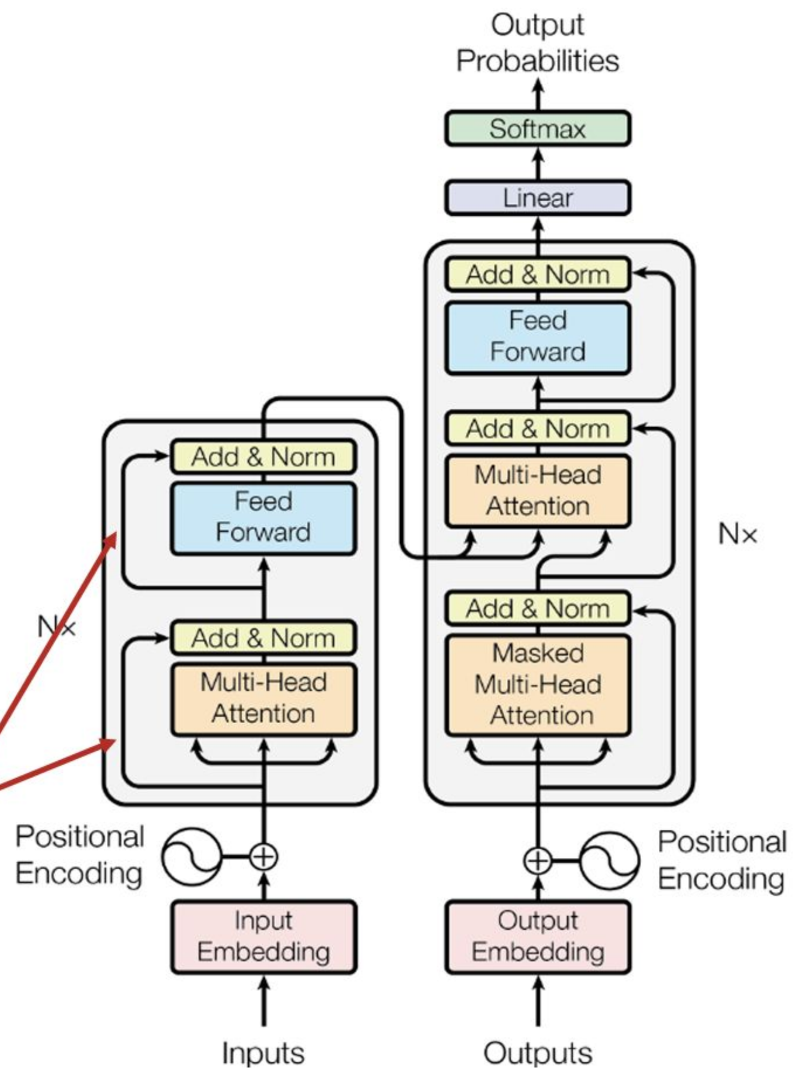
Transformer Architecture



Position embeddings are *added* to each word embedding. Otherwise, since we have no recurrence, our model is unaware of the position of a word in the sequence!

Summary: Transformer

Transformer Architecture

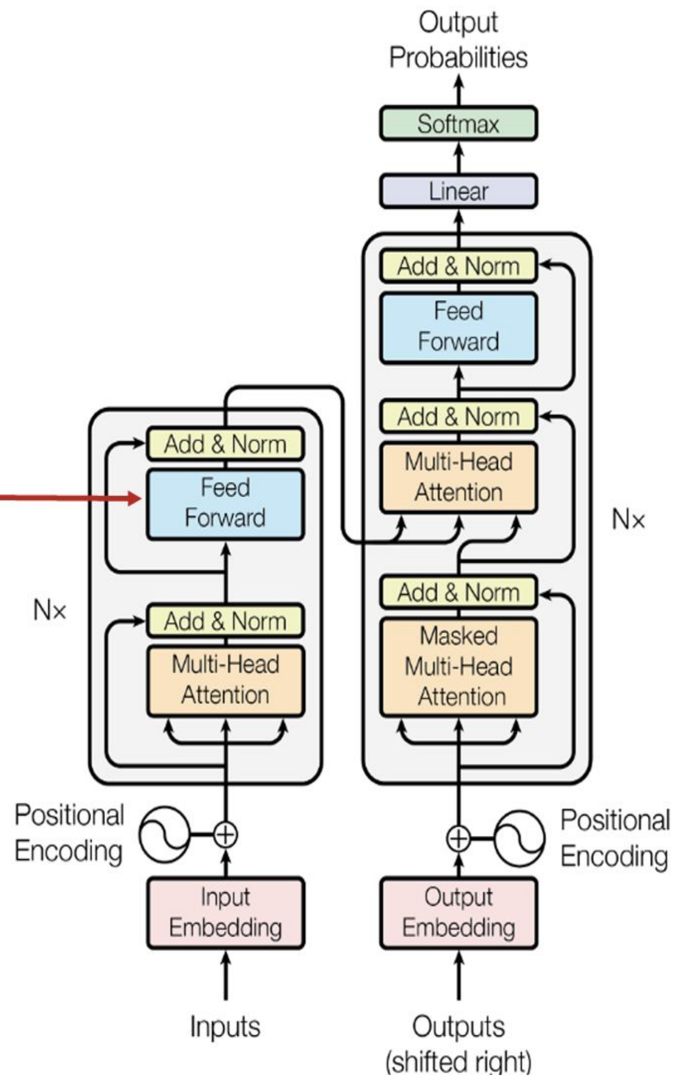


Residual connections, which mean that we add the input to a particular block to its output, help improve gradient flow

Summary: Transformer

Transformer Architecture

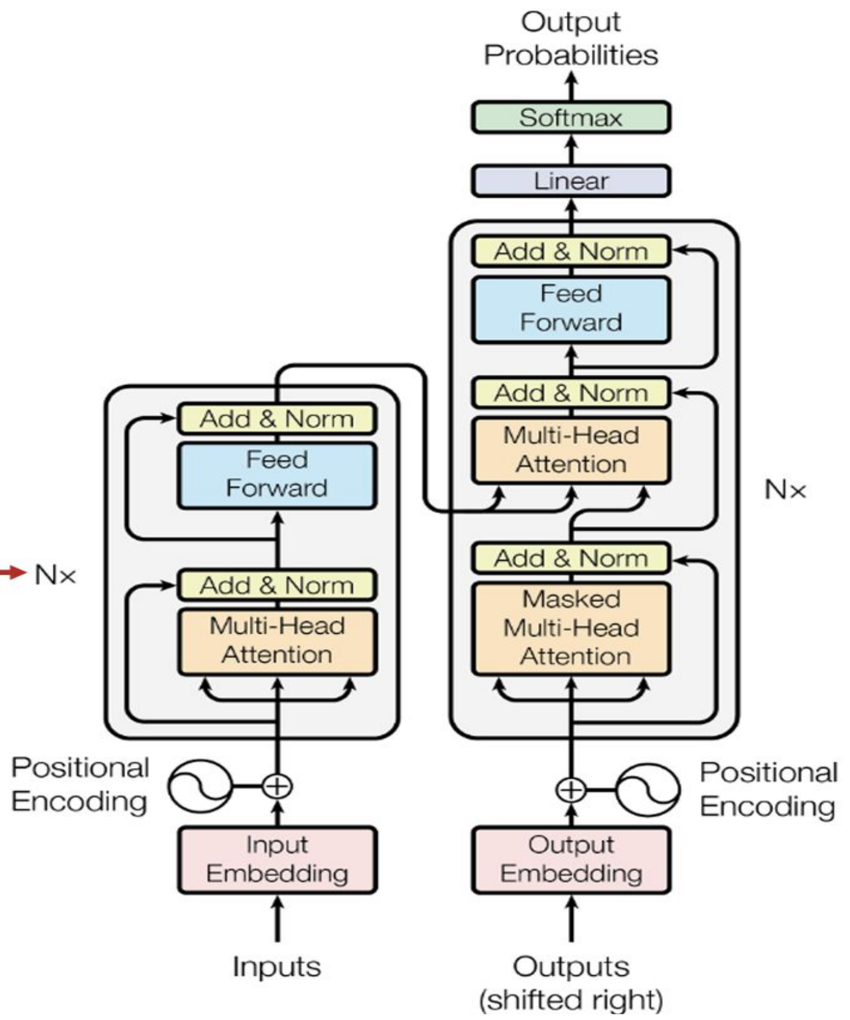
A **feed-forward layer** on top of the attention- weighted averaged value vectors allows us to add more parameters / nonlinearity



Summary: Transformer

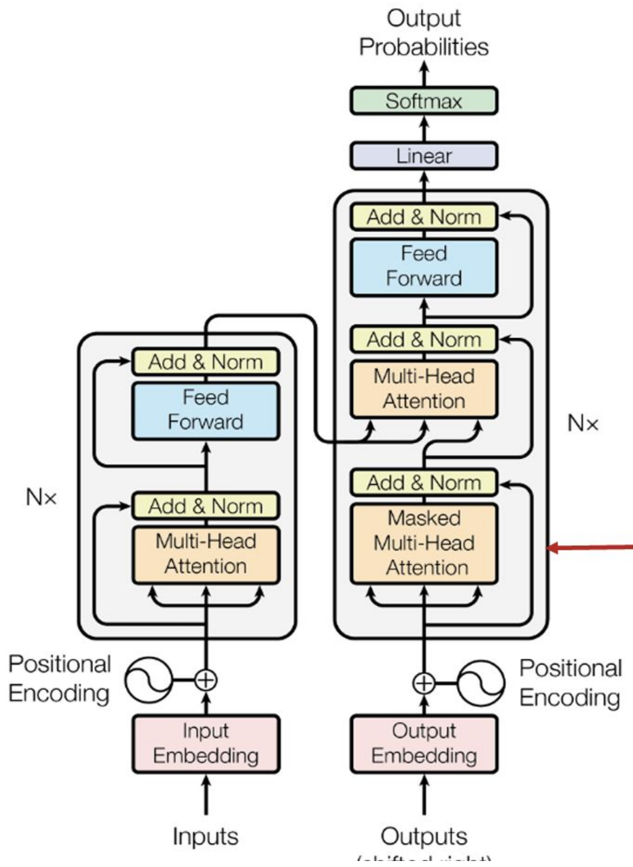
Transformer Architecture

We stack as many of these ***Transformer* blocks** on top of each other as we can (bigger models are generally better given enough data!)



Summary: Transformer

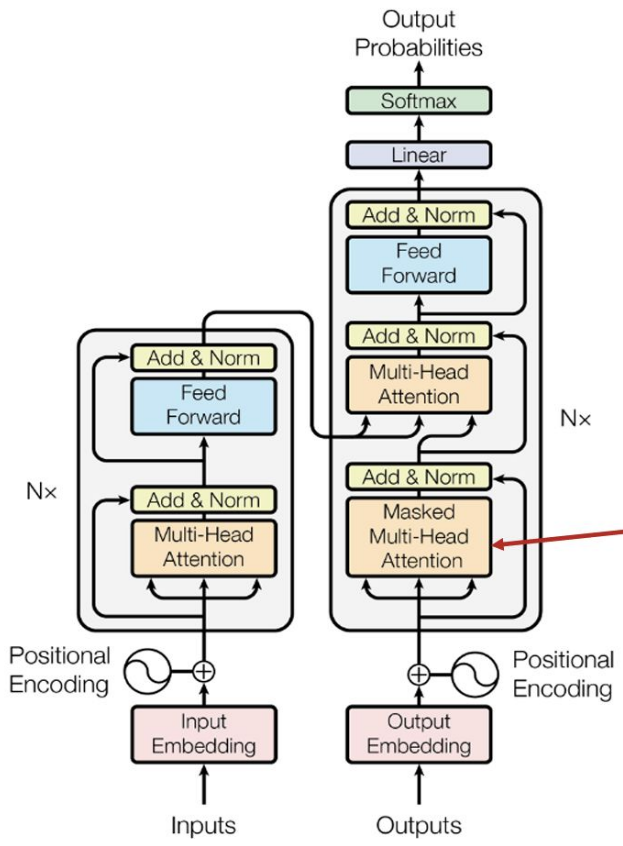
Transformer Architecture



Moving onto the decoder, which takes in English sequences that have been shifted to the right (e.g., *<START> schools opened their*)

Summary: Transformer

Transformer Architecture

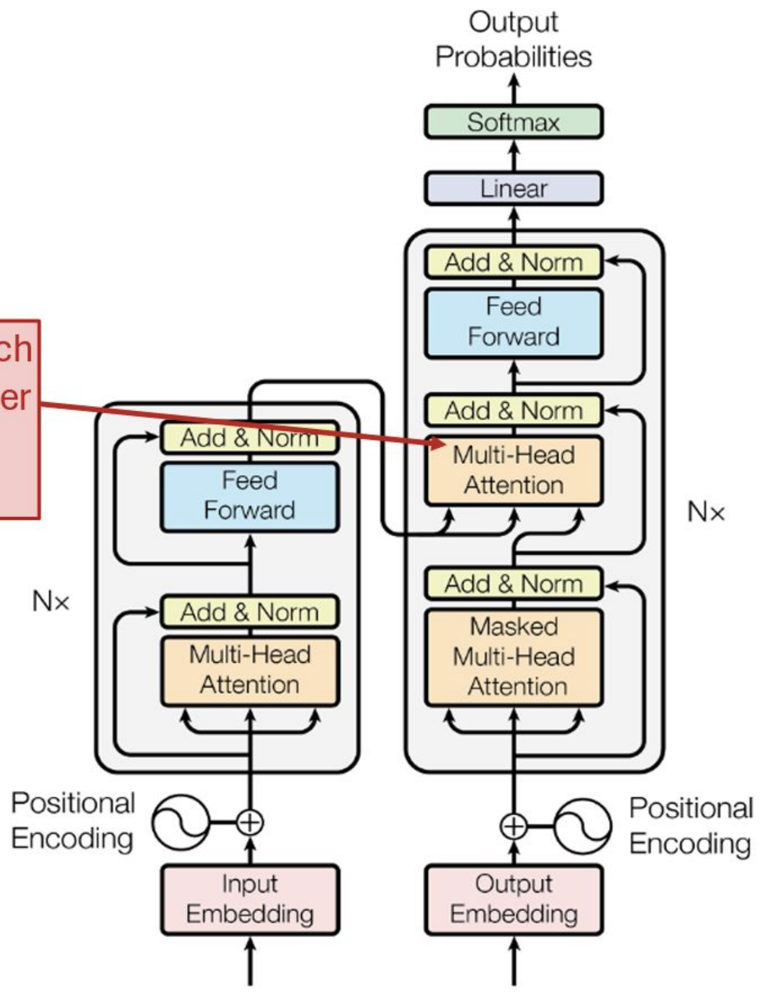


We first have an instance of **masked self attention**. Since the decoder is responsible for predicting the English words, we need to apply masking as we saw before.

Summary: Transformer

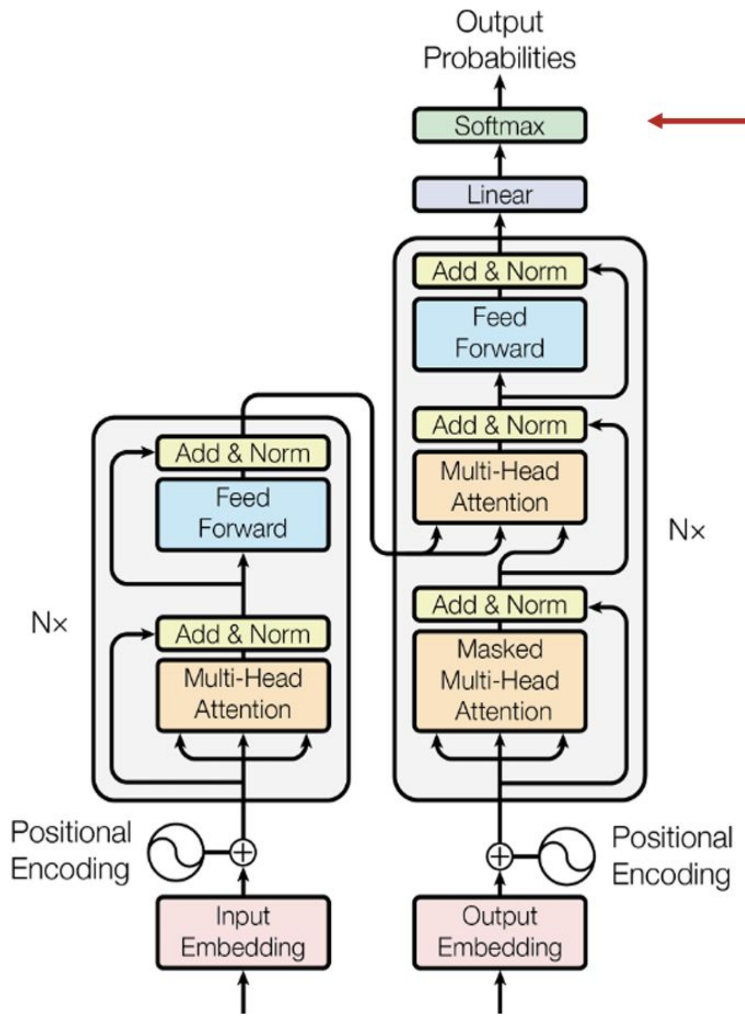
Transformer Architecture

Now, we have ***cross attention***, which connects the decoder to the encoder by enabling it to attend over the encoder's final hidden states.



Summary: Transformer

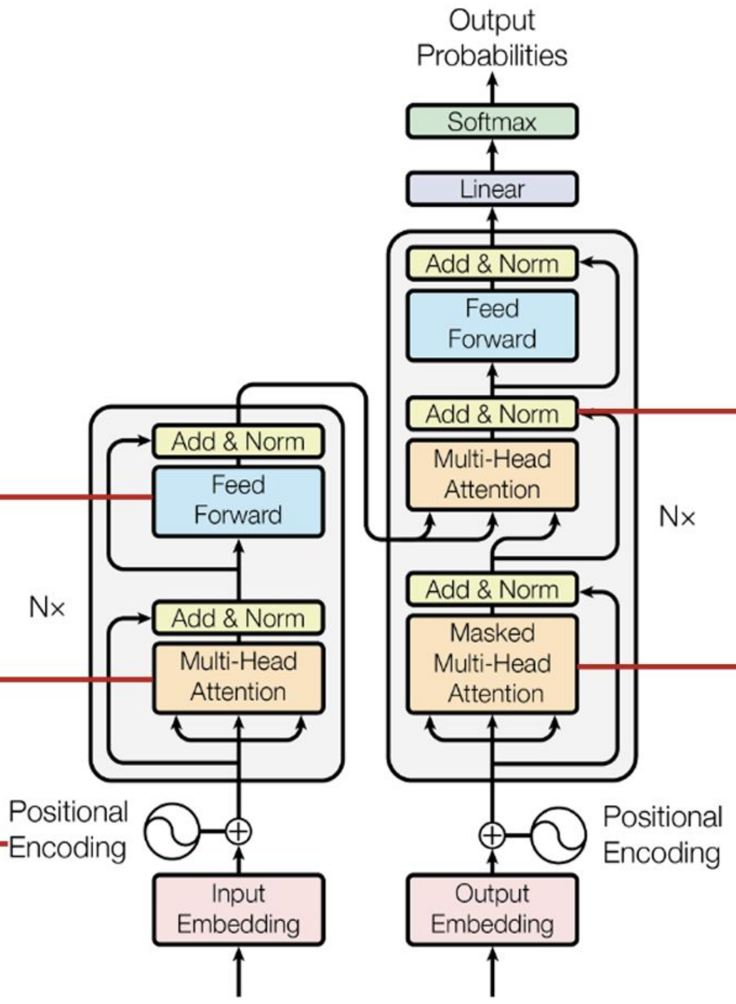
Transformer Architecture



After stacking a bunch of these decoder blocks, we finally have our familiar **softmax** layer to predict the next English word.

Summary: Transformer

Transformer Architecture



Reduces covariance shift and makes the system stable

Adding non-linearities

Allows querying multiple positions at each layer

Prevents attention lookups into the future while decoding

Adds positional information

Transformer Implementation in Pytorch

https://github.com/harvardnlp/annotated-transformer

IMPERIAL

Q and A