How LLM's work?

LLMs have been an integral part of all the research and also our day-to-day lives. So lets dive deep into understanding what exactly is happening behind the scenes.

How LLMs work? What do you think?

What happens at core?

Mathematical foundation

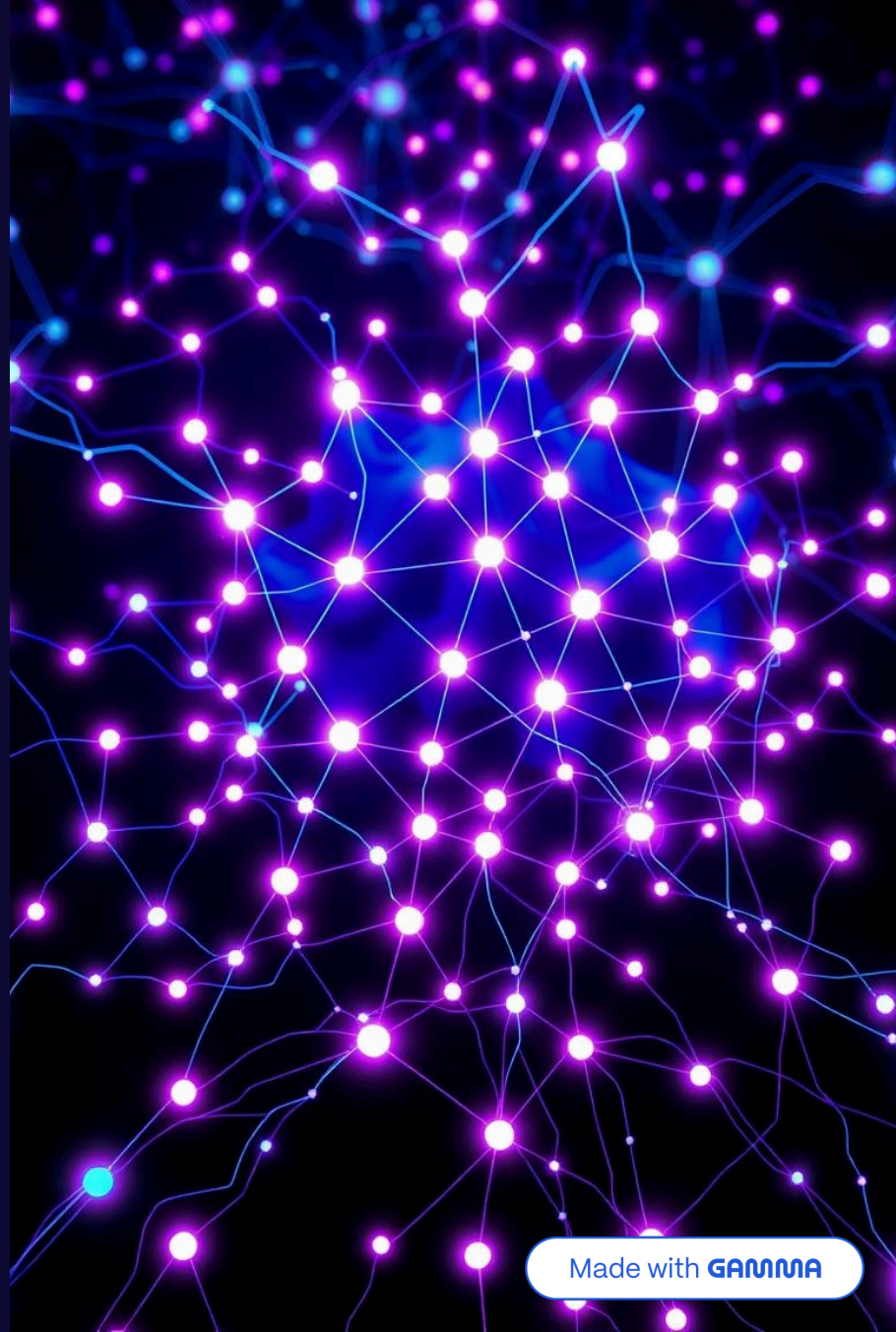
Its a giant neural network that do a lot of math.

Matrix multiplications

The LLMs learn to make sense out of data by multiplying matrices

Lets think of an analogy

Lego building blocks



How exactly LLMs make use of math

LLMs are made of linear layers

Each layer is a linear algebra with some non-linear function

This can be explained by an equation:

output = Activation(W × input + b) where,

W = weights and b = bias

Now what are these linear layers and non-linear functions?

Linear layers are matrix multiplication of weights and input matrix whereas, the non-linear functions are activation functions that help the model look at the data non-linearly

Tokenization: Breaking Text Into Pieces

Text Conversion

Sentences break into smaller units called tokens for processing.

Token Types

Tokens can be words, subwords, or characters based on the model.

Why is it required

Tokenization helps models handle language complexities effectively.

Lets think,

What do you think the LLMs really understands?

Encoding

1

What happens during encoding?

Mapping them to the LLM's vocabulary

2

Analogy

A - 1, B - 2, C - 3, etc. So if we have HI it is encoded into 89.

3

TikTokenizer

This is a platform that shows how a same sentence can be tokenized and encoded differently by different models.

Embeddings

The text encodings are embedded

The encodings, numerical values are converted into vectors i.e. projected into high dimensional space.

What do the embeddings represent?

They carry semantic meaning and context for example:

Synonyms tend to have similar embeddings. Simple arithmetic on embeddings can do things like “King – Man + Woman ≈ Queen”

Can you guess what is the dimension of embedding?

In Chatgpt, each word is represented as a point in a 12,288-dimensional space.

You can think of that space as an enormously high-resolution map where each coordinate captures some aspect of the word’s meaning or usage.

Transformers

A must read

Attention is All You Need

Introduced by Vaswani et al. in 2017

How does it process input?

The Transformer allowed language models to consider an entire sentence at once, rather than word-by-word sequentially.

In simple terms,

Transformers let every word look at every other word when computing its context, using a mechanism called self-attention.

Self Attention

Attention Mechanism

Each token’s vector is used to compute three new vectors – called Query (Q), Key (K), and Value (V) where,

Query (Q) represents what this word is looking for.

Key (K) represents "descriptor" of each potential answer.

Value (V) represents the actual "content" that the query is interested in.

“Alice gave Bob her book,” the word “her” will match strongly with “Alice” via a mechanism.

Mechanism

Let's consider, "A cat sat."

Query - cat is compared with key vectors A, cat and sat.

Attention scores are calculated which tells how much the query should attend to other words.

Weighted sum is calculated using value vectors and weights/ attention scores.

The weighted sum of the value vector forms the updated vector.

Feed Forward

What is feed forward?

After self-attention, each token’s new vector goes through a small neural network (the same one for every token) to further process the combined information.

This is again just a simple matrix multiplication.

Why is it important?

This layer adds more mixing of the features learned by attention.

$h = W * z + b$

$a = f(h)$

In generis terms,

"Feed-Forward" refers to a process or system where information flows in one direction, typically from the input to the output

Lets walk through the whole process

Tokenization

For simplicity, our “vocabulary” is just three tokens:

hello : 0

world : 1

! : 2

Input : world

world is mapped to ID 1.

The NNs don't understand integers. Hence we turn ID 1 into one-hot vector:

v = [0,1,0]

$$e = v\ E = [0,\ 1,\ 0] \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix} = [0.3,\ 0.4]$$

Embedding

We have an embedding matrix E, which we multiply with the input matrix.

$$[0.3,\ 0.4] \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + [0.5,\ 0.5]$$

Passing embedding through a neural network

output[2.0, 2.7]

This output is hidden vector. This vector is converted into logits and soft-max is applied to it to get the probability.

Based on the probability, the most probable token is chosen and detokenized into text.