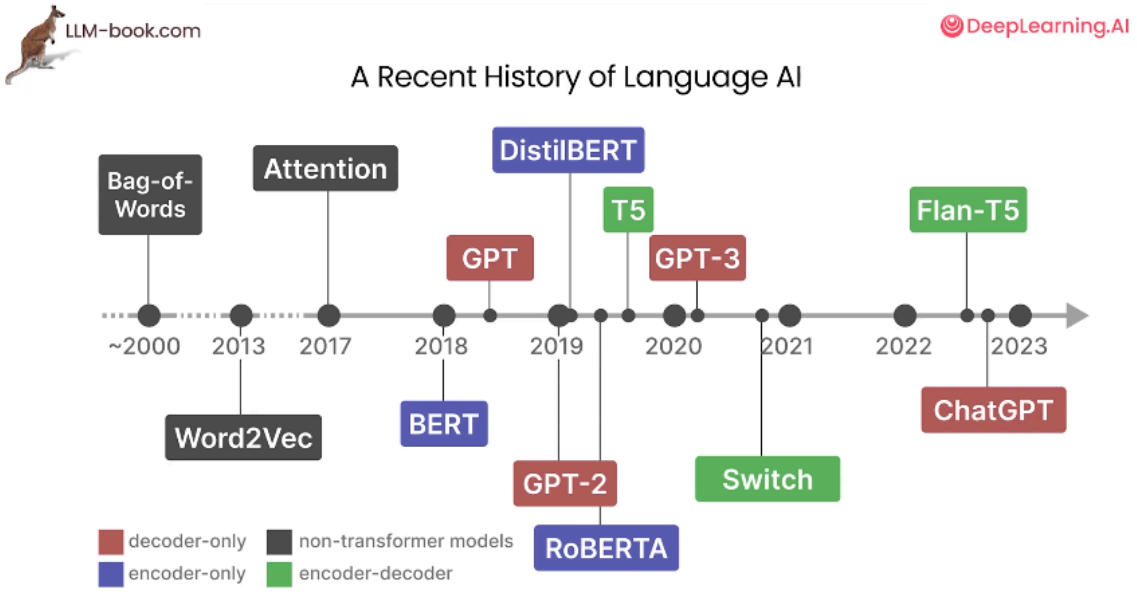
How Transformer LLMs Work

Created	@March 20, 2025 5:37 PM
Class	Transformer
Created by	malivinayak

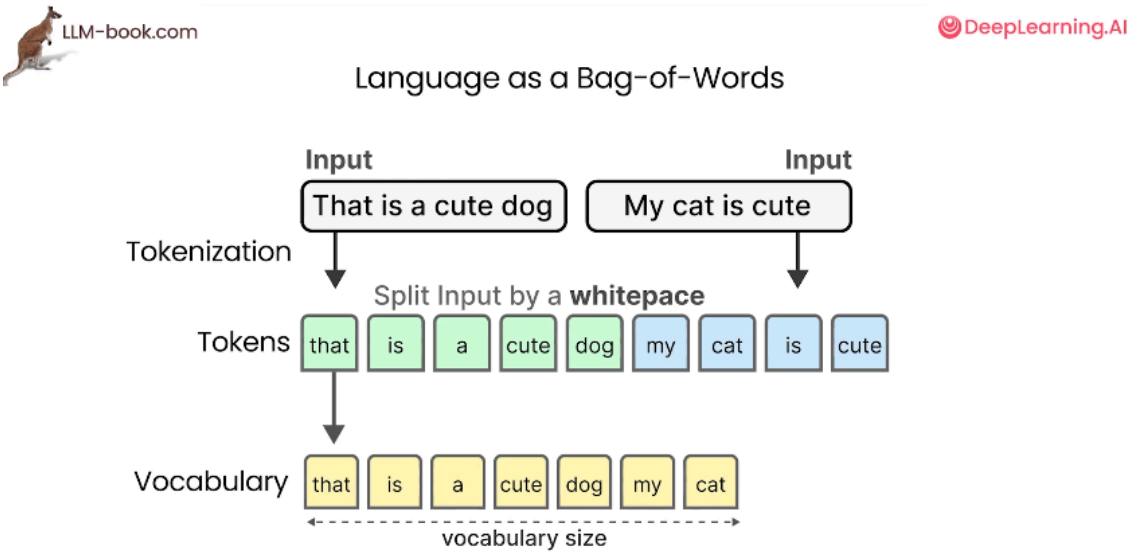
Course Link: https://learn.deeplearning.ai/courses/how-transformer-llms-work/lesson/nfshb/introduction

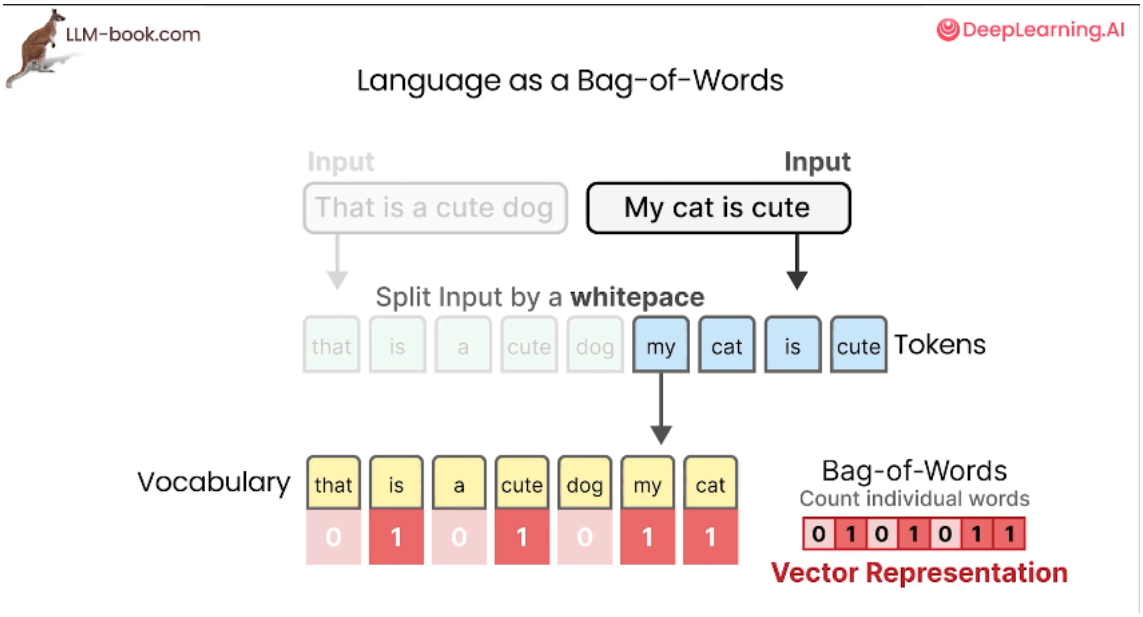
A recent History of Language AI

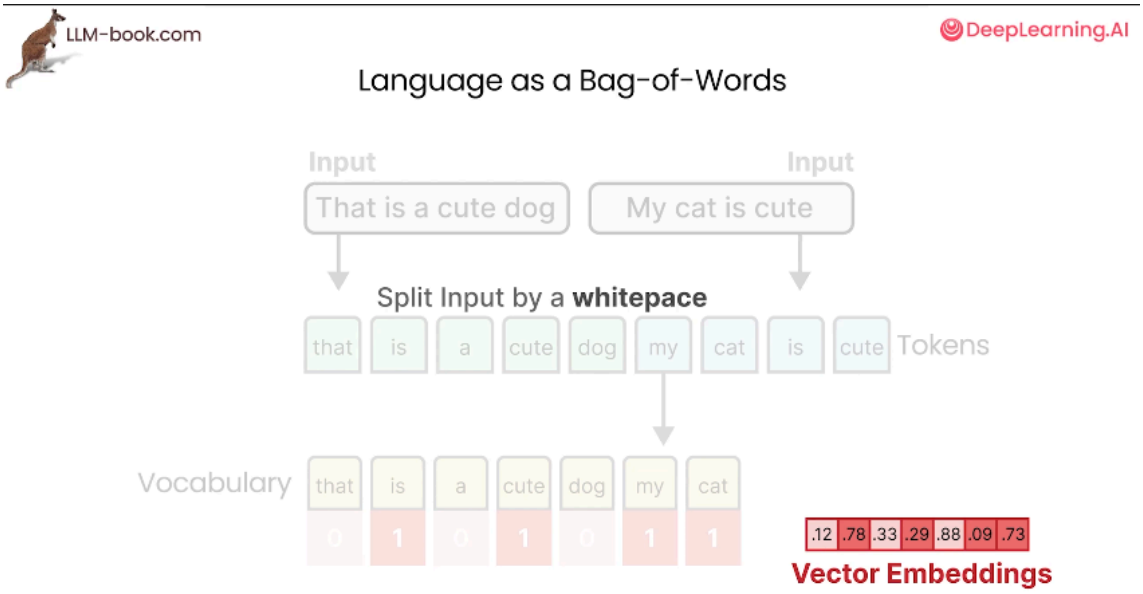


- Non-transformer model: base of all model
- Encoder-only model: Great at representing language in numerical representation
- Decoder-only model: Generative in nature (main usage is generating text)
- encoder-decoder model: attempt to get best of both

Language as a Bag-of-Words


LLM-book.com
DeepLearning.AI
Language as a Bag-of-Words
Input
That is a cute dog
Input
My cat is cute
Tokenization
Split Input by a whitepace
Tokens
that
is
a
cute
dog
my
cat
is
cute
Vocabulary
that
is
a
cute
dog
my
cat
vocabulary size


LLM-book.com
DeepLearning.AI
Language as a Bag-of-Words
Input
That is a cute dog
Input
My cat is cute
Split Input by a whitepace
that
is
a
cute
dog
my
cat
is
cute
Tokens
Vocabulary
that
is
a
cute
dog
my
cat
0
1
0
1
0
1
1
Bag-of-Words
Count individual words
0 1 0 1 0 1 1
Vector Representation

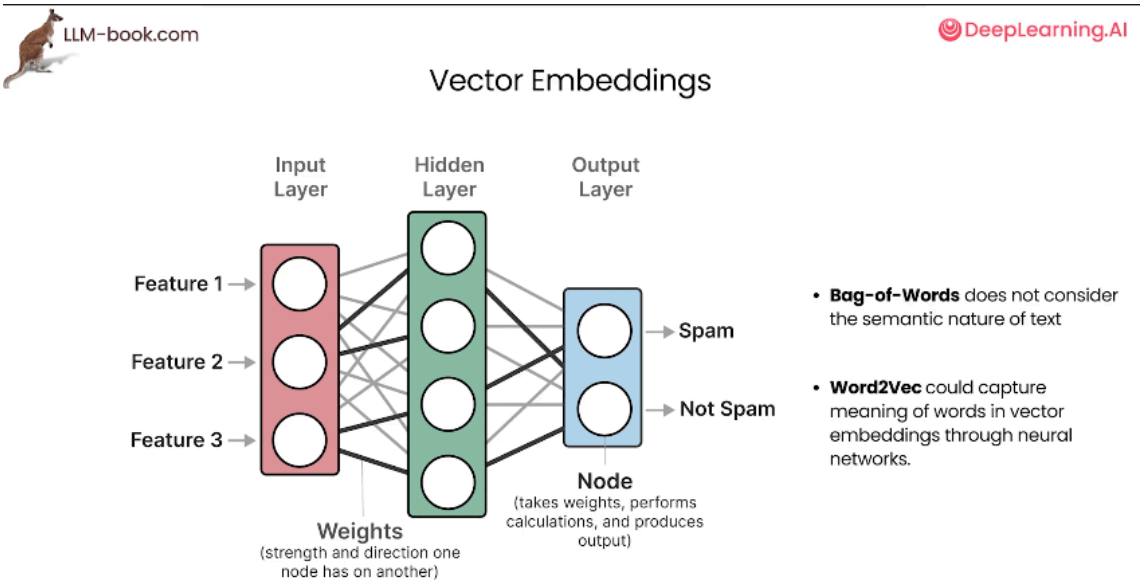


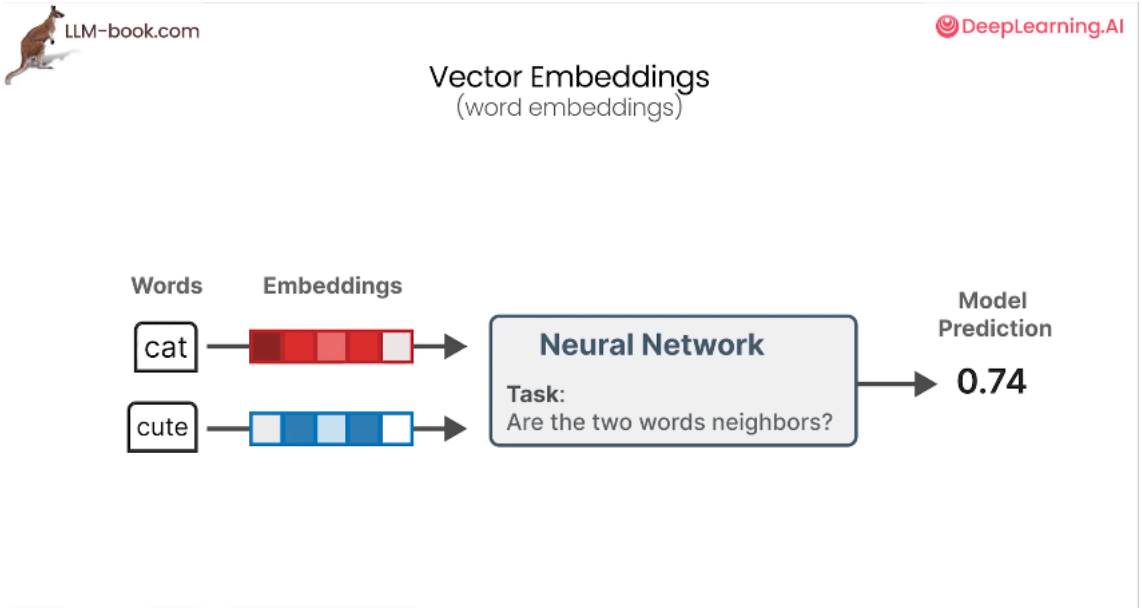
Word Embeddings

Bag of words has a flow that it consider the language nothing but a literal bag of words.

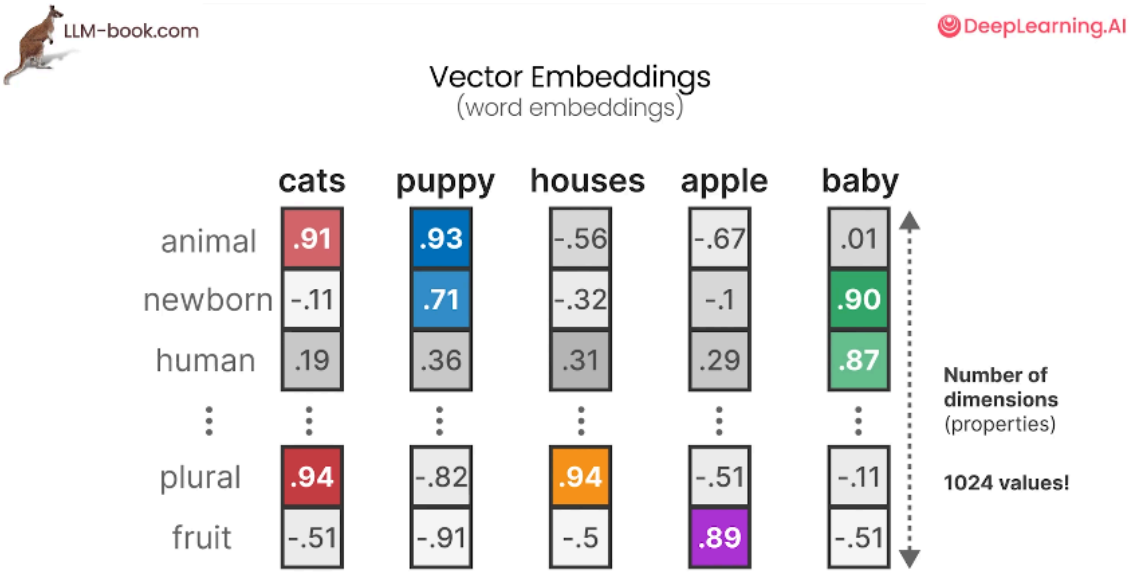
- Bag of word does not consider semantic nature of text.

Word2Vec: Word2Vec could captures the meaning of words in vector embedding through neural network. It learns semantic nature by training vast amount of data.





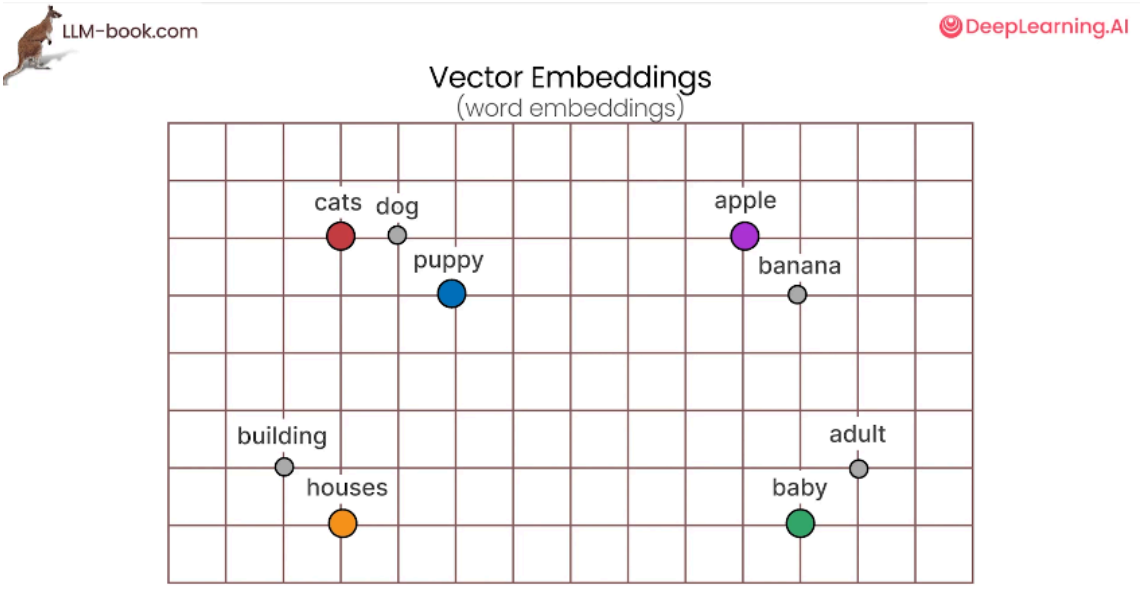
How exactly the embedding values learned. It learned via neural network. embedding values are set between -1 and 1



for example: for cat values for newborn, human, fruit are very low on the other hand values for animal and plural is high as this embedding captures the meaning of the word.

number of dimension in embedding can be very large and it is based on number of properties included. We do not need to know what are the values of embedding, what are the properties exactly represent as these are learned through complex mathematical calculations.

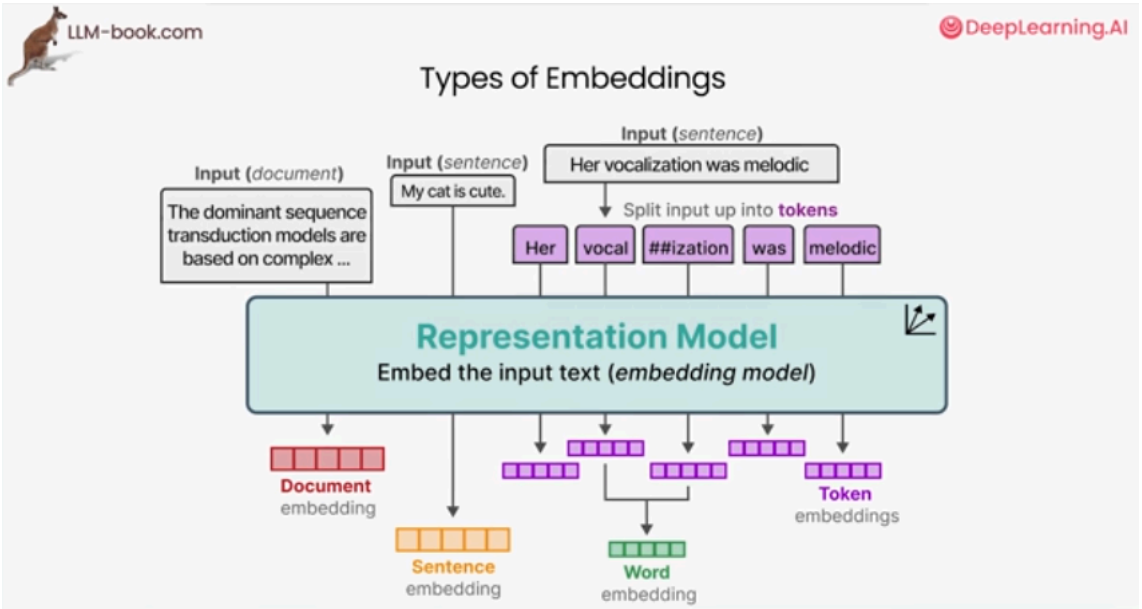
Each properties allow us to compare embedding one another therefore word itself. With this word with similar meaning group together whereas different word are further apart.



How words are similar or dissimilar are depends on its training.

Types of embedding:

1. Word embedding
2. Sentence embedding
3. Document embedding



Encoding and Decoding Context wit Attention

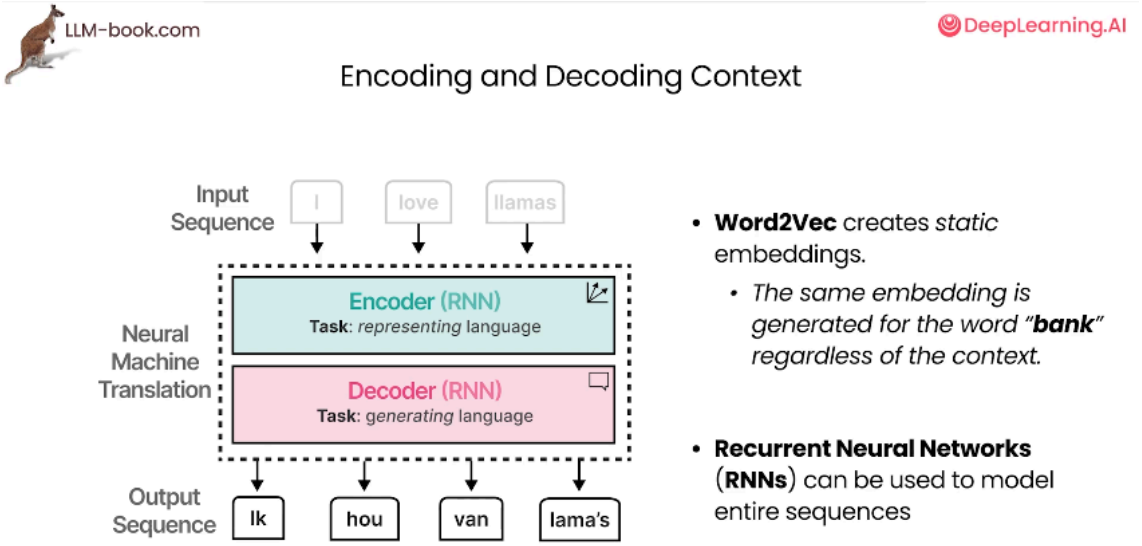
Previously we learn how to encode embeddings but with limited contextual capabilities, here we explore how to encode and decode context with attention.

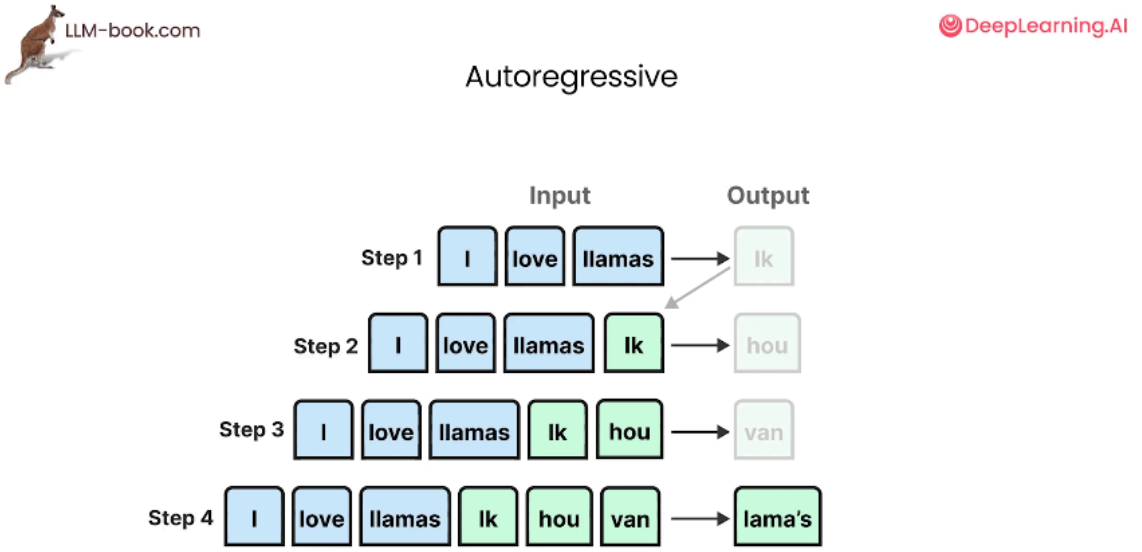
What exactly it mean?

→ Let's take word bank. In word embedding 'bank' get embedding. This embedding is same in both 'Bank has lot of money' and 'That river bank is very beautiful'. But we can see that this both

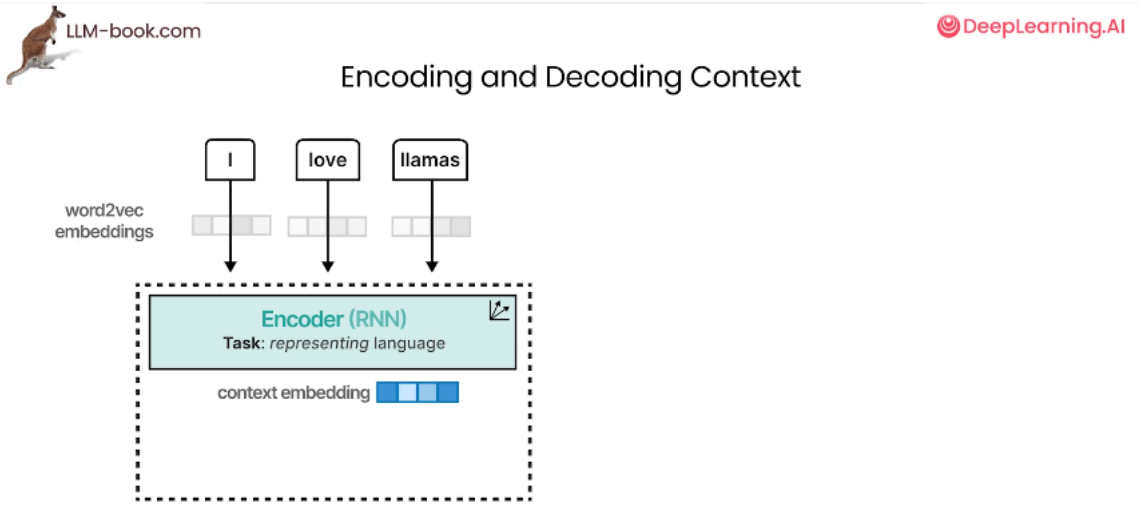
sentence carry different meaning for word ‘bank’. This meaning is not captured in word embedding. This happen because the embedding is created without the context.

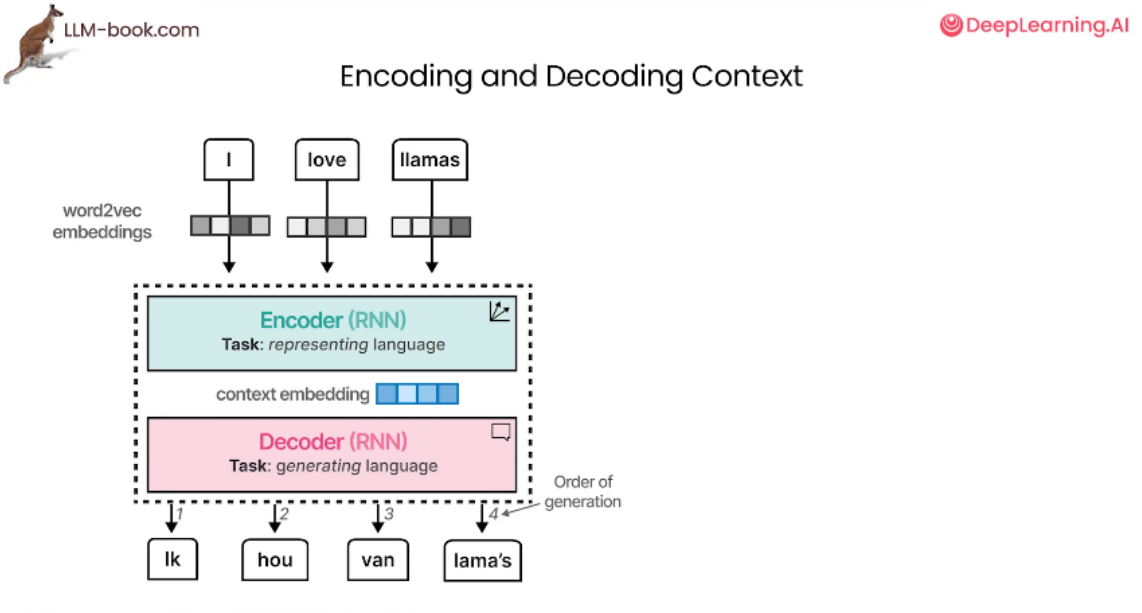
Capturing context is important to perform language task such as translation. To achieve this RNN models are used.





Encoder passes the embeddings of input in one go and takes into account the context of embedding and then generate context embedding.

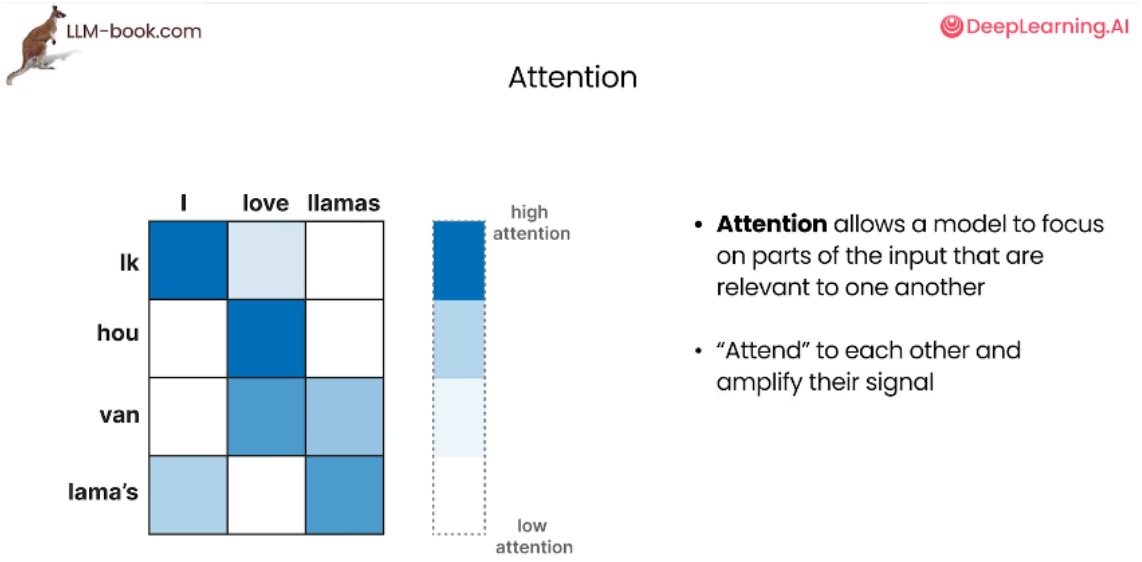


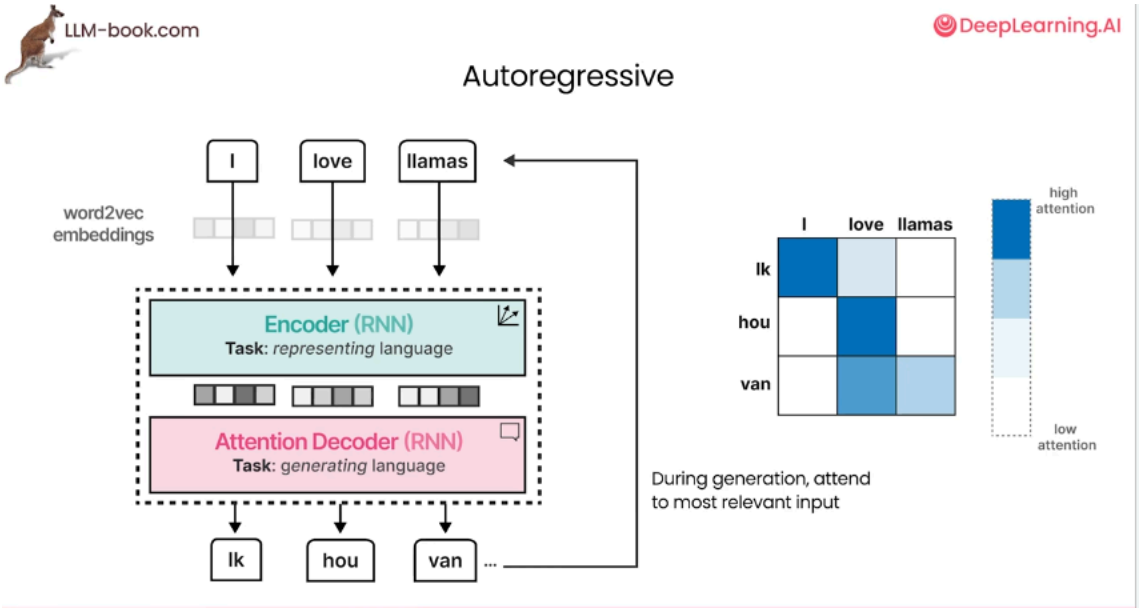


In above image we can see that with single context embedding translation is made one-by-one as shown in autoregressive image. But this fails to capture context of long sentence.

Attention:

Attention is introduced to capture the relevance of word with each other.

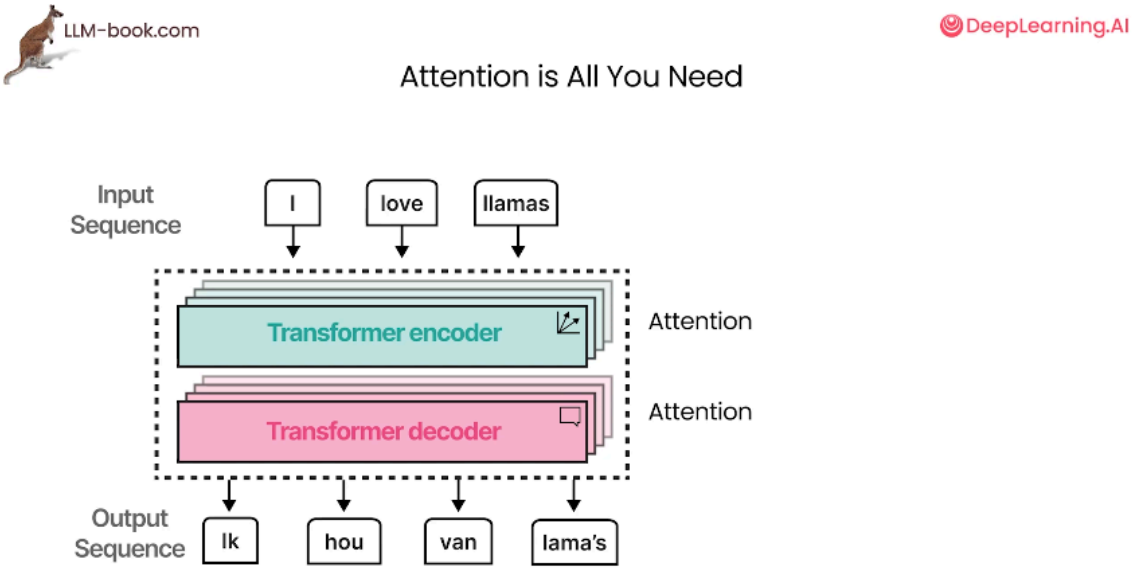




In this RNN generate sequence to each input word related to output. (Unlike single context embedding). Instead of passing only single context to decoder the hidden state for all input encoder are passed. State for word is a internal factor for a hidden layer of a RNN that contain information from previous ones. The decoder uses the attention Mechanism to look at the entire sequence. Due to this attention mechanism the output must be much better instead of using smaller context embedding. After generating output the attention mechanism highlight words that are more relevant in particular example.

Transformers

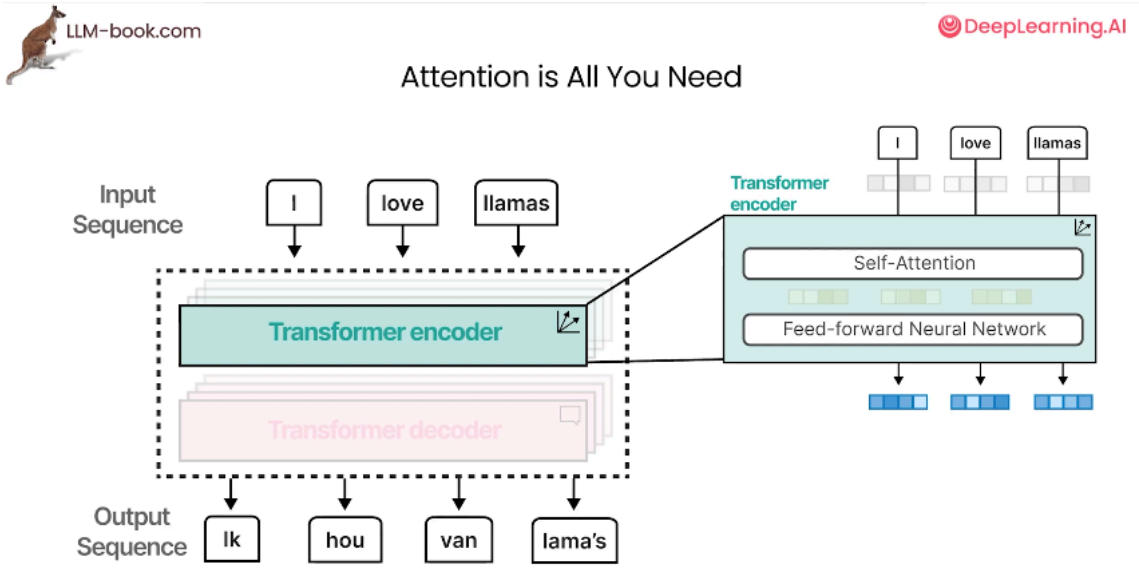
- A new architecture solely based on attention and without the RNN.
- Could be trained in parallel which speeds up calculation significantly

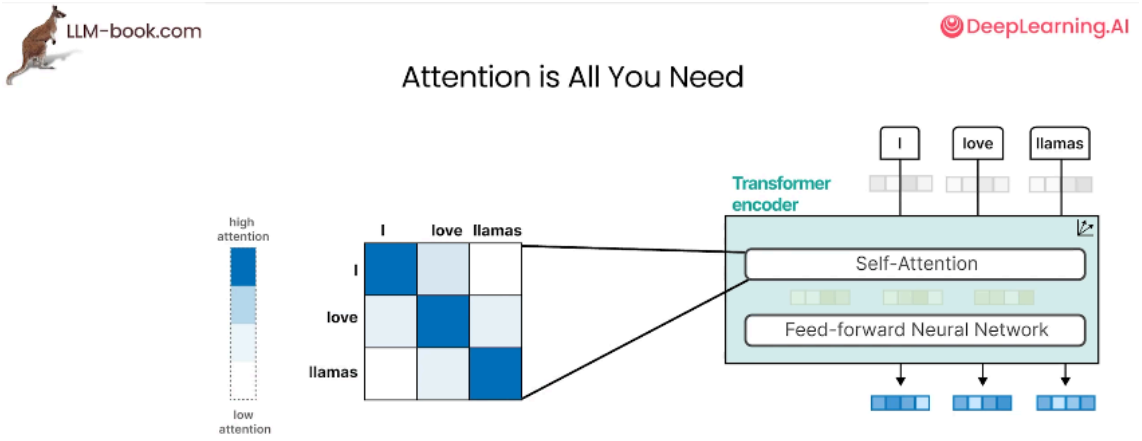


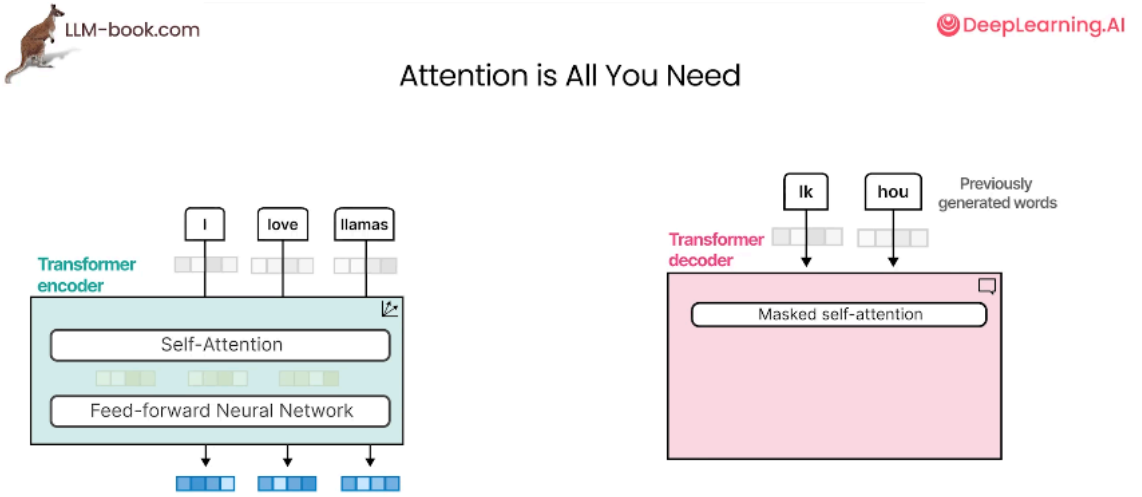
We stack the transformer encoder and decoder block. By staking this block we amplify the strength of encoder and decoder.

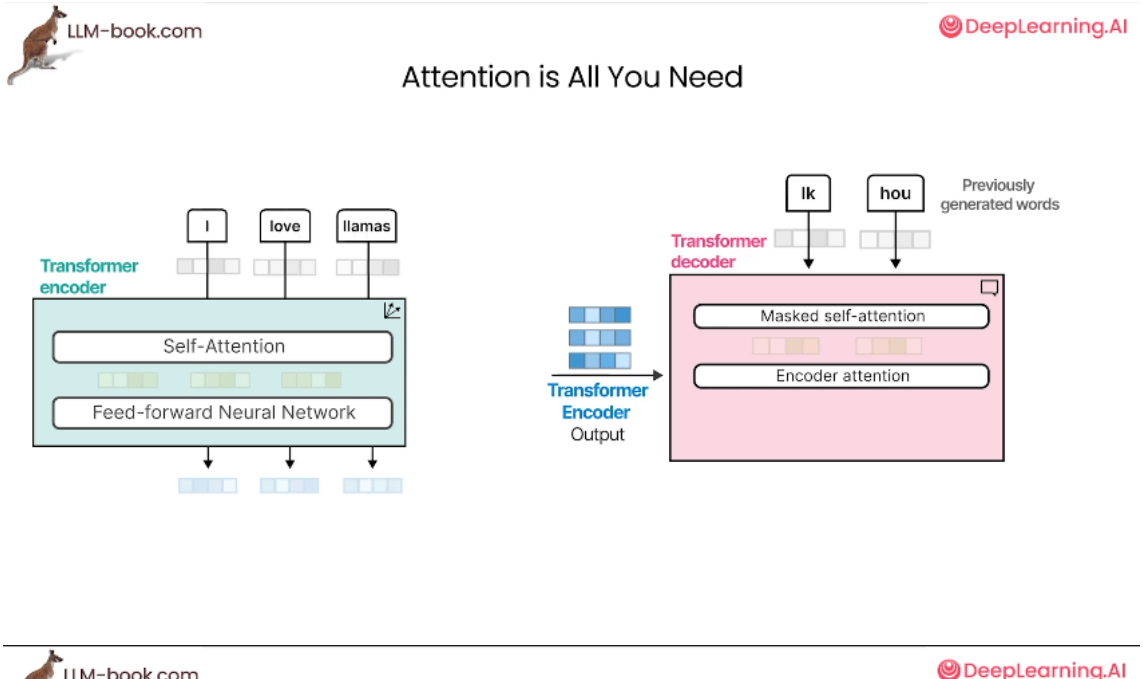
Encoder:

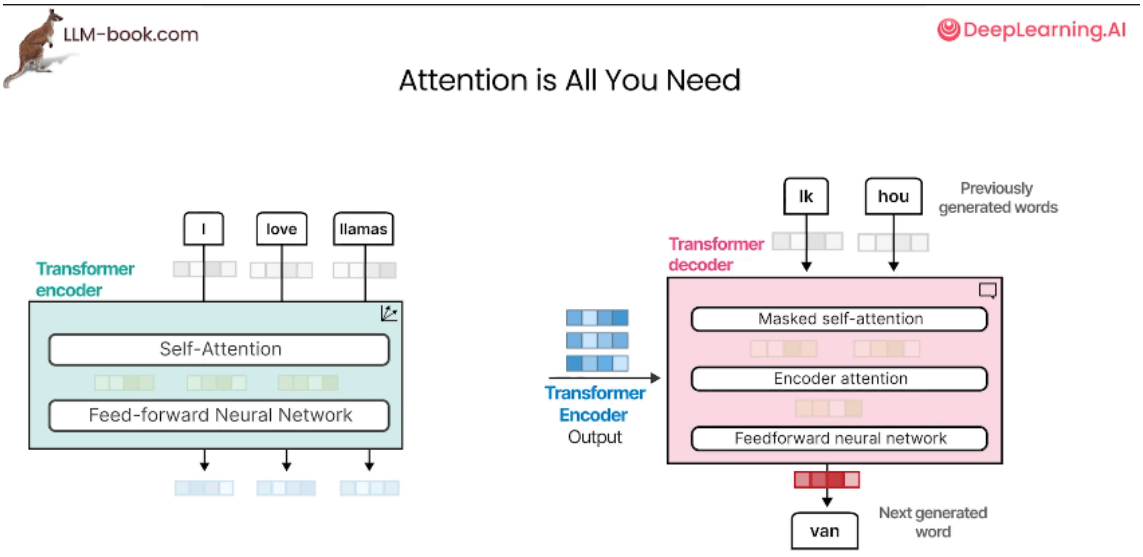
Instead of word2vec embedding we start with values. then self attention which attention focus on only the inputs process the embedding and update them. These update embedding contain more contextualize information as a result of the intention mechanism. They are passed with feed-forward neural network. to finally create contextualize token/word embeddings.




LLM-book.com
DeepLearning.AI
Attention is All You Need
high attention
low attention
I
love
llamas
Transformer encoder
Self-Attention
Feed-forward Neural Network

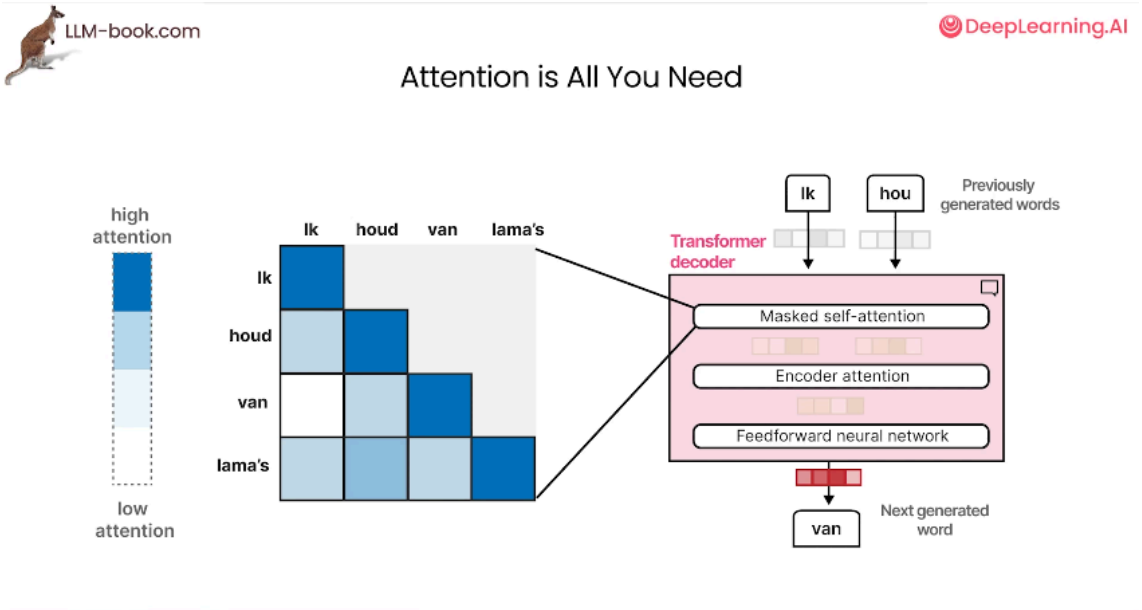

LLM-book.com
DeepLearning.AI
Attention is All You Need
Transformer encoder
Self-Attention
Feed-forward Neural Network
Transformer decoder
Masked self-attention
Previously generated words





Masked self attention

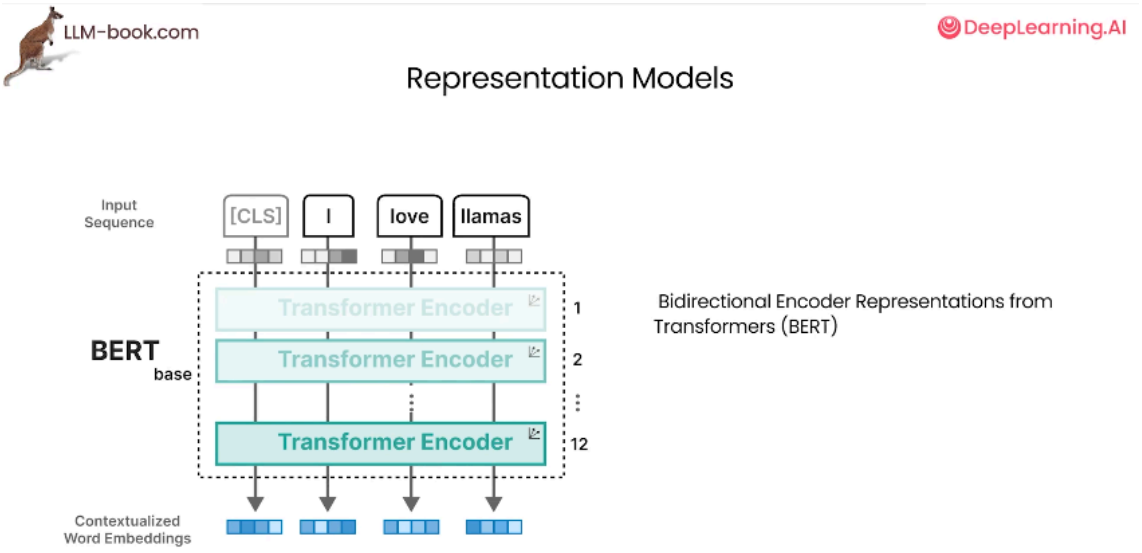
Removed attention values from upper diagonal therefore it masks future position so that any given token can attend to token that came before it. It helps leaking information when generating the output.



Bidirectional Encoder Representation Transformer (BERT)

This is encoder only architecture that focuses on representing language and generating contextual word embedding.

Encoding block are same as we saw before → self attention flows by NN → The inputs contain an additional token (like [CLS]/classification token in following example)

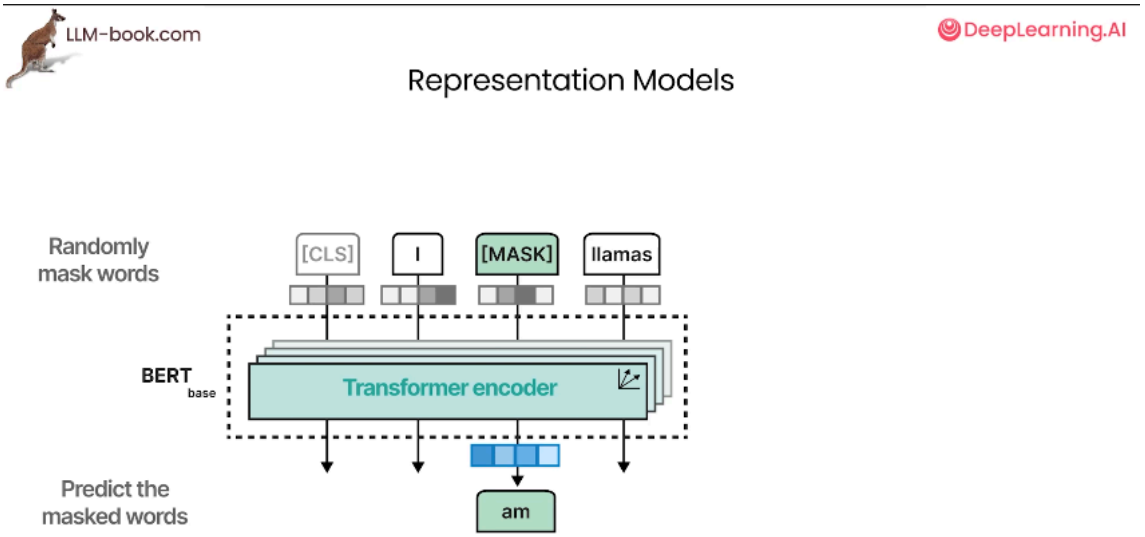


To train BERT like model we can use technique like mask kanguage modelling

Mask Language Modelling Technique

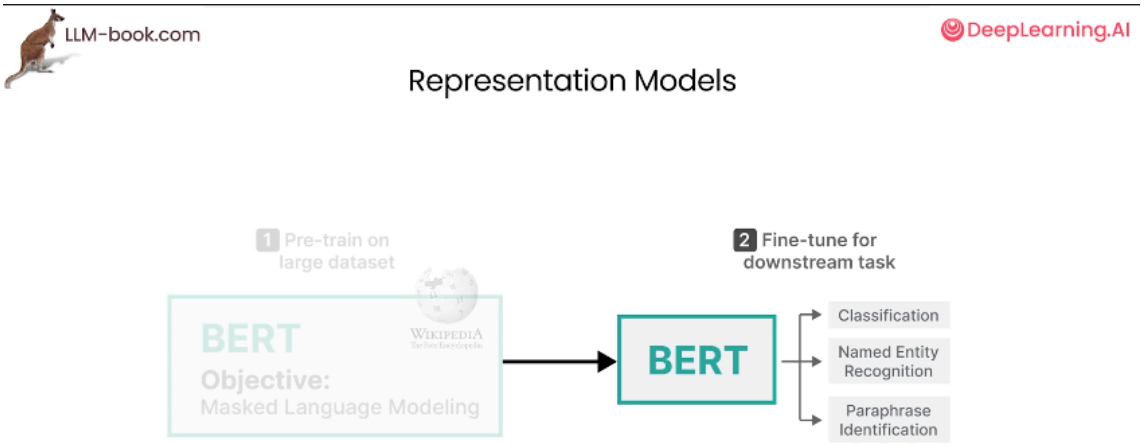
randomly mask words from input sequence → let the model predict it.

By doing so model learns to represent the language as it attempts to diconstruct this masked words.



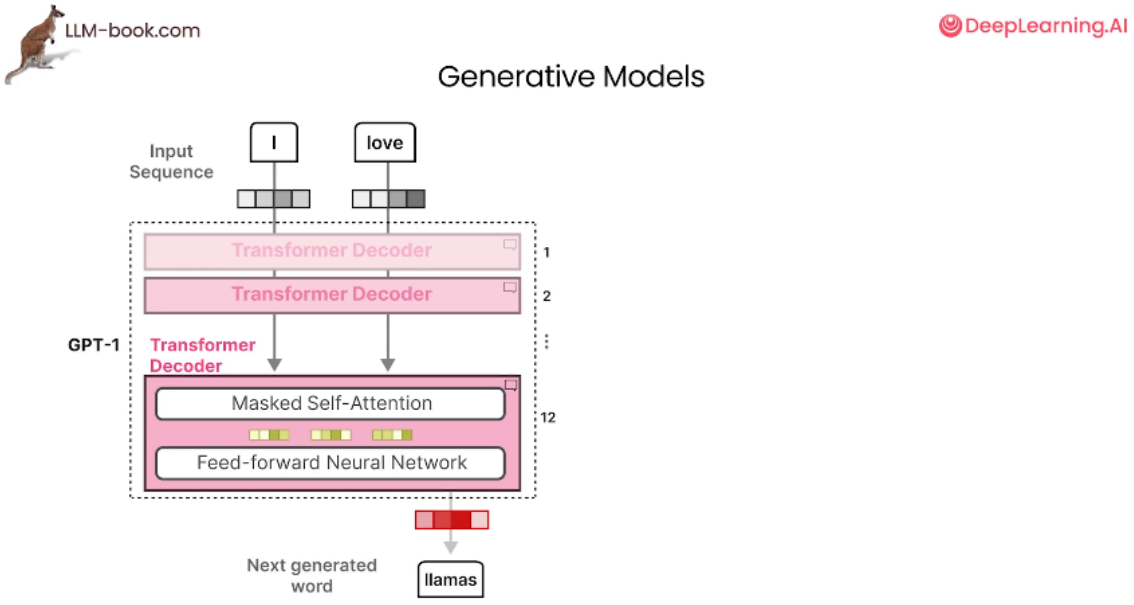
Training:

- Representation models: 2 step approach [we saw above]

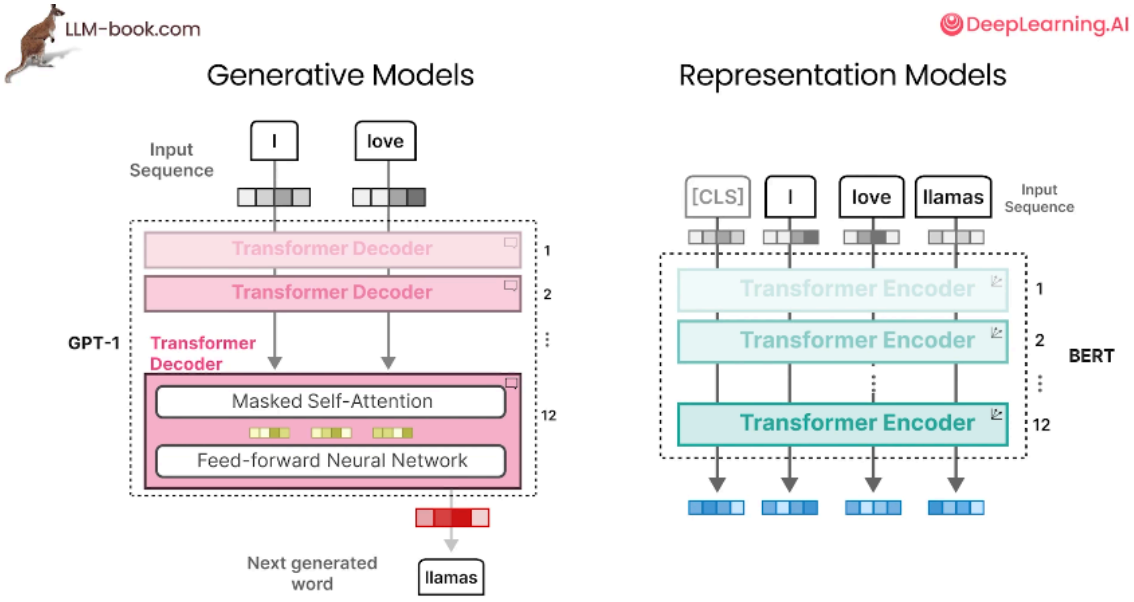


- Generative Models:

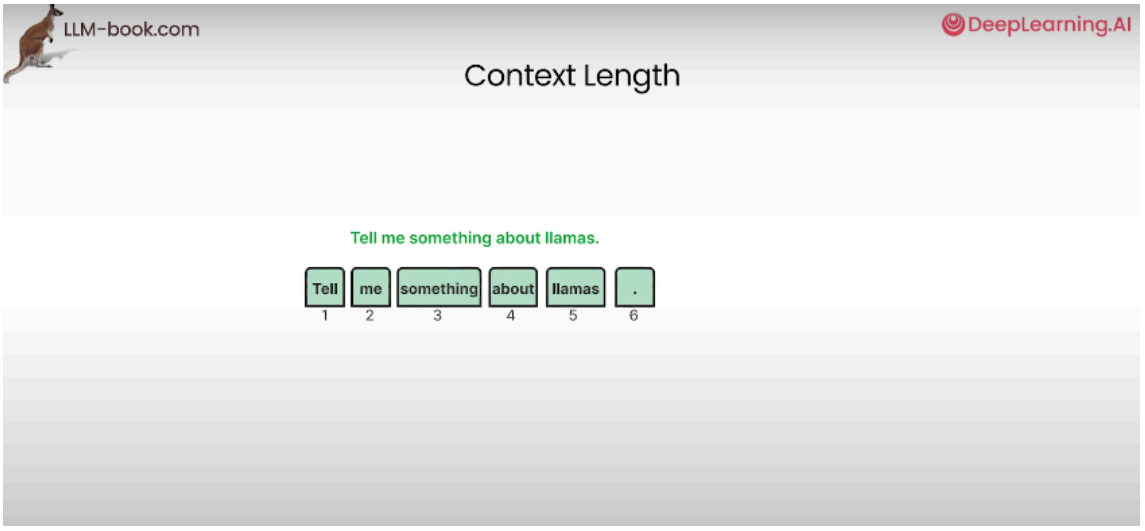
It uses another approach by generating randomly initialized embeddings. Then this pass to decoder. [Note: Generative models are only stacked with decoder]. Decoder block again use mask self attention. and finally the ext word is generated



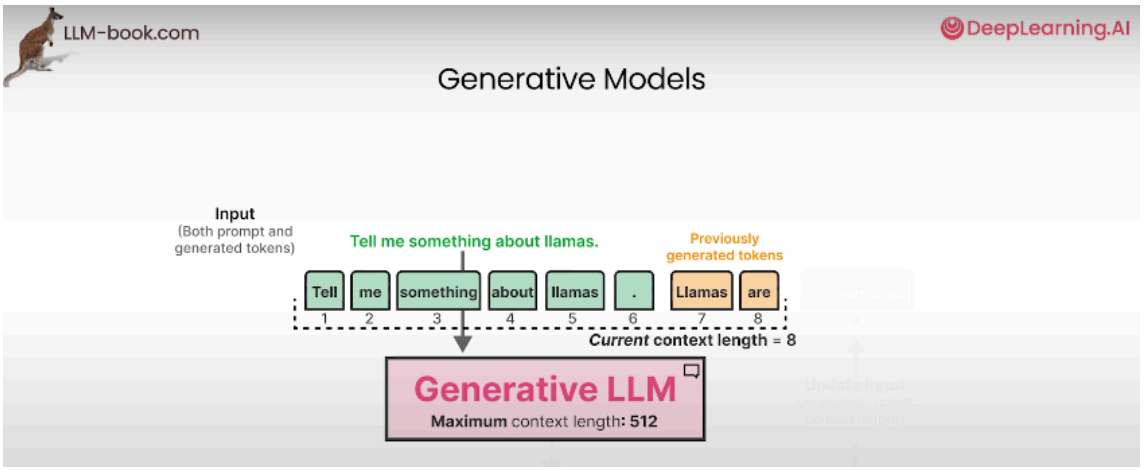
Most often used flavours



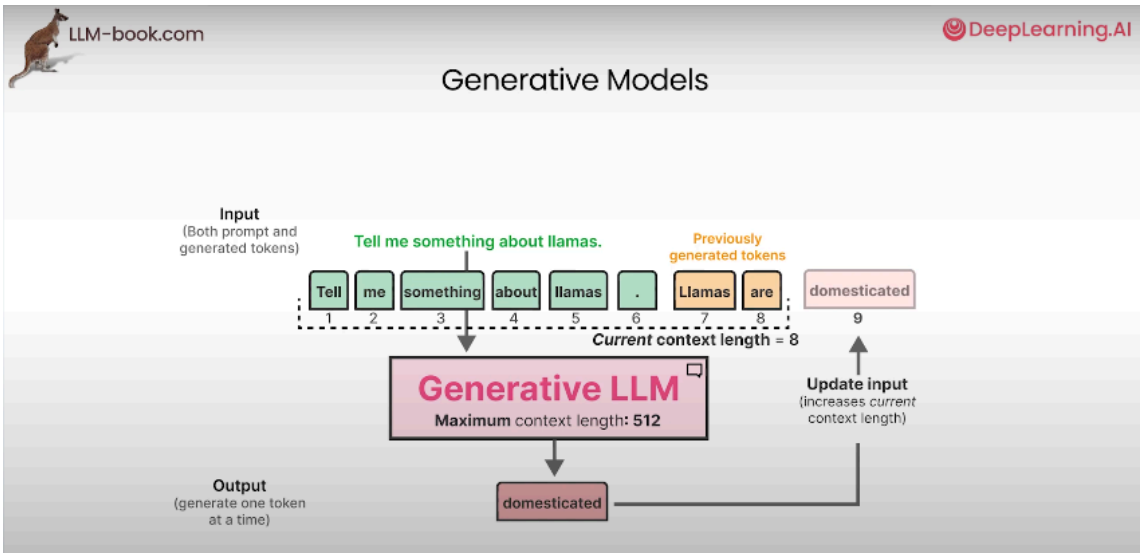
Both of this have something in common that is context length.



If maximum context length is 512 then model can process maximum 512 token at a given/same time.

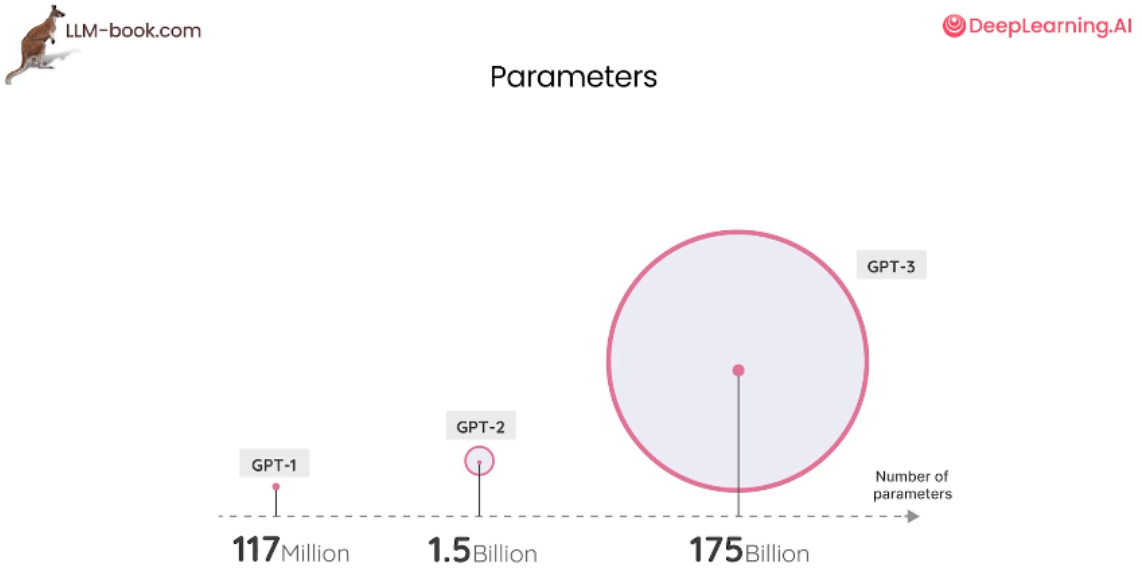


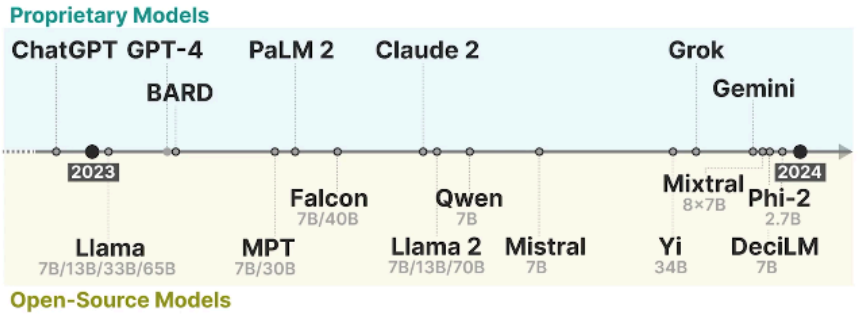
Important point is it also include the tokens that are being generated as they update the current context.



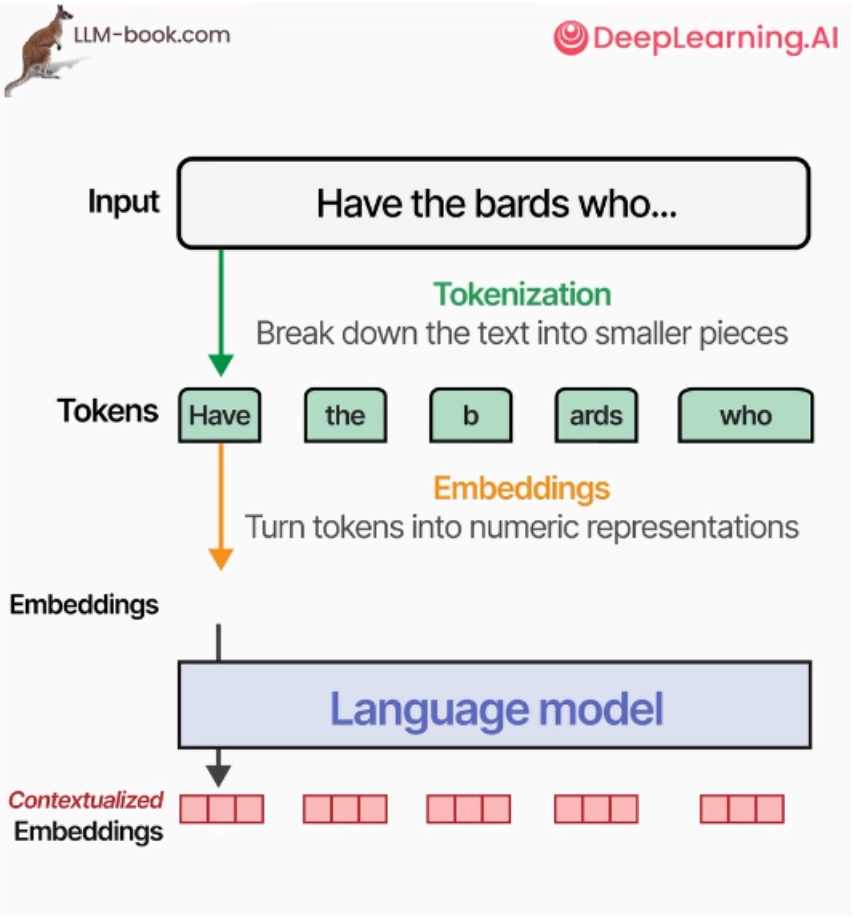
Parameters

As number of parameters gow so does its capabilities

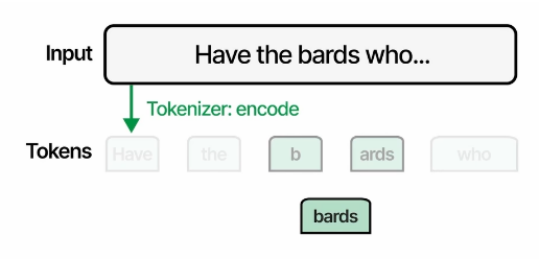


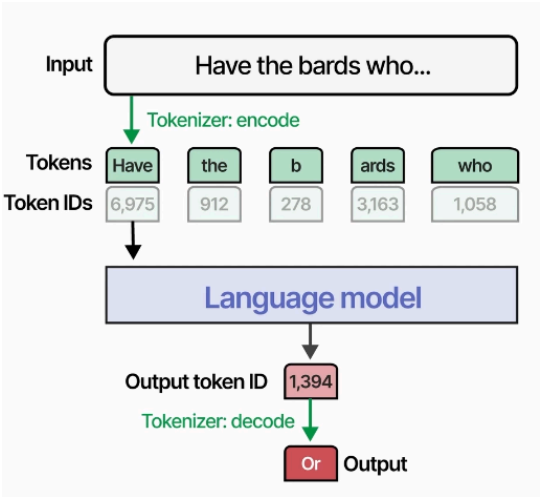


Tokenization

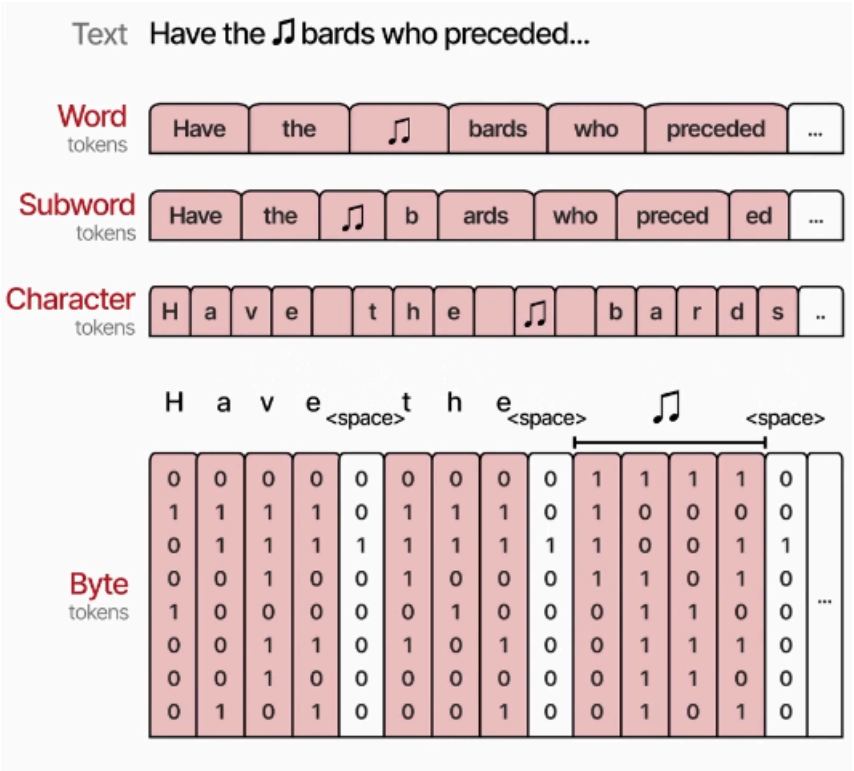


tokens can be words or part of words



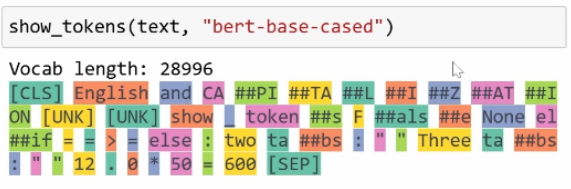


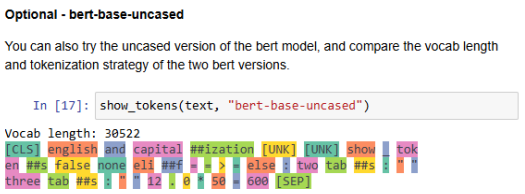
Types of tokens:



Most model prefer to use subword token. As its vocabulary is flexible and allows to represent the most word. Word token might not fit in token size so it should be splitted which is done in subword token.

Hands on code:





We can see that this doesn't have tokens like [CLS] [UNK] or special character. As it is meant for generative representation.

The larger the vocabulary length is less token require but more embedding needs to be calculated. [This is trade-off]

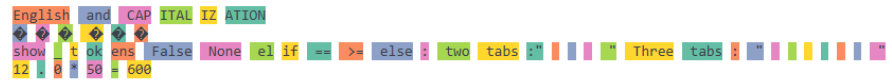
Make sure to consider the following features when you're doing your comparison:

- Vocabulary length

- Special tokens
- Tokenization of the tabs, special characters and special keywords

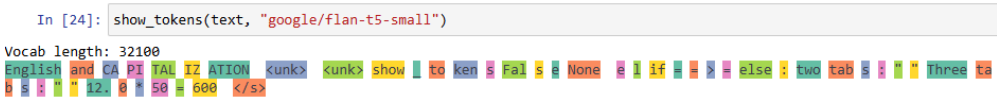
gpt2

```
In [23]: show_tokens(text, "gpt2")
Vocab length: 50257
```



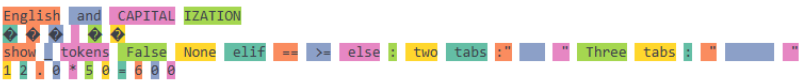
Flan-T5-small

```
In [24]: show_tokens(text, "google/flan-t5-small")
Vocab length: 32100
```



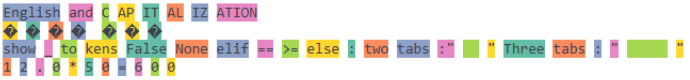
Starcoder 2 - 15B

```
In [26]: show_tokens(text, "bigcode/starcoder2-15b")
Vocab length: 49152
```



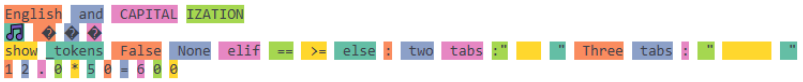
Phi-3

```
In [28]: show_tokens(text, "microsoft/Phi-3-mini-4k-instruct")
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
Vocab length: 32011
```

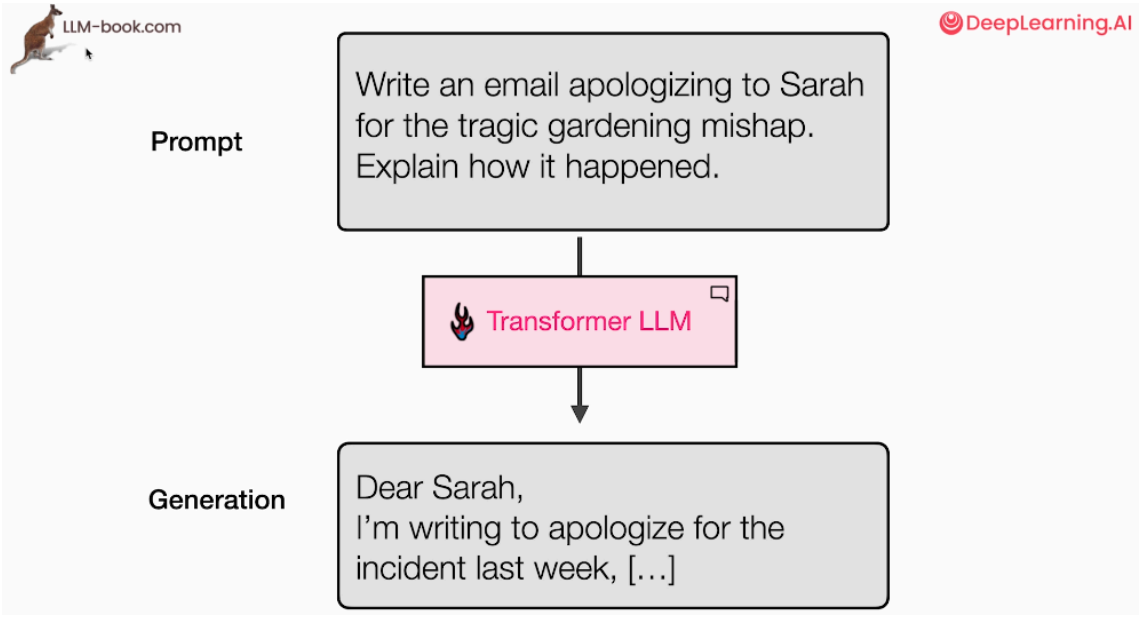


Qwen2 - Vision-Language Model

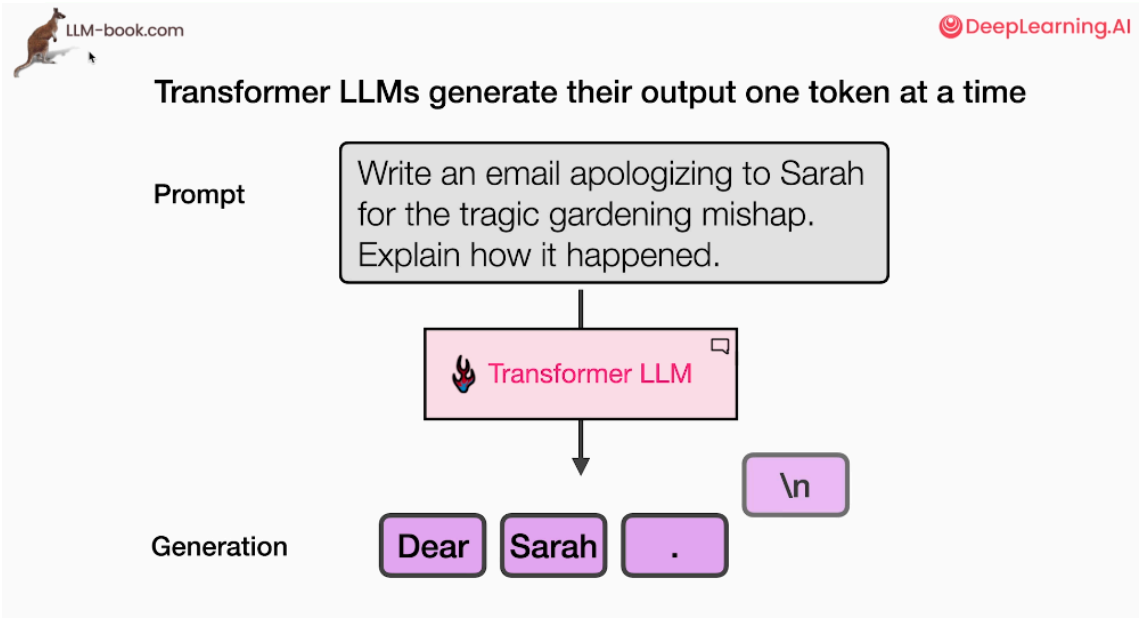
```
In [30]: show_tokens(text, "Qwen/Qwen2-VL-7B-Instruct")
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
Vocab length: 151657
```



Transformer Architecture Overview



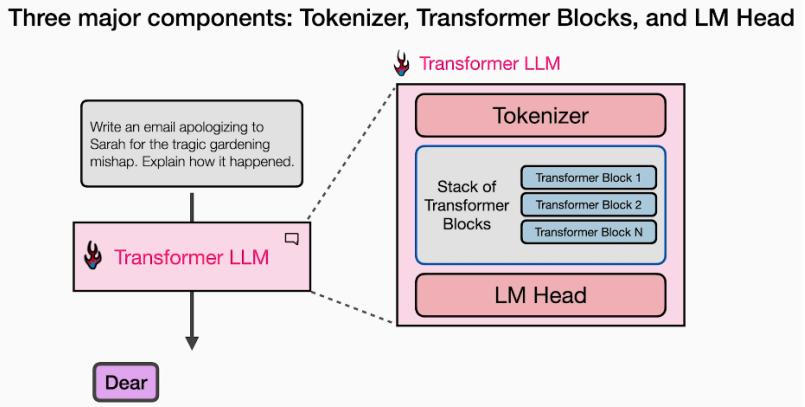
Transformer generates the output one by one



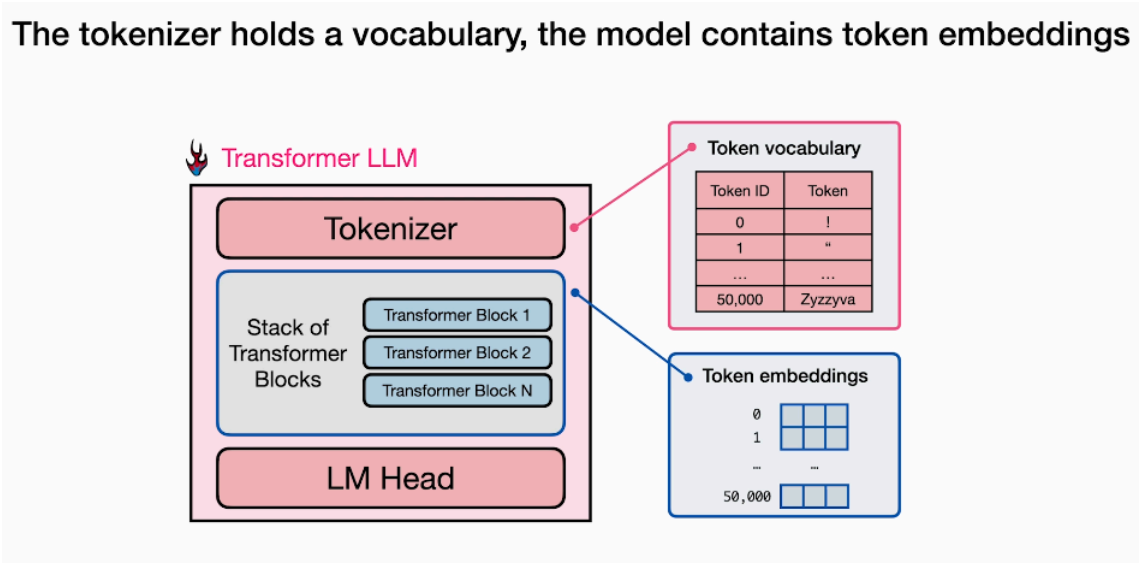
Three major component of transformer:

1. Tokenizer
2. Transformer blocks

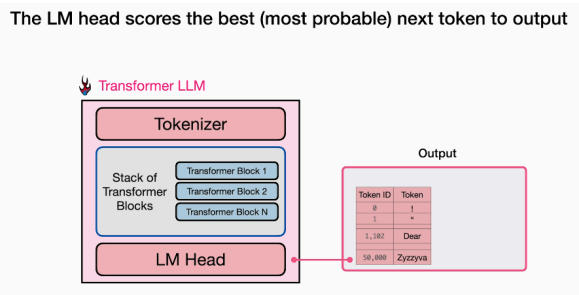
   (NN is here, major computation present)
3. LM Head

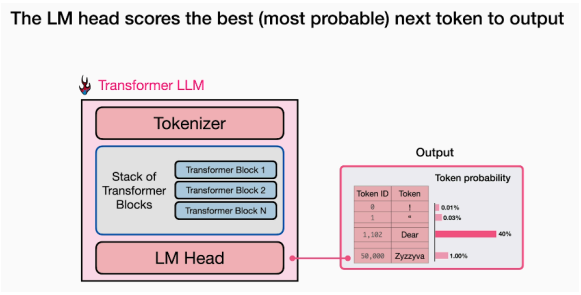


If we have 50k token in vocabulary then transformer contains 50k embeddings

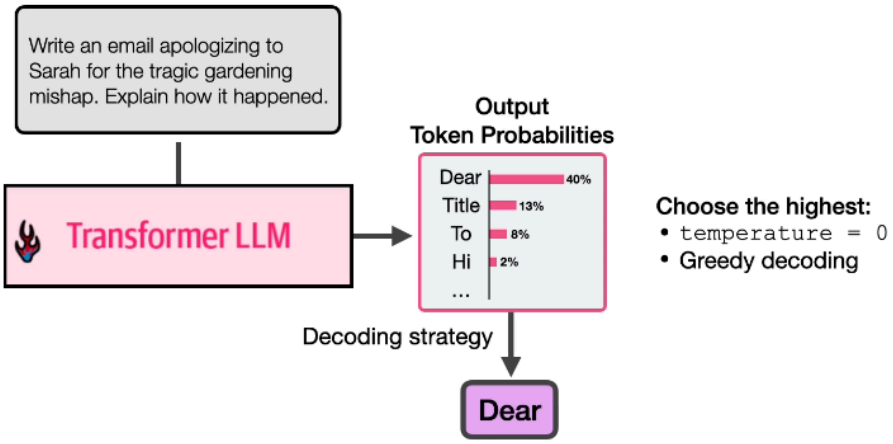


The result is token probability for each of token it has.





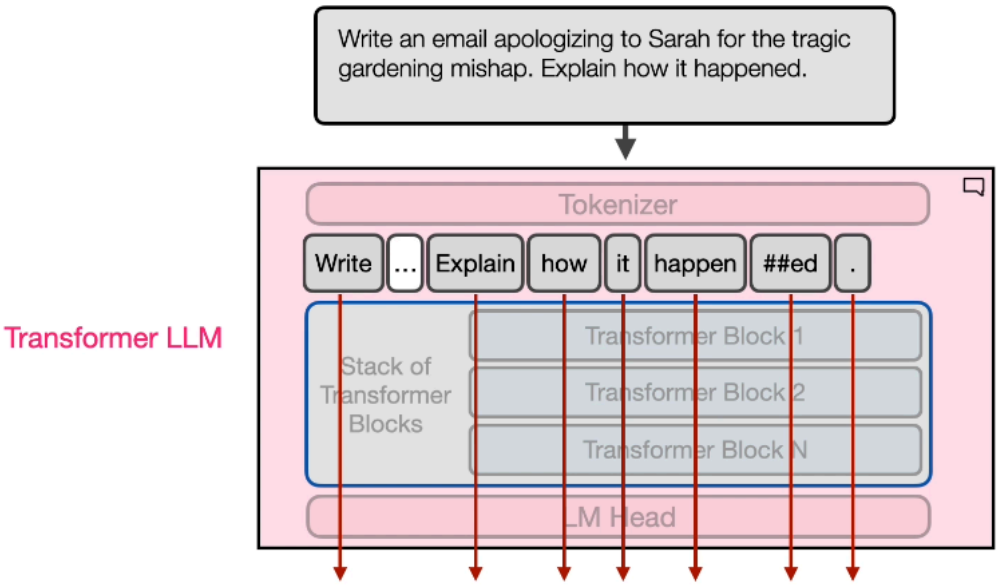
Multiple decoding strategies exist to choose the best output token



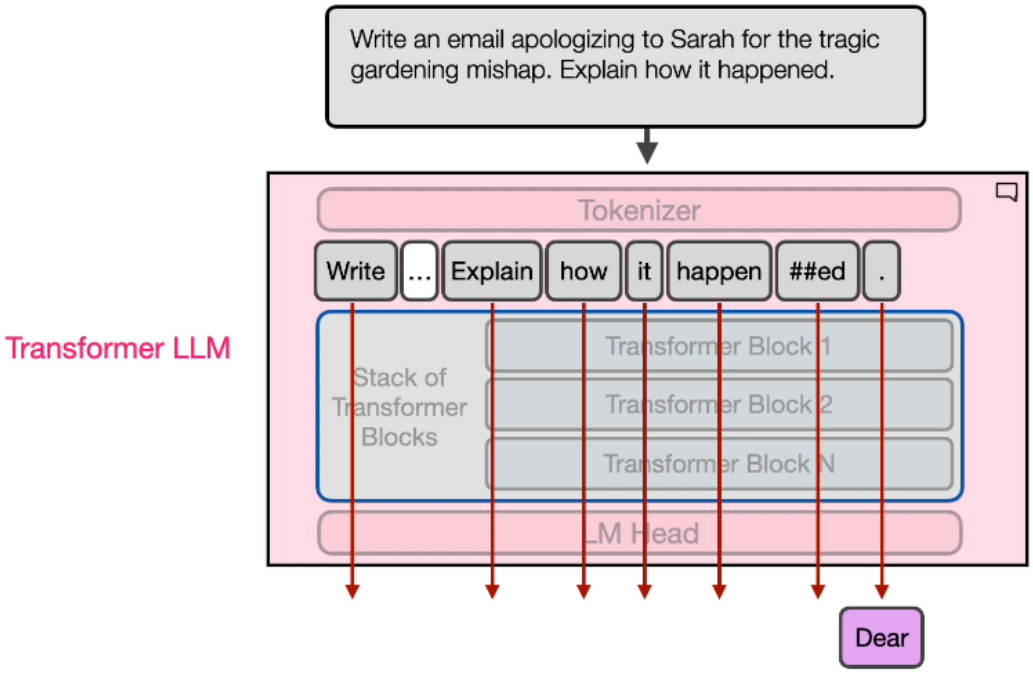
There are multiple decoding strategies. The simpler one is pick a highest probability that setting temperature to 0. This is also called greedy decoding.

Other are like top_p and randomness (temperature>0)

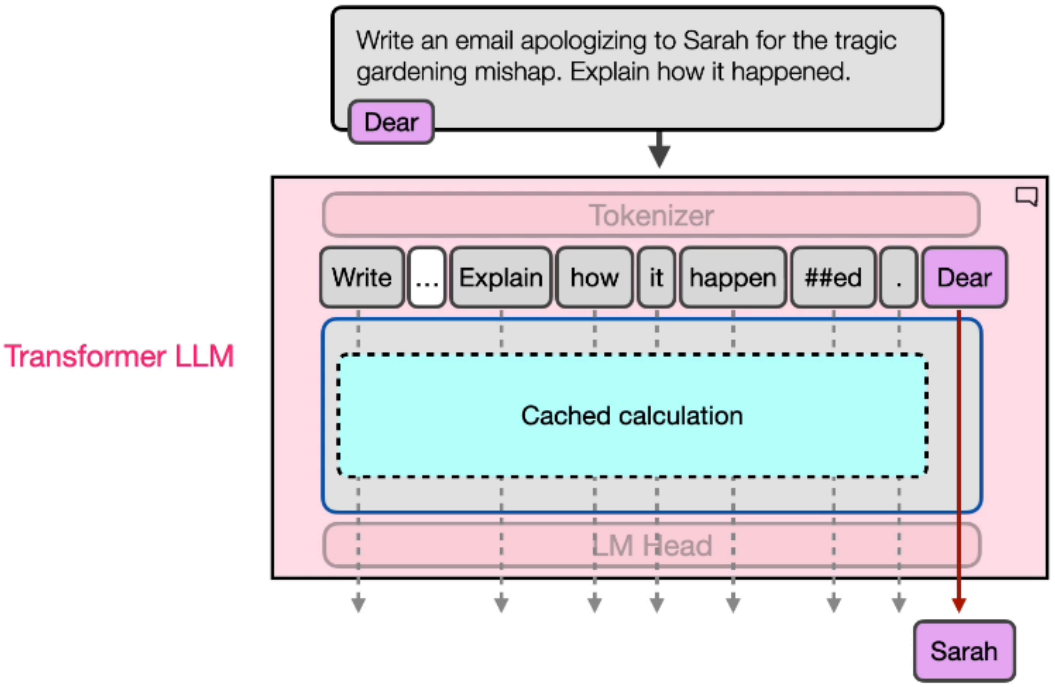
Another idea in transformer is they process all the inputs in parallel and that parallelization make it time efficient. The way to envision this is to think of multiple track flowing through this stack of transformer blocks and the number of tracks here is context size of model. If model has context size of 16k tokens then it can process 16k tokens at the same time.



The generated token in decoder LLM transformer is the output of the final token in the model

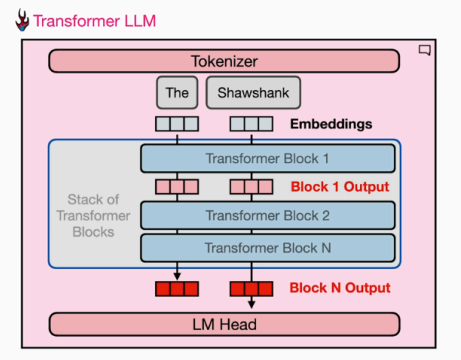


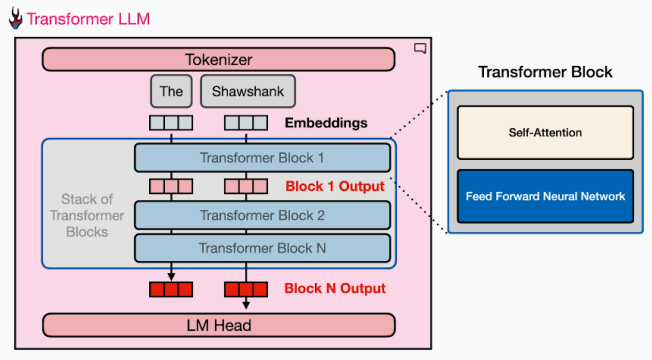
Next time we add he generated word to the input so you can think it as a loop. so we can cache this calculations. You can see n below image that only one red arrow is highlighted. This is called KV caching. K means Keys and V means Values.



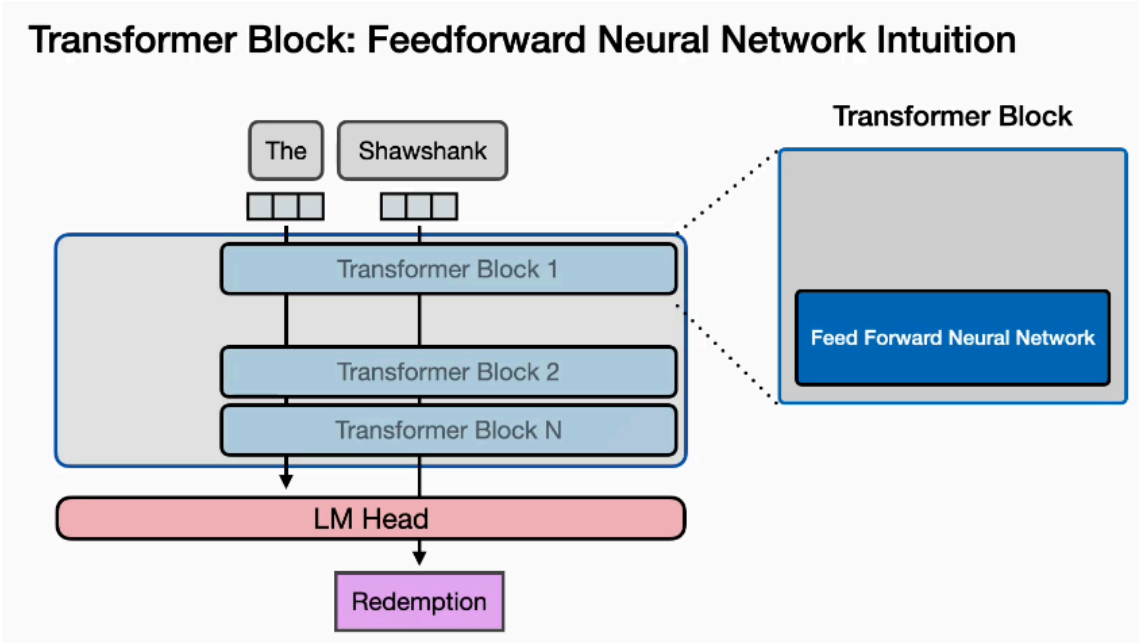
In terms of efficiency it take the time to generate the first token (how long the model takes to process all of these) and then generating every other token is slightly different.

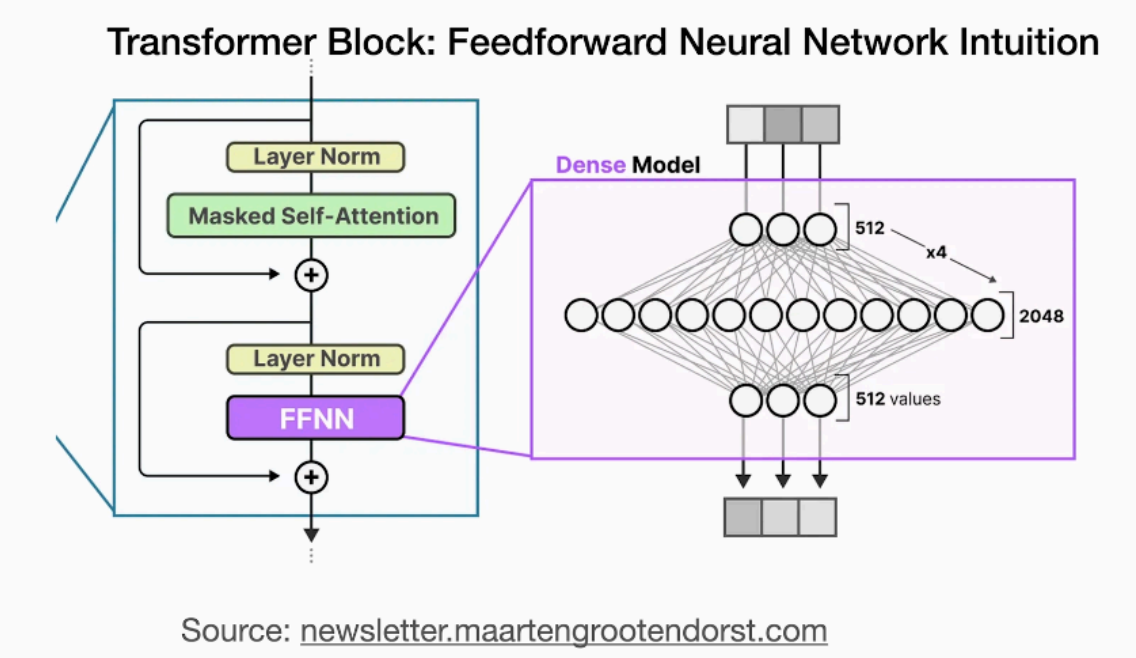
Transformer Block

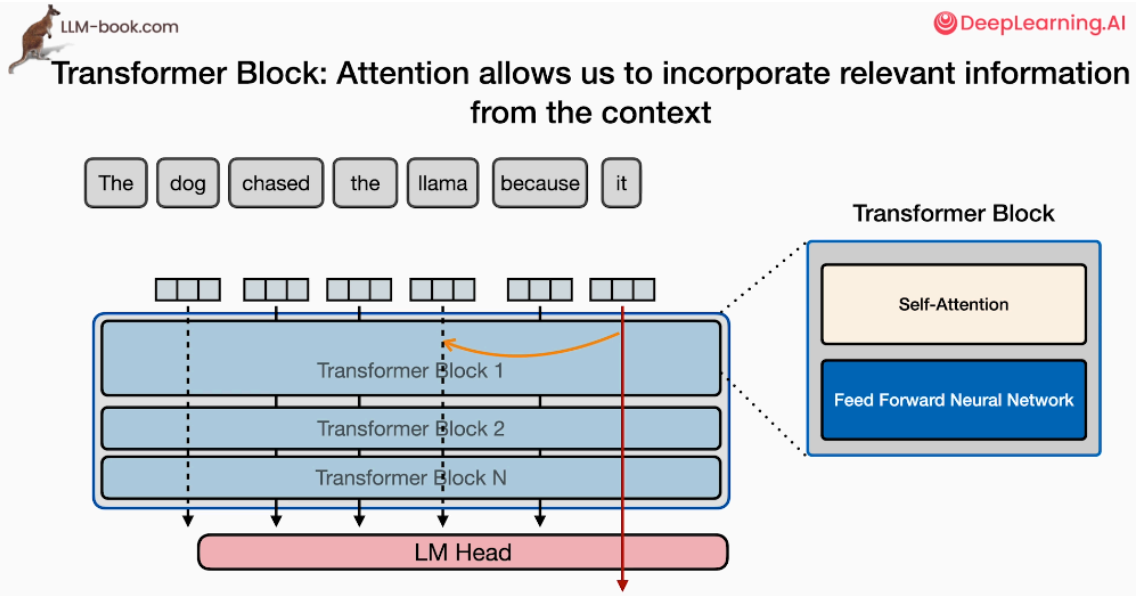




With only Feed-forward NN model able to generate the next word after The Shawshank. We can assume FF NN as storage and statistics of next word comes in after the input token.





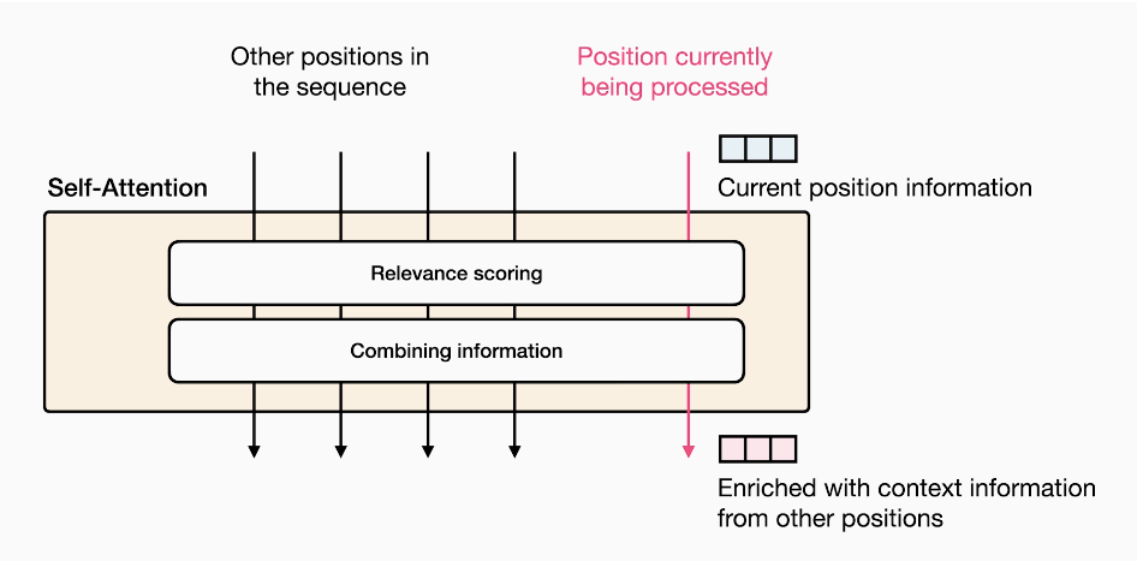


Model is trying to figure out what that ‘it’ refers to

Self Attention to capture this context

Now what self attention does:

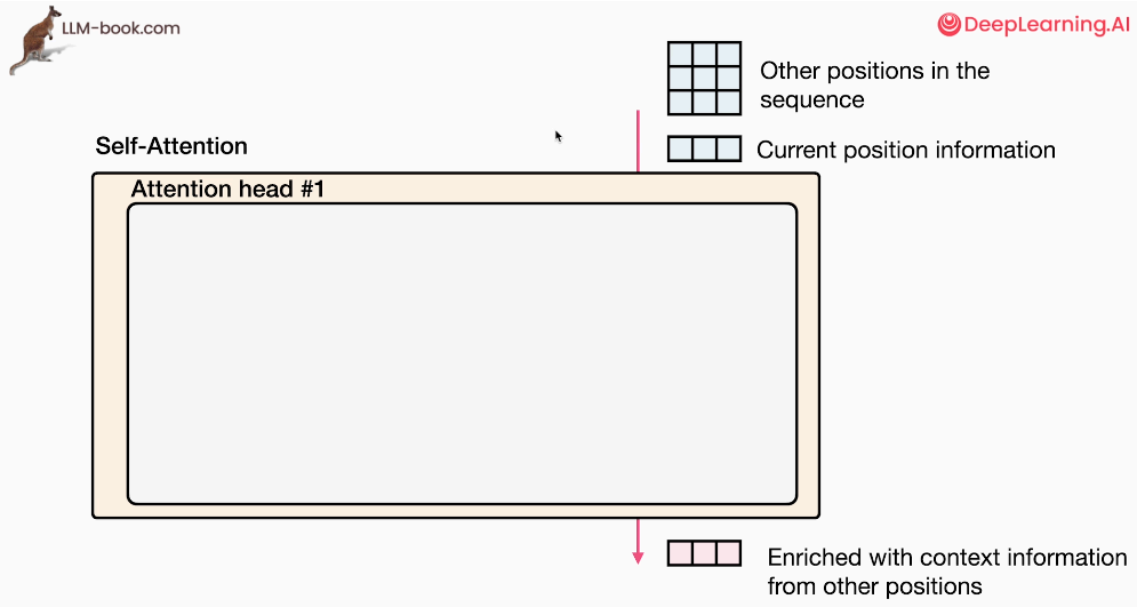
- Relevance scoring
- after scoring combing the information



Self-Attention

Two steps are Relevance scoring and Combing information

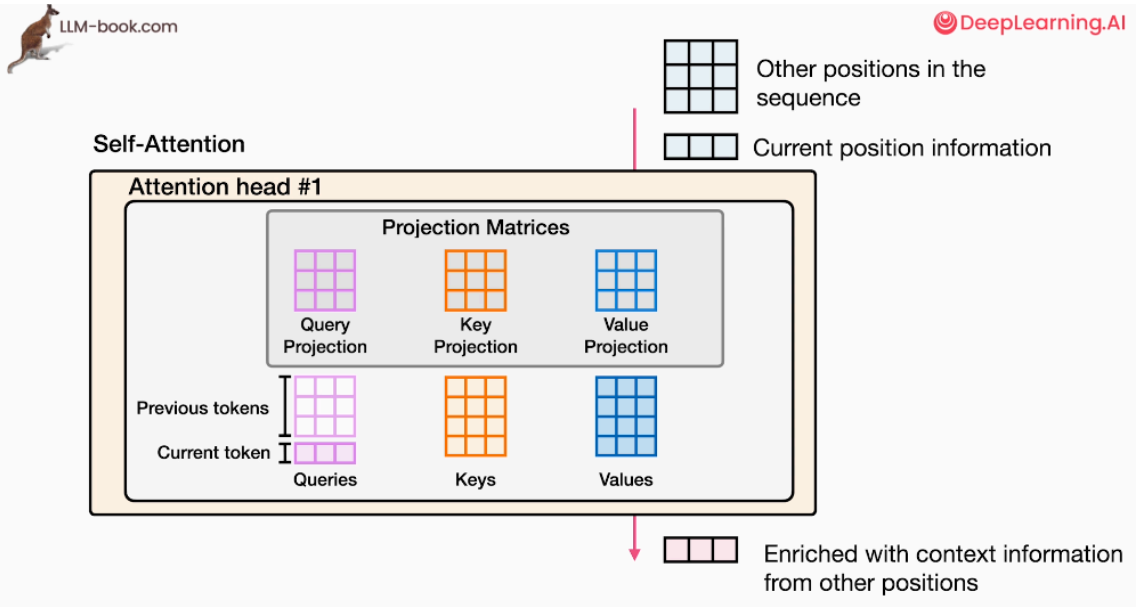
Self-Attention happens in attention head. we have current position sequence and also other positions in the sequence. The output will be the embedding enriched with context information from other positions. (These all are vector representation of tokens)

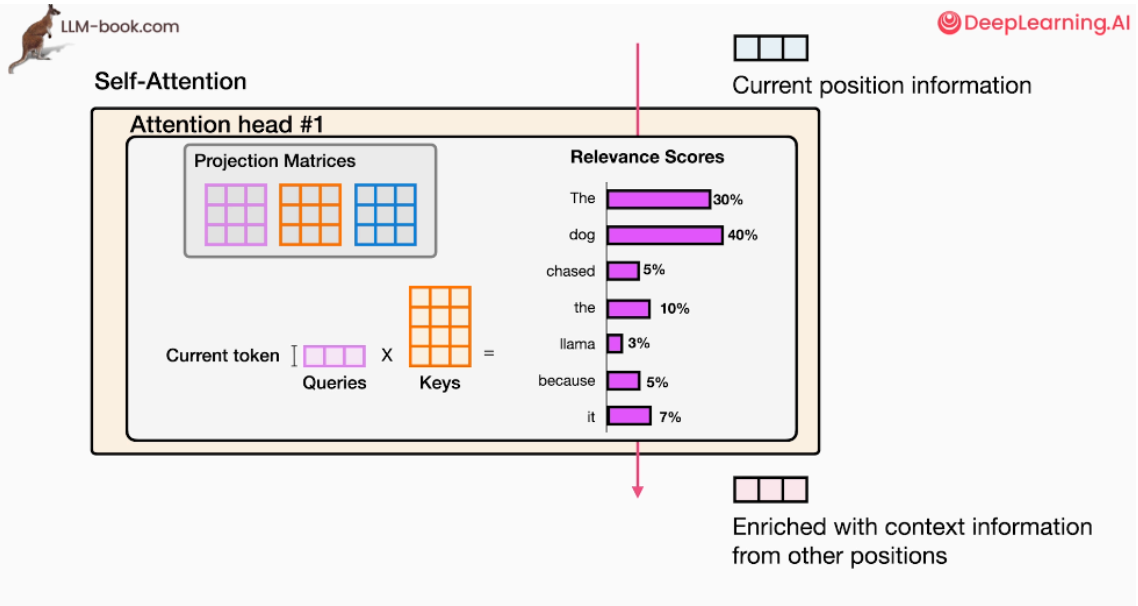


Self-Attention is conducted with three projection matrix:

1. Query Projection Matrix
2. Key Projection Matrix
3. Value Projection Matrix

This weight matrices are used to calculate query, key and value matrices.

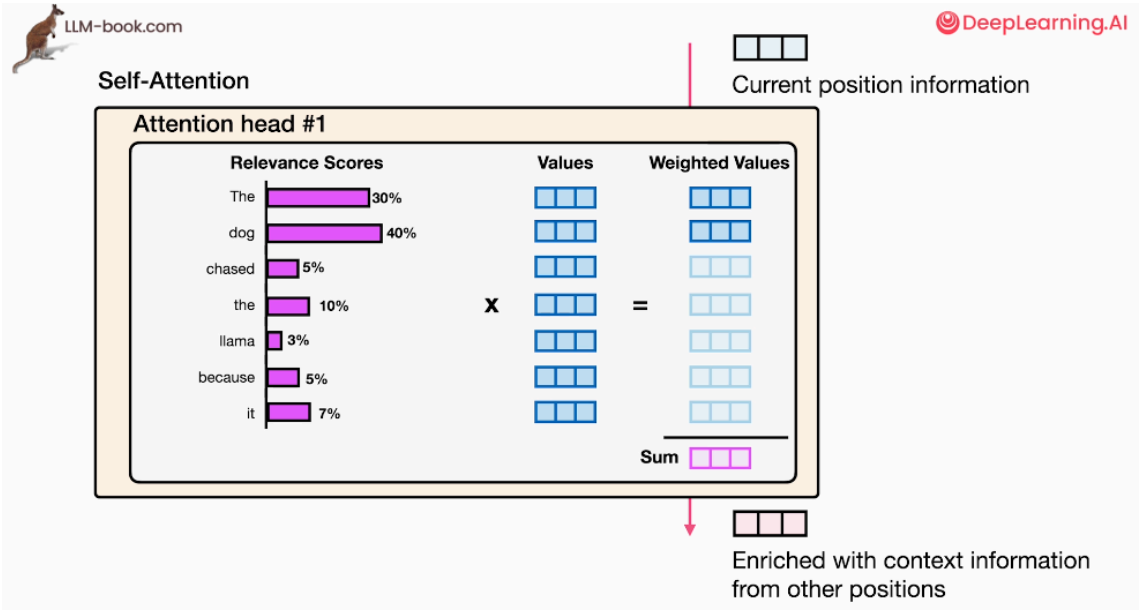




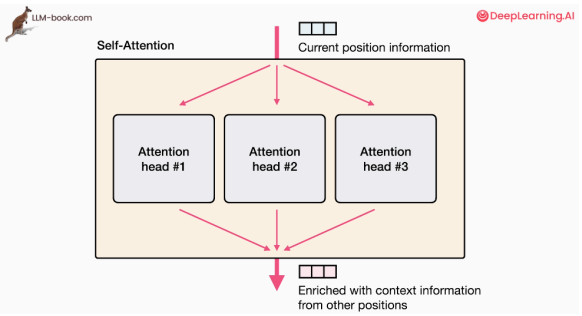
Now we got relevance score we will start with second step

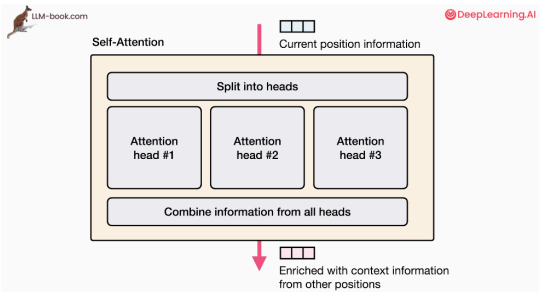
Coming information:

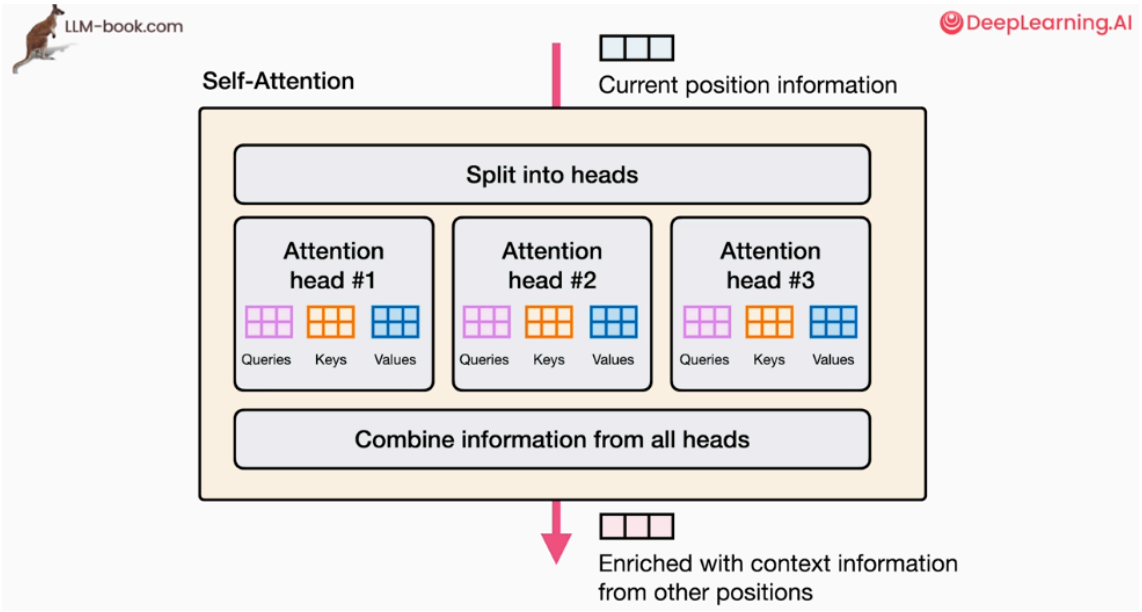
we combine relevance score with the values and it will give us weighted values. In which dog will have more value. Once we have weighted value we just sum it and it will be the output value.



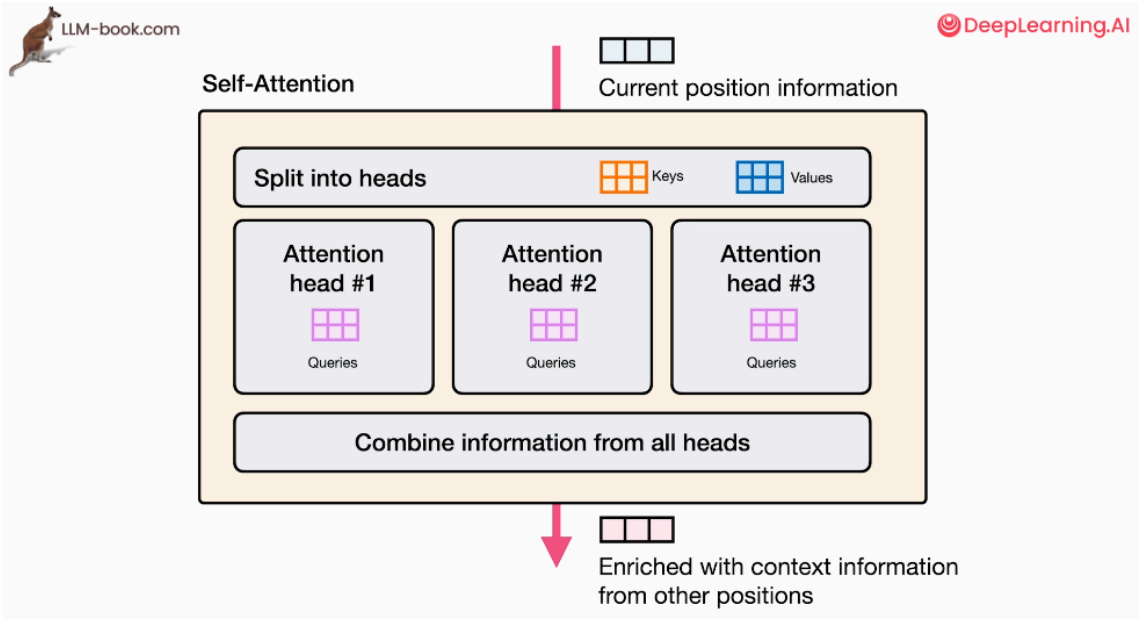
Calculation happen within attention head but in self attention the same operation happen in parallel in multiple attention head. Each attention head has its own set of key, value and query set so the attention assigned to various vector is different.



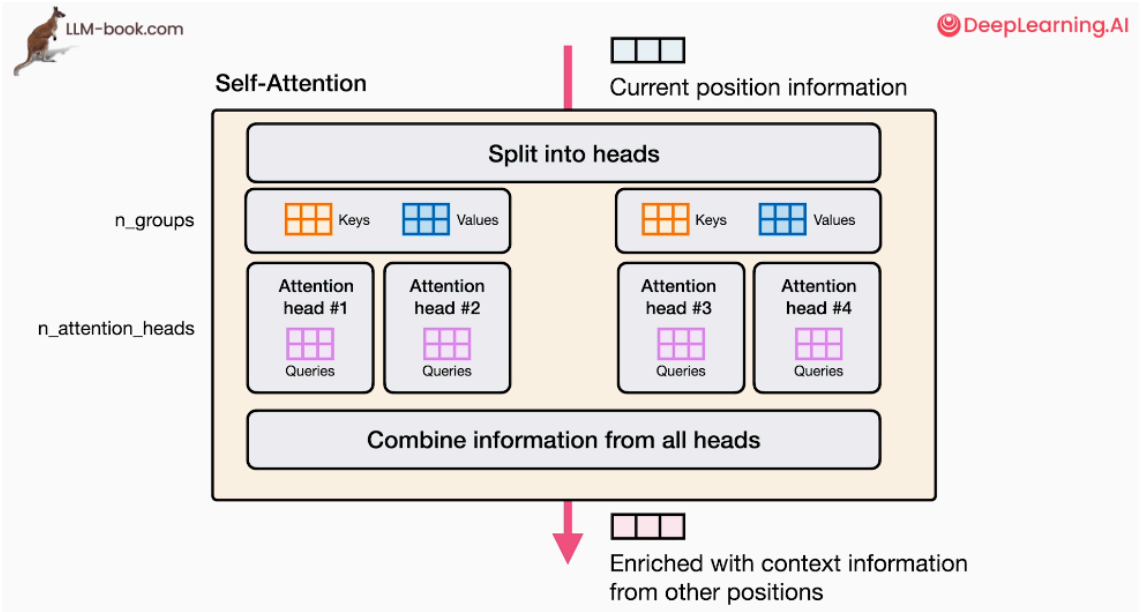




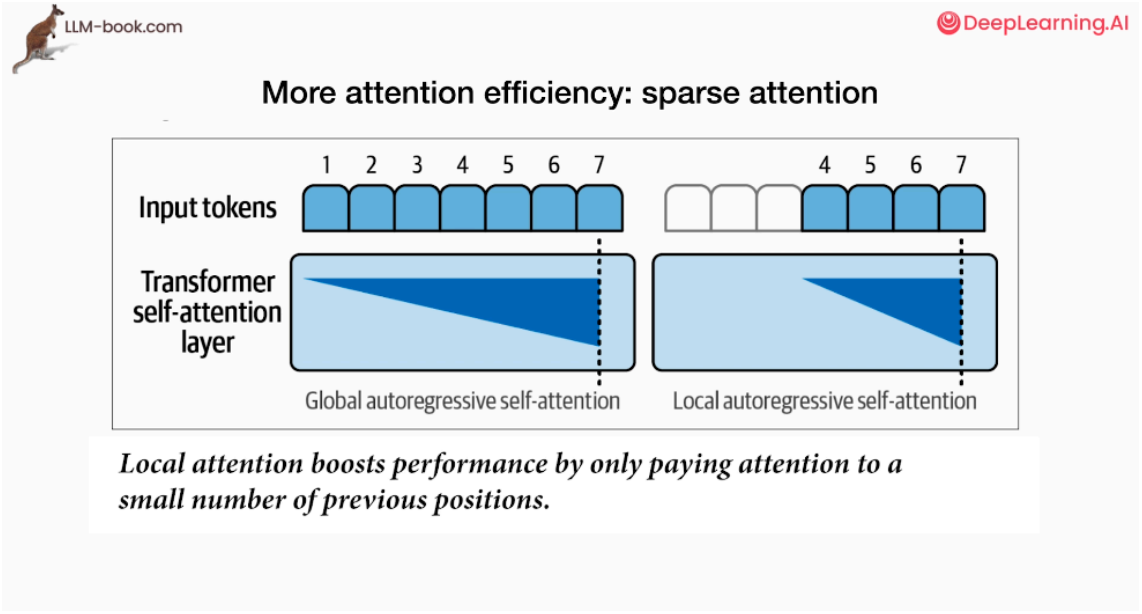
As discussed each attention has its own projection of query, key and value. This section is most time taking section so one idea proposed is let all attention head has shared keys and values matrix. That is smaller number of parameter.



More recently there is group based attention for better result. special important to larger model.

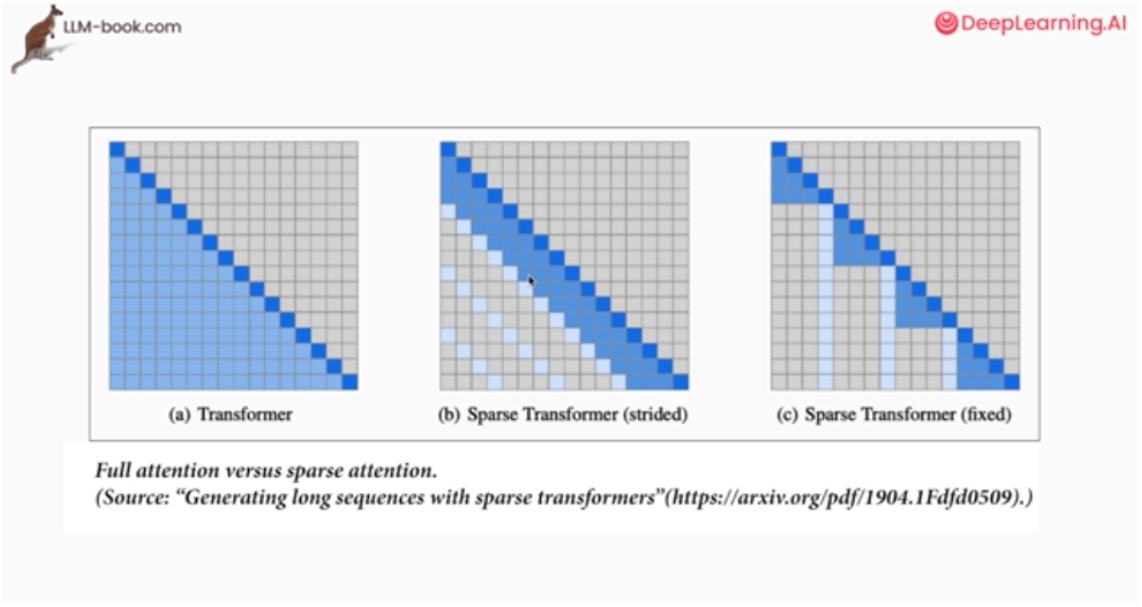


Another attention efficiency is: Sparse attention



Local attention boosts performance by only paying attention to a small number of previous positions.

a. full attention
b. sparse attention with 3 or 4 words
c. fixed position: after you reach token which is multiple of 4 then you are only allowed to look back up to that token



Full attention versus sparse attention.
(Source: "Generating long sequences with sparse transformers"(https://arxiv.org/pdf/1904.1Fdfd0509).)

More recently to allow large model to go 100 million parameters ideas like ring attention are used.

Ring Attention Explained

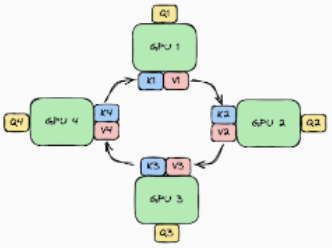
10 Apr 2024

▶ Table of Contents

By Kilian Haefeli, Simon Zirui Guo, Bonnie Li

Context length in Large Language Models has expanded rapidly over the last few years. From GPT 3.5's 16k tokens, to Claude 2's 200k tokens, and recently Gemini 1.5 Pro's **1 million** tokens. The longer the context window, the more information the model can incorporate and reason about, unlocking many exciting use cases!

However, increasing the context length has posed significant technical challenges, constrained by GPU memory capacity. What if we we could use multiple devices to scale to a near **infinite context window**? Ring Attention is a promising approach to do so, and we will dive into the tricks and details in this blog.



https://coconut-mode.com/posts/ring-attention/

Example of model architecture

Paper: The Llama 3 Herd of Models

3.2 Model Architecture

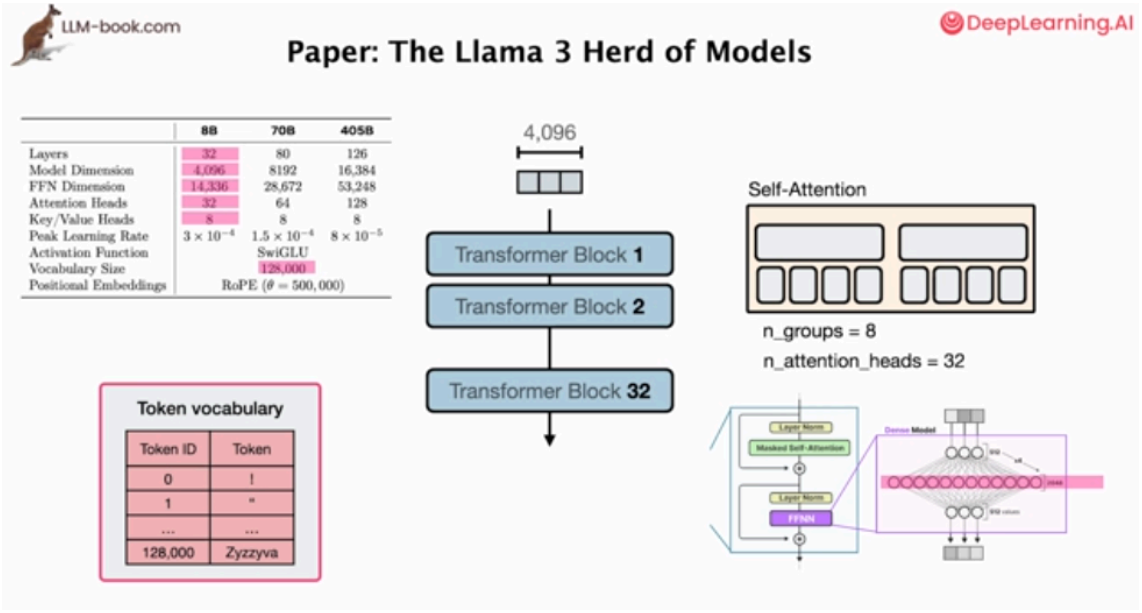
Llama 3 uses a standard, dense Transformer architecture (Vaswani et al., 2017). It does not deviate significantly from Llama and Llama 2 (Touvron et al., 2023a,b) in terms of model architecture; our performance gains are primarily driven by improvements in data quality and diversity as well as by increased training scale.

We make a few small modifications compared to Llama 2:

- We use grouped query attention (GQA; Ainslie et al. (2023)) with 8 key-value heads to improve inference speed and to reduce the size of key-value caches during decoding.
- We use an attention mask that prevents self-attention between different documents within the same sequence. We find that this change had limited impact during in standard pre-training, but find it to be important in continued pre-training on very long sequences.

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	14,336	28,672	53,248
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

Table 3 Overview of the key hyperparameters of Llama 3. We display settings for 8B, 70B, and 405B language models.



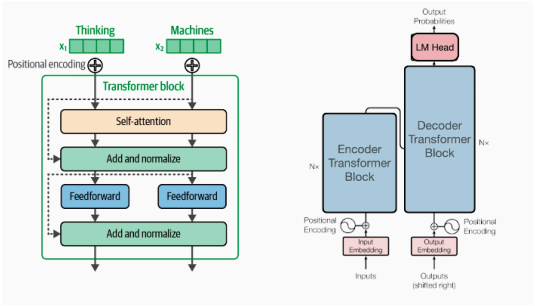
Hands On model

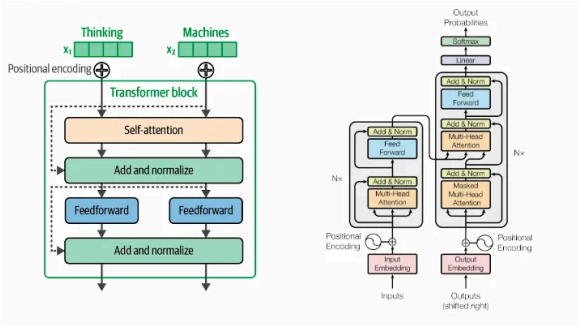
L6.ipynb

l6.pdf

Recent Improvements

Original 2017





Transformer Decoder

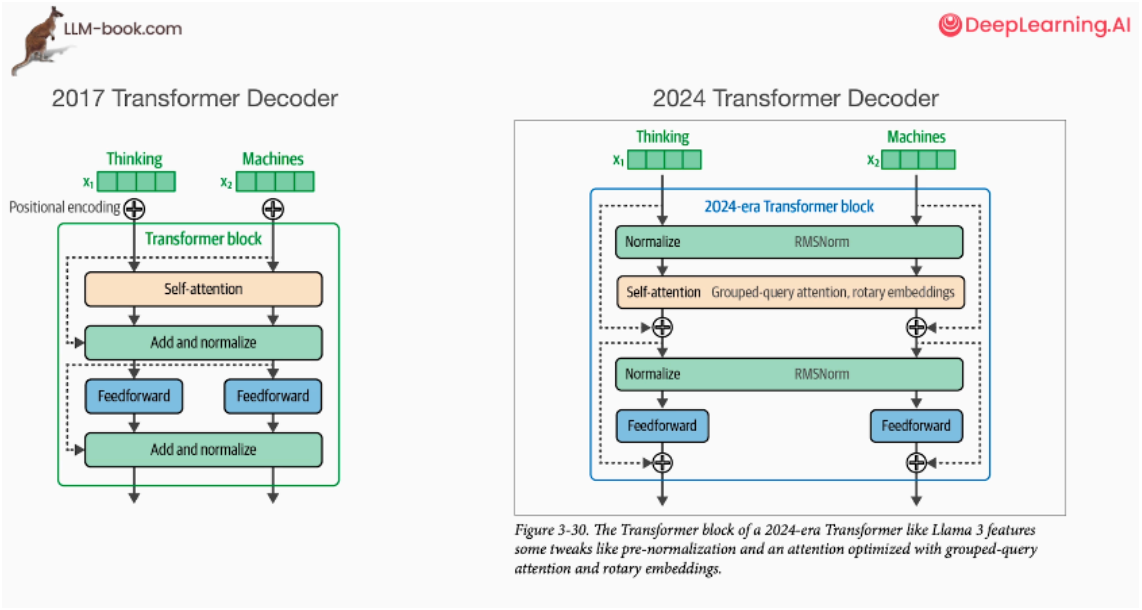
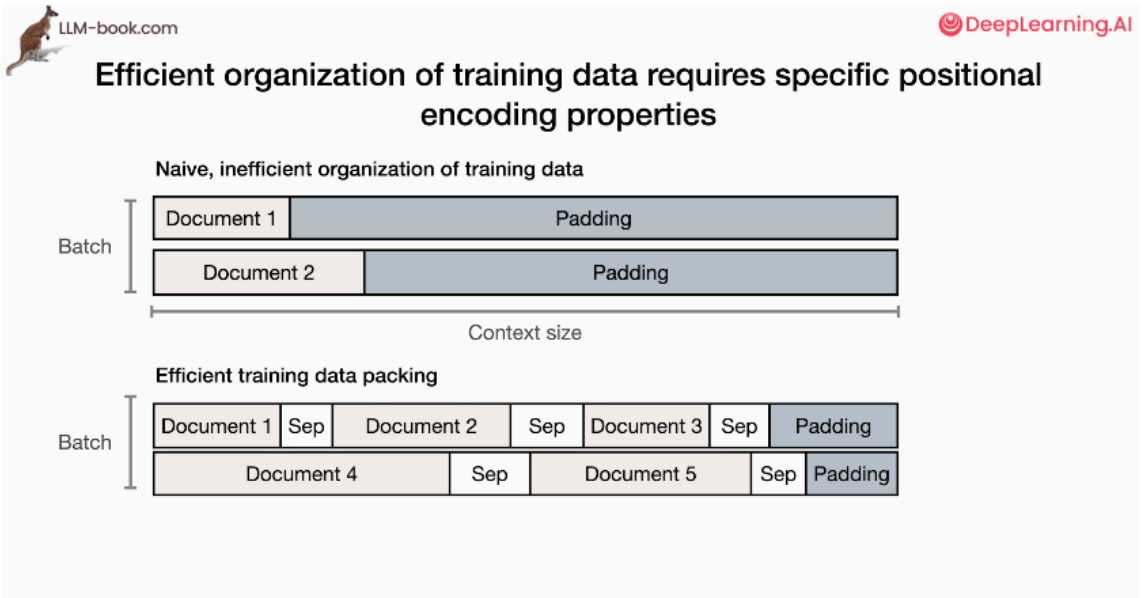


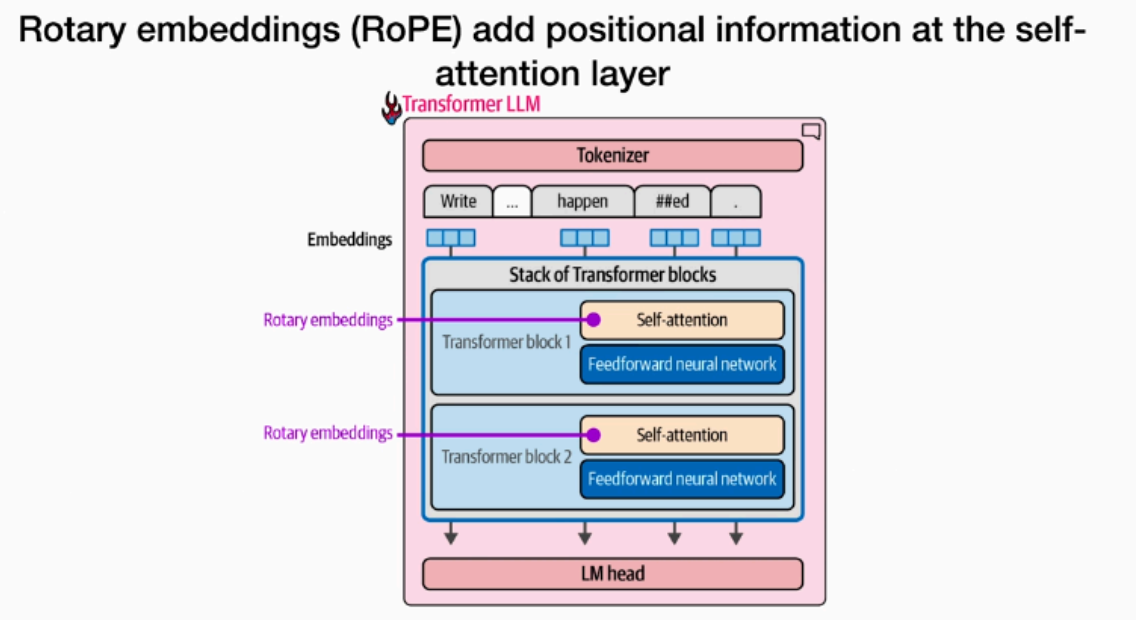
Figure 3-30. The Transformer block of a 2024-era Transformer like Llama 3 features some tweaks like pre-normalization and an attention optimized with grouped-query attention and rotary embeddings.

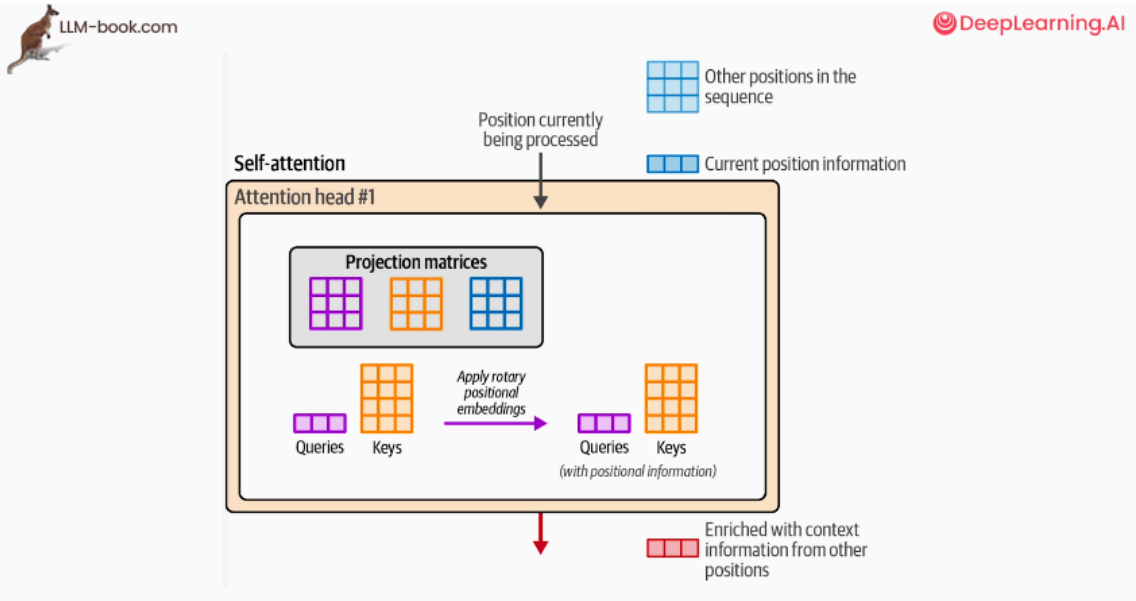
What happening:

- No longer positional encoding at the beginning of processing instead we rotary embeddings in self attention block
- The layer normalization has moved before the self attention and feedforward layer. (Some experimental results shows model do better with this change)
- Models have grouped query attention as self attention
- Both have residual connection



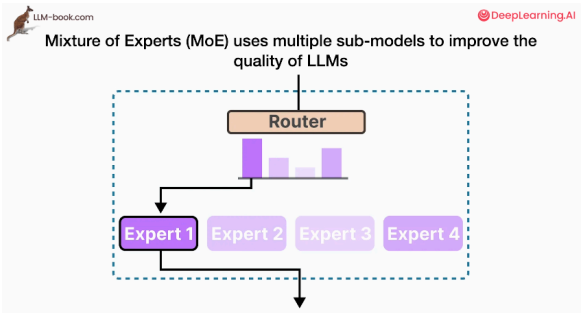
Rotary embedding



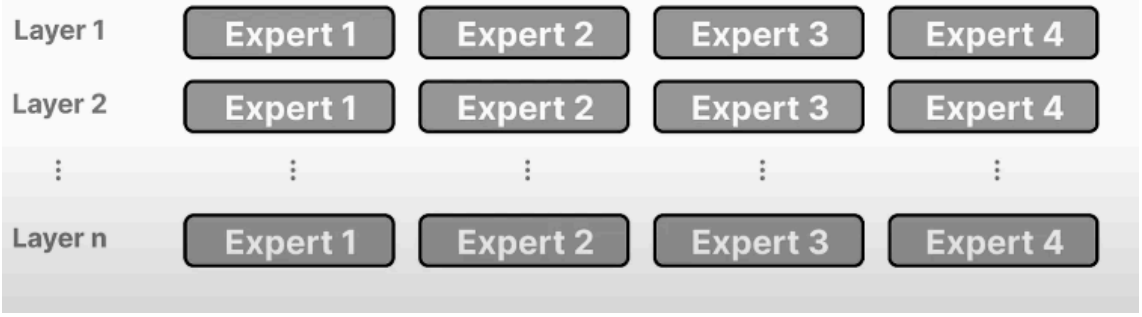


Mixture of Experts

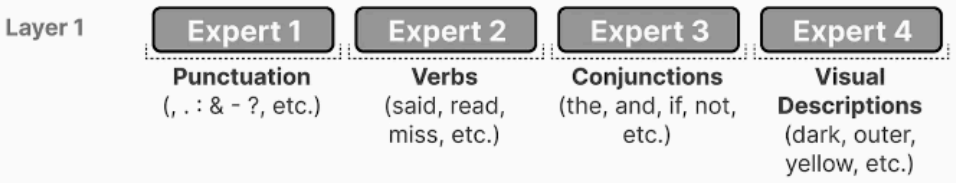
This method improves by dynamically chosen experts.



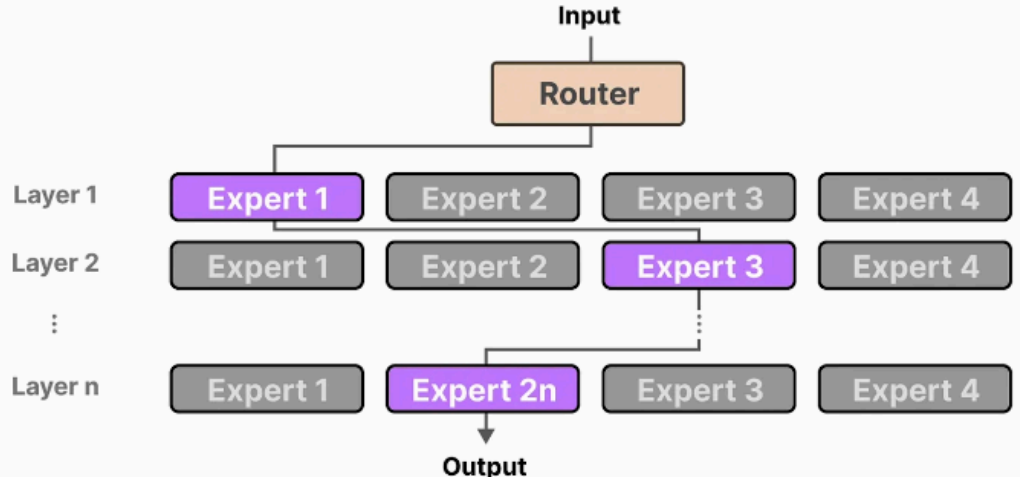
Each layer has a set of of experts that specialize in dealing with specific tokens

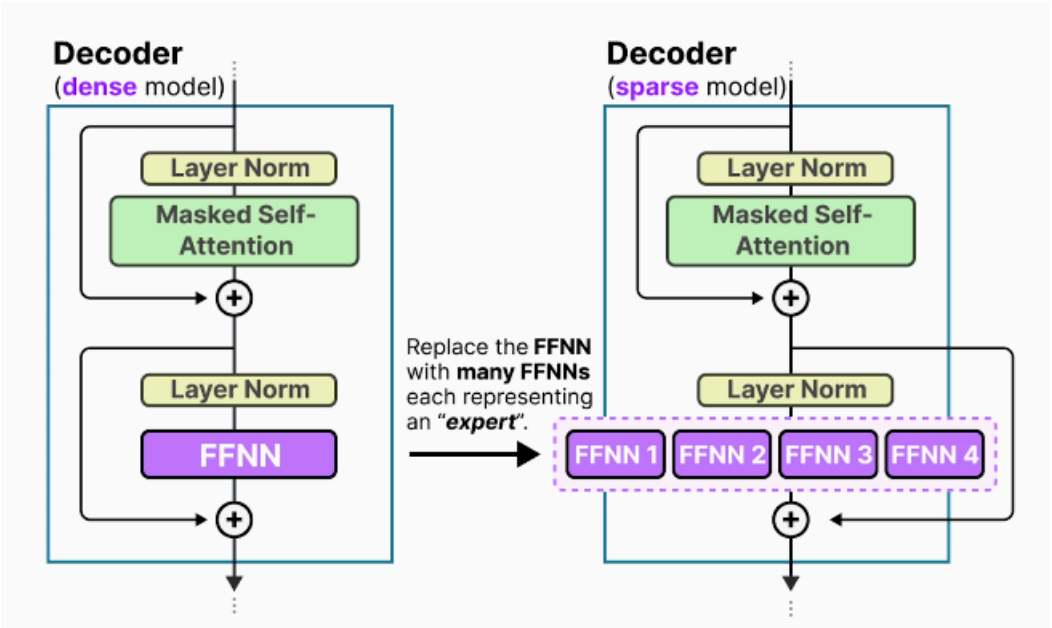


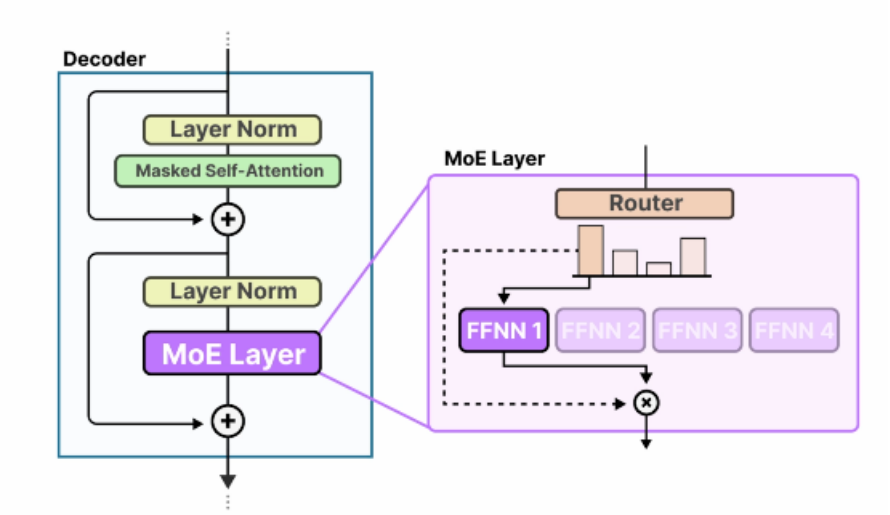
"Experts" are not specialized in specific domains like "Psychology" or "Biology"



A router at each layer chooses the best expert to process the input vector




Decoder
(dense model)
Layer Norm
Masked Self-Attention
Layer Norm
FFNN
Replace the FFNN with many FFNNs each representing an "expert".
Decoder
(sparse model)
Layer Norm
Masked Self-Attention
Layer Norm
FFNN 1
FFNN 2
FFNN 3
FFNN 4


Decoder
Layer Norm
Masked Self-Attention
Layer Norm
MoE Layer
MoE Layer
Router
FFNN 1
FFNN 2
FFNN 3
FFNN 4