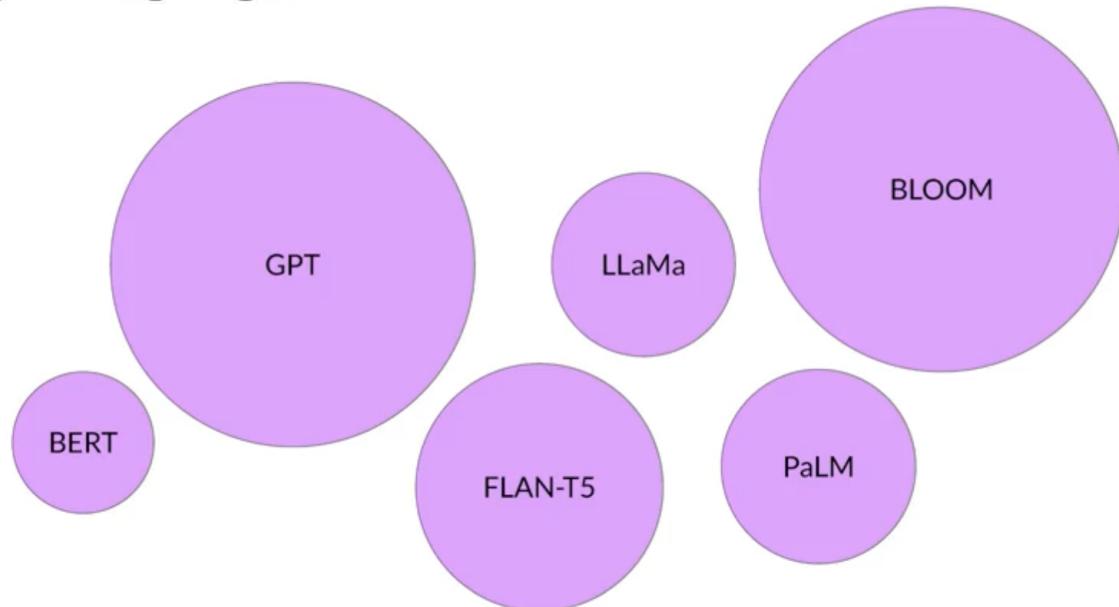


# Generative AI & LLMs

Generative AI is a subset of traditional ML. ML models that underpinned learn the statistical pattern of the data.

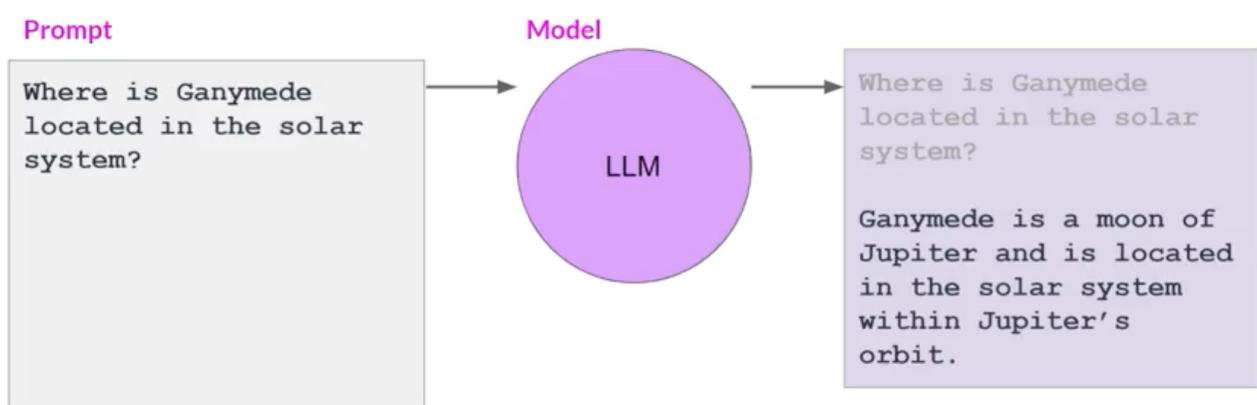
## Fundation (Base) Models

### Large Language Models



more parameters = more memory

### Prompts and completions



Context window

- typically a few 1000 words.

The output of the model is called a **completion**. The act of using the model to generate text is known as inference.

# The significance of scale: language understanding



Generative algorithms are not new, Recurrent Neural Networks.

## Generating text with RNNs



## Generating text with RNNs



# Understanding language can be challenging

I took my money to the bank.

River bank?

## Understanding language can be challenging

The teacher's book?

The teacher taught the student with the book.

The student's book?

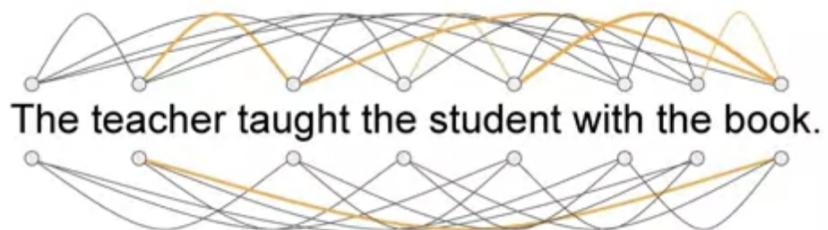
## Transformer Architecture

# Transformers

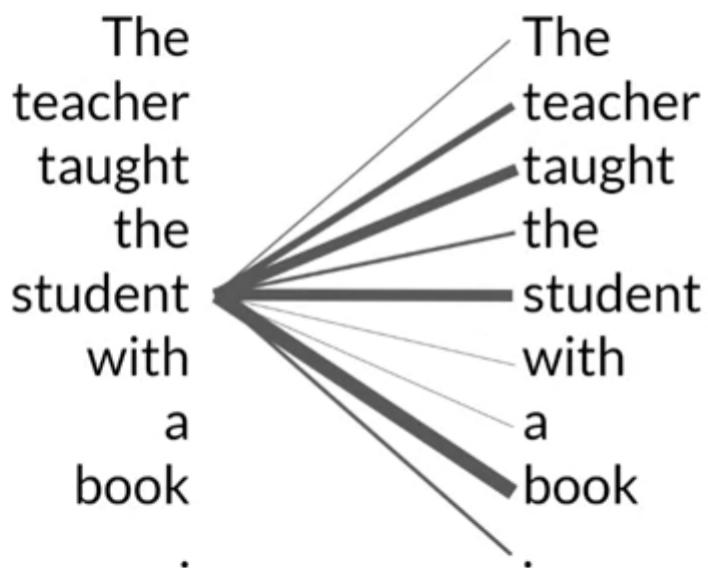


# Transformers

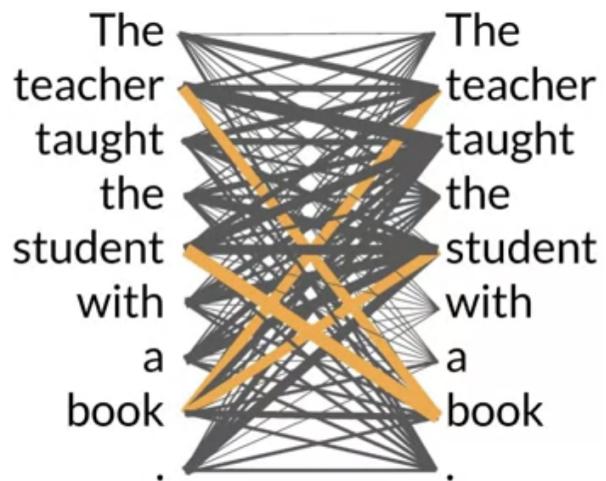
- Scale efficiently
- Parallel process
- Attention to input meaning



# Self-attention

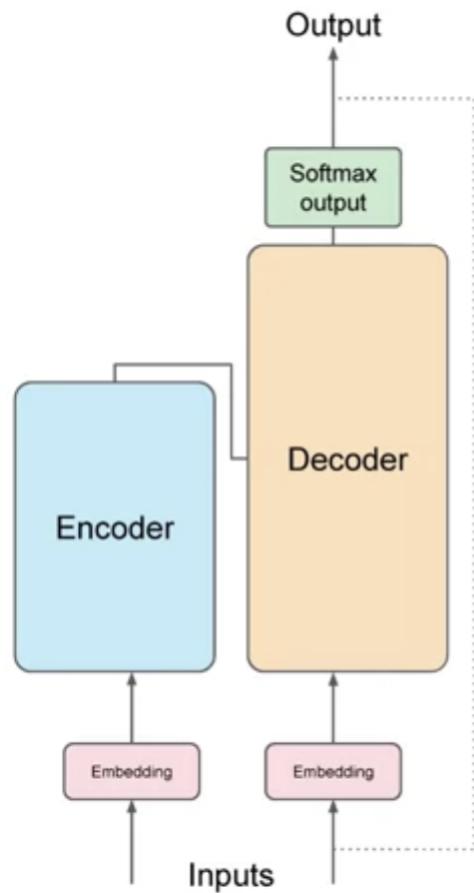


# Self-attention

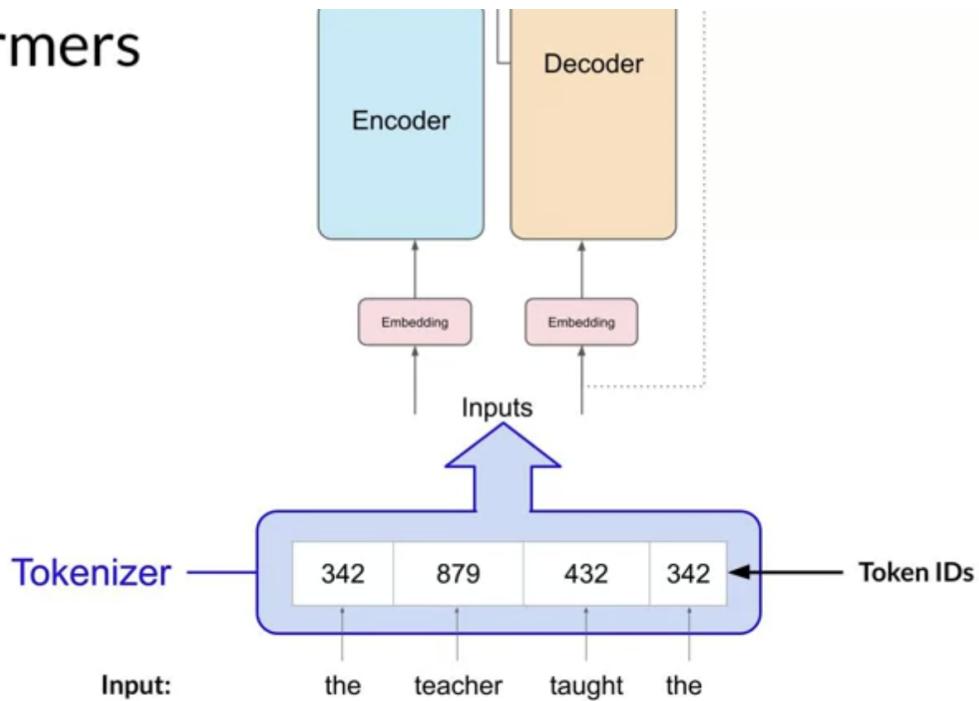


How the model works?

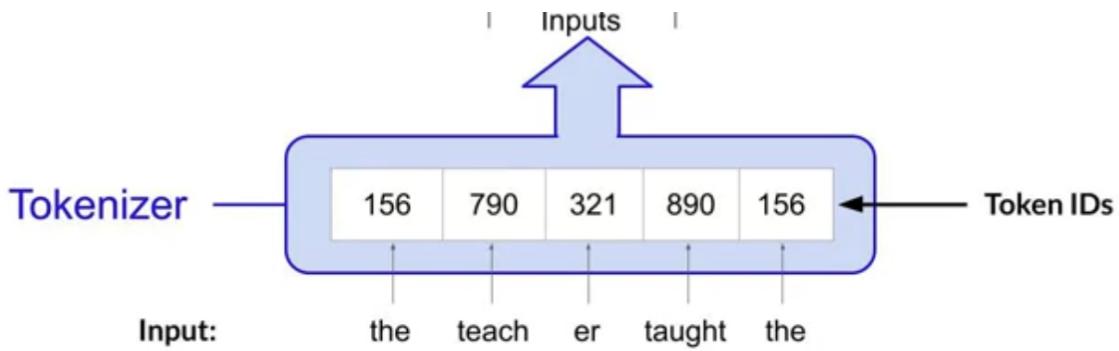
# Transformers



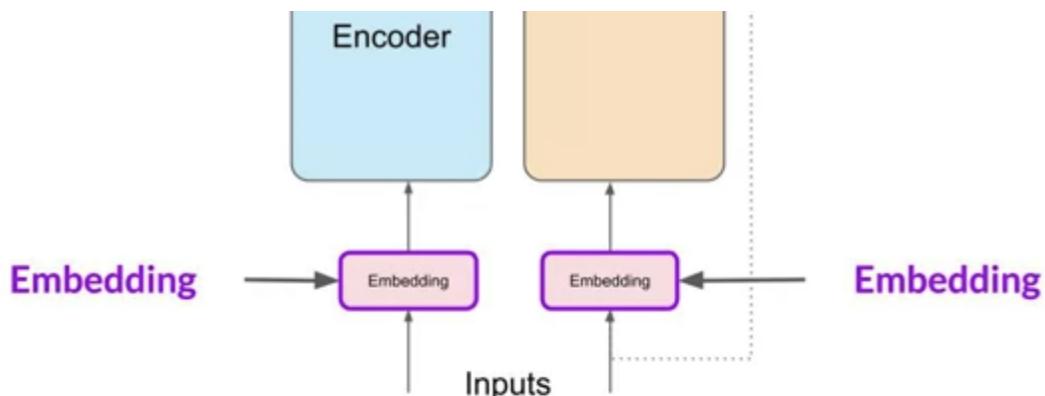
# Transformers



Now, machine-learning models are just big statistical calculators and they work with numbers, not words. So before passing texts into the model to process, you must first tokenize the words. Simply put, this converts the words into numbers, with each number representing a position in a dictionary of all the possible words that the model can work with. You can choose from multiple tokenization methods.

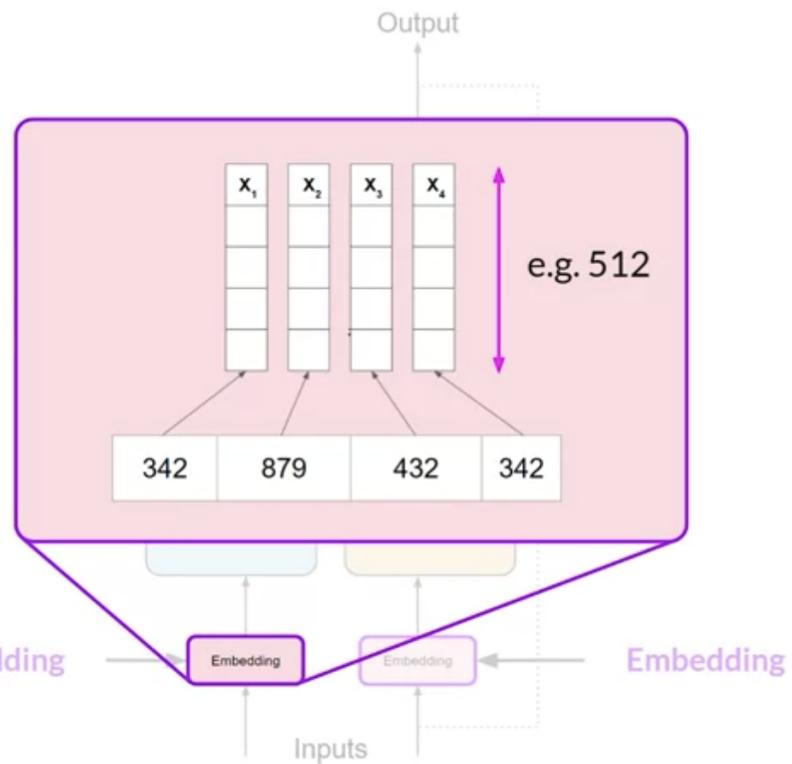


Now that your input is represented as numbers, you can pass it to the embedding layer. This layer is a trainable vector embedding space, a high-dimensional space where each token is represented as a vector and occupies a unique location within that space. Each token ID in the vocabulary is matched to a multi-dimensional vector, and the intuition is that these vectors learn to encode the meaning and context of individual tokens in the input sequence.



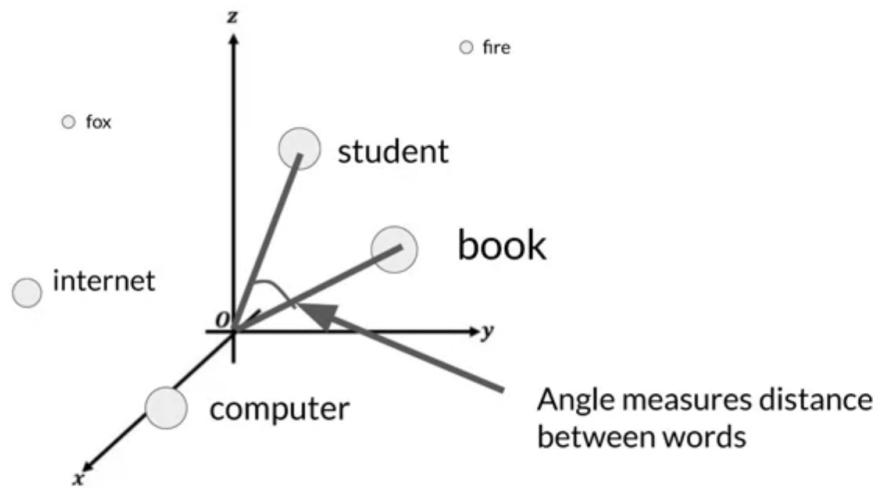
From the Transformer paper

# Transformers



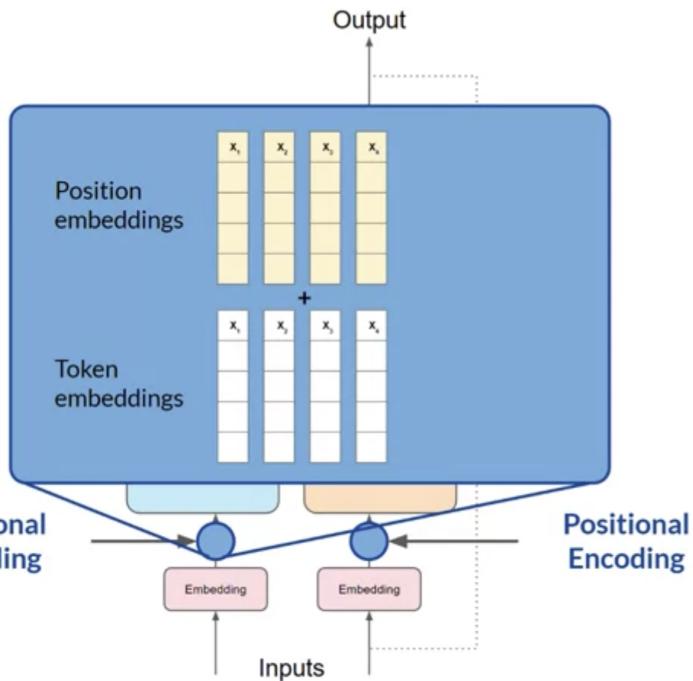
In case of 3 dimension per embedding:

# Transformers



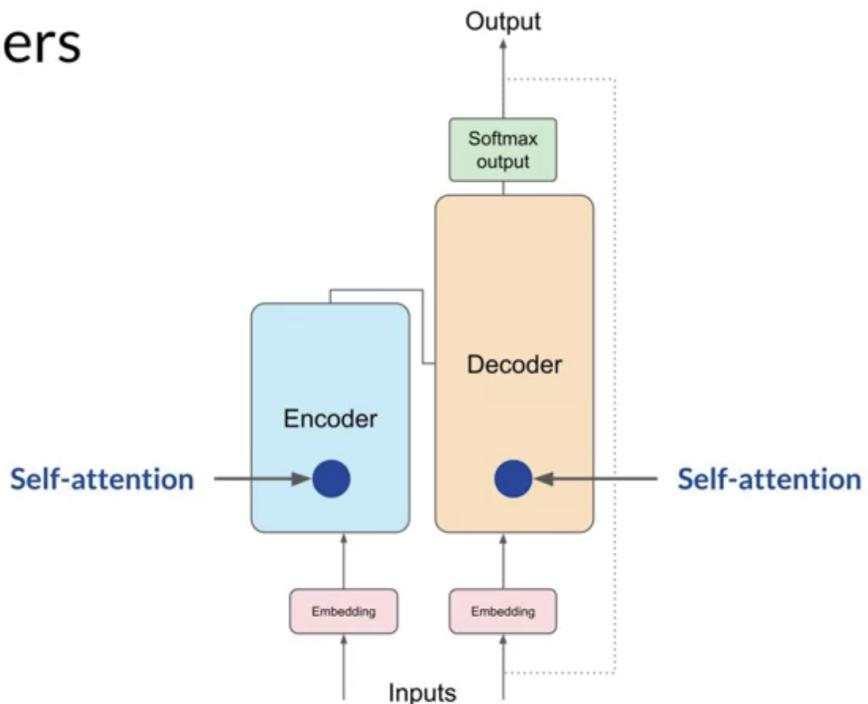
Positional Encoding to preserve the word order (because everything append in parallel). So by adding the positional encoding to the token embeddings you preserve the information and don't lose the relevance of the position of the words in the sentence.

# Transformers

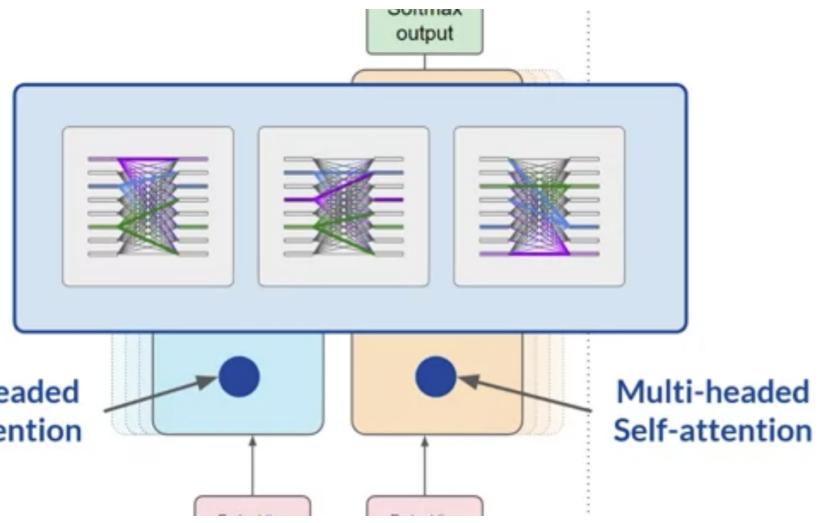


The model analyzes the relationships between the tokens in your input sequence. As you saw earlier, this allows the model to attend to different parts of the input sequence to better capture the contextual dependencies between the words. The self-attention weights that are learned during training and stored in these layers reflect the importance of each word in that input sequence to all other words in the sequence. But this does not happen just once, the transformer architecture actually has multi-headed self-attention. This means that multiple sets of self-attention weights or heads are learned in parallel independently of each other.

# Transformers

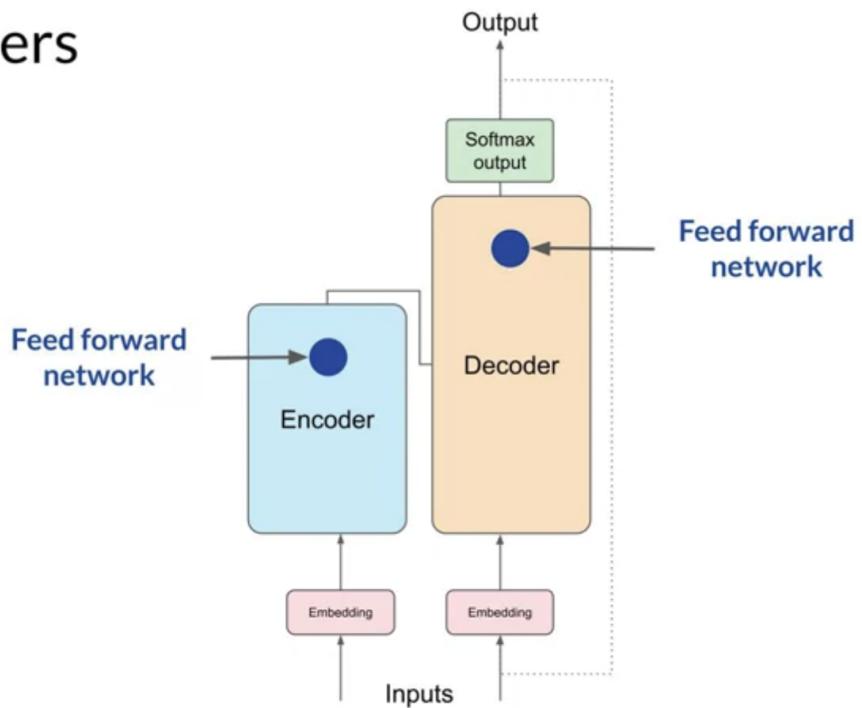


The number of attention heads included in the attention layer varies from model to model, but numbers in the range of 12-100 are common. The intuition here is that each self-attention head will learn a different aspect of language. For example, one head may see the relationship between the people entities in our sentence. Whilst another head may focus on the activity of the sentence. Whilst yet another head may focus on some other properties such as if the words rhyme. It's important to note that you don't dictate ahead of time what aspects of language the attention heads will learn. The weights of each head are randomly initialized and given sufficient training data and time, each will learn different aspects of language.



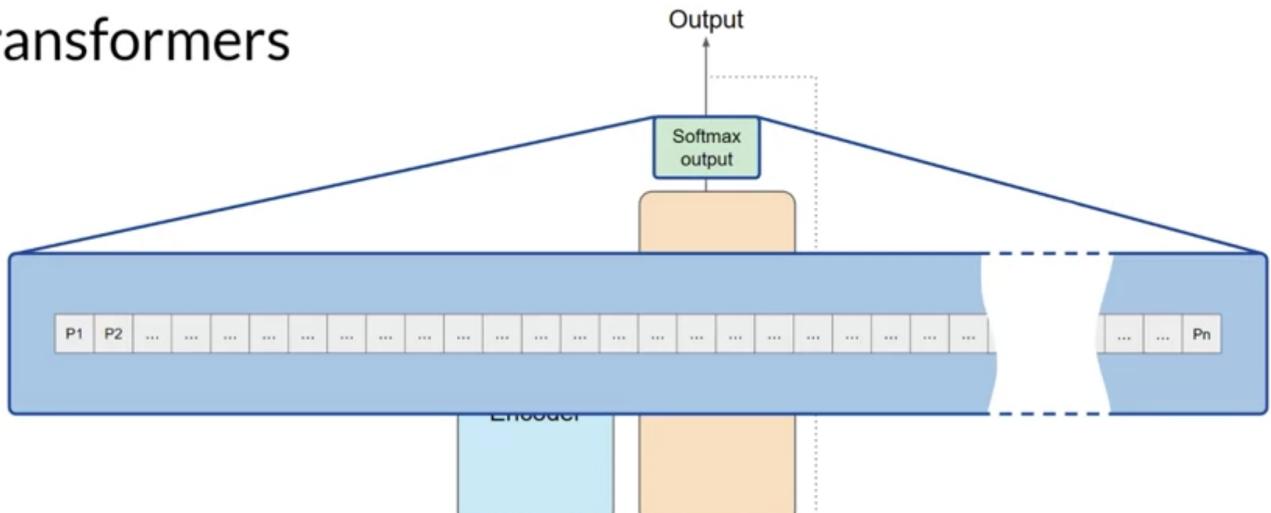
The output is processed through a fully connected feed-forward network. The output of this layer is a vector of logits proportional to the probability score to each and every token in the tokenizer dictionary.

## Transformers



Those logits pass to a softmax layer, where they are normalized into a probability score for each word.

## Transformers



The output is a probability for every single word in the vocabulary. So there is likely to be thousands of scores. One single token will have the score higher than the rest. **This is the most likely predicted token.**

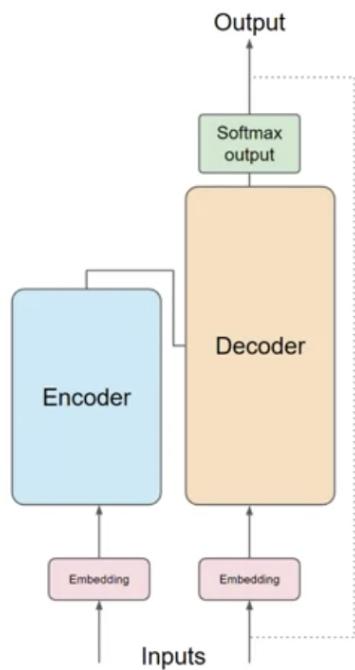
## Example End-2-End Translation

### Transformers

Translation:  
sequence-to-sequence task

J'aime l'apprentissage  
automatique

J'ai l'apprentissage  
automatique



1. Tokenizer

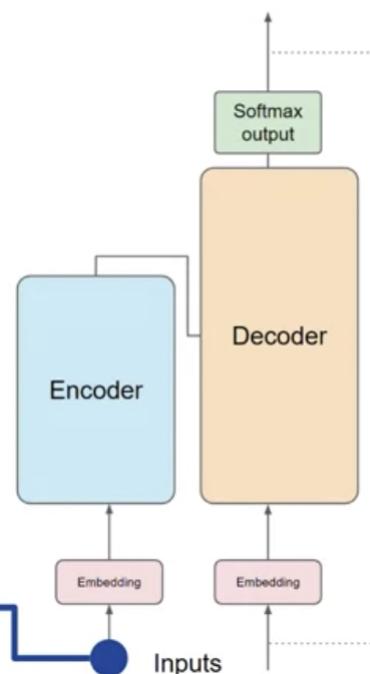
### TRANSFORMERS

Translation:  
sequence-to-sequence task

J'aime l'apprentissage  
automatique

J'ai l'apprentissage  
automatique

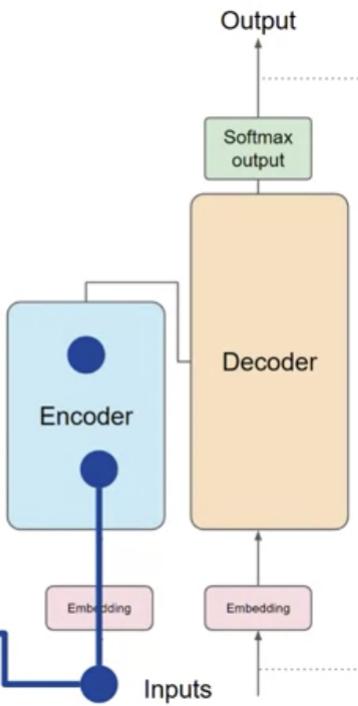
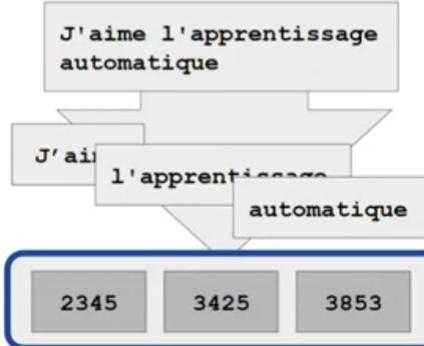
2345 3425 3853



2. Embedding Layer + Encoder Layer (Multi-head attention layers)

# Transformers

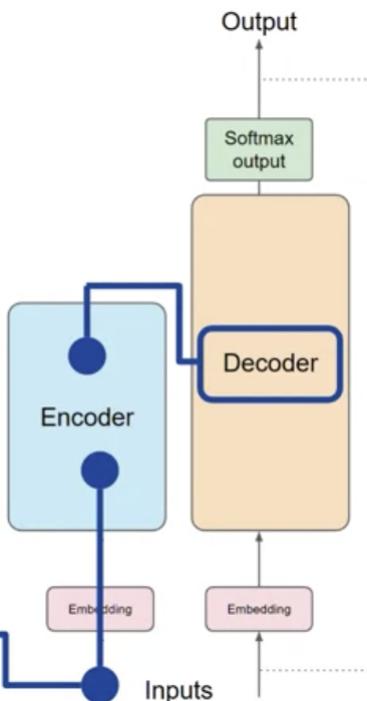
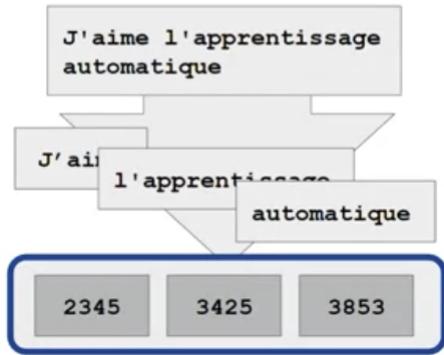
Translation:  
sequence-to-sequence task



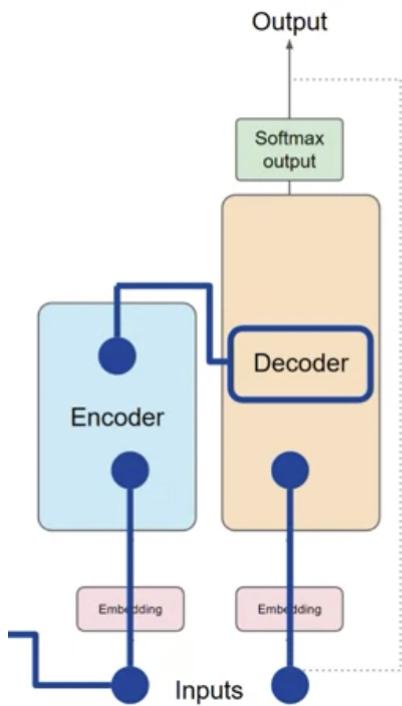
3. The output of the multi-head attention is injected into the feed-forward NN to the output of the encoder.
4. At this point, the data that leaves the encoder is a deep representation of the structure and the meaning of the input sequence. This is inserted into the middle of the Decoder (i.e., self-attention of the decoder):

# Transformers

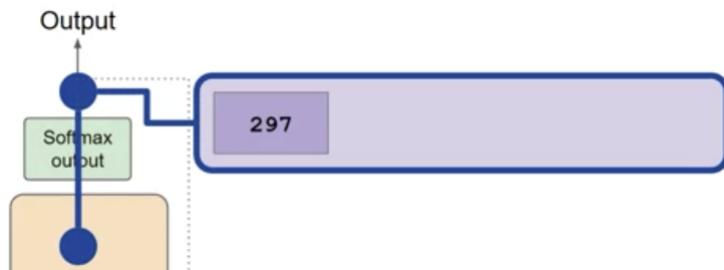
Translation:  
sequence-to-sequence task



5. Decode, a start of sequence token is added to the input of the decoder triggering the decoder to predict the next token. It does this based on the contextual understanding (output of the encoder to the decoder).



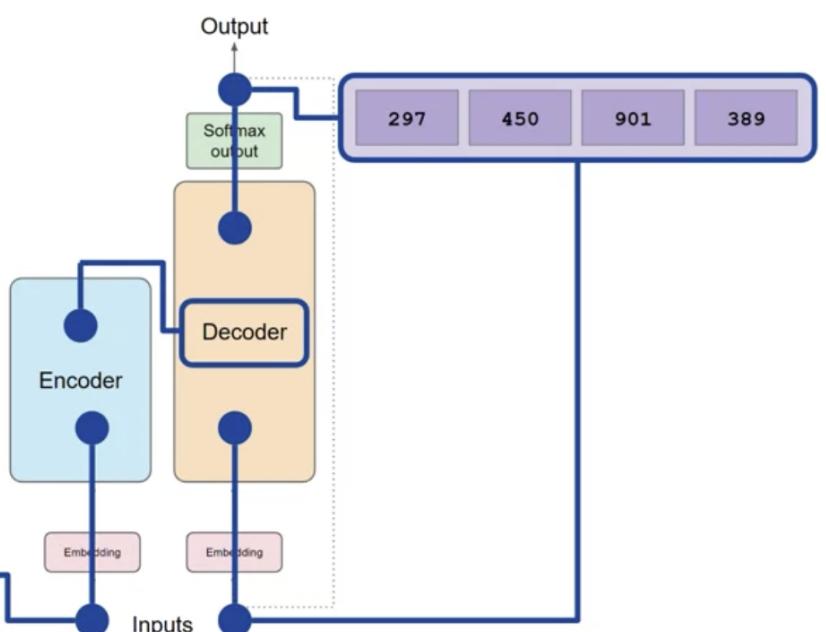
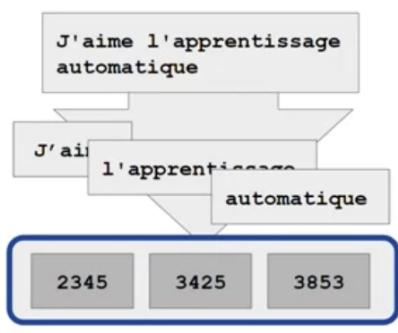
6. The output of the decoder passes through the decoder feed-forward NN and a final softmax output layer.



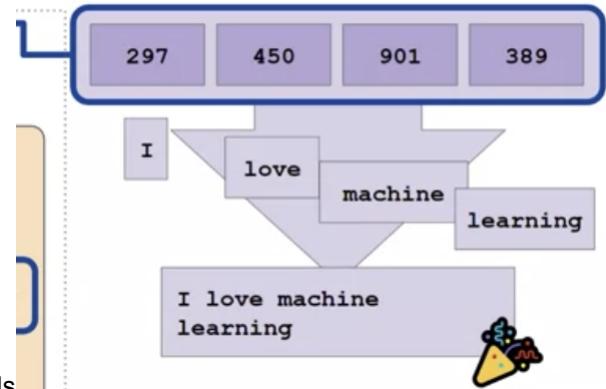
7. Here we have the first token, we can continue by using it as input to the decoder for the prediction of the next token.

## Transformers

Translation:  
sequence-to-sequence task

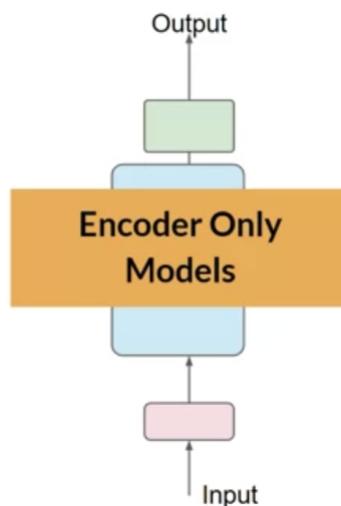


- The final sequence of token can be de-tokenized into words



## In summary:

```
![[Pasted image 20240716114258.png]]
```



## Encoder (input and output same length):

You can try encoder-only model to perform classification task such as sentiment analysis. BERT is an example of an encoder-only model.

## Encoder-Decoder Model

```
![[Pasted image 20240716114619.png]]
```

Sequence to Sequence tasks such as translation where the input sequence and the output sequence can be different lengths. You can scale this model to do generation tasks. BART and T5 are example of this architecture.

## Decoder-only model, most common used.

GPT, BLOOM, Llama and more.

The multi-head self-attention mechanism allows the model to attend to different parts of the input sequence, while the feed-forward network applies a point-wise fully connected layer to each position separately and identically. The Transformer model also uses residual connections and layer normalization to facilitate training and prevent overfitting.

In addition, the authors introduce a positional encoding scheme that encodes the position of each token in the input sequence, enabling the model to capture the order of the sequence without the need for recurrent or convolutional operations.

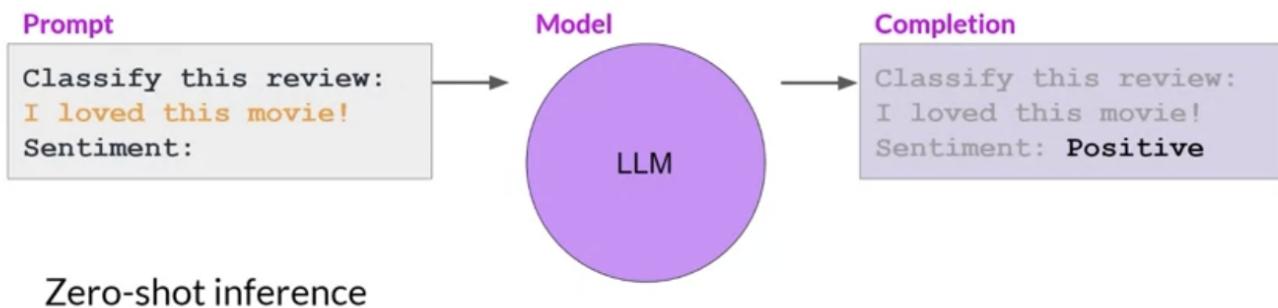
## Prompting and prompt engineering

The input is called prompt, the act of generating text is known as inference, the output of the model is called completion. The context window is where you type the prompt and it appears as output at the beginning.

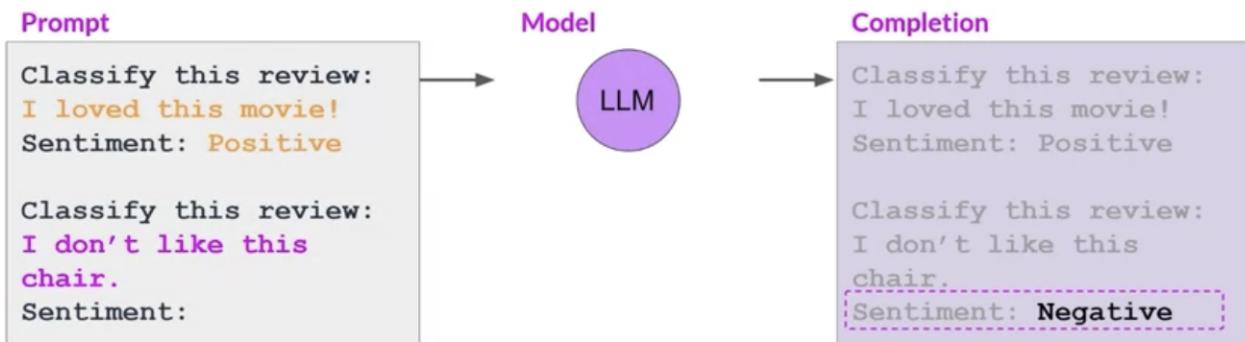
### in-context learning (ICL)

When the user provides the example inside the context window.

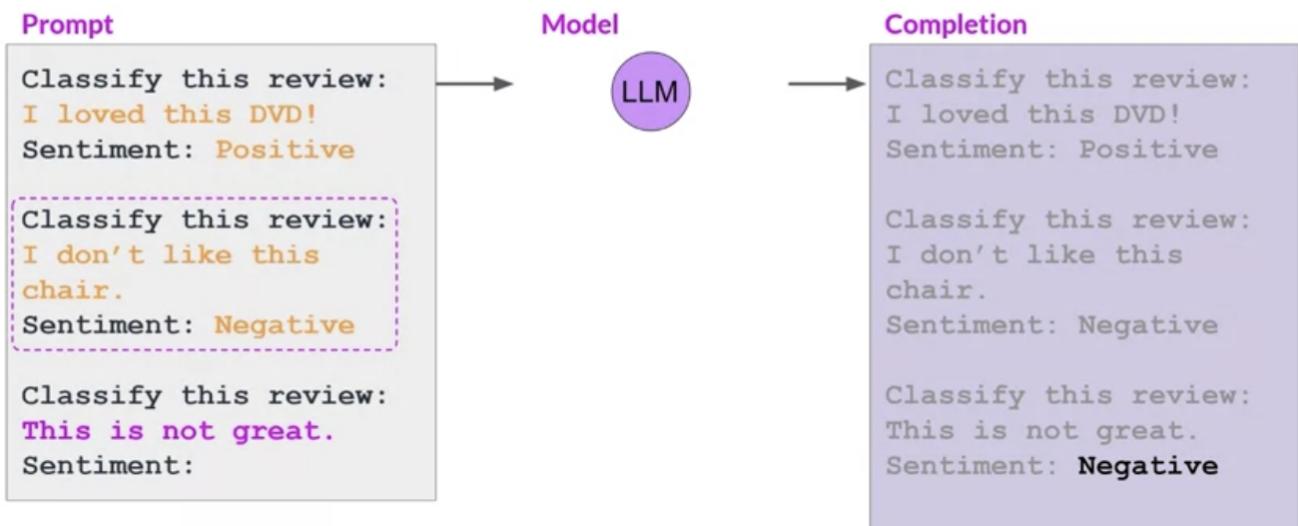
## In-context learning (ICL) - zero shot inference



## In-context learning (ICL) - one shot inference



## In-context learning (ICL) - few shot inference



## Summary of in-context learning (ICL)

### Prompt // Zero Shot

Classify this review:  
**I loved this movie!**  
Sentiment:

### Prompt // One Shot

Classify this review:  
**I loved this movie!**  
Sentiment: Positive

Classify this review:  
**I don't like this chair.**  
Sentiment:

### Prompt // Few Shot

Classify this review:  
**I loved this movie!**  
Sentiment: Positive

Classify this review:  
**I don't like this chair.**  
Sentiment: Negative

Classify this review:  
**Who would use this product?**  
Sentiment:

## Parameters

# Generative configuration - inference parameters

Max new tokens

Sample top K

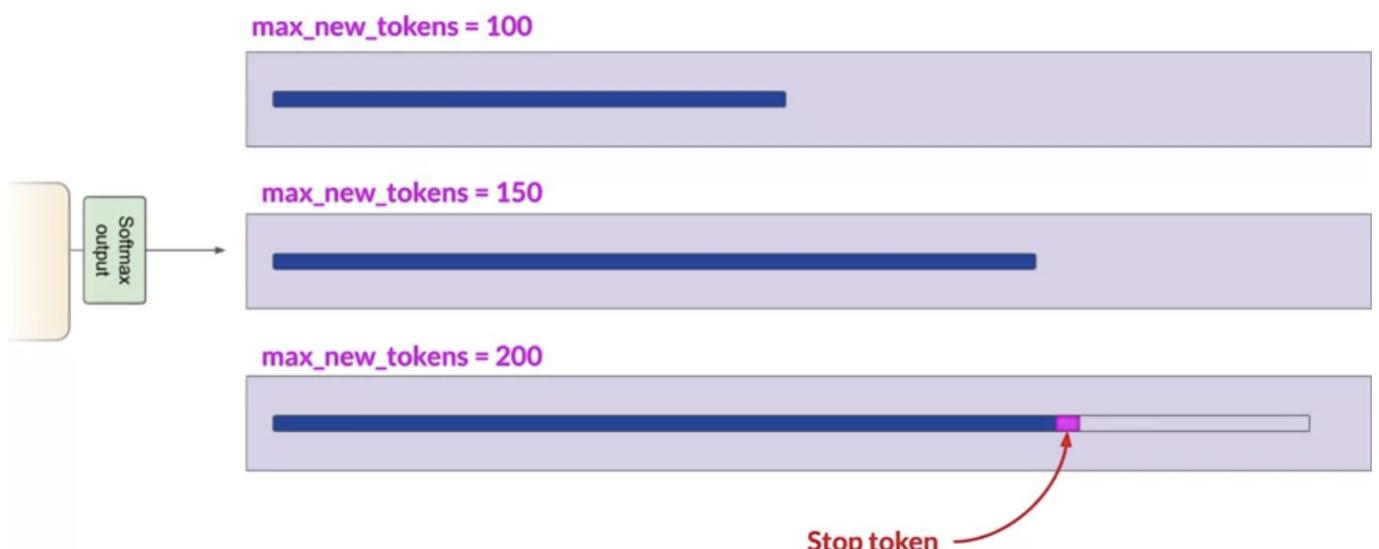
Sample top P

Temperature

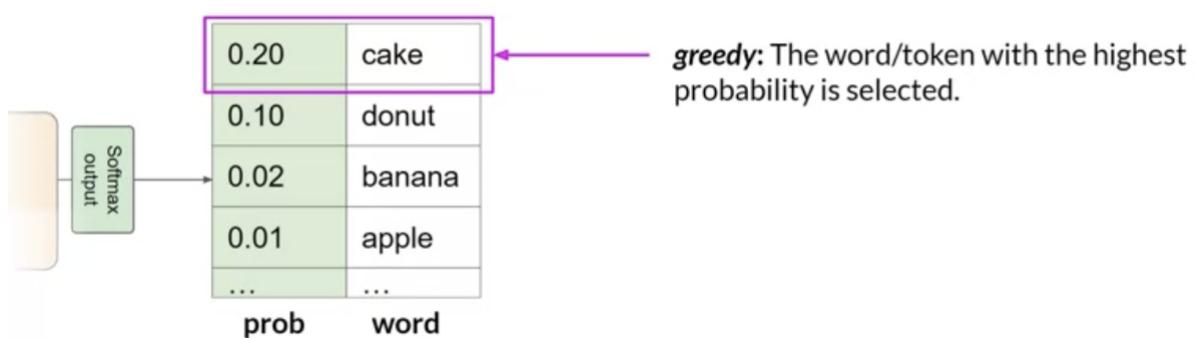
Submit

Inference configuration parameters

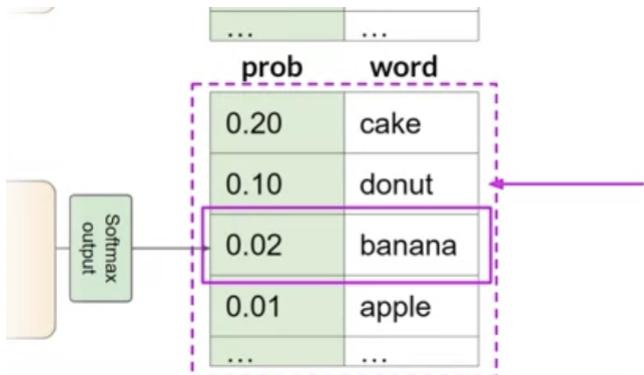
## Generative config - max new tokens



## Generative config - greedy vs. random sampling



be careful with repetitions. Instead random sampling is more creative:

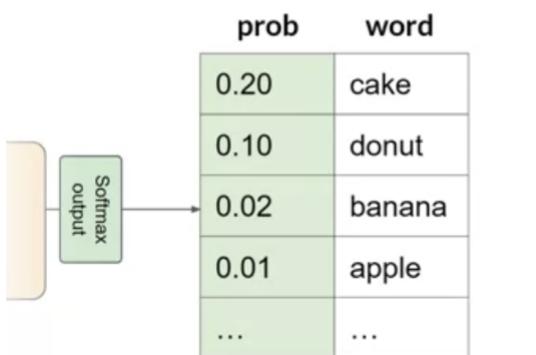


**random(-weighted) sampling:** select a token using a random-weighted strategy across the probabilities of all tokens.

Here, there is a 20% chance that 'cake' will be selected, but 'banana' was actually selected.

Top-K based on the probability, then random.

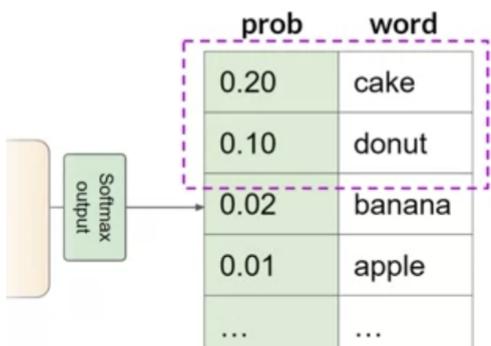
## Generative config - top-k sampling



**top-k:** select an output from the top-k results after applying random-weighted strategy using the probabilities

Top-p based on the probability threshold, then random.

## Generative config - top-p sampling



**top-p:** select an output using the random-weighted strategy with the top-ranked consecutive results by probability and with a cumulative probability  $\leq p$ .

$$p = 0.30$$

Temperature controls randomness

# Generative config - temperature

